service-service Dokumentation

Christian Brunzendorf; Frank Jäger; Michael Korta; Renko Ortgies

Inhaltsverzeichnis

1. Dokumentation . . . 1
1.1. Beschreibung . . . 1
1.2. Schnittstellenbeschreibung . . . 9
1.3. Prozessbeschreibung . . . 13
2. Architektur Dokumentation . . . 16
2.1. Architektur . . . 16
2.2. Architektur-Referenzen . . . 22
2.3. Modell . . . 22
3. HowTo . . . 22
3.1. Erstellen . . . 22
3.2. Konfiguration . . . 23
3.3. Starten . . . 25
3.4. Code . . . 25
3.5. Unit-Test . . . 26
4. Test . . . 27
4.1. Einleitung 27
4.2. Testkonzept 27
4.3. Testdurchführung 27
5. Anhang . . . 28
5.1. Get Started . . . 28
5.2. Variations . . . 28

Firma	PSI Software AG
Autoren	Christian Brunzendorf; Frank Jäger; Michael Korta; Renko Ortgies
E-Mail	info@psi.de
Datum	2019-02-15
Version	2.20.0

1. Dokumentation

1.1. Beschreibung

Der CIM-Cache ist eine Adapter-Software die Quellsysteme von Versorgern (z. B. für elektrische Energie) an ein nutzendes System (z. B. Einspeisemanagement, Abschaltmanagement,...) anschließt. Das System ist so flexibel, dass es verschiedene CIM (Common Information Model) und an CIM angelehnte Modelle (z. B. CGMES) versteht. Auch ohne ein nutzendes System (Endanwendersoftware) kann der CIM-Cache direkt über einen Browser abgefragt werden.

Technisch besteht der CIM-Cache aus mehreren (SPRING)-Komponenten (Microservices) die auch auf mehrere Systeme verteilt installiert werden können. Die Komponenten sind zum Teil voneinander abhängig (gemeinsame Basiskomponenten) und bauen aufeinander auf (z. B. über gemeinsame Datenmodelle). Es existieren sowohl quelloffene (open source) als auch proprietäre (Erweiterungs-)Komponenten. Einen Überblick über die "open source" Komponenten bietet folgende Übersicht.

"open source" Komponenten (org-eclipse)

domain (Basiskomponenten)

- common
- service
- build, build-codeconventions
- cim17v07
- profile-openkonsequenz

domain (CIM-Cache -Services)

- assetdata
- dynamictopology
- measurement
- statictopology
- topologystate

source-system (Quellsystem -Services)

- mockupassetdata

- mockupmeasurement
- mockupstatictopology
- mockuptopologystate

In den jeweiligen "domain" und "source-system" Diensten werden folgende allgemeine Funktionalitäten bereitgestellt:

- Import von Modellen oder Referenzmodellen aus Quellsystemen über REST-Services (POST command "import")
- Bereitstellung von REST-Services zum Abrufen (POST command "reset", "refresh", "add", "clear") oder Laden von Modellen (POST command mit gültigem Modell im RDF-Format)

1.1.1. Anforderungen des QC-Handbooks

Gemäß des QC-Handbooks müssen die Qualitätsanforderungen eines jeden oK-Projektes oder -Moduls in der Dokumentation der Architektur festgehalten werden. Hierbei verweist das QC-Handbook auf den Standard DIN/ISO 25010, welcher als Referenzrahmen zur Bewertung der Qualitätsattribute herangezogen werden soll. Jedes individuelle Projekt besitzt gleichermaßen seine individuellen Ansprüche an Qualität. Für das Projekt CIM-Cache 2.0 - Stufe 2 wurden zu Beginn des Projektes folgende Aspekt des DIN/ISO 25010 in den Fokus der Betrachtung gerückt.

1.1.2. Maßnahmen zur Einhaltung der Qualitätsanforderungen im CIM-Cache 2.0 - Stufe 2

- Performanz
  - Zeitverhalten
    An den CIM-Cache 2.0 werden keine kritischen Performanzanforderungen gestellt. Des Weiteren ist die Performanz in erster Linie von den Quellsystemen abhängig, da diese den CIM-Cache mit Daten beliefern.
- Kompatibilität
  - Interoperabilität
    Der CIM-Cache 2.0 ermöglicht die Anbindung beliebiger Quellsysteme und datennutzender Anwendungen wie bspw. das Abschaltmanagement und das Einspeisemanagement über offene und standardisierte Schnittstellen .
- Zuverlässigkeit
  - Fehlertoleranz
    Um einen unbestimmten Status des CIM-Cache 2.0 durch Fehler zu vermeiden, werden alle zu erwartenden Fehler abgefangen und in einer Logdatei protokolliert. Dadurch wird ein hoher Grad an Fehlertoleranz erreicht.
  - Wiederherstellbarkeit
    Der CIM-Cache 2.0 ist ein rein lesendes System, das dessen Daten aus Quellsystemen bezieht. Nach einem ungeplanten Systemende können alle Daten aus den Quellsystemen neu eingelesen werden.
- Sicherheit

- Nachweisbarkeit
  Verarbeitungsschritte werden in Logdateien protokolliert. Diese können zur Analyse der Nachweisbarkeit verwendet werden.

- Wartbarkeit
  - Modularität
    Die in der Architektur des CIM-Cache 2.0 konsequent durchgeführtes Bereitstellung von diskreten Services stellt sicher, dass Änderungen an einem Service sich nicht auf andere bereitgestellte Services auswirken. Ausgenommen sind Services, die aus Komponenten anderer Services bestehen (z. B. Dynamic-Topology).
  - Wiederverwendbarkeit
    Mit dem CIM-Cache 2.0 wird ein Framework bereitgestellt, welches es unterschiedlichen Anwendungen ermöglicht seine Dienste über standardisierte Schnittstellen zu nutzen. Die Schnittstellen zum CIM-Cache 2.0 sind allgemein formuliert und nicht speziell für einzelne Anwendungen modelliert. Damit ist ein hoher Grad an Wiederverwendbarkeit erreicht. Die Basiskomponenten aus dem Service-Framework bieten einen standardisierten Programmablauf für den jeweiligen Anwendungsfall. Das Ergebnis ist eine Gleichartigkeit im Aufbau und in der Funktionalität aller Services.
  - Analysierbarkeit
    Der CIM-Cache 2.0 ist ein rein lesendes System. Über die standardisierten Schnittstellen können Fehler in den Daten der angebundenen Quellsysteme diagnostiziert werden. Fehler bei der Kommunikation mit den Schnittstellen und bei der Datenverarbeitung im CIM-Cache 2.0 werden in den bereitgestellten Logdateien protokolliert.
  - Modifizierbarkeit
    Aufgrund der Modularität der gewählten Architektur des CIM-Cache 2.0 sind die einzelnen Services insoweit modifizierbar, dass andere Services des CIM-Cache 2.0 nicht beeinträchtigt werden. Unter Anderem wird dies durch die Versionierung sichergestellt. Die unter einer Release-Version veröffentlichten Softwareartefakte sind ausgiebig getestet und auf Kompatibilität geprüft.
  - Testbarkeit
    Der CIM-Cache 2.0 kann sowohl über ein Referenzmodell als auch über eine Schnittstelle für individuelle Testmodelle getestet werden. Die Schnittstellen sind als REST-Services bereitgestellt, die über einen Internet-Browser abgefragt werden können.

- Übertragbarkeit
  - Anpassbarkeit
    Durch die Bereitstellung der unterschiedlichen Services des CIM-Caches als Java Archiv Dateien, ist der Betrieb sowohl auf den Betriebsystem Windows als auch Linux möglich.
  - Installierbarkeit
    Die Installierbarkeit wird durch die Bereitstellung von Java Archiv Dateien, die zusammen mit Konfigurationsdateien in einer Verzeichnisstruktur liegen, gewährleistet. Ausser den benannten Dateien ist nur eine Java Laufzeitumgebung notwendig. Es werden keinerlei zusätzliche Bibliotheken benötigt.
  - Austauschbarkeit
    Durch die konsequente Verwendung von Standards (z. B. IEC-61968, IEC-61970) für die implementierten Schnittstellen ist ein hoher Grad an Austauschbarkeit erlangt.

- Funktionale Tauglichkeit
  - Vollständigkeit
    Die Vollständigkeit wird dadurch sichergestellt, dass für die aus den Anforderungen erzeugten Prozesse der Anwendungsfälle die benötigten Services und Methoden bereitgestellt werden. Mit den implementierten Services und Methoden des CIM-Caches können die beschriebenen Anwendungsfälle vollständig erfüllt werden.

1.1.3. Verwendete externe Bibliotheken

SLF4J

1. slf4j-api (Version: 1.7.25)

logback

1. logback-classic (Version: 1.2.3)

org.apache

1. commons-csv (Version: 1.4)
2. commons-lang3 (Version: 3.6)

org.eclipse.persistence

1. org.eclipse.persistence.jpa (Version: 2.7.1)

apache commons

1. commons-io (Version: 2.5)

mockito

1. mockito-core (Version: 2.9.0)
2. mockito-inline (Version: 2.9.0)

org.springframework

1. springweb (Version: 4.3.10.RELEASE)
2. spring-boot-starter-data-rest (Version: 1.5.6.RELEASE)
3. spring-boot-starter-web (Version: 1.5.6.RELEASE)
4. spring-boot-starter-hateoas (Version: 1.5.6.RELEASE)
5. spring-boot-starter-actuator (Version: 1.5.6.RELEASE)
6. spring-boot-starter-test (Version: 1.5.6.RELEASE)

org.assertj

1. assertj-core (Version: 3.8.0)

hamcrest

1. hamcrest-library (Version: 1.3)

junit

1. junit (Version: 4.12)

xml-unit

1. xmlunit (Version: 1.6)
2. xmlunitmatchers (Version: 2.4.0)

yaml

1. snakeyaml (1.17)

no maven-project-info

1. maven-project-info-reports-plugin (Version 2.9)

1.1.4. Benutzung

Das Service-Framework wird von den fachlichen Komponenten (Dynamic-Topology, Static-Topology, Topology-State, Measurement und Asset-Data) benutzt. Die Kommunikation der einzelnen Komponenten untereinander, sowie zu Quellsystemen und Endanwendersoftware ist über die REST-Technologie (Representational State Transfer) realisiert.

1.1.5. Administration

Administration und Installation vom CIM-Cache

Derzeit besteht der CIM-Cache nur aus dem TopologieDatenManagement, weitere Inhalte folgen bald.

Voraussetzungen

Hardware

Die Anforderungen an die Hardware für den Rechner für Serverkomponenten hängen von verschiedenen Rahmenbedingungen ab. Mit zunehmender Leistung der Hardware wird ein besseres Laufzeitverhalten erreicht.

Prozessor	Intel Core i7-7700 CPU Prozessor, mind. 4x 3.60GHz
RAM	mind. 16 GB
Festplattenspeicher	mind. 20 GB
Internetverbindung	Ja

Software

Folgende Installationen und Konfigurationen wurden gemäß Dokumentation ausgeführt.

Betrieb

64-Bit-Betriebssystem	• Windows Version 10 oder • Ubuntu Server 16.04.3 LTS
Oracle Java Runtime Environment	http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html Version 8
aktueller Browser	• https://www.apple.com/de/safari/ apple safari • https://www.google.de/chrome google chrome • https://www.mozilla.org Mozilla Firefox

Erstellen

64-Bit-Betriebssystem	• Windows Version 10 oder • Ubuntu Server 16.04.3 LTS
Oracle Java SE Development Kit	https://java.com/de/download Version 8
Apache Maven Umgebung	https://maven.apache.org/download.cgi Version 3.5
aktueller Browser	• https://www.google.de/chrome google chrome • https://www.mozilla.org Mozilla Firefox
Internet-Zugriff	https://repo.maven.apache.org Maven-Repository

Die für das Projekt notwendigen Sourcen (veröffentlichte Quelltexte) können über das https://projects.eclipse.org/projects/technology.openk-platform/developer eclipse-git bezogen werden.

Projektstruktur

Die Projektverzeichnisse und Dateien befinden nach dem Entpacken bzw. Herunterladen auf dem lokalen Rechner in einem gemeinsamen Verzeichnis (z. B. "D:\demo\git.openk\cim-cache"), das im Folgenden als "ROOT_PATH" bezeichnet wird:

org.eclipse.openk.build
org.eclipse.openk.cim.cim17v07
org.eclipse.openk.cim.profile.openkonsequenz
org.eclipse.openk.common
org.eclipse.openk.domain.dynamic-topology
org.eclipse.openk.domain.static-topology
org.eclipse.openk.domain.topology-state
org.eclipse.openk.service
org.eclipse.openk.source-system.mock-up-static-topology
org.eclipse.openk.source-system.mock-up-topology-state

gitignore

pom.xml

Es sind die folgenden Quelltexte und Definitionen enthalten:

- der CIM-Standard in der Version 17.07
- die Domänen-Module des TopologieDatenManagements
- die Quellsystem-Module der Mock-Up-Topology
- gemeinsam genutzte Basis- und Servicefunktionalität
- eine Hilfsdatei für die Versionsverwaltung Git (".gitignore")
- die notwendigen Definitionen zum automatisierten Kompilieren und Testen der Quellen ("pom.xml")

Kompilieren des CIM-Caches auf der Basis der veröffentlichten Quelltexte

1. Innerhalb der Konsole in das Verzeichnis "ROOT_PATH" wechseln:

```
$ cd demo\git.openk\cim-cache
```

2. Anschließend den Bau des CIM-Caches mit Hilfe des Tools "Maven" starten:

```
$ mvn clean install
```

Auf dieser Basis führt Maven folgende Schritte durch:

- Auflösen der zum Bauen notwendigen Informationen
- Herunterladen der externen Abhängigkeiten
- Durchführung der statischen Code-Analyse
- Bauen der einzelnen Module in der festgelegten Reihenfolge
- Durchführung von Unit- und Integrationstest

Starten des CIM-Caches

Der Start des CIM-Caches setzt die Verfügbarkeit der Quellsysteme voraus. Für Tests können hier Mock-Up-Static-Topology und Mock-Up-Topology-State als Quellsystem-Mock-Up verwendet werden.

Dabei ist zu berücksichtigen, dass alle notwendigen Applikationen auf unterschiedlichen Ports laufen. Diese lassen sich, wie im Abschnitt Prüfen des CIM-Cache-Start durch einen Service-Aufruf beschrieben, konfigurieren.

Starten der Mock-Up-Static-Topology bzw. der Mock-Up-Topology

1. Innerhalb der Konsole in das folgende Verzeichnis wechseln:

```
$ cd ROOT_PATH/org.eclipse.openk.source-system.mock-up-static-topology/service
```

2. Das Quellsystem-Mock-Up mit dem folgenden Befehl starten (Windows):

```
$ java -jar target\mock-up-static-topology-service-<version>.jar
```

3. Das Quellsystem-Mock-Up mit dem folgenden Befehl starten (Linux):

```
$ nohup java -jar target/mock-up-static-topology-service-<version>.jar > mock-up-
static-topology.log 2>&1 &
```

Starten des TopologieDatenManagement

1. Innerhalb der Konsole in das folgende Verzeichnis wechseln:

```
$ cd ROOT_PATH/org.eclipse.openk.domain.dynamic-topology/service
```

2. Den CIM-Cache mit dem folgenden Befehl starten (Windows):

```
$ java -jar target\dynamic-topology-service-<version>.jar
```

3. Den CIM-Cache mit dem folgenden Befehl starten (Linux):

```
$ nohup java -jar target/dynamic-topology-service-<version>.jar > dynamic-
topology.log 2>&1 &
```

Beenden der Applikation

Die Applikation wird beendet, indem die Konsole(n), in der die Applikation(en) gestartet wurden,

geschlossen werden.

Prüfen des CIM-Cache-Start durch einen Service-Aufruf

WARNUNG: Die folgende Beschreibung gilt für eine lokale Installation.

Zur Überprüfung des Starts des CIM-Caches kann im Browser die folgende URL aufgerufen werden: topologische Elemente Das Ergebnis ist die Mock-up-Topologie als RDF formatiertes XML-Dokument.

Log- und Konfigurationsdateien

Die Anwendung wird durch die Konfigurationsdatei application.properties angepasst. Folgende Parameter können hier konfiguriert werden:

- server-Port
- logging-level

Die Datei befindet sich im Verzeichnis:

`ROOT_PATH/org.eclipse.openk.domain.dynamic-topology.service/src/main/resources/`

Die Konfiguration dieser Datei können Sie beim Start der Anwendung überschreiben. Dazu fügen Sie dem Befehl zum Starten der Applikation folgenden Parameter hinzu und verweisen damit auf eine alternative Konfigurationsdatei mit den gewünschten Einstellungen:

```
$ java -jar dynamic-topology-service.jar
--spring.config.location=./Configuration/application.properties
```

Konfiguration

Sie können den CIM-Cache mit Hilfe der Datei dynamic-topology-service.config konfigurieren. Die Datei befindet sich im Verzeichnis:

`ROOT_PATH/org.eclipse.openk.domain.dynamic-topology.service`

1.2. Schnittstellenbeschreibung

Die Kommunikation mit dem service-service erfolgt über REST-Schnittstellen, welche im OpenKonsequenz-Wiki veröffentlicht sind.

Von diesem Service angebotene Schnittstellen.

internes Modell abfragen
Gibt den Inhalt des internen Modells (Repository) mit der angegeben Kennung als RDF-Dokument zurück.

Beschreibung	Wert
http-Methode	GET
Bezeichnung	/model
aktuelle Version	1.0.0
Sprache (ISO 639-1)	de
Spezifikation	service.model_GET_1_de.open-api.yaml

Status-Code	Beschreibung
200	Das interne Modell als RDF-Dokument.
400	Die Schnittstelle wurde mit ungültigen Parametern aufgerufen.
404	Die interne Modell-Kennung ist unbekannt.
500	Es trat ein allgemeiner Fehler bei der Verarbeitung der Anfrage auf.

internes Modell aktualisieren
Aktualisiert den Inhalt des internen Modells (Repository) mit der angegeben Kennung.

Beschreibung	**Wert**
http-Methode	PUT
Bezeichnung	/model
aktuelle Version	1.0.0
Sprache (ISO 639-1)	de
Spezifikation	service.model_PUT_1_de.open-api.yaml

Status-Code	**Beschreibung**
200	Die Antwortnachricht.
400	Die Schnittstelle wurde mit ungültigen Parametern aufgerufen.
500	Es trat ein allgemeiner Fehler bei der Verarbeitung der Anfrage auf.

Service-Kommando ausführen
Führt das angegebene Service-Kommando aus.

Beschreibung	**Wert**
http-Methode	POST
Bezeichnung	/execute
aktuelle Version	1.0.0
Sprache (ISO 639-1)	de
Spezifikation	service.execute_POST_1_de.open-api.yaml

Status-Code	**Beschreibung**
200	Die Antwortnachricht.
400	Die Schnittstelle wurde mit ungültigen Parametern aufgerufen.
404	Das angegebene Service-Kommando wird nicht unterstützt.
500	Es trat ein allgemeiner Fehler bei der Verarbeitung der Anfrage auf.

1.3. Prozessbeschreibung

Alle "domain" und "source-system" Services bieten zwei allgemeine Schnittstellen an. Damit lassen sich Daten in den CIM-Cache importieren und exportieren oder Informationen über den CIM-Cache abfragen.

Der Prozess 01 ist ein Beispiel für eine grundlegende Funktion: Filtern und sortieren einer Entität (ein bestimmtes Element im Netzgebiet).

Der Prozess 02 aktualisiert das Referenzmodell. Je nach Service werden die Befehle "reset" (Neuaufbau des Referenzmodells), "refresh" (Aufbau des Referenzmodells mit z. B. neuen Zuständen oder Messwerten), "add" (z. B. Hinzufügen von Messwerten) und "clear" löschen des Modells unterstützt.

Der Prozess 03 führt ein Kommando aus. Damit können von Quellsystemen Daten in den CIM-Cache importiert werden oder ein Referenzmodell neu aufgebaut oder modifiziert werden (POST command).

Die Prozesse 04 (GET command) und 05 (POST command) fragen das jeweilige aktuelle Modell (z. B. die statische Topologie, den aktuellen Zustand einzelner Topologieelemente, Messwerte wie die aktuelle Leistung eines Einspeisers, sowie Stammdaten) aus dem Quellsystem oder dem CIM-Cache ab, bzw. laden dieses in das jeweilige System (Service).

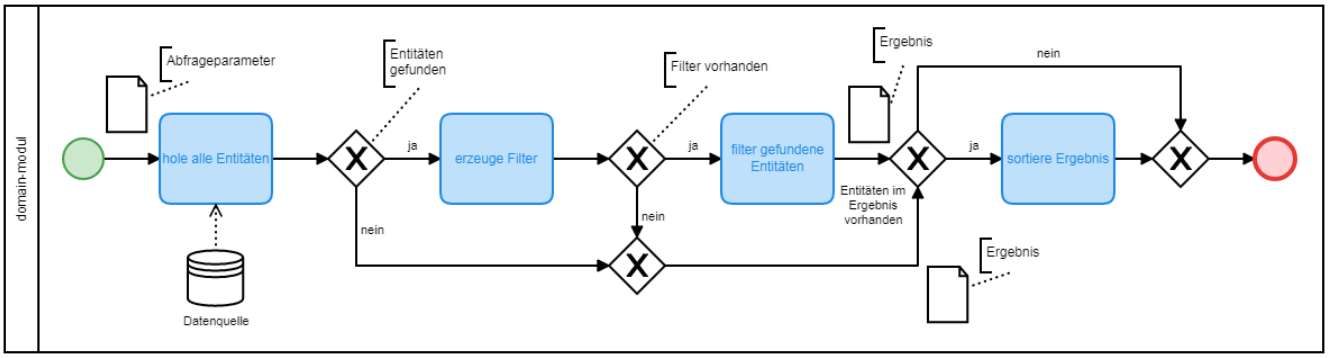

Figure 1. Entitäten filtern und sortieren

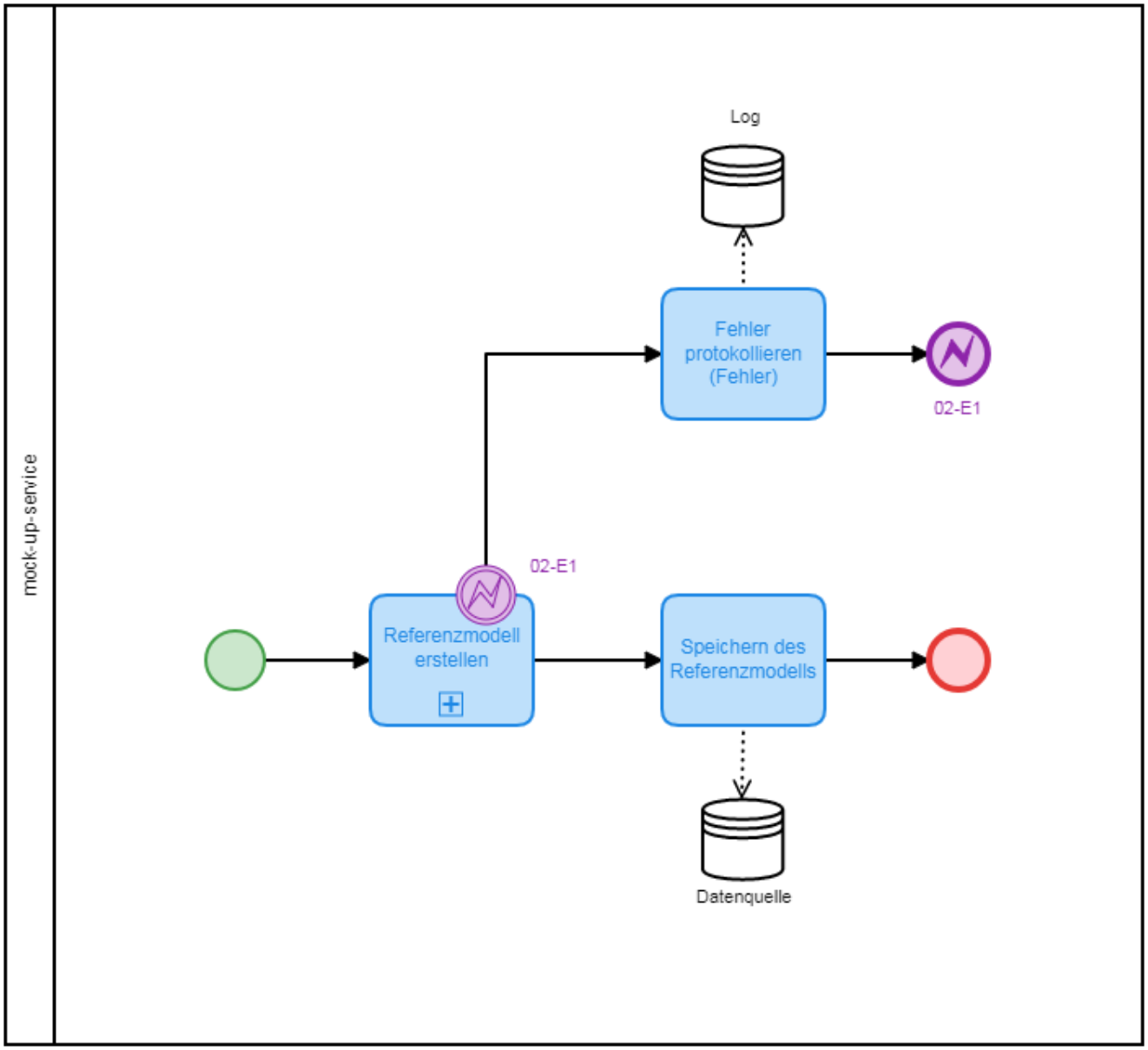

Figure 2. Referenzmodell aktualisieren

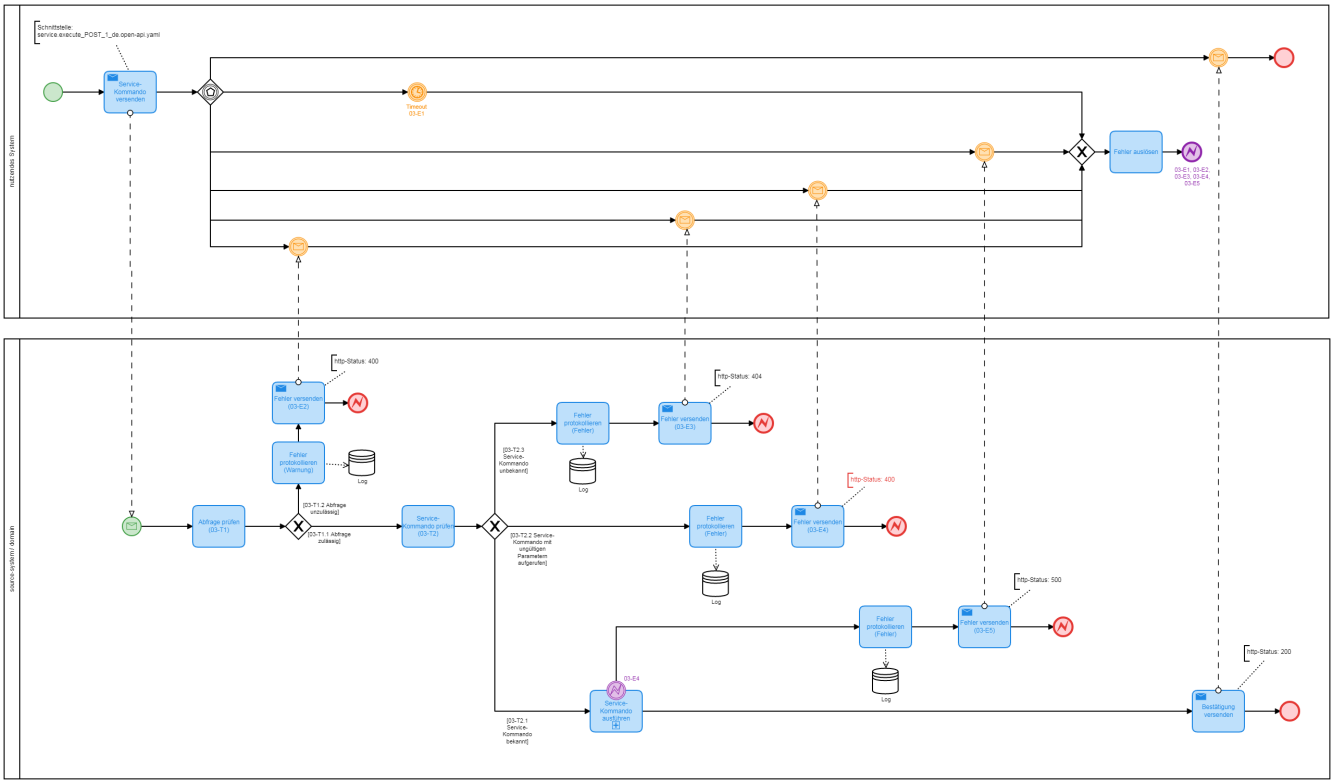

Figure 3. Service Kommando ausführen

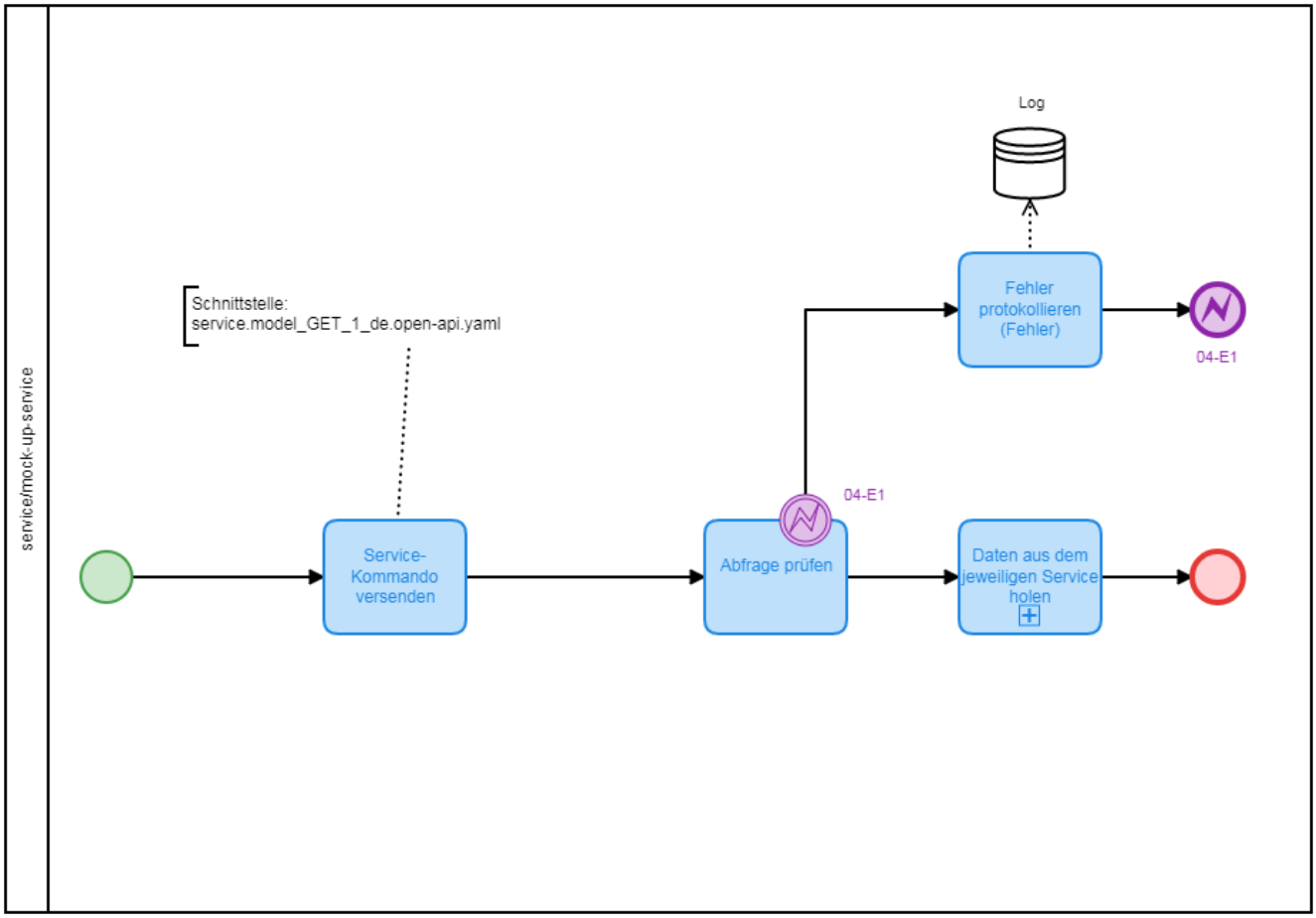

Figure 4. Modell abfragen

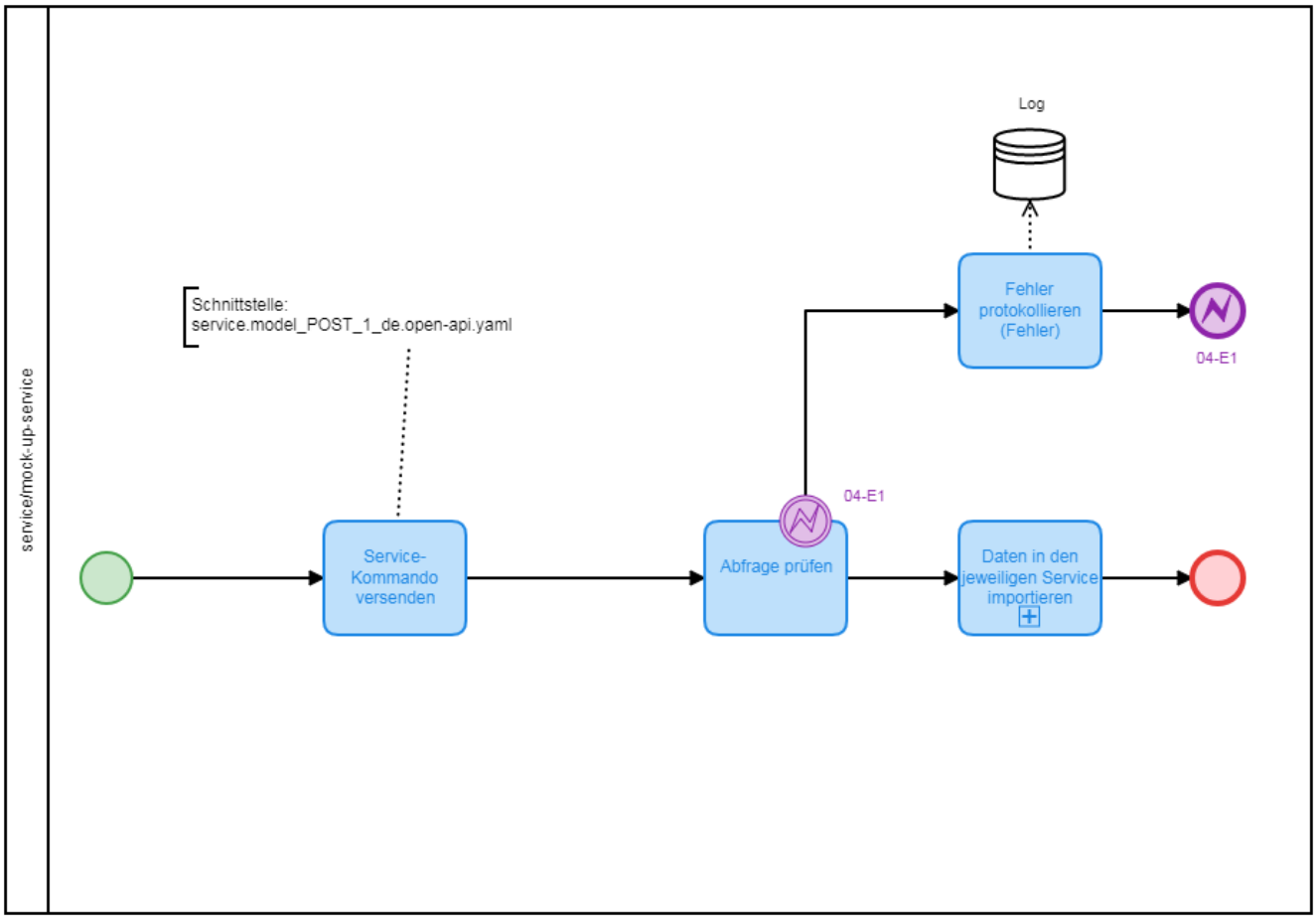

Figure 5. Modell laden

2. Architektur Dokumentation

2.1. Architektur

2.1.1. Microservices mit spring.boot Entscheidung

Im "openKONSEQUENZ Architecture Committee Handbook" werden verschiedene Anforderungen hinsichtlich Flexibilität, Erweiterbarkeit, Verfügbarkeit, Gebrauchstauglichkeit, Integrierbarkeit und Sicherheit beschrieben (Kapitel 1.2 - Qualitätsziele). Eine Microservice-Architektur ist ideal geeignet, um diese Anforderungen zu erfüllen. Die Idee dieses Baustils ist die Schaffung autonomer Dienste, die im Idealfall jeweils für eine isolierte Aufgabe verantwortlich sind. Der Vorteil dieser Architektur ist, dass Dienste unabhängig voneinander (auch in verschiedenen Versionen und Erweiterungsebenen) entwickelt und flexibel in die Zielplattform integriert werden können. Microservices verwenden eine gemeinsame Infrastruktur, die verschiedene "Kernmodule" bereitstellt (siehe Kapitel 4.9 des AC-Handbooks Architecture Committee Handbook @ openKONSEQUENZ). Die Integration des Microservice in die Infrastruktur zeigt das folgende Diagramm:

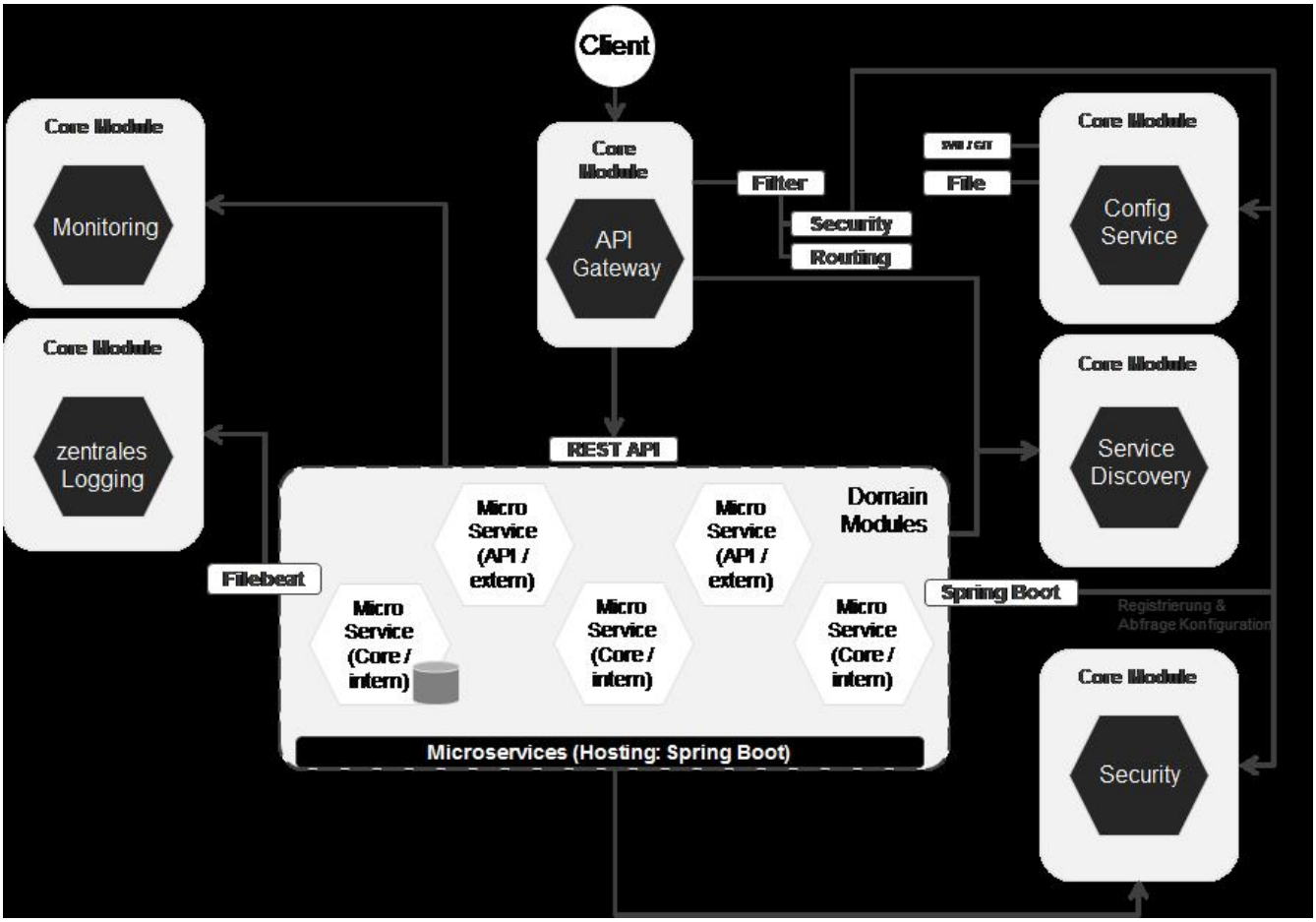

Figure 6. Einsatz von Microservices (High Level)

Derzeit sind noch nicht alle Technologien für die Kernmodule festgelegt / implementiert. Dennoch erfolgt hier eine Beschreibung der Vorteile. Die Stärke dieses Architekturstils liegt in der Verteilung und Skalierung der Microservices. Perspektivisch greift der Benutzer (Client) auf die einzelnen Microservices über ein zentrales Gateway namens "API Gateway" zu, über das die Autorisierung durchgeführt wird. Das API-Gateway kann auch eine Verteilung der Aufrufe für andere Instanzen durchführen und im Fehlerfall ein Failover durchführen. Bei der Durchführung z. B. eines Versionswechsel können Routing-Regeln festgelegt werden. Eine Service Discovery wird für die Registrierung von Microservice-Instanzen verwendet. Die Konfiguration des Microservice erfolgt ebenfalls an zentraler Stelle, um eine Trennung zwischen Konfiguration und Programmcode zu erreichen. Dadurch können die einzelnen Instanzen wesentlich flexibler in den verschiedenen Umgebungen verteilt werden. Spring.boot wird als Hosting-Technologie für die Erstellung von Microservices verwendet. JavaEE wird nicht als Hosting-Technologie verwendet, da die Erstellung und Verteilung kleiner Dienste mithilfe von JavaEE wesentlich komplexer ist, als bei der Erstellung eines leichten, auf spring.boot basierenden Mikroservice. Darüber hinaus bietet der gewählte Ansatz wesentlich flexiblere Optionen für den Betrieb dieser Dienste in Cloud-Umgebungen vor Ort oder als Managed Service in der Zukunft.

2.1.2. Interne Softwarestruktur für die Entscheidung für Domänenmodule

Um die Domain-Module "Static-Topology", "Topology-State", "Dynamic-Topology", "Asset-Data" und "Measurement" zu realisieren, wurde eine logische Softwarestruktur ausgewählt, die im Wesentlichen aus folgenden Komponenten besteht:

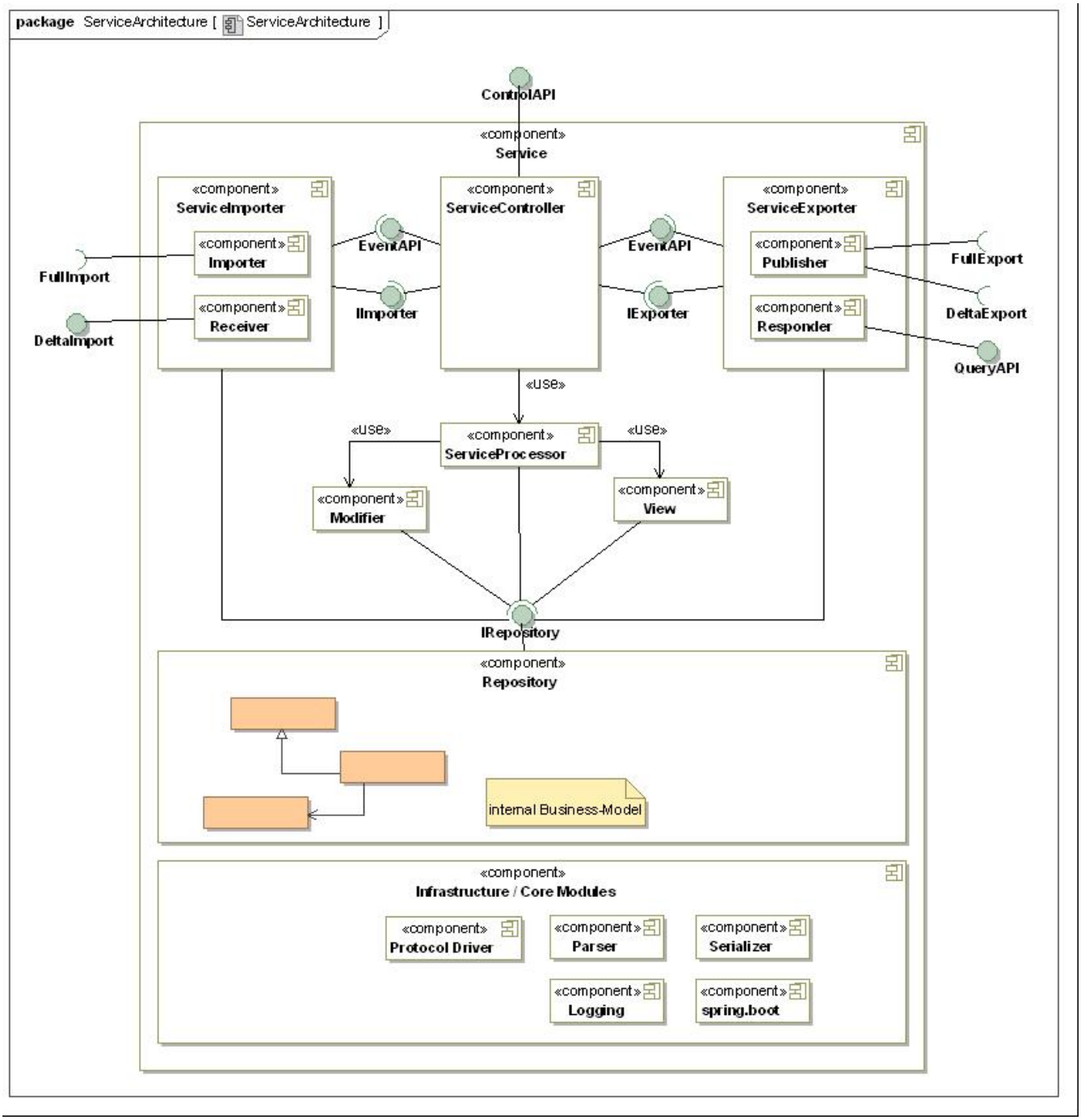

Figure 7. Logische Softwarestruktur

Service-Importer implementieren die Quellsystem-APIs für den Import von Daten aus den Quellsystemen. Diese Daten werden dann in das interne Repository (internes spezifisches Strukturmodell) übertragen. Der Zugriff auf das interne Modell erfolgt entweder direkt über den Service-Importer, den Service-Exporter oder über dedizierte Modifikatoren (Modifier) für Schreibzugriff oder Ansichten (View) für Lesezugriff. Service-Importer werden je nach Anrufrichtung unterschieden. Importer greifen auf eine Quellsystem-API zu und fragen die erforderlichen Daten ab. Empfänger (Receiver) stellen eine Quellsystem-API als Endpunkt bereit und importieren die Daten aus den Quellsystemen. Publisher und Responder sollten als analog betrachtet werden. Publisher übertragen Daten an verschiedene Abonnenten, während Responder die Domänen-API definieren und eine Anfrage (z. B. von einem Benutzermodul) gemäß der Spezifikation beantworten. Mithilfe von Service-Prozessoren können algorithmische Aufgaben an einem Modell ausgeführt werden (z. B. Validierung der Topologie). Die Service-Importer und Exporter sowie die zugehörigen Service-Processoren, Modifier und Views sind so konzipiert, dass

sie flexibel ausgetauscht werden können.

2.1.3. Schichtenmodell-Entscheidung

Um eine bessere Wartbarkeit zu gewährleisten, sind die Komponenten der logischen Softwarearchitektur in einzelne Modelle unterteilt, die auf der hexagonalen Architektur (d. h. Ports und Adapter) von Alistair Cockburn basieren (siehe folgendes Diagramm):

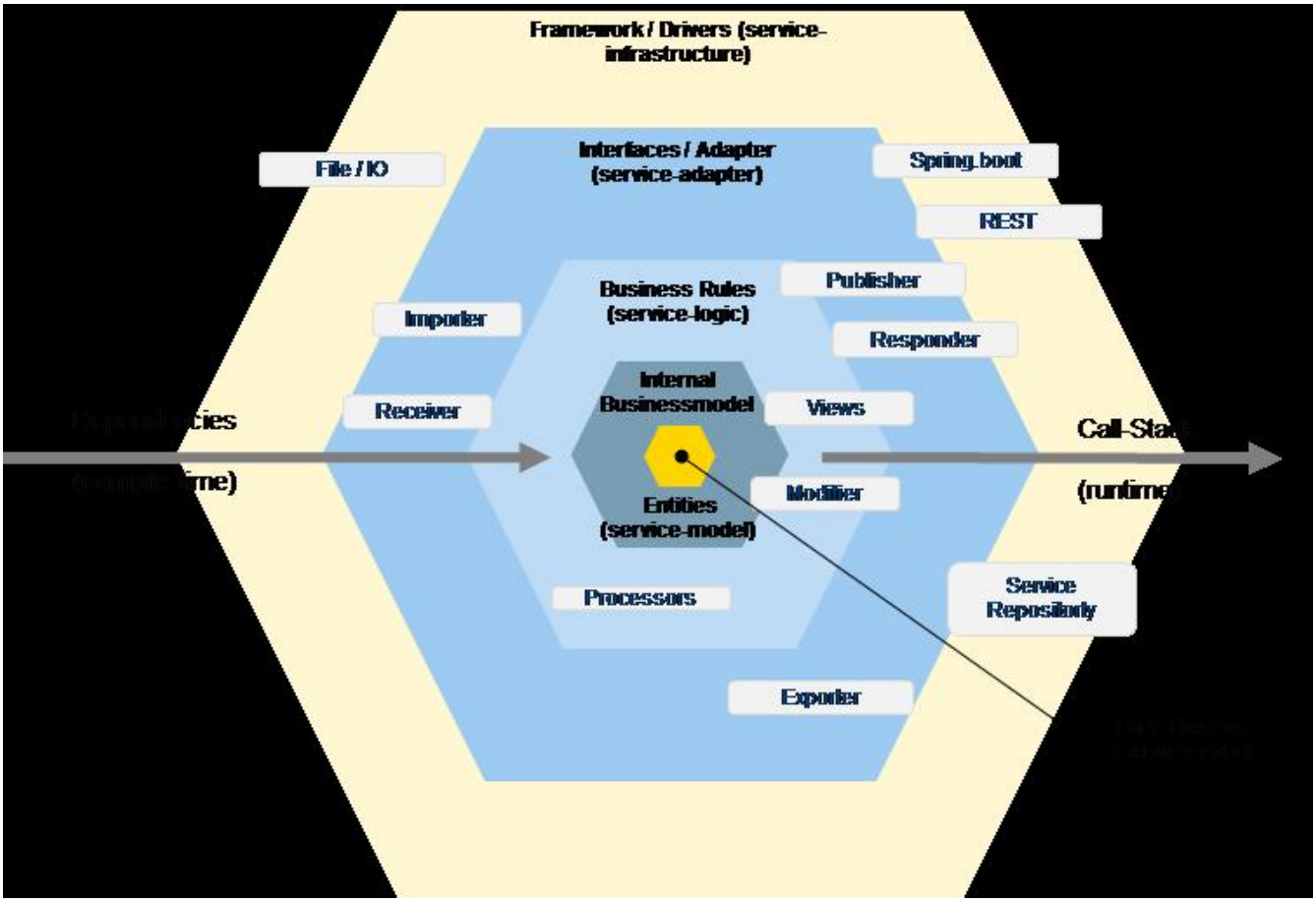

Figure 8. Schichtarchitektur

Die Schichtarchitektur bestimmt, welche Abhängigkeiten der Module während der Kompilierzeit untereinander bestehen können. Die Einhaltung dieser Struktur erleichtert die Erstellung eines intern spezifischen Gebäudemodells. Im Idealfall wird dies nicht oder nur minimal durch technischen Code "verunreinigt" und ist für die Kommunikation mit Systemen oder Komponenten sauber von Verarbeitungslogik und Adaptern getrennt.

2.1.4. Entscheidung für isolierte Datenmodelle

Der CIM-Cache enthält kein integriertes Datenmodell der einzelnen Komponenten "Dynamic-Topology", "Static-Topology", "Topology-State", "Asset-Data" und "Measurement". Stattdessen ist die Architektur auf der Grundlage der Notwendigkeit von Flexibilität, inkrementeller Erweiterbarkeit und Verfügbarkeit so konzipiert, dass die spezifisch gebauten Modelle voneinander isoliert sind. Die Modelle können zusammen in einem Microservice installiert oder auf mehrere Microservices verteilt werden. Diese Verteilung wird im folgenden Diagramm gezeigt:

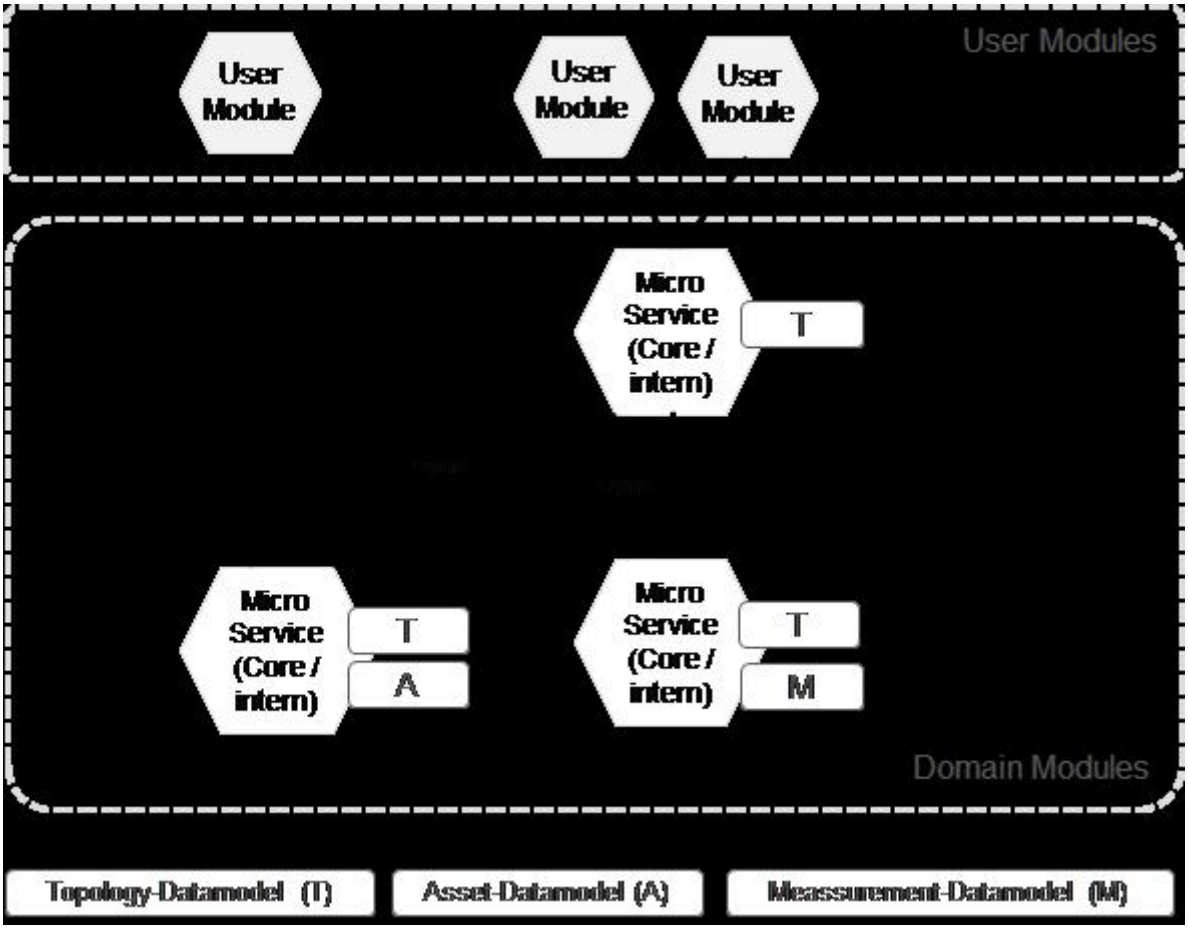

Figure 9. Verteilung isolierter Datenmodelle an Microservices

Im Diagramm wurde ein Microservice erstellt, der nur topologische Daten für andere Benutzermodule bereitstellt. Diese Daten werden anderen Mikrodienstleistungen über einen noch zu erstellenden Synchronisationsmechanismus bereitgestellt, so dass sie in andere Datenmodelle (z. B. Asset- oder Messdaten) integriert werden können. Spezielle Exporter können diese Daten im gewünschten Format anderen Anwendermodulen zur Verfügung stellen. Durch diese Architektur können schrittweise neue Dienste erstellt werden, ohne dass die vorhandenen Dienste neu erstellt werden müssen. Idealerweise werden verschiedene robuste zentrale Dienste erstellt, die dann von anderen Diensten verwendet werden. Bei den Varianten, die die Verteilung der Datenmodelle auf mehrere Microservices ermöglichen, muss sichergestellt sein, dass diese durch entsprechende Service-Importer und Exporter synchronisiert werden, die als Teil der Infrastruktur erstellt werden müssen. Durch die Verteilung der Datenmodelle auf einzelne Microservices wird ein hohes Maß an Flexibilität bei der Verteilung über verschiedene Computersysteme und bei der Verwendung verfügbarer Ressourcen (wie z. B. Hauptspeicher, Verarbeitungsleistung) erreicht. Datenmodelle müssen in dieser Architektur zu einem Microservice zusammengefasst werden, wenn diese zur Verwendung integriert und die Daten über Exportschnittstellen aggregiert und exportiert werden sollen.

2.1.5. Logging-Entscheidung

Die Protokollierung hat einen besonderen Wert für das Verfolgen von Fehlerzuständen. Die Einheitlichkeit der Protokollausgaben und Fehlercodes ist von entscheidender Bedeutung, damit diese in Umgebungen mit verteilten Software-Artefakten richtig ausgewertet werden können und um auf Fehler reagieren zu können (z. B. bei der Überwachung). Zur Protokollierung wird die "Simple Logging Facade for Java" (SL4J) verwendet, