BOX: Browsing Objects in XML

Christian Nentwich, Wolfgang Emmerich, Anthony Finkelstein and Andrea Zisman
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{C.Nentwich|W.Emmerich|A.Finkelstein|A.Zisman}@cs.ucl.ac.uk

Abstract

The latest Internet markup languages support the representation of structured information and vector graphics. In this paper we describe how these languages can be used to publish software engineering diagrams on the Internet. We do so by describing BOX, a portable, distributed and interoperable approach to browsing UML models with off-the-shelf browser technology. Our approach to browsing UML models leverages XML and related specifications, such as the Document Object Model (DOM), the XML Metadata Interchange (XMI) and a Vector Graphic Markup Language (VML). BOX translates a UML model that is represented in XMI into VML. VML can be directly displayed in Internet browsers, such as Microsoft's Internet Explorer 5. BOX enables software engineers to access, review and browse UML models without the need to purchase licenses of tools that produced the models. BOX has been successfully evaluated in two industrial case studies. The case studies used BOX to make extensive domain and enterprise object models available to a large number of stakeholders over a corporate intranets and the Internet. We discuss why XML and the BOX architecture can be applied to other software engineering notations. We also argue that the approach taken in BOX can be applied to other domains that already started to adopt XML and have a need for graphic representation of XML information. These include browsing gene sequences, chemical molecule structures and conceptual knowledge representations.

1 Introduction

The size and complexity of current software-intensive systems necessitates distributed development. It is important to support collaboration and co-ordination of physically distributed teams of people during system development. Such collaboration often demands that developers access documents that other developers have produced. The wide-spread development and use of the Internet, the World-Wide-Web and related technologies can foster such collaboration. Software engineering documents can be published on web servers so that developers can search for and browse documents with web browsers. Semantically related document parts can be connected by hyperlinks so that they can be traversed in order to help the developer to understand the document better.

The Unified Modeling Language (UML) has been gained widespread attention for the development of software systems. UML is used to visualize, specify, construct and document a software system from multiple perspectives. In a distributed development scenario it is important to allow software engineers and customers to exchange, access, review, query and browse UML models.

Until recently HTML was the language that was used to publish information on the Internet. HTML, however, had no mechanisms to represent graphics directly. To represent UML or any other graphical

software engineering notation in HTML pages, these pages have to refer to images in GIF, TIF or JPEG format. These image formats have the disadvantage that they cannot easily be hyperlinked and can become rather big to download. Moreover, they do not scale well when printed on high-resolution printers. Another approach would be to publish the documents in the format of the CASE tool that was used to support them. Downloading the engineering documents in the storage representation of the CASE tool, however, necessitates the availability of the tool on the developer's workstation – which incurs substantial installation, training and licensing costs in large-scale developments.

These limitations are resolved by the approach we describe in this paper. We discuss a portable, distributed and interoperable mechanism to browsing software engineering documents using standard off-the-shelf browser technology. We exemplify the approach by discussing BOX. BOX supports access to graphic UML models with Internet-scale distribution. The approach is built on top of existing Internet technologies such as eXtensible Markup Language (XML), the Document Object Model (DOM), XML Metadata Interchange (XMI), and Vector Graphic Markup Language (VML). Box translates XMI representations of UML models into VML so that they can be directly viewed using the latest generation of Internet browsers, such as Microsoft's Internet Explorer 5.

The remainder of the paper is structured as follows. In Section 2 we briefly describe XML and the related technologies that were used in our approach. Section 3 contains a description of our approach and its associated architecture. In Section 4 we present issues related to the implementation of our toolkit and examples of two case studies. In Section 3 we evaluate the proposed approach. Section 6 contains a review of related work. Finally, Section 7 outlines future directions and suggests ways of extending the overall approach.

2 XML and Related Technologies

In this section we provide a succinct account of XML (eXtensible Markup Language) and the related technologies which constitute the major part of the infrastructure on which we build. We introduce XML itself and discuss how the OMG XML Metadata Interchange (XMI) specification utilizes XML to define an open interchange format for UML models. We then give a very brief account of the Document Object Model (DOM) that we use to traverse the XMI representation and discuss VML, the output format that we use to generate the graphic representation. It must be appreciated that this is a necessarily brief review of a complex set of technologies. In each case the base technologies are pre-standard, or only just beginning to harden as standards, and hence are subject to continuing change.

2.1 XML

XML (eXtensible Markup Language) [Bray et al., 1998] is a data description language, which has been standardized by the World Wide Web Consortium (W3C) XML is a subset of the Standard Generalized Markup Language (SGML). XML is designed to bring structured information to the Web, it facilitates the definition of markup tags relating to the content of documents and thus delivering both extensibility and the potential for validation. XML 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

```
<Foundation.Core.Class xmi.id = 'S.10028'>
<Foundation.Core.ModelElement.name>SampleClass
</Foundation.Core.ModelElement.name>
<Foundation.Core.ModelElement.visibility xmi.value = 'private'/>
<Foundation.Core.ModelElement.stereotype>
<Foundation.Extension_Mechanisms.Stereotype xmi.idref = 'G.40'/>
</Foundation.Core.ModelElement.stereotype>
</Foundation.Core.Class>
```

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

of distributed data in a manner that is mutually understood. Programmers can build simple parsers to read XML files, 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 Modeling Language) class with a fragment of the associated mark up in XMI [OMG, 1998], using a DTD for mark up of UML (see below).

