

# Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

## Java wächst weiter



### Microservices

Schnell und einfach implementieren

### Container-Architektur

Verteilte Java-Anwendungen mit Docker

### Java-Web-Anwendungen

Fallstricke bei der sicheren Entwicklung



**ijug**  
Verbund

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977





Neues von den letzten Releases



Sich schnell einen Überblick über bestehende Strukturen und deren Beziehungen verschaffen

- |    |   |    |  |    |   |
|----|---|----|--|----|---|
| 3  | Editorial   | 28 | Weiterführende Themen zum Batch Processing mit Java EE 7<br><i>Philipp Buchholz</i>                  | 53 | Profiles for Eclipse – Eclipse im Enterprise-Umfeld nutzen und verwalten<br><i>Frederic Ebelshäuser und Sophie Hollmann</i> |
| 5  | Das Java-Tagebuch<br><i>Andreas Badelt</i>  | 34 | Exploration und Visualisierung von Software-Architekturen mit jQAssistant<br><i>Dirk Mahler</i>      | 57 | JAXB und Oracle XDB<br><i>Wolfgang Nast</i>   |
| 8  | Verteilte Java-Anwendungen mit Docker<br><i>Dr. Ralph Guderlei und Benjamin Schmid</i>    | 38 | Fallstricke bei der sicheren Entwicklung von Java-Web-Anwendungen<br><i>Dominik Schadow</i>          | 61 | Java-Enterprise-Anwendungen effizient und schnell entwickeln<br><i>Anett Hübner</i>   |
| 12 | JavaLand 2016: Java-Community-Konferenz mit neuem Besucherrekord<br><i>Marina Fischer</i> | 43 | Java ist auch eine Insel – Einführung, Ausbildung, Praxis<br><i>gelesen von Daniel Grycman</i>       | 66 | Impressum   |
| 14 | Groovy und Grails – quo vadis?<br><i>Falk Sippach</i>                                     | 44 | Microservices – live und in Farbe<br><i>Dr. Thomas Schuster und Dominik Galler</i>                   | 66 | Inserentenverzeichnis   |
| 20 | PL/SQL2Java – was funktioniert und was nicht<br><i>Stephan La Rocca</i>                   | 49 | Open-Source-Performance-Monitoring mit stagemonitor<br><i>Felix Barnsteiner und Fabian Trampusch</i> |    |   |
| 25 | Canary-Releases mit der Very Awesome Microservices Platform<br><i>Bernd Zuther</i>        |    |  |    |   |



Daten in unterschiedlichen Formaten in der Datenbank ablegen



# Exploration und Visualisierung von Software-Architekturen mit jQAssistant

Dirk Mahler, buschmais GbR

*Für die Arbeit in umfangreichen Code-Strukturen ist es sehr hilfreich, wenn sich Entwickler oder Architekten schnell einen Überblick über bestehende Strukturen und deren Beziehungen verschaffen können. Das Werkzeug jQAssistant leistet dabei wertvolle Unterstützung.*

Der Ausdruck „gewachsene Strukturen“ versinnbildlicht oft deutlich – ob mit oder ohne ironischen Unterton – den Zustand von Java-Anwendungen, die im Unternehmensumfeld über Jahre hinweg gediehen sind und viele Entwickler kommen, aber auch wieder gehen gesehen haben. Ein stets wiederkehrendes Problem für die im Projekt Verbliebenen ist die Frage, welche grundlegenden Struktur-Einheiten im Code existieren, wie diese zueinander in Beziehung stehen und in welchem Maß sie mit der Architektur-

Dokumentation (sofern vorhanden) beziehungsweise den Vorstellungen in den Köpfen der Entwickler (hoffentlich vorhanden) übereinstimmen.

