Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Electronic Notes in Theoretical Computer Science 137 (2005) 57–64
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Using Meta-Model Transformation to Model Software Evolution

Tudor Gîrba [2,4]

Software Composition Group
University of Bern, Switzerland

Jean-Marie Favre[3]

LSR-IMAG Laboratory
University of Grenoble, France

Stéphane Ducasse [1,4]

Software Composition Group
University of Bern, Switzerland

Abstract

Understanding how software systems evolve is useful from different perspectives: reverse engineering, empirical studies etc.. For an effective understanding we need an explicit meta-model. We introduce Hismo, a meta-model which is centered around the notion of history and we show how we can obtain it from a snapshot meta-model. Based on our experience in developing the Hismo reverse engineering system, we show how we can transform a snapshot meta-model in a history meta-model.

Keywords: reverse engineering, software evolution, software history, model driven engineering, MDA, metamodel

[1] Email: ducasse@iam.unibe.ch
[2] Email: girba@iam.unibe.ch
[3] Email: jean-marie.favre@imag.fr
[4] Ducasse and Gîrba gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects "Tools and Techniques for Decomposing and Composing Software" (SNF Project No. 2000-067855.02, Oct. 2002 - Sept. 2004) and "RECAST: Evolution of Object-Oriented Applications" (SNF Project No. 620-066077, Sept. 2002 -

1571-0661 © 2005 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2005.07.005

source: https://doi.org/10.7892/boris.104529 | downloaded: 19.4.2024

1 Introduction

During the 1970's it became more and more clear that keeping track of software evolution was important, at least for very pragmatic purposes such as undoing last changes. Early *versioning systems* such as SCCS made it possible to record the successive versions of software products. This led to text-based delta algorithms. Some basic services were also added in order to model information such as who changed files and why. However only very rudimentary models were used to represent this information – typically a few unstructured lines of text to be inserted in a log file.

While versioning systems enabled recording the history of each source file independently, *configuration management systems* (CMS) attempted to record the history of software products as a collection of versioned source files. Research on configuration management was very active in the 80's and 90's, but the emphasis was still on *controlling and recording* software evolution.

The importance of *modeling and analyzing* software evolution started to be recognized in the early 1970's with the work of Lehman[15]. Yet, it was only until recent years that extensive research has been spent on exploiting the wealth of information residing in versioning repositories. While it was possible to find which specific lines of code were changed between two versions of a particular file, this led to too much and two detailed information to be really useful. However, most of the approaches developed so far, do not rely on an explicit meta model for evolution analysis and do not facilitate the comparison of different evolutions.

Various approaches have been proposed to analyze different aspects of software evolution [1,2,3,4,8,12,13,14,17,18,19,20,21]. Each of these approaches typically focuses on only some traits of software evolution (*e.g.,* which parts are changed the most, what kind of changes happened in a particular part etc.), and do not rely on explicit meta models. In such conditions, is it is difficult to understand what the models exactly refer to. The lack of explicit meta-model makes it difficult to compare and integrate tools even when they provide similar yet different or complementary results.

The authors have already built the case for an explicit meta-model centered around the notion of history [6]. The name of the proposed meta-model is Hismo. We implemented Hismo in a tool called Van which is built on top of the Moose reengineering environment [7]. We use Hismo for characterizing the evolution of software and we validated it in a number of occasions for reverse engineering purposes [9,10,11,16].

In our implementation, Hismo is based on the FAMIX meta-model [5].

Aug. 2006).

However, the concept of history is by no means dependent on FAMIX and the approach can be applied to other meta-models as well. In other words from a conceptual view Hismo could be seen as a *meta-model transformation* transforming a snapshot meta-model into an history meta-model. This paper investigates this idea by considering the problem of evolution analysis from a Model Driven (Reverse) Engineering perspective.

In the next section we briefly present the concepts in Hismo, in Section 3 we describe the transformation needed for obtaining Hismo and before concluding, we discuss our approach.

