Evaluating Embedded Relational Databases for Large Model Persistence and Query

Xabier De Carlos[1], Goiuria Sagardui[2], Salvador Trujillo[1], Alain Perkaz[1], Mikel Cañizo[1], and Aitziber Iglesias[1]

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

Abstract. Large models are increasingly used in Model Driven Development. Different studies have proved that XMI (default persistence in Eclipse Modelling Framework) has some limitations when operating with large models. To overcome them, recent approaches have used databases for the persistence of models. EDBM (Embedded DataBase for Models) is an approach for persisting models in an embedded relational database, providing scalable querying mechanism by runtime translation of model-level queries to SQL. In this paper, we present an evaluation of EDBM in terms of scalability with existing approaches. GraBaTs 2009 case study (models from 8.8MB to 646MB) is used for evaluation. EDBM is 70% faster than the compared approaches to persist XMI GraBats models into databases and executes the GraBats query faster, as well as having a low memory usage. These results indicate that an embedded relational database, combined with an scalable query mechanism provides a promising alternative for persisting and querying large models.

Keywords: Model-Driven Development, Large-Scale Models, Persistence, Query, Runtime Translation, Evaluation

1 Introduction

Automatizing and optimizing development processes is crucial to reduce development efforts and time to market of industrial projects, which are the main drivers of competitiveness. Model Driven Development (MDD) promises improvements in the development process through an intensive use of abstractions, specified by models. Models are considered first class entities during the development process, so that engineers may use them for different purposes such as code and documentation generation. Operating with models requires querying, editing and transforming them.

Eclipse Modelling Framework (EMF) is a mature and a widely used modelling framework provided within the Eclipse IDE, and XML Metadata Interchange (XMI) is the default mechanism to persist models. However, XMI entails memory and execution problems as the size of the model increases [1,2]. Thus, in industrial domains (e.g. windpower or railway) where systems can comprise a large number of elements such as sensors, actuators and control units, using XMI is not an option. Consequently, effectively supporting such domains requires using additional persistence mechanisms. Recent approaches opt for the use of databases to persist and operate with models, for example Morsa [2] or Neo4Emf[1]. In previous works [3][4], we have presented Embedded DataBase for Models (EDBM), an alternative persistence mechanism for EMF models based on an embedded relational database. Model operations are also provided through an scalable solution to query them based on a runtime translation.

In this paper, we briefly introduce EDBM and evaluate it comparing with other persistence mechanisms. While technical description of our approach has been previously presented [4], this paper contributes with a detailed evaluation which validates its usefulness and scalability. The approach is evaluated using models of different sizes (from 8.8MB, containing 14 Java classes and 70447 model elements, to 646MB, containing 5984 Java classes and 4961779 model elements) extracted from the GraBaTs 2009 case study [5]. The experiments show that EDBM is able to query persisted models maintaining low memory footprint but also taking a reasonable execution time. We compare results (storage size, insertion time, memory usage and execution time) of our approach with XMI and other database-based persistence solutions.

The rest of the paper is organised as follows: Section 2 describes some background and motivation of this work. Section 3 reviews related work, and compares it with EDBM. Then, the approach is presented in Section 4 and it is evaluated in Section 5. This paper ends with conclusions and future work in Section 6.

2 Background and Motivation

In EMF, models are persisted by default using XMI, an XML-based information persistence format standardised by the OMG. Before operating with models, all the information persisted in the file has to be loaded in memory. Once the model is in memory, information is operated: editing, querying, generating code, executing transformations, etc. If the model is modified, information in memory has to be stored again in the XMI file. However, transferring the information from physical file to memory and vice-versa entails problems with large models [6]. To overcome these problems, most recent approaches opt to leverage database capabilities. Morsa[2], Neo4EMF[1], MongoEMF[7] or EMF Fragments[8] use databases for model persistence. Each approach provides useful mechanisms for operating with large-scale models: partial load of the information, loading the information on-demand, caching, etc.

The aforementioned approaches, fully delegate the physical storage of information in models to the underlying database management system. This motivates us to explore different alternatives for providing an efficient model persistence mechanism that could act as a drop-in replacement for XMI when working with large models. Besides persistence, scalability should also be provided when operating models (e.g. querying, editing or executing transformations). Our aim is to provide a mechanism that supports (i) leverage database capabilities but using a file-level persistence mechanism; and (ii) operating models at a meaningful level of abstraction without needing to fully load them into memory. In this sense, our solution aims to provide a scalable mechanism for persisting and querying large-scale models. EDBM uses Epsilon Object Language (EOL), a model-level language that supports querying models, but can also be used to edit and transform models.

