Tools for XML-based Maintenance of Heterogeneous Software

Elaine Isnard, Jean-Charles Mathey
Prologue Software/MEMSOFT Multilog Edition, Sophia-Antipolis, Nice, France
Email: eisnard@prologue-software.fr

Radu Bercaru, Alexandra Galatescu, Vladimir Florian, Laura Costea, Dan Conescu
National Institute for R&D in Informatics, 71316 Bucharest 1, Romania
Email:{radu,agal,vladimir,laura,dconescu}@ici.ro

Enrique Perez
Virtual Desk, 28020 Madrid, Spain
Email: eperez@virtualdesk.es

Abstract. The paper presents research results related to an ongoing European IST project called MECASP (Maintenance and improvement of component-based applications diffused in ASP mode), aiming at developing a set of tools for tracking the evolution of heterogeneous software (built using heterogeneous development tools, such as Java IDEs, relational DBMSs, CASE tools, document editors etc). MECASP cannot benefit from existing version management tools like CVS or Microsoft VSS because these tools deal with versioning text files only and have a primitive mechanism for change tracking and version merge. The paper first presents the XML-based solution for software modeling in MECASP. Then, the basic platform and the general architecture of MECASP are given, along with the limits of the existing open source software for the implementation of the XML repository manager, the core of MECASP. The paper also presents the specific features of the MECASP browser, conversion tools and merger.

1 Introduction

The paper presents a set of tools for tracking the evolution of heterogeneous software, under implementation in an ongoing European IST project called MECASP (Maintenance and improvement of component-based applications diffused in ASP mode).

MECASP specific features, that differentiate it from the existing version management products, are:

- maintenance and adaptation of *heterogeneous software*, built using heterogeneous development tools, such as Java IDEs, relational DBMSs, CASE tools, document editors etc.
- *versioning XML descriptions of the applications or generic software* (models in MECASP), correlated with the changes in the corresponding physical pieces of software;
- *automatic merge of the versions* 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 version of a running application*;
- *uniform interface for editing heterogeneous types of resources* allowed in MECASP, using a MECASP-specific browser.

For managing the software evolution, MECASP cannot benefit from existing version management tools like CVS or Microsoft VSS because these tools deal with versioning text files only and have a primitive mechanism for change tracking and version merge.

In order to maintain and adapt heterogeneous software (Java projects, objects in relational DBs, forms, reports, graphical objects, documents etc), an XML repository is created and managed. This repository contains *versioned XML models* that describe the existing (application or generic) software, subject to maintenance and adaptation (i.e. creation and management of their versions). Each model should comply with an existing *meta-model in XML*, predefined or imported from existing development tools. The meta-models represent templates in XML for the types of projects allowed in MECASP (e.g Java projects, database applications, graphical applications, Web applications etc).

An implementation objective in MECASP is to rely on existing open source, standard and portable software. The architecture of the repository manager relies entirely on open source software (see Section 3). For this reason, its implementation raises many problems and implies 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.

In order to reach its intended features, MECASP is supposed to unify the most important research results and to integrate the most important related products/ software. The main research fields to be considered for MECASP implementation and the authors' conclusions on the existing research solutions in these fields are briefly enumerated below.

Regarding the software description using meta-models and models, the starting point and the inspiration source for MECASP was Oxygene ++ [18], devoted to application development using external descriptions (models) of the graphical objects and interfaces, of the menus, reports and databases. Another project is PIROL (Project Integrating Reference Object Library) [19] that relies on the Portable Common Tool Environment (PCTE), the reference model of ECMA [20]. Concerto [21] is a project for XML representation of parallel adaptive components. The components (described in XML) gather from the environment information on the hardware or software resources and dynamically create XML definitions for these resources. GraphTalk [22] is a generator of graphical editors adapted to semi-formal and informal models. It builds meta-models relying on design technologies (e.g. UML).

Also, research results have been obtained for the automatic generation of XML documents from MOF meta-models [23], for the conversion between XML and relational schemas [24], for the semantic enhancement of the XML schemas [25], etc.

