

XMI Opens Application Interchange

Introduction: the Value of Open Interchange

We live in an increasingly connected world where it's become natural to expect that we should be able to access some of our most important business assets wherever needed. This is just beginning to happen in the area of software application design through the new interchange standard, XMI.

This document covers how XMI provides open interchange of application components and related business assets. The document describes how XMI enables open scenarios, reviews the relationship of the new XMI standard from the OMG, and shows how this leads to open solutions. The major features and examples of XMI technology will be presented. We'll finish by describing the proof of concept demonstration that underscores both the commitment to and value of XMI.

Open Interchange with XMI

Let's start with diagrams describing different architectures for application interchange. Figure 1a shows the open world of XMI, where the major types of application development tools interchange their information using XMI as the standard. These applications include:

- Design tools, including object-oriented UML tools such as Rational Rose and Select Enterprise.
- Development tools, including integrated development environments like VisualAge for Java and Symantec Café.
- Databases, Data Warehouses and Business Intelligence tools, including IBM DB/2, Visual Warehouse, Intelligent Miner for Data, and Oracle/8i.
- Software assets, including program source code (C, C++, and Java) and CASE tools such as TakeFive's SniFF+.
- Repositories, including as IBM VisualAge TeamConnection and Unisys Universal Repository.
- Reports, report generation tools, documentation tools, and web browsers.

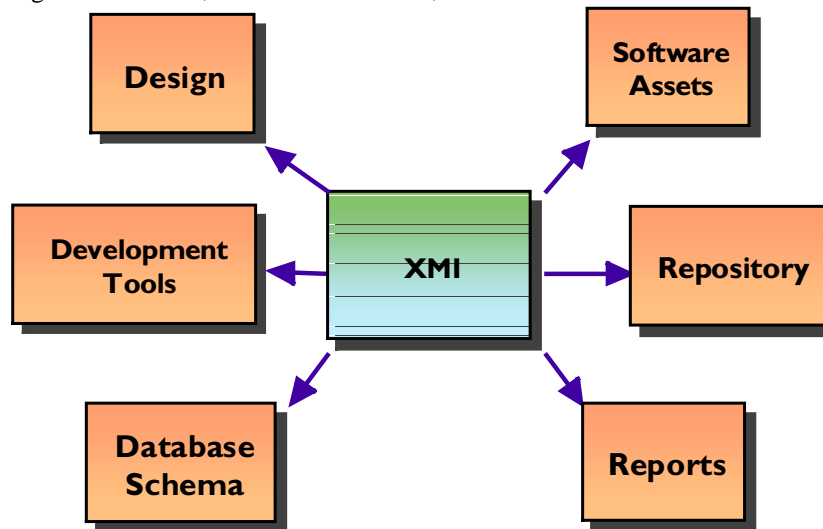


Figure 1a. Open interchange with XMI. 6 bridges written by 6 vendors.

To participate in this architecture, each vendor only needs to add XMI support to their product to leverage access to all the other tools. The system is open and everyone can participate immediately with XMI-enablement.

Furthermore, participants now gain a web-enabled collaborative environment by the close relationship between XMI and XML, a standard technology of the Internet being added to the major web browsers.

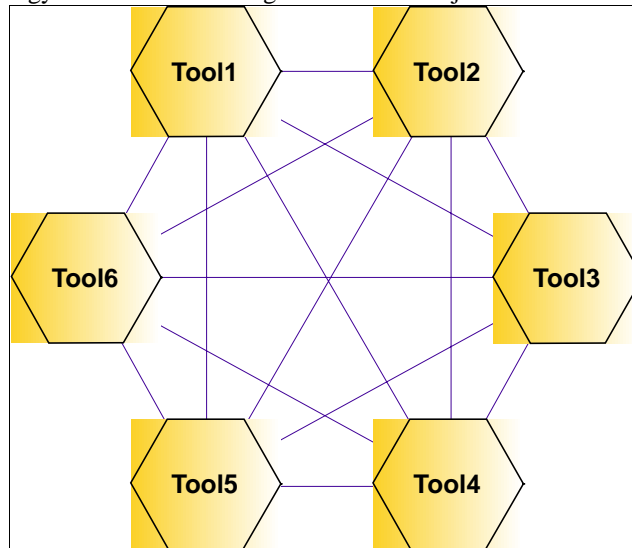


Figure 1b. A web of point bridges. $N*N-N = 30$ bridges by $N = 6$ vendors.

