Managing the software design documents with XML

Junichi Suzuki
Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama, 223-8522, Japan
+81-45-563-3925
suzuki@yy.cs.keio.ac.jp

Yoshikazu Yamamoto
Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama, 223-8522, Japan
+81-45-563-3925
yama@cs.keio.ac.jp

ABSTRACT

It is hard to manage the software design documents within a distributed development team. The issues include the format, distribution and evolution of data. This paper mainly focuses on the issues of the format and distribution, and addresses how we can manage the software design documents for the distributed software development in the standard based way.

In the software engineering community, Unified Modeling Language (UML) has been widely accepted as an object-oriented software analysis/design methodology, since it provides most of the concepts and notations that are essential for documenting object oriented models. UML, however, does not have an explicit format for interchanging its models intentionally. This paper addresses this lack and proposes UXF (UML eXchange Format), which is an exchange format for UML models, based on XML (Extensible Markup Language). It is a format powerful enough to express, publish, access and exchange UML models and a natural extension from the existing Internet environment. It serves as a communication vehicle for developers, and as a well-structured data format for development tools.

We demonstrate some proof-of-concept applications that show the merits of UXF. We are especially interested in a distributed model management system that manages the software design documents over the Internet with UXF. This system leverages the team development, reuse of design documents and tool interoperability by publishing a set of CORBA interfaces. Our work shows an important step in sharing and exchanging software design documents, and indicates the future direction of the interoperable software development tools.

Keywords

Software model interchange, CASE data interchange, UML, XML

1. Introduction

It is hard to share and manage the software design models and their documents within a distributed development team. The issues to manage them include:

- Describing and interchanging software model information, which involves the exchange format.
- Evolution of software model information, which involves the data consistency and storage facility.
- Defining a set of published interfaces for the network communications, which involves the remote manipulation of the model information.

In most cases, software models are dynamically changed during the analysis/design, revision and maintenance phases, and also the software tools used by a development team employ their own proprietary formats to describe the software model information. Thus, it is required to interchange the model information between development tools throughout the lifecycle of software development.

In addition, the Internet is an emerging infrastructure to distribute and share the software model information, because it is an effective and economical for making information available to the widely separated group of individuals. Within the Internet/Intranet environment, especially the Web environment, we can represent, encode and ultimately communicate the software modeling insights and understandings with each other.

Coupled with the above problems and requirements, a framework is needed which allows the software design documents described with an application neutral (standard based) format to be distributed and shared over the Internet.

In the software engineering community, the Unified Modeling Language (UML) [2-9] has been widely accepted as an object oriented software analysis/design methodology, since it provides most of the concepts and notations that are essential for documenting object oriented models. Thus, we use UML as a tool to specify the software structure and dynamics in this paper. UML, however, does not offer an explicit format to describe and exchange analysis/design models.

This paper describes how our work aims to make UML models interchangeable over the Internet and/or between development tools. To provide a standard-based means, we propose a stream-based exchange format called UXF (UML eXchange Format), which is based on XML (eXtensible Markup Language) [14]. We

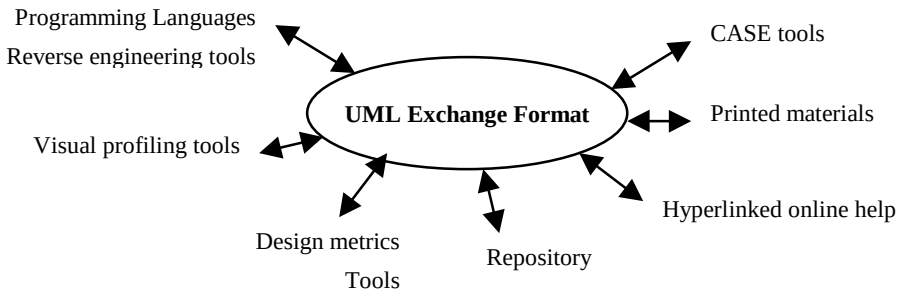

Figure 1: UXF allows the seamless exchange of UML models between development tools.

consider the use of XML as a mechanism for encoding and exchanging the structured data defined with UML. We outline the rationale and fundamentals of UXF, and then discuss how we can use it to express, publish, share and exchange UML models.

The remainder of this paper is organized as follows. Section 2 overviews UML and introduces the current limitations in encoding and exchanging the complex structured information with HTML (HyperText Markup Language), which is a traditional markup language in the Internet. Then, the motivation and merits of UXF is presented. Section 3 outlines the comparison with related work. Section 4 defines the scope and syntax of UXF. Section 5 provides a solution to interchange UML models on the Internet and between various development tools. We conclude with a note on the future work, in Section 6 and 7.

2. UML, XML and UXF

2.1 Unified Modeling Language (UML)

UML is the union of the previous leading object modeling methodologies; Booch [10], OMT [11] and OOSE [12]. When UML was published in late 1996, it quickly gained momentum and became the de-facto standard for object-oriented modeling. UML has been also submitted to the Object Management Group (OMG) to become a public standard, and included additional constructs that ancestors did not address, such as the extension for business modeling [7], Object Constraint Language (OCL) [8] and Object Analysis & Design CORBAfacility Interface Definition [9]. It is the state of the art convergence of practices in the academic and industrial community and expected that the most developers will eventually choose UML for their modeling work.

