Integrating Open-Source Modeling Projects: Collaborative Modeling with Papyrus and EMF Compare

Maximilian Koegel and Philip Langer
EclipseSource Munich GmbH, Germany
{mkoegel|planger}@eclipsesource.com

Abstract. While many open-source modeling tools have traditionally been mostly the basis for building commercial tools on top, we recently observe an increasing direct adoption of those underlying open-source tools bundled together into industrial-ready modeling products. Adopters of those open-source tools thereby gain independence from respective vendors and win flexibility and the freedom to tailor and extend the open-source modeling tools to their specific needs. When tying together a number of open-source tools in order to build a modeling environment, however, adopters partly face a lack of seamless integration among the diverse open-source tools, edgy usability, and heterogeneous user guidance – aspects which traditionally have been addressed by the commercial tool providers in their final products. In this paper, we outline an example of a challenging integration and sketch a technical solution. Based on this example, we draw conclusions regarding organizational concerns that foster the successful resolution of integration issues.

1. Introduction

Open-source software has played a major role in model-driven engineering (MDE) since many years. In fact, many software components that are used under the hood of MDE tools is available under the terms of business-friendly open-source licenses. In many cases, tool vendors use open-source modeling frameworks as a basis for building their MDE products. This strategy allowed them to decrease the development costs of those base components and benefit from bug fixes or extensions introduced by others while not endangering their unique selling proposition; the value of their products lies in seamlessly integrating and shaping those open-source frameworks according to specific methodologies or needs, as well as a proprietary, consistent layer on top. Also academia is successfully basing their research prototypes on top of those open-source modeling frameworks, which not only speeds up the development of the research prototypes but also increases the visibility and transferability of their research results. As a consequence, active and diverse communities emerged around open-source modeling tools, such as the Eclipse Modeling Framework (EMF)[1], the

[1] https://eclipse.org/modeling/emf/

Graphical Modeling Framework (GMF)[2], Papyrus[3], and EMF Compare[4], to name just a few. Moreover, the fact that these tools have been used for years as a basis of different and diverse products nurtured their extensibility and flexibility.

In the last years, however, we observe an increased *direct* adoption of open-source modeling tools. Instead of using off-the-shelf MDE products that may use open-source modeling tools under the hood, more and more organizations aim to tie together their own modeling environment by using open-source modeling tools directly. Thereby, they gain independence from the respective vendors and – often even more importantly – they win flexibility and the freedom to tailor and extend the open-source modeling tools to their specific needs. This strategy opens up an opportunity for them to significantly increase innovation and to explore new ways of applying MDE in their organizations. Organizations may control the speed of evolving their modeling environments themselves and may benefit from working more closely together with academia and service providers, who maintain, customize, and extend the open-source modeling tools [1].

However, when putting together available open-source modeling tools in order to build a dedicated, complete modeling environment for an organization, adopters partly face a lack of seamless integration among the diverse open-source tools, edgy usability and heterogeneous user guidance – especially at interaction points with other open-source tools. Traditionally, these aspects have been addressed by the commercial tool providers in their final products, but with this new strategy, these issues need to be sorted out in collaboration among adopting organizations and service providers offering expertise in the used open-source projects.

In this paper, we outline challenges that may arise when tying together modeling environments from different open-source modeling tools. In particular, we therefore use an example of a challenging integration in the context of our work on a *collaborative modeling environment based on Papyrus and EMF Compare*[5]. Moreover, we sketch a technical solution for the discussed integration challenge and draw conclusions regarding general organizational concerns that foster the successful resolution of integration issues and edgy usability at the interactions with other open-source modeling tools.

---

[2] http://www.eclipse.org/modeling/gmp/
[3] https://eclipse.org/papyrus/
[4] https://www.eclipse.org/emf/compare/
[5] http://collaborative-modeling.org/

2. Background: Generic Model Versioning

When a model-based development process is applied in an industrial setting, models have to be developed and evolved in teams. This usually means that models are shared among team members who independently evolve these shared models. Parallel model changes are inevitable and hence need to be managed and eventually consolidated in order to obtain a new version of the shared models. In analogy to source code versioning, this process is often referred to as model versioning.