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, and building a tree-like document structure that can be passed on to the application. The XML application acts upon the structure and processes the data it contains.

2.2 XMI

XMI (XML Metadata Interchange) [OMG, 1998] has been standardized by Object Management Group (OMG) to enable easy interchange of metadata between modeling tools, based on the OMG UML, and metadata repositories, based on OMG MOF, in distributed heterogeneous environments. XMI eases the problem of tool interoperability by providing a flexible information interchange format. It can be used by tool vendors to save and load UML models in a common format, i.e. XMI format, thus allowing teams of developers working with object technology and using different tools to exchange information over the Internet. With XMI data is interchanged as streams or files with a standard format based on XML.

The XMI specification contains a complete specification for syntax and encoding needed to export and import models, with complete DTDs for UML and MOF. Figure 2 shows part of the UML metamodel related to the Core package and its respective XMI DTD specification.

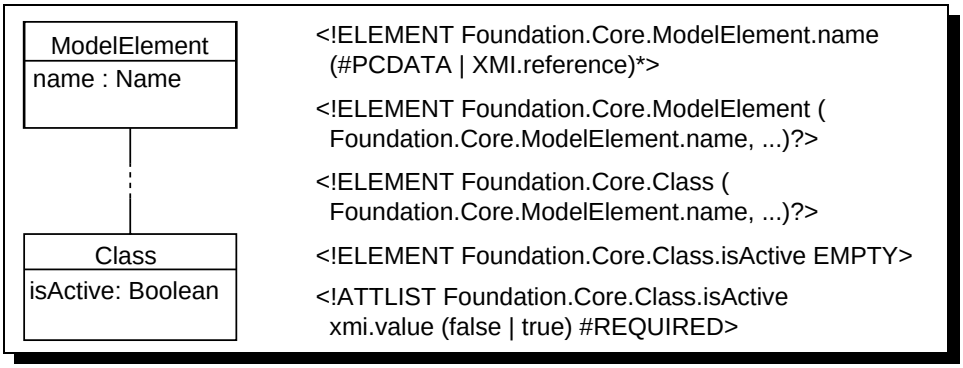

Figure 2: Part of the UML Core package and the respective XMI UML DTD

The UML DTD follows the UML description in the UML Semantics specification [OMG, 1997b]. Each class, attribute and association in the UML metamodel is represented in the XMI DTD as an XML element. The classes, attributes and associations are referred by its fully qualified name; i.e. the package name, followed by the subpackage name, if any, followed by the class name, followed by either the attribute name or the association name, when appropriate.

In XMI, XML representations can also be defined for software engineering notations other than the UML. It is rather straightforward to define a mapping between CDIF and XML in such a way that any CDIF definition can be translated into an equivalent XML document type. We therefore assume that the content of software engineering documents are represented in XML.

2.3 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 modeling of documents using objects, and the model encompasses the structure of a document and its behaviour. The DOM identifies (a) the interfaces used to represent and manipulate a document, (b) the semantics of these interfaces, and (c) the relationships and collaborations among these interfaces. The DOM interfaces are designed in UML and their detailed interface definition is given in a programming-language independent way using the OMG Interface Definition Language (IDL). IDL programming language bindings then define how the DOM is made available in particular programming languages.

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.

```
VML   <v:rect style='top:20; left:50; width:80; height:150'/>
      <v:line from="20 50" to="100 200"/>
SVG   <g:rect x="20" y="50" width="80" height="150"/>
      <g:line x1="20" y1="50" x2="100" y2="200"/>
PGML  <rectangle x="20" y="50" width="80" height="150"/>
      <path>
      <moveto x="20" y="50"/>
      <lineto x="100" y="200"/>
      </path>
```

Figure 3: Drawing a rectangle and line in VML, PGML and SVG

2.4 Vector Graphic Markup Languages

Vector graphic markup languages are still in their infancy. So far, two major proposals have been made: the Vector Markup Language (VML) [Mathews et al., 1998], mainly proposed by Microsoft, and the Precision Graphics Markup Language (PGML) [Al-Shamma et al., 1998], mainly proposed by Adobe. Microsoft have implemented a VML renderer in Internet Explorer 5 and their Office 2000 product also makes use of the format.

Vector graphic markup languages are set to become standard components in future generation browsers. Transmitting graphics in a vectorised form brings with it a host of advantages for web design and web programming. For example, a parsed vector graphics markup file will be incorporated in the DOM tree and can thus be manipulated, the remote user can select different output representations without having to regenerate the whole image, it is possible to search the contents for information, etc. None of these would be very easy to realize using bitmap images. In addition, as vector graphics gain maturity and modification of their content using inline HTML scripting becomes feasible, features like client side zooming can be added. The implications of this for software engineering which makes much use of complex diagrams are very significant.

Following the specification of VML and PGML, a W3C working group was set up to create an open standard for vector graphics markup called Scalable Vector Graphics (SVG) using the available formats as a starting point. The lineup of key industry players supporting SVG is more than impressive, including Microsoft and Adobe themselves, and the language looks set to become a major open standard in the near future.

For our purpose of diagram layout, only the most basic features of any of the available markup languages are required: rectangles, lines and text. We have chosen VML primarily because it is the only language for which there is a working implementation in the form of Internet Explorer 5. As soon as the first SVG implementations appear, it should be easy to adapt our prototype because of the similarity of the languages, as demonstrated in Figure 3.