Figure 1b shows the closed situation as it exists today. Each application needs to know about every other tool on the market and build bridges to the products with which their customers insist on sharing data. This creates a tangle of import-export point translators. It's even worse once one factors-in the different versions of each product pair, all following different release schedules and using proprietary formats. In many cases, a bridge might not exist at all. This leaves users stranded without a way to get their products working together. Figure 1b doesn't scale. It grows quickly with the number of products, basically as the number of products squared. We've shown just six tools here, requiring 30 bridges. For hundreds of products, that's thousands of bridges. (Note: The number of bridges is $N*N-N$ instead of $(N*N-N)/2$ because both ends of the point bridges must be built.)

A key observation is that the XMI scenario is not just a lot simpler, it is also far more cost effective. Less work is needed to provide a better solution. Everyone, from tool vendors to ISVs and end users, wins because of the reduced complexity reflected in lower costs and faster time to market.

XMI Open Scenarios

An example of how XMI may be used follows the architecture of Figure 1a. A business analyst makes a business model using an OOAD design tool using the UML (Unified Modeling Language) standard. The design is expressed in XMI and used by a software developer in their language IDE. Next, reports and documentation are published to the web, generated from the XMI. By accessing the design in XMI, database schemas and data warehouses may be created by database designers and business intelligence analysts.

With XMI, users are able to focus directly on their roles, working as a team in an open, distributed environment. Users can employ the right products for each role and interchange their designs in XMI using the Internet.

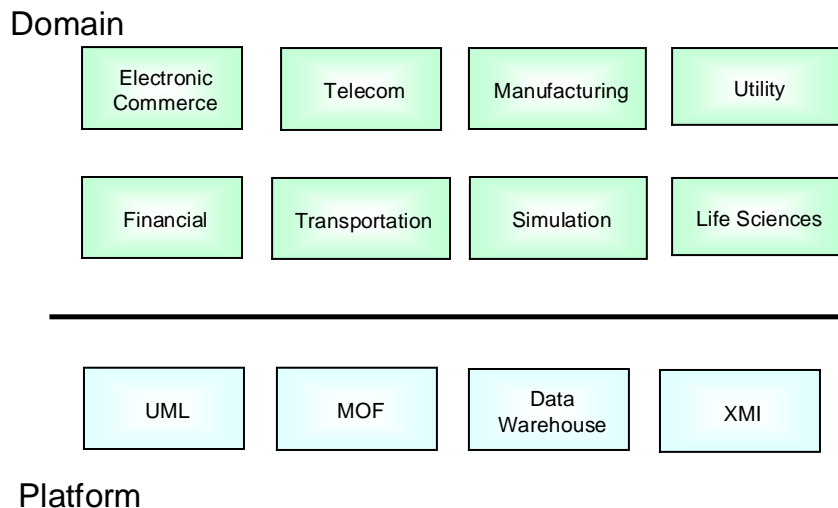
Report generation from XMI is especially easy. The upcoming XSL (XML Style Language) specification from the W3C makes creating HTML web pages from XML a snap by applying style sheets to XML documents. A report may be generated by applying an XSL style sheet to an XMI document. Examples of reports are: "How many components are in my design?" and "What use cases does this design address?". These reports can be created live in a web browser driving directly from the actual XMI designs.

Using XMI opens up entirely new paths between tools, allowing the customer to choose which tools work best in their distributed environment. A customer can choose modeling tools, IDEs, repositories, and databases based on each product's individual merits. Vendors and ISVs win because their best-of-breed solutions can work with the customer's extensive infrastructure.

