Towards Hybrid Model Persistence

Alfa Yohannis[1,3], Horacio Hoyos Rodriguez*[1], Fiona Polack**[2], and Dimitris Kolovos[1]

[1]Department of Computer Science, University of York, United Kingdom
[2]School of Computing and Maths, Keele University, United Kingdom
[3]Department of Computer Science, Institut Teknologi dan Bisnis Kalbis, Indonesia
{ary506, dimitris.kolovos}@york.ac.uk
*horacio_hoyos_rodriguez@ieee.org
**f.a.c.polack@keele.ac.uk

Abstract. Change-based persistence has the potential to support faster and more accurate model comparison, merging, as well as a range of analytics activities. However, reconstructing the state of a model by replaying its editing history every time the model needs to be queried or modified can get increasingly expensive as the model grows in size. In this work, we integrate change-based and state-based persistence mechanisms in a hybrid model persistence approach that delivers the best of both worlds. In this paper, we present the design of our hybrid model persistence approach and report on its impact on time and memory footprint for model loading, saving, and storage space usage.

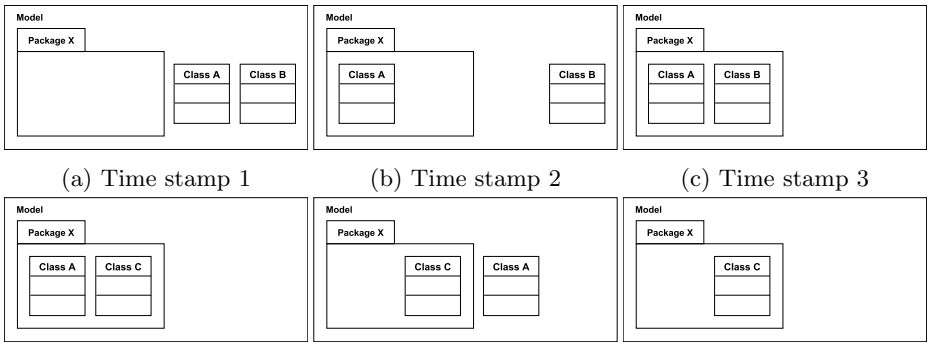
1 Introduction

Change-based persistence (CBP) of models [1] conforming to metamodelling architectures such as MOF/EMF [2,3] comes with notable advantages over state-based persistence (SBP): it provides support for fast comparison and differencing of versions of the same model [4,5,6,7] – which can also substantially speed up incremental model management activities, and enables novel model analytics activities (e.g. pattern detection in the editing history to understand how modellers use modelling languages and tools) [8]. However, CBP comes at the cost of ever-growing model files [6,8] since all changes (even deleting model elements) are recorded in an editing log, which naturally leads to longer loading times [9]. In this work, we address the latter challenge by introducing the concept of hybrid persistence of models. In hybrid model persistence the change-based representation is augmented with a state-based representation (which may be derived from the change-based representation) of the latest state of the model which is used to speed up model loading and querying.

The paper is structured as follows. Section 2 introduces the concept of change-based model persistence and recent work on state-based model persistence. Sections 3 and 4 present our approach to hybrid model persistence and its implementation. Section 5 presents experimental results and evaluation. Section 7 provides an overview of related work, and Section 8 concludes with a discussion on directions for future work.

2 Change and State-based Model Persistence

To explain the differences, benefits and drawbacks of CBP and SBP, consider a modelling activity on a UML model as presented Fig. 1. The sub-figures 1a to 1f depict the evolution of a UML model at different time stamps. Classes are created and added/removed from `Package X`. In SBP, for each session, only the final state of the model is persisted (the state of previous session are overridden by the state of the latest session). Thus, to represent the final state of the UML model, only the information about `Package X` and `Class C` needs to be persisted, as presented in Listing 1 (XMI format). In CBP, all the changes in the model are persisted. Thus, a list of all the events generated by the model editor is needed to represent the final state of the model.



(a) Time stamp 1 (b) Time stamp 2 (c) Time stamp 3

(d) Time stamp 4 (e) Time stamp 5 (f) Time stamp 6

Fig. 1: The states of the example model after certain changes and their corresponding lines in Listing 2.

A session depicts a set of changes made between *save* events, i.e. a session comprises all the changes that happened since the last time that the model was persisted. The CBP representation is shown in Listing 2[1]. Lines 1-7 represent the initial state (Fig. 1a), followed by lines 8 (Fig. 1b), 9 (Fig. 1c), 11 (Fig. 1d), 12 (Fig. 1e), and 13 (Fig. 1f).