UML defines the following diagrams for the object modeling, according to various perspectives to a target problem domain:

- Structural diagrams:
  - Class diagram
  - Object diagram
- Behavioral diagrams:
  - Object diagram
  - Use case diagram
  - Sequence diagram
  - Collaboration diagram
  - State transition diagram
  - Activity diagram
- Implementation diagrams:
  - Activity diagram
  - Component diagram
  - Deployment diagram

Using these diagrams with the fine level of abstraction, complex systems can be modeled through a small set of nearly independent diagrams. UML provides two aspects for constructs in the above diagrams:

- Semantics
  The UML metamodel defines the abstract syntax and semantics of object modeling concepts.
- Notations
  UML defines graphical notations for the visual representation of its model elements.

While UML defines the above coherent constructs and its interchangeable semantics, it does not intentionally provide the explicit format to exchange the model information. The ability to exchange models is quite important, because the network environment such as the Internet grows exponentially and it is likely that a development team resides in separate places. In addition, such an ability facilitates the application interconnectivity so that the model information can be exchanged between various tools such as CASE (Computer Aided Software Engineering) tools, diagram editors, reverse engineering tools and design metrics tools. As such, the application-neutral format expands the ability to encode, exchange and reuse the model information.

2.2 Limitation of HTML

Since the World Wide Web is becoming the ubiquitous environment for viewing information, HTML is now a major document format. It is also used for software documentation. Examples of such tools include javadoc included in Java Development Kit (JDK) [13], which is a translator from the comments in source code of Java into the specification documents written in HTML. While such a tool is valuable and helpful for everyday development work, some important information within software models is unfortunately thrown away in the process of producing HTML documents, due to its fixed tag set. Contents in such HTML documents are meaningful only for human usage, and its semantics cannot be unambiguously recognized by software tools. In other words, HTML documents generated by documentation tools cannot be reused in other applications other than HTML browsers. In this situation, if a more semantics-rich markup language is used, and development tools support it, we can interchange UML model information without losing its semantics.

Also, as described above, it is likely that the members in a software development team, including requirement analysts, system architects, designers and programmers, are working in

separated places and relying on electronic communication. Currently, the predominant means to distribute UML diagrams on the Web is the image-based method, in which GIF or JPEG images representing UML diagrams are included within a HTML text stream. However, it is difficult and expensive (i.e. time consuming) to author, read and maintain these images. It is also inadequate since model information is hidden within images and can not be available for other development tools (e.g. for searching and drawing). Another problem with encoding the model information as images is that it requires more network bandwidth. With markup-based encoding, more of the rendering process can be moved to the client machine. Markup language representation of model information is typically smaller and more compressible than an image of the model.

Coupled with the above problems, the most important factor in exchanging the UML models between software programs is that the semantics within the models should be explicitly described, keeping its semantics. For this purpose, Extensible Markup Language (XML) is a reasonable and practical candidate for a vehicle to interchange UML models.

2.3 XML (eXtensible Markup Language)

XML is a data description language standardized by the World Wide Web Consortium (W3C) [14]. While HTML is defined by SGML (Standard Generalized Markup Language: ISO 8879), XML is a sophisticated subset of SGML, and designed to describe document data using arbitrary tags. One of the goals of XML is to be suitable for use on the Web; thus to provide a general mechanism for extending HTML. As its name implies, extensibility is a key feature of XML; users or applications are free to declare and use their own tags and attributes. Therefore, XML ensures that both the logical structure and content of semantics-rich information is retained. XML is widely accepted in the Web community now, and the current applications of XML includes MathML (Mathematical Markup Language) [15] to describe mathematical notation, CDF (Channel Data Format) for push technologies, OFX (Open Financial Exchange) [16] to describe financial transactions and OSD (Open Software Distribution) for the software distribution on the Web.

XML emphasizes description of information structure and content as distinct from its presentation. The data structure and its syntax are defined in a DTD (Document Type Definition) specification, which is a derivative from SGML and defines a series of tags and their constraints. In contrast to information structure, the presentation issues are addressed by XSL (XML Style Language) [17], which is also a W3C's emerging standard for expressing how XML-based data should be rendered. XSL is based on DSSSL (Document Style Semantics and Specification Language ISO/IEC 10179) and interoperable with CSS (Cascading Style Sheet), which was originally a style definition language specific to HTML. In addition to XML and XSL, Xpointer [18] and Xlink [19] are also in the process of standardization, which is a specification to define anchors and links within XML documents. As such, XML has great potential as an exchange format for many kinds of structured data, and increases the productivity to author, maintain and view this data, together with the style sheet and linking mechanisms. XML improves on the features that HTML has provided.

