Issues in Implementing an Open Source-based XML Repository Manager for Application Maintenance and Adaptation

Alexandra Galatescu, Vladimir Florian, Laura Costea, Dan Conescu
National Institute for R&D in Informatics,
8-10 Averescu Avenue, 71316 Bucharest 1, ROMANIA
E-mail:{agal, vladimir, laura, dconescu}@ici.ro

Abstract

During an European IST (Information Society Technologies) project, called MECASP (Maintenance and improvement of component-based applications diffused in ASP mode), an open source-based architecture of an XML repository manager aiming at application maintenance and adaptation has been conceived. It raises many problems and implies an additional work for the connection of the involved open source software, for adding missing functionality (usually with a high degree of complexity) and for wrapping it with MECASP-specific functionality.

The repository contains versioned descriptions (models in XML) of the existing applications, subject to maintenance and adaptation (i.e. creation and management of their revisions or variants).

The paper will briefly present the open source-based architecture of the repository manager and will emphasize the main problems to be solved for its implementation. The involved open source software products are Castor, Xindice, XML:DB, Slide. The problems to be solved are correlated with MECASP specific features for application maintenance and adaptation, introduced at the beginning of the paper.

1. Introduction

The XML repository manager (RM) represents the core of the set of tools for software maintenance and adaptation, which are going to result from the on going European IST (Information Society Technologies) project, called MECASP (Maintenance and improvement of component-based applications diffused in ASP mode).

MECASPls mainly devoted to the maintenance of application software, but it will be useful for general purpose software as well.

The repository contains versioned models (descriptions in XML) of existing applications and resources, subject to maintenance and adaptation (i.e. to the creation and management of their revisions or variants). The content of the models and the maintenance strategy in MECASP are beyond the scope of the paper.

MECASPls features for software maintenance and adaptation. The main features that differentiate MECASP from the existing version management products, are:

- maintenance and adaptation of heterogeneous applications and software resources (built with heterogeneous development tools). MECASP is supposed to maintain and adapt Java projects, objects in relational DBs, forms, reports, graphical objects, documents etc.
- versioning the XML descriptions of the applications and resources (models in MECASP), correlated online with the changes in the corresponding physical applications and resources.
- automatic merge of the variants for heterogeneous resources, relying on rule-based automatic decisions for inconsistencies solving;
- synchronous and asynchronous multi-user work on the same version of the project;
- installation of a new variant of a running application;
- uniform interface for editing heterogeneous types of resources allowed in MECASP, using a MECASP-specific browser.

In Section 3 are enumerated the main issues to be solved for the repository manager implementation, with respect to the first five features.

Open source software involved in the repository manager architecture. A MECASP implementation objective is to rely on existing open source, standard and portable software. For the implementation of the repository manager, four open source software products have been chosen:

- Castor, an open source data binding framework for Java [1]. It is used in MECASP for the mapping
between (1) XML schema and Java classes and between (2) XML documents and Java objects;

- **Xindice**, a native XML (semi-structured) database server [2]. It is used in MECASP to store XML meta-models (descriptions of software templates allowed in MECASP) and models (descriptions of user specific software, complying with MECASP meta-models).

- **XML:DB API** for accessing the XML database [10]. It is accepted by Xindice and, at the same time, it brings portability to the XML database in MECASP, because it is vendor neutral and can be used as an API for many existing XML databases.

- **Slide**, a Jakarta project [3] for managing hierarchical content. Its functions for content and structure management and for security and locking are wrapped with the MECASP repository manager functionality.

In MECASP, **CVS** (Concurrent Versions System), an open source and portable version management system [11] will be used as well. But its use will confine to the maintenance of the black-box applications with respect to MECASP (applications that cannot be described in MECASP repository, by meta-models and models)

The paper will briefly present the open source-based architecture of the repository manager (in Section 2) and will emphasize the main problems for its implementation (in Section 3).

Similar issues for software maintenance or software deployment using XML-based architecture descriptions are treated in other projects as well (e.g. [4], [5], [6]). Versioning and detecting changes in XML documents are still subjects to debates (e.g. [7], [8], [9]). The presentation of MECASP solutions regarding the application architecture and maintenance and the comparison with the existing research solutions in this respect are beyond the scope of this paper.