XMI and the OMG

The OMG (Object Management Group) is an internationally recognized standards committee broadly organized as Domain and Platform task forces (Figure 2). Domain task forces focus on specific industry technologies and the Platform task forces focus on core technologies which can be broadly applied, such as UML, data warehousing, and XMI. The OMG is involved in a number of important industry domains, such as finance, manufacturing, and telecommunications. We will describe how XMI can be used to help provide open solutions as the domains evolve.

Figure 2. The Object Management Group.



XMI is an acronym for “XML Metadata Interchange” and in turn XML stands for “eXtensible Markup Language.” The “X” in XMI means both XML and eXtensible. The XMI generation rules described below provide that extensibility. We’ve all heard the expression, “Provide food and eat for a day, but provide tools and knowledge and eat for a week.” XMI specifies DTD generation rules that allow the XMI technology to be extended to incorporate open interchange with many types of information assets.

Each of the OMG domains shown can create models of their key information that may be interchanged and used to generate XMI DTDs. As the domains grow, XMI is applied to generate the appropriate DTD. XMI solves the general interchange problem by creating an open interchange format from a domain representation using UML or MOF, the widely adopted OMG standards for expressing design information. If you can represent your knowledge in UML or MOF, you will be able to transfer that information using XMI.

XMI and the W3C

Although XMI is a technology from the OMG, it is based on the XML standard from the W3C, the World Wide Web Consortium, the standards organization responsible for HTML. The OMG and W3C are cooperative standards bodies that leverage work developed by their peer organizations. For example, the OMG's use of W3C recommended technologies in XMI has the precedent of the W3C's DOM specification incorporating OMG's IDL.

XMI defines sets of rules for using XML productively in an environment of object-oriented information applications, such as application development and data warehousing. These conventions use the XML specifications directly, but do not modify or extend XML. Since XMI is an upcoming standard, it is based on the set of XML recommendations available from the W3C at the time of final submission (October 20, 1998). XMI is designed to be compatible with, but not dependent upon or a replacement for, upcoming XML technologies. These include Namespaces (now a W3C recommendation), XLinks, XPointers, and XML-Schema. In particular, XMI will use the future capabilities of XML-Schema, directly using new features such as XML data types and improved mechanisms for document type declarations.

As the W3C continues to produce and evolve XML standards, XMI will evolve to incorporate those additions.

Architecture of XMI

We've seen that the XMI interchange architecture provides value in terms of open standards, open choices, and lower costs and complexity. Let's look at how XMI works, starting with the relationship between XMI and XML.

XML

XML consists of two parts: documents and DTDs (Document Type Declaration). Documents contain the information as a set of tags, while DTDs specify the rules for how tags may be used in a document. A simple analogy would be filling out a form with fields for name, address, and phone number. The document would be the actual specifics filled in, and the DTD would be the form itself. Both the document and DTD work together to provide meaningful information.

An example of a XML document for an automobile is shown in Figure 3a. Tags, also called Elements, are enclosed in angle brackets, where each start tag has a matching end tag, denoted with a slash symbol, so that the tags form a tree-structured hierarchy. When tag is nested inside another, it is referred to as the "content" of the containing tag. Here, a particular Auto consists of tags specifying a Make of Ford, a Model of Mustang, a Year of 98, a Color of blue, and a price of 25000.

```
<Auto>
  <Make>   Ford           </Make>
  <Model>  Mustang       </Model>
  <Year>   99            </Year>
  <Color>  blue          </Color>
  <Price>  25000         </Price>
</Auto>
```

Figure 3a. XML document for an automobile.

The corresponding auto DTD is shown in Figure 3b. The auto element is declared as being able to contain elements for Make, Model, Year, Color, and Price. The DTD can optionally specify multiplicities for contained elements, such as '*' meaning 0 or more and '?' for 0 or 1. The auto DTD effectively states what an auto may have.

```
<!ELEMENT Auto (Make, Model, Year, Color, Price)>
```

Figure 3b. XML DTD for an automobile.

There are two key considerations for using XML. One is that the human comprehensibility and verbosity of XML documents leads to very large documents, often on the order of megabytes. XML is a natural candidate for compression, often reducing the size of documents by over 90% with standard algorithms such as those used in zip.