2 Hismo in a Nutshell

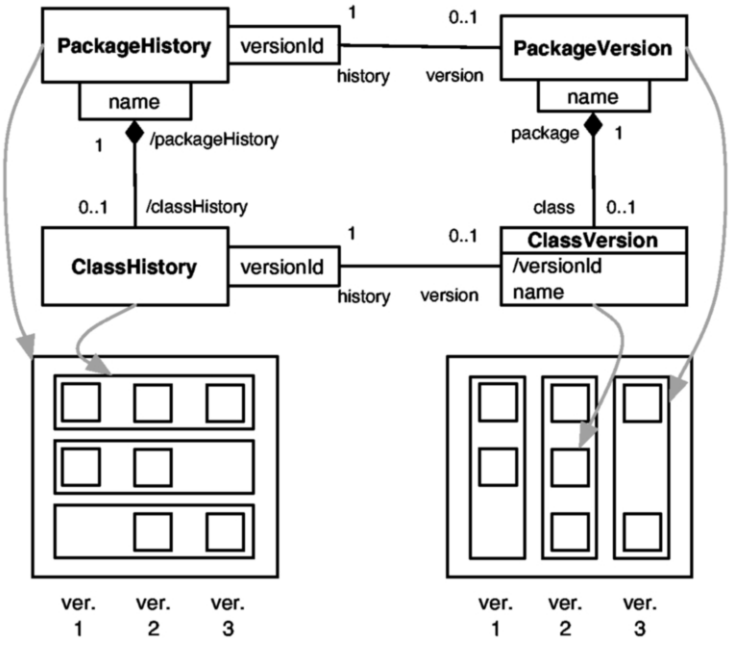

Fig. 1. The relationship between the history and the versions.

The Hismo meta-model is based on the explicit notions of history and versions. As these concepts are generic, they have to be applied to specific entities such as packages, classes, methods or any entity related to software that we want to study and for which having a version makes sense. Figure 1 shows Hismo applied to packages and classes. In the figure we also show the relation between our meta-model and the Evolution Matrix [14]. In the lower part of the figure we represent two Evolution Matrixes in which each cell represents a ClassVersion and each column represents PackageVersions.

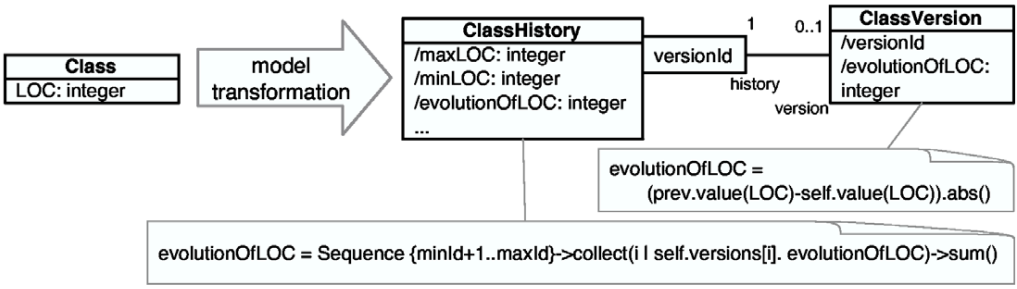

Fig. 2. Transforming the snapshot entities to obtain history as a collection of versions and deriving history properties.

We define a *history* to be a sequence of *versions*. Thus, each line in the Evolution Matrix represents a ClassHistory (left matrix). Moreover, the whole matrix is actually a line formed by PackageVersions (right matrix), which means that the whole matrix can be seen as a PackageHistory (left matrix). In the right side of the figure we see the relationship between the version entities while in the left side of the figure we show the relationships between the history counterparts.

From the figure we see there is a parallelism between the version entities and the history entities. Each version entity has a correspondent history entity. Also, the relationship at version level (*e.g.,* a Package has many Classes) has a correspondent at the history level (*e.g.,* a PackageHistory has more ClassHistories).

Further, we describe how we can generate Hismo based on the snapshot meta-model.

3 Transforming Snapshot Meta-Models into History Meta-Models

In this section we discuss the transformation needed for obtaining Hismo starting from a snapshot model like FAMIX or UML.

In Figure 2 we show in details the transformation which generates from a Class entity in the snapshot meta-model the corresponding ClassHistory-ClassVersion meta-model. Thus, a ClassHistory is a sequence of ClassVersions. Also the model allows us to define history properties based on structural properties.

For example, having the number of lines of code (LOC) as an attribute in a Class, we can derive the minimum or the maximum lines of code in the history. In the figure we show how we derive the Evolution of Lines of Code, as the sum of the absolute differences of the lines of code in subsequent versions.

