Version control of graphs

Version Control of Graphs

Marcel van Amstel, Mark van den Brand, and Zvezdan Protić

Department of Mathematics and Computer Science
Eindhoven University of Technology
Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{M.F.v.Amstel|M.G.J.v.d.Brand|Z.Protic}@tue.nl

Software versioning is an important part of the software development process. A version (of a software artifact) represents a state of an evolving artifact. Versioning of software artifacts is done by assigning unique identifiers to the artifacts that are being versioned. These identifiers are in a form that allows a temporal relation between different versions of an artifact, e.g. version 1.0 is created before version 2.0 etc. Versioning is used in conjunction with revision control. Revision control assumes the existence of a repository and enables efficient storage, manipulation, and retrieval of any version of an artifact in the repository. Software versioning together with revision control allows teams of developers to work on multiple versions of software at the same time, as well as for teams of developers to work on the same version at the same time, and to synchronize (merge) their work when needed.

Data put under version control is usually contained in (text)files [1]. Hence, file is the unit of versioning (UOV) as described by Murta et al. [2]. Software versioning systems have several important features. One of the most important features is the way storage of different versions of a UOV is handled. This is done by storing only the differences between the current and the previous version. In this way it suffices to store only the initial version completely and all other versions as a list of differences with the previous version. In order to describe the differences between units of versioning there should be a way for comparing them. In software versioning systems, the units of versioning (files) are being compared on the level of paragraph (a line of text ending with carriage return). Hence, the unit of comparison (UOC) in software versioning systems is paragraph [2].

Versioning approaches with file as unit of versioning and paragraph as unit of comparison are generic because they handle all file types in the same way. This is not always advantageous. This approach is appropriate for textual documents, where the required UOC is paragraph. It is less appropriate for source code files, where the preferred UOC is statement, when a line of source code contains more than one statement. It is inappropriate for XMI documents representing UML models, where the preferred UOC is a model element, and not a line of text. Figure 1 gives an overview of the unit of version and the unit of comparison in classical software versioning systems for several file types [2].

With the emergence of model driven engineering, there is a shift from developing code to developing models. Since models are becoming the main design artifacts, the problem of model versioning is becoming more apparent. Existing software versioning systems do not provide immediate solutions to this problem. However, in recent years several authors have proposed solutions to the problem of model versioning. Nevertheless, most of the authors focus only on UML models [2] [3] [1] [4]. There are only a

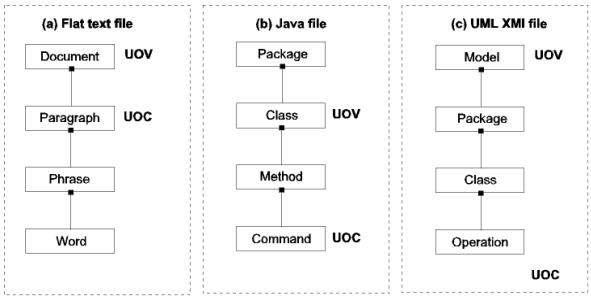

Fig. 1. Units of versioning and comparison for different file types

few approaches in specifying a generic method for versioning models. Rho and Wu [5] present a method for versioning software diagrams. Alanen and Porres [6] present a metamodel independent approach to model versioning. Neither of the two has achieved general public acceptation, most likely because of a lack of a tool that would support those methods.

Our solution to the problem of versioning models is based on the assumption that most models can be transformed into graphs. Therefore, we created a method and system for graph versioning. Figure 2 illustrates our method.

Fig. 2. Schema of a method for putting graphs under version control

This method is based on a bidirectional transformation from graphs to text. The text resulting from the graph-to-text transformation can be put under version control by an existing version control system. The graph to text transformation sets the UOV to a graph, and UOC to a node or an edge. The original graph can be retrieved from the version control system using the text-to-graph transformation.

Next, we propose to use bidirectional transformations between models and graphs. A model can then be put under version control by transforming it into a graph and thereafter into text. Example models include UML models, finite state machine diagrams, flowcharts, Petri-nets, etc. We claim that all models that can be transformed into graphs and back can be versioned using this method.

An example of a class diagram and a possible graph representation of that diagram are depicted in Figure 3. It is easy to give a graph representation of other elements of UML class models, like methods or stereotypes. It is also easy to give a graph representation of features like positioning, color and shape that are not part of the class diagram meta-model, but are introduced by a tool used to create class diagram.

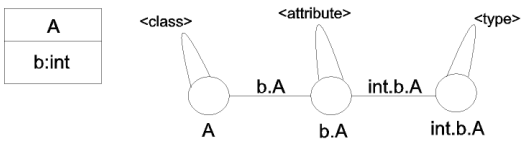

Fig. 3. An example class and its graph representation

References

1. Bartelt, C.: Consistence preserving model merge in collaborative development processes. In: Proceedings of the 2008 international workshop on Comparison and versioning of software models, New York, NY, USA, ACM (2008) 13–18
2. Murta, L., Oliveira, H., Dantas, C., Lopez, L.G., Werner, C.: Odysey-scm: An integrated software configuration managemet infrastructure for uml models. In: Science of Computer Programming, Elsevier (2006) 249–274
3. El-khoury, J.: Model data management: towards a common solution for pdm/scm systems. In: SCM '05: Proceedings of the 12th international workshop on Software configuration management, New York, NY, USA, ACM (2005) 17–32
4. Zündorf, A.: Merging graph-like object structures. In: Proceedings of the Tenth International Workshop on Software Configuration Management. (2001)
5. Rho, J., Wu, C.: An efficient version model of software diagrams. In: APSEC '98: Proceedings of the Fifth Asia Pacific Software Engineering Conference, Washington, DC, USA, IEEE Computer Society (1998) 236
6. Alanen, M., Porres, I.: Difference and union of models. TUCS Technical Report No 527, TUCS Turku Centre for Computer Science (2003)