

# SHORE – A Hypertext Repository in the XML World

**Barbara Zündorf**  
barbara.zuendorf@sdm.com

**Helge Schulz**  
helge.schulz@sdm.com

**Dr. Klaus Mayr**  
klaus.mayr@sdm.com

sd&m Corporation  
software design & management  
4000 Town Center, Suite 290  
Southfield, MI 48075 USA  
+1 248 351 3535

## ABSTRACT

Large software projects using several programming languages, modeling and documentation methods make high demands on their repository. The sd&m Hypertext Object Repository (SHORE) stores information extracted from XML documents. These documents are generated by parsers that convert project documents of any type into XML. SHORE uses its own model architecture. The submission shows how SHORE's model architecture is mapped to the four level model architecture of the XML Metadata Interchange (XMI) described with the Meta Object Facility (MOF). Using this mapping, SHORE can be complemented by other XMI capable tools. This approach proves especially valuable in multi-language environments and for reengineering purposes. The mapping is only one example of how the repository benefits from the integration of XML-based technologies. The paper closes with a discussion on our experience with XML up to now and how its use strengthens the extensibility of the repository for the future.

## Keywords

exchange format, meta data model, model architecture, reengineering, repository, XMI, XML

## 1 INTRODUCTION TO SHORE

SHORE combines the features of a repository with a hypertext system based on Internet technologies. It is used in development projects at sd&m to create a consistent view of documents developed during different project phases, e.g. requirement specifications, class models, and source code.

Figure 1 shows an overview of SHORE. Project documents and a project specific meta data model are the

input for the repository. The model is described in a document using meta meta model elements. Project documents are integrated by transforming them into a uniform XML based format. Their navigable information is stored in an object-oriented database (Object Store from Object Design [3]).

The user interface of SHORE is a web browser and the repository itself functions as an HTTP server. As long as XML support is not complete in standard web browsers, the XML documents are converted into HTML by the SHORE server.

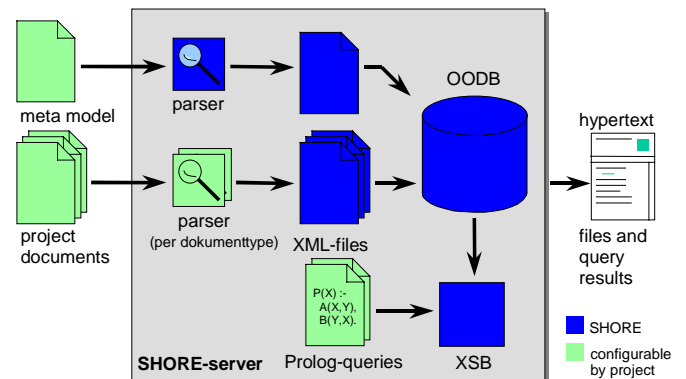


Figure 1: SHORE Overview

Besides HTML navigation, the user can work with XSB Prolog [6] queries to investigate the repository content. Add-ons consists of a graphical representation by a visual toolkit, report generators and full text search capability based on the search engine from Verity [7].

## 2 SHORE MODEL ARCHITECTURE

SHORE's meta meta model is quite simple. It contains three hierarchical types of elements: *document types*, *object types* and *relationship* (Figure 2). An extension to the former SHORE meta meta model is the element *attribute*. By an attribute a (name, value)-pair can be associated to document, object and relationship type instances. An attribute must be exclusively bound to a resource or relationship type.

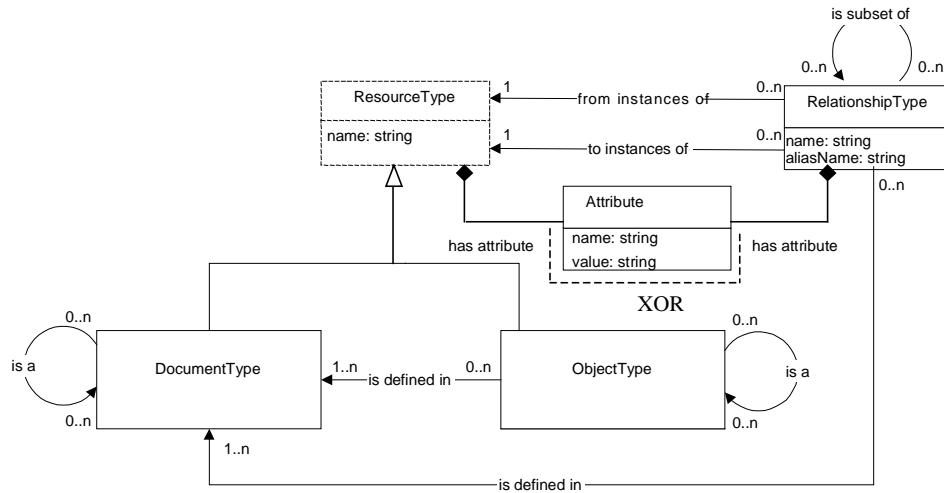


Figure 2: SHORE Meta Meta Model

By the architecture all different kinds of documents can be loaded into SHORE. In the following discussion, we focus on documents that contain source code. The integration of different programming languages can be done with a layered meta model. The first layer describes the concepts that the programming languages have in common as abstract object types. Here you can find object types like “Namespace”, “Type”, “Variable” or “Function”. For each programming language, concrete object types can be derived from the abstract types in a second layer of the meta model: “JavaMethod” can be derived from “Method”, “Method” from “Function” and “Function” from “Namespace”. Relationship types like “VariableBelongs ToNamespace” or “NamespaceCalls Function” can also be defined in terms of the abstract object. The relationship type “NamespaceCallsFunction” can be used for example to describe inter language calls.