Software maintenance using XML-based software architecture descriptions is treated in other projects as well. The representation languages for software architectures (e.g. [12, 13, 14]) are now adapted to XML (e.g. [4, 5, 6]). Versioning and detecting changes in XML documents are still subject to debates (e.g. [7, 8, 9]). Trans-

action management and locking on XML databases (e.g. [26]) are still in an incipient research stage and do not have practical results.

Important theoretical results have also been obtained in version management and merge algorithms (e.g. [15, 16, 17]), unfortunately without practical results and tools.

One may notice that, most of today's research results treat separate aspects needed in MECASP: software architecture representation, software meta-modeling, generation of XML meta-models from existing applications, version management and merge algorithms, management of the changes in XML documents, transaction management etc.

The paper presents the solution in MECASP for software modeling in XML (Section 2), the basic platform and general architecture of MECASP, as well as the limits of the existing open source software for the implementation of the XML repository manager (Section 3), the specific features of MECASP browser (Section 4), the types of conversion tools in MECASP (Section 5), the basic features of the merger tool in MECASP (Section 6).

2 Software Modeling in XML

For each application, the XML repository contains the application model and the version graph for the application maintenance and adaptation. Each application is seen as a MECASP project. The *representation of the application architecture* in MECASP has three levels:

- *XML schema,* that describes the general architecture for all applications maintained with MECASP,
- *XML meta-models* (predefined or imported from existing development tools), complying with the general architecture in the XML schema. The meta-models represent templates in XML for the types of projects allowed in MECASP. MECASP will provide meta-models for Java projects, database applications, graphical applications, Web applications etc.
- *XML models*, that describe domain specific applications maintained with MECASP. Each XML model externally represents a MECASP project and results from the customization of the definitions in a certain meta-model provided by MECASP.

The XML schema and XML meta-models are predefined in MECASP and the XML models are user-defined and application-specific.

The XML meta-models (and implicitly, the models) integrate data, control and presentation levels in the application architecture.

From the *data* point of view, the meta-models are *object-oriented*, hierarchically organized in XML. Their common schema was conceived as general as possible, in order to comply with any type of application. It contains the following types of elements:

```
Project
    Object …
        Property …
            Attribute ...
```

The project is composed of objects that are qualified by properties and can embed other objects (atomic or container-like objects). The object properties are described and managed by attributes. The relationships between objects are composition and property inheritance.

Figure 1 exemplifies several types of objects allowed in the Oxygene++ meta-model (application, package, component etc).

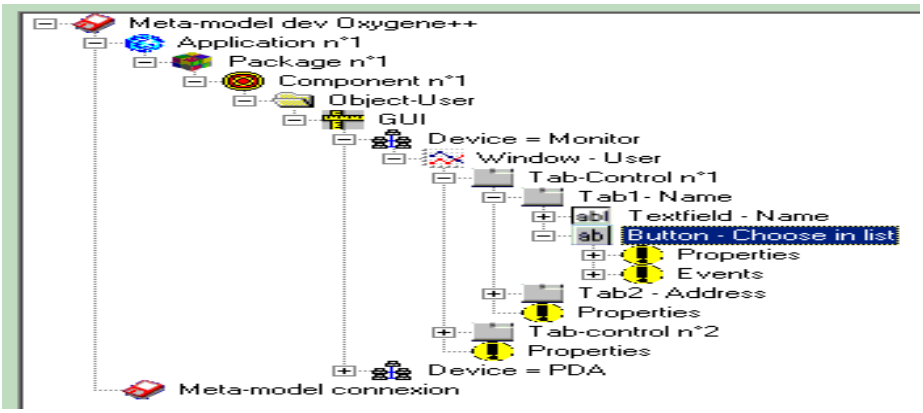

Fig. 1 Types of objects in Oxygene++ meta-model

From the *control* point of view, the XML meta-models are based on *actions*. MECASP manages predefined actions that represent change actions upon application objects. These actions can be: (1) standard actions like "create", "delete", "update", "move" objects or properties or (2) non-standard actions like "compile", "search and replace" etc. In the application meta-model, the actions are represented as particular types of properties of the objects upon which they act. These actions can be correlated with *scripts* or *external tools*, represented in the meta-model as particular types of objects.