2.4 UXF (UML eXchange Format)

Faced with the problems described in Section 2.2 and taking advantage of the emerging data description language called XML, we propose an exchange format of UML models called UXF (UML eXchange Format). UXF is an application of XML and is designed to be flexible enough to encode and exchange any UML constructs. UXF facilitates:

- **Intercommunications between software developers:**
  UXF is a powerful transfer vehicle for UML models between software developers. It simplifies the circulation in UML models with each other, using a human-readable and intuitive format.
- **Interconnectivity between development tools:**
  UXF is a well-structured and portable (i.e. application neutral) format between various development tools. Once encoded with UXF, the information of UML models can be reusable for a wide range of usage with different strengths of different tools (Figure 1).
- **Natural extension from existing Web environments:**
  UXF is a natural and transparent extension from the existing Web environment. Thus, it allows to edit, publish, access and exchange the UXF data as easily as is currently possible with HTML. In addition, most of the existing applications around the Web can be used for handling UXF encoded information with relatively minor modifications.

In order to author and view UML models encoded with UXF, existing markup languages could be converted to UXF, and most development tools such as CASE tools, documentation tools, visual profiling tools and document repositories, can be modified so that they recognize UXF. In the current situation where many XML-aware applications exist, it is relatively easy to extend existing tools. Also, UML-related technical materials formatted in UXF can be handled by every Web application which manipulates HTML as well as Web browsers and Web servers, in the near future. As a result, UXF allows the borderless uses of UML models among development tools (Figure 1), and provides the tight integration with Web environments. This feature increases our productivity of UML modeling.

The potential use cases of UXF covers a broad spectrum. Developers including analysts, designers and engineers can communicate their insights, understanding or intention on their UML models, by interchanging UXF formatted files. Also, the ability to maintain technical information during software lifecycle is vital to development teams for archival purpose, because every team typically has large volumes of materials. Engineers use these materials in their work to refer and revise the current information, record results of experiments or historical logs. For such uses, well-structured UXF provides a standard way of sharing information, in which we can easily read, process and generate UML models using commonly available tools (see Section 4.2).

In addition, UXF ensures a variety of possibilities of its output representations; how UXF data should be rendered or viewed. Since UXF can apply arbitrary XSL style sheets, it can be converted into materials in a wide range of media like RTF (Rich Text Format), HTML, LaTeX, PDF (Portable Document Format) (see Section 5.3). Moreover, UXF data can embed hypermedia links with the linking mechanisms of XPointer and XLink. This allows us to link UML constructs each other. As such, developers

can involve in technical materials at all level from electronic versions, printed documents to interactive versions.

3. Related work

The well-known and mature format for exchanging the software modeling information is CDIF (CASE Data Interchange Format) [20]. CDIF is a generic mechanism and format to interchange the software models between CASE tools, and a family of standards defined by the Electronic Industries Association (EIA) and International Standard Organization (ISO). CDIF defines a meta-metamodel, a tool interchange format, and a series of subject areas:

- CDIF Framework for Modeling and Extensibility
- CDIF Integrated Metamodel
    - Foundation Subject Area
    - Common Subject Area
    - Data Modeling Subject Area
    - Data Flow Model Subject Area
    - Data Definition Subject Area
    - State/Event Model Subject Area
    - Presentation Location and Connectivity Subject Area
- CDIF Transfer Format
    - General Rules for Syntaxes and Encodings
    - SYNTAX.1
    - ENCODING.1

CDIF separates the semantics and syntax from the encoding, and thus provides flexibility in the representation and transfer mechanism. SYNTAX.1 and ENCODING.1 defines the means that allows for a tool-independent exchange of models. CDIF has provided the mapping to UML [21], by using the Foundation Subject Area and CDIF Transfer Format, and by defining the UML subject area that provides the definitions of metamodel entities and their relationships in UML. The UML Subject Area is dependent on the CDIF Foundation Subject Area.

UXF is a UML-specific exchange format and an alternative vehicle to transfer UML models. Since it is a straightforward extension from and transparent to the Web-based distributed environment, it can be easy-to-understand and use for the huge amount of people that is familiar with HTML or SGML. We believe UXF is a practical approach for encoding and exchanging

UML Package	UML Model Element	UXF Representation
Core	Association	<Association>
	AssociationEnd	<AssocRole>, <PeerAssocRole>
	Attribute	<Attribute>
	Class	<Class>
	Dependency	<Dependency>
	Generalization	<Generalization>
	Interface	<Interface>
	Operation	<Operation>
	Parameter	<Parameter>
Auxiliary Elements	Refinement	<Refinement>
Extension Mechanisms	TaggedValue	<TaggedValue>
Common Behavior	Exception	<Exception>
	Action	<Action>
	ActionSequence	<ActionSequence>
	Instance	<Instance>
Model Management	Model	<Model>
	Package	<Package>
Collaborations	Collaboration	<collaboration>
	Interaction	<Interaction>
	Message	<Message>
StateMachines	CompositeState	<CompositeState>
	Event	<Event>
	Guard	<Guard>
	State	<State>
	Transition	<Transition>
	PseudoState	<PseudoState>

Table 1: Comparision of UML model elements and UXF elements

UML models over the Internet.