Das Open-Source-Werkzeug jQAssistant bietet die Möglichkeit, bestehende Software-Strukturen zu erfassen und in einer Graphen-Datenbank abzulegen. Über Abfragen können diese Informationen um Konzepte (etwa Abstraktionen wie „Modul“) angereichert sowie Reports erzeugt werden, die in textueller oder grafischer Form

dem Nutzer Einblicke in den Status quo seiner Anwendung ermöglichen.

## *Eureka!*

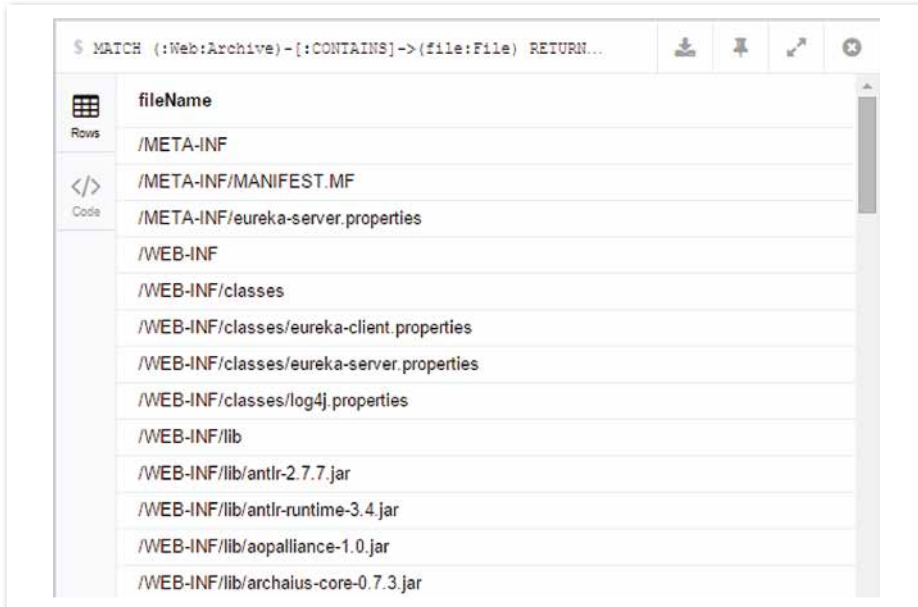
Das Unternehmen Netflix [1] stellt viele der von ihm entwickelten Lösungen unter Open-Source-Lizenzen zur Verfügung. Dazu zählt auch „Eureka“, ein Discovery-Dienst für Microservices, dessen Server als WAR-Artefakt via Maven-Central (groupId: com.netflix.eureka, artifactId: eureka-server) bezogen werden kann und analysiert werden soll. Die

```
bin jqassistant.sh scan -f eureka-server-1.4.2.war
```

Listing 1

```
MATCH (:Web:Archive)-[:CONTAINS]->(file:File) RETURN file.fileName as file-Name ORDER BY fileName
```

