Scalable Model Edition, Query and Version Control Through Embedded Database Persistence

Xabier De Carlos[1], Goiuria Sagardui[2], and Salvador Trujillo[1]

[1] IK4-Ikerlan Research Center, P^{o} .J.M^{o}. Arizmendiarrieta, 2 20500 Arrasate, Spain
{xdecarlos,strujillo}@ikerlan.es
[2] Mondragon Unibertsitatea, Goiru 2, 20500 Arrasate, Spain
gsagardui@mondragon.edu

Abstract. Persisting and managing models larger than a few tens of megabytes using XMI introduces a significant time and memory footprint overhead to MDE workflows. Several approaches provide alternative persistence mechanisms based on databases that can be integrated with EMF. However, to the best of our knowledge there is less coverage of approaches on models persistence which include model querying, versioning and edition capabilities leveraging persistence capabilities.
In this poster we present an approach that provides model persistence based on embedded databases. This approach aims to provide scalability through model edition, querying and versioning mechanisms that leverage database capabilities.

Keywords: Model Driven Development, Large-Scale Models, Run-Time Query Translation, Version Control, Model Edition

1 Background and Motivation

XML Metadata Interchange (XMI) is an XML-based model interchange format standardised by the OMG, and the default model persistence format in the widely-used Eclipse Modelling Framework (EMF). Being an XML-based format, models stored in XMI need to be fully loaded in memory before they can be queried or modified. As models grow in size, this can have a significant impact both on the overall time needed to execute a query on a stored model, and on the memory footprint of the host application. As such, there is a need for alternative model persistence mechanisms that scale better in terms of speed and memory footprint.

Approaches like Morsa[1], Neo4EMF [2], MongoEMF [3] or EMF Fragments [4] have been proposed to provide alternative persistence mechanisms. These approaches also provide integration with EMF and support querying models. Integration is provided by implementing the *Resource* interface of EMF. Persistence approaches provide persistence-specific query languages that take advantage of database capabilities. However, modelling engineers use query languages closer to model-level, such as the Object Constraint Language (OCL) or the Epsilon Object Language (EOL) to query models.

Additionally, models are used within collaborative modelling environments where different versions of models are managed and shared between developers. Version Control Systems (VCSs) provide support for version management, and different approaches have been proposed for version control of models. Some approaches such as ModelCVS [5], AMOR [6], CDO [7] or EMFStore [8] provide model-specific VCSs that allow performing version operations (i.e. commit, update, locking, etc.) at model-level. However, most of these approaches (ModelCVS, AMOR and EMFStore) are based on a central repository where XMI models are stored. Consequently as XMI has scalability problems, these approaches do not scale well. CDO also stores models in a centralised repository, but at model-level. But it also has scalability problems, partially due to version control [9].

Other approaches like EMF Compare [10], EMF Diff/Merge [11] or Epsilon Comparison Language (ECL) and Epsilon Merging Language (EML) [12] provide model-level differencing and merging. However we have not identified differencing/merging solutions that leverage database persistence capabilities.

2 Contribution

While some approaches focus on providing persistence for large-scale models (i.e. Morsa, Neo4EMF, etc.), other approaches focus on model-specific VCSs (i.e. EMFStore, ModelCVS, etc.). However, to the best of our knowledge, a database persistence mechanism to be integrated with a distributed VCS (i.e. GIT) has not been explored yet, and we believe that this mechanism can be a good alternative for large-scale model versioning. Providing a persistence mechanism also implies to implement model query and edition mechanisms. Consequently, as we are looking for a scalable solution to be used for large-scale model persistence, version control, edition and querying mechanisms should be scalable.

We propose a persistence mechanism to be integrated within VCSs. Scalability on version control, and model-level edition and querying is supported through mechanisms that leverage capabilities of the database (i.e. partial load and modifications). The approach is divided in four parts: persistence (core of the approach) and edition, query and version control layers that are created around the persistence and leveraging its capabilities.