4. UXF design principles

In terms of exchanging model information between development tools, there can be two types of information that should be exchanged [21]:

- Model-related information
- View-related information

While model-related information is a series of building blocks that are used to represent a given problem domain, e.g. classes, attributes and associations, view-related information is composed of the way in which the model is rendered, e.g. the shapes and position of graphical objects. This paper concentrates on exchanging the model-related information. The interchange of the view-related information is future work, but it would be easy to describe the view-related information with XSL (see also Section 5.3).

4.1 UXF DTDs

As described above, the UXF specification actually consists of a series of XML DTDs. It provides the mapping of UML model information into document tags in the DTDs. UXF captures the constructs (i.e. model elements) in a UML metamodel, and defines each construct as a tag (i.e. a document element) straightforwardly. The attributes of each UML construct are mapped into attributes of the corresponding UXF tag.

We have specified UXF DTDs for the following three UML diagrams. These diagrams are fundamental for the analysis and design of problem domains.

- Class diagram
  A class diagram shows the static structure of classes and relationships between them. This diagram also defines the foundation for other diagrams that specify different aspects of the problem domain.
- Collaboration diagram
  A collaboration diagram shows an interaction organized around the objects in the interaction and their links to each other.
- Statechart diagrams
  A statechart diagram shows the sequences of states that an object goes through during its life in response to received stimuli, together with its responses and actions.

Table 1 depicts the comparison of UML model elements and UXF tags, and Appendices shows the complete UXF DTDs for the above diagrams, which include all tags, entities and attributes of XML. Current UXF largely extends its format described in [1] and supports most elements in the UML's Core package, Collaboration package, State Machines package and some elements in other packages (see Table 1).

Using UXF, most concepts and constructs in UML can be mapped to the stream-based exchange format seamlessly. Sample markup (encoding) examples can be found at [22]. Note that constructs described with UXF are not shared between different diagrams for the simplicity. Sharing the model information between different diagrams consistently is considered as the responsibility of UXF-aware applications.

4.2 Processing UXF documents

This section outlines how a UXF documents might be created, processed and displayed. In every phase, we can reuse various existing XML or SGML tools.

4.2.1 Authoring

UXF data can be created with any text editor because it is a text-based format and human-readable format. In practice, however, it is not primarily intended for manual use by developers. It is likely that the manual generation of UXF is verbose and error-prone. Instead, it is anticipated that they use CASE tools, conversion tools, UXF editors and other specialized software to recognize and generate UXF. It is practical in order to increase productivity that we use any editing tool that helps user's input or conversion tools, described below. In our work, a SGML/XML major mode for Emacs called psgml has been used when UXF data is manually written from scratch (left of Figure 2).

4.2.2 Conversion

As described above, the authoring process sometimes involves data conversion. It makes the authoring work simple and productive. In the context of UXF, it can be categorized into two schemes; converting legacy documents, program source code or data representation in a development tool (e.g. CASE tool) to UXF, and converting UXF to other formats for printed materials or development tools. UXF allows such conversion programs to be written easily. In our work, we have prepared a conversion tool from source code written in Java into UXF, and a translator that generates the proprietary data representation in a CASE tool from UXF (see Section 5.1 and 5.2).

4.2.3 Parsing

Parsing is the process to analyze and validate the syntax of UXF documents. XML allows for two kinds of data; valid and well-formed. Validity requires that a document contains a proper DTD and obeys its constraints. Well-formness is a less strict criteria and requires that a document just obeys the syntax of XML. UXF requires a validating parser that confirms to be valid in creating UXF documents, and a non-validating parser that confirms just to be well-formed in browsing or delivering the document. We are using a validating parser called nsgmls in creating UXF document, and non-validating parser called Lark in distributing the document.

4.2.4 Distributing

UXF has been designed to distribute UML models precisely over the network environment. It can be used in existing SGML systems that retrieve UXF components and assemble them for the output on the fly. Also, it can be used within the existing Web environment so that a Web browser downloads UXF components and displays them using a stylesheet or Java applets. In our work, UXF has been transferred within a CORBA based distributed environment and a HTTP based Web environment (see Section 5.3).

4.2.5 Rendering and Browsing

Rendering and browsing involves the delivery of stylesheets or any specialized software for display such as Java applets. In our work, UXF is intended to use stylesheets based on XSL or Java classes associated with each UXF element. Also, UML model information can be browsed with existing XML browsers like