3 Approach

In this section we describe our approach to converting XML representations of software engineering diagrams into displayable web pages that use the Vector Markup Language (VML). We exemplify the discussion by showing how we translate XMI representations of UML models into VML. We describe

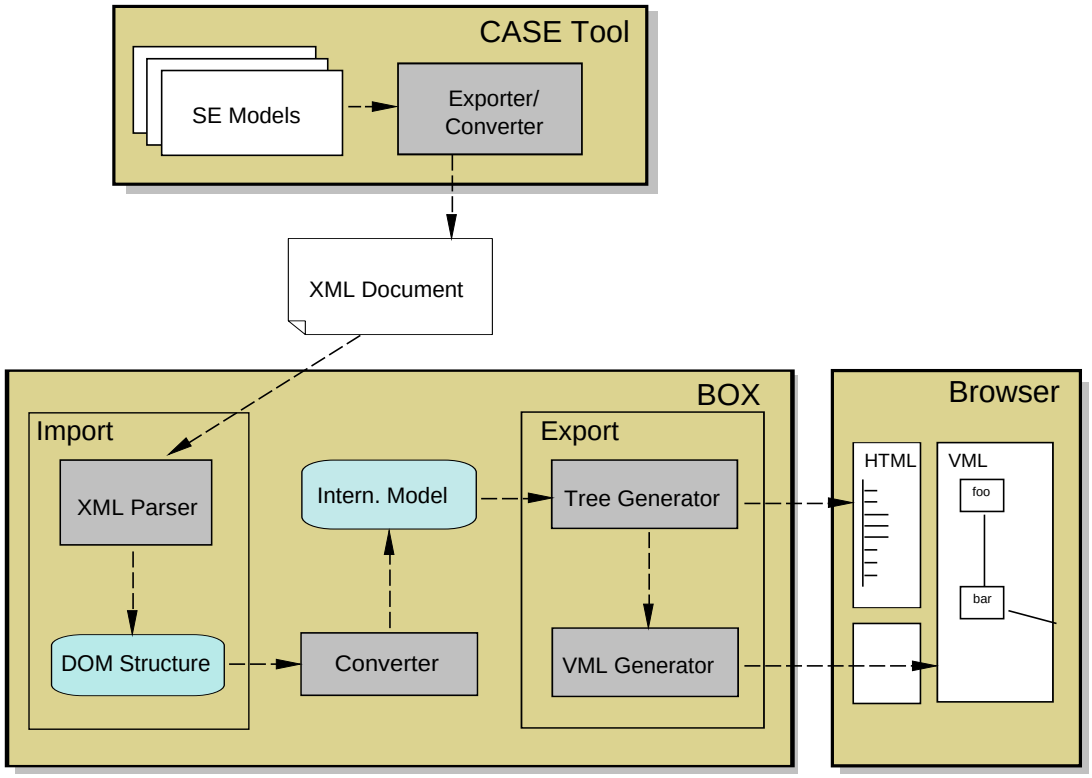

Figure 4: Overview of the prototype architecture

the methods used to visualize UML diagram elements and a user interface to facilitate manipulation and browsing of a UML model. We give a general overview of how to store the geometric information necessary for diagram transfer inside XMI files and outline the architecture of our prototype.

3.1 Architecture

Figure 4 gives a general overview of the architecture of our prototype system. The architecture is applicable for every software engineering notation that has a representation in XML.

The process starts with a CASE tool that allows the creation of a model in a particular software engineering notation. In case of BOX, this notation is the UML and we use Rational Rose [Quatrani, 1998] because of its current market leadership in graphical UML modelling software. After creating the software engineering document, we export it into a standard XML representation. In case of BOX, this export creates an XMI file. In the general case this translation process demands understanding of the proprietary CASE tool format and generating the XML equivalent. For the UML and Rational rose, we were able to utilize an off-the-shelf exporter from Unisys Corporation, which uses the Rose extension capabilities to traverse the internal Rose model and generate the XMI representation. The exporter appends geometric information at the end of the file, inside an XMI extension tag that has been designed to allow arbitrary, user-defined content. This geometric information is also taken from the internal model of Rational Rose.

In the next step, BOX is invoked with the name of the XML input file as a parameter. BOX uses a standard XML parser [IBM, 1999] which conforms to the Simple API for XML (SAX) [Megginson, 1998] to construct a DOM tree-like representation of the XML document in memory.

The DOM structure now contains all elements found in the XML file and can be traversed in order to select relevant elements. The information stored in the tree allows an Internal Model to be built that is suitable for the software engineering notation at hand. In the case of the UML, we have chosen a representation that is in accordance with the UML Semantic Specification [OMG, 1997b]. At this stage, BOX has reconstructed the whole UML model from the XMI file and generated the necessary ViewElement objects to display the model.

The information generated can now be exported in a variety of ways. In our toolkit, we take advantage of the fact that most software engineering notations are hierarchically arranged. Processes in dataflow diagrams are refined by child dataflow diagrams, transitions in Petri nets may be refined by another complete Petri net and so on. In case of the UML, packages contain a set of diagrams, model elements and nested packages. This refinement relationship forms a tree. The first pass of our tool creates an HTML document containing a view of this tree that is displayed in a frame of the browser. The second pass exports a VML representation of each diagram. After the converter has exited, the output directory contains a number of HTML files together with one index file which can now be viewed in a browser.

