Consistency Management of Distributed Documents using XML and Related Technologies

Ernst Ellmer, Wolfgang Emmerich, Anthony Finkelstein,
Danila Smolko and Andrea Zisman
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{E.Ellmer|W.Emmerich|A.Finkelstein|D.Smolko|A.Zisman}@cs.ucl.ac.uk

Abstract

In this paper we describe an approach and associated techniques for managing consistency of distributed documents. We give an account of a toolkit which demonstrates the approach. The approach supports the management of consistency of documents with Internet-scale distribution. It takes advantage of XML (eXtensible Markup Language) and related technologies. The paper contains a brief account of the base technologies and an extended discussion of related work. The approach and the toolkit are described in detail in the context of a typical application.

1 Motivation and Context

Most organisations produce documents. These documents contain business and technical information and generally have some sort of structure. Examples of such documents include policy statements, reports, manuals, specifications business forms and similar. The documents may form part of an organised process or workflow or be produced on an ad-hoc basis to respond to some requirement within the organisation. The documents are commonly instances of standard "document types" conforming to some agreed template or standard, though they can be "one-off". Documents are generally produced by different actors for example by a system manager, a system designer, a system user and so on. These actors may be physically distributed, forming part of distinct workgroups located in different places. The documents these actors produce are of varied form and produced using heterogeneous applications, word processors, specialised applications, software engineering tools and similar.

The documents we are concerned with typically have overlapping content that is they refer to common objects or phenomena within the domain of discourse. Overlapping content gives rise to consistency relationships between documents and document elements. Similarly there may be consistency relationships between documents produced within the organisation and documents published by other organisations. The setting we have described is "open" in the sense that new documents, document types and consistency relationships can be added at any time.

Managing this complex situation and assuring the integrity of the documents at key points in the process, for example when a document is baselined, while at the same time tolerating inconsistency which naturally arises during the document production process is very difficult and generally recognised as beyond the existing state-of-the-art.

2 Contribution of this paper

In this paper we describe an approach and associated techniques for managing consistency of distributed documents. We give an account of a toolkit which demonstrates the approach.

We provide means for managing consistency of documents with Internet-scale distribution. Our approach is very simple and light-weight and can be readily deployed in a variety of different settings. This simplicity is achieved by building on existing Internet technologies which form a very powerful and widely used base. It is also achieved by exploiting an emerging standard XML (eXtensible Markup Language) and related technologies. The work we have carried out originates in our interest in software engineering and particularly in the problem of managing consistency among software engineering documents. Though the work we describe has clear and immediate application in this area and we will continue to use this as an example, our concerns are wider and the approach applies, we believe to all classes of structured document.

In the paper which follows we set out an example problem, managing software engineering documents produced in the UML (Unified Modelling Language). We describe XML and related technologies. We outline our approach and provide an account of our implementation of the architecture. The approach is evaluated and related work reviewed. The paper outlines future directions and suggests ways in which the overall approach could be extended

3 XML and Related Technologies

In this section we provide a succinct account of XML (eXtensible Markup Language) and related technologies which constitute the major part of the infrastructure on which we build. We introduce XML itself, XML-Linking and XSL (eXtensible Stylesheet Language) which is required in order to view XML documents. We give a very brief account of the DOM (Document Object Model) associated with XML. It must be appreciated that this is a necessarily brief review of a complex set of technologies and that in each case the base technologies are pre-standard or only just beginning to harden as standards and are subject to continuing change.

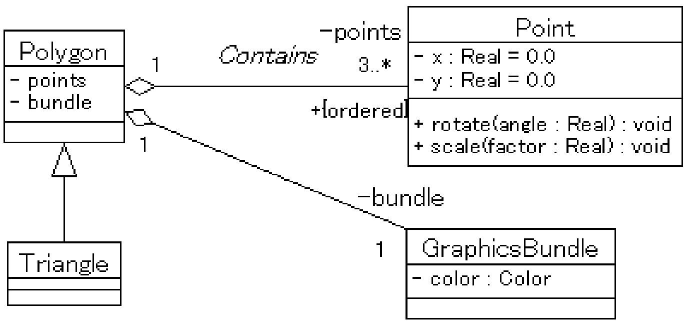
3.1 XML

XML (eXtensible Markup Language) [Bray et al., 1998] is a data description language, a subset of the Standard Generalized Markup Language (SGML), which has been standardised by the World Wide Web Consortium (W3C). XML is designed to bring structured information to the Web, it allows users to define their own set of markup tags relating to the content of their documents thus delivering both extensibility and potential for validation. It is a major move away from the fixed markup tags embedded in HTML. XML provides a data standard that can encode the content, semantics and schemata for a wide variety of cases – whether as a wire format for sending data between client and server, a transfer format for sharing data between applications, or a persistent storage format on disk.

XML provides a general method for describing data. It allows identification, exchange and processing of distributed data in a manner that is mutually understood. Programmers can build simple parsers to read XML data, making it a good format for interchanging data. It is designed to be straightforwardly usable over the Internet, and support a wide variety of applications. XML maintains the separation of presentation details from structured data and, therefore, it allows the integration of data from diverse sources. XML seeks to achieve a compromise between flexibility, simplicity, and readability by both humans and machines.

XML provides a set of element types, which serve to define types of documents and are referred to as Document Type Definitions (DTDs). A DTD contains a set of rules to control how documents and tags are structured, which elements are presented and the structural relationship between the elements for documents of a particular type. A DTD contains the definition of the syntax of an XML document, i.e. DTDs are schemas for documents. A range of XML DTDs are emerging in particular domains for example MathML (the mathematical markup language).

Figure 1 shows a document containing a UML (Unified Modelling Language) class diagram with a fragment of the associated mark up in UXF [Suzuki and Yamamoto, 1999], DTD for mark up of UML. The complete DTD is presented in the Appendix B of this paper.




```
<?xml version="1.0"?>
<!DOCTYPE Model SYSTEM "UML.DTD">
<Model>
 <TaggedValue>
  <Tag>Title<Value>Polygon Sample
  </Value></Tag>
 </TaggedValue>
 <Package NAME = ”PolygonSample">
  <ClassDiagram>
   <Class NAME = "Point” … >
    <Attribute VISIBILITY = "private"
               TYPE       = "Real"
               NAME       = "x"
               INITVAL    = "0.0"/>
    <Operation VISIBILITY = "public”...>
    </Operation>
   </Class>
  </ClassDiagram>
 </Package>
</Model>
```

Figure 1: An XML Representation of a Class Diagram using the UXF DTD

The XML specification refers to two components: the XML processor and the XML application. The XML processor is the parser with the task of loading the XML and any related files, checking to make sure that a XML document follows all the necessary rules (i.e. related DTD), and building a document tree structure that can be passed on to the application. The XML application acts upon the tree structure and processes the data it contains.

3.2 XML-Linking

The problems of linking in HTML are well known. Links are hard-coded into documents, difficult to maintain, limited in behaviour and notoriously fragile. With the development of XML there has been a concerted attempt to overcome these problems and to provide more powerful linking schemes, XML-Linking is the result. XML-Linking is currently specified in a W3C working draft [Maler and DeRose, 1998]. It provides hyperlinking support in two parts. XLink defines a language for asserting link existence and for describing link characteristics. XPointer is a language that supports linking to particular parts within XML documents. Figure 2 shows the concept of XLink and XPointer.

XLink defines a syntax for locators of linked elements. A locator consists of two parts, a URI identifying the file of the destination element of the link and an optional XPointer expression identifying

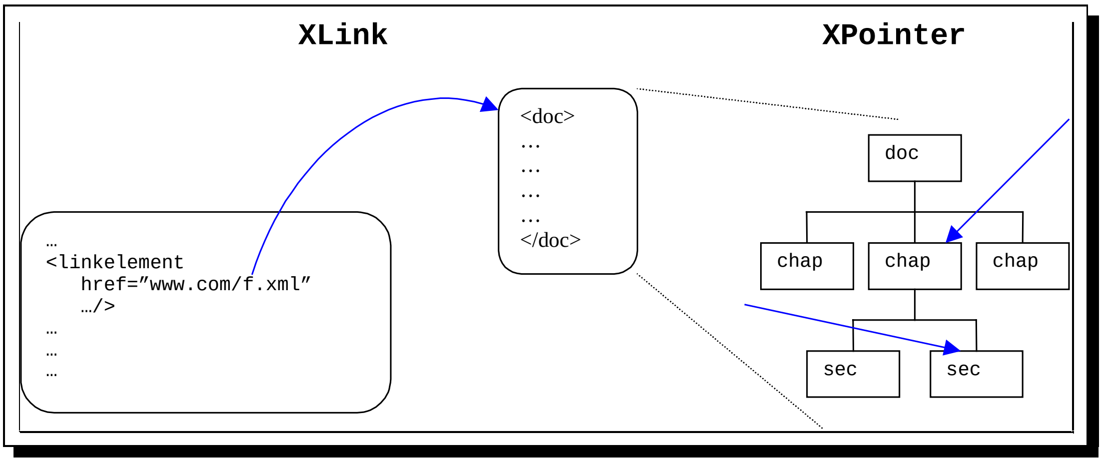


Figure 2: Relationship between XLink and XPointer

the XML element within the destination document. An example locator might look like that shown in Figure 3 below.