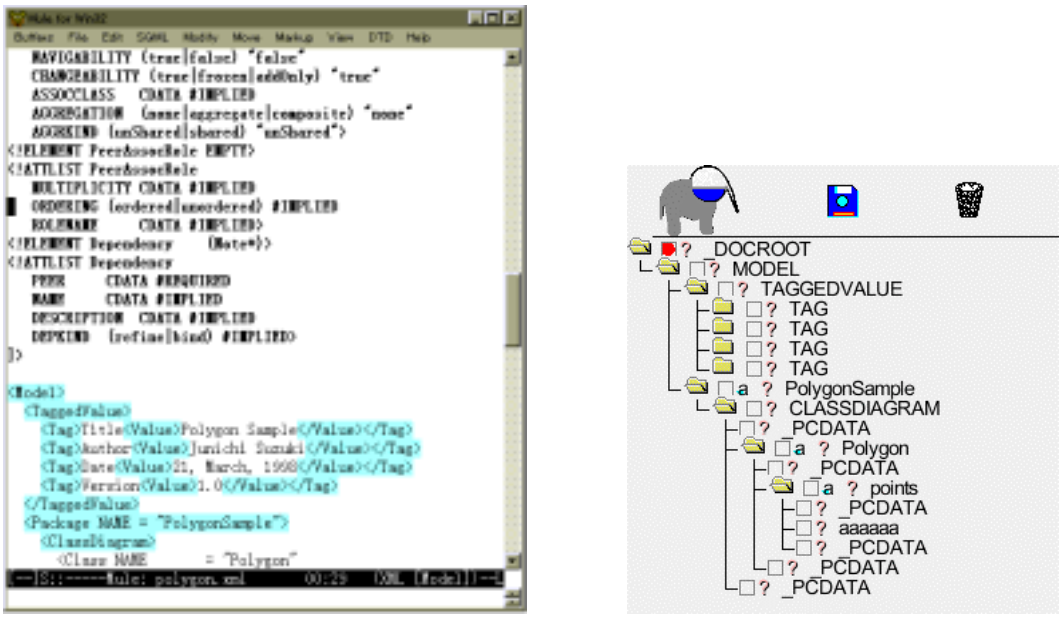

Figure 2: Editing UXF formatted data with an existing XML editor (left), and browsing the hierarchical structure of UXF elements with an existing XML browser (right).

```
<Model>
  <ClassDiagram>
    ......
    <Class NAME = "ReservationAgent">
      <Operation NAME   = "reserve"
                 RETURN = "boolean">
        <Parameter NAME = "info"
                   TYPE = "Reservation"/>
      </Operation>
    </Class>
    ......
  </ClassDiagram>
</Model>
```

```
(object Petal version 41
                charSet 128)
(object Design "Logical View"
  ......
root_category(object Class_Category "Logical View"
  ......
  logical_models(list unit_reference_list
    (object Class "ReservationAgent"
       operations(list Operations
         (object Operation "reserve"
            parameters(list Parameters
              (object Parameter "info"
                 type "Reservation"))
            result "boolean")))
  ......
)
```

Figure 3: Example of mapping of a UXF description (left) into an importable file of Rational Rose (right).

Jumbo (right side of Figure 2). Section 5.3 touches on the use of XSL styleshhets for UXF documents.

5. Applications

This section presents some efforts to distribute and manage design UXF documents. Some UXF-aware tools have been developed as the following sections describes. At present, we are especially interested in the distributed software development with UXF over the Internet.

5.1 Source code documentation tools

In general, source code documentation tools is a tool that imports the source code of a programming language and generates documents on the program itself, along with any specialized format.

We have prepared such a documentation tool that parses source code written in Java and generates UXF formatted documents. This tool is developed by creating a class named `UxfDocumentationGenerator` extending the class `DocumentationGenerator` included in JDK. It translates the constructs in Java to the corresponding representation of UXF described in Table 1. Note that this tool cannot generate the UXF representation of associations because there is not explicit differences between attributes and associations in Java.

Using such a documentation tool, the model information can be obtained directly from source code, and reusable for other applications including CASE tools and repositories.

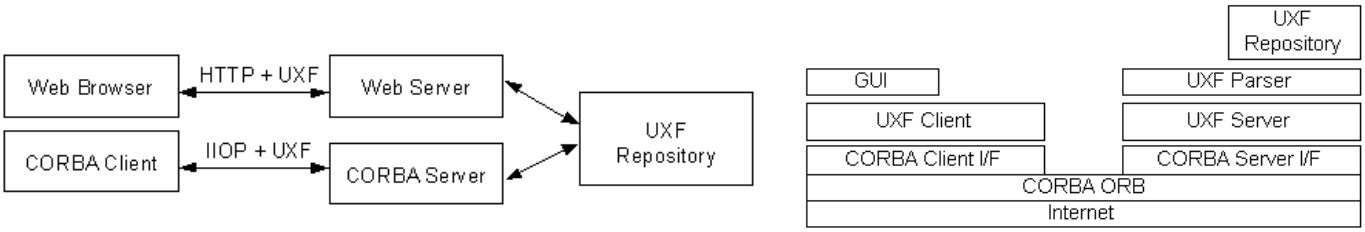

Figure 4: Deployment architecture of our prototype system that allows to share UML models over the Internet (left), and Layered architecture for IIOP based system (right).

