Chronos
Scalable Model Versioning, Querying & Persistence

Martin Haeusler

University of Innsbruck, Department of Computer Science
martin.haeusler@uibk.ac.at

Abstract. The discipline of model engineering has matured considerably over the last years and is applied in a wide array of domains, ranging from embedded software development to IT landscape documentation. A variety of different model repositories attempt to meet the requirements that emerge when working on models, most notably efficient persistence, versioning and query capabilities. However, existing tools scale poorly when faced with large models used in practice, in particular in scenarios where (semi-)automated element generation pushes model sizes to hundred thousand individual elements and beyond. In this paper, we present our research agenda for Chronos[1], an effort which aims to provide a solution to this problem in the form of a novel model repository for EMF Ecore models.

1 Problem Description

In recent years, the importance and popularity of model engineering has increased considerably, both in academic as well as in industrial settings [3]. Models are being created in a variety of languages and frameworks, with EMF Ecore [12] and UML [8] as the most prominent examples. As models grow larger and get more sophisticated, the demand for tools that support collaboration on model editing also increases. Such tools are often referred to as *model repositories*. Their core features encompass storing, versioning and querying model data. Pierantonio et al. have recently assembled and published a list of existing model repositories [2]. The list contains open-source pojects like Eclipse Common Data Objects (CDO)[2] and EMFStore[3] as well as commercial systems such as MagicDraw Teamwork Server[4]. In our experience, none of these tools scales well with models of large sizes, most notably due to performance issues and/or lack of features (e.g. versioning capabilities and expressiveness of queries). An important factor for scalability is the employed persistence technology, and all aforementioned model repositories use one of two technologies:

[1] This work was partially funded by the research project “QE LaB - Living Models for Open Systems” (FFG 822740) and “txtureSA” (FWF-Project P 29022).
[2] https://eclipse.org/cdo/
[3] http://www.eclipse.org/emfstore/
[4] http://www.nomagic.com/products/teamwork-server.html

- **File / XML-Based**
  Model repositories in this category store serialized forms of models in files (typically XML, or XMI [9] in the case of Ecore). For versioning, this implies the existence of one file per version. EMFStore and MagicDraw Teamwork server are representatives of this category.
- **Relational / SQL-Based**
  Tools based on relational technology aim to perform a mapping from the object representation into an equivalent relational representation. This process is referred to as object-relational mapping, or O/R-mapping. The relational information is then stored in a traditional database system. Versioning introduces additional entries in the relational tables to store the state of an element at a given version. Eclipse CDO makes use of this approach.

Both techniques suffer from severe drawbacks and are not optimal for storage of model data. XML-based systems struggle with per-element versioning and the absence of indexing structures for querying, as well as having difficulties in providing lazy loading capabilities, because any given XML file must usually be processed in its entirety before individual elements can be extracted. Such processing (i.e. serialization or deserialization) is also very costly with respect to CPU power, RAM and runtime. Relational backends (e.g. SQL databases) provide indexing structures and per-element loading while also working on a user-defined schema which allows for versioning, provided that the repository takes care of this aspect on its own. The major problem with relational backends is the expensive Object–Relational Mapping (O/R mapping) process that converts objects into table entries and vice versa. O/R mapping increases considerably in complexity when objects with many connections to other objects have to be processed, which is a very common use case for modeling. Typical O/R mapping algorithms are often implemented in a recursive fashion[5] which can lead to call stack overflows on sufficiently large models. Converting the relational representation back into model element objects requires at least one SQL *JOIN* operation per connection. These operations have inherent quadratic complexity and therefore scale poorly.

We aim to improve the situation by adressing the hot spots of resource consumption: object (de-)serialization, storage of versioned data and queries on the persisted information. We do so by applying concepts from the NoSQL area and combining them with suitable mapping strategies and a novel approach to storage of versioned data.

The remainder of this document is structured as follows. Section 2 provides an overview of the related work. In Section 3 we present an outline of our solution. We present the expected contributions of the thesis in Section 4 and outline a plan for evaluation and validation of our approach in Section 5. Finally, Section 6 provides details on the current state of the project and Section 7 concludes the paper with a summary.

[5] As for example in Hibernate: `http://hibernate.org/`

2 Related Work

Having realized the shortcomings of the existing storage solutions for large models as described in Section 1, Gómez et al. proposed and implemented alternatives for EMF based on NoSQL technology, using graphs [1] and later also using key-value stores [5]. Their results clearly demonstrate that storing model data in a graph or key-value format is not only feasible, but also performs a lot better than traditional relational storage mechanisms.

In 2014, Felber et al. proposed a set of algorithms that enable versioning on a key-value store [4]. However, the work of Felber focuses exclusively on versioning of non-connected data, while Gómez considered graph-based storage without versioning aspects. The idea of graph-based model persistence in a versioned key-value store backend forms the foundation of our own work.