In the past ten years, the research community achieved significant advances in developing dedicated methods and techniques for enabling proper model versioning (cf. [2] for a survey). Theories and approaches emerged for computing the changes that have been applied to a model between one model version and another, for analyzing conflicts among the detected changes, and for merging those changes to build a new consolidated version reflecting the union of all non-conflicting changes that have originally been applied independently of another (e.g., [3,4,5]).

The model versioning community was strongly influenced by the differencing and merging techniques available for source code. However, as opposed to programmers, who interact with source code directly, modelers interact with models and usually do not see the underlying textual serialization of model. Thus, one of the basic ideas in model versioning was to raise the level of abstraction from characters and lines, as in source code versioning, to the elements a model consists of, e.g., use cases and actors in UML Use Case Diagrams. To avoid the necessity to define dedicated algorithms for detecting, representing, and merging model changes for each modeling language, such as UML or BPMN, again and again, the idea of generic model versioning gained popularity. Generic model versioning exploits the information specified in a modeling language's metamodel to obtain the knowledge on which modeling concepts exist (such as UML Use cases and Actors), which properties those modeling concepts may hold, and which relationships they may have. This knowledge in combination with a powerful reflection mechanism, as available in EMF, is enough to allow algorithms for detecting changes among model versions, determining incompatible changes, and merging changes to build a consolidated new version to operate generically.

Many of those ideas from academia, such as generic model versioning, have been successfully transferred to practice. Remarkable frameworks and tools are now available that enable, for instance, generic model versioning of all EMF-based models, independently of which modeling language they conform to. Those tools not only offer standalone model comparison and merging capabilities, but are often also shipped with dedicated operation-based model repositories, such as EMFStore, or are integrated with source code versioning tools, as for instance with EMF Compare and Git.

3. Challenging Integration: Model Versioning and Modeling Editors

While generic model versioning was a significant achievement, it soon becomes clear that the metamodel of modeling languages indeed allows to learn about the structure of a modeling language and hence enables to compute and process structural changes of models, but it does not provide the necessary information on the model editing dynamics of a modeling language (cf. [6] for more discussions on generic vs. language-specific model versioning). The model editing dynamics are implemented in modeling editors and, for instance, define typical concepts that users may add with one click, even though these concepts may consist of several connected and pre-configured modeling elements under the hood. They also define typical changes that affect several model elements at once, or even abstract from how information is expressed in the model underneath with dedicated user interface widgets, such as tables or connected lists. An example for such an abstraction is the language and body properties of UML's Opaque Actions. In the UML metamodel, they are defined to be two independent multi-valued String-typed properties. However, their editing dynamics is rather an index-based map, whereas the language value on position *n* defines language of the corresponding body value on position *n*. Thus, user-friendly modeling editors, such as the Papyrus UML editor, provide user interfaces that logically connect the values of those two properties and provide user-friendly ways of editing those two value lists in combination.

Obviously, generic model versioning systems will not be able to respect such an editing dynamics and will provide users with comparison results that are not comprehensible for users, who otherwise interact only with the potentially sophisticated user interfaces of modeling editor. In fact, users will only understand changes easily that are presented in the same way as they can perform them in the modeling editors they are familiar with. Thus, if model changes are performed in a diagramming editor, modeling versioning tools must represent those changes in a diagrammatic representation, irrespectively of whether only the diagram layout or also the underlying model has been changed by the user's interaction with the diagram. If model changes are performed in a form-based user interface, which is common for properties of model elements, model versioning tools must then represent those changes in a form-based user interface that look and behave exactly as the forms of the modeling editor, users are familiar with. Accordingly, also changes have to be merged by the model versioning tool at the same level of granularity (atomic versus grouped changes) as they usually can be performed in a modeling editor.

