Distributed Versioning Model for MOF*

Petr Hnětynka[1], František Plášil[1,2]

[1]Charles University, Faculty of Mathematics and Physics, Department of Software Engineering
Malostranské naměstí 25, 11800 Prague 1, Czech Republic
{hnetynka,plasil}@nenya.ms.mff.cuni.cz, http://nenya.ms.mff.cuni.cz

[2]Academy of Sciences of the Czech Republic
Institute of Computer Science
plasil@cs.cas.cz, http://www.cs.cas.cz

Abstract: This paper describes a distributed versioning model, DVM, suitable for the OMG Meta Object Facilities (MOF). We show that the commonly used versioning model, such as CVS, is not sufficient for MOF (and nor is the one proposed in the response [17] to OMG RFP for MOF 2.0 versioning) and propose a solution based on location identifications and sequence numbers together with the rules for creating successor and branch versions. Based on a proof-of-the-concept implementation of DVM and its application to repositories in our SOFA component model, we convey to the reader our positive experience with DVM.

1 Introduction

Configuration management, i.e. mainly management of multiple versions [4] of software entities, is a very important area of software engineering. Not surprisingly, it is also a part of the latest activities of OMG: It is addressed in the Model-Driven Architecture (MDA), namely in the Meta Object Facility (MOF), one of the tools MDA provides for developers.

1.1 MDA

Model-Driven Architecture [19] is the latest approach of OMG to designing and implementing distributed systems. Not being intended as a framework for implementing such systems directly, it is a particular approach to using models in software development (model is a description or specification of a system and its environment). The main objective of MDA is to separate the functional specification of a designed system and the implementation of the desired functionality for a specific platform. There are three primary goals of MDA: (1) portability, (2) interoperability, and (3) reusability, all of them to be addressed through architectural separation of concerns. To achieve the goals, MDA proposes several tools for manipulating and using models in software development, e.g. MOF and its repositories (Section 1.2), UML design tools, tools supporting transformation of models, etc.

1.2 MOF

Meta-Object Facility (MOF) [14] defines an abstract language and framework for specifying, constructing, and managing technology-neutral metamodels. Importantly, MOF defines a framework for implementing repositories that hold the metadata described by a metamodel. Also, MOF specifies the XMI (XML Metadata Interchange) format for meta-data interchange among repositories.

MOF is based on four-layered architecture (by convention the layers are denoted M0, … M3). The lowest layer, M0, is the *information layer* containing the implementation entities (officially "data"). The *model layer*, M1, contains a particular model; i.e. it describes the entities in the information layer. The *metamodel layer* M2 determines the structure and semantics of a particular model). Finally, *meta-metamodel layer*, M3, describes the structure

* This work was partially supported by the Grant Agency of the Czech Republic under the grants 102/03/0672 and 201/03/0911.

and semantics of the abstract language of MOF. Figure 1 illustrates the MOF architecture on two examples showing the hierarchies of modeling abstractions of CORBA IDL and UML.

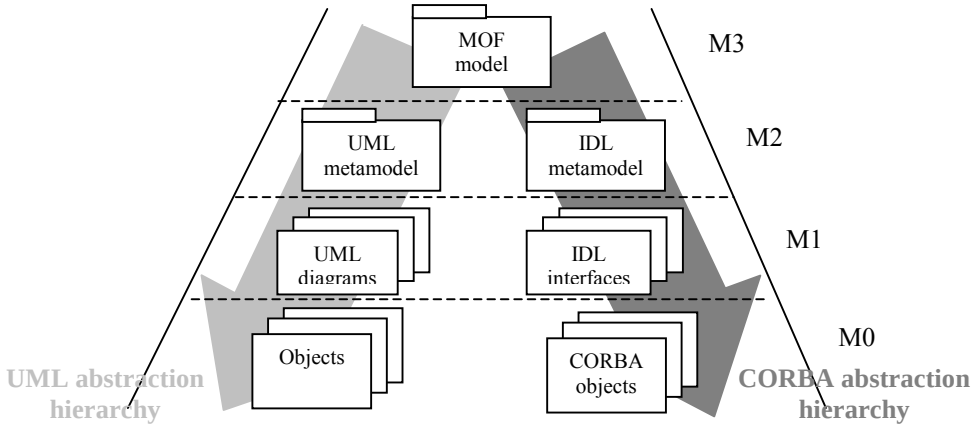

Figure 1. Two examples of MOF architecture layers.