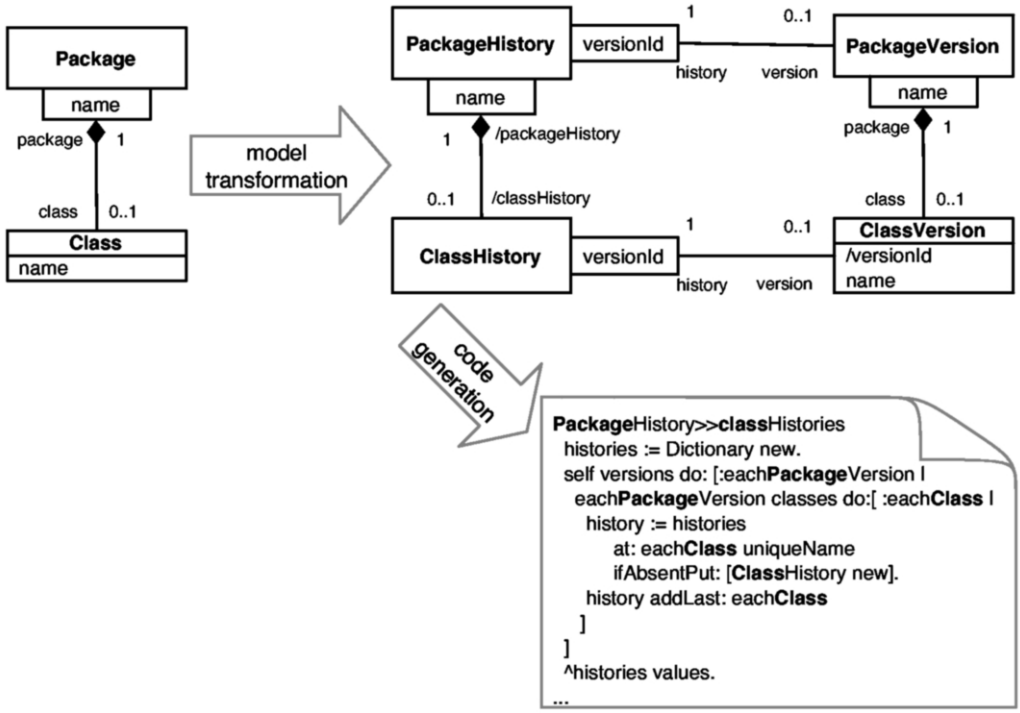

Fig. 3. Obtaining the relationships between history entities by transforming the snapshot meta-model. The bold portions of the generated code show how the algorithm only depends on the snapshot entities and their relationships.

The history properties obtained in this manner, characterize and summarize the evolution. Using such measurements can pinpoint the places which were changed a lot, or places which were hardly changed at all.

Figure 3 shows how we can obtain the relationships between the meta-model entities starting from the structural relationships. On the left side of the figure we have a Package containing multiple Classes. After the transformation we have the PackageHistory as containing multiple ClassHistories. In the down-right side of the figure we see the generated code in Smalltalk for obtaining the ClassHistories starting from a PackageHistory.

4 Discussion

Having history as a first class entity encapsulates the evolution of structural entities. In this paper we showed an example of deriving history properties (*e.g.,* Evolution of Lines of Code) based on structural properties (*e.g.,* Lines of Code). History properties characterize the evolution of entities from a particular point of view. This approach allows one to manipulate time information just like structural information. In our example, we have a number

characterizing the number of lines of code in a Class and we have a number characterizing the evolution of lines of code in a ClassHistory.

The drawback of such properties resides in compressing large quantity of data into a limited set of properties. Yet, we used history properties and showed their usefulness in different evolution analyses: characterize how changes appear in the system[9]; use time to improve the detection of design flaws [16]; visualize the evolution of class hierarchies [11]; detect patterns of change [10].

In Figure 3 we showed an example of how to reach our Smalltalk implementation for the navigation from PackageHistory to ClassHistory. In the same way we can generate the code for another language (*e.g.,* Java). Also, in the example we just talked about Package and Class, but in a similar manner we could extend the diagram for other entities as well.

For example, in Figure 4 we show an excerpt of Hismo, as implemented in our tool. The structural meta-model consists of different entities (*e.g.,* Method). These entities are wrapped by a Version correspondent (*e.g.,* MethodVersion) and the Versions are contained in a History (*e.g.,* MethodHistory). We create Versions as wrappers for StructuralEntities because in a Version we store the relationship with the History. Thus, we are able to compute properties for a particular Version in the context of the History. For example, having a version we can navigate to the previous or the next version.