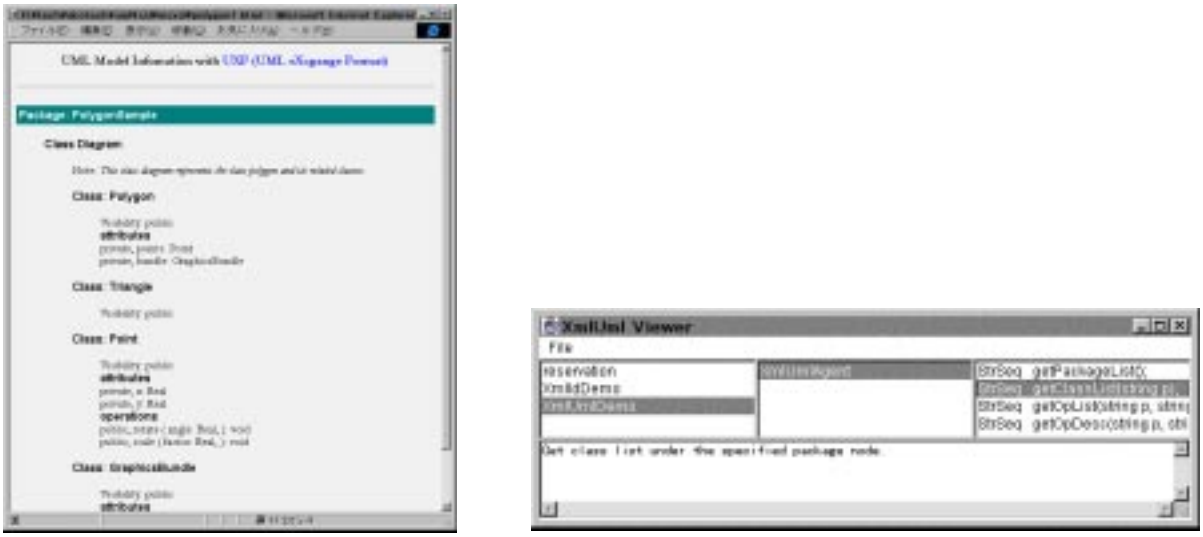

Figure 5: Sample screenshots of a Web browser that displays a UXF document together with a XSL stylesheet and a GUI profiling tool that communicates with CORBA servers.

5.2 Translator between UXF and CASE tools

It is also important that UXF descriptions can be translated into any other (even proprietary) formats used by development tools. Translators from UXF descriptions to the importable formats of CASE tools would be highly required, because CASE tools are quite helpful in the object-oriented development.

We have developed a translator that generates the importable files of Rational Rose™. Currently, this translator generates files only for class diagrams. Figure 3 depicts a simple example of the translation from UXF description into an importable file (*.mdl file) of Rational Rose.

5.3 Distributed model management system

The last application is a system that manages the UML design information within a distributed environment based on the Internet. The Internet-oriented centralized management of design information leverage:

- Team development
  allows designers and programmers to continue their work concurrently at the physically separated places.
- Reuse of design documents
  allows developers to change (i.e. revise) them at single point, and increases productivity of the above team development.
- Interoperability between development tools
  combines various development tools with a single infrastructure, and allows developers to easily change their tools throughout the development lifecycle.

We have developed such a system on top of the existing Web environment and an Java-based ORB (Object Request Broker) that is compliant with CORBA (Common Object Request Broker Architecture). It is based on the three-tier deployment architecture (Figure 4), and provides two kinds of accesses to UXF documents; via HTTP and IIOP (Internet Inter-ORB Protocol), which is a TCP/IP based standard protocol of CORBA. The communications via IIOP is achieved through the CORBA standard IDL (Interface Definition Language) interfaces (see the right of Figure 4).

The HTTP access aims to allow client applications including Web browsers to refer the UXF documents that are stored in Web servers or any backend databases. The left of Figure 5 is a sample screenshot of a Web browser that displays a UXF document together with a corresponding XSL stylesheet. If other stylesheets are prepared, different outputs suited to the specific purpose can be displayed.

The IIOP access aims to allow developers at separated places to consistently register, refer, process and change UXF documents. A server application parses the UXF documents at the system's start-up time or on the fly, and creates an in-memory structure of

```
typedef sequence<string> StrSequence;
interface UxfHandler {
    StrSequence getPackageList();
    StrSequence getClassList( in string pkgName );
    StrSequence getOperationList( in string pkgName, in string className );
    string getPackageDescription();
    string getClassDescription( in string pkgName );
    string getOperationDescription( in string pkgName, in string className );
    boolean findConstruct( in string name );
}
```

Figure 6: Selected IDL interfaces of a CORBA server for accessing UXF documents

these documents; tree structures of parsed UXF elements. Client applications include simple command-line tools, GUI profiling tools and development environments (see also Figure 1). The right side of Figure 5 is a sample screenshot of a client-side GUI profiling tool that is similar to the system browser in Smalltalk. In this tools, the left-side pane shows the list of UML packages, the central pane lists classes in the package, the right one is the list of operations (i.e. methods) of the class selected in the central pane, and the bottom pane shows the comments (Note and TaggedValue in the UML term) for each package, class or method. This tool accesses a CORBA server to obtain the necessary data to display, according to the user's mouse manipulation.

Extending this mechanism, we can maintain large document collections stored in a server-side repository, which ensures the consistency of information and provides the capabilities like searching, indexing or sorting. This system shows UXF is valuable as a well-structured data format.

The selected IDL interfaces of a CORBA server are shown in Figure 6. In general, APIs that handle XML documents via middleware are categorized into three group [23]:

- Source document APIs
  manage the XML document instances directly.
- Element APIs
  manage the parsed elements in the document instances. The navigation of the XML document and selection, control and update of the element are achieved through this API.
- Custom APIs
  depend on certain applications with related DTDs, and provides application specific interfaces.