The back-end of BOX is by no means limited to exporting graphical information. We have also added a package to generate an XML DTD from a UML model. It took about four person hours to write this back-end from scratch so the construction of other applications, such as collecting software engineering metrics should be very straightforward.

3.2 Visualisation of XML

The appendix of the XMI specification [OMG, 1998], provides a DTD that can be used to store UML models in XML. However, both the UML standard and XMI do not clearly specify how to handle geometric information and this section attempts to briefly explain the problems this presents.

The UML describes a metamodel for system modelling. In this metamodel, there are a variety of metaclasses available that allow a software engineer to describe a complete system in different ways. It is important to maintain a distinction between a UML model and the visual representation of that model.

The UML specification is composed of two parts, UML Semantics [OMG, 1997b] and UML Notation [OMG, 1997a]. The former defines the metaclasses (e.g. Class, Association, ...) that can be used to model the system. The latter defines how each of the metaclasses can be displayed graphically. What remains is the question of how to tie a model element in a UML model to its graphical representation.

The UML Semantic specification stops short of specifying how the graphical information of a model is supposed to be handled. In the specification, a meta-class called *ModelElement* is defined as a (meta)-superclass for almost every other meta-class in the model. Each *ModelElement* has a many-to-many mapping to another meta-class called *ViewElement*. However, while there is a complete hierarchy of model elements, the descendants of the ViewElement metaclass are described as *"proper to a graphic editor tool and [are] not specified here"* [OMG, 1997b].

A CASE tool that intends to display UML diagrams therefore needs a complete hierarchy of meta-classes below the ViewElement metaclass to implement view support for all elements, like classes and associations. A diagram is then a collection of ViewElements and descendants of this meta-class. Since view elements have a many to many mapping to their respective model elements, each model element can appear in multiple diagrams.

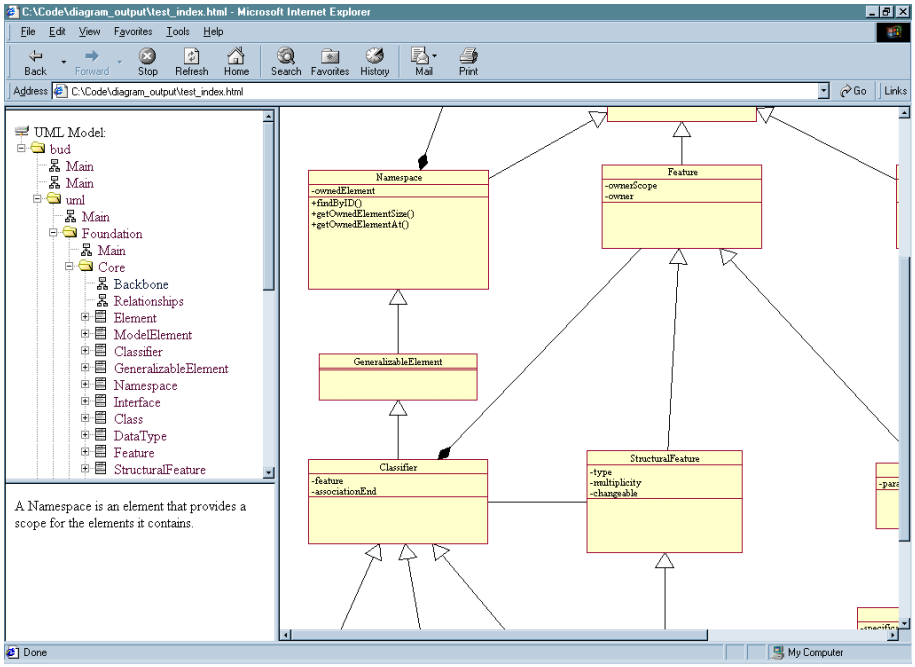

Figure 5: The basic web interface

The XMI standard defines how to build an XML DTD from a metamodel and has as an example a DTD generated from the UML Semantic metamodel. Since this model is not concerned with diagrams, XMI is currently unable to carry geometric information without the use of proprietary extension tags. Even if automatic layout features are available, the diagram information has been lost and no tool will be able to reconstruct the original view of the model without extensive interaction with the original creator of the model.

4 Toolkit

This section briefly describes the main features of our prototype together with some sample screenshots of diagram representations produced by BOX.

The converter is written in Java, using the Java Development Kit (JDK) 1.2. The generated HTML documents use Dynamic HTML (DHTML) and JavaScript to provide reactive documents. In particular, we support expand and collapse operations for the tree of the hierarchical document structure.

Figure 5 presents a screenshot of our interface. Similar to Rational Rose, our interface is divided into three regions. The top left frame contains a tree representation of the software engineering model that can be expanded as needed, the main right frame contains the diagrams themselves and the small frame on the bottom left automatically displays any documentation associated with model elements as soon as the mouse is moved over the respective shape in the diagram.

We use icons to allow rapid distinction of model elements in the tree. For some elements, it is possible to activate hyperlinks to bring up related diagrams, for example it is possible to click on a package in a diagram and display the topmost diagram inside that package.

In the diagram, it is also possible to activate detailed information about the elements of the model.