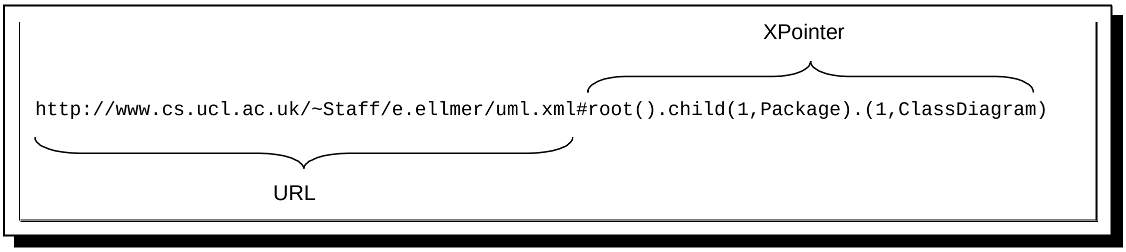


Figure 3: Links in XLink

The XLink approach to link recognition is to reserve attributes for characterising links. An element is identified as an element possessing a link through an attribute xlink:form which can have values simple, extended, etc. The locator shown in the example above is given as the value of an attribute href.

XPointer consists of a series of location terms that represent an element or a set of elements within an XML document by navigating from the root element to the target elements. XPointer defines absolute location terms such as `root()`, `origin()` and `id()`. `Root()` refers to the root element of an XML element tree, `origin()` refers to an element where a user started traversal, `id()` allows to unambiguously identify one element in an XML document by evaluating id attributes of the elements. Relative location terms are used to navigate through the XML element tree starting at an absolute location. Among them are child, descendent, preceding and following.

A set of further attributes is reserved for specifying link semantics. An attribute inline determines if a link is in-line or an out-of-line link, the attributes show and actuate specify traversal policies. Show can have values new, replace, and embed meaning that the link destination should be shown in a new window, replace the current document, or be embedded into the current document respectively. Actuate can have values auto and user. The attributes role and title contain information about the linking source for the processing application and for the user respectively. Content-role and Content-title provide the same information for remote resources, i.e. link destinations.

3.3 XSL

XSL (eXtensible Stylesheet Language) [Deach, 1999] is the language designed as part of the overall XML effort, to be used to specify a presentation for an XML document. A key component of the XML philosophy is the separation beween presentation and content. XSL is a description of the mapping between content elements and presentation elements. A single XML document can have many presentations defined in XSL corresponding to different views of the document. An XSL stylesheet can be used to translate a document into another format (most commonly, HTML), thus providing a mechanism for viewing XML documents within a standard browser.

An XSL stylesheet processor accepts an XML document and an XSL stylesheet and produces a presentation document for the XML source, based on the stylesheet. The process is divided into two sub-processes. The first sub-process constructs a result tree from the XML source tree. A stylesheet is composed of tree construction rules. A tree construction rule contains a pattern that identifies the set of elements in the source tree for which this rule is relevant, and a template that is instantiated to construct a portion of the result tree. A template contains elements that specify literal result element structure and instructions for creating fragments of a result tree. The instructions can also select descendant elements. Figure 4 shows an example of an XSL tree construction rule. The second sub-process then interpretes the result tree to produce a formatted presentation.

```
<xsl:stylesheet xmlns:xsl=''http://www.w3.org/XSL/Transform/1.0''>
  <xsl:template match=''Document''>
    <DIV style=''font-weight:bold; colour:blue''>
      <xsl:apply-templates/>
    </DIV>
  </xsl:template>
</xsl:stylesheet>
```

Figure 4: Example of an XSL Construction Rule.

3.4 DOM

Associated with XML is the DOM (Document Object Model) [Apparao et al., 1998]. The DOM is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and how to access and manipulate a document. DOM allows programmers to build documents, navigate their structures, and add, modify, or delete elements and content. In DOM, the documents have a logical structure that is similar to a tree. However, DOM is a logical model that may be implemented in any convenient manner, not necessarily as a tree.

The name *Document Object Model* reflects the modelling of documents using objects, and the model encompasses the structure of a document and its behaviour. The DOM identifies (a) the interfaces and objects used to represent and manipulate a document, (b) the semantics of these interfaces and objects, and (c) the relationships and collaborations among these interfaces and objects.

The DOM consists of two parts: DOM Core and DOM HTML. The DOM Core contains interfaces for accessing and manipulating XML documents and serves as the basis for DOM HTML. The DOM HTML contains interfaces for accessing and manipulating HTML contents.

4 Example Problem

In this section we outline a consistency management problem, which we will use for illustrative purposes in the discussion which follows. The example problem relates to the management of software engineering documents expressed using the Unified Modelling Language (UML) [Booch et al., 1999]. UML yields good examples for our purposes because documents constructed in UML have a rich set of consistency relationships of varying degrees of complexity derived from the common UML metamodel.

The particular documents are drawn from an analysis of a meeting scheduler, a standard problem in software engineering [Feather et al., 1997]. A meeting scheduler handles the scheduling of meetings amongst different participants that are in different places and have personal diaries.

In our example problem we consider the situation in which a meeting scheduler is being developed by a distributed team. Multiple distributed documents are created and it is necessary to manage consistency of these distributed documents. We consider for illustrative purposes three distributed documents constructed using the UML: an associate participant collaboration diagram; a business entities class diagram; and a create meeting collaboration diagram. These diagrams are shown in Appendix A. We use an XML representation of these diagrams. The DTDs for the collaboration and class diagrams are described in Appendix B. These DTDs are based on the UXF (UML eXchange Format) DTD. We have selected the UXF DTD for simplicity, though a more complete DTD is available for UML, the XMI SMIF (Stream-based Model Interchange Format) adopted as a standard by the OMG (Object Management Group) [OMG, 1998].

5 Approach

In this section we outline our approach to consistency management and describe how it works. We discuss how consistency relationships are defined, how checks are made and how the results of the checks can be viewed. We look at consistency notification, the process by which changes are notified and the consistency of the relevant documents are incrementally determined. Figure 5 below gives an overview.

The basic goal of our appoach is to establish consistency links among elements of distributed documents bound together by consistency relationships. These relationships are specified in rules. A link is a relationship between two or more data objects or portions of data objects. We will use the consistency links for navigation and identifying inconsistencies.

5.1 Definition

The first component of the approach consists of defining consistency in a manner appropriate to the particular context.

The initial step involves identifying the types of documents that are relevant to the process. After identifying these documents it is necessary to obtain or specify the DTDs for these documents. For example, DTDs for the different types of UML diagram. It is, we believe, reasonable to assume that for most standard document types DTDs will be readily available, as for the XMI DTDs.

Based on the DTDs, the next step consists of specifying consistency relationships between the con-

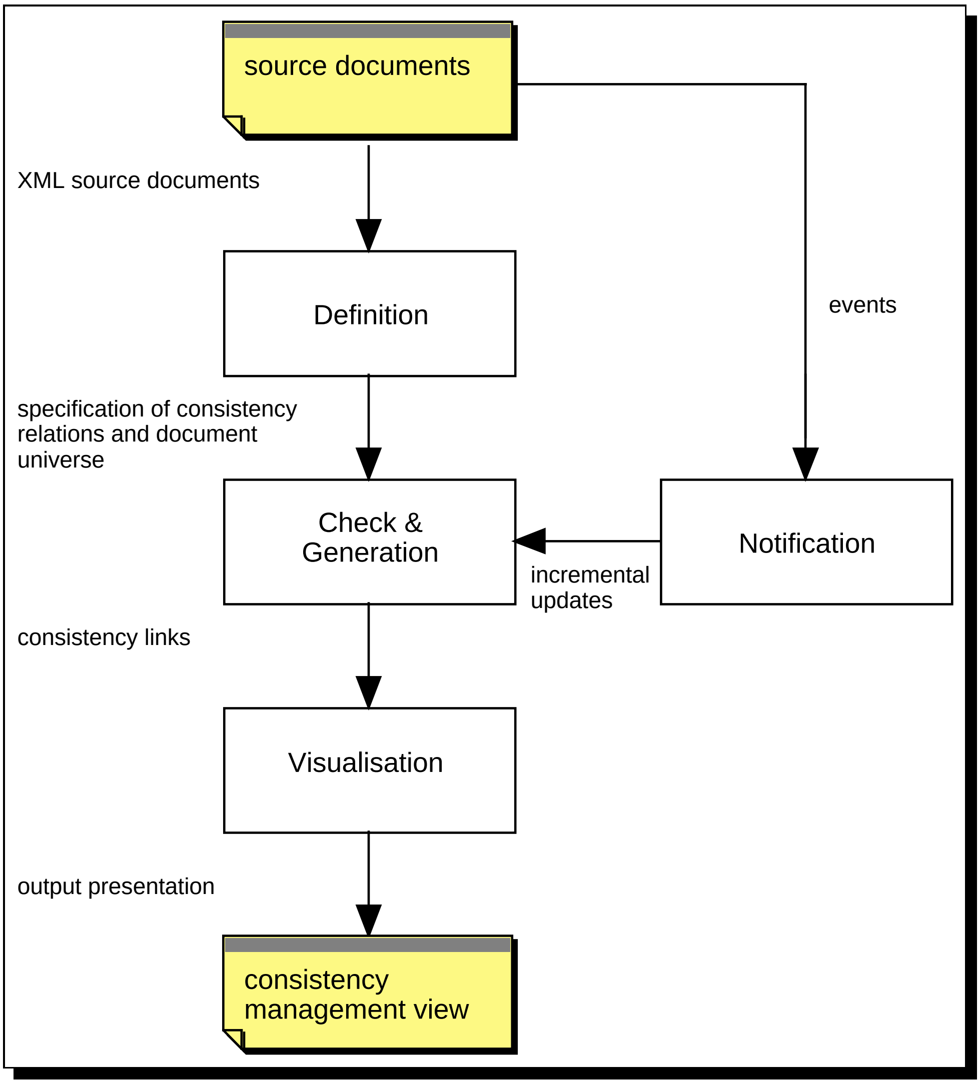


Figure 5: An Overview of our Approach to Consistency Management

cepts in the documents. Thus, the person responsible for managing the consistency of the documents analyses the concepts deployed in the documents and defines the consistency relationships between them. This needs only be done once for each set of document types. The consistency relationships are expressed through consistency rules, where a rule describes some form of relationship or fact that it is required to hold.