In summary, there is a gap between how users interact with and change models and how model versioning systems handle, represent, and merge changes based on the underlying metamodel of the respective modeling language. This gap is one of the main causes for misleading or incomprehensible representations of model changes, leads to inefficiency and frustration of users during manual merge sessions, and often even causes incorrect merge decisions [7].

4. Bridging the Gap: Seamless Model Versioning

In EMF Compare, this issue is well known and recently several of the negative effects of generic model versioning have been mitigated with dedicated customizations. For instance, EMF Compare integrates with the EMF.Edit framework, which is used by modeling editor developers to customize the labels and icons that are being depicted for modeling concepts and their features. As a result, the look and feel of model elements and their features in comparison and merge viewers of EMF Compare is closer to the look and feel of the models in the modeling editor, as they both share the same underlying definitions concerning labels and icons.

Another significant improvement is the integration with GMF[6], a framework that is frequently used to implement diagramming editors. With GMF, the diagram layout information of a model – also called notation – is again represented as an EMF-based model. Thus, EMF Compare also compares this notation model and hence can detect layout changes. If layout changes of GMF-based diagrams are detected, the GMF runtime is adopted to render the diagram differences, which significantly eases the understandability of layout changes. Nevertheless, if model changes are detected that have originally been performed by changing the diagram, for instance, a UML Component has been moved into another UML Package, EMF Compare still represents the model changes in tree-based viewers, although they may have been induced by interactions with the diagram. This gap remains open currently in EMF Compare, because it is hard to determine generically whether a model change has been applied in the model directly or has been induced by a change in the diagram.

To enable resolving issues stemming from a modeling language's editing dynamics, such as the above-mentioned implicit connection between the language and body properties of UML's Opaque Actions, EMF Compare offers several extension points that enable injecting custom behavior in the different phases of the model versioning process. Thus, some editing dynamics are currently addressed using dedicated, handwritten code that customizes a particular generic behavior in EMF Compare. Although this is a good start, this certainly is not a scalable and sustainable solution.

[6] https://wiki.eclipse.org/GMF

There are numerous such editing dynamics in a modeling language and addressing all of them manually in EMF Compare is cumbersome and would lead to inconsistencies between how the editing dynamics is implemented in the editor and in EMF Compare, especially when considering that this editing dynamics may evolve over time. Moreover, the editing dynamics may significantly differ from modeling language to modeling language, or even may be different depending on how a modeling language is applied in different organizations or projects. Domain-specific modeling languages inherently have their domain-specific editing dynamics, which stems from the purpose and pragmatics of the respective language. If now UML Profiles are employed to define domain-specific modeling languages, as it is increasingly the case with Papyrus UML, even the application of a UML Profile may have a tremendous impact on how a model is edited using the modeling editor. Even though the necessary knowledge on the editing dynamics is usually already implemented in the modeling editor, this knowledge is currently not available explicitly in a reusable manner and has to be redundantly re-implemented in the model versioning system to mitigate the gap between modeling editors and model versioning systems.

We argue that the model editing dynamics need to be explicitly defined and should be available as shared knowledge among modeling editors and model versioning systems, and potentially other components, in order to work more closely together and to provide a more consistent user experience. We suggest that, comparable to the EMF.Edit framework, a shared API is required. We collected an initial list of requirements that this API should fulfill: in particular, the API should enable users to (i) define the modeling concepts that can be added or deleted – potentially consisting of multiple connected and pre-configured model elements, (ii) specify logical connections between properties of model elements (e.g., the language and body attribute of UML Opaque Actions), (iii) provide the user interfaces that should be used to change and represent values of a certain property or a set of properties, as well as (iv) implement composite operations [8] that can be applied on models and hence should also be grouped when representing changes to the user.

Such a shared API, which aims to be used for model editing, as well as for model versioning, has to be designed and integrated into the respective tools in close collaboration with the developers of both the modeling editor and the model versioning system. Without mutual agreement and joint forces, it will not be possible to build such an important basis for two distinct projects. In the case of Papyrus UML and EMF Compare, we fortunately are in such a fruitful collaboration[7] with the developers of Papyrus UML, which is crucial for the success of accomplishing a seamless user experience when developing models collaboratively with EGit.