3 Related Work

We have classified the related approaches into two different groups: (i) model persistence and (ii) model query languages.

3.1 Model Persistence

Model persistence using relational and non-relational database management systems allows to load models on-demand, overcoming the memory problems of XMI. CDO[9] provides a repository, where models can be persisted using different database management systems (NoSQL and RDBMS). It also offers other features such as multi-user access, collaboration and concurrent model access of the repository or mechanisms for querying persisted models. Besides models, other information (metamodels, history) is also persisted together within the database, to provide model version control and sharing. However, scalability seems not to be the design goal of CDO [6]. Teneo[10] uses a relational database for persistence of models. It supports mapping between EMF objects and a relational database. The database schema can be metamodel-independent or metamodel-specific. Schemas are customized through metamodel annotations. Different evaluations show that Teneo does not scale well [11].

In a similar vein, Morsa[2] provides large-scale model persistence using MongoDB, a document-based NoSQL database back-end. The approach provides on-demand loading mechanisms and cache replacement policies that can be chosen by the end-user. This allows working with large models with a low memory footprint. Neo4EMF[1] is based on a transactional property-graph NoSQL database back-end (Neo4J). The approach supports the mapping between EMF models and Neo4J graphs. Neo4EMF provides different strategies for on-demand loading. Mongo EMF[7] is based on the document-based MongoDB. This approach provides an extensible and flexible framework based on OSGi declarative services. EMF Fragments[8] is a framework for persisting model fragments, and can be used with different NoSQL back-ends such as MongoDB, HBase or with distributed file systems. Each fragment contains different model elements and relations, and the fragmentation strategy is specified by the users at the metamodel level. Then, for querying models, only required fragments are loaded in memory. The Mondo Project[12] aims to provide an scalable MDD environment, which includes persistence mechanisms based on databases. In [11], the authors present two prototypes that use a NoSQL database for persistence of models (Neo4J and OrientDB) and compares benchmark of these prototypes with benchmarks obtained from models persisted in a XMI file and in a relational database. In [13], the authors present a framework and a methodology for benchmarking persistence of models on different NoSQL stores. To the best of our knowledge, approaches based on different NoSQL back-ends (suitable for distributed databases and large quantities of data (a.k.a. Big Data)) overcome the memory problems of XMI while relational database approaches already have scalability problems. We opt for a persistence based approach on a relational and embedded database, since it facilitates integration of the persistence at the same-level of XMI (file-level) but leveraging the capabilities of a database. Additionally, we base our work on the hypothesis that using a relational embedded database with a metamodel-agnostic data schema overcomes the memory problems of XMI. On the contrary to other relational database proposals we persist a single model in the database.

3.2 Model Query Languages

Languages for querying models are useful when specifying rules that obtain elements from models (e.g. all elements satisfying a condition). Some approaches provide model-level languages closer to modelling engineers: Object Constraint Language (OCL), EMF Query[14], IncQuery[15] or EOL[16]. EOL also provides support for specifying query expressions to modify models.

Other approaches propose persistence specific and dependent languages that leverage capabilities of the persistence (optimising queries): COCL (a.k.a. CDO-OCL[3]), for models persisted using CDO; MorsaQL [2] for models persisted using Morsa; database specific languages such as SQL or Cypher. In [2], the authors describe benchmarks of different query languages (OCL, MorsaQL, EMF Query, etc.) executed over models persisted using XMI, Morsa and CDO.

There are also some proposals to generate persistence level queries from model-level queries. In [17], the authors describe an approach focused on generating MySQL code from a given OCL expressions. An approach to generate SQL queries from OCL invariants is presented in [18]. In [19], the authors describe an approach that generates views using OCL constraints, and then uses them to check the integrity of the persisted data. This approach has been implemented in OCL2SQL[4], a tool that generates SQL queries from OCL constraints. A similar approach for integrity checking is proposed in [20]. Another approach described in [21] details a method that executes queries in persistence-level (SPARQL) and the results are the input of model-level queries (OCL).