In our approach we define consistency rules using extended XPointer. Aside from the obvious benefits of standardisation the reasons for using XPointer are twofold. Firstly, XPointer eases the implementation of a tool for generating consistency links since it can be based on existing support for parsing and the like (for example, IBMs XML for Java). Secondly, changes in the DTDs do not require the link generator itself to be modified.

The scope of consistency rules currently supported by our approach includes rules for checking existence of related documents, existence of related elements in different documents, transitive closure properties, and comparison between elements and fixed values. Examples of the different types of rules are described in subsection 5.1.1.

Figure 6 presents the DTD specification for documents expressing consistency rules. The id element contains a unique ID of a rule within a document. The type element contains three different values used by the link generator; a detailed description of this element is described in subsection 5.1.1. The Description element contains a natural language explanation of the rule. The Source and Destination elements contain XPointer expressions for identifying possible related elements to be linked. A set of one or more Condition elements represent the conditions of the rule to be satisfied in order to establish a consistency link between an instance of the source elements and an instance of the destination elements. A condition consists of three elements. An expression that is evaluated for the source element (expsource), an expression that is evaluated for the destination element (expdest), and an operator (op). The evaluation of each expression produces a value of type string. The two values are then compared using the specified operator. If this comparison returns TRUE, the condition is fulfilled.

```
<!ELEMENT ConsistencyRule(id,Description,Source,Destination,Condition+)>
<!ATTLIST ConsistencyRule  id          CDATA          #REQUIRED
                           type        (CT|CF|IF)     #REQUIRED >
<!ELEMENT id (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Source (XPointer)>
<!ELEMENT Destination (XPointer)>
<!ELEMENT XPointer (#PCDATA)>
<!ELEMENT Condition EMPTY>
<!ATTLIST Condition expsource   CDATA   #REQUIRED
                    op          CDATA   #REQUIRED
                    expdest     CDATA   #REQUIRED>
```

Figure 6: Consistency Rule Syntax

As an example consider the following consistency rule r1 for UML diagrams: *for every instance in a collaboration diagram there must exist a class in a class diagram*. Figure 7 gives an example of this consistency rule (for the DTDs in Appendix B).

We assume that the distributed documents participating in the consistency management process are available in an XML format and are accessible using http. The XML documents can either be stored in XML by the tools that are used to process them as for example in Office2000 or IBM VisualAge tools, or they might be converted into XML as for example in the Rose Petal to XML as developed as part to the UXF effort. The universe of the documents participating in the consistency management process is identified by their URLs.

5.1.1 Check & Generation

After defining the document universe and establishing the consistency relationships at the document type level, the next step consists of executing consistency checks, constructing relevant consistency links and identifying inconsistencies, if any.

We have developed a *link generator* to generate consistency links automatically, based on the consistency rules. The tool evaluates the consistency rules relevant to each pair of documents. In the first step, the link generator identifies potential Source and Destination elements, based on the respective XPointer specification (see figure 7). For each pair of Source and Destination elements, all the conditions of the rule are checked. Depending on the type of the consistency rule and on the result of the evaluation of the conditions, a consistency link element is created.

```
<ConsistencyRule        id="r1"                        type="CT">
   <id>r1</r1>
   <Description>
    For every instance in a collaboration diagram there must be a
    class in a class diagram with the same name.
   </Description>
   <Source>
      <XPointer>
       root().child(all,Package).(all,CollaborationDiagram).
       (all,Collaboration).(all,Instance)
      </XPointer>
   </Source>
   <Destination>
      <XPointer>
       root().child(all,Package).(all,ClassDiagram).(all,Class)
      </XPointer>
   </Destination>
   <Condition   expsource="origin().attr(CLASS)"
                op="equal"
                expdest="origin().attr(NAME)"/>
</ConsistencyRule>
```

Figure 7: An Example Consistency Rule

Figure 8 presents the DTD specification for consistency link elements. The Locator element contains XPointer expressions for a linked element. The State element can contain value “consistent” or “inconsistent”, meaning that the linked element is either consistent or inconsistent for the current rule (`ruleid` attribute).

```
<!ELEMENT ConsistencyLink (State, Locator+)>
<!ATTLIST ConsistencyLink
        xml:form        CDATA           #FIXED "extended"
        inline          (true|false)    "false"
        ruleid          CDATA           #REQUIRED>
<!ELEMENT Locator (#PCDATA)>
<!ATTLIST Locator
        xml:form        CDATA           #FIXED "locator"
        href            CDATA           #REQUIRED
        html-href       CDATA           #IMPLIED>
<!ELEMENT State (#PCDATA)>
```

Figure 8: Consistency Link Syntax

Before we describe the possible types of consistency links it is necessary to explain the different types of consistency rules: CT, CF, and IF. The first argument (C or I) determines whether two elements are linked because they are in a consistent or inconsistent State, respectively. The second argument (T or F) specifies if the consistency rule is or not mandatory (true or false, respectively). If the value is true (T), then for every Source element it is necessary to have a Destination element, otherwise an inconsistency occurs. On the other hand, if the value is false (F) and there is no Destination element for a Source element, it does not mean that an inconsistency has occurred. Note that it does not make sense to have the case IT (inconsistent and mandatory), which would mean that there have to be inconsistencies in the document universe!

In order to illustrate the different types of consistency links consider consistency rule r1. As presented in figure 7, the type is CT since there has to be a class (Destination) for every instance in a collaboration diagram (Source). Figure 9 presents the consistency link for instance Meeting, in Create Meeting collaboration diagram, and class Meeting, in business entities class diagram presented in appendix A. In this case, the consistency link element has State consistent and both Locators exist. However, applying rule r1 to instance OrganiserWindow in the Create Meeting collaboration diagram the consistency link element is as shown in Figure 10. Due to the fact that OrganiserWindow does not have an associated class in any class diagram (i.e. business entities class diagram), the consistency link element has State inconsistent and only one specified Locator.

```
<ConsistencyLink
      xml:form="extended"
      inline="false"
      ruleid="r1">
<State>consistent</State>
<Locator xml:form="locator"
 href="http://www.cs.ucl.ac.uk/XML/create_meeting_col_diag.xml#root().
 child(1,Package).(1,CollaborationDiagram).(1,Collaboration).(4,Instance)"
/>
<Locator xml:form="locator"
 href="http://www.doc.ic.ac.uk/XML/business_entities_class_diag.xml#
 root().child(1,Package).(1,ClassDiagram).(5,Class)"
/>
</ConsistencyLink>
```

Figure 9: Example Link indicating Consistency

```
<ConsistencyLink
      xml:form="extended"
      inline="false"
      ruleid="r1">
<State>inconsistent</State>
<Locator xml:form="locator"
 href="http://www.cs.ucl.ac.uk/XML/create_meeting_col_diag.xml#root().
 child(1,Package).(1,CollaborationDiagram).(1,Collaboration).(2,Instance)"
/>
</ConsistencyLink>
```

Figure 10: Example Link indicating Inconsistency

Consider consistency rule r2: *classes with the same name in different class diagrams are considered to be identical*. In this case the consistency rule type is CF, since it is not mandatory for a class to exist in all class diagrams. Therefore, if a class name exists in two class diagrams a consistency link element is created with State consistent and with two respective Locators. On the other hand, for every class name that exists in only one class diagram, a consistency link is created with State consistent and only one Locator. In this case, it does not mean that an inconsistency occurred.

Our approach allows the representation of rules which relate to transitive closure properties. Consider rule r3: *for every class C1 with subclass C2 in a class diagram, it is not permitted to have C2 as a superclass of C1 in another class diagram*. This is a consistency rule of type IF, where two elements are linked if they are inconsistent and a Destination element does not have to exist for every Source element. Thus, if class C2 is a subclass of C1 in one diagram and class C2 is a superclass of C1 in another diagram, then a consistency link is created with State inconsistent and two Locators representing element class C1 in each diagram. However, if class C2 is not a superclass of C1 in another diagram, then a consistency link with State consistent and only one Locator is created. In this case it does not mean that an inconsistency occurred.

The consistency links specified by the link generator are stored in special documents named XML'. A XML' document is created for each XML document participating in the consistency management process. A XML' document is composed of the information in the related XML document and relevant consistency links. The idea of storing the consistency links in XML' documents permits a distributed execution of the consistency management process and preserves the original information in the XML documents. In addition, it supports the visualization process, as presented in subsection 5.2, since XSL requires that link elements are specified in the same XML document as the elements they relate to.

5.2 Visualization

For convenience XML documents are converted into common HTML markup format to be displayed in the popular browser software.

The example XSL stylesheet we have developed contains construction rules which allow the generation of HTML views from source XML documents. We have used the Microsoft XSL processor as a formatter to generate HTML output. However, popular web browsers (Internet Explorer and Netscape) provide internal rendering mechanisms and directly apply XSL stylesheet formatting to XML documents.

In our example every class on the Class Diagram has its own ID attribute, and a construction rule defines the formatting for element ID representation of element Class. Since Messages on the Collaboration Diagram also carry an ID attribute, another rule specifies formatting specific to the ID attribute of the Message element.

When all XML' documents (that is the XML documents plus the consistency links) in the document universe have been transformed to HTML by means of an XSL script, the resulting set of interlinked HTML pages provides users with ease of browsing through the document universe with a popular browser client. Again, in our example, the HTML representation for the Class Diagram XML' document will list all classes, class attributes, class operations and class associations. Each class operation may contain references to other classes, these references are represented as links, and a user is able to follow these links within their browser to navigate to a particular element within the document universe.

Users can navigate to objects within the same document as well as in any other document within the document universe. This is achieved by using the same name for the HTML representation as the source XML' file, with the exception of an additional .html extension. Each element within the HTML representation is marked up with its ID as a NAME anchor tag, which allows the user to navigate directly to the element of interest in the document the link points to.

