NeoEMF

Christian Langenbach

ABSTRACT

NeoEMF ist ein Projekt des AtlanMod Research Teams, welches darauf abzielt, große EMF-Modelle (*Eclipse Modeling Framework*) effizient handzuhaben[1]. Bei NeoEMF handelt es sich um eine Anwendung für Model-driven engineering (Graph-Datenbanken / NoSQL-Datenbanken). Es ist ein Multi-Backend Framework, das die Integration von angepassten Backends, auf den Nutzer zugeschnitten, ermöglicht. NeoEMF ist durch mehrere Eigenschaften gekennzeichnet, die vom Backend abhängen. Alle Backends unterstützen den Lazy Loading Mechanismus, der dafür sorgt, dass das Modell stückweise geladen wird. Weitere Features sind Caching, Auto-commit und Dirty-Saving, die im Zuge dieser Arbeit näher erläutert werden. Es werden außerdem die unterstützten Backends und ihr Einfluss auf die Features von NeoEMF beschrieben.[2]
Die Arbeit wird sich in folgende Teile gliedern: Was ist NeoEMF? Hier wird ein wenig Hintergrundwissen vermittelt. Da es sich bei NeoEMF um eine Anwendung für NoSQL-Datenbanken handelt, wird dieser Begriff anschließend kurz verdeutlicht. Fortgesetzt wird dieses, durch eine kurze Anleitung zur Installation. Der Hauptteil der Arbeit wird von den Eigenschaften und Features von Neo-EMF handeln. Neo4j ist ein wichtiger Teil von NeoEMF, daher wird diesbezüglich auch ein wenig Wissen vermittelt. Abgerundet wird dies durch einen Vergleich mit der Vorgängerversion Neo4EMF und einem abschließenden Fazit.[2]

1. EINLEITUNG

NeoEMF ist eine Open Source Software von dem AtlanMod Research Team. Es ist eine Neuveröffentlichung eines früheren Tools namens Neo4EMF.
In der heutigen Zeit ist EMF zum Standard für MDE[1]-Tools geworden.[3]
Standardmäßig kann NeoEMF dazu genutzt werden Datenbestände grafisch anzuzeigen. Allerdings ist NeoEMF so designt, dass es einfach erweiterbar ist. NeoEMF ist stark verknüpft mit Neo4j [2] und beinhaltet außerdem alle Features von Neo4EMF, wodurch der Einstieg für frühere Nutzer erleichtert wird. NeoEMF ist nicht performancelastig und schafft es selbst große Modelle mit nur wenig Speicher zur Verfügung zu stellen. Das wird durch folgende Features ermöglicht[4]:

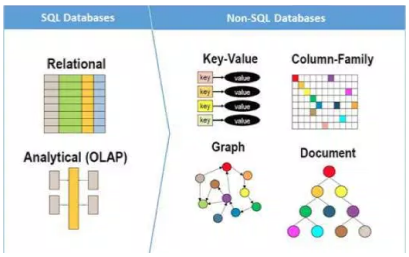

Abbildung 1: SQL vs NoSQL [3]

- Lazy-loading
- Caching
- Auto-commit
- Dirty saving

NeoEMF ist ein junges Projekt, an dem immer noch weitergearbeitet wird. Das lässt sich vor allem dadurch erkennen, dass die Dokumentation von NeoEMF bei weitem nicht so umfangreich ist, wie die für Neo4EMF. Große Teile der Dokumentation für Neo4EMF lassen sich allerdings auf NeoEMF übertragen.
Schlussendlich sollte man noch wissen, dass es sich bei NeoEMF nicht um ein eigenständiges Programm handelt, sondern vielmehr um ein Eclipse Plugin.