Table 1 summarises the benefits (+) and drawbacks (-) of change and state-based model persistence. To load an SBP model, only the elements that exist in the final state need to be loaded into memory. To load a CBP model, all the events that lead to the final state must be replayed to load the model in memory. Loading times for SBP models are proportional to the size of the model. Loading times for CBP models are proportional to the number of events. As a result, loading times of CBP models will always increase over time and are considerably longer than for SBP [10,9].

[1] We use a natural language pseudo-code for CBP, introduced in [1,10]

To store an SBP model, all the elements that exist in the final state must be persisted. To save a CBP, only the change events in the last session need to be persisted. Storing times of SBP models are proportional to the size of the model. Storing times of CBP models are proportional to the number of events in a session. As a result, storing times of CBP models can be considerably shorter than for SBP models [10]. Comparing and finding the differences between two versions of a state-based model is expensive [11] ($O(N^2)$ in the general case) which affects the efficiency of change visualisation and comprehension, and has a substantial impact on downstream activities such as incremental model transformation [12] and validation.

Listing 1: The UML2 model of the example model in Fig. 1.

```
1   <uml:Package xmi:id="1" name="X">
2   <packagedElement xsi:type="uml:Class"
        xmi:id="3" name="C"/>
3   </uml:Package>
```

Table 1: Comparison of model persistence approaches.

Dimensions	Change-based	State-based
Load Time	−	+
Save Time	+	−
Comparison Time	+	−
Storage Space	−	+

Listing 2: The textual CBP for producing state-based model in List. 1. Its visual illustration is in Fig. 1.

```
1   session 1
2   create p1 type Package
3   set p1.name to "X"
4   create c1 type Class
5   set c1.name to "A"
6   create c2 type Class
7   set c2.name to "B"
8   add c1 to p1.
        packagedElement
9   add c2 to p1.
        packagedElement
10  session 2
11  set c2.name to "C"
12  remove c1 from p1.
        children
13  delete c1
```

By contrast, in CBP, changes are first-class entities in the persisted model file and as such, model comparison and differencing is relatively inexpensive. The main downsides of CBP are it's model file sizes [8,6] and ever-increasing loading times [9]. Loading times can be reduced by around 50% by processing the changelog, detecting, memorising and subsequently ignoring change events that have no impact to the final state of the model. The loading times are still substantially longer – more than 6.4 times slower and even longer as the persisted changes increase – than loading times for state-based approaches [10].

3 Hybrid Model Persistence

To achieve the best of both worlds we introduce a hybrid model persistence approach which combines change-based and state-based model persistence, to work together side-by-side. An overview of the proposed approach is illustrated in Fig. 2. In the proposed approach a *hybrid* model is stored in two representations at the same time: a change-based (e.g. using CBP) and a state-based representation (e.g. using XMI or a database-backed approach such as NeoEMF). The change-based representation is perceived as the main representation of a model, while the state-based representation can be fully derived from the change-based representation.

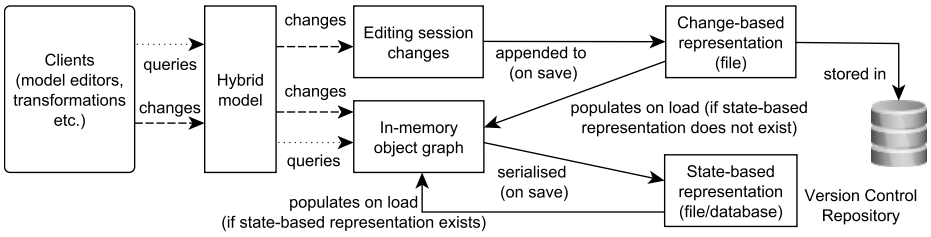


Fig. 2: The mechanism of hybrid model persistence.

Loading a hybrid model. Models are loaded into in-memory object graphs that clients (e.g. editors, transformations) can then interact with[2]. In the proposed hybrid approach, if the state-based counterpart already exists, the in-memory object graph is populated from it; otherwise, it is populated by replaying the complete editing history recorded in the change-based representation.