Listing 2

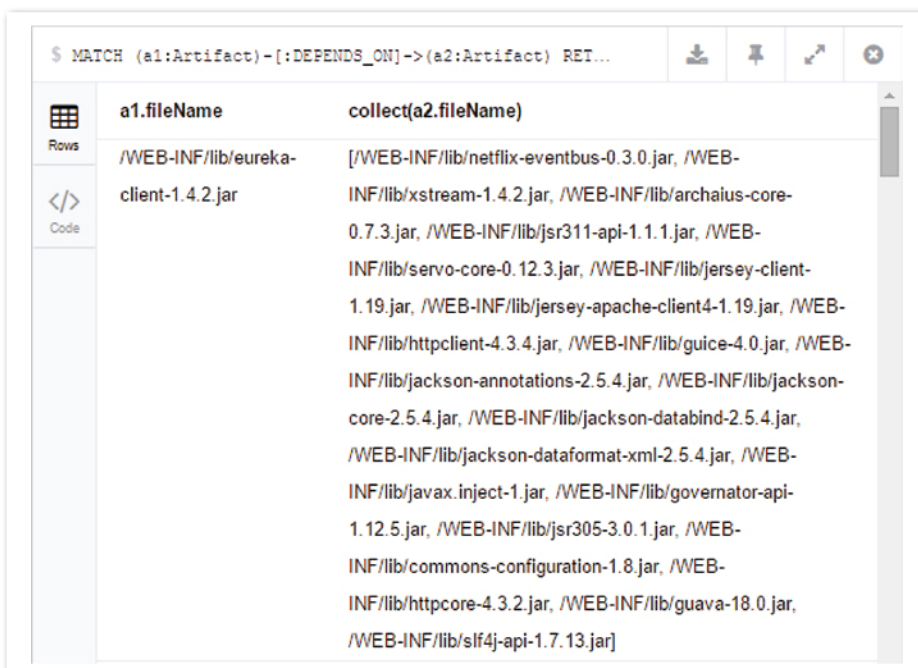


fileName
/META-INF
/META-INF/MANIFEST.MF
/META-INF/eureka-server.properties
/WEB-INF
/WEB-INF/classes
/WEB-INF/classes/eureka-client.properties
/WEB-INF/classes/eureka-server.properties
/WEB-INF/classes/log4j.properties
/WEB-INF/lib
/WEB-INF/lib/antlr-2.7.7.jar
/WEB-INF/lib/antlr-runtime-3.4.jar
/WEB-INF/lib/aopalliance-1.0.jar
/WEB-INF/lib/archaius-core-0.7.3.jar

Abbildung 1: Dateien im WAR-Archiv

```
MATCH (a1:Artifact)-[:DEPENDS_ON]->(a2:Artifact) RETURN a1.fileName, collect(a2.fileName)
```

Listing 3



a1.fileName	collect(a2.fileName)
/WEB-INF/lib/eureka-client-1.4.2.jar	[/WEB-INF/lib/netflix-eventbus-0.3.0.jar, /WEB-INF/lib/xstream-1.4.2.jar, /WEB-INF/lib/archaius-core-0.7.3.jar, /WEB-INF/lib/jsr311-api-1.1.1.jar, /WEB-INF/lib/servo-core-0.12.3.jar, /WEB-INF/lib/jersey-client-1.19.jar, /WEB-INF/lib/jersey-apache-client4-1.19.jar, /WEB-INF/lib/httpclient-4.3.4.jar, /WEB-INF/lib/guice-4.0.jar, /WEB-INF/lib/jackson-annotations-2.5.4.jar, /WEB-INF/lib/jackson-core-2.5.4.jar, /WEB-INF/lib/jackson-databind-2.5.4.jar, /WEB-INF/lib/jackson-dataformat-xml-2.5.4.jar, /WEB-INF/lib/javax.inject-1.jar, /WEB-INF/lib/governator-api-1.12.5.jar, /WEB-INF/lib/jsr305-3.0.1.jar, /WEB-INF/lib/commons-configuration-1.8.jar, /WEB-INF/lib/httpcore-4.3.2.jar, /WEB-INF/lib/guava-18.0.jar, /WEB-INF/lib/slf4j-api-1.7.13.jar]

Abbildung 2: Beziehungen zwischen JAR-Dateien

jQAssistant-Distribution ist auf der Projekt-Seite [2] als ZIP-Archiv verfügbar und nach dem Entpacken kann die WAR-Datei mit dem Kommando „cd jqassistant.distribution-1.1.3“ gescannt werden (siehe Listing 1).

Als Ergebnis existiert nun im aktuellen Arbeitsverzeichnis eine Ordnerstruktur „jqassistant/store“, die die Datenbank beinhaltet. Jetzt kann mit „bin/jqassistant.sh server“ der integrierte Neo4j-Server gestartet werden. Die Oberfläche steht im Browser unter der URL „http://localhost:7474“ zur Verfügung und ermöglicht erste Explorierungen mit Cypher [3], der Abfragesprache von Neo4j.