2. NOSQL

Bei NeoEMF handelt es sich um eine Software für NoSQL Datenbestände. Deshalb soll dieser Begriff hier kurz umrissen werden.
Der Begriff NoSQL steht für *Not Only SQL*, oder für *Not Relational*. Nicht Relational bedeutet, dass keine Tabellen zur Schemaverwaltung genutzt werden, sondern beispielsweise Graphen. Große Datenbestände werden dadurch performanter zu handhaben.[5] NoSQL-Datenbanken besitzen

[1]Model Driven Engineering

[2]Neo4j ist eine NoSQL Graph-Datenbank

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

mehrere Kern-Funktionen / Eigenschaften[6]:

- Die Möglichkeit zur horizontalen Skalierung. Das bedeutet, dass ein System nicht durch seine Hardware eingeschränkt ist und einfach durch Hinzufügen weiterer Knoten gestärkt werden kann.[7]
- Die Möglichkeit Daten über mehrere Knoten zu vermehren / verbreiten. Das liefert vor allem eine gewisse Redundanz als Sicherheitsvorkehrung.
- Ein einfaches Call Level Interface[4] / Protokoll
- Ein schwächeres Modell der Concurrency Control[5] als *ACID*-Transaktionen, die in herkömmlichen Datenbanksystemen oft greifen, ist möglich. ACID steht für *Atomarität, Konsistenzerhaltung, Isolation, Dauerhaftigkeit*. Werden diese Kriterien erfüllt, spricht man von ACID-Transaktionen (der Begriff *Transaktion* wird im weiteren Verlauf erklärt werden).
- Effiziente Nutzung von Indizes und RAM
- Die Möglichkeit Attribute dynamisch zu Datenbeständen hinzuzufügen

Ein bekannter Vertreter für eine NoSQL Datenbank ist Neo4j. Abbildung 1 veranschaulicht den Unterschied von SQL- zu NoSQL-Datenbanken.

3. INSTALLATION VON NEOEMF

Die Original-Anleitung zur Installation von NeoEMF ist auf *Github*[6] zu finden, allerdings soll sie an dieser Stelle noch einmal zusammengefasst werden.

- Da es sich bei NeoEMF um ein Ecipse-Plugin handelt, sollte zunächst Eclipse installiert werden (Version Luna oder höher).
- Nach diesem Schritt wird in der Titlebar unter *Help → Install New Software...* ausgewählt.
- Unter *Work With...* wird folgender Link eingetragen: *http://atlanmod.github.io/NeoEMF/*
- Nach kurzer Wartezeit sollte es möglich sein, die Backends und die Base von NeoEMF zur Installation auszuwählen.
- Es sollte nun auf *Select All* geklickt werden. Das ist nicht zwangsweise erforderlich, allerdings sollten wenigstens das Persistence Framework (die Grundlage von NeoEMF) und ein Backend ausgewählt werden. Wenn mit Neo4j gearbeitet werden soll, dann müssen alle Pakete von Blueprints ausgewählt werden.
- Setzen Sie die Installation fort und bestätigen Sie die Lizenzvereinbarung.

[3]Bild von *https://kvaes.files.wordpress.com/2015/01/1401269083847.jpg?w=788* (zuletzt besucht am 06.02.2016)
[4]Datenbankschnittstelle - Bekanntes Beispiel: *JDBC* für die Verbindung mit SQL-Datenbanken
[5]Ein DBMS, dass nur Verzahnungen zuzulässt, bei denen keine Interferenzen auftreten, nutzt eine sogennante Nebenläufigkeitssteuerung (Concurrency Control)[8, Sperrverfahren]
[6]https://github.com/atlanmod/NeoEMF/

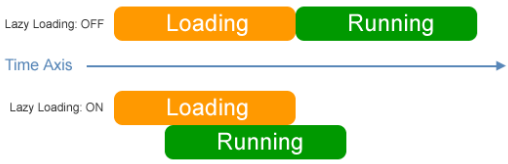

Abbildung 2: Lazy Loading[7]

- Im Anschluss kann unten rechts in der Ecke eine Progressbar mit der Beschriftung *Download New Software* ausgemacht werden. Der Vorgang kann mehrere Minuten in Anspruch nehmen. Möglicherweise erscheint eine Warnung, dass unbekannte Quellen genutzt werden. Diese kann getrost ignoriert werden.

Wenn die Installation erfolgreich war, sollten beispielsweise Ecore-Modelle mittels NeoEMF importiert werden können. Bei der Installation gibt es allerdings wohlbekannte Probleme. Es ist z.B. unter anderem nicht sichergestellt, dass NeoEMF unter Eclipse-Versionen läuft, die älter sind als Eclipse Luna.