EDBM provides a solution able to query models using a model-level query language (EOL) with the efficiency of SQL to query models persisted in a database . While existing approaches translate OCL constraints into SQL queries at compile-time, our approach generates SQL queries from OCL-like expressions

[3] More info at "https://wiki.eclipse.org/CDOQuery_OCL"

[4] Read more at "http://dresden-ocl.sourceforge.net/usage/ocl22sql/"

at runtime. We have based our work on [22], which describes an approach where EOL is used to query large datasets stored on relational databases composed by one table. While in this approach naive translation provided by Epsilon Model Connectivity Layer (EMC) is used to query information persisted in a single-table database, EDBM-Query provides custom translations of SQL queries that leverage persistence capabilities when querying models persisted in an embedded database.

4 Embedded DataBase for Models (EDBM)

Embedded DataBase for Models (EDBM) is an approach that is focused on the scalable persistence of models, but also on the scalable operation of models through a querying mechanism that leverage database capabilities.

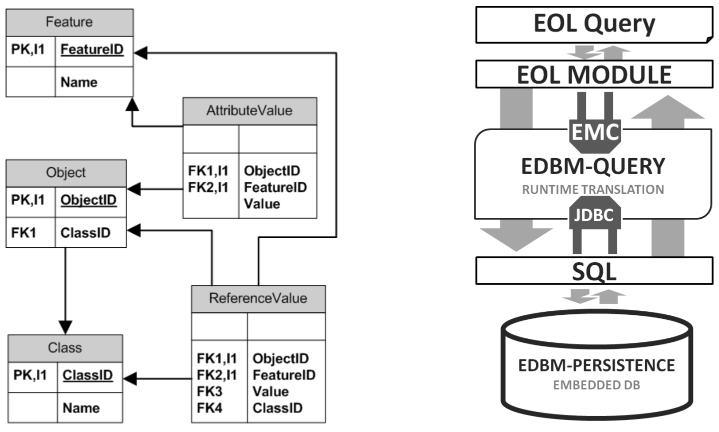
4.1 EDBM: Scalable Persistence

EDBM follows the main principle of using an embedded relational database for persisting the models. In this sense, a survey has been performed to identify systems that provide support for embedded relational databases in Java. The survey revealed that currently, SQLite[5] and H2[6] are the most mature options. After performing some preliminary tests and consulting existing benchmarks[7], we concluded that H2 has a better performance profile.

The approach uses a metamodel-agnostic schema for model persistence. This way, EDBM supports the persistence of models that conform to arbitrary metamodels and does not require to modify the schema when the domain metamodel evolves (only stored information has to be updated). Figure 1a illustrates the metamodel-agnostic schema used by EDBM and it is described next:

- **Metamodel information:** *Class* and *Feature* tables are used to store each meta-class and each structural feature existing in the domain metamodel. With these tables, the approach is able to know metamodel-related information, but using a domain-agnostic data schema. An unique id (`ClassID`, `FeatureID`) is used to identify classes and features.
- **Model elements:** `Object` table is used to persist all the elements of the model. An unique id (`ObjectID`) and the id of the meta-class (`ClassID`) is stored for each model element.
- **Structural features:** `AttributeValue` and `ReferenceValue` tables are used to store feature values of each model element. `ObjectID` and `FeatureID` are used to identify each value (`Value` column). In case of attributes, the value contains a primitive value. And in case of references, id of the referenced value and its meta-class id are stored (`Value` and `ClassID` columns).

[5] `https://sqlite.org`
[6] `http://www.h2database.com`
[7] SQLite vs. H2 comparative at "http://tinyurl.com/puxdllm"



(a) (b)

Fig. 1: (a) Database-schema of EDBM, and (b) overview of the approach.

4.2 EDBM: Scalable Model-Level Query

Using EDBM, models are persisted in an embedded relational database. In this case, leveraging SQL is more efficient than naive iteration. However, model-level queries are focused on interaction with models (and are closer to modelling engineers). Being so, use of model-level queries is more appropriate than exposing modelling engineers to the database directly with a persistence-level language (SQL).

EDBM allows to execute queries expressed in a model-level query language (EOL) and translate then in runtime to a persistence-level query language (SQL). Reasons for EOL[16] selection include: (i) it is an OCL-based language that also provides features of imperative languages such as the use of variables on the queries, query expressions for model modification and the specification of query expression chains; and (ii)it is the base of other model-specific languages like Epsilon Transformation Language (ETL) or Epsilon Generation Language (EGL) for model-to-model or model-to-text transformations, etc.