In practice, the extra time for compression is similar to the savings of accessing smaller documents on local disk. Sending compressed XML across networks, which are often slower than local disks, is likely to result in net savings of time. Other speed improvements are possible through XML's control of the level of document validation and through the choice of standard XML API, such as using SAX instead of DOM.

The other key consideration for XML is identifying how to structure the DTD so that everyone can share information. The determination of what should be allowed in an auto DTD is based on what the concept of an auto is expected to have. If there was a formal definition for an auto, it would make sense for this definition to be reflected in the DTD. Keeping the formal definition of the auto and the DTD synchronized is very important. Updating the auto's definition and requires updating the DTD to reflect the change. XMI provides a way to keep these definitions synchronized using DTD generation.

XMI Generation Rules

XMI defines two sets of rules that provide open interchange and leverage the capabilities of XML. The two sets of rules in XMI are DTD generation and document generation. The DTD generation is used to specify an interchange format, and the document generation creates documents that use a given XMI DTD.

Suppose that one constructs a model for the auto. Figure 4a shows the auto as a model in UML. The auto is a UML class containing several class-attributes: Make, Model, Year, Color, and Price.

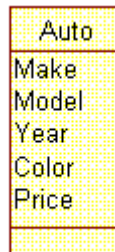


Figure 4a. Auto as a class in UML.

An XMI DTD can be generated from the UML model of the auto as shown in Figure 4b. There is one XML element for each class and class-attribute in the model. The elements are Auto.Make, Auto.Model, Auto.Year, Auto.Color, and Auto.Price. The prefix of Auto ensures that any other classes with similar names, such as a house class with a color class-attribute, will end up with unique tags (House.Color). Each class-attribute has its own XML element declaration, which may contain text values directly (“#PCDATA”) or may contain references (XMI.reference) to another location for defining the value externally. (Useful in cases where the values aren't ideal for XML, such as blocks of binary data or bitmaps.)

```

<!ELEMENT Auto (Auto.Make, Auto.Model, Auto.Year,
                Auto.Color, Auto.Price,
                XMI.extension*)? >
<!ATTLIST Auto %XMI.element.att; %XMI.link.att;>

<!ELEMENT Auto.Make (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Model (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Year (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Color (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Price (#PCDATA | XMI.reference)* >

```

Figure 4b. Auto as an XMI DTD.

Figure 5 shows an example of application exchange using this DTD. The applications are exchanging an XMI document containing a specific auto definition. The document uses the elements from the generated XMI DTD generated from the UML auto model.