4. FEATURES UND BACKENDS

4.1 Features

Im Folgenden werden nun die Features, die NeoEMF zu bieten hat, behandelt. Nicht alle Features stehen unter allen Backends bereit, deshalb ist es wichtig bei jedem Feature das gewählte Backend zu beachten.
NeoEMF unterstützt die MapDB und die Blueprints Datenbank vollständig, außerdem unterstützt es die Verbindung zu graphbasierten Datenbanken, wie z.B. Neo4j.[2]

4.1.1 Lazy Loading

"Lazy Loading, wörtlich „faules Laden“ (treffender etwa: „müßiges Laden“), bezeichnet in der Softwareentwicklung ein Entwurfsmuster, bei dem Datenobjekte grundsätzlich Werte oder andere, abhängige Objekte bereitstellen, diese aber erst bei einer konkreten Anfrage aus der Datenquelle holen. Das Gegenteil wird als „eager loading“ bezeichnet; hierbei werden möglichst effizient sofort alle absehbar benötigten Daten geholt."[9]
Im Falle von NeoEMF bedeutet dies, dass ein Modell nicht vollständig geladen werden muss, sondern nur die momentan benötigten Teile. Das macht das Tool vor allem bei sehr großen Modellen sehr effizient.[2]
Abbildung 2 veranschaulicht den Lazy Loading Mechanismus.

4.1.2 Caching

Dieses Feature ist nahezu selbsterklärend. Wie beim Caching üblich werden zuvor genutzte Daten eine gewisse Zeit lang vorgehalten, damit sie, wenn sie erneut benötigt werden, schneller geladen werden können.[10]
NeoEMF nutzt Datenbank Caches. Das ist insofern sinnvoll, da hauptsächlich mit EObjects gearbeitet wird. Wenn es bei großen Modellen dennoch zu Performanceeinbrüchen

[7]Bild von *http://www.foreui.com/wordpress/wp-content/uploads/2013/01/compare_lazy_loading.gif* (zuletzt besucht am 06.02.2016)

kommt, können sogenannte Application-Level Caches genutzt werden, um für einen schnelleren Informationsfluss zu sorgen. [2]

4.1.3 Auto-commit

Auto-commit ist ein Operationsmodus von Datenbanken. Betrachtet man z.B. einen SQL-Datenbestand, dann wird für jeden SQL-Befehl eine eigene Transaktion genutzt.[11]
"Eine Transaktion ist eine vom Benutzer definierte Folge von Aktionen, sozusagen eine "elementare Arbeitseinheit"." [8, Transaktionen]
Unter einer Transaktion versteht man, aus Sicht des Nutzers, eine scheinbar atomare Aktion. Man kann also Folgen von Aktionen, die ohne Nebenwirkungen auf beispielsweise parallel laufende Aktionen stattfinden können, als Transaktion bezeichnen. Dies ist wichtig für Sperrverfahren, bzw. Scheduler. Man möchte zu einem festen Zeitpunkt eine Serialisierung garantieren können.[8, Sperrverfahren]
Ähnlich einer relationalen Datenbank kann der Auto-commit Modus von NeoEMF gesehen werden. Liegen große Transaktionen vor, so können sie in mehrere Kleine unterteilt werden. Das liegt vor allem an Restriktionen bezüglich der maximalen Transaktionsgröße, die vom Backend getroffen werden. Der Auto-commit Modus ist nur verfügbar, wenn Blueprints als Backend genutzt wird.[2]

4.1.4 Dirty saving

Dieses Feature steht wiederum nur für das Blueprints Backend zu Verfügung. Es wird genutzt um Speicherüberläufe ("*memory overheads*") bei nicht persistent gespeicherten großen Modellen zu vermeiden. Andernfalls wären mögliche Ergebnisse plötzlich verloren, da die Daten bis dato nur im Hauptspeicher vorlagen.[2]
Bei Dirty Saves werden nur Datenbestände gespeichert, die tatsächlich verändert wurden.[12] Üblicherweise werden solche Daten mit einem Dirty-Bit versehen.
Es ist ersichtlich, dass sich dieser Mechanismus, kombiniert mit dem Lazy Loading Mechanismus besonders auszahlt. Daraus kann sich bei sehr großen Modellen ein starker Performancegewinn bemerkbar machen.

4.2 Backends

Die oben genannten Backends sollen im Folgenden etwas näher erläutert werden. Sie haben nicht nur Einfluss auf die Features von NeoEMF, sondern liefern auch eigene interessante Möglichkeiten.