For a new version of MOF, OMG encourages support for accessing and managing multiple versions of entities [16]. The only available proposal for versioning [17] suggests a versioning model akin to CVS and similar tools. Here, it is not assumed that an entity stored in a particular repository can be moved to another repository. But interchange of entities among MOF repositories is one of the key features of MOF. This is the main reason why the version model [17] is not sufficient.

1.3 Goal and structure of the paper

This paper proposes a versioning model for MOF that takes into account the distributed character of MOF and the whole MDA. To address the goal, the paper is structured as follows: Section 2 analyses the versioning model for MOF [17] and explains why this proposal is not sufficient. Section 3 outlines our proposal for distributed versioning in MOF (called DVM) and Section 4 describes a proof of the concept implementation of DVM. In Section 5, other approaches to version support are discussed, while Section 6 concludes the paper.

2 MOF and versioning

Following the current version of MOF is 1.4 [14], a new version (2.0) [15] is being under development. Support for versioning in the MOF is one of the requirements imposed on the new version and it was addressed by a separate RFP [16]. To our knowledge, only one response (the one mentioned above) to this RFP has been available at the time of writing [17].

In this response, the proposed versioning model stems from the approach taken in CVS (Concurrent Versions System) [3]: Each version of an entity has two identifications: (1) An *ID*, which is a string composed as a sequence of integer pairs <major, minor>. The addition of a new member to the sequence introduces a *branch*; merging two branches implies shortening the sequence in general. A version of an entity can always have just one direct successor - these differ in the last pair <major, minor>. (2) A *key*, an integer starting from 0 and incremented by 1 for each new version of the entity.

The version identifiers in *successor relation* create a lattice, typically represented as an acyclic oriented graph (*version graph* of an entity).

Users can look up a version of a given entity by its id or key, obtain the *history* of the version (union of all paths in the version graph leading from the initial version to the given version), timestamp of the version (creation time), and also remove the version from the version graph. Moreover, a version can be associated with labels and annotation (comment in

principle) providing its additional descriptions.

Working satisfactorily in practice, the idea has been employed in several versioning systems, such as in [3,10]. However, it does not work well for MOF as it ignores the inherent distributed character of MDA. The versions of an entity stored in a CVS repository are not intended to be moved/copied to another repository – the developers involved in the same project use a single repository. But in MOF, by definition, the entities in a particular repository can be copied to another repository (technically, trough the XMI format). Consequently, if CVS approach were used, a version with the same identification could exist in several repositories and successors of this version could be created independently in different repositories. In the worse case, two (or more) different versions featuring the same version identifier could enter a specific repository.

Moreover, in MOF the version history of an entity has to be known in all the repositories in which the version is stored. This would not be also addressed by the CVS approach.

To conclude, we have identified two problems related to the versioning model proposed in [17]: (1) Different objects with the same version identification in a single repository, (2) Version history is not available in the repository to which the version was copied.

3 Distributed versioning model (DVM)

In this section, we propose a versioning model (DVM) which solves the problem (1) articulated in Section 2. Even though the problem (2) is not hard to address (e.g. by providing a list of older versions), it is out of the scope of this paper. Nevertheless, researching different techniques of handling (2) is a subject of our future work (Section 6).

3.1 Version identification

In distributed environment, version identification has to be globally unique. An approach to creating unique identifiers is used in COM/DCOM [12]. But such identifiers are opaque, they do not show any relation among versions and potentially their name space is limited.