2.1 Persistence

While most of the existing approaches perform model versioning through centralised repositories, we propose database based persistence of models to be integrated within distributed VCSs. Persistence is based on single-file embedded databases and it provides different benefits: (i) drop-in replacement for XMI files facilitating integration in a VCS and in modelling tools; (ii) partially load and modify models; and (iii) model comparison at different levels (file, database or model levels).

Work done and open issues. We have implemented a mechanism that persists models in embedded and single-file relational databases. We have used H2 as the database back-end. The designed data schema is metamodel-agnostic and it supports persistence of models that conform to arbitrary metamodels. We have validated the solution with positive results after performing preliminary evaluation. Open issues related to this task are: (i) to perform a more complete evaluation comparing performance with other database configurations (indexes, table views, cache, etc.) and also with other existing solutions; (ii) to improve the database schema; (iii) to analyse alternative embedded database back-ends; and (iv) to analyse how to persist more complex models (mega-models, models with decorator models, etc.)

2.2 Model-Level Query Layer

While model-level query languages such as OCL or EOL are commonly used by engineers, persistence-level query languages leverage capabilities of the used persistence mechanism. To solve this issue, Model-Level Query Layer translates at run-time model-level EOL queries to persistence-level (SQL). Performing the translation at run-time provides transparency to the modelling engineers that can specify and execute model-level queries without worrying on the used persistence mechanism. In this way, the model-level queries are automatically translated to a persistence-specific query language, leveraging persistence mechanism's capabilities.

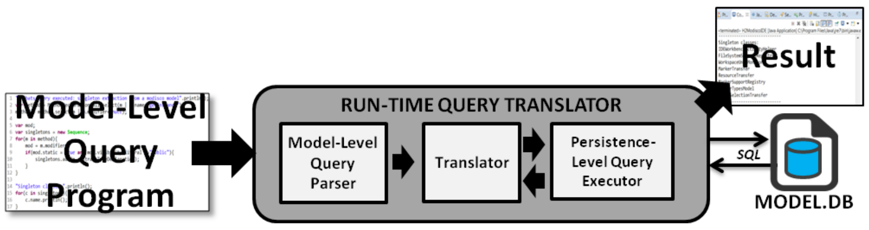
Model-Level Query Program
RUN-TIME QUERY TRANSLATOR
Model-Level Query Parser
Translator
Persistence-Level Query Executor
Result
SQL
MODEL.DB

Fig. 1: Model-Level Query Layer, run-time translation from model-level to persistence-level.

As Figure 1 illustrates, the run-time translation process parses query expressions within the query program. Each query expression is translated at run-time. Translated query is executed over the database (the model) only when the information is required (by other queries or by the user).

Work done and open issues. We have implemented a first prototype that translates EOL queries to SQL queries and also executes over the persistence mechanism. Additionally, we have analysed the time required by the approach for performing the query translation, concluding that the translation does not imply a significant time overload. The prototype supports all the EOL sentences that obtain information from models. Model modification sentences are not supported, but we plan to support them in a future prototype. Moreover, the prototype supports only EOL at model-level and SQL at persistence level. For future work we plan to add extensibility, providing more query languages at both sides.

2.3 VCS Integration Layer

The approach provides a database persistence mechanism to be used in distributed mainstream VCSs. While source code differencing/merging is natively supported by VCSs (i.e. GIT or Subversion), they only support versioning models at file-level. Thus, the VCS Integration Layer provides support for model-specific and scalable differencing and merging.

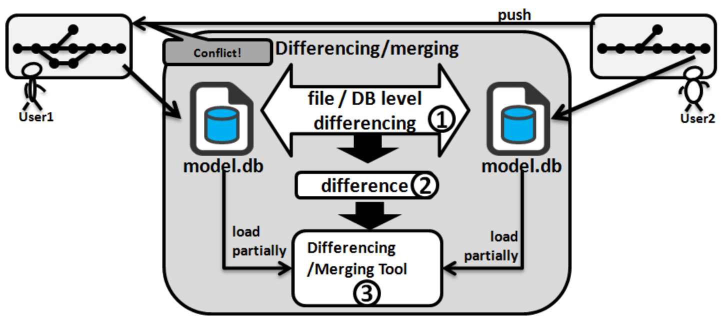

Fig. 2: VCS Integration Layer, differencing and merging for large-scale models.