As the resulting artifact of this PhD thesis is going to be a model repository, all the tools listed by Pierantonio et al. [2] are considered as related work. There are two major conceptual differences between our solution and existing repositories. The first difference is the level of abstraction in the technology. All relevant existing tools are either built on top of a pre-existing database or file format. We are going to develop and provide the entire data management stack, from the file format to the database and transaction management to model querying. The second difference is that our approach takes the latest advances in NoSQL research into account, which opens many possibilities, in particular with respect to model queries and performance.

3 Proposed Solution

We propose a model repository that is capable of handling tightly coupled, large-scale models with 100000 individual elements and beyond. The features will include a rich query API and full per-element versioning support, as well as important collaboration features such as lightweight branching and conflict detection.

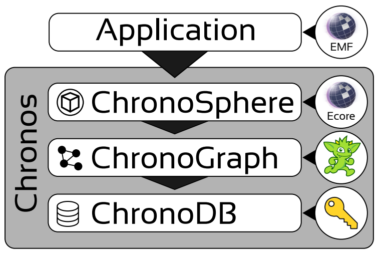

Fig. 1. The Chronos Data Management Stack

This project is named *Chronos*, and consists of three main parts (collectively called *Chronos Components*). These components build on top of each other:

- **ChronoDB** is the storage backend, and the lowest layer in the Chronos project. It is a key-value store with built-in versioning capabilities, to which we refer to as a *Temporal Key-Value Store*. ChronoDB is intended primarily as a lightweight storage backend for embedding into an application that writes to the local hard drive. In the repository, it is responsible for storage, versioning and branching. We implement these features on the lowest level because the key-value format is conceptually simple compared to an object graph, which reduces the complexity of the versioning problem.
- **ChronoGraph** is an implementation of the Apache Tinkerpop[6] graph computing API, mapping graph structures onto the key-value schema provided by ChronoDB. Several other libraries, including Titan DB[7], have demonstrated that it is possible and feasible to implement a graph database based on a key-value store. ChronoGraph provides the low-level query API and a standardized storage format to the repository.
- **ChronoSphere** is an EMF Ecore model repository. Built on top of ChronoGraph, it maps incoming model data onto a standardized graph structure. By leveraging the capabilities of the TinkerPop graph query language *Gremlin*, it will provide expressive model-level queries to programmers, as well as the versioning and branching features provided by ChronoDB.

In combination, these three components form a new type of model repository that implements the full data management stack. Aside from this top-level goal, due to the modular design, these three components can also be used individually in other projects: ChronoDB is a general-purpose key-value store with versioning support, and ChronoGraph implements the Apache TinkerPop API, the *de-facto* standard API for graph databases, enabling it to act as drop-in replacement for other implementations.

4 Expected Contributions

The thesis will provide the following contributions to the theory of temporal data stores and model repositories:

4.1 Contributions to Theory

A formal model for a Temporal Key-Value Store: This formalization is based on an infinite two-dimensional matrix structure that defines and exclusively relies on history–preserving append–only operations. It provides the formal semantics which guide the implementation of the

[6] `http://tinkerpop.incubator.apache.org/`
[7] `http://thinkaurelius.github.io/titan/`

prototype.

A novel query framework: By utilizing the features offered by the NoSQL graph database, ChronoSphere will provide an entirely new approach to model queries. It will allow developers to write queries at model level in an internal, Java-embedded domain-specific language that can be checked by the compiler, providing additional compile-time recognition of e.g. type system errors or spelling mistakes in comparison to string-based alternatives, such as OCL [10] which can only be checked at run-time. This language makes use of index structures that are updated automatically when the model is changed. These indices are aware of the versioning aspects, allowing equally fast queries on any version of the model. Lazy evaluation will be the default mode of operation, enabling the execution of queries without prior need for full resolution of a model version.

4.2 Contributions to Practice

Alongside and based on the contributions to theory, the thesis is expected to make a number of contributions to practice. We categorize them by software artefact.

Graph Database and Key–Value store: To the best of our knowledge, ChronoGraph will be the first implementation of the Apache Tinkerpop API that offers full *versioning support* which is provided by the temporal key–value store in ChronoDB. Lightweight *branching*, as seen in popular version control systems such as Git or SVN, is also part of the versioning engine. ChronoGraph will be the first graph database with full *ACID* transaction support with the highest possible isolation level ("serializable" [7]). By taking advantage of versioning, ChronoDB and ChronoGraph support *long–running transactions* without sacrificing throughput of concurrent short-lived transactions, which is of particular importance in modeling scenarios that involve model analysis. Both ChronoDB and ChronoGraph offer *timestamp–agnostic queries* that can be executed on any model version in time without modifications by injecting the desired timestamp from the transaction metadata. This can be used in a variety of ways, e.g. for after–the–fact collection of time series data in the history, or for comparing the result of a model query at two different points in time. *Temporal indices* allow for equal query performance on any model version, while *temporal conflict detection* protects users from history corruption, data loss and other anomalies.