The display of complex diagrams in browsers is another interesting issue. We have developed a portable, distributed and interoperable approach called *Browsing Object in XML (BOX)* to support viewing complex UML models in standard browsers. BOX translates UML models represented in XMI into VML (Vector Graphic Markup Language) [Mathews et al., 1998] so that they can be displayed in Internet Explorer 5. Figure 11 shows the business entities class diagram generated by BOX and viewed in Internet Explorer 5. For a detailed description of BOX we refer to [Nentwich et al., 1999].

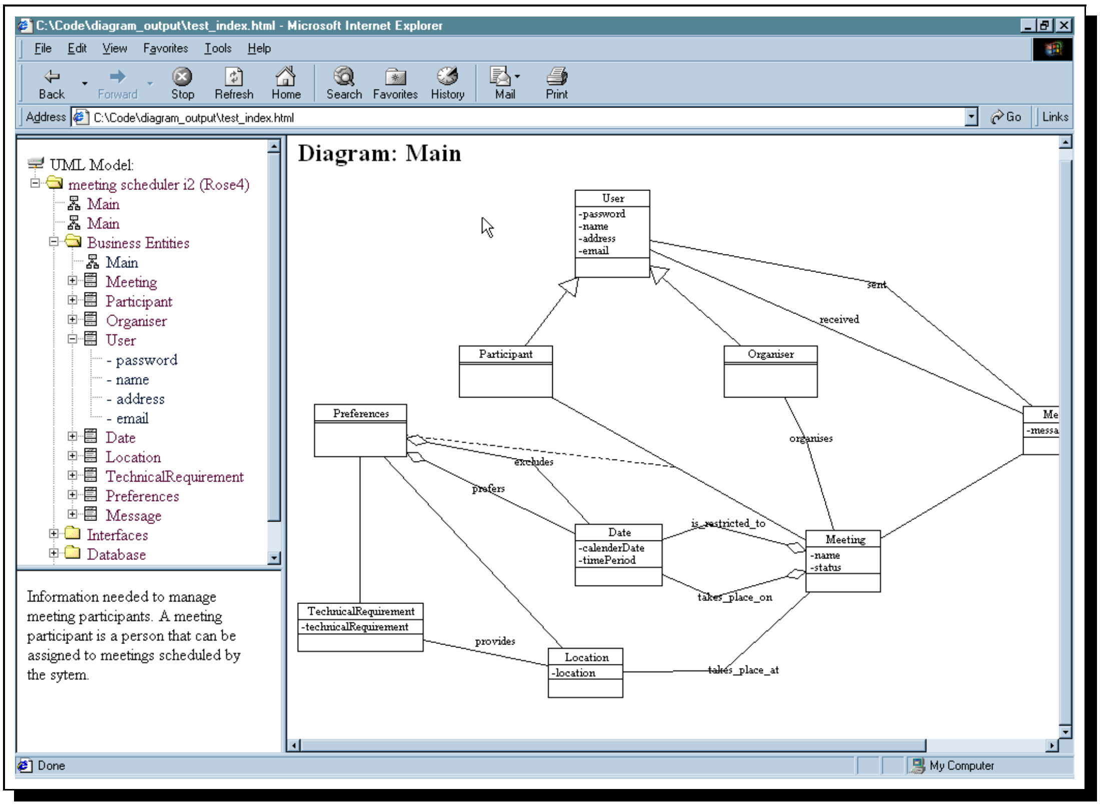


Figure 11: BOX View of Business Entities Class Diagram

5.3 Monitoring

After establishing consistency links for a document universe, changes to the target documents have to be monitored, and if a change occurs, further consistency checks may have to be initiated. We have adopted a relatively straightforward approach to this problem, though more sophisticated strategies are possible. In particular using the WebDAV protocol [Goland et al., 1998] which provides support for distributed authoring and versioning and/or by tools which publish relevant events.

In our toolkit monitoring is performed by a "watch dog", which compares documents regularly (every x minutes or after receiving a change event notification from an external tool) with their previous version. To do this comparison we rely on an algorithm for computing the difference between two trees, in our case the DOM representations of the previous and current version of the XML document. A change notification and an account of the relevant changes are generated. We can then invoke incremental consistency checks. The elements that are affected by a change and their consistency relationships are identified. The consistency links are recalculated, a new XML' generated, the stylesheet applied and the new state of the document universe can then be visualised in the manner above.

6 Toolkit

In this section we set out the architecture of our consistency management toolkit and highlight the major components of that architecture.

The toolkit is written in Java using JDK 1.1.6 together with Swing 1.1 beta for the graphical user interface. The components are implemented as distributed objects communicating through Java/RMI.

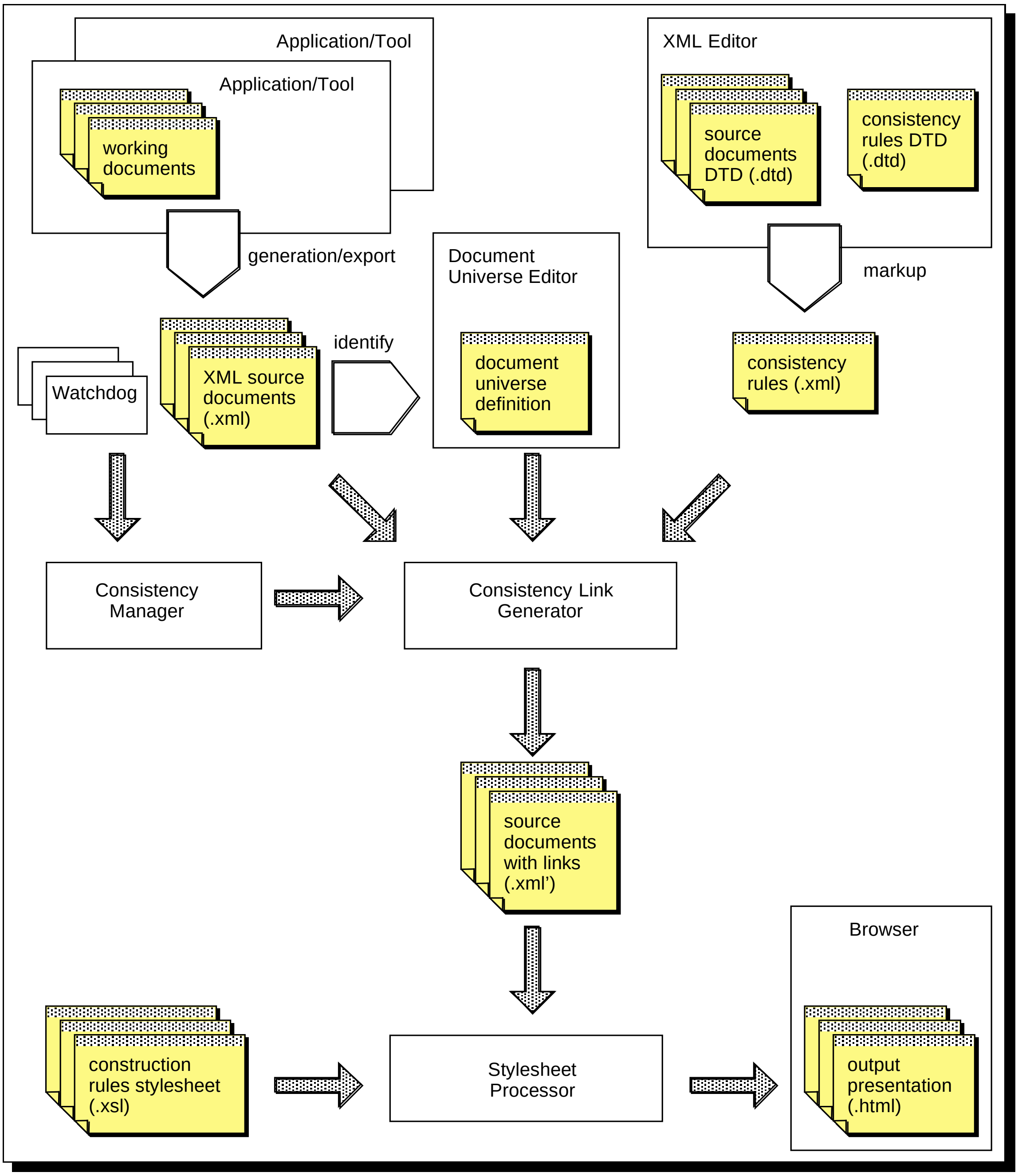


Figure 12: Architecture

The proof-of-concept prototype we developed to support our approach comprises six components, namely:

- XML Editor
- Document Universe Editor
- Consistency Link Generator
- Consistency Manager

- Watch Dog
- Stylesheet Processor

The front-end of the toolkit we developed is the XML Editor. It provides functionality for creating and editing XML, DTD and XSL files as well as to parse and process them. The user interface of the XML Editor is shown in Figure 13. There is one pane for each type of file, representing an XML document as text and as tree structure, representing a DTD as text and as tree structure, representing an XSL stylesheet as text, and presenting the output of processing an XSL stylesheet as text. The text panes provide the usual functionality for copying, pasting, deleting, and the like. The example screenshot in figure 13 shows an open XML document that has been parsed without an error. The text window in the lower part of the user interface presents an area for displaying messages, such as parsing results. The XML Editor can invoke the Stylesheet Processor component for generating HTML files that can be viewed with a standard web browser such as Netscape Navigator or Microsoft Internet Explorer. The XML Editor uses the IBM XML for Java library for parsing XML documents.

The Document Universe Editor allows a user to edit and control a document universe (see Figure 14). The Documents pane of the Document Universe Editor shows all the documents that are currently part of the universe, if they are currently being monitored, and if they were changed. Documents can also be added to or removed from the universe. The Consistency Rules pane provides a graphical interface for editing the rules. This is particularly important because users cannot be expected to write their rules using the extended XPointer expressions we introduce. The Consistency Links pane allows the consistency link generator to be invoked and the transformation of consistency information into HTML format to be started, i.e. calls the XSL Processor component.