### 2. Architecture of the Repository Manager

The repository manager architecture in MECASP has three levels (see Figure 1):

- **data level** containing:
  - XML schema that describes the general architecture of the MECASP projects,
  - XML database (Xindice), initially populated with XML meta-models and, then, with user-defined models.

The XML meta-models (predefined or imported from existing applications) represent templates in XML for the MECASP projects. They comply with the general architecture in XML schema.

The **XML models** for MECASP projects describe domain specific applications maintained with MECASP. Each model results from the customization of the definitions in a certain meta-model provided in MECASP.

- Descriptors of the hierarchies built with Slide. Now, they are stored in mySQL, but later they will be stored in the XML database as well.

- **general-purpose functionality** for the correlation of the MECASP-specific functionality with the XML database (data level). It implies:
  - **off-line functions** (during the repository development), such as: (1) the creation of the Java classes from XML schema (with Castor) and (2) the initial population of XML database with meta-models (using the XML resource manager component in Figure 1).
  - **on-line functions** (during the repository utilization), such as: instantiation, unmarshal, marshal, serialization of MECASP objects and their attachment to Slide's hierarchy (using the XML content store), and the access to the Slide descriptions (now in the JDBC descriptor store).

- **MECAS-specific functionality** for meta-model and model management and versioning, model configuration, filtering the user access, management of the registries and files (e.g. user/ team registry, refresh registry, configuration files etc), initial installation of the repository and then the installation of the changes, repository administration, management of user sessions etc.

The connection of the repository manager with the MECASP browser is performed by means of a Java servlet object that (1) receives queries for services from the browser, (2) calls the appropriate RM function, (3) gets the results from the RM function execution and (4) sends them to the browser.

### 3. Issues in Implementing the Open Source-based XML Repository Manager

This section enumerates the problems for the implementation of the open source-based architecture of the repository manager, in correlation with the MECASP intended features (see Section 1).

**Connecting and wrapping open source software.** The MECASP specific functionality in the repository manager is obtained by:

- the correlation of Xindice and Slide, by the development of the XML Content Store (instead of JDBC Content Store). This correlation helps us attach the MECASP projects and objects to the Slide's hierarchy.
Figure 1 Architecture of the Repository Manager in MECASP
• the wrapping of Slide's functions for the meta-model and model management and versioning. The version graphs are created and maintained using Slide's functions.

• the wrapping of Slide's functions for the implementation of the MECASP-specific security and locking mechanism.

Population of the XML database with meta-models. Most complex meta-models in MECASP are obtained by the conversion from the definitions/ schemas of the existing application/resource types (e.g. the structure and types of properties of the objects in a generic database schema, in a generic Java project, of any generic type of graphical object etc).

This conversion is accomplished in two phases: (1) conversion of application schema into an XML document; (2) conversion of the XML document into a MECASP meta-model.

The problem is that today, most development tools (open source and also commercial products) do not provide functions for the conversion of the application/resource definitions/ schemas to XML (first step above). Until the generalization of this capability, MECASP will provide a limited number of meta-models.

Versioning heterogeneous resources. To implement this feature, MECASP cannot benefit from existing version management tools like CVS or Microsoft VSS (VisualSource Safe), because (1) they deal with the versioning of text files only and (2) they have a primitive mechanism for change tracking and version merge.

For instance, in the CVS delta-like files (that contain the differences between two revisions/ variants of the same application), any change is tracked by a combination of the 'delete' and/ or 'append' operations. In the case of a database, these two operations are not appropriate to switch two columns, for example, in an already populated database, because the existing data will be lost during 'delete'. So, a 'move' operation and a MECASP specific delta representation and processing are necessary for non-text resources.

Delta management. Slide helps us manage versions of XML models, but does not help us manage delta structures (changes from the initial version to the new one). MECASP repository manager provides its own mechanism for delta management. The deltas are bi-directional and this mechanism allows the merge and restoration in both directions (forward and backward), in comparison with the existing tools for version management that allow only backward restoration of the versions.

Also, in MECASP, the changes are semantically interpreted, not only literally processed, as in the existing version management tools.

Locking and transaction management on hierarchical objects. Because multi-user work will be the basic work method with MECASP, it must implement powerful locking and transaction mechanisms.