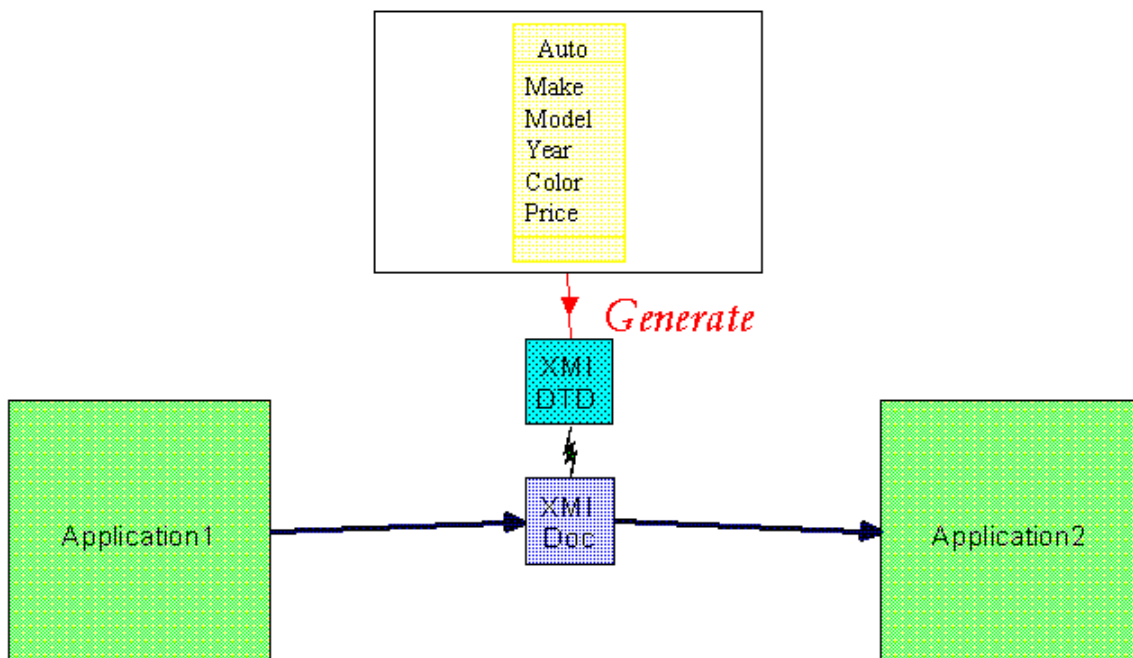


Figure 5. Application interchange of an XMI document using an XMI DTD generated from a UML model.

XMI's rules for DTD generation automatically create new transfer formats with a common, enabling both semantic and data interchange. For example, an XMI-generated DTD for UML allows interchange of object-oriented UML models and class definitions. The Auto UML model of Figure 4a could be interchanged between design tools and IDEs using the UML DTD. An XMI-generated DTD for Java provides interchange of Java classes, a DTD for IDL enables interchange of IDL interfaces, etc. The XMI DTD architecture provides the necessary infrastructure for

advanced information transfer by providing a uniform treatment of object identity, internal and external references, document partitioning, tool-specific extensions, round-trip exchanges, incomplete models, and differences.

Standardizing a set of DTDs alone is not in itself enough for interchange in the general case, since DTDs do not have the ability to express the semantic meaning appropriate for the model. For example, what are the allowed colors for an auto of a given year and style? XML DTDs have no mechanism for expressing this type of information since it requires a whole set of additional concepts which are only available through complete information architectures, such as UML, MOF, and others being developed by the OMG.

Instead of looking to the DTD alone to standardize interchange, UML or similar models must be the source for standards. Once the type of information needed to be exchanged is expressed in UML, XMI will automatically create the DTD and transfer format. Only starting with the semantics results in the ability to interchange at both the data (XML) and semantic (UML) levels.

XMI DTD architecture

Every XMI DTD contains the elements generated from an information model, plus a fixed set of element declarations that may be used by all XMI documents. These fixed elements provide a default set of data types and the document structure, starting with the top-level XMI element. Each XMI document contains one or more elements called XMI that serves as a top level container for the information to be transferred. XMI is a standard XML element and may stand alone in its own document or may be embedded in XML or HTML documents. The XMI element contains the following structural elements:

- Header, which contains version declarations and optional documentation regarding the transfer.
- Content, which contains the core information that is to be transferred.
- Differences, which specify the differences between two XMI documents to be transferred. This is useful in cases such as small changes to a large set of information, for example, to add a new auto to another document containing a large number of existing autos.
- Extensions, which allows the transfer of private tool information beyond that already present in a DTD. The extensions are especially useful for round-trip exchanges between tools that benefit from preservation of private additional information due to their own internal architecture.

Generating element declarations provides an architectural consistency for each element in several important areas: Object identity, extensibility, navigation, and linking. This consistency lets tools know how to traverse any XMI document or DTD in a regular manner to find the information needed.

- Object identity is standardized through the `xmi.element.att` macro (called an XML entity) which declares markers for optionally declaring unique identity via uuids, plus optional shorthand labels and local document ids.
- Extensibility is standardized by the optional `XMI.extension` element which allows nested extensions in addition to those present in the extensions section.
- Navigation is standardized by the regular pattern from which DTDs are generated. The naming and containment of the `Auto.Color` element is an example.
- Linking is standardized through the `xmi.link.att` macro, compatible with the upcoming XLink and XPointer specifications from the W3C to provide both internal and external linking by ids and by Internet hrefs. In addition, every element may contain a reference to another location for its definition, allowing transparent and consistent linking. An example would be for large designs contained in multiple documents, where cross references take the form of hrefs. Since hrefs may contain queries, it is possible for a link to activate a query in a repository that returns additional XMI information on demand.