[7] http://www.collaborative-modeling.org

5. Conclusions

Based on the example integration discussed in the previous sections, we illustrate that building a modeling environment by tying together different modeling frameworks often requires collaboration with service providers across the borders of single open-source projects. Integration issues may sometimes be resolvable from within one project, as mentioned above, e.g., by manually developing UML-specific customization code in EMF Compare. In many scenarios, however, such an approach does not yield satisfactory solutions on the long run: edgy user experience, inconsistent user guidance, as well as high maintenance costs of the integration code, are often inevitable. A seamless and efficiently maintainable solution often requires coordinated evolution of multiple projects, such as novel shared APIs or new common generic components. In return, such solutions usually produce value not only for the modeling environment to be built but also for the involved projects, as well as other projects that may benefit from the new shared components or APIs in the future.

The necessity for changes among multiple open-source projects also suggests that a designated role should be established in a project consortium that aims at a particular integration. This role should drive and design the technical development of the integration work, while considering and moderating the specific characteristics of the involved projects. Therefore, this role obviously is required to have both a solid knowledge of all projects that need to be integrated, as well as a keen understanding of the requirements of integration, which makes close interaction with the organizations' stakeholders a key ingredient for success. Finally, a sustainable open-source model should be guaranteed to ensure a long-term availability of the software; thus, the artifacts resulting from the integration effort should – wherever appropriate – be contributed back to the respective open-source projects. Thus, selecting service providers that are well-connected and experienced in the respective open-source communities is of uttermost importance.

Another successful strategy for organizations that aim at employing a customized open-source modeling environment is to establish or join a common initiative, such as the Papyrus Industrial Consortium, in order to cooperatively build with other organizations a common basis for the intended modeling environment in a coordinated manner. This way, the common basis can not only be more efficiently evolved but participating organizations can also ensure that the common basis will satisfy their needs in terms of features, customizability, and flexibility.

References

[1] F. Bordeleau and E. Fiallos. Model-Based Engineering: A New Era Based on Papyrus and Open Source Tooling. In the First Workshop on Open Source Software for Model Driven Engineering (OSS4MDE'14), co-located with MODELS, 2014.

[2] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer: An Introduction to Model Versioning. In Formal Methods for Model-Driven Engineering, Springer, LNCS 7320 (2012), ISSN: 0302-9743; 336 – 398, 2012.

[3] G. Taentzer, C. Ermel, P. Langer, M. Wimmer: A fundamental approach to model versioning based on graph modifications: from theory to implementation. In the Journal of Software and Systems Modeling, 13 (1), 239 – 272, 2014.

[4] M. Koegel: Operation-based Model Evolution, Dr. Hut Verlag, ISBN: 3843900817, 2011.

[5] B. Westfechtel: Merging EMF models. In the Journal of Software and Systems Modeling, 13 (2), 757 – 788, 2014.

[6] P. Langer, M. Wimmer, J. Gray, G. Kappel, A. Vallecillo: Language-Specific Model Versioning Based on Signifiers. In the Journal of Object Technology, 11(3), 4: 1 – 34, doi:10.5381/jot.2012.11.3.a4, 2012.

[7] K. Wieland, P. Langer, M. Seidl, M. Wimmer, G. Kappel: Turning Conflicts into Collaboration - Concurrent Modeling in the Early Phases of Software Development. In Computer Supported Cooperative Work: The Journal of Collaborative Computing, 22 (2013), 2-3; 181 - 240.

[8] P. Langer, M. Wimmer, P. Kaufmann, M. Herrmannsdoerfer, M. Seidl, K. Wieland, G. Kappel: A Posteriori Operation Detection in Evolving Software Models. In the Journal of Systems and Software, 86 (2013), 2; 551 – 566.