### Artefakte und ihre Beziehungen

Zunächst stellt sich die Frage, welche Dateien überhaupt im WAR-Archiv enthalten sind (siehe Listing 2 und Abbildung 1). Es ist erkennbar, dass das Archiv keinerlei Klassen, dafür jedoch JAR-Dateien enthält. Es wäre interessant zu wissen, welche Abhängigkeiten zwischen diesen JARs bestehen. Diese Information ist in den erfassten Daten jedoch nicht direkt vorhanden, kann aber über ein Konzept, also eine vorgefertigte Abfrage, angereichert werden. Dies wird nach Stoppen des Servers auf der Kommandozeile mit „<Enter>“ durch den Befehl „bin/jqassistant.sh analyze -concepts dependency:Artifact“ erreicht.

Das Konzept „dependency:Artifact“ wird mit dem Java-Plug-in von jQAssistant ausgeliefert und erzeugt eine Beziehung „DEPENDS\_ON“ zwischen zwei Artefakten „a1“ und „a2“ (etwa JAR-Dateien), wenn „a1“ einen Java-Typen beinhaltet, der eine Abhängigkeit zu einem Java-Typen in „a2“ besitzt. Nach erneutem Starten des Servers können diese Beziehungen über eine Abfrage ermittelt werden (siehe Listing 3 und Abbildung 2).

Die präsentierten Informationen sind zwar vollständig und korrekt, eine grafische Repräsentation wäre jedoch wesentlich anschaulicher. Hier kann das GraphML-Report-Plug-in von jQAssistant zum Einsatz kommen, um das Ergebnis eines Konzepts als GraphML-Datei zu exportieren, die wiederum durch Werkzeuge wie yEd [4] oder Gephi [5] visualisiert werden kann.

### Mehr Graph bitte!

Konzepte werden in XML- oder AsciiDoc-Dateien hinterlegt, für das Beispiel im letzteren Format. Hierzu wird relativ zum aktuellen Arbeitsverzeichnis eine Datei „jqassistant/