Changing a hybrid model. When an element in a loaded model is created, modified or deleted, the change is applied to the in-memory object graph and is also recorded in an in-memory list of changes (*Editing session changes* in Fig 2). We use the term *editing session* for the period between loading a model and saving back to disk.

Saving a hybrid model. The current version of the in-memory object graph is stored in the preferred state-based representation. The list of changes recorded in the current editing session (with optional processing, as described above) is appended to the change-based representation.

Versioning a hybrid model. Since the state-based representation is fully derived from the change-based representation, if a model needs to be versioned (e.g. in a Git repository), only the change-based representation needs to be stored. The first time it is loaded after being checked out/cloned, the state-based representation is computed and persisted locally and is used in subsequent model loading steps.

Comparing hybrid models. To compare two hybrid models[3], their change-based representations are used: this is much more efficient than state-based comparison.

[2] Depending on the state persistence mechanism, the object graph may be loaded in its entirety at startup (e.g. XMI) or loaded progressively, in a lazy manner (e.g. NeoEMF/CDO)

[3] The work of the hybrid model comparison is still in the preliminary stage and out of the scope of this paper.

4 Implementation

We have implemented the proposed hybrid model persistence approach in a prototype[4] on top of the Eclipse Modeling Framework (EMF) [3]. The prototype makes use of an existing implementation of change-based model persistence, the Epsilon CBP [1], augmented with two state-based persistence implementations: NeoEMF [13] and XMI [14].

XMI has been selected as a standard state-based model persistence format (natively supported by EMF), and NeoEMF as a best-of-breed representative of database-backed state-based model persistence frameworks. The core components of the prototype are presented in Fig. 3.

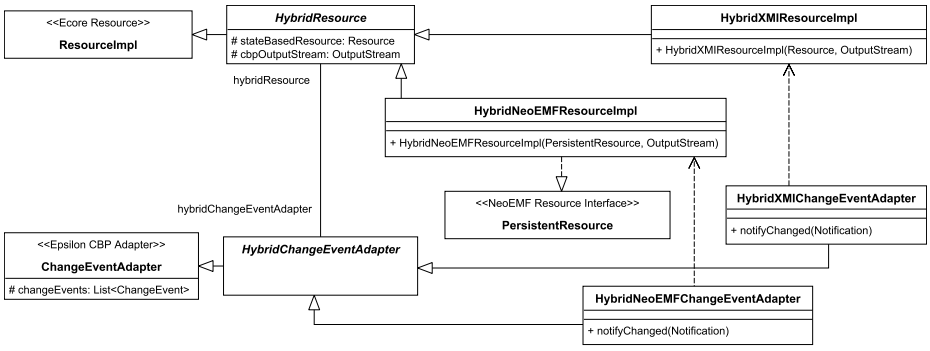


Fig. 3: Class diagram showing the core components of the hybrid model persistence implementation.

The Epsilon CBP provides a ChangeEventAdapter class [1] that extends from Ecore's EContentAdapter adapter class. This class collects changes made to the in-memory object graph of an EMF model in the form of a list of events changeEvents. Based on this class, we derived an adapter class, HybridChangeEventAdapter, for the hybrid model persistence implementation. It is an abstract class so that it can be further derived to create different implementations of adapter classes for different types of state-based persistence. The HybridNeoEMFChangeEventAdapater is the adapter class for NeoEMF, and the HybridXMIChangeEventAdapater for XMI. These classes override notifyChanged(Notification) in the ChangeEventAdapter class, to handle events that are specific to NeoEMF and XMI, respectively.

We also created a resource class for hybrid persistence, HybridResource (a resource class is a class dedicated to interacting with a persistence, e.g. save, load, get contents), derived from the Ecore's ResourceImpl. The class is again abstract so that it can be realised in different resource implementation classes for different state-based persistence. The HybridResource class contains the stateBasedResource field which is used to refer to a state-based persistence that is being

[4] The prototype is available under https://github.com/epsilonlabs/emf-cbp.

used, and the cbpOutputStream field that refers to an OutputStream (e.g. file, in-memory) as the representation of the CBP for saving changes. HybridResource has an association with HybridChangeEventAdapater, so that the former can access the events collected by the latter, and the latter can also use facilities provided by the former (e.g. getting the identity of an element in the resource; saving changes to a change-based model representation).

The resource implementation classes for NeoEMF and XMI are HybridNeoEMFResourceImpl and HybridXMIResourceImpl respectively. HybridNeoEMFResourceImpl also implements the NeoEMF's PersistenceResource interface so that specific NeoEMF's methods can be used (e.g. close(), to close a connection with a back-end database).