4.2.1 Blueprints

"Blueprints ist eine Modell-Schnittstelle für Eigenschaftsgraphen. Es stellt Implementierungen, Test-Suiten, und die Unterstützung für Erweiterungen bereit. Graph-Datenbanken und Frameworks, die die Blueprints-Schnittstellen implementieren, unterstützen automatisch Blueprints-fähige Anwendungen. Ebenso können Blueprints-fähige Anwendungen verschiedene Blueprints-fähige Graph-Backends per Plug 'n Play einbinden."[13]
Als Implementierung werden unter anderem Neo4j, Tinkergraph und OrientDB unterstützt.[13]
Man kann Blueprints als Datenbankconnector ansehen, ähnlich wie JDBC, allerdings zielt Blueprints dabei auf Modelldatenbanken ab. Blueprints dient als grundlegende Technologie für[14]:

- Pipes
- Gremlin
- Frames
- Furnace
- Rexster

Zudem liefert Blueprints den Datenbank-Connetor, der nötig ist um mit Neo4j-Graphen zu arbeiten. Im Allgemeinen lässt sich auch festellen, dass NeoEMF für das Blueprints-Backend mehr Features zu Verfügung stellt.

Pipes.

"Pipes ist ein Dataflow Framework das Prozessgraphen nutzt."[15]
Das kann man sich in diesem Fall tatsächlich als Rohr (*Pipe*) vorstellen. Jeder Kommunikationspunkt wird im Graphen als Kante der *Pipe* dargestellt.[15] Ein einfaches Beispiel für die Implementierung einer Pipe, sowie ihr Durchlauf mittels Iterator, lässt sich auf *https://github.com/tinkerpop/pipes /wiki/Basic-Pipes* finden.
Bei Pipes unterscheidet man die vier Typen: Transform, Filter, Side-Effect und Branch.[16]
Hierbei handelt es sich jeweils um Unterklassen der Basispipe.

Gremlin.

Bei Gremlin handelt es sich um eine Sprache für die Verbindungen zwischen Graphen. Unterstützt werden Java und Groovy. Weitere Informationen (Syntax und Befehle) können auf *https://github.com/tinkerpop/gremlin/wiki* gefunden werden.

Frames.

Frames ist im Wesentlichen ein Objekt-zu-Graph-Mapper. Im Frame Interface werden Eigenschaften, oder Verbindungen zwischen Objekten durch @-Zuweisungen implementiert, z.B.: @Property.[17]
Das Interface definiert quasi einen Graphen(-abschnitt). Mit diesem kann interagiert werden.

Furnace.

"Furnace ist eine Sammlung von 'Graph-Algorithmen' für Blueprint-Graphen".[18] Mit diesen sollen komplexe Graphen in einfachere aufgespalten werden können. Z.B.: Multi-Relationale Graphen in Einfach-Relationale Graphen.[18]

Rexster.

Bei Rexster handelt es sich um einen Graph-Server. Sein Webservice liefert die Standardbefehle *GET*, *POST*, *DELETE* und *PUT*.[19]

4.2.2 MapDB

Ein weiteres Backend, das mit NeoEMF genutzt werden kann, ist MapDB. MapDB ist eine Datenbank-Engine für Java. Es stellt Maps und Collections zur Verfügung, ist selbst aber keine Datenbank.[20]
MapDB enthält zwei wichtige Klassen. Zunächst wäre da der DBMaker. Diese Klasse wird für die Einstellungen und die Verbindung zur Datenbank genutzt. Die DB-Klasse wiederum ist eine Instanz der Datenbank. Anders ausgedrückt wird durch den DBMaker die Verbindung zu einer Datenbank hergestellt; der Rückgabewert dieser Funktion ist dann