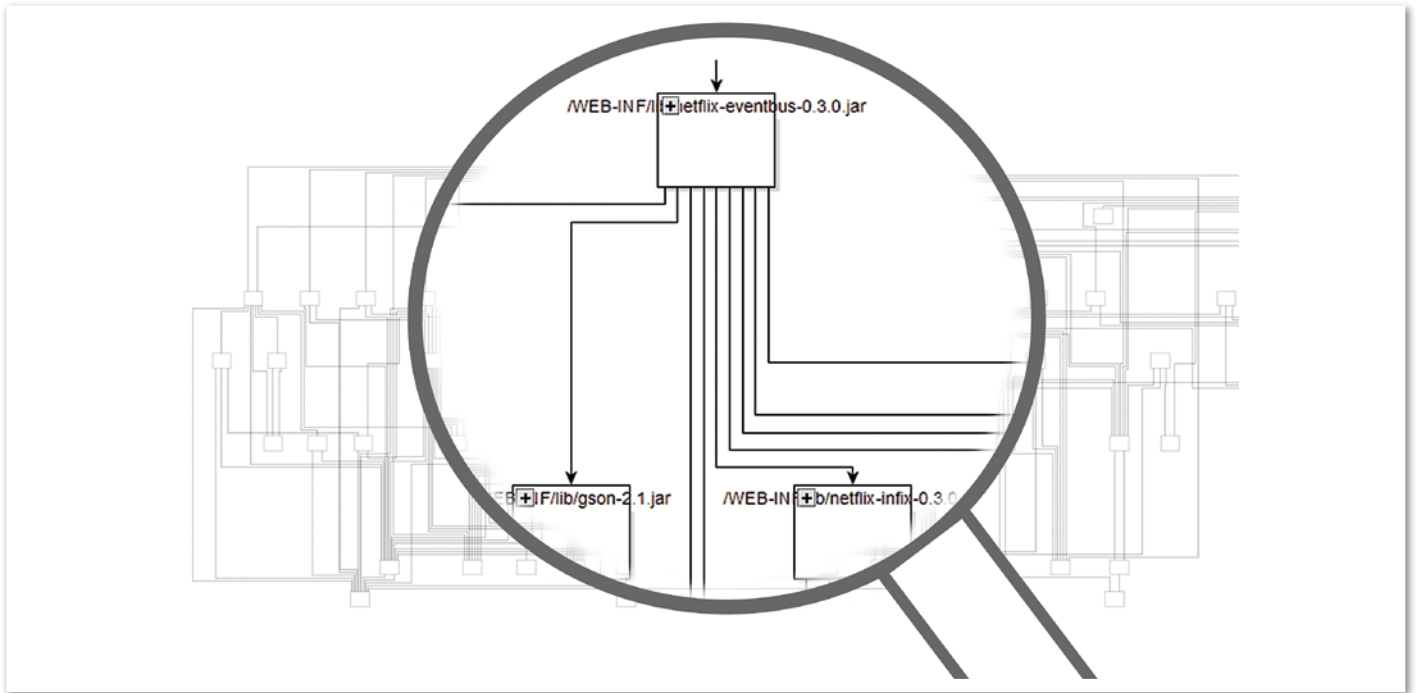


Abbildung 3: Visualisierung der Beziehungen zwischen JAR-Artefakten

rules/eureka.adoc“ [6] angelegt (siehe Listing 4).

AsciiDoc [7] ist eine einfach zu erlernende, aber gleichzeitig mächtige Markup-Language zum Erstellen von HTML- oder PDF-Dokumenten, die unter anderem von jqAssistant verwendet wird, um darin ausführbare Regeln wie Konzepte einzubetten. Im konkreten Beispiel handelt es sich um einen Cypher-Code-Block mit einer ID, einer kurzen Beschreibung sowie der Meta-Information, dass dieses Konzept die vorhergehende Ausführung des Konzepts „dependency:Artifact“ voraussetzt. Der „graphml“-Suffix der ID „artifactDependencies.graphml“ wird vom GraphML-Report-Plugin erkannt und als Hinweis darauf gewertet, dass das Ergebnis exportiert werden soll.

Der Aufruf von „bin/jqassistant.sh analyze -concepts artifactDependencies.graphml“ erzeugt im Verzeichnis „jqassistant/report“ eine Datei „artifactDependencies.graphml“, die mit yEd geöffnet werden kann und nach Anwendung eines hierarchischen Layouts die in Abbildung 3 gezeigte Visualisierung erzeugt.

Der Graph ist recht umfangreich und auch nur bedingt übersichtlich. Jedoch ist bereits erkenntlich, dass Artefakte mit ausgehenden Abhängigkeiten oben dargestellt werden (Netflix-Artefakte wie „eureka-core“ oder „netflix-eventbus“). Solche mit eingehenden Abhängigkeiten sind entsprechend weiter unten zu finden (hauptsächlich Bibliotheken und APIs wie „stax-api“ oder „http-core“).

```
= Eureka Example
== Artifact Dependencies
[[artifactDependencies.graphml]]
.Creates a GraphML report for artifact dependencies
[source,cypher,role=concept,requiresConcepts="dependency:Artifact"]
----
MATCH (:Web:Archive)-[:CONTAINS]->(artifact:Artifact)
OPTIONAL MATCH
  (artifact)-[dependsOn:DEPENDS_ON]->(:Artifact) // equivalent to left-outer-join in SQL
RETURN artifact, dependsOn
----
```

Listing 4

```
MATCH (:Web:Archive)-[:CONTAINS]->(artifact)-[:CONTAINS]->(package:Package)
RETURN artifact.fileName, collect(package.fqn)
```