5 Evaluation

In this section, we compare hybrid model persistence (Epsilon CBP with each of NeoEMF and XMI) vs state-based persistence (NeoEMF or XMI only) on storage space usage, loading and saving time and memory footprint, and demonstrate that hybrid model persistence can still perform fast model loading and saving.

The evaluation was performed on Intel® Core™ i7-6500U CPU @ 2.50GHz 2.59GHz, 12GB RAM, and the Java™ SE Runtime Environment (build 1.8.0_162-b12). For the evaluation, we used models reverse-engineered from the Java source code of the Epsilon [15,16] and BPMN2 [17] projects. For state-based representation of the models, we used the MoDisco tool [18] to generate XMI-based UML2 [19] models that reflect the classes, fields, and operation signatures of the source code of the project and then imported the generated models into NeoEMF. We also derived MoDiscoXML models [20] from the Wikipedia article on the United States [21]. We then used reverse-engineering to generate a CBP for each project based on the differences between consecutive versions of the models.

Table 2: Space usage for the Epsilon and BPMN2 projects, and the Wikipedia's United States article.

Case	**Epsilon**			**BPMN2**			**Wikipedia**		
Generated From	940 commits			192 commits			10,187 versions		
Type	XMI	NeoEMF	CBP	XMI	NeoEMF	CBP	XMI	NeoEMF	CBP
Element Count	88,020	88,020	—	62,062	62,062	—	13,112	13,112	—
Event Count	—	—	4.3 m	—	—	1.2 m	—	—	62.3 m
Space Size	9.44 MBs	188 MBs	406 MBs	6.55 MBs	134 MBs	109 MBs	1.28 MBs	31.8 MBs	5.85 GBs
Average Space Size	112 bytes/ element	2 KBs/ element	98 bytes /event	110 bytes/ element	2 KBs/ element	92 bytes /event	102 bytes/ element	2 KBs/ element	98 bytes /event

m = million events, MB = Megabytes, KB = Kilobytes

5.1 Storage Space Usage

For the Epsilon project, we have successfully generated a CBP from version 1 up to version 940 and also CBPs for the BPMN2 project and Wikipedia article up to version number 192 and 10,187 respectively. The details (element count, event count, space size, and average space size per element or event) of their models, when persisted in XMI, NeoEMF, and CBP are shown in Table 2. The last row of the table derives an average space usage per element (for the SBPs) or event (for the CBP). We can estimate the storage space usage for a hybrid model persistence to be the combination of CBP and the appropriate SBP space usage.

Table 3: The comparison on time and memory footprint for loading and saving models of the hybrid and state-based-only persistence.

Dimension	**Case**	**Backend**	**Hybrid**		**State-based**		**Significance**	
			mean	*sd*	*mean*	*sd*	*W*	*p-value*
Loading Time	Epsilon	NeoEMF	0.292	0.061	0.279	0.023	258	**0.72**
		XMI	0.317	0.006	0.270	0.018	26	< 0.05
	BPMN2	NeoEMF	0.308	0.071	0.286	0.025	230	**0.79**
		XMI	0.212	0.016	0.179	0.016	37	< 0.05
	Wikipedia	NeoEMF	0.262	0.048	0.273	0.062	250	**0.86**
		XMI	0.045	0.001	0.040	0.001	0	< 0.05
Saving Time	Epsilon	NeoEMF	0.0892	0.0421	0.0829	0.0494	216	**0.55**
		XMI	0.411	0.023	0.397	0.015	78	< 0.05
	BPMN2	NeoEMF	0.0777	0.0424	0.0775	0.0452	213	**0.51**
		XMI	0.33	0.007	0.28	008	0	< 0.05
	Wikipedia	NeoEMF	0.135	0.048	0.120	0.024	218	**0.59**
		XMI	0.024	0.048	0.020	0.002	42	< 0.05
Loading Memory Footprint	Epsilon	NeoEMF	38.601	0.878	10.014	1.088	0	< 0.05
		XMI	10.72018	0.00022	10.72009	0.00024	0	< 0.05
	BPMN2	NeoEMF	40.78	1.29	27.20	1.05	0	< 0.05
		XMI	6.73367	1.29305	6.73367	0.00056	101	< 0.05
	Wikipedia	NeoEMF	35.91	1.03	27.25	0.54	27.25	**0.54**
		XMI	8.4079	0.0008	8.0933	0.0009	0	< 0.05
Saving Memory Footprint	Epsilon	NeoEMF	2.64	1.29	2.61	0.78	283	**0.34**
		XMI	1.56355	0.0005	1.56326	0.0018	408	< 0.05
	BPMN2	NeoEMF	1.86	3.86	1.52	0.77	308	**0.12**
		XMI	0.8378	0.00361	0.8375	0.00362	58	< 0.05
	Wikipedia	NeoEMF	1.32	1.51	0.97	0.76	189	**0.22**
		XMI	0.0010	0.00044	0.0005	0.00001	0	< 0.05