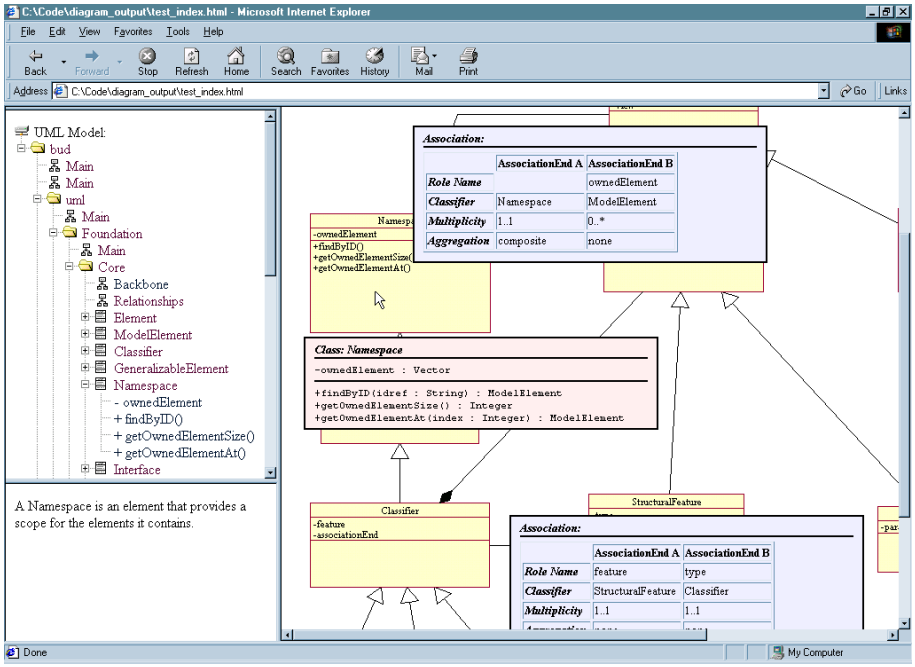

Figure 6: Hiding details in layered popups

This is executed by moving the mouse over a particular element. Figure 6 presents this situation. The idea of hiding detailed information of the model elements is to simplify the presentation of complex diagrams. Any number of popups can be pinned to the screen to allow comparisons.

The two graphical exporters discussed above are just two out of many possible back-ends to the XML importer and model generator front-end. The simplicity of manipulating XML means that it was fairly straightforward to write the front end and it should be similarly straight-forward to add any kind of back-end.

5 Evaluation

This section describes three case studies that we used in order to evaluate Box. We present an overview of the main problem areas and describe alternative routes of design that were considered.

Our initial goal was to display UML diagrams in a browser so that they could be accessed by distributed teams or clients without the need to purchase the license of a CASE tool. Moreover, we wanted to facilitate interoperability between different CASE tools based on the standard defined by the XMI. Initial feedback from demonstrations suggested that there was a real need for such a tool and in general, people seem to be enthusiastic about this application of web technology.

The prototype has been tested in two industrial case studies. The first of these is an enterprise object model of one of the largest European airlines. This model contains 147 classes and 49 diagrams. The second model is a domain object model from Eurocontrol, the European air traffic control authority. The result of our translation of the Eurocontrol model can be viewed using Internet Explorer 5 [Abbot and Watson, 1999]. This model is slightly larger than the airline enterprise model and contains 170 classes but only 18 diagrams. Figure 7 shows a fragment of a diagram from the converted Eurocontrol model. Furthermore, we have also tested Box on an academic example, a UML design for a

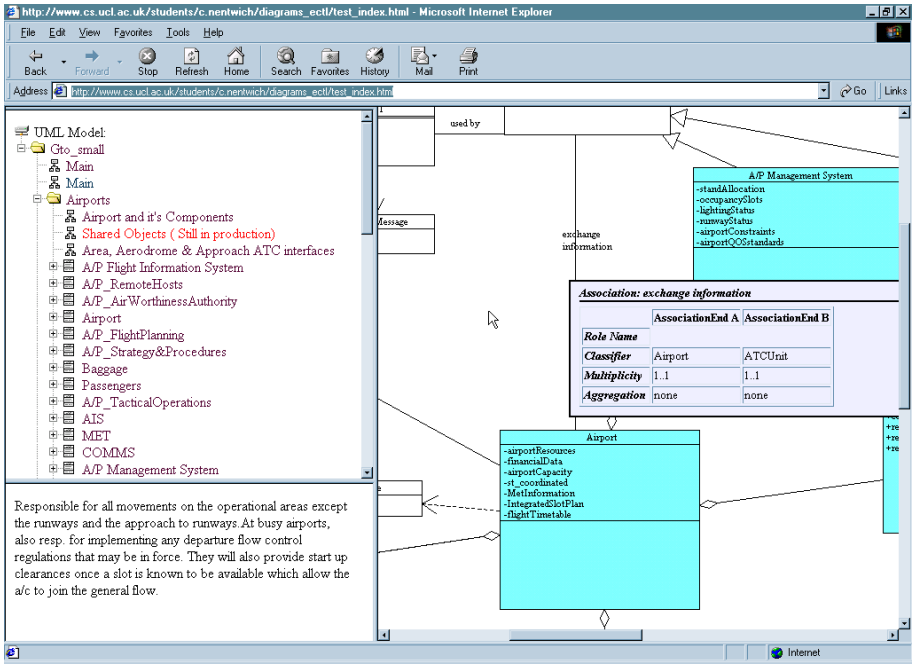

Figure 7: Eurocontrol model fragment

meeting scheduler as described in [Feather et al., 1997]. This is a small-scale model with 12 classes and 6 diagrams.