Listing 5

```
[[internalArtifact]]
.Labels all artifacts containing the package "com.netflix" as "Internal".
[source,cypher,role=concept]
----
MATCH (:Web:Archive)-[:CONTAINS]->(artifact:Artifact)-[:CONTAINS]->(package:Package)
WHERE package.fqn = "com.netflix"
SET artifact:Internal
RETURN artifact
----

[[internalArtifactDependencies.graphml]]
.Creates a GraphML report for internal artifact dependencies.
[source,cypher,role=concept,requiresConcepts="dependency:Artifact,internalArtifact"]
----
MATCH (artifact:Artifact:Internal)
OPTIONAL MATCH (artifact)-[dependsOn:DEPENDS_ON]->(:Artifact:Internal)
RETURN artifact, dependsOn
----
```

Listing 6

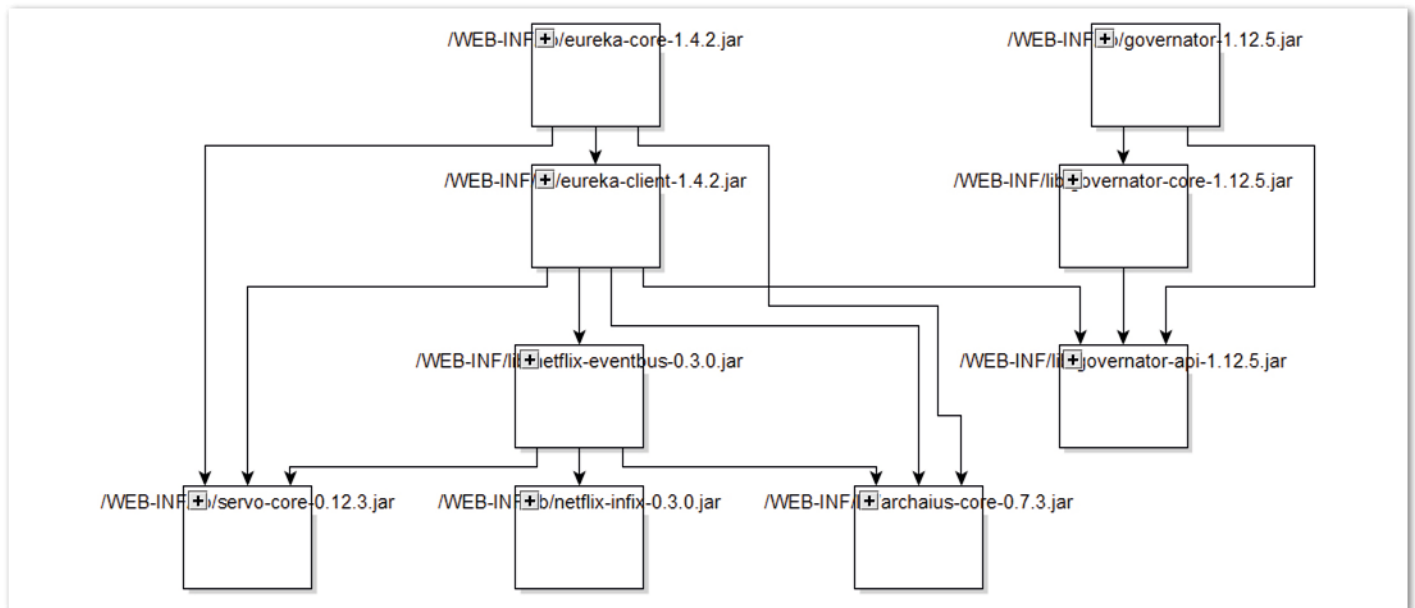


Abbildung 4: Visualisierung der Beziehungen zwischen internen Artefakten

### Weniger ist manchmal mehr

Letztere spielen für eine erste Analyse meist nur eine untergeordnete Rolle – die eigenen Artefakte und deren Abhängigkeiten sind oft von höherer Bedeutung. Es wird in diesem Zusammenhang oft von internen und externen Abhängigkeiten gesprochen (etwa im Kontext von Maven-Projekten). Doch wie können diese voneinander unterschieden werden?