In this way, we propose a differencing and merging mechanism that perform the difference and merge at different-levels. Figure 2 illustrates the proposed strategy. As it is shown on the figure, a conflict appears when *User2* pushes a new model version to *User1*. Consequently, a difference or merge should be performed. Both versions (*model.db*) are a large-scale model persisted within an embedded single-file database. Being so, versions are compared first at file or database level (*step 1*) and then the differences are extracted (*step 2*). Finally, differences are used to load models partially on the differencing/merging (*step 3*). These partial models only contain the information required to resolve the conflict.

Work done and open issues. We have described directions that we will follow to provide a scalable model differencing/merging solution which leverage capabilities of the proposed persistence mechanism. However, the approach is not yet implemented. Moreover we have to analyse feasibility to extend existing differencing/merging tools (i.e. EMF Compare) with this strategy.

2.4 Tool Integration Layer

Providing an alternative persistence mechanism requires the support of integration with modelling tools, to be able to edit persisted models. The Tool Integration Layer aims to provide integration of the persistence mechanism with EMF. This layer will implement the Resource interface of EMF to integrate the persistence mechanism with EMF. Moreover, scalable solutions for model edition such as partial load or load on demand will be provided.

Work done and open issues. We have described directions to be followed to integrate persistence within EMF-based modelling tools and they will be implemented in the future.

References

1. Espinazo Pagán, J., Sánchez Cuadrado, J., García Molina, J.: Morsa: A Scalable Approach for Persisting and Accessing Large Models. In Whittle, J., Clark, T., Kahne, T., eds.: Model Driven Engineering Languages and Systems. Volume 6981 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 77–92
2. Benelallam, A., Gómez, A., Sunyé, G., Tisi, M., Launay, D.: Neo4EMF, a Scalable Persistence Layer for EMF Models. In: ECMFA- European conference on Modeling Foundations and applications, York, UK, Royaume-Uni, Springer (July 2014)
3. Bryan Hunt: Mongo EMF. `https://github.com/BryanHunt/mongo-emf/wiki` Accessed March 17, 2014.
4. Scheidgen, M.: Reference Representation Techniques for Large Models. In: Proceedings of the Workshop on Scalability in Model Driven Engineering. BigMDE '13, New York, NY, USA, ACM (2013) 5:1–5:9
5. Johannes Kepler University of Linz and Vienna University of Technology: The ModelCVS Project. `http://www.modelcvs.org/versioning/index.html` Accessed June 23, 2014.
6. Johannes Kepler University of Linz and Vienna University of Technology: The AMOR Project. `http://www.modelversioning.org/index.php?option=com_content&view=article&id=46&Itemid=54` Accessed June 23, 2014.
7. Eike Stepper: CDO Model Repository Overview. `http://www.eclipse.org/cdo/documentation/` Accessed March 17, 2014.
8. EMFStore: What is EMFStore and Why Should I Use It? `https://eclipse.org/emfstore/` Accessed June 21, 2014.
9. Pagán, J.E., Cuadrado, J.S., Molina, J.G.: A Repository for Scalable Model Management. Software & Systems Modeling (2013) 1–21
10. EMFCompare: Compare and Merge Your EMF Models. `http://www.eclipse.org/emf/compare/` Accessed June 06, 2014.
11. Diff/Merge, E.: a Diff/merge Component For Models. `http://eclipse.org/diffmerge/` Accessed June 06, 2014.
12. Kolovos, D., Rose, L., Paige, R., Polack, F.A.: The Epsilon Book. Structure **178** (2010)