EOL and SQL are query languages that do not have direct mapping, since EOL provides constructs that are not provided by SQL (e.g. variables). In this situation, translation from EOL to SQL can be performed in two ways: at compilation-time or at runtime. Compilation-time translation requires performing a static analysis of the queries and reordering them before the translation. By contrast, if the translation is performed at runtime, query expressions are translated one by one, and they are executed against the database only when the results are required by a following expression. Therefore, we have opted to execute translation at runtime.

Figure 1b illustrates an overview of EDBM. As shown in the figure, EOL Module is the responsible for parsing and executing queries expressed using

Table 1: Execution environment of each evaluation.

	NoSQL_Eval[11]	**Morsa_Eval[2]**	**EDBM_Eval**
Processor	Intel Core I5-2300 @ 2.80 GHz	Intel Core I7-260 @ 3.70 GHz	Intel Core I7-3520M @ 2.90 GHz
Physical Memory	8GB	8GB	8GB
Operative System	Windows 7 (64-Bit)	Fedora Core 17 (64-Bit)	Windows 7 SP1 (64-Bit)
JVM	Java SE v1.6.0	OpenJDK JVM 1.7	Java SE v1.8.0
Eval. Persistence	XMI, CDO, Neo4J, OrientDB	XMI, CDO, Morsa	XMI, EDBM
Eval. Query Leng.	EOL	Plain EMF, OCL, COCL, EMFQuery, IncQuery, MorsaQL	Plain EMF, EOL
Query Repetitions	20	2	100

EOL and the EMC provides different interfaces that make possible to connect and communicate with the EOL Module. In this sense, EDBM implements such interfaces to provide the connection with the EOL queries. To be able to connect and execute queries against the database, EDBM uses the JDBC driver of H2. More details of how each artifact is used during the translation, and sample translation and execution of an EOL query had been provided in a previous paper [4]. At this point of development, EDBM supports translation of read-only EOL expressions (e.g. queries containing query expressions such as selects, collects, etc.). However, we are already working to add the support for expressions that modify models (e.g. query expressions for creating new element instances) in a future prototype of the approach.

5 Evaluation

This section presents the evaluation of persistence and query mechanisms provided by EDBM. The GraBaTs 2009 case study [5] has been selected, since it is widely used to evaluate model persistence and querying approaches.

The GraBaTs models [8] have been persisted using EDBM. These models specify source code of different Java packages and conform to the JDTAST metamodel which contains abstractions of the Java source code. The size of models ranges from 8.8MB, containing 14 java classes and 70447 model elements (set0), to 646MB, containing 5984 java classes and 4961779 model elements (set4).

We have evaluated EDBM and XMI applying the GraBaTs query to the models. GraBaTs query returns all the singleton classes of a model. The query has been expressed using EOL for EDBM, and EOL and Plain EMF for XMI.

To compare EDBM with existing approaches we have used the results of *NoSQL_Eval* [11] and *Morsa_Eval*[2], where alternative mechanisms for persisting and querying large-models are evaluated. Table 1 illustrates the execution environment of such studies. *NoSQL_Eval* evaluates XMI, CDO, Neo4J

[8] More information and resources available at "http://www.emn.fr/z-info/atlanmod/index.php/GraBaTs_2009_Case_Study"

Table 2: Time required to insert XMI model into database and storage size.

		XMI	Neo4J[11]	OrientDB[11]	CDO (H2)[11]	Morsa[2]	EDBM
Set0	Insertion time	-	12.4	19.6	11.8	-	9.2
	Storage size	8.8	29.4	53.6	26	-	36
Set1	Insertion time	-	32.5	57.1	19.2	-	24.6
	Storage size	27	85.9	134	67	-	102
Set2	Insertion time	-	499.1	590.1	778.5	-	247.2
	Storage size	271	794	1197	539	-	643
Set3	Insertion time	-	2210	2245	-	-	647.9
	Storage size	598	1750	2591	-	-	1526
Set4	Insertion time	-	2432	2397	-	-	746.7
	Storage size	646	1890	2789	-	-	1659

and OrientDB and *Morsa_Eval* evaluates XMI, CDO and Morsa. In case of the evaluation of the querying mechanisms, while *NoSQL_Eval* only evaluates the approaches using EOL query language, Morsa_Eval performs evaluations combining approaches with different query languages (Plain EMF, OCL, COCL, EMFQuery, IncQuery and MorsaQL).

5.1 EDBM-Persistence

Two measures have been used for persistence evaluation: (i) time taken to insert each model from XMI to the database (in seconds, s); and (ii) storage size of the models persisted in the databases (in Megabytes, MB). Results are the average duration of the repetitions made for each model insertion (Table 1 describes number of repetitions that have been performed on each case).