The application XML models are hierarchical, composed of embedded elements/ objects. This representation leads to the need for specific mechanisms for locking and transaction management on XML hierarchical objects. These mechanisms are not implemented yet in the existing open source XML database servers (including Xindice).

Consequently, for the open source-based repository manager, these mechanisms must be implemented from scratch, with a high degree of generality (in order to foresee further substitution of Xindice with another XML database).

For the implementation of these mechanisms, one needs to know and analyze the most important research and practical results today, with the greatest chances to become the standard solutions for locking and transaction management on XML hierarchical objects [12].

Synchronous and asynchronous multi-user work on MECASP projects. In MECASP, the implementation of the multi-user work will be directed to:

• asynchronous sharing of the same project/ object version, by independent users. In this case, the save operations are independent and result into different variants of the project/ object.

• synchronous sharing of the same project/ object version, by the users of the same team. In this case, a two phase save is necessary in order to synchronize the work of all users of the team. It results into a unique (common) revision/ variant.

The main issues regarding the implementation of the multi-user work on an XML database are:

• the need for powerful locking and transaction mechanisms (see above);

• an automatic, two phase save (commit) mechanism;

• the implementation of a multi-user synchronous refresh mechanism;

• the implementation of a mechanism for the multi-user work recovery from the server and repository manager crashes;
Automatic merge algorithm for heterogeneous resources. The existing version management products (among them, CVS is the most important open source product) have a primitive merge mechanism that only specifies the differences between two lines in the compared (only) text files, without any rule for automatically solving the inconsistencies between them.

MECASp will provide a more complex merge mechanism with the following objectives to reach and problems to solve:

- the merge of the changes on heterogeneous types of applications and resources (not only on text files). This objective implies the merging of the XML models of two revisions/variants of an application.
- the implementation of an automatic rule-based decision mechanism for inconsistencies resolution. According to these rules, the list of change actions is simplified and the change operations are combined. Special types of change operations (e.g. compile, search and replace etc), that are also tracked in deltas, will be treated by specific merge rules.
- the creation of the change files, further used for the installation of a new variant of a running application. These files depend on the type of application. For example, for a database application they are SQL scripts and for a Java project they might represent the new executable file.

Version restoration and bidirectional deltas. The existing version management products usually allow only reverse deltas, i.e. the last version of an object is fully stored and the previous versions are stored by deltas (differences from the last version).

In MECASp, bidirectional deltas are stored, in order to allow version restoration in either direction. Apparently, the 'bidirectional deltas' seems an useless capability, because the last revision is usually most often required and should be fully stored and the reverse deltas appear enough.

In MECASp, this capability is necessary to track the non-standard change actions like "compile", "search and replace" etc and to interleave them with the standard ones ("create", "delete", "update", "move" objects or properties).

Installation of a new variant of a running applications. Besides the initial installation of the repository and repository manager, using an existing installer, MECASp will provide for the installation of a new variant of a running application, by the installation of the changes relative to the schema of the original variant.

The installation of the changes uses the results of the merge operation: the change files created according to the application type. For instance, for installing a new variant of a database application, without interrupting the work with it, the following operations should be done:

- transform the schema of the running application, by the execution of the SQL scripts resulting from the merge of the two variants of the application.
- import the data from the running variant into the new one;
- discard the old variant and start the new one.

The schema transformation and the data import are supposed to run without schema inconsistencies (which must be all solved during the previous merge operation).

Recovery from crashes. The repository and repository manager crashes should be prevented by the implementation of a specific mechanism for: (1) the creation and management of temporary files for storing the currently used versions and the current changes, not saved yet in the XML database (e.g. not committed by all users in a team session); (2) the restoration of the user/team working space.

4. Conclusions

After the brief presentation of the XML repository manager architecture, the problems for its implementation are enumerated and motivated according to MECASp intended features.

The implementation of an open source-based XML repository manager for software maintenance and adaptation is a complex objective, with many problems that must find the best research and practical solutions. Unfortunately, they cannot benefit from existing (even theoretical) solutions, as it is stressed along this paper. Most of them must be solved from scratch, by MECASp specific implementation ideas.

5. References


61