Our current interfaces listed in Figure 6 are custom APIs, which is specific to UXF class diagram documents. We are now extending them and defining the element APIs.

The applications given in Section 5 show straightforward and reasonable ways to share and exchange the UML design documents between various tools or over the distributed environment, though they are simple. We are working on the refinement of these applications and the additional ones, to demonstrate the value of UXF and improve it.

6. Future work

As for UXF format itself, we are refining the existing DTDs and developing ones for all the UML diagrams.

As for UXF-aware tools, a UXF translator from/to Python programming language is currently developed. With multiple source code generation tools, the model information implemented in various programming languages can be fully shared with multiple development tools, through a single exchange format. In addition, a UXF-aware diagram editing/drawing tool is planned.

As for the Internet based model management system described in Section 5.3, we are enhancing it with various features. For example, we are introducing a mechanism of the revision control with two tags of XML; <![INCLUDE[…]]> and <![IGNORE[…]]>, and their corresponding stylesheets. Also, the current transient CORBA server is being extended to be persistent one, which can maintain in-memory structures of parsed UXF element trees after the shutdown of the server. In addition, the current IDL interfaces of CORBA server are re-designed so that they are compliant with the DOM (Document Object Model) standard interfaces [24], which is an emerging standard by W3C. It would be quite important step that UXF documents are distributed within the CORBA system that implements DOM interfaces.

7. Conclusion

This paper addresses how the software design documents based on UML can be shared and managed in the distributed environment. We proposed a key enabler for interchanging UML model information called UXF, and showed its applications. With UXF, UML models can be distributed universally. We believe our work shows an important step in exchanging and sharing analysis/design models on the open environment. Information on UXF project can be obtained at [22].

At last, we would like to thank Yuu Tanaka for his help to design UXF DTDs, and Kenji Shirane for his initial input for UXF applications.

8. References

[1] Suzuki, J. and Yamamoto, Y. (1998). Making UML models exchangeable over the Internet with XML. In Proceedings of UML '98.

[2] Rational Software et.al. UML Proposal Summary, OMG document number: ad/97-08-02.

[3] Rational Software et.al. UML Summary, OMG document number: ad/97-08-03.

[4] Rational Software et.al. UML Semantics, OMG document number: ad/97-08-04.

[5] Rational Software et.al. UML Notation Guide, OMG document number: ad/97-08-05.

[6] Rational Software et.al. UML Extension for Objectory Process for Software Engineering, OMG document number: ad/97-08-06.

[7] Rational Software et.al. UML Extension for Business Modeling, OMG document number: ad/97-08-07.

[8] Rational Software et.al. Object Constraint Language Specification, OMG document number: ad/97-08-08.

[9] Rational Software et.al. OA&D CORBAfacility, OMG document number: ad/97-08-09.

[10] Grady Booch. Object-Oriented Analysis and Design 2nd Edition, The Benjamin/Cummings Publishing, 1994.

[11] Rumbaugh, J et.al. (1991). Object-Oriented Modeling and Design, Prentice Hall.

[12] Jacobson, I. (1995). Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley.

[13] JavaSoft. (1997). JDK 1.1.6 Documentation.

[14] Bray, T et.al (ed.). Extensible Markup Language (XML) 1.0, W3C.

[15] Ion P et.al (ed.). Mathematical Markup Language, W3C.

[16] CheckFree Corp et.al. Open Financial Exchange Specification 1.0.2, Open Financial Exchange.

[17] Adler, S et.al. (1998) A Proposal for XSL, W3C.

[18] Eve Maler et.al. (ed.), XML Pointer Language (XPointer), W3C Working Draft 3,.

[19] Maler, E et.al. (ed.), XML Linking Language (XLink), W3C Working Draft 3.

[20] A series of CDIF specifications are available at http://www.cdif.org/

[21] .Rational Software. UML-Compliant Interchange Format, OMG document number: ad/97-01-13.

[22] http://www.yy.cs.keio.ac.jp/~suzuki/project/uxf/

[23] Ohno, K. and Bayer, M. (1998) Development of SGML/XML Middleware Component. in Proceedings of SGML/XML'98 Europe.

[24] Apparao, V et al. (1998) Document Object Model Specification, W3C Working Draft 16.

Appendix A:

```
<!ELEMENT Model        (TaggedValue?, Package*)>
<!ELEMENT TaggedValue  (Tag*)>
<!ELEMENT Tag          (#PCDATA, Value*)>
<!ELEMENT Value               (#PCDATA)>
<!ELEMENT Note         (#PCDATA)>
<!ELEMENT Package      (TaggedValue?,
                        Note*,
                        Dependency*,
                        ClassDiagram?,
                        CollaborationDiagram?
                        StatechartDiagram?)>

<!ATTLIST Package
   NAME        CDATA #REQUIRED>
<!ENTITY  % ObjectElements "(TaggedValue?,
                             (Attribute
                              |Operation
                              |Generalization
                              |Association
                              |Dependency
                              |Note)* )">
<!ENTITY % ClassDiagram
   SYSTEM "class_diagram.dtd">
%ClassDiagram;
<!ENTITY % CollaborationDiagram
   SYSTEM "collaboration_diagram.dtd">
%CollaborationDiagram;
<!ENTITY % StatechartDiagram
   SYSTEM "statechartdiagram_diagram.dtd">
%StatechartDiagram;
```