Table 2 describes the insertion time and storage size values obtained from EDBM and compares them with the values obtained at *NoSQL_Eval*[11]. *Morsa-_Eval*[2] does not contain insertion time and storage size values for Morsa, consequently these values have been omitted.

CDO fails to insert largest models (set3 and set4). Insertion time is similar for all NoSQL based options (Neo4J- and OrientDB-based prototypes) and it rounds 2400 seconds on the largest model (set4). Storage size is around 0.5 times bigger using OrientDB-based prototype than using the Neo4J-based prototype. EDBM performs insertion faster than other approaches, and specially for large models. Insertion times on the largest models (set3 and set4) is around 70% faster than in NoSQL approaches. In terms of storage size, models persisted using EDBM are between 2 and 4 times bigger than the models persisted using XMI but similar to the Neo4J-based prototype.

5.2 EDBM-Query

Query mechanisms have been evaluated using two measurements (executed 100 times each): (i) execution time to return the results of a query (in milliseconds,

		XMI+EMF	XMI+EOL	EDBM
	Time	3419	4462	182
Set0	Mem (max)	155	175	159
	Mem (avg)	122	155	158
	Time	4888	5382	232
Set1	Mem (max)	226	305	159
	Mem (avg)	200	297	156
	Time	20584	26517	1430
Set2	Mem (max)	1161	1138	299
	Mem (avg)	1051	1110	265
	Time	72052	50951	2907
Set3	Mem (max)	2398	2342	318
	Mem (avg)	2297	2289	315
	Time	112406	59722	4021
Set4	Mem (max)	2807	2508	345
	Mem (avg)	2626	2456	322

Table 3: Time (ms) and memory (MB) results obtained from GraBaTs query execution.

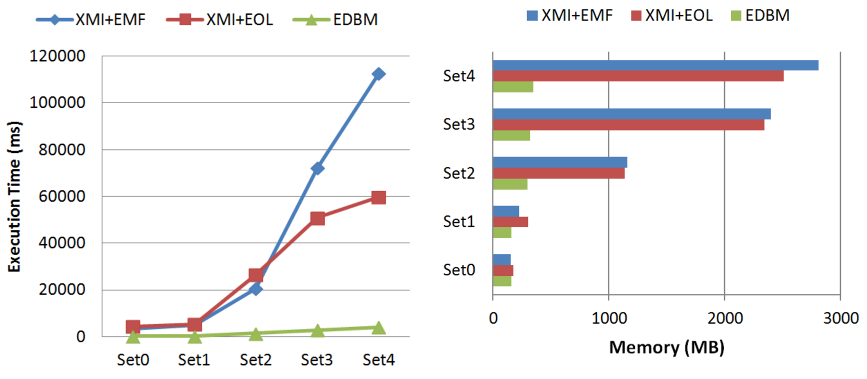
	XMI+EMF	XMI+EOL
Set0	97	74
Set1	99	89
Set2	477	782
Set3	2218	4034
Set4	1090	4339

Table 4: Execution time (ms) to query models that are previously loaded.

ms); and (ii) memory usage (in Megabytes, MB). In the case of EDBM, memory values include: memory used by the embedded database + memory used by the JVM instance. We provide results from: (a) XMI persistence with the query expressed using Plain EMF; (b) XMI persistence with the query expressed using EOL; and (c) EDBM persistence with the query expressed using EOL and runtime query translation.

Execution Time. As is shown in Figure 2a, the size of the model has a great impact over the time required to execute the GraBaTs query if XMI is used. However, results are different depending on whether Plain EMF or EOL has been used to express the query, having greater impact when using Plain EMF (querying the smallest model takes around 3.5 seconds while more than 110 seconds are required for the largest model). XMI+EOL requires around 60 seconds to execute the query over set4, half the time required by XMI+Plain EMF. Table 4 illustrates execution time (in milliseconds) for XMI if the model is previously loaded in memory. In this conditions, XMI+EMF is the fastest option. EDBM, scales better than XMI in terms of execution time: the execution time goes from 182 milliseconds for set0 to 4 seconds for set4. Also, the required execution time grows slower than using XMI. Even comparing with XMI+EMF when models are loaded in memory, the difference is small (3 seconds) and better than XMI+EOL.

Memory Usage. As shown in Figure 2b, memory usage is similar in all three options (155-175 MB) for set0. In case of set1, the maximum memory requirement increases for both XMI-based options (an increase of 71MB for XMI+EMF



(a) Execution time. (b) Memory usage.