The time is in seconds, and the memory footprint is in MBs.

5.2 Time and Memory Footprint of Loading and Saving Models

We evaluated the performance of our hybrid persistence prototype against XMI and NeoEMF regarding time and memory footprint for loading and saving. We repeated our experiments 22 times for each dimension measured. Since the data

were not normally distributed, we used the nonparametric Mann-Whitney U test [22] with a significance level of 5%.

As it can be noticed in Table 3, all cases experience a slight slowdown on loading and saving time (hybrid approach's *mean* > state-based approach's *mean*). However, almost for all NeoEMF cases, the slowdown is not significant, which means that side-effect of the hybrid approach on loading and saving time is still acceptable. The hybrid approach also produces more memory footprint compared to the state-based-only approach. Nevertheless, considering the cost of main memory, this condition is acceptable in almost all real-world scenarios.

6 Discussion

The use of state-based persistence in hybrid model persistence enables faster model loading, as shown by the result of loading time evaluation in Section 5.2, without having to replay all the changes persisted in its CBP – the main challenge for the change-based approach [10,9]. Hybrid model persistence performs slightly slower – statistically significant for Hybrid XMI but insignificant for Hybrid NeoEMF – compared to loading a state-based model. A slight slowdown also appears on model saving – statistically significant for Hybrid XMI but insignificant for Hybrid NeoEMF (Section 5.2). The slowdown is because changes have to be persisted into two representations, state-based and change-based.

The main drawback of hybrid model persistence is that it consumes more memory when loading and saving and storage space for persisting models compared to state-based representation only (Sections 5.2 and 5.1). However, considering the cost of main memory and storage, the trade-off can be acceptable in most real-world scenarios.

7 Related Work

There are several non-XMI approaches to state-based model persistence, using relational or NoSQL databases. For example, EMF Teneo [23] persists EMF models in relational databases, while Morsa [24] and NeoEMF [13] persist models in document and graph databases, respectively. None of these approaches provides built-in support for versioning and models are eventually stored in binary files/folders which are known to be a poor fit for text-oriented version control systems like Git and SVN. Connected Data Objects (CDO) [25], which provides support for database-backed model persistence, also provides collaboration facilities, but CDO adoption necessitates the use of a separate version control system (e.g. a Git repository for code and a CDO repository for models), which introduces fragmentation and administration challenges [26]. Similar challenges arise in relation to other model-specific version control systems such as EMFStore [7].

8 Conclusions and Future Work

In this paper, we have proposed a hybrid model persistence approach and evaluated its impact on time and memory footprint for model loading and saving,

and storage space usage. Based on the evaluation results, the hybrid model persistence provides benefits on model loading time with an acceptable trade-off on memory footprint and storage space usage.

Currently, we are still working on the hybrid model comparison (Section 3 – Comparing hybrid models). So far, the progress is promising. Based on our preliminary investigation, it can detect atomic changes of models faster than state-based model comparison, e.g. detecting elements that have been removed from older versions. In the future, we plan to evaluate hybrid model persistence on even larger models and perform experiments where software modellers are asked to construct change-based models. We also plan to develop a solution for the efficient merging of change-based and hybrid models.

Acknowledgements. This work was partly supported by through a scholarship managed by *Lembaga Pengelola Dana Pendidikan Indonesia* (Indonesia Endowment Fund for Education).