The testing machine used was a PC with Windows-98, on an Intel Pentium processor running at 400Mhz and 128 Megabytes of RAM. As can be seen from our timing figures, on-the-fly conversion from XMI to VML is not an option for the time being, although our prototype has not been optimised for speed. For no model conversion did the memory utilization exceed the amount of physical memory available. The maximum amount of main memory utilized for both the Eurocontrol and the airline case study was 34 Megabytes and the memory utilization of the meeting scheduler case study was 6 Megabytes. This suggests that our approach scales to industrial-scale projects.

Figure 8 compares the different file sizes that we observed for the Rational Rose input model files, the intermediate XMI representation and the output representation in VML. For both the XMI representation and the VML representations, the sizes grow more or less linearly with the number of model elements. The Rational Rose representation, however, shows an anomaly in that a model with more elements is smaller in size. This can be attributed to the fact that the enterprise object model has fewer elements, which are however more heavily annotated with documentation than the larger Eurocontrol model. We can also note that the VML representation of all models is not only a lot smaller but also grows at a slower rate with more model elements. This strongly suggests that the VML representation is better suited for transport across the Internet or corporate intranets than the XMI representation (with client side translation).

Figure 9 shows the elapsed time that was needed for the conversion of the three models. The limited data seem to suggest that the total conversion time is dominated by the time needed for reading and parsing the XMI file and that it grows linearly with the number of model elements. Note also that the time needed for VML export does not depend on the number of model elements, but on the amount of diagram information stored in the XMI files. The larger Eurocontrol case study has fewer diagrams than the airline model and it is thus natural that the VML export of the airline case study takes longer even though it has fewer model elements overall.