Model Repository: The ACID nature of ChronoGraph transactions allows ChronoSphere clients to work on *consistent views* on the EMF Ecore model data, which is important e.g. for analysis and refactoring tasks. The repository is designed to handle hundred thousands of individual elements and beyond. This is achieved by *lazy loading* of EObjects and their *automatic unloading* in case they are no longer needed by the application. In order to support the insertion of large models into the repository, ChronoSphere provides batch–based *incremental commits* that are internally merged into a single model version when the last batch is written. ChronoSphere aims for the best possible *EMF ecosystem integration* and to provide EObjects that are compatible with popular Ecore–based frameworks, such as EMF Compare.

5 Plan for Evaluation and Validation

The research methodology applied for this thesis follows the principles of *Design Science* as defined by Peffers [11] and Hevner [6]. The system will be evaluated using a variety of techniques:

- **Prototype Implementation:** The implementation of the prototype serves as the proof–of–concept for the theoretical foundations and is the main design artefact. An extensive automated test suite developed alongside the prototype will assert that its functionality behaves as intended and in particular properly implements the theoretical foundations defined in the thesis.
- **Performance Measurements:** Measuring the performance (e.g. CPU and RAM usage) of the prototype implementation and comparing it to other model repositories will provide the necessary data for a discussion on the factual scalability of the model repository.
- **Industrial Case Study:** Several Chronos components are already being used in an industrial setting as the primary storage backend for the IT Landscape Documentation tool *Txture*[8] [13]. The deployment of Txture with Chronos components at industrial research partners allows to conduct case studies in real world scenarios beyond laboratory conditions.

6 Current Status

As of July 2016, large parts of Chronos have already been implemented. The core of ChronoDB is complete[9], and a paper titled *Scalable Versioning for Key-Value Stores* has been accepted by and will be published at the 5th *International Conference on Data Management Technologies and Applications (DATA 2016)*. This paper covers the theoretical foundations and implementation aspects of ChronoDB. The evaluation of the core components of ChronoGraph as a backend for Txture is currently ongoing and shows very promising early results with respect to performance. A publication on this topic is the next step. The implementation of ChronoSphere is currently work-in-progress. We plan a publication on this topic at the MODELS Conference 2017. For the final publication, we aim for a journal paper in 2018 that summarizes our findings and combines them with the results of an industrial case study. We aim for the conclusion and publication of the PhD thesis by the end of 2018.

8 `www.txture.tools`

9 `https://github.com/MartinHaeusler/chronos/tree/master/chronodb`

7 Summary and Conclusion

In this paper, we have provided an overview over the *Chronos* project that aims to provide a scalable solution for storing, versioning and querying model data, based on NoSQL graph and key-value techniques. In contrast to existing solutions, it will not rely upon XML serialization or object-relational mappings. Instead, the entire data management stack will be implemented from scratch, with the use case of storing large models in mind. The resulting artefact will serve as a proof–of–concept and will be tested against an automated test suite and in an industrial case study. Comparative benchmarks with existing repositories are also part of the evaluation. The planned end of the PhD thesis is in 2018, with a total of three conference publications and one journal paper.

References

1. Benelallam, A., Gómez, A., et al.: Neo4EMF, a scalable persistence layer for EMF models. In: Modelling Foundations and Applications, pp. 230–241. Springer (2014)
2. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Collaborative Repositories in Model-Driven Engineering. IEEE Software 32(3), 28–34 (May 2015)
3. Di Ruscio, D., De Lara, J., Pierantonio, A.: Proceedings of the 3rd Workshop on Extreme Modeling at MoDELS 2014. CEUR, Valencia, Spain (2014), `http://ceur-ws.org/Vol-1239/`
4. Felber, P., Pasin, M., et al.: On the Support of Versioning in Distributed Key-Value Stores. In: 33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014. pp. 95–104 (2014)
5. Gómez, A., Tisi, M., Sunyé, G., Cabot, J.: Map-Based Transparent Persistence for Very Large Models. Fundamental Approaches to Software Engineering 9033, 19–34 (2015)
6. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information System Research. MIS Quaterly 28(1), 75–105 (2004)
7. ISO: SQL Standard 2011 (ISO/IEC 9075:2011) (2011)
8. Object Management Group: UML 2.3 Superstructure (2010), `http://www.omg.org/spec/UML/2.3`
9. OMG: XML Metadata Interchange (XMI). OMG (2007), `http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI`
10. (OMG), O.M.G.: Object constraint language (ocl). version 2.3.1 (2012), `http://www.omg.org/spec/OCL/2.3.1/`
11. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. Journal of management information systems 24(3), 45–77 (2007)
12. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)
13. Trojer, T., Farwick, M., Häusler, M., Breu, R.: Living Models of IT Architectures: Challenges and Solutions. Software, Services and Systems 8950, 458–474 (2015)