The Consistency Link Generator is responsible for generating consistency links between elements of the document universe according to the rules defined. It is invoked from the Document Universe Editor. It takes as input the document universe definition containing the documents that are part of the universe and the consistency rules that have to hold between them. The consistency link generator collects the documents that are part of the universe from their web location and checks them with respect to the consistency rules. The output of the Consistency Link Generator are XML' documents containing relevant consistency link elements.

The Watch Dog component is responsible for monitoring the state of an XML source document within the document universe. It compares the document contents with the last version and if a change occurs, notifies the Consistency Manager about the change and about the elements that have changed. For doing this, the Watch Dog uses the XMLTreeDiff library from IBM AlphaWorks. The input for the Watch Dog therefore is the XML document that is currently monitored. At the moment each Watch Dog monitors only one document.

The Consistency Manager is responsible for triggering actions arising from changes that are reported by a Watch Dog. The Consistency Manager takes the change notification as input, as well as the related XML' documents containing the linking information, and triggers an incremental regeneration of those links that might be affected by the changes reported by the Watch Dog.

The Stylesheet Processor is a component that takes as input an XML document and a XSL stylesheet and generates an HTML file. We have used the Microsoft msxsl processor for this purpose. Figure 15 presents an example of the HTML representation for the associate participant collaboration diagram. The example contains two classes that are inconsistent with respect to rule r1 (*MeetingOrganiser* and *OrganiserWindow*), and a consistent example with its respective link (*Message*). In this latter case, the Object Reference links the current instance Message to its respective class in the business entities class

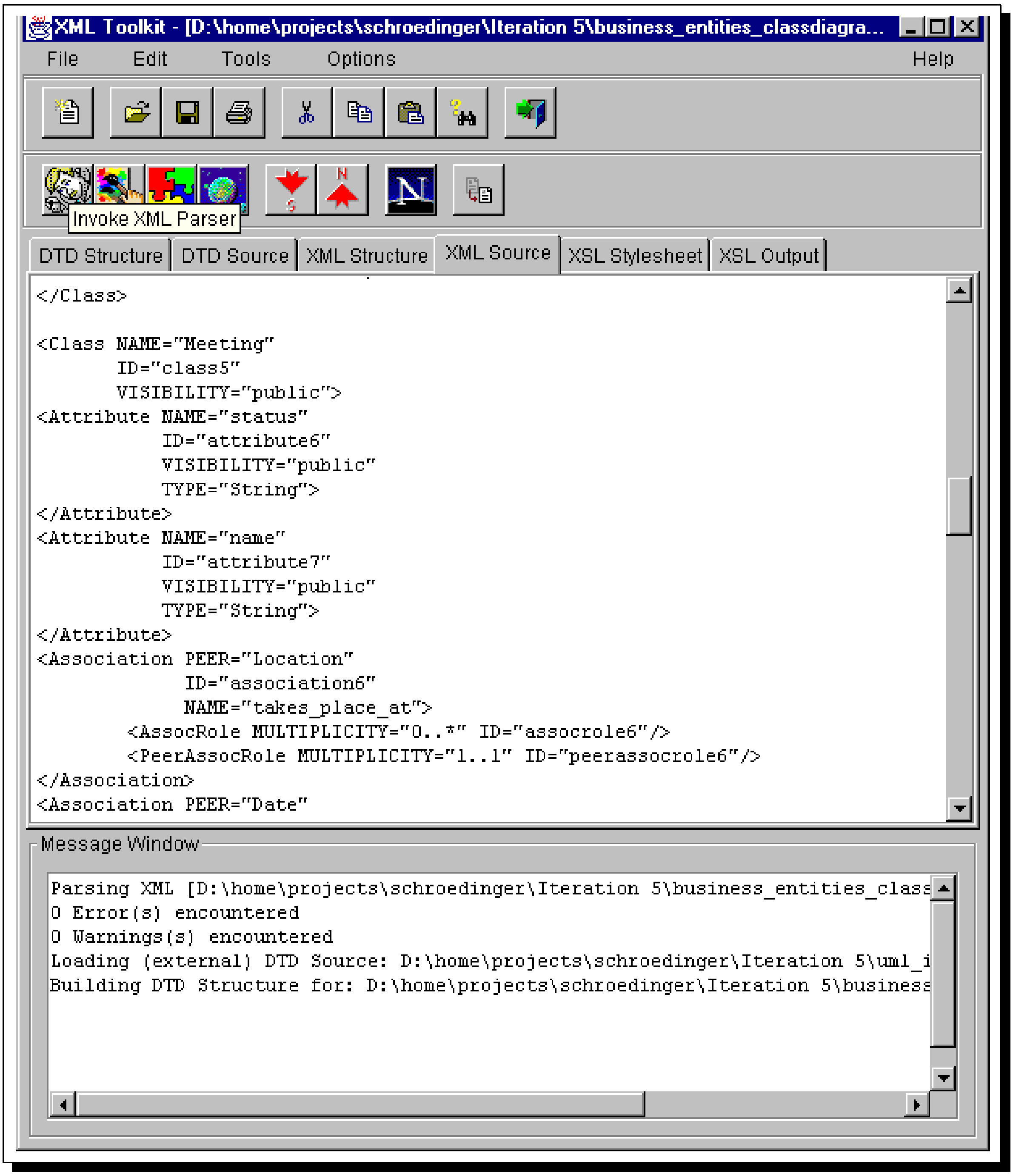


Figure 13: XML Editor User Interface

diagram, as shown in Figure 16.

7 Related Work

In this section we point to work in areas related to that described above. Particular emphasis is placed on those pieces of work which have been influential in the development of the key ideas. We briefly

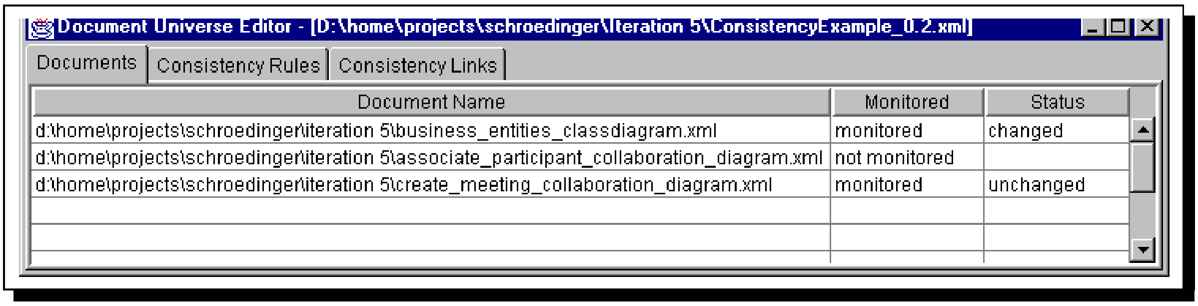


Figure 14: Document Universe Editor Screenshot

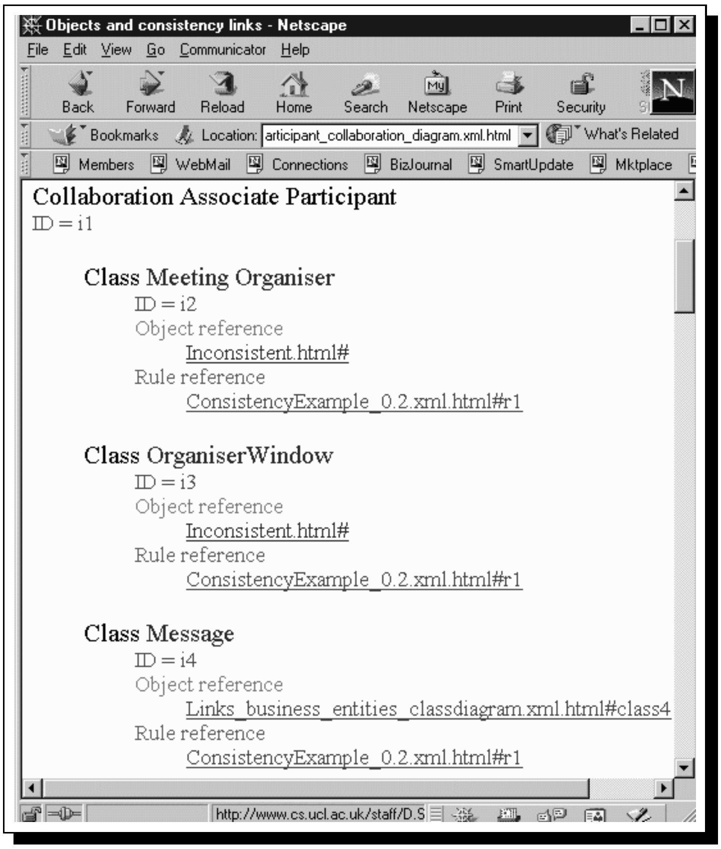


Figure 15: Browser View of Output Presentation

review contributions in the growing area of consistency management, work on mutiple views in software development, work on graph-based programming and development environments, recent work on internet scale event notification and change notification, and work in the area of dynamic hot-linking of hypertexts.