XMI document architecture

Figure 6 shows the auto example as an XMI document using elements from the generated auto XMI DTD. The document begins with XML processing instructions declaring the character set and the location of the DTD to use. Since this is a reference, the DTD does not need to be transmitted along with the document. The top level element is XMI which declares that the version of XMI standard being used is 1.0 and sets the structure for the rest of the

transfer. The header contains optional documentation and the content is an Auto element with the appropriate values for each of its tags.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI SYSTEM "auto.dtd">
<XMI xmi.version="1.0" >
  <XMI.header>
    <XMI.documentation>
      An example of an auto.
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <Auto>
      <Auto.Make>Ford</Auto.Make>
      <Auto.Model>Mustang</Auto.Model>
      <Auto.Year>99</Auto.Year>
      <Auto.Color>blue</Auto.Color>
      <Auto.Price>25000</Auto.Price>
    </Auto>
  </XMI.content>
</XMI>
```

Figure 6. Auto as an XMI Document using the Auto XMI DTD.

XMI and non-UML information

XMI works with a wide range of information beyond UML. Examples include database information, Java and C++ class definitions, IDL, Enterprise Java Beans, and Electronic Commerce. Database information is especially important due to its critical role in business operations. The OMG, IBM, database industry leaders, and customers are working together on a standard called Common Warehouse Metadata Interchange (CWMI) to enable the expression of database information and schemas in a common form that can be interchanged using XMI. The CWMI initiative will provide a general definition for the interchange of relational and hierarchical databases, database management and data warehousing, important components of information management and business intelligence.

One of the most common types of CWMI information is the structure of relational databases. A possible representation for relational databases is that a RelationalDatabase contains Tables that in turn contain Columns. Each of these elements also has a name. An example XMI DTD with tags for this information is shown in Figure 7a.

```

<!ELEMENT RelationalDatabase (RelationalDatabase.name, XMI.extension*,
                               RelationalDatabase.tables*)? >
<!ELEMENT RelationalDatabase.name (#PCDATA | XMI.reference)* >
<!ELEMENT RelationalDatabase.tables (Table)* >

<!ELEMENT Table (Table.name, XMI.extension*, Table.columns*)? >
<!ELEMENT Table.name (#PCDATA | XMI.reference)* >
<!ELEMENT Table.columns (Column)* >

<!ELEMENT Column (Column.name, XMI.extension*)? >
<!ELEMENT Column.name (#PCDATA | XMI.reference)* >

```

Figure 7a. Relational Database XMI DTD.

There are tags in the DTD for relational databases, tables, and columns and their names. Relational databases and tables also contain tables and columns respectively, which are also represented as tags. (These relationship tags are especially useful in complex models where there is more than one way for tags to be related.)

The structure of a database is commonly called a “schema.” An example of a relational database schema (Figure 7b) that could be interchanged using the above XMI DTD is a database for accounts (“accountsDB”) containing a table (“customers”) with columns for customer id, customer name, and balance.

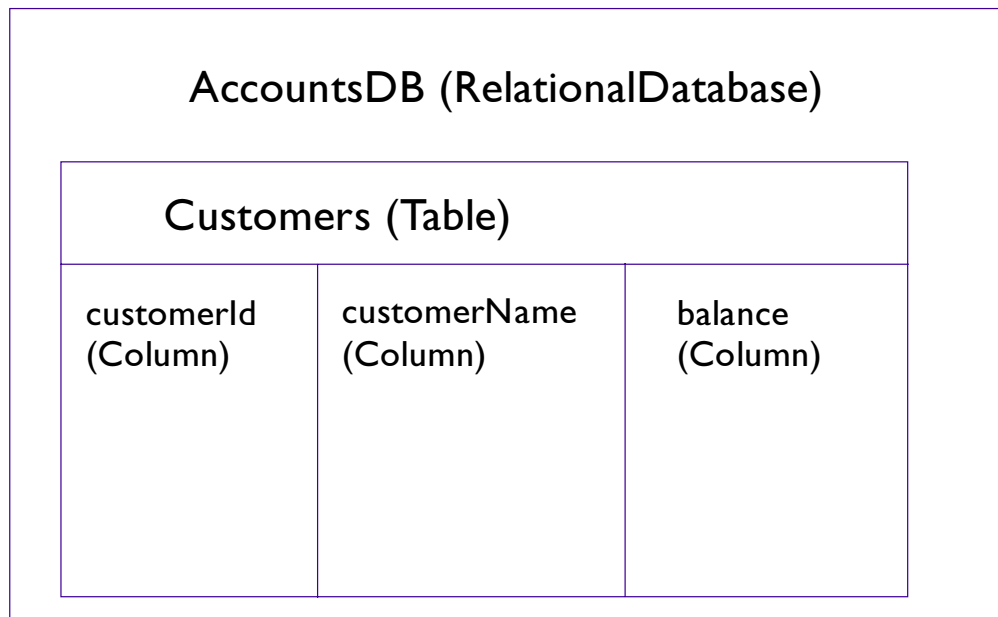


Figure 7b. Database schema example.