Fig. 2: Execution time and memory usage for GraBaTs query in XMI and EDBM

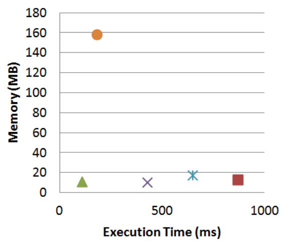
and 130MB for XMI+EOL). The model size has higher impact on memory usage in set2 when XMI is used and it is increased in both cases near 1.1GB. By contrast, memory usage impact is lower if EDBM is used and it increases 140MB. In set3, while memory usage is duplicated respect to set2 when XMI is used (using around 2.3GB), EDBM only requires 318MB. The trend is similar in set4 where XMI+EMF and XMI+EOL both require more than 2.4GB and EDBM only uses 345MB. Memory usage increase is similar on both XMI-based options, they do not scale well in terms of memory. However, EDBM does not require upfront memory loading, and consequently it scales better than XMI (increasing from 159MB on set0 to 345MB on set4).

5.3 EDBM vs. Database Persistence Approaches

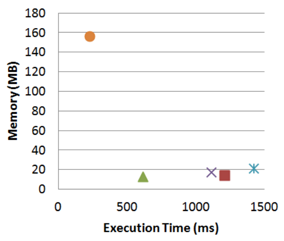
Results provided by NoSQL_Eval[11] and Morsa_Eval[2] have been used to compare EDBM performance. As Table 1 describes, different combinations of persistence and query approaches have been evaluated on each study. To facilitate comprehension only the most scalable combinations have been selected: Neo4J-based prototype +EOL and OrientDB-based prototype+EOL from NoSQL_Eval and Morsa+MorsaQL from Morsa_Eval. Since CDO also uses H2 for persistence, we have decided to include it in the comparison. We have used the results of CDO provided by Morsa_Eval.

It is important to note that the execution environment is not the same in all cases. We have not been able to reproduce the evaluation of NoSQL_Eval and Morsa_Eval. Consequently, results of evaluation obtained for such studies are used to compare existing approaches with EDBM. We use these values as a reference to analyse the trend and scalability of our approach.

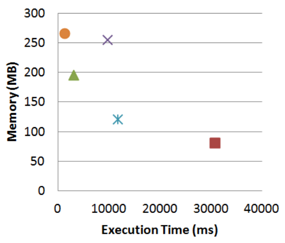
Figure 3 illustrates time versus memory for each model. We have used the average value of the time measures and the maximum value for the memory usage.



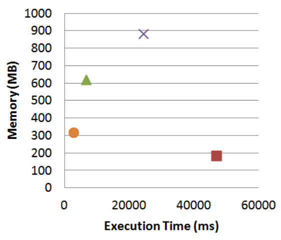
(a) Set0



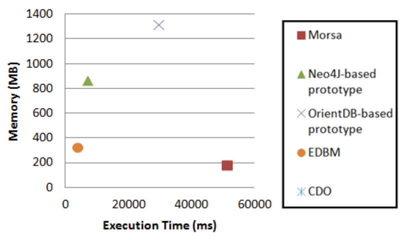
(b) Set1



(c) Set2



(d) Set3



(e) Set4

Fig. 3: Relation of execution time and memory usage results.

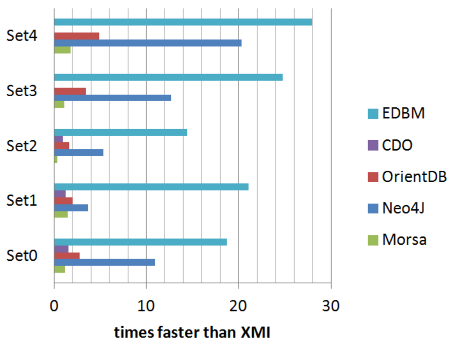
It is important to remark that in EDBM 100 executions have been performed, while 20 and 2 have been made in NoSQL_Eval and Morsa_Eval respectively.

Figure 3a illustrates the time versus memory results for set0. Except EDBM the rest of the approaches use a low amount of memory (between 10-17MB). Neo4J-based prototype is the fastest approach (110ms) having also one of the lowest footprints in memory usage (15MB). In case of Morsa, although memory usage is low (13MB), requires more time than the other options (870ms). In EDBM the memory usage is higher (182MB), but regarding execution time it is placed second (182ms).