With this abstraction principle, a common meta model has been defined for different programming languages that describes the relationships and resource types on a level especially relevant for reengineering. Programming language specific meta models still exist in SHORE but they are now specializations of the common meta model.

### 3 EXCHANGE FORMAT BASED ON XML

SHORE uses an exchange format based on XML. For each document type, there is a parser that transforms the source code into a well-formed XML document. A parser for the document type “JavaSource”, for example, will insert the markup `<JavaClass Name=“myClass”> ... </JavaClass>` for a class “myClass” into the source. For each document, object or relationship type in the meta model, there exists a corresponding XML element type. Each XML element that describes a resource has the attribute “Name” that identifies the document or object. XML elements that describe relationships use the two pairs of attributes “SourceType”/“SourceName” and

“TargetType”/“TargetName” to identify the source and target objects that they connect.

The location of elements is quite obvious because the elements are described within the source. For example, the opening XML element `<JavaClass Name=“myClass”>` will appear at the beginning, and the closing element `</JavaClass>` will appear at the end of the class declaration.

Connections between objects in the repository can be followed by hyperlinks that are inserted at locations where objects or relationships are defined. SHORE uses a special XML element “Hotspot” to mark these locations in the source code. For more information on the exchange format see [2].

### 4 SHORE AND XMI

When we compare the SHORE model architecture with the model architecture of MOF [4], the foundation of XMI [5], both can be distinguished into the following four layers:

	SHORE	XMI
<b>M0:</b> information layer <i>described by</i>	project documents, e.g. source files	data
<b>M1:</b> model layer <i>instance of</i>	SHORE XML files	XMI files
<b>M2:</b> meta model layer <i>instance of</i>	SHORE meta models	MOF-compliant meta models
<b>M3:</b> meta meta model layer	SHORE meta meta model	MOF meta meta model

In both worlds, meta models are defined to describe groups of models of the lower level. Both technologies are built to work with models but with different goals. In SHORE models are used to store documents and all types of relationships between them. XMI is a standard to exchange models between tools that use models (such as SHORE).

To use the XMI-Standard to exchange SHORE-models, the SHORE meta meta model has to be mapped to the MOF meta meta model. The following table gives a summary of this mapping:

SHORE	MOF
<i>Metamodel</i>	Package
Object Type	Class
Document Type	Class
Attribute	Attribute
Relationship Type	Association

### Metamodel

Every SHORE meta model is mapped to a *MOF:Package* instance. The package serves as namespace for all elements of the meta model. The resulting MOF-meta model contains this one MOF:Package instance.

### Object Type

Object types are mapped to *MOF:Class* instances. The name of the MOF class corresponds to the name of the SHORE object type. Inheritance is mapped to the *Generalize* relationship between classes.

### Document Type

Every document type is represented as a *MOF:Class* instance named by the SHORE document type. For all object types defined in a document type, one *representative class* is introduced. All mapped MOF classes of SHORE object types contained in the document type, that have no other supertype become subclasses of this representative class. The container relationship ('defined in') between document and object types can then be preserved by one *aggregation* between document type class and the new representative class.

### Attribute

SHORE attributes are mapped to *MOF:Attribute* instances. For each SHORE attribute, a *MOF:DataType* instance is derived that contains the value of the attribute as typecode. The MOF attribute belongs to the MOF class mapped to the SHORE model element with the relationship to the attribute. Name attributes exist implicit for all SHORE model elements and would not be mapped by this approach. Therefore all Top-Level classes of the MOF model are subtypes of a special (abstract) SHORE-

Objects class with a name attribute.

### Relationship Type

Primarily, a SHORE relationship type is mapped to a *MOF:Association* with the same name. If there are relationship types that are subtypes of the first one, the association has to be marked as derived by setting its *isDerived* attribute to true. Besides the association instance, an association in MOF requires two *MOF:AssociationEnd* instances and two *MOF:Reference* instances. The associationEnd and the reference related to the MOF class mapped to the source ('from instances of') SHORE resource type are named by the SHORE relationship type 'name'. The associationEnd with the type of the class mapping the target ('to instances of') resource type and the corresponding reference are named by the 'aliasName' of the relationship type.

### Export/Import

As a speciality, SHORE's meta meta model distinguishes a specific resource type for documents. Nevertheless, its model architecture is very similar to the one used by XMI. By the presented mapping between the meta meta models of both worlds, XMI Import and Export is possible for the repository.