```

<RelationalDatabase>
  <RelationalDatabase.name>accountsDB</RelationalDatabase>
  <RelationalDatabase.tables>
    <Table>
      <Table.name>customers</Table.name>
      <Table.columns>
        <Column><Column.name>customerId</Column.name></Column>
        <Column><Column.name>customerName</Column.name></Column>
        <Column><Column.name>balance</Column.name></Column>
      </Table.columns>
    </Table>
  </RelationalDatabase.tables>
</RelationalDatabase>

```

Figure 7c. Database schema XMI document using Relational Database XMI DTD.

This XMI document uses the relational database XMI DTD in Figure 7a, filling in the tags with information from the example database schema.

XMI companies and products

IBM and Unisys lead the XMI standardization effort and are joined by other key members of the software industry. In addition, IBM is incorporating XMI in several products and will have a formal announcement. Proof of concept demonstrations (below) have included WebSphere, VisualAge for Java, VisualAge TeamConnection Enterprise, DB/2, and the XML for Java parser from AlphaWorks.

The 29 submitting and supporting companies are: (submitters) International Business Machines Corporation, Unisys Corporation, Cooperative Research Centre for Distributed Systems Technology (DSTC), Oracle Corporation, Platinum Technologies, Inc., Fujitsu, Softeam, Recerca Informatica, Daimler-Benz, and (supporters) Cayenne Software, Genesis Development, Inline Software, Rational Software, Select Software, Sprint Communications Company, Sybase, Inc., Xerox, MCI Systemhouse, Boeing, Ardent, Aviatis, ICONIX, Integrated Systems, Verilog, Telefonica I+D, Universitat Politecnica de Catalunya, NCR, Nihon Unisys, NTT.

XMI Proof of Concept

Nine products from five vendors were demonstrated working together using XMI technology at the OMG meeting in November 1998, just three weeks after the final XMI specification was made available. The demonstration (Figure 8) showed a round trip exchange, starting from a UML design of an electronic commerce application in Rational Rose, converted into an XMI document, stored as components in IBM's TeamConnection running on a DB2 database, then imported into VisualAge for Java and also into WebSphere Enterprise Component Broker, all using IBM's XML for Java parser. Select imported the same XMI document into Select Enterprise, modified the XMI document, and passed the file