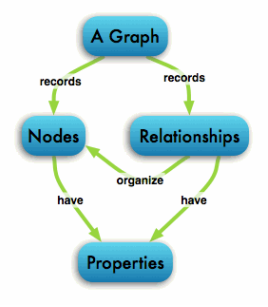

Abbildung 3: Eigenschaftsgraph[9]

ein Objekt der Klasse DB.[21]
DB stellt außerdem Methoden bereit um Collections zu öffnen / empfangen, zu erstellen oder diese zu schreiben (commiten).[21]
Die Vorgehensweise ist quasi analog zu der mit JDBC.
MapDB unterstützt verschiedene Performance-Modi. Standardmäßig sorgen die Einstellungen für Sicherheit und MapDB reagiert langsam. Diese Einstellungen können zu einer performanteren Lösung geändert werden, können gleichzeitig aber auch Memory Leaks [8] verursachen. Ein Kompromiss aus Konsistenz, Langlebigkeit und Geschwindigkeit sollte die besten Ergebnisse erzielen. Dafür werden verschiedene Speicherimplementierung angeboten auf die an dieser Stelle nicht weiter eingegangen werden soll (commit, disk sync, caches, ...).[23]

5. NEO4J

Die Unterstützung von Neo4j ist einer der wichtigsten Bestandteile von NeoEMF.
Neo4j arbeitet auf Eigenschaftsgraphen. Bei Eigenschaftsgraphen liegt das Hauptaugenmerk auf den Verbindungen zwischen Objekten. Die Verbindungen können ein oder mehrere Attribute / Eigenschaften besitzen, ebenso verschiedene Beziehungen zueinander, oder verschiedene Rollen einnehmen. Beziehungen haben hierbei die Besonderheit, dass sie gerichtet sein müssen. Abbildung 3 veranschaulicht einen solchen Graphen.[24]

In Graph-Datenbanken sind keine defekten Verbindungen erlaubt. Eine Beziehung muss immer einen Start- und Zielknoten haben. Wird einer der beiden Knoten gelöscht, so wird auch die zugehörige Beziehung gelöscht.[24]
Neo4j ist eine Open-Source NoSQL Graph-Datenbank, die größtenteils in Java[10] entwickelt wurde. Der Source Code von Neo4j kann in Github eingesehen werden. Das gilt auch für die anderen hier vorgestellen Software(-teile).[24]
Neo4j verzeichnet in seinem Bereich (NoSQL Graph-Datenbanken) die meisten Nutzer weltweit.[25]
Das Eigenschaftsgraphen-Modell wird von Neo4j bis zur Datenhaltungsebene implementiert. Ebenso bietet Neo4j Eigenschaften herkömmlicher Datenbanken, z.B. die Kompatibilität von *ACID Transaktionen* (vgl. 2 und 4.1.3).
Eine der beliebtesten Eigenschaften von Neo4j ist die zeitnahe Bearbeitung. Ohne lange Laufzeiten werden Beziehungen direkt nach der Erstellung sichtbar. Eine weitere sehr beliebte Eigenschaft (vorallem für die private Nutzung) ist die geringe Hardwareauslastung. Die Daten der Graphen werden kompakt vorgehalten, wodurch selbst mit schwacher Hardware Graphen mit sehr vielen Knoten genutzt und bearbeitet werden können. Kompakt vorgehalten bedeutet in diesem Kontext, dass selbst große Datenbestände wenig Speicher erfordern. Ebenso ist keine hohe Leistung für die Bearbeitung dieser Daten erforderlich.[24]
Von Neo4j gibt es zwei verschiedene Versionen. Zum einen gibt es die Community Edition, die gerade für den privaten Nutzer interessant ist. Diese Version ist kostenlos und beinhaltet die oben genannten Vorteile vollständig. Die Enterprise Version von Neo4j bietet noch ein wenig mehr und ist daher kommerziell besser geeignet. Unter anderem werden hier Live Backups möglich, die während der Laufzeit erstellt werden können, außerdem wird die Administration der Datenbestände durch das sogenannte "Advanced Monitoring" vereinfacht.[24]
Neo4j kommt, wie beispielsweise auch MySQL, mit eigenem Server. Mit diesem kann dann über eine Webschnittstelle interagiert werden.[24]

6. NEO4EMF

Neo4EMF ist die Vorgängerversion von NeoEMF. Infolgedessen soll diese hier nun auch etwas erläutert werden.
Ziel von Neo4EMF war es, mit großen Datenbeständen effizient auszukommen. Durch die zunehmende Modernisierung wurde es notwendig, ältere Software neu zu gestalten, bzw. aufzubereiten. Das beinhaltet natürlich einen großen Workflow, wobei ein solches Projekt meist zunächst durch Modelle abstrahiert wird, um die Planung zu vereinfachen, oder Probleme frühzeitig feststellen zu können. Das sorgt in der heutigen Zeit meist für sehr komplexe Modelle, die gehandhabt werden müssen. An dieser Stelle möchte man mit Neo4EMF aushelfen.[3]
EMF-Modelle sind aus zwei Gründen begrenzt:

- EMF-Modelle erlauben kein stückweises Laden der "Modellteile". Hier hilft der Lazy-Loading Mechanismus von Neo4EMF (siehe Kapitel 4.1.1), der das Laden von Modelldaten ermöglicht, genau in dem Moment wenn sie gebraucht werden. Dadurch wird auch die Begrenzung der Modellgröße durch die Speichergröße größtenteils behoben (vgl. 4.1.4).[3]
- Ein weiteres Problem wäre, dass Modelle von der Struktur prinzipiell Graphen sind, wobei die meisten Datenbestände aber auf relationalen Datenbanken aufbauen. Relationale Datenbanken sind nicht dazu gedacht Graphen zu speichern. Hierfür bietet Neo4EMF auch eine Lösung.[3]

Neo4EMF bietet zwei große Vorteile:

- Skalierbaren Zugriff auf sehr große Modelle, wobei Modellelemente geladen werden können, sobald diese benötigt werden. Das entspricht dem oben genannten Lazy-Loading Mechanismus.[3]
- Mit Neo4EMF können außerdem die Enterprise-Features von Neo4j ausgenutzt werden (z.B. das Nutzen von Live-Backups).

[8]"Ein Memory Leak ist der allmähliche Verlust des verfügbaren Computerspeichers, wenn ein Programm wiederholt keinen Speicher zurückgibt, den es für den temporären Einsatz erhalten hat."[22]
[9]Bild von *http://neo4j.com/developer/graph-database/* (zuletzt besucht am 06.02.2016)
[10]Das macht insofern Sinn, da mit Eclipse hauptsächlich Java entwickelt wird. Eclipse selbst basiert auch auf Java.

Für die Software-Entwicklung ist Persistenz ein wichtiger Faktor. Man möchte Objekte zu manchen Zeitpunkten persistent speichern. Um dies effizient zu lösen, liefert Neo4EMF einen *Persistence Layer*. Das ist zum Beispiel eine Datenbank-Abstraktion. Also ein Mechanismus, der Datenbankdetails versteckt und die Kopplung von Klassen zu den tatsächlichen Datenbeständen verringert, um auch auf großen Datenbeständen Persistenz zu garantieren.[3]
Zusammenfassend lässt sich sagen, dass Neo4EMF eben genau die oben genannten Probleme löst, sowie die genannten Vorteile mit sich bringt. Es liefert eine EMF API-kompatible Schicht, sowie ein graphenbasiertes Speichersystem. Ein sehr großer Vorteil ist, dass Neo4EMF auf Neo4j aufsetzt, das momentan eine sehr wichtige NoSQL-Datenbank ist.[3]
All diese Features sind auch in NeoEMF enthalten, wobei in NeoEMF noch einige hinzugekommen sind, beispielsweise der Multi-Backend Support.