Another approach is adding a location identifier to the version identification. The whole version identification is then a pair with location identifier and version number (e.g. `<NodeID, 1.3>`). This approach is taken, e.g., in CORBA persistent references [18] and in WebDAV [9]. A characteristic feature in these approaches is that location identifier defines the entity's location absolutely. On the contrary, we propose that location identifier carries the location where the version was created and the history of locations whenever a branch was created after copying (Section 3.2). The location identification format is based on the naming convention of Java packages [5] (technically, the location is given as an Internet domain name).

3.2 Rules for creating versions

For an effective distributed versioning model, the introduction of location identifier is not sufficient - rules for creating successor versions have to be defined as well.

To avoid creating of two different versions with the same version identifier in separate repositories, the direct successor version can be created only in the same location as the direct predecessor version. From the version with different location identifier only a new branch can be created (not a direct successor). Importantly, this new branch will have the location identifier composed of identification of both origin and target repository *(name chaining*). This is because of a possibility of the existence of an entity with the same name initially created in the target repository.

By including locations in version identifiers a version graph becomes an unconnected graph in general. A component of the version graph is created if the initial versions of two entities are created in two different repositories under the same name. These initial versions

have the same sequence numbers but different location identifiers, so that their identification uniqueness is preserved.

To illustrate the idea, consider an entity with the version identifier `<NodeA, 1.3>` stored in a MOF repository with the location identification `NodeA` (Figure 2). Let this entity be copied to a repository with the location identification `NodeB`. Also, let evolution of the entity takes place in both repositories: In `NodeA`, the next version can be a direct successor; on the contrary, in `NodeB` the creation of a direct successor is prohibited so that a branch is to be created. Employing name chaining, the location identifier of this new branch is composed of both the predecessor location identifier (`NodeA`) and the location of the creating repository (`NodeB`). This way, consistency and uniqueness is preserved. Obviously if name chaining were not used, the boxes with the same filling would represent identification conflicts.