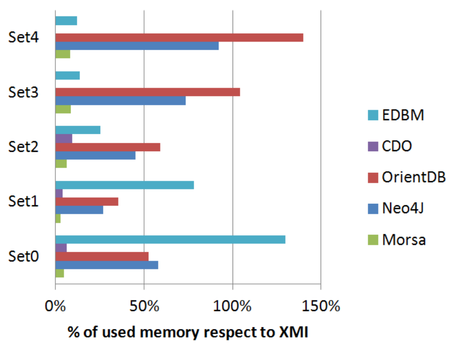
Figure 3b illustrates the values for set1. EDBM continues being the approach with the highest memory usage (159MB), but it is also the fastest approach (232ms). Neo4J-based prototype requires the lowest memory amount (18MB) being the second fastest (620ms). CDO is the approach that requires most time (1427ms).

As Figure 3c illustrates set2, Morsa is the approach less memory requirements (81MB), but the slowest (30872ms). EDBM is the fastest one (1430ms) and it is followed by Neo4J-based prototype (3100ms). Although the memory usage continues being higher in EDBM (299MB), in this case, the difference is lower compared to the others.

Results of set3 are illustrated on Figure 3d. Morsa continues being the approach with lowest memory usage value (182MB), but now is followed by EDBM (318MB). EDBM is the fastest one (2907ms), followed by Neo4J-based prototype (6710ms). The OrientDB-based prototype is the approach that uses more



(a) execution time



(b) memory usage

Fig. 4: Result comparison in different approaches.

memory (2229MB) and the execution time is 24410ms. CDO values have been omitted since the insertion for set3 failed.

Finally, Figure 3e illustrates the results of set4, very similar to the previous one. In terms of memory the best option is Morsa (180MB), but with a high execution time (51322ms). In terms of time EDBM is the best option with 4021ms and a low memory (345MB).

EDBM is one of the fastest approaches for the five models. However, it is the approach using more memory from set0 to set2. The situation changes in set3 and set4, where it becomes one of the approaches using less memory. These results indicate that EDBM provides an scalable alternative for the persistence and query of large models.

This comparative is more a reference-guide, since execution environments vary. To be able to equally compare, we have standardised the results comparing them with the result of the XMI persistence on each approach. Figure 4 shows: (i) how many times faster is the approach respect to XMI of its related study; and (ii) how much percentage of the memory uses each approach respect to XMI. As Figure 4a illustrates, EDBM and Neo4J-based approach provide more scalability in terms of time to execute the GraBaTs query. Regarding memory, Figure 4b shows that Morsa is the option that scales better. Concerning EDBM, while memory usage is higher in the small models, it scales better as model size increases.

5.4 Threats to Validity

The obtained memory and execution time results, show that our approach is promising in terms of scalability comparing to XMI. Moreover, results indicate that EDBM provides similar scalability to other existing approaches when large models are persisted and queried. However, it has the highest memory usage with smallest models, but still acceptable (e.g. 299MB for set2).

The comparison includes values extracted from different studies, and even if it is useful as reference-guide, executing all the approaches using the same execution environment is more appropriate. Regarding the evaluation, we have selected the GraBaTs 2009 case study, since it is widely used to evaluate the scalability of similar approaches. But using a real industrial domain with real use cases would be more realistic. We plan to perform this task in a future work.

6 Conclusions and Future Work

In this paper, we have evaluated EDBM, an approach for persisting and querying large-scale models. The evaluation is based on the GraBaTs 2009 case study, where large models and a complex query are used to evaluate approaches. Results of the performed evaluations show that (i) EDBM is able to persist large models using a metamodel-agnostic embedded relational database; (ii) runtime translation of EOL queries to SQL, providing a scalable solution to query models persisted in an embedded relational database. We have compared the evaluation results of EDBM, with the results of other existing approaches. These results show that our approach is promising in terms of scalability in contrast to XMI and other persistence approaches.

For future work, we plan to add support for translating model modifications by using EOL. Moreover, we plan to evaluate this approach using an industrial case study. Although EDBM is focused on scalable persistence and query, providing version-control and integration with existing modelling editors is also planed for a future version [23].

Acknowledgments

This work is partially supported by the EC, through the Scalable Modelling and Model Management on the Cloud (MONDO) FP7 STREP project (#611125).