7. FAZIT

EMF ist mittlerweile zum Standard geworden, was modellgetriebene Entwicklung angeht. Hierzu wird mit NeoEMF ein interessantes Tool geboten, welches gerade für die kommerzielle Nutzung bei großen Datenbeständen kaum wegzudenken ist. NeoEMF ist perfekt für zeitkritische Anwendungen, da der Fokus auf der Performance und der Skalierbarkeit im Umgang mit graphbasierten Daten liegt. Hierzu bietet NeoEMF eine gute Persistenzebene, die interessante Features mit sich bringt. Eine hohe Skalierbarkeit, mit gleichzeitig hoher Performance kann gut durch den Lazy Loading Mechanismus realisiert werden. Ebenfalls wird speicherfreundliches Caching angeboten, was allerdings in der heutigen Zeit keine große Seltenheit mehr ist und an dieser Stelle nicht als besonderes Feature verkauft werden sollte.
Neo4j ist momentan der wichtigste Vertreter für graphbasierte Datenbanken und erfreut sich an einer großen Community. Wie auch sein Vorgänger bietet NeoEMF eine hervorragende Unterstützung von Neo4j an. Dies gelingt auch dank des Multi-Backend Supports von NeoEMF. Für die Handhabung von Neo4j Datenbeständen bietet sich das Blueprints Backend an, welches auch durch eigene Vorteile hervorzuheben ist. Beispielsweise bietet NeoEMF für das Blueprints Backend eine Auto-commit Unterstützung, ebenso die Möglichkeit des Dirty Savings, wodurch die Performance auf einen sehr hohen Level gehoben wird.
Abschließend lässt sich sagen, dass NeoEMF für die modellgetriebene Entwicklung mehr als nur in Betracht gezogen werden sollte. Die Vorteile stechen klar hervor.
Die nahezu nicht vorhandene Dokumentation von NeoEMF ist allerdings ein kleiner Wermutstropfen. In Anbetracht der Aktualität (Vorstellung 2014-2015) von NeoEMF, und der Tatsache, dass es sich hierbei um ein OpenSource Tool handelt, sollte dieser Nachteil die Vorteile jedoch nicht überwiegen.

8. REFERENCES

[1] Neoemf. http://www.neoemf.com/. zuletzt besucht am 14.11.2015.

[2] Neoemf wiki. https://github.com/atlanmod/NeoEMF/wiki. zuletzt besucht am 14.11.2015.

[3] Amine Benelallam, Abel Gómez, Gerson Sunyé, Massimo Tisi, and David Launay. Neo4emf, a scalable persistence layer for emf models. In *Modelling Foundations and Applications*, pages 230–241. Springer, 2014.

[4] Neoemf. https://raweb.inria.fr/rapportsactivite/RA2014/atlanmod/uid49.html. zuletzt besucht am 10.12.2015.

[5] Non-relational database. https://www.techopedia.com/definition/25218/non-relational-database. zuletzt besucht am 05.02.2016.

[6] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010.

[7] Skalierbarkeit. https://de.wikipedia.org/wiki/Skalierbarkeit. zuletzt besucht am 17.12.2015.

[8] Prof. Dr. Udo Kelter. *Datenbanksysteme II - Skript*.

[9] Lazy loading. https://de.wikipedia.org/wiki/Lazy_Loading. zuletzt besucht am 14.12.2015.

[10] What is caching? https://www.citrix.de/glossary/caching.html. zuletzt besucht am 15.12.2015.

[11] Autocommit. https://en.wikipedia.org/wiki/Autocommit. zuletzt besucht am 15.12.2015.

[12] Persistence (computer science). https://en.wikipedia.org/wiki/Persistence_(computer_science)#Dirty_writes. zuletzt besucht am 16.12.2015.

[13] Blueprints. https://github.com/tinkerpop/blueprints. zuletzt besucht am 16.12.2015.

[14] Blueprints. https://github.com/tinkerpop/blueprints/wiki. zuletzt besucht am 16.12.2015.

[15] Pipes. https://github.com/tinkerpop/pipes/wiki. zuletzt besucht am 16.12.2015.

[16] Pipe types. https://github.com/tinkerpop/pipes/wiki/Pipe-Types. zuletzt besucht am 16.12.2015.

[17] Frames. https://github.com/tinkerpop/frames/wiki. zuletzt besucht am 16.12.2015.

[18] Furnace. https://github.com/tinkerpop/furnace/wiki. zuletzt besucht am 16.12.2015.

[19] Rexster. https://github.com/tinkerpop/rexster/wiki. zuletzt besucht am 16.12.2015.

[20] Mapdb: Database engine. http://www.mapdb.org/. zuletzt besucht am 16.12.2015.

[21] Db and dbmaker. http://www.mapdb.org/doc/db-and-dbmaker.html. zuletzt besucht am 16.12.2015.

[22] memory leak definition. http://searchwindowsserver.techtarget.com/definition/memory-leak. zuletzt besucht am 16.12.2015.

[23] Performance and durability. http://www.mapdb.org/doc/performance.html. zuletzt besucht am 06.02.2016.

[24] What is a graph database? http://neo4j.com/developer/graph-database/. zuletzt besucht am 17.12.2015.

[25] Why neo4j? top ten reasons. http://neo4j.com/top-ten-reasons/. zuletzt besucht am 17.12.2015.