A large body of work has been carried out on programming language environments from which systems, such as the Cornell Synthesizer Generator [Reps and Teitelbaum, 1984], Pecan [Reiss, 1984], Centaur [Borras et al., 1988], Gandalf [Habermann and Notkin, 1986] and IPSEN [Engels et al., 1992] evolved. These systems detect consistency constraints of software engineering documents that are derived from static semantic rules of the underlying programming languages. Inconsistency detection in these systems is based on navigation in attributed abstract syntax trees or graphs. The XML representation of software engineering documents that we use directly reprensents attributed abstract syntax

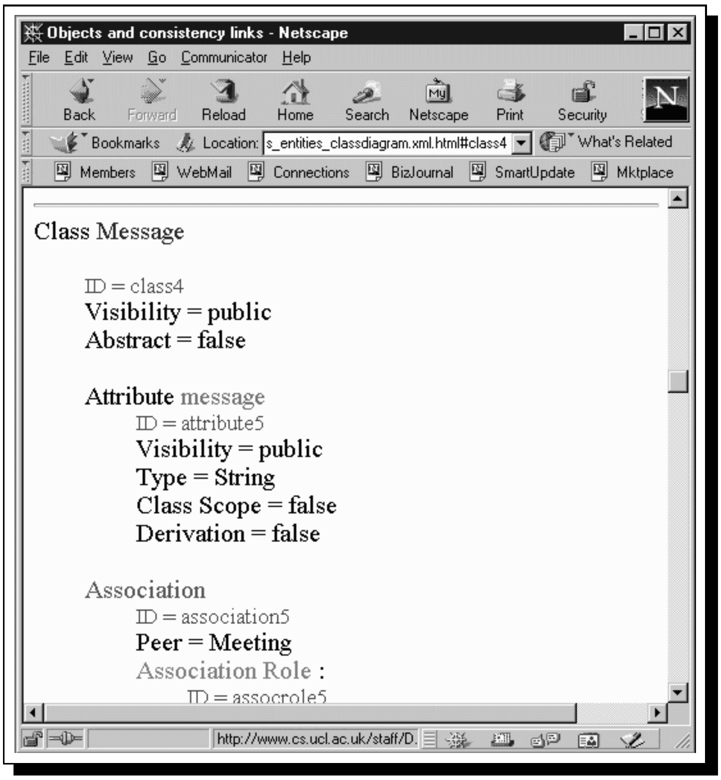


Figure 16: Browser View of Output Presentation

trees and the consistency links that we generate turn them into abstract syntax graphs. However, unlike these systems we do not build on proprietary formalisms and technology. We use XML for the representation of both the rules and the documents and we use standard XML tools for inconsistency detection and achieve Internet scale distribution in by these means.

The work on programming environments was generalized into research on software development environments in projects, such as Arcadia [Taylor et al., 1988], ESF [Schäfer and Weber, 1989], ATOMOSPHERE [Boarder et al., 1989] and GOODSTEP [GOODSTEP Team, 1994]. These environments integrate tools for different languages and incorporate consistency constraints that span across different documents. All of them utilize a repository, such as PCTE [Boudier et al., 1989] or an object database for storing documents. The repository's schema definition language is used to define the structure of the document types supported by the environments. Our use of XML DTDs corresponds to the schema definitions used in these SDEs. Most of the research into software development environments assumed a centralized repository which limits scalability and commitment on the part of both users and tool vendors to a heavyweight integration mechanism. By contrast our approach uses distributed web servers for storing documents and the standard http protocol for gaining distributed access to these.

Several development environments pursued the idea of presenting multiple different views of the documents produced during a software process for different users. The Gandalf system was extended by [Garlan, 1987] with capabilities to present multiple views of programs to a user. In their work on Multiview [Marlin, 1990] extended that idea to present different views to the different users involved in a software process by means of a centralized server. Both Multiview and Gandalf used bespoke languages to specify these views and implemented the technology needed themselves. In the GOODSTEP project we deployed the view mechanism of object databases [Abiteboul and Bonner, 1991] as an off-the-shelf mechanism for implementing different document views on the same conceptually centralized database [Emmerich, 1995]. With XSL, we use different standard mechanisms to achieve the same result. Our approach, however facilitates different views of fully distributed documents and therefore avoids the performance and scalability bottlenecks of the centralized server components of previous approaches.

The work presented here continues our research on viewpoints [Finkelstein et al., 1992]. Viewpoints are loosely-coupled, locally managed objects encapsulating a document, knowledge about how that document is represented and knowledge about the process by which the document is constructed. [Finkelstein et al., 1994] analyses techniques for consistency checking and management for distributed viewpoints. The work on consistency checking also includes [Easterbrook et al., 1994], which discusses in detail the structure and dynamics of such checks. These techniques have not been implemented in a properly distributed manner and the work reported above constitues a major step forward in this direction.

Consistency management itself is a topic in which there is a growing interest. We have set out an agenda for research in this area [Finkelstein et al., 1996]. Our approach to consistency management has been significantly influenced by work on the problem of "compliance" that is managing adherence of complex sets of documents to practices set down in standards [Emmerich et al., 1999]. In this work we have mapped out the relationship between event monitoring and consistency checking.

At the time of writing, there are several event notification and messaging schemes under development within the Web community. These are reviewed in [Khare and Whitehead, 1998] on which we have drawn for the comments below. An account of the architecture and properties of such schemes is given in [Rosenblum and Wolf, 1997].

In addition to the existing internet-scale messaging mechanisms (SMTP for mailing lists, NNTP for newsgroups, HTTP 1.1 for callbacks), there are a number of new candidate event-notification protocols under development. Basic Lightweight Information Protocol [Jensen, 1998] is based on a publish-subscribe model, delivers MIME notifications in real-time streams and makes application transactions services available. Several protocols have been proposed for notification of changes to the resources of interest to users or groups of users: Session Invitation Protocol is loosely based on HTTP, deploys multicast and unicast relations, and supports user mobility by proxying and request redirection; General Event Notification Architecture [Aggarwal and Cohen, 1998] allows notifications to be pushed or pulled, and enables a resource to establish a subscription relationship with any other resource to receive notifications of future events; RendezVous Protocol is designed to accommodate notifications for people subscribing to other people's status, and uses XML to mark-up different kinds of notifications. These protocols define extensions to HTTP 1.1 which are specific to event notification.

There is currently much activity around WebDAV – Distributed Authoring and Versioning on the World Wide Web – a set of extensions to HTTP 1.1. WebDAV enables server-based access control of remotely edited, distributed resources, promotes collaborative authoring and provides internet-scale event notification mechanisms. Event Notification Protocol [Reddy and Fisher, 1998] is based on WebDAV extensions to HTTP, and allows users to register interest in resources for later notifications of property and state value changes. Most of the protocols are still in the development stage and exist as either working drafts to the W3C Consortium or as proposals.

Other important developments are emerging event notification APIs, such as Java Message Service which has achieved wide industry support. JMS provides a common way for Java programs to create, send, receive and read messages in the distributed setting. It allows distributed applications to asynchronously exchange messages, and supports both major notification models: point-to-point and publish-subscribe.

There is a growing body of work concerned with hypertext applied in software engineering. CHIMERA [Anderson et al., 1994] demonstrates the application of many concepts now embedded in XML technologies to software engineering, notably multiple document views, the capability of separating linking information from the underlying documents and the interesting possibilities that combining these two presents. CHIMERA does not support consistency checking. CHIME [Devanbu et al., 1999] provides a framework for hyperlink insertion into legacy software documents using information from software analysis tools. The work provides a strong case for the sort of browsing which our approach provides. Our approach however delivers a significantly greater genericity, does not rely on an underlying repository, and uses linking technologies which are a great deal more powerful. A very interesting approach has been proposed by [Cattaneo et al., 1998]. LABYRINTH superimposes a schema mechanism on top of web-based distributed documents. The schema mechanism itself sits on top of ”shadow” HTML pages which describe the contents of the underlying documents. It would seem possible to extend our work to provide a schema mechanism of the form proposed by LABYRINTH, XML provides a much more convenient basis for doing so than HTML.

8 Evaluation & Future Work

In this section we outline our preliminary evaluation of the work described above. We outline our short, medium and long term plans with respect to work on the approach and more specifically the toolkit.

Currently our work is at an initial stage. We have a proof of concept implementation which has been evaluated on a number of relatively small examples. This has provided us with good grounds for confidence in our approach and of its scalability though clearly we would like to extend these to larger examples, this is an immediate priority. We are looking at a large domain object model provided by Eurocontrol, the European Air Traffic Control Organization. We have a number of other areas for future work which reflect current shortcomings in the toolkit.

Currently the document universe has to be directly specified. Providing a level of indirection through a registration service is an obvious extension to our approach and could be simply implemented using existing publish-subscribe services.

We recognise that XPointer provides an awkward syntax for specifying consistency checks. We therefore envisage a simple graphical tool which will help in writing such checks by displaying the DTDs and allowing links to be made between them. This involves some relatively trivial preprocessing but is an important step to make the approach more accessible.

We plan to extend the syntax rules of our consistency rules to allow for the specification of more complex conditions. Specifically to bind expressions to variables and reuse them in conditions. Work is under way to extend our approach to support the XMI DTDs. In addition, we strive to support consistency management among different types of documents. We are currently working on consistency check specifications between UML models and Z specifications. We are also looking at XML-QL [Deutsch et al., 1998], a query language for XML documents.

As discussed above, our notification scheme is very simple. We plan to build a more sophisticated scheme. Our preference is to use WebDAV but there are no useable implementations to date. Short of implementing WebDAV ourselves we will be looking at building a simple protocol on JMS as a practical alternative.

At the moment we have generally assumed that source XML documents are generated from tools and that the consistency management activity sits ouside those tools. With tools that manage internal storage in XML an interesting possibility is to use our approach to consistency management as a backplane to an existing tool or tool suite. The implications of doing so need to be further analysed.

9 Acknowledgements

The authors gratefully acknowledge the comments and critical feedback of their colleagues in particular Jon Crowcroft, Alfonso Fugetta, Jeff Kramer and Jeff Magee. We would like to thank Jerry Watson and Patrick Abbott of Eurocontrol for providing an industrial size case study. We would also like to acknowledge financial support from the European Union (RENOIR and PROMOTER), Schroedinger Foundation (support for Ernst Ellmer) and Unipower Ltd. (support for Danila Smolko).