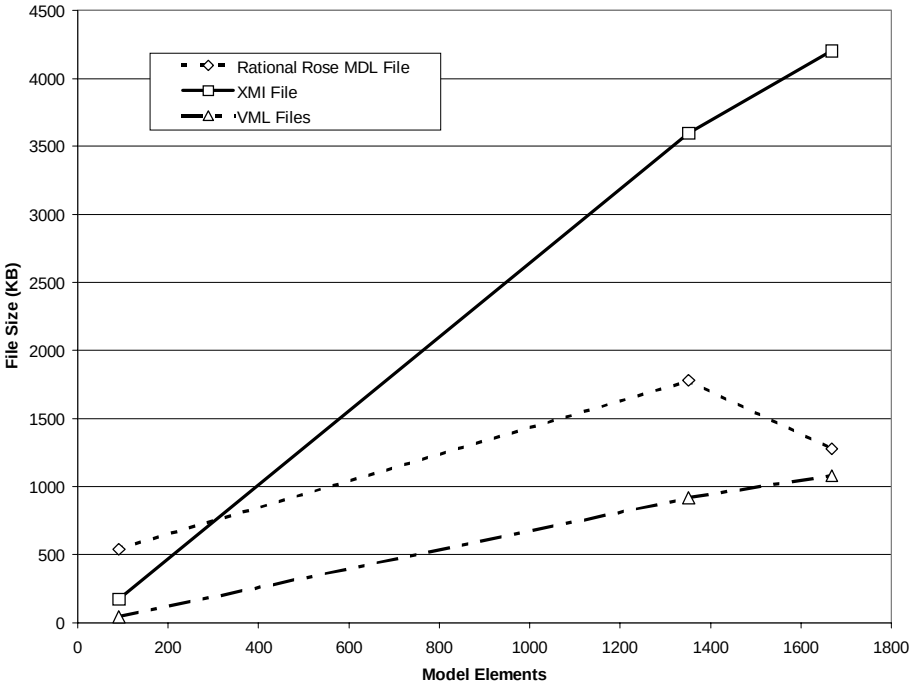

Figure 8: File sizes observed

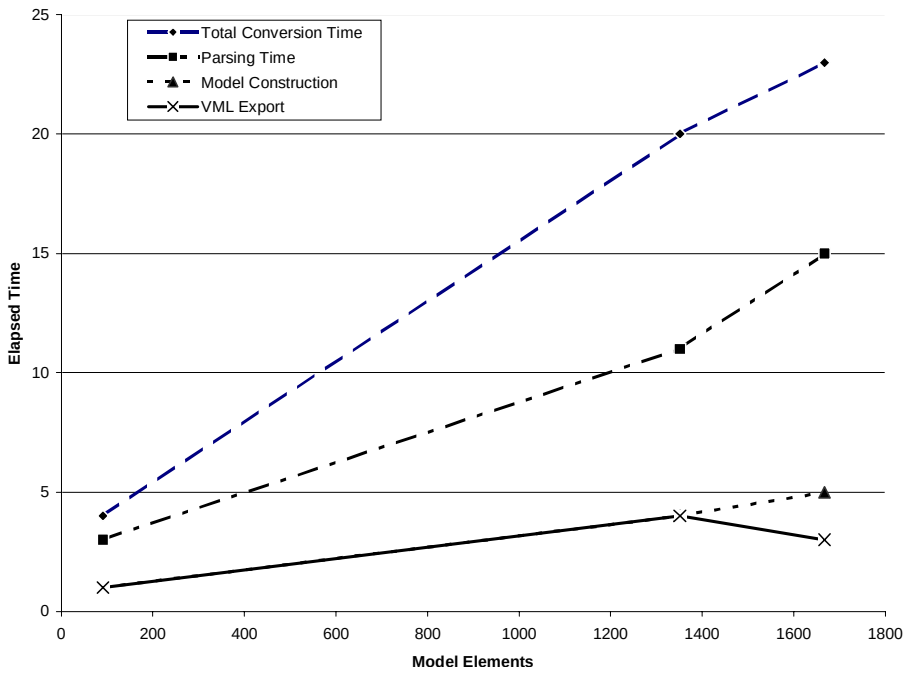

Figure 9: Timings obtained during Case Studies

In all case studies, one of the problems we have encountered was the lack of diagram geometry information being made available by existing tools. It was possible to obtain information about the location and size of classifiers (classes, use cases, actors, etc.) on the screen, but associations between classifiers were generally assumed to stretch between the midpoints of boxes. The problem occurs when users insert extra vertices in the lines representation associations in diagrams, for example, to navigate associations around classes when they would normally penetrate them. These extra vertices are lost in

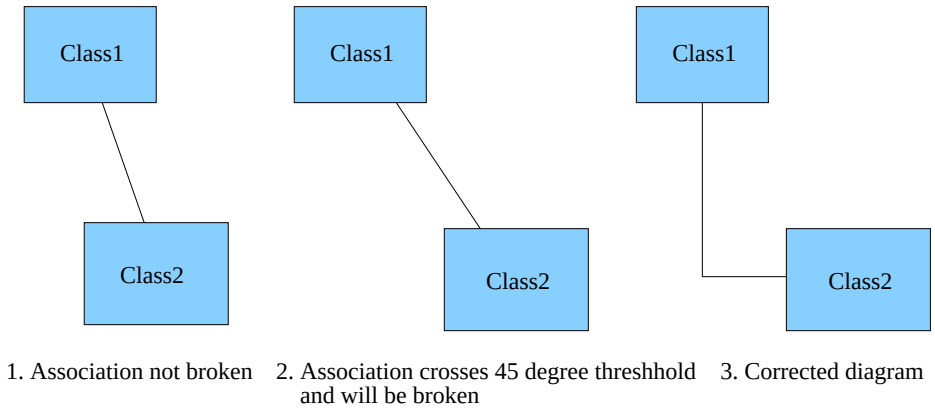

Figure 10: Breaking a line into two orthogonal segments

the XMI generation process. In one of our case studies, the association lines were fairly long and the generated diagrams looked very messy, with lines crossing.

We have so far implemented heuristics for adjusting the routing of lines that give satisfactory results most of the time. When an association is found in the XMI file, it is not possible to know anything about the geometry of the association itself. However, information about where to place the label of the association is known. We therefore propose an algorithm that will check how far the label deviates from the midpoint between two classifiers. If the distance is too big, the association will be "broken" through the label.

With a dependency or association, the angle between the two boxes it connects is measured and if the line has the right slope (around 45 degrees $+-20$), it is broken into two orthogonal lines, as demonstrated in Figure 10. This approach worked well in our case study because it turned out that whenever orthogonal lines are used, the connected classes are quite far apart and if connected with a straight line, the line would be at an angle of 45 degrees ($+-$ the threshold). When a line becomes almost horizontal or vertical, the initial creator of the diagram would draw a straight line so the workaround would not be required.

6 Related Work

In this section we briefly describe current developments in the area of XMI and the web and present the research and technologies that is related to our prototype.

Efforts are currently under way to supplement the XMI standard with a standard for diagram transfer. Two possible solutions that could be envisaged are:

- Use the XMI extension facilities to append geometric data at the end of the model definition. Since XMI allows a unique identifier to be associated with each model element, it is then possible to link every model element to its geometric information. The beta version of the XMI exporter from Unisys Corp. uses this approach: diagram packages are stored in an extension element at the end of the file. Each diagram packages contains multiple diagrams and is linked to the model package it represents. The diagrams themselves are a set of UML Presentation objects and are linked to the ModelElements they represent using the ID facility.
- Store the diagrams in separate files that use a vector markup language. Every shape in the dia-

gram file can then be linked to its corresponding element in the XMI file using the ID facility or more sophisticated technology like XLink [Maler and DeRose, 1998]. This approach is implemented by the open source tool Argo/UML [Robins et al., 1999] and has the advantage that no translation is necessary and that diagrams can now be distributed separately.

The disadvantages are mainly that a model and its diagrams are now stored in several files, which can be inconvenient, and that diagrams do not consist of UML elements but vector graphic information which means that a commitment to a specific vector graphics language has to be made.

A number of CASE tool vendors are currently extending their tools with support for publishing models on the WWW. Rational have recently launched the Internet edition of Rational Rose. It does however not utilize vector graphic languages to create graphical representations of UML models but creates the models as GIFs, which unlike our models are not hyperlinked, not amenable to access via search engines and also bigger to download.

[Ciancarini et al., 1999] introduce "displets" for embedding Z fragments [Spivey, 1992] in web pages. Displets are tiny Java programs that can be used to extend browsers with additional rendering capabilities. The author of a document can then introduce new tags and define which displet to call when a specific tag is encountered. Displets are a lightweight solution because of their relatively small size compared to proprietary plugins or full-blown Java applets. However, the problem of having to transfer a huge XMI file across a network remains, as does the problem of the relatively long layout time.