From the *presentation* point of view, the objects are described in the meta-models by default values of their properties that can be further changed by the users.

The meta-models in MECASP are generated using a Meta-model Generator tool and, then, are integrated into the XML repository. Each XML meta-model provided in MECASP can be further customized and transformed into models for domain specific applications (see Figure 2). For one meta-model, many models can be built to create many different applications.

When an application is changed, the application model is changed as well and a new version is created and managed in the XML repository.

On the developer's request, a new version can also be created by merging two existing versions of the application (see Section 4).

Each new version of the XML model contains a delta structure composed of all standard and non-standard change actions applied to the original version of the model.

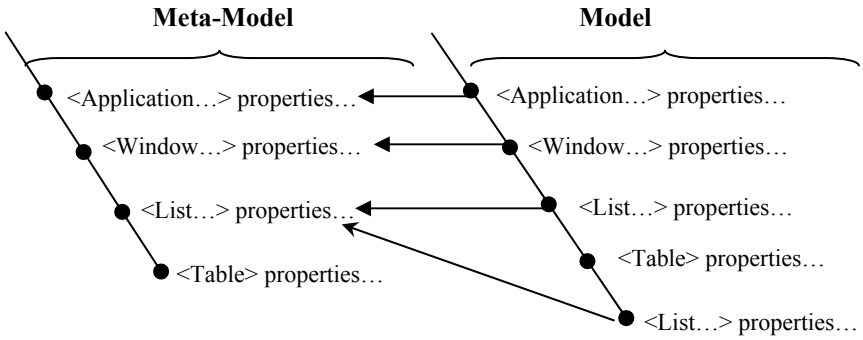

Fig. 2 Building software models based on the definitions in a MECASP meta-model

3 MECASP Platform for Software Maintenance and Adaptation

3.1 Layers in MECASP General Architecture

MECASP platform is composed of tools spread on three layers (see Figure 3):

- an XML repository that includes descriptions of the software entities (application, component, site, client, object, method, DBMS table, etc.) and of the relationships between them;
- an open-ended *set of horizontal* (configuration management, browsing, administration) *and vertical* (editors, compilers, merge manager etc) *tools*. All the market tools can potentially be integrated with MECASP. The integration level is dependent on the tool capability to interface with the outer world;
- a graphical browser that allows browsing through the XML models and their visualisation. It also activates the various tools as appropriate.

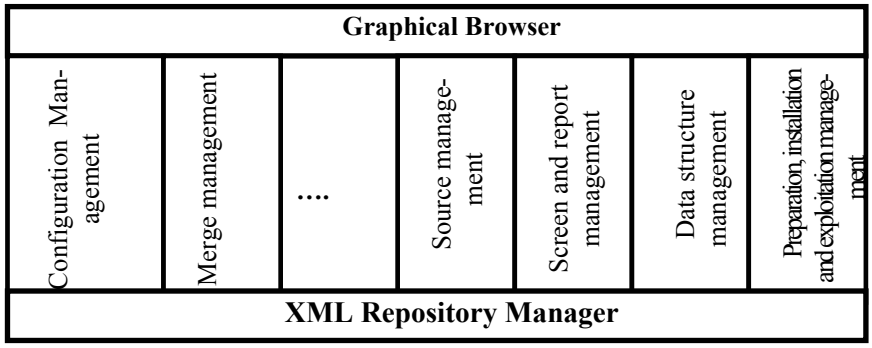

Fig. 3 The three layers in MECASP general architecture

3.2 Functional Architecture of MECASP

Figure 4 represents the basic functional architecture of MECASP.

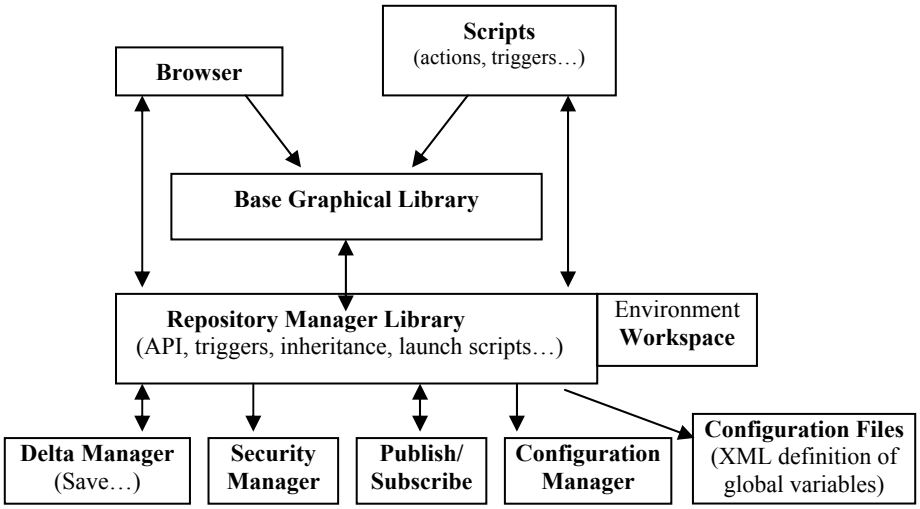

Fig. 4 Functional architecture of MECASP

The *repository manager* plays the central role in MECASP. All the other modules can be considered its 'slaves'. The repository manager handles the storage of the XML models and meta-models and it also co-ordinates the action of the other pieces of software in MECASP. It allows the extraction or building of an XML model or a part of a model. It gives information on the model, manages versions, workspaces and transactions. It sends the 'Refresh' message to clients.

The *browser* instantiates four graphical clients (model explorer, object editor, property editor, action menu) which are graphical views on the information inside the XML models. These clients allow the navigation of the model, object update in the model (attributes of the objects and their structure, e.g. move an object, update a property, etc) and launching actions or specific editors. The browser sends messages to the repository manager providing the types of changes the user wants to perform.

The repository manager proceeds the updates and replies with a 'refresh' message to all clients handling the updated nodes.

The *scripts* are used to code tools and to specialize MECASP generic behaviour. They execute code when an action is launched or when a trigger is fired. When an object is updated, the repository manager fires an update action on this object which, in its turn, calls a script. A user can also launch directly an action, for example by selecting an item in the menu.

The *graphical library* contains the physical graphical objects displayed in the browser's panels and the functions to perform the graphical operations.

The *security manager* is accessed by the repository manager to establish a user session or to check the access rights on the software object handled by the user.

MECASP builds a *workspace* for each user. The workspace is a view on the XML model from the user viewpoint. It contains the current data on which the user works.

When the user is versioning his work, a delta is calculated. It contains the user's changes on the current version of the model.

When a physical resource is versioned, the *configuration manager* stores it physically in the repository.

A *configuration file* (represented as an XML tree) is created for each tuple (user, environment). It contains specific configuration data for the user in his environment, in order to specialize the model. The model is the same for all users, but can be specialized by substituting model parameters (e.g. path names for the working folders, files etc) with specific values existing in the configuration file (the real physical path name).

MECASP components are implemented in Java. Scripts are also written in Java and can be called from an XML scripted definition (e.g. in Ant).

3.3 Open Source Platform for XML Repository Manager

For the implementation of the repository manager (RM), 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 the XML meta-models and 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 (version graphs in MECASP). Its functions for content and structure management, and also 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 software (software that cannot be described in the MECASP repository, by XML meta-models and models).

The implementation of the XML repository manager in MECASP raises many problems with respect to the following aspects:

- connection and wrapping of open source software;
- population of the XML database with meta-models for real applications;
- versioning heterogeneous software;
- management of the deltas (changes from the initial version to the new one);
- locking and transaction management on hierarchical objects;
- synchronous and asynchronous multi-user work on MECASP projects;
- installation of a new version of a running application;
- recovery from crashes.

In the following, the *limits of the involved open source software* in MECASP repository manager will be analyzed, in accordance with the main problems to be solved (see above).

Conversion of the application definitions/ schemas to XML. Most complex meta-models in MECASP are obtained by the conversion from the definitions/ schemas of the existing applications/ resource types (e.g. a database schema, a Java project, graphical objects, etc). The conversion is accomplished in two conversion phases: (1) from application schema into an XML document; (2) from the XML document into a MECASP-specific meta-model.

The limit of the today's development tools (open source and also commercial products) is that they do not provide functions for the conversion of the application and 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. 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 tracking and merging changes. E.g., in CVS delta-like files (that contain differences between two versions of the same software), any change is tracked by a combination of '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 is necessary, along with an algorithm for the semantic interpretation of the change (standard and non-standard) actions. Also, a MECASP specific delta representation and processing have been implemented in order to maintain non-text resources.

Delta management. Slide [3] helps manage versions of XML models, but it does not help manage deltas (changes between two versions). In MECASP, a model is stored along with the list of changes. MECASP RM provides its own mechanism for delta management. Deltas are bi-directional, suitable for both forward and backward merge and version restoration, in comparison with the existing tools, that allow only the version backward restoration.

Also, MECASP RM has its own mechanism for delta interpretation and merge. E.g., suppose a field is deleted in a table. This action fires a trigger and launches a script that deletes the reference to the field in all windows in the application. In the delta, the delete action is stored only once. During the application's version restoration or merge process, the trigger is fired to adapt the change to the current context, depending on the object relationships.

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 hierarchical representation of the XML models 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) [26]. In MECASP

RM, these mechanisms are implemented with a high degree of generality (to cope with a potential further substitution of Xindice with other XML server).

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

- asynchronous sharing of the same version, by independent users. Save operations are independent, resulting into different versions of the project.
- synchronous sharing of the same version, by users in the same team. A Publish/Refresh mechanism is implemented to synchronize the work of all users. This is not appropriate while the users work on text documents (e.g. source code), when the first solution is suitable. The source code should not be synchronized in real time (in standard cases) but, a web server or a database definition could be.

Because several tools can access the same resource, the locking must be implemented even in the single user mode in order to prevent data corruption.

Installation of a new version of a running application. Besides the initial installation of the repository and RM, using an existing installer, MECASP provides for the installation of a new version of a running application, by the installation its changes relative to the schema of the original version. It uses the results of the merge operation (change files). E.g., for installing a new version of a database application, the following operations are needed: (1) change the schema of the running application, by executing the SQL scripts resulting from the merge of the two versions; (2) import the data from the running version into the new one; (3) discard the old version and start the new one. Schema transformation and data import are supposed to run without schema inconsistencies (all solved during the previous merge).

Recovery from crashes. Repository and RM crashes are prevented by a specific mechanism for: (1) the management of temporary files for the currently used versions and the current changes, not saved yet in the XML database; (2) the restoration of the user/ team working space.

4 MECASP-specific Graphical Browser

The graphical browser is a generic tool representing the user interface in MECASP. It allows the visualization (in a graphical form) and the management of all elements in the XML meta-models and models.

The graphical browser has four visual parts:

- *meta-model and model explorer*. The user navigates the list of existing meta-models and models, the hierarchy of the software components, the objects in each component etc (see the left side in Figure 5)
- *object editor*. The user visualizes or edits the object selected in the explorer panel. This editor displays, using a unitary graphical representation, objects of different types (source code, DBs, forms, etc) (see the right side in Figure 5).

This editor displays, using a unitary graphical representation, objects of different types (source code, databases, forms, reports, etc).

- *contextual menu.* The user selects an action from a pop-up menu containing the actions allowed on the element selected in the explorer panel.
- *property editor*. The user changes the properties of the selected element in the explorer panel. In the case of a complex property, a specific editor is launched to edit that property (e.g a style editor).

The four components of the browser rely on a graphical library that contains (1) graphical objects displayed in the panels and (2) functions to perform the operations on the graphical objects. The browser also includes a call-back graphical service, a call-back requests service and a communication layer with RM.

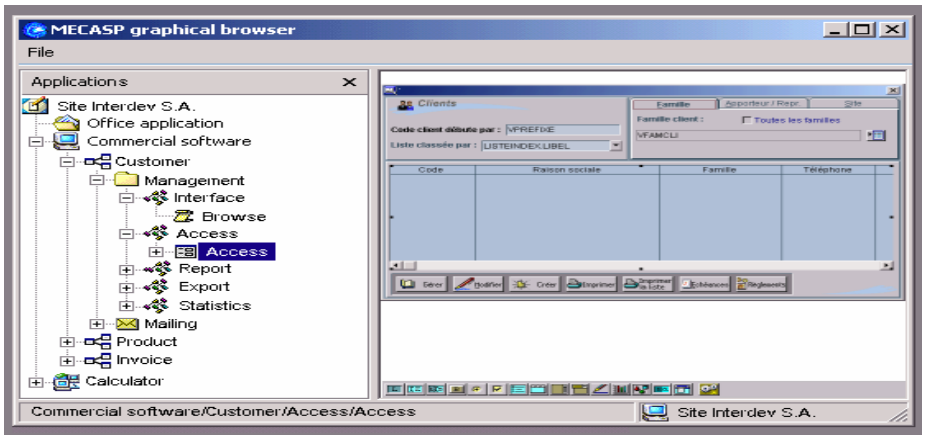

Fig. 5 Explorer (left) and object editor (right) in MECASP

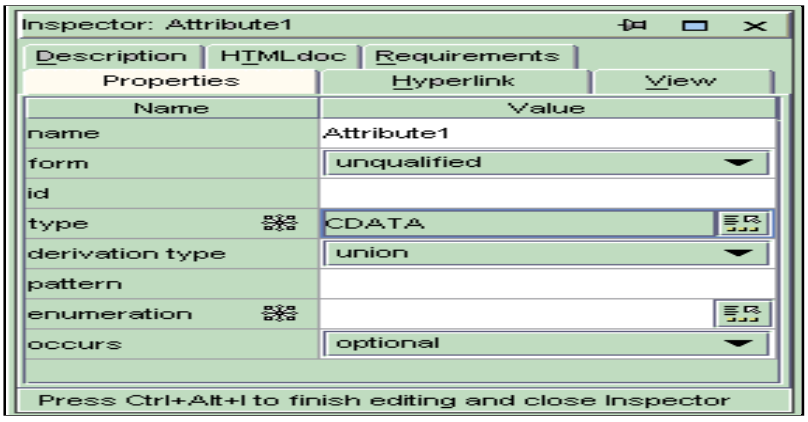

Fig. 6 Property editor in MECASP

5 Conversion Tools in MECASP

The conversion process of the existing software to and from the XML repository includes two functions (see Figure 7):

- import of the user's files into an MECASP-compliant XML model. The files can contain descriptions of 4GLs, java code, forms, reports, database structures etc. The import has two steps: (1) conversion of the user's file into a standard XML file and (2) transformation of the XML file into an XML model in MECASP.
- generation of a file from a part of an MECASP-compliant XML model. When a user modifies an object (e.g. java source code, menu, screen etc), he needs to compile, execute or view the corresponding physical object. The generation process will convert the nodes in the model into the corresponding files.

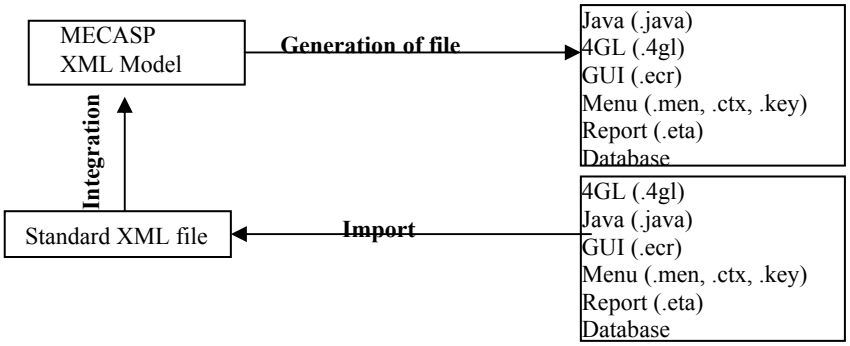

Fig. 7 Convertion tools in MECASP

6 Merge of Versions in MECASP

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 has its own representation of the deltas (enhanced with the semantics of the change operations) and its own mechanism for delta management and for merging two versions of an application. When the user changes the definition of a physical object (e.g. the schema of a table in a database), automatically the changes are reflected into the XML model of the respective object. The merge operation applies to the versions of the application XML models, not to the versions of the physical applications. This strategy allows the merge of versions for any type of objects, not only for text files.

Other distinctive features of the MECASP Merger are:

- it semantically interprets and processes the change actions stored in deltas (e.g. move an object, delete an object, delete or change a property, etc);

- it implements an automatic rule-based decision mechanism for conflicts resolution. According to these rules, the list of change actions is simplified and the change operations of several users are chronologically interleaved. Special types of change operations (e.g. compile, search and replace etc), also tracked in deltas, are treated by specific merge rules.
- it creates change files, further used for the installation of a new version 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.

The MECASP Merger implements two kinds of merge (see Figure 8):

- *merge by integration*. In this case, the merge of B with C is an adaptation of B.
- *complete merge*. In this case, the new version is generated starting from A, by applying the changes in 'delta1' and 'delta2'. The new version D becomes the child of A, the parent of the merged versions.

The sequence of change actions on the same element is simplified according to predefined rules. For example, the sequence 'move'/'update' + 'delete' becomes 'delete', the sequence 'create + 'move'/'update' + 'delete' becomes an ineffective operation, etc.

The automatic decision on inconsistencies solving and on the merge result also relies on predefined rules. These rules depend on the type of change actions, on the type of objects they act upon, and on the role of the version in the merge operation: donator or receptor. These premises impact on the type of merge and on the merge result: fully automatic merge, merge after user decision, refused merge, recommended merge.

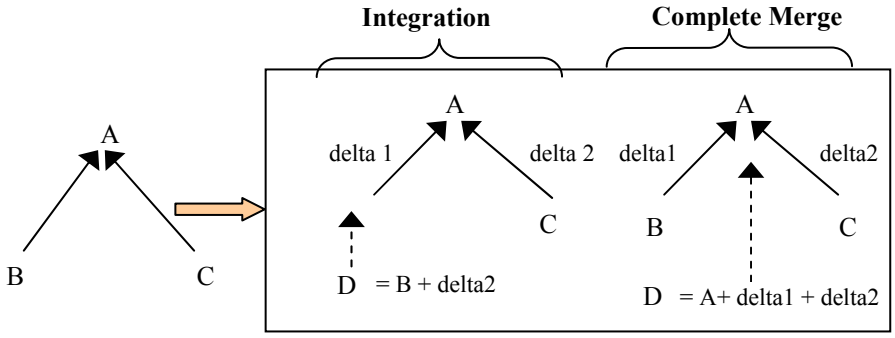

Fig. 8 Two kinds of merge: by integration and complete merge

The graphical user interfaces (GUIs) involved in the merge process are :

- a browser tree, representing a model, with a red mark for the nodes in conflict. When clicking on this mark, a specialized panel is opened explaining the nature of the conflict and proposing choices. This panel can call, via an interface, a specialized window to refine the choice. When a choice is made, the browser tree is updated and so are the edition panels.

- three grouped edition panels. Two panels represent the current state of the original versions (non editable) and the third one represents the merged version (editable). This third panel can be used to edit the new merged version during the merge process. When clicking on the red mark in the browser tree, the panels are also auto-positioned on the selected object.

Each time a choice is made, the tables of deltas in memory are recalculated, and the GUIs are refreshed automatically.

7 Conclusions

The paper gives a brief presentation of the basic features and general architecture of MECASP. It also presents the basic elements for software description using XML meta-models and models. The most important benefit drawn from the external description of the software is the possibility to maintain heterogeneous types of software, in comparison with the existing tools for version management that maintain only text files.

Among the tools for managing the software evolution with MECASP, the paper presents the basic features of the browser, the conversion tools and the merger. Also, related to the implementation of the repository manager, the paper enumerates the limits of the open source software with respect to the problems to be solved. It reveals the fact that, for most of MECASP intended features and functionality, the developers cannot benefit from existing (even theoretical) solutions. Most of the functionality has been solved from scratch, by MECASP specific implementation ideas.

References

1. Exolab,"Castor project", http://castor.exolab.org/index.html
2. Apache, "Xindice Users Guide", http://xml.apache.org/xindice/
3. Jakarta, Slide project, http://jakarta.apache.org/
4. E.M. Dashofy, A. Hoek, R. N. Taylor, "A Highly-Extensible, XML-based Architecture Description Language", *Proceedings Working IEE/IFIP Conference on Software Architecture*, (2001)
5. E.M. Dashofy, "Issues in Generating Data Bindings for an XML Schema-based Language", *Proceedings. XML Technology and Software Engineering*, (2001)
6. R.S. Hall, D. Heimbigner, A.L. Wolf, "Specifying the Deployable Software Description Format in XML", *CU-SERL-207-99*, University of Colorado
7. S-Y. Chien, V.J. Tsotras, C. Zaniolo, "Version Management of XML Documents", *Proceedings 3rd International Workshop on the Web and Databases (WebDB)*, Dallas, TX, (2000)
8. Marian, S. Abiteboul, G. Cobena, L. Mignet, "Change-Centric Management of Versions in an XML Warehouse", *Proceedings 27th International Conference on Very Large Databases (VLDB)*, Roma, Italy, (2001)
9. Y. Wang, D.J. DeWitt, J. Cai, "X-Diff: An Effective Change Detection Algorithm for XML Documents", *Proceedings 19th International Conference on Data Engineering (ICDE)*, Bangalore, India, (2003)

10. XML:DB, "XML:DB Initiative", http://www.xmldb.org/
11. Cederqvist P. et al., "Version Management with CVS", http://www.cvshome.org/
12. Open Group, "Architecture Description Markup Language (ADML)", (2002), http:// www.opengroup.org/
13. Kompanek A. "Modeling a System with Acme", (1998), http://www-2.cs.cmu./~acme/ acme-home.htm
14. Garlan D., Monroe R., Wile D. (2000). "Acme: Architectural Description of Component-Based Systems". In *Foundations of Component-based Systems*, Cambridge University Press, (2000)
15. Conradi R., Westfechtel B. "Version Models for Software Configuration Management". In *ACM Computing Surveys, 30(2), (*1998), .http://isi.unil.ch/radixa/
16. Christensen H.B. "The Ragnarok Architectural Software Configuration Management Model". *Proeedings 32nd Hawaii International Conference on System Sciences, (*1999). http:// www.computer.org/proceedings/
17. Christensen H. B. Ragnarok: An Architecture Based Software Development Environment. *PhD Thesis,* 1999, Centre for Experimental System Development Department of Computer Science Univ. of Aarhus DK-8000 Arhus C, Denmark. http://www.daimi.aau.dk/
18. Prologue-Software. "Documentation of Oxygene++". Technical documentation at Prologue Software/MEMSOFT Multilog Edition.
19. Groth B., Hermann S., Jahnichen S., Koch W. "PIROL: an Object-oriented Multiple-View SEE". *Proceedings Software Engineering Environments Conference (SEE)*, The Netherlands, (1995)
20. ECMA (European Computer Manufacturers Association). "Reference Model for Frameworks of Software Engineering Environments". Technical Report, ECMA, (1993)
21. Courtrai L., Guidec F., Maheo Y. "Gestion de Ressources pour Composants Paralleles Adaptables". *Journees "Composants adaptables"*, Grenoble, (2002)
22. Parallax (Software Technologies). "GraphTalk Meta-modelisation Manuel de Reference, (1993)
23. Blanc X., Rano A., LeDelliou. "Generation Automatique de Structures de Documents XML a Partir de Meta-models MOF", Notere, (2000)
24. Lee D., Mani M.,Chu W. W. “Efective Schema Conversions between XML and Relational Models”, *Proceedings European Conference on Artificial Intelligence (ECAI), Knowledge Transformation Workshop*, Lyon, France, (2002)
25. Mani M., Lee D., Muntz R. R.. "Semantic Data Modeling using XML Schemas", *Proceedings 20th International Conference on Conceptual Modeling (ER)*, Yokohama, Japan, (2001)
26. Helmer S., Kanne C., Moerkotte. "Isolation in XML Bases". Technical Report, The University of Mannheim, (2001)