Im Beispiel wird der Eureka-Server von Netflix untersucht – es ist naheliegend, dass interne Artefakte Package-Strukturen mit entsprechenden Namensmustern aufweisen. In der Tat ist es so, dass im Neo4j-Server eine Abfrage für Netflix-eigene Artefakte jeweils Packages mit dem Prefix „com.netflix“ liefert (siehe Listing 5). Mit diesem Wissen lassen sich in „eureka.adoc“ zwei weitere Konzepte erstellen (siehe Listing 6).

Das Konzept „internalArtifact“ versieht alle Artefakte, die ein Package mit dem Namen „com.netflix“ enthalten, mit dem Label „Internal“. Das darauf aufbauende Konzept „internalArtifactDependencies.graphml“ nutzt dieses, um die Abfrage für den GraphML-Report filtern zu können. Letzterer wird durch den Aufruf „bin/jqassistant.sh analyze -concepts internalArtifactDependencies.graphml“ erzeugt.

Wird die erzeugte Datei „jqassistant/report/internalArtifactDependencies.graphml“ in yEd geöffnet und das hierarchische Layout angewendet, ergibt sich *Abbildung 4*, die den Ansprüchen einer Architektur-Visualisierung bereits gerecht wird.

### Fazit

jqAssistant ermöglicht es, Artefakte zu scannen und über Abfragen zu untersuchen. Daraus können bei Bedarf textuelle oder graphische Reports („GraphML“) erzeugt werden. Dem Anwender wird dabei jegliche Freiheit gegeben, die Daten nach seinen individuellen Bedürfnissen anzureichern (wie interne vs. externe Abhängigkeiten) und die Ausgaben entsprechend zu filtern. Über das Anlegen sogenannter „Konzepte“ sind diese Vorgänge automatisierbar und können bei Bedarf auch in einen Build-Prozess integriert werden. Damit ist kontinuierliches Reporting über den aktuellen Zustand einer Architektur ermöglicht.

Das demonstrierte Vorgehen auf der Ebene von Artefakten stellt im Übrigen nur die sehr kleine Spitze des Eisberges dar. Es sind weitere Abhängigkeits-Analysen denkbar, etwa das Auffinden von Querschnitts-Aspekten (Logging, Injection) oder die Prüfung, welche Technologien überhaupt zum Einsatz kommen (etwa Hibernate oder JAX-RS) oder welche Auswirkungen das Anheben einer Bibliotheksversion haben könnte.

Bei Bedarf können die Abfragen auch auf Package- oder Klassenebene heruntergebrochen werden. Ein Beispiel dafür ist ein Report, der aufzeigt, welche Teile einer umfangreichen Bibliothek (etwa Guava) tatsächlich verwendet werden. Der Fantasie sind kaum Grenzen gesetzt.

### Links

- [1] Netflix: <http://netflix.com>
- [2] jqAssistant: <http://jqassistant.org>

- [3] Cypher: <http://neo4j.com/docs/stable/cypher-query-lang.html>
- [4] yEd: <https://www.yworks.com/products/yed>
- [5] Gephi: <https://gephi.org>
- [6] Gist: <https://gist.github.com/DirkMahler/8b5b8551de275cbe76c1>
- [7] AsciiDoc: <http://www.methods.co.nz/asciidoc>

Dirk Mahler

[dirk.mahler@buschmais.com](mailto:dirk.mahler@buschmais.com)



Dirk Mahler ist Senior-Consultant bei der buschmais GbR, einem Beratungshaus mit Sitz in Dresden. Der Schwerpunkt seiner mehr als zehnjährigen Tätigkeit liegt im Bereich Architektur und Entwicklung von Java-Applikationen im Unternehmensumfeld. Den Fokus setzt er dabei auf die Umsetzung von Lösungen, die im Spannungsfeld zwischen Pragmatismus, Innovation und Nachhaltigkeit liegen. In diesem Rahmen engagiert er sich für das Open-Source-Projekt jqAssistant.