The XMI Import/Export has been used successfully to complement SHORE with other XMI capable tools – e.g. UML supporting tools like Rational Rose. The model transformations have been done with XSLT [8]. Besides the SHORE – MOF meta meta model mapping, two more mappings have been necessary: (a) mapping between the common meta model in SHORE for programming languages and the UML meta model and (b) mapping between the MOF meta meta model and the UML meta model that already existed [4].

There are two scenarios that yield high benefits in a software project:

#### *Meta model export/import*

Export/import on this level enables graphical representation of new meta models with UML tools as well as the documentation, modification and extension of existing meta models.

#### *Model level export*

SHORE fits reengineering needs as design recovery and redocumentation and often deals with an overwhelming amount of information. It is important to visualize the results on an adequate level that must not be too detailed. Useful are views of the system structure, module structure, dependencies between modules and from external data. By visualizing this model information by UML tools, software engineers profit from the repository and these tools with their support for analysis and design of new requirements.

## 5 CONCLUSION

The meta model driven approach to software engineering has many advantages. The SHORE meta meta model with document types, object types, relation types and inheritance within these types is fixed and very simple. The SHORE meta models can be designed flexibly for distinct purposes to connect the entities of specific software projects. In this paper, we have shown how model based tools can interact with the help of exchange formats, here XMI. In an heterogeneous environment, this is often a better solution than one complex meta model for all tasks.

XML is an extremely helpful intermediate language for storing project documents of any type into repositories. Our source code parsers meet a simple, but still very expressive and powerful target language. The resulting XML documents can be loaded into SHORE or transformed into XMI to bridge the gap between repositories and CASE tools.

For reengineering purposes this is an essential aspect. With a parser for COBOL, for example, we might detect which variables are used in which sections. After transforming the resulting SHORE XML documents into XMI, a lot of associations between (important) variables and (important) functions become traceable.

By importing the XMI-document into a CASE tool and visualizing the existing (or non-existing) design, deeper analysis can be performed. Possibly the software engineer detects, that redesign is more promising than starting from scratch.

Today, XML is very well supported by tools and libraries. New solutions emerge quickly. The developers, administrators and users of the repository can take advantage of all these resources. Using XML makes the repository scaleable and easily adapted for future needs. This is especially true and important for the import parsers, the largest investment in SHORE.

XML separates contents, structure and layout. In this way the re-layout of an existing structure becomes easy. In SHORE layout information is added when a user requests a document. This is currently done by our own web server. In the future, this task can be taken over by web browsers that support XSLT scripts. We have already successfully applied Mozilla [9] with its built-in XSLT-engine to perform the XML to HTML transformation online.

When more browsers will support XML, the conversion to HTML at request time will no longer be required. And with Xlink, more specific targeting will be available.

Beyond the scope of Mozilla, structural information can be extracted and combined new for reports with the help of our query language in combination with XSLT. In one

of our projects, the SHORE report generator generated an 800-pages document that combines WORD documents with specification documents in Rational Rose (class diagrams, event trace diagrams and use case descriptions).

XML is a neutral structured format for the contents that we parse. After having imported the resulting XML documents into SHORE, we have with PROLOG a powerful query language to extract objects and relations from that structure. XSLT is a means to re-structure information. As long as software engineering involves navigating in large contexts and transformations of existing structures, an XML-based repository like SHORE is the right choice.

## ACKNOWLEDGEMENTS

We want to thank Axel Großmann. For his master thesis at the University of Dresden, he worked with the SHORE team at sd&m to define the mappings and implement the (meta) model export/import [1].

## REFERENCES

1. Axel Großmann, XMI für prozedurale Programmstrukturen und Transformation in UML. Master Thesis, June 2000, University of Dresden.
2. Andreas Hess, SHORE – An Example of an Exchange Format based on XML, Workshop on Standard Exchange Format, ICSE 2000.
3. Gary Jones, Object Store 6.0, White Paper, October 1999, Butler Group, (<http://www.objectdesign.com/whitepapers/objectstore.html>)
4. OMG, Meta Object Facility (MOF) Specification, Version 1.3, March 2000, Object Management Group, Inc.
5. OMG, XML Metadata Interchange (XMI) Specification, Version 1.1, November 2000, Object Management Group, Inc.
6. The XSB Programming System (Version 2.2), Department. of Computer Science, SUNY at Stony Brook, 2000 (<http://www.cs.sunysb.edu/~sbprolog/xsb-page>)
7. Verity, Verity Search: The Advantages of Advanced Information Retrieval, November 2000, Verity White Paper ([http://www.verity.com/pdf/MK0348\\_Search\\_WP.pdf](http://www.verity.com/pdf/MK0348_Search_WP.pdf))
8. W3C, XSL Transformations (XSLT), Version 1.0, W3C Recommendation, November 16, 1999. On-line at <http://www.w3.org>
9. XSLT in Mozilla. On-line at <http://www.mozilla.org/projects/xslt>