to Unisys which read, modified and wrote the XMI using Universal Repository. The modifications were shown in Rational Rose. The process was repeated in Oracle Designer and the updated XMI was sent back to IBM which showed the final XMI contents in Java XML software available from the AlphaWorks site. All this interchange resulted in very long round trip. These cross-product, cross-vendor demonstrations underscore the commitment of the submitting companies to XMI, as well as the practical value of XMI in delivering open solutions to real customer problems.

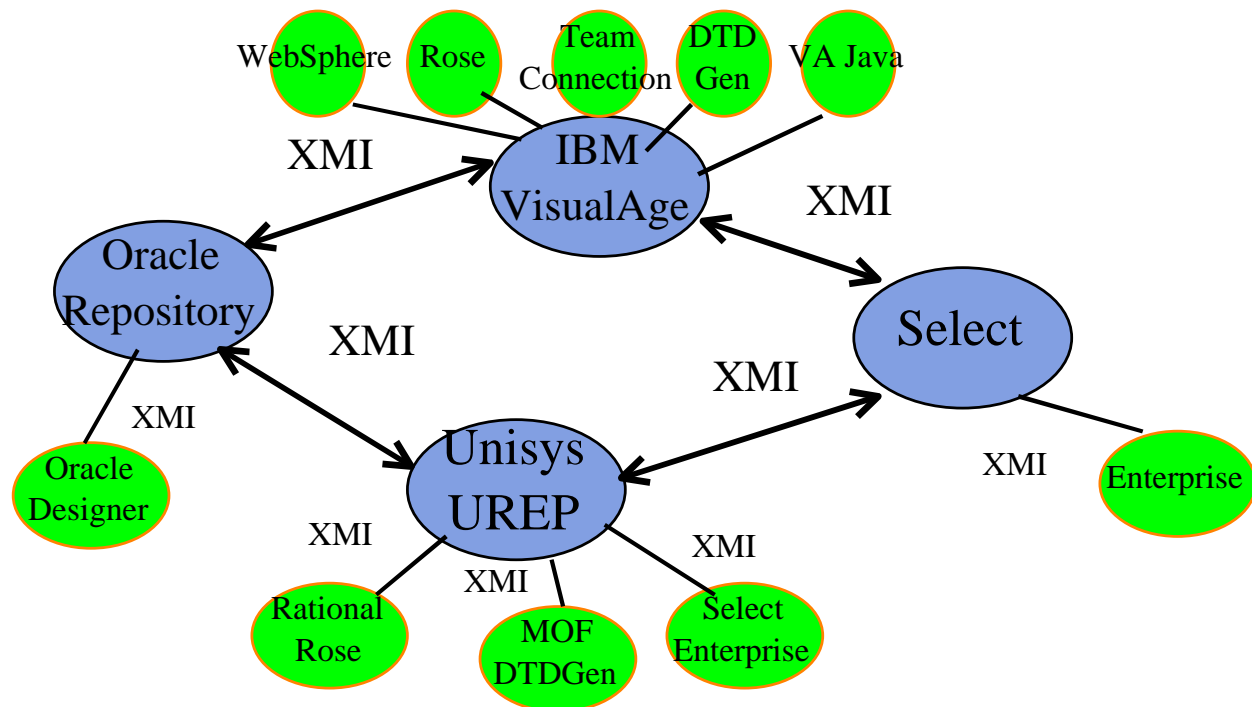


Figure 8. XMI proof of concept demonstrated to the OMG in November 1998.

XMI Open Solutions

Standards-based technologies allow customers to choose the best tool, platform and solution to suit their needs, regardless of vendor -- that's especially key in the heterogeneous environments in which developers operate today. XMI enables open solutions on the critical decision points for application development: platform, language, vendors, middleware, and tools. Only an open solution can provide the best and most complete set of application development tools for distributed computing. And only an open solution lets customers leverage existing investments in assets and skills, modernize them, and extend them to the Web.

Future

The future for XMI appears bright. Final approval of XMI occurred in March 1999, followed shortly by product announcements. As a standard practice in the OMG, an XMI revision task force will be created and chaired by IBM to ensure XMI remains up to date as more new technologies emerge.

Summary

The XMI proposal is the "marriage" of an object technology standard (UML) and a standard for exchanging data over the Internet (XML). Developers can now create applications in a team environment with greater interoperability and consistency. XMI allows developers working in collaborative development environments to build, store, and exchange models over the Internet, regardless of platform, programming language or tool.

References

- XMI <http://www.software.ibm.com/ad/features/xmi.html>
- XML <http://www.w3c.org/XML>, <http://www.ibm.com/XML>
- UML http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#tbl_UML_Specification
- MOF http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#tbl_MOF_Specification

- OMG <http://www.omg.org>
- W3C <http://www.w3c.org>