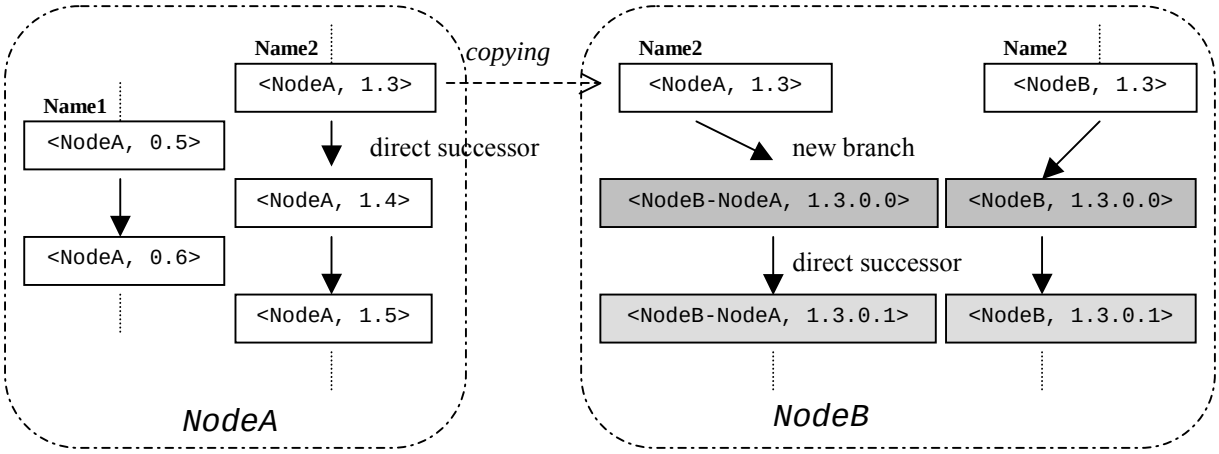

Figure 2. Example of versioning model

4 Proof of the concept

In the current version of SOFA/DCUP [6] (SOFA is a platform for designing applications composed of hierarchical, dynamically updatable components, tied via connectors [20,21,1,7]), type information is stored in the Type Information Repository (TIR) [11]. Each entity stored in TIR is subject to versioning; the versioning model is based on DVM: Location identifier is based on identification of a SOFAnode (SOFAnode is a single environment for developing, distributing and running SOFA components; a SOFAnode has a single TIR). The version identifier for a new branch is composed from the whole identifier of the predecessor version concatenated with a new sequence number and the identification of the SOFAnode the branch is created in. TIR has worked satisfactory in practice since 2000.

After MOF was published, we realized that TIR versioning can be easily generalized for versioning of entities in MOF repositories and introduced DVM as described in Section 3. As a proof of the concept, in a new version of SOFA/DCUP project, we are currently designing and implementing a MOF-based repository with DVM support to replace TIR. The preliminary experience indicates that the idea is sound.

5 Evaluation and related work

Evaluation. The idea of indicating a location name chain in DVM has the following advantages: (1) Being based on Internet domain names, it inherently employs an existing and widely known unique name registry instead introducing a dedicated one “by hand”. (2) Identifiers are human-readable. (3) There is a clear association of the location identifier with the location itself. (4) The name space is potentially unlimited. (5) The successor relation is

reflected in the sequence numbers, still preserved in version identification, while the location identifier ensures identification uniqueness in a distributed environment.

In comparison with the proposal [17] for MOF versioning, DVM enhances the classical versioning model (with dot-separated, number-based version and branches identifiers) by a location identifier. This allows avoiding name conflicts induced by entity copying among multiple repositories typically used in distributed environments.

A potential problem with DVM could be the increased length of a location identifier (because of name chaining), when an entity is repeatedly modified and moved among several repositories. Based with our practical experience with design, such the situation is rare.

Other related work. CVS (Concurrent Versions System) [3] is a system for managing groups of files (e.g. program sources) allowing for simultaneous work with these files in a group of users. It also keeps the whole history of versions of the files with information about changes in the files (time, date, annotation, etc.). CVS also provides a mechanism for solving the name conflict triggered by trying to submit two or more different successor versions with the same identifier (by multiple users). Each version has just one direct successor version; more successor versions imply branches. Versions with branches form a classical version graph. CVS is not directed to use for a development employing multiple repositories.

The Web-based Distributed Authoring and Versioning (WebDAV) [9] is an extension to the HTTP protocol, which allows users to collaboratively edit and manage files on remote web servers. Moreover, its further extension by versioning [22] adds recording the version history of a document over time. The format of version identifiers is not specified (it is WebDAV implementation specific). The version model provides support for creating branches of versions and their subsequent merging. Like CVS, WebDAV does not consider copying versioned files between storage servers.

In the Microsoft's COM/DCOM environment [12], a client-server component-based framework, a component is uniquely identified by a fixed-size (128 bits) Global Unique Identifier (GUID). These identifiers are formed by employing hardware-based identifiers and pseudo-random values. This approach has certain disadvantages, namely: (1) Hardware-based identification employs network card address (MAC) – not reliable should no card be present. (2) Even though the name space is huge, it is limited to 128 bits. (3) The identifier is "opaque". (4) No relation is managed among different versions; a new version of a component is assigned new GUID.

In the Microsoft's .NET framework [13], assemblies (building blocks of applications) are identified by a textual name, version number, culture information (optional), public key, and digital signature. In principle, the version number is composed as a four tuple <major version, minor version, build number, revision>.

CORBA Interface Repository [18], a part of any ORB, provides persistent storage for the IDL types employed in the applications currently known to a specific ORB. Even though the format of identification of entities within an Interface Repository reserves space for version identification, no actual versioning model is (there is no relation among different versions).

There are several distributed configuration management systems with support of distributed repositories like [2, 8] (good comparison of similar systems is provided in [8]). But in all of these systems, distributed repositories form a single logical repository, thus the principles they are based on are not applicable to the MOF repositories.

6 Conclusion and future work

In this paper, we pointed out the necessity of a versioning support in MOF to cope with the existence of multiple entity repositories in distributed environments. The MOF repositories are intended for copying entities among each other, so that versioning must reflect this feature and, at the same time, avoid creating different versions with same identifier in

multiple repositories.

We proposed a solution to this problem by providing a versioning model, DVM, based on location identification and rules for creating successor versions and branches of the stored entities. We conveyed our positive experience with DVM-based versioning in our SOFA/DCUP project.

Currently, we are designing and implementing MOF-based repositories for SOFA/DCUP, employing the DVM versioning model. Also, we are researching the techniques for storing version history in the repositories.

References

1. Adámek, J., Plášil, F.: Component composition errors and update atomicity: Static analysis, Accepted for publication in the Journal of Software Maintenance and Evolution: Research and Practice, 2003
2. Bitkeeper source management: http://www.bitkeeper.com/
3. Concurrent Versions System (CVS), http://www.cvshome.org/
4. Feiler, P. H.: Configuration management models in commercial environments, CMU/SEI-91-TR-7, Carnegie-Mellon University, Pittsburgh, March 1991
5. Gosling, J., Joy, B., Steele, G., Bracha, G.: The Java Language Specification, Second Edition, http://java.sun.com/docs/books/jls
6. Hnětynka, P.: SOFA implementation, http://sofa.forge.objectweb.org/userdoc/doc.html
7. Hnětynka, P., Tůma, P.: Fighting Class Name Clashes in Java Component Systems, Proceedings of JMLC 2003, Klagenfurt, Austria, Springer Verlag, August 2003
8. Hoek, A., Heimbigner, D., Wolf, A. L.: A Generic, Peer-to-Peer Repository for Distributed Configuration Management, Proceedings of ICSE-18, IEEE CS Press, March 1996
9. HTTP Extensions for Distributed Authoring - WebDAV, RFC 2518, 1999, http://www.ietf.org/rfc/rfc2518.txt
10. MacDonald, J., Hilfinger, P. N., Semenzato, L.: PRCS: The Project Revision Control System, Proceedings of SCM-8, Brussels, Belgium, Springer Verlag, July 1998
11. Mencl, V., Hnětynka, P.: Managing Evolution of Component Specification using a Federation of Repositories, Tech. Report No. 2001/2, Dep. of SW Engineering, Charles University, Prague, June 2001
12. Microsoft's Component Object Model, http://www.microsoft.com/com/
13. Microsoft's .NET Framework, http://www.microsoft.com/net/
14. Object Management Group: Meta Object Facility Specification, version 1.4, OMG document formal/02-04-03
15. Object Management Group: MOF 2.0 Core Final Submission, OMG document ad/03-04-07
16. Object Management Group: MOF 2.0 Versioning and Development Lifecycle RFP, OMG document ad/02-06-23
17. Object Management Group: Adaptive/IBM Initial MOF 2.0 Versioning and Development Lifecycle Submission, OMG document ad/03-02-08
18. Object Management Group: CORBA 3.0, OMG document formal/02-06-01
19. Object Management Group: Model Driven Architecture (MDA), OMG document ormsc/01-07-01
20. Plášil, F., Bálek, D., Janeček, R.: SOFA/DCUP: Architecture for Component Trading and Dynamic Updating, Proceedings of ICCDS'98, Annapolis, Maryland, USA, IEEE CS Press, May 1998
21. Plášil, F., Višňovský, S.: Behavior Protocols for Software Components, IEEE Transactions on Software Engineering, vol. 28, no. 11, November 2002
22. Versioning Extensions to WebDAV, RFC 3253, 2002, http://www.ietf.org/rfc/rfc3253.txt