We have also considered using the eXtensible Stylesheet Language (XSL). This would ensure good use of existing standards and avoid writing Java code, however the following reasons explain why we did not choose XSL:

- Scalability - industrial models are huge. We exported one third of a model from a large investment bank and the resulting XMI file was about thirty megabytes in size. The diagrams generated from such big models are typically only a few hundred kilobytes big. Thus, for client side diagram generation XSL is completely unsuitable with today's bandwidth constraints. As for the use of server side XSL to generate diagrams on demand, that would require hugely expensive equipment to meet the tight time constraints (see also Figure 9).
- One to many mapping - The previous points do not exclude the use of XSL to pre-generate diagrams the way we are doing now using Java. However, XSL was designed to provide a one to one mapping between XML documents with one stylesheet (one source document, one target). With the exporter we are using to generate the XMI file, all the diagrams of a model are specified in one XMI file. Since we did not want to display all diagrams in one page, and specifying one stylesheet per diagram would have limited the amount of diagrams that can be stored in the XMI file, XSL was not suitable for our approach.

7 Future Work

The work presented in this paper is part of a large programme of research to allow consistency checking of distributed documents on the World Wide Web [Ellmer et al., 1999]. We intend to extend our approach to allow semantic and consistency relationships among distributed software engineering documents to be represented as hyperlinks. The consistency relationships can then be traversed to understand the semantic context of documents by following hyperlinks.

Our current implementation of BOX for the UML is restricted to class and use case diagrams due to limitations of current XMI export tools. Future XMI export tools will make those parts of the UML metamodel accessible that cannot currently be reached by our tool. When that information is made available, support for more types of UML diagrams, e.g. sequence diagrams, can be added quite easily.

We expect that in the foreseeable future, our architecture can be used as a base for different applications, such as different notations employed ubiquitously in Software Engineering. As XML dialects for such notations become available, so will the need to present graphical representations of the hard to read XML content. The basic architecture of BOX that parses the XML content, generates an internal model of an instance of a domain-specific metamodel and then compiles different output representations in subsequent export passes is strong candidate to achieve this aim.

Finally, vector graphics markup languages are slowly reaching the stage of maturity. The upcoming SVG language [Ferraiolo et al., 1999] will be a strong standard, supported by key industry players. We will provide a back-end for this language as soon as working implementations appear.

In [Arlow et al., 1999], we describe literate modelling, the application of Knuth's literate programming ideas to earlier phases of software engineering. Literate modelling overcomes the problems of stakeholders, who are not software engineers, in understanding formal UML models. Stakeholders, such as project managers and users, also do not have any training for using a CASE tool and are generally reluctant to incur substantial CASE tool licensing costs. We therefore plan to integrate the graphic export mechanisms of BOX with standard web publishing tools so that literate models can be produced and be published on the web.

Acknowledgements

We would like to thank Gene Mutschler from the Computer Systems Group at Unisys for providing us with a beta copy of the XMI exporter, Jerry Watson from Eurocontrol for the Rose models, Jim Arlow and John Quinn for evaluating BOX with the enterprise object model of a large airline and providing valuable feedback. Moreover we wish to thank Morten Wang, who implemented the Javascript tree menu.

8 References

Abbot, P. and Watson, J. (1999). Gate to Gate Object Model. http://www.cs.ucl.ac.uk/research/box.

Al-Shamma, N., Ayers, R., Cohn, R., Ferraiolo, J., Newell, M., de Bry, R. K., McCluskey, K., and Evans, J. (1998). Precision Graphics Markup Language. Technical Report http://www.w3.org/TR/1998/NOTE-PGML-19980410.html, World Wide Web Consortium.

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. W3C Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium.

Arlow, J., Emmerich, W., and Quinn, J. (1999). Literate Modelling – Capturing Business Knowledge with the UML. In Bezivin, J. and Muller, P. A., editors, *Proc. <<UML>> '98, Mulhouse, France*, volume 1618 of *Lecture Notes in Computer Science*, pages 165–172. Springer.

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.

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

Ellmer, E., Emmerich, W., Finkelstein, A., Smolko, D., and Zisman, A. (1999). Consistency Management of Distributed Documents using XML and Related Technologies. Technical report, University College London, Dept. of Computer Science.

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

Ferraiolo, J., .Bowler, J., Capsimalis, M., Cohn, R., Donoho, A., Duce, D., Evans, J., Furman, S., Graffagnino, P., Henderson, L., Hester, A., Hopgood, B., Lawrence, K., Lilley, C., Mansfield, P., Nguyen, K. M. T., Sandal, T., Santangeli, P., Sheikh, H., State, G., Stevahn, R., and Zhou, S. (1999). Scalable Vector Graphics. Working Draft http://www.w3.org/1999/08/WD-SVG-19990812, World Wide Web Consortium.

IBM (1999). IBM XML Parser for Java. http://alphaworks.ibm.com/tech/xml4j.

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.

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.

Megginson, D. (1998). Simple API for XML.

OMG (1997a). *UML Notation Guide*. Object Management Group, 492 Old Connecticut Path, Framingham, Mass., ad/97-08-05 edition.

OMG (1997b). *UML Semantics*. Object Management Group, 492 Old Connecticut Path, Framingham, Mass., ad/97-08-04 edition.

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.

Quatrani, T. (1998). *Visual Modeling with Rational Rose and UML*. Addison Wesley.

Robins, J., Redmiles, D., and Hilbert, D. (1999). Argo/UML. http://www.ics.uci.edu/pub/arch/uml/.

Spivey, J. M. (1992). *The Z Notation - A Reference Manual*. Prentice Hall, 2 edition.