Appendix B: UXF DTD for class diagram

```
<!ELEMENT ClassDiagram (TaggedValue?,(Class
                                      |Interface
                                      |Note)*)>
<!ELEMENT Class     %ObjectElements;>
<!ELEMENT Interface %ObjectElements;>
<!ATTLIST Class
  NAME         CDATA #REQUIRED
  ABSTRACT     (true|false) "false"
  VISIBILITY   (public|private) #REQUIRED
  ACTIVE       (true|false) #IMPLIED>
<!ELEMENT Attribute    (Note*)>
<!ATTLIST Attribute
  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
  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
   TYPE        CDATA #IMPLIED
   NAME        CDATA #REQUIRED
   DEFAULTVAL  CDATA #IMPLIED
   DIRECTION   (in|out|inout) #IMPLIED>
<!ELEMENT Exception    EMPTY>
<!ATTLIST Exception
   NAME        CDATA #REQUIRED
   BODY        CDATA #IMPLIED>
<!ELEMENT Generalization EMPTY>
<!ATTLIST Generalization
   FROM        CDATA #REQUIRED
   TYPE        (public|private|protected)
"public">
<!ELEMENT Association ((AssocRole,
PeerAssocRole)| Note*)>
<!ATTLIST Association
   PEER        CDATA #REQUIRED
```

```
   NAME        CDATA #IMPLIED>
<!ELEMENT AssocRole    EMPTY>
<!ATTLIST AssocRole
   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
   MULTIPLICITY       CDATA #IMPLIED
   ORDERING    (ordered|unordered) #IMPLIED
   ROLENAME           CDATA #IMPLIED>
<!ELEMENT Dependency   (Note*)>
<!ATTLIST Dependency
   PEER        CDATA #REQUIRED
   NAME        CDATA #IMPLIED
   DESCRIPTION CDATA #IMPLIED
   DEPKIND     (refine|bind) #IMPLIED>
```

Appendix C: UXF DTD for collaboration diagrams

```
<!ELEMENT CollaborationDiagram (TaggedValue?,
                                (Collaboration
                                 |Note)*)>
<!ELEMENT Collaboration (TaggedValue?,
                         (Instance
                          |Interaction
                          |Message
                          |Note)*)>
<!ATTLIST Collaboration
    NAME       CDATA #REQUIRED
    CLASSIFIER CDATA #IMPLIED
    OPERATION  CDATA #IMPLIED>
<!ELEMENT Instance (Note*)>
<!ATTLIST Instance
    NAME       CDATA #IMPLIED
    CLASS      CDATA #REQUIRED
    CONSTRAINT CDATA #IMPLIED>
<!ELEMENT Interaction (Message)*>
<!ATTLIST Interaction
    NAME       CDATA #REQUIRED
    CONTEXT    CDATA #IMPLIED>
<!ELEMENT Message (Label)>
<!ATTLIST Message
    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
    PREDECESSOR     CDATA #IMPLIED
    GUARD_CONDITION CDATA #IMPLIED
    SEQUENCE_EX     CDATA #IMPLIED
    RETURN          CDATA #IMPLIED
    MESSAGE_NAME    CDATA #IMPLIED
    ARGUMENTS       CDATA #IMPLIED>
```

Appendix D: UXF DTD for statechart diagrams

```
<!ELEMENT StatechartDiagram (TaggedValue?,
                             (State
                             |CompositeState
                             |PseudoState
                             |Note)*)>
<!ATTLIST StatechartDiagram
  name CDATA #REQUIRED>
<!ELEMENT State (ActionSequence
                |Transition
                |Note)*>
<!ATTLIST State
  name    CDATA #REQUIRED>
<!ELEMENT CompositeState (ActionSequence
                         |State
                         |CompositeState
                         |PseudoState
                         |Transition
                         |Note)*>
<!ATTLIST CompositeState
  name                      CDATA #REQUIRED
  isConcurrent (true|false)       "false"
  isRegion (true|false)           "false">
<!ELEMENT PseudoState (ActionSequence
                      |Transition
                      |Note)*>
<!ATTLIST PseudoState
  kind (initial|deepHistory|shallowHistory
        |join|fork|branch|final) #REQUIRED>
<!ELEMENT ActionSequence (Event*,Action*)>
<!ELEMENT Event (Parameter|Note)*>
<!ATTLIST Event
  name CDATA #REQUIRED>
<!ELEMENT Action (#PCDATA)>
<!ELEMENT Transition (TransitionLabel |Note)*>
<!ATTLIST Transition
  source CDATA #IMPLIED
  target CDATA #REQUIRED>
<!ELEMENT TransitionLabel (Event?, Guard?,
                           ActionSequence?,SendClause?)>
<!ELEMENT Guard (#PCDATA)>
<!ELEMENT SendClause (#PCDATA)
```