References

1. Yohannis, A., Kolovos, D.S., Polack, F.: Turning models inside out. In: Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium, Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017), Austin, TX, USA, September, 17, 2017. (2017) 430–434
2. OMG: Metaobject Facility. http://www.omg.org/mof Accessed: 2018-02-21.
3. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Eclipse Series. Pearson Education (2008)
4. Lippe, E., van Oosterom, N.: Operation-based merging. In: SDE 5: 5th ACM SIGSOFT Symposium on Software Development Environments, Washington, DC, USA, December 9-11, 1992. (1992) 78–87
5. Ignat, C., Norrie, M.C.: Operation-based merging of hierarchical documents. In: The 17th Conference on Advanced Information Systems Engineering (CAiSE '05), Porto, Portugal, 13-17 June, 2005, CAiSE Forum, Short Paper Proceedings. (2005)
6. Koegel, M., Herrmannsdoerfer, M., Li, Y., Helming, J., David, J.: Comparing state- and operation-based change tracking on models. In: Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2010, Vitória, Brazil, 25-29 October 2010. (2010) 163–172
7. Koegel, M., Helming, J.: Emfstore: a model repository for EMF models. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010, Cape Town, South Africa, 1-8 May 2010. (2010) 307–308
8. Robbes, R., Lanza, M.: A change-based approach to software evolution. Electr. Notes Theor. Comput. Sci. **166** (2007) 93–109
9. Mens, T.: A state-of-the-art survey on software merging. IEEE Trans. Software Eng. **28**(5) (2002) 449–462

10. Yohannis, A., Rodriguez, H.H., Polack, F., Kolovos, D.: Towards efficient loading of change-based models. In: Modelling Foundations and Applications - 14th European Conference, ECMFA 2018, Held as Part of STAF 2018, Toulouse, France, June 25-29, 2018. Proceedings. (2018 (to be presented http://eventmall.info/ecmfa2018/program)) Accessed: 2018-04-19.
11. Kolovos, D.S., Di Ruscio, D., Pierantonio, A., Paige, R.F.: Different models for model matching: An analysis of approaches to support model differencing. In: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models. CVSM '09, Washington, DC, USA, IEEE Computer Society (2009) 1–6
12. Ogunyomi, B., Rose, L.M., Kolovos, D.S.: Property access traces for source incremental model-to-text transformation. In: Modelling Foundations and Applications - 11th European Conference, ECMFA 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-24, 2015. Proceedings. (2015) 187–202
13. Daniel, G., Suny, G., Benelallam, A., Tisi, M., Vernageau, Y., Gmez, A., Cabot, J.: Neoemf: A multi-database model persistence framework for very large models. Science of Computer Programming **149** (2017) 9 – 14 Special Issue on MODELS'16.
14. OMG: About the XML Metadata Interchange Specification Version 2.5.1. http://www.omg.org/spec/XMI Accessed: 2018-02-21.
15. Eclipse: Epsilon. https://www.eclipse.org/epsilon/ Accessed: 2018-02-12.
16. Eclipse: Epsilon Git. http://git.eclipse.org/c/epsilon/org.eclipse.epsilon.git/commit/?id=ebd0991c279a1f0df1acb529367d2ace5254fe87 Accessed: 2018-02-19.
17. Eclipse: MDT/BPMN2. http://wiki.eclipse.org/MDT/BPMN2 Accessed: 2018-01-15.
18. Brunelière, H., Cabot, J., Dupé, G., Madiot, F.: Modisco: A model driven reverse engineering framework. Information & Software Technology **56**(8) (2014) 1012–1032
19. Eclipse: MDT/UML2. http://wiki.eclipse.org/MDT/UML2 Accessed: 2018-01-15.
20. Eclipse: XML Metamodel. http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.modisco.xml.doc%2Fmediawiki%2Fxml_metamodel%2Fuser.html Accessed: 2018-02-19.
21. Wikipedia: United States. https://en.wikipedia.org/w/index.php?title=United_States&oldid=45118452 Accessed: 2018-02-19.
22. McKnight, P.E., Najab, J. In: MannWhitney U Test. American Cancer Society (2010) 1–1
23. Eclipse: Teneo. http://wiki.eclipse.org/Teneo Accessed: 2017-10-15.
24. Espinazo-Pagán, J., Cuadrado, J.S., Molina, J.G.: Morsa: A scalable approach for persisting and accessing large models. In: Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings. (2011) 77–92
25. Eclipse: CDO The Model Repository. https://www.eclipse.org/cdo/ Accessed: 2017-10-15.
26. Barmpis, K., Kolovos, D.S.: Evaluation of contemporary graph databases for efficient persistence of large-scale models. Journal of Object Technology **13**(3) (2014) 3: 1–26