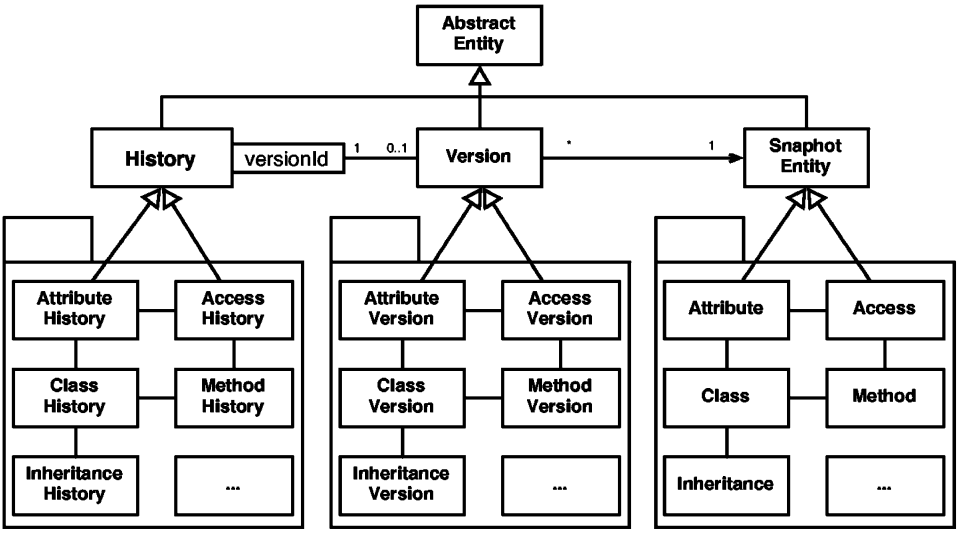

Fig. 4. An excerpt of Hismo and its relation with a source code meta-model. We did not represent all the inheritance relationships to not affect the readability of the picture.

5 Conclusions

We defined history as being a sequence of versions and we briefly presented Hismo as a history meta-model centered around the notion of history. We then argued that we can obtain Hismo starting from a structural meta-model, we specified the transformations needed, and we showed how we can reach our Smalltalk (Figure 3) implementation based on the transformations.

In the future we want to implement the automated transformations in our environment and validate the approach by using our evolution analysis tools on models generated with an industrial case tool.

References

[1] T. Ball and S. Eick. Software visualization in the large. *IEEE Computer*, pages 33–43, 1996.

[2] A. Capiluppi. Models for the evolution of os projects. In *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, pages 65–74, 2003.

[3] A. Capiluppi, P. Lago, and M. Morisio. Evolution of understandability in oss projects. In *Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR 2004)*, pages 58–66, 2004.

[4] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 77–86. ACM Press, 2003.

[5] S. Demeyer, S. Tichelaar, and S. Ducasse. FAMIX 2.1 — The FAMOOS Information Exchange Model. Technical report, University of Bern, 2001.

[6] S. Ducasse, T. Gîrba, and J.-M. Favre. Modeling software evolution by treating history as a first class entity. In *Workshop on Software Evolution Through Transformation (SETra 2004)*, pages 71–82, 2004.

[7] S. Ducasse, T. Gîrba, M. Lanza, and S. Demeyer. Moose: a collaborative and extensible reengineering environment. In *Reengineering Environments*. tba, 2004. to appear.

[8] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, pages 23–32, Sept. 2003.

[9] T. Gîrba, S. Ducasse, and M. Lanza. Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes. In *Proceedings of ICSM '04 (International Conference on Software Maintenance)*, pages 40–49. IEEE Computer Society Press, 2004.

[10] T. Gîrba, S. Ducasse, R. Marinescu, and D. Raţiu. Identifying entities that change together. In *Ninth IEEE Workshop on Empirical Studies of Software Maintenance*, 2004.

[11] T. Gîrba and M. Lanza. Visualizing and characterizing the evolution of class hierarchies. In *Fifth International Workshop on Object-Oriented Reengineering (WOOR 2004)*, 2004.

[12] A. Hassan and R. Holt. Predicting change propagation in software systems. In *Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 284–293. IEEE Computer Society Press, Sept. 2004.

[13] M. Jazayeri. On architectural stability and evolution. In *Reliable Software Technlogies-Ada-Europe 2002*, pages 13–23. Springer Verlag, 2002.

[14] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. In *Proceedings of LMO 2002 (Langages et Modèles à Objets*, pages 135–149, 2002.

[15] M. M. Lehman and L. Belady. *Program Evolution – Processes of Software Change*. London Academic Press, 1985.

[16] D. Raţiu, S. Ducasse, T. Gîrba, and R. Marinescu. Using history information to improve design flaws detection. In *Proceedings of CSMR 2004 (European Conference on Software Maintenance and Reengineering)*, pages 223–232, 2004.

[17] C. M. B. Taylor and M. Munro. Revision towers. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, pages 43–50. IEEE Computer Society, 2002.

[18] F. Van Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings of The 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, 2004. to appear.

[19] J. Wu, R. Holt, and A. Hassan. Exploring software evolution using spectrographs. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 80–89. IEEE Computer Society Press, Nov. 2004.

[20] X. Wu, A. Murray, M.-A. Storey, and R. Lintern. A reverse engineering approach to support software maintenance: Version control knowledge extraction. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 90–99. IEEE Computer Society Press, Nov. 2004.

[21] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *26th International Conference on Software Engineering (ICSE 2004)*, pages 563–572, 2004.