10 References

Abiteboul, S. and Bonner, A. (1991). Objects and Views. *ACM SIGMOD Record*, 20(2):238–247. Proc. of the 1991 ACM SIGMOD Conf. on Management of Data, Denver, Co.

Aggarwal, S. and Cohen, J. (1998). General Event Notification Architecture Base. Internet Draft http://www.ietf.org/internet-drafts/draft-cohen-gena-p-base-01.txt, Internet Engineering Task Force.

Anderson, K. M., Taylor, R. N., and Whitehead, E. J. (1994). Chimera: Hypertext for Heterogeneous Software Environments. In *Proc. of the European Conference on Hypermedia, Edinburgh, UK*.

Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Hors, A. L., Nicol, G., Robie, J., Sutor, R., Wilson, C., and Wood, L. (1998). Document Object Model (DOM) Level 1 Specification. Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium.

Boarder, J., Obbink, H., Schmidt, M., and Völker, A. (1989). Advanced techniques and methods of system production in a heterogeneous, extensible, and rigorous environment. In Madhavji, N., Schäfer, W., and Weber, H., editors, *Proc. of the 1st Int. Conf. on System Development Environments and Factories, Berlin, Germany*, pages 199–206, London. Pitman Publishing.

Booch, G., Jacobson, I., and Rumbaugh, J. (1999). *The Unified Modeling Language User Guide*. Addison Wesley.

Borras, P., Clément, D., Despeyroux, T., Incerpi, J., Kahn, G., Lang, B., and Pascual, V. (1988). CENTAUR: The System. *ACM SIGSOFT Software Engineering Notes*, 13(5):14–24. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Mass.

Boudier, G., Gallo, F., Minot, R., and Thomas, I. (1989). An Overview of PCTE and PCTE+. *ACM SIGSOFT Software Engineering Notes*, 13(2):248–257. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Mass.

Bray, T., Paoli, J., and Sperberg-McQueen, C. M. (1998). Extensible Markup Language. Recommendation http://www.w3.org/TR/1998/REC-xml-19980210, World Wide Web Consortium.

Cattaneo, F., Fuggetta, A., Lavazza, L., and Valetto, G. (1998). Labyrinth: schema-based distributed management of documents. Technical report, CEFRIEL.

Deach, S. (1999). Extensible Stylesheet Language (XSL). Technical Report http://www.w3.org/TR/1999/WD-xsl-19990421, World Wide Web Consortium.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1998). XML-QL: A Query Language for XML. Technical report, World-Wide-Web Consortium. Submitted for Adoption.

Devanbu, P., Chen, Y.-F., Gansner, E., Muller, H., and Margin, J. (1999). CHIME – Customizable Hyperlink Insertion and Maintenance Engine for Software Engineering Environments. In *Proc. of the* 22^{nd} *Int. Conf. on Software Engineering, Los Angeles, Cal.* ACM Press. To appear.

Easterbrook, S., Finkelstein, A., Kramer, J., and Nuseibeh, B. (1994). Coordinating Distributed ViewPoints: The Anatomy of a Consistency Check. *Int. Journal of Concurrent Engineering: Research & Applications*, 2(3):209–222.

Emmerich, W. (1995). *Tool Construction for process-centred Software Development Environments based on Object Database Systems*. PhD thesis, University of Paderborn, Germany.

Emmerich, W., Finkelstein, A., Montangero, C., Antonelli, S., Armitage, S., and Stevens, R. (1999). Managing Standards Compliance. *IEEE Transactions on Software Engineering*, 25(6). To appear.

Engels, G., Lewerentz, C., Nagl, M., Schäfer, W., and Schürr, A. (1992). Building Integrated Software Development Environments — Part 1: Tool Specification. *ACM Transactions on Software Engineering and Methodology*, 1(2):135–167.

Feather, M. S., Fickas, S., Finkelstein, A., and v. Lamsweerde, A. (1997). Requirements and Specification Exemplars. *Automated Software Engineering*, 4(4):419–438.

Finkelstein, A., Gabbay, D., Hunter, H., Kramer, J., and Nuseibeh, B. (1994). Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578.

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., and Goedicke, M. (1992). Viewpoints: a framework for integrating multiple perspectives in system development. *Int. Journal of Software Engineering and Knowledge Engineering*, 2(1):21–58.

Finkelstein, A., Spanoudakis, G., and Till, D. (1996). Managing Interference. In Vidal, L., Finkelstein, A., Spanoudakis, G., and Wolf, A. L., editors, *Joint Proc. of the SIGSOFT '96 Workshops*, pages 172–174. ACM Press.

Garlan, D. (1987). *Views for Tools in Integrated Environments*. PhD thesis, Carnegie Mellon University.

Goland, Y. Y., Whitehead, J., Faizi, A., Carter, S., and Jensen, R. (1998). Extensions for Distributed Authoring on the World Wide Web – WebDAV. Internet Draft (Work in Progress) http://www.ietf.org/internet-drafts/draft-ietf-webdav-protocol-10.txt, IETF.

GOODSTEP Team (1994). The GOODSTEP Project: General Object-Oriented Database for Software Engineering Processes. In Ohmaki, K., editor, *Proc. of the Asia-Pacific Software Engineering Conference, Tokyo, Japan*, pages 410–420. IEEE Computer Society Press.

Habermann, A. N. and Notkin, D. (1986). Gandalf: Software Development Environments. *IEEE Transactions on Software Engineering*, 12(12):1117–1127.

Jensen, M. (1998). Basic Lightweight Information Protocol (BLIP). Draft http://www.blip.org/protocol.htm.

Khare, R. and Whitehead, J. (1998). XML and WebDAV: Emerging Web Standards and Their Impact on Software Engineering. Tutorial Notes ACM SIGSOFT '98.

Maler, E. and DeRose, S. (1998). XML Linking Language (XLink). Technical Report http://www.w3.org/TR/1998/WD-xlink-19980303, World Wide Web Consortium.

Marlin, C. (1990). A Distributed Implementation of a Multiple View Integrated Software Development Environment. In *Proc. of the 5^{th} Conf. on Knowledge-Based Software Assistant, Syracuse, NY*, pages 388–402.

Mathews, B., Lee, D., Dister, B., Bowler, J., Cooperstein, H., Jindal, A., Nguyen, T., Wu, P., and Sandal, T. (1998). Vector Markup Language. Technical Report http://www.w3.org/TR/1998/NOTE-VML-19980513, World Wide Web Consortium.

Nentwich, C., Emmerich, W., Finkelstein, A., and Zisman, A. (1999). Browsing Objects in XML. Research Note RN/99/41, University College London, Dept. of Computer Science.

OMG (1998). XML Meta Data Interchange (XMI) – Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF). Technical Report AD Document AD/98-10-05, Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA.

Reddy, S. and Fisher, M. L. (1998). Event Notification Protocol – ENP. Internet Draft (Work in Progress) http://www.ietf.org/internet-drafts/draft-reddy-enp-protocol-00.txt, IETF.

Reiss, S. P. (1984). PECAN: Program Development that Supports Multiple Views. *IEEE Transactions on Software Engineering*, 11(3):276–285.

Reps, T. W. and Teitelbaum, T. (1984). The Synthesizer Generator. *ACM SIGSOFT Software Engineering Notes*, 9(3):42–48. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, Penn.

Rosenblum, D. S. and Wolf, A. L. (1997). A design framework for internet-scale event observation and notivication. In Jazayeri, M. and Schauer, H., editors, *Software Engineering — ESEC/FSE '97, Zurich, Switzerland*, volume 1301 of *Lecture Notes in Computer Science*, pages 344–360. Springer.

Schäfer, W. and Weber, H. (1989). European Software Factory Plan – The ESF-Profile. In Ng, P. A. and Yeh, R. T., editors, *Modern Software Engineering – Foundations and current perspectives*, chapter 22, pages 613–637. Van Nostrand Reinhold, New York.

Suzuki, J. and Yamamoto, Y. (1999). Making UML models exchangeable over the Internet with XML: The UXF Approach. In Muller, P.-A. and Bezivin, J., editors, *Proc. of Int. Workshop on UML '98, Mulhouse, France*, Lecture Notes in Computer Science. Springer. To appear.

Taylor, R. N., Selby, R. W., Young, M., Belz, F. C., Clarce, L. A., Wileden, J. C., Osterweil, L., and Wolf, A. L. (1988). Foundations of the Arcadia Environment Architecture. *ACM SIGSOFT Software Engineering Notes*, 13(5):1–13. Proc. of the 4^{th} ACM SIGSOFT Symposium on Software Development Environments, Irvine, Cal.

A UML Diagrams

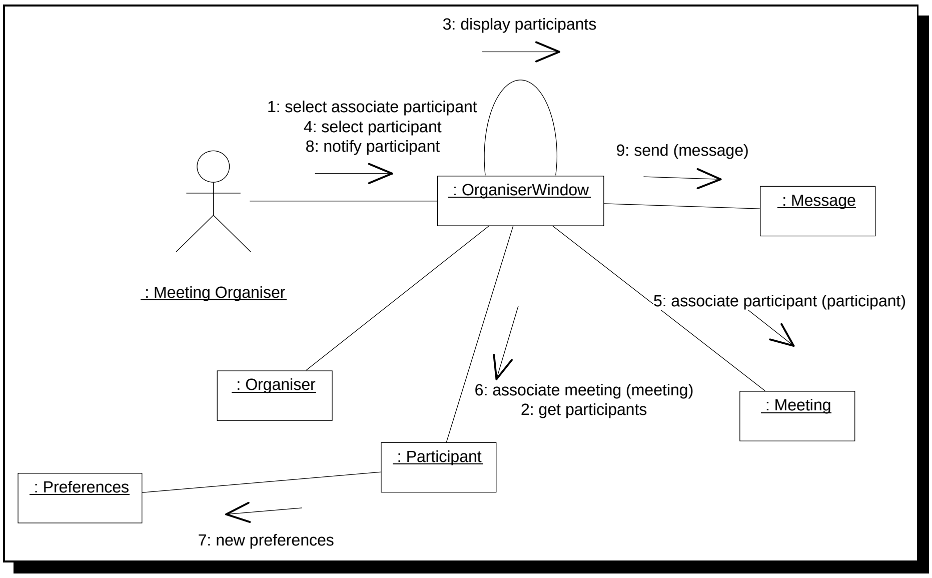


Figure 17: Associate Participant Collaboration Diagram

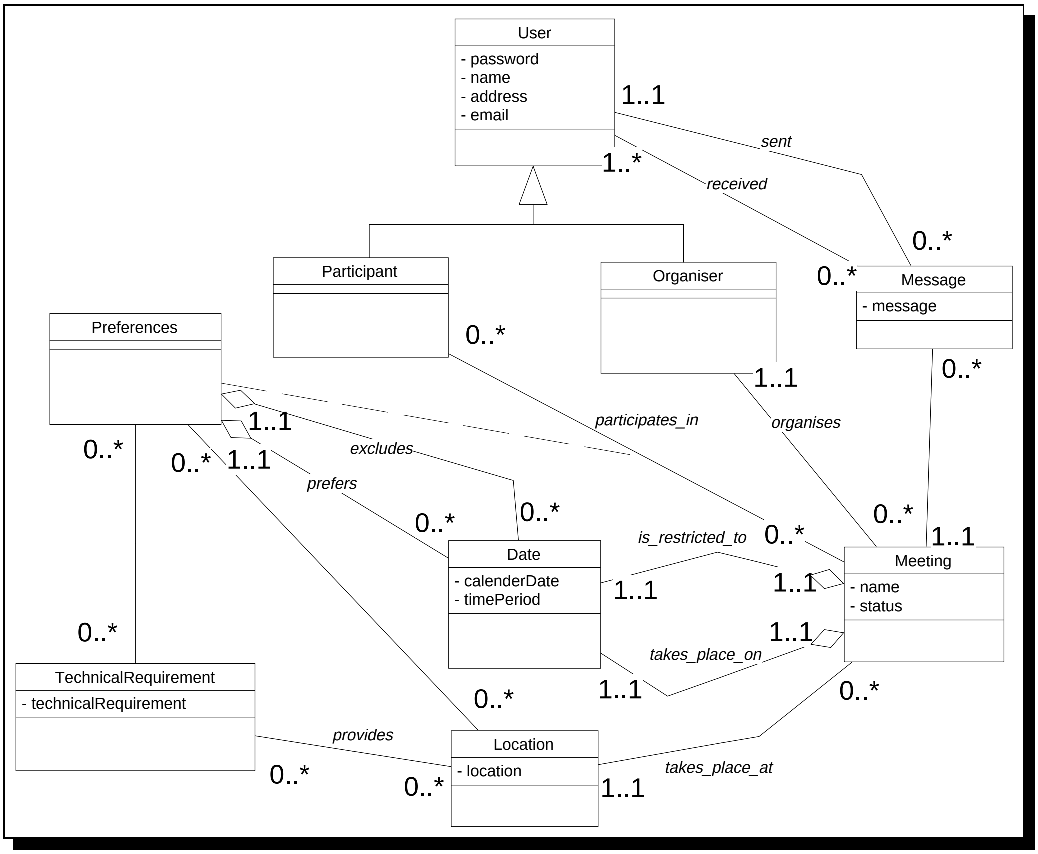


Figure 18: Business Entities Class Diagram

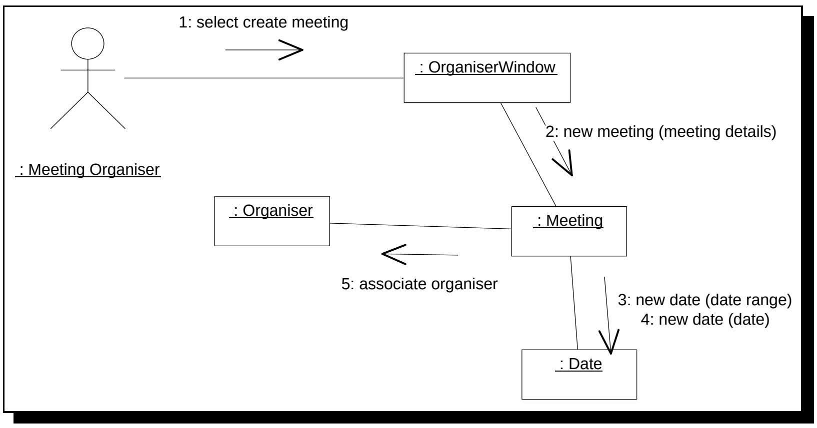


Figure 19: Create Meeting Collaboration Diagram

B Document Type Definitions

B.1 DTD Class Diagram

```
<!ELEMENT ClassDiagram (TaggedValue?, (Class
                                       |Interface
                                       |Note)*)>
<!ELEMENT Class %ObjectElements;>
<!ELEMENT Interface %ObjectElements;>

<!ATTLIST Class
   %id;
   NAME CDATA #REQUIRED
   ABSTRACT (true|false) "false"
   VISIBILITY (public|private) #REQUIRED
   ACTIVE (true|false) #IMPLIED>
<!ELEMENT Attribute (Note*)>
<!ATTLIST Attribute
   %id;
   VISIBILITY (public|protected|private) #REQUIRED
   TYPE CDATA #REQUIRED
   NAME CDATA #REQUIRED
   INITVAL CDATA #IMPLIED
   CONSTRAINT CDATA #IMPLIED
   DERIVATION   (true|false) "false"
   CLASSSCOPE (true|false) "false">
<!ELEMENT Operation ((Parameter|Exception|Note)*)>
<!ATTLIST Operation
   %id;
   VISIBILITY (public|protected|private) #REQUIRED
   NAME CDATA #REQUIRED
   RETURN CDATA #REQUIRED
   CLASSSCOPE (true|false) "false"
   CONCURRENCY (sequential|guarded|concurrent) "sequential"
   EXCEPTION CDATA #IMPLIED>
<!ELEMENT Parameter EMPTY>
<!ATTLIST Parameter
   %id;
   NAME CDATA #REQUIRED
   TYPE CDATA #IMPLIED
   DEFAULTVAL CDATA #IMPLIED
   DIRECTION (in|out|inout) #IMPLIED>
<!ELEMENT Exception EMPTY>
<!ATTLIST Exception
   %id;
   NAME CDATA #REQUIRED
   BODY CDATA #IMPLIED>
<!ELEMENT Generalization EMPTY>
<!ATTLIST Generalization
   %id;
   FROM CDATA #REQUIRED
   TYPE (public|private|protected) "public">
<!ELEMENT Association ((AssocRole, PeerAssocRole)| Note*)>
<!ATTLIST Association
   %id;
   PEER CDATA #REQUIRED
   NAME CDATA #IMPLIED>
<!ELEMENT AssocRole EMPTY>
<!ATTLIST AssocRole
```

```
   %id;
   MULTIPLICITY CDATA #IMPLIED
   ORDERING (ordered|unordered) #IMPLIED
   QUALIFIER CDATA #IMPLIED
   ROLENAME CDATA #IMPLIED
   NAVIGABILITY (true|false) "false"
   CHANGEABILITY (true|frozen|addOnly) "true"
   ASSOCCLASS CDATA #IMPLIED
   AGGREGATION (none|aggregate|composite) "none"
   AGGRKIND (unShared|shared) "unShared">
<!ELEMENT PeerAssocRole EMPTY>
<!ATTLIST PeerAssocRole
   %id;
   MULTIPLICITY CDATA #IMPLIED
   ORDERING (ordered|unordered) #IMPLIED
   ROLENAME CDATA #IMPLIED>
<!ELEMENT Dependency (Note*)>
<!ATTLIST Dependency
   %id;
   PEER CDATA #REQUIRED
   NAME CDATA #IMPLIED
   DESCRIPTION CDATA #IMPLIED
   DEPKIND (refine|bind) #IMPLIED>
```

B.2 DTD Collaboration Diagram

```
<!ELEMENT CollaborationDiagram (TaggedValue?, (Collaboration
                                              |Note)*)>
<!ELEMENT Collaboration (TaggedValue?, (Instance
                                       |Interaction
                                       |Message
                                       |Note)*)>
<!ATTLIST Collaboration
   %id;
    NAME       CDATA #REQUIRED
    CLASSIFIER CDATA #IMPLIED
    OPERATION  CDATA #IMPLIED>

<!ELEMENT Instance (Note*)>
<!ATTLIST Instance
   %id;
    NAME       CDATA #IMPLIED
    CLASS      CDATA #REQUIRED
    CONSTRAINT CDATA #IMPLIED>

<!ELEMENT Interaction (Message)*>
<!ATTLIST Interaction
   %id;
    NAME       CDATA #REQUIRED
    CONTEXT    CDATA #IMPLIED>

<!ELEMENT Message (Label)>
<!ATTLIST Message
   %id;
    NAME        CDATA #IMPLIED
    TYPE        (simple|sync|async|others) "sync"
    SENDER      CDATA #REQUIRED
    RECEIVER    CDATA #REQUIRED
    ACTIVATOR   CDATA #IMPLIED
```

```
    ACTION      CDATA #IMPLIED>
<!ELEMENT Label EMPTY>
<!ATTLIST Label
   %id;
    PREDECESSOR     CDATA #IMPLIED
    GUARD_CONDITION CDATA #IMPLIED
    SEQUENCE_EX     CDATA #IMPLIED
    RETURN          CDATA #IMPLIED
    MESSAGE_NAME    CDATA #IMPLIED
    ARGUMENTS       CDATA #IMPLIED>
```