References

1. Benelallam, A., Gómez, A., Sunyé, G., Tisi, M., Launay, D.: Neo4EMF, a Scalable Persistence Layer for EMF Models. In Cabot, J., Rubin, J., eds.: Modelling Foundations and Applications. Volume 8569 of Lecture Notes in Computer Science. Springer International Publishing (2014) 230–241
2. Espinazo Pagán, J., García Molina, J.: Querying Large Models Efficiently. Inf. Softw. Technol. **56**(6) (June 2014) 586–622
3. Carlos, X.D., Sagardui, G., Trujillo, S.: MQT, an Approach for Run-Time Query Translation: From EOL to SQL. In: Proceedings of the 14th International Workshop on OCL and Textual Modelling co-located with MODELS 2014, Valencia, Spain, September 30, 2014. (2014) 13–22
4. Carlos, X.D., Sagardui, G., Murguzur, A., Trujillo, S., Mendialdua, X.: Model Query Translator - A Model-Level Query Approach For Large-Scale Models. In: MODELSWARD 2015 - Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, Angers, France. (2015)

5. Sottet, J.S., Jouault, F., et al.: Program comprehension. In: Proc. 5th Int. Workshop on Graph-Based Tools. (2009)
6. Pagán, J.E., Cuadrado, J.S., Molina, J.G.: A Repository for Scalable Model Management. Software & Systems Modeling (2013) 1–21
7. Hunt, B.: Mongo-EMF. `https://github.com/BryanHunt/mongo-emf/wiki` (2014) [Online; accessed 30-Jan-2015].
8. 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
9. Stepper, E.: CDO. `http://eclipse.org/cdo/` (2009) [Online; accessed 30-Jan-2015].
10. Irawan, H., Taal, M.: Teneo/Hibernate. `http://wiki.eclipse.org/Teneo/Hibernate` (2012) [Online; accessed 30-January-2015].
11. Barmpis, K., Kolovos, D.S.: Evaluation of Contemporary Graph Databases for Efficient Persistence of Large-Scale Models. Journal of Object Technology **13**(3) (July 2014) 3:1–26
12. The Mondo Project. `http://www.mondo-project.org/` (2015) [Online; accessed 21-Feb-2015].
13. Shah, S.M., Wei, R., Kolovos, D.S., Rose, L.M., Paige, R.F., Barmpis, K.: A Framework to Benchmark NoSQL Data Stores for Large-Scale Model Persistence. In Dingel, J., Schulte, W., Ramos, I., Abraho, S., Insfran, E., eds.: Model-Driven Engineering Languages and Systems. Volume 8767 of Lecture Notes in Computer Science. Springer International Publishing (2014) 586–601
14. Hunter, A.: Emf query. `https://projects.eclipse.org/projects/modeling.emf.query/` (2015) [Online; accessed 02-February-2015].
15. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., Varró, D.: EMF-IncQuery: an Integrated Development Environment for Live Model Queries. Sci. Comput. Program. **98** (2015) 80–99
16. Kolovos, D.S., Rose, L.M., Paige, R.F.: The epsilon book (2010)
17. Egea, M., Dania, C., Clavel, M.: MySQL4OCL: A Stored Procedure-Based MySQL Code Generator for OCL. Electronic Communications of the EASST **36** (2010)
18. Heidenreich, F., Wende, C., Demuth, B.: A Framework For Generating Query Language Code From OCL Invariants. Electronic Communications of the EASST **9** (2007)
19. Demuth, B., Hussmann, H., Loecher, S.: OCL as a Specification Language for Business Rules in Database Applications. In Gogolla, M., Kobryn, C., eds.: UML 2001 The Unified Modeling Language. Modeling Languages, Concepts, and Tools. Volume 2185 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 104–117
20. Marder, U., Ritter, N., Steiert, H.: A DBMS-Based Approach for Automatic Checking of OCL Constraints. In: Proceedings of OOPSLA. Volume 99. (1999) 1–5
21. Parreiras, F.S.: Semantic Web and Model-driven Engineering. John Wiley & Sons (2012)
22. Kolovos, D.S., Wei, R., Barmpis, K.: An Approach for Efficient Querying of Large Relational Datasets with OCL-based Languages. In: XM 2013–Extreme Modeling Workshop. (2013) 48
23. Carlos, X.D., Sagardui, G., Trujillo, S.: Scalable Model Edition, Query and Version Control Through Embedded Database Persistence. In: Joint Proceedings of MODELS 2014 Poster Session and the ACM Student Research Competition, Valencia, Spain, September 28 - October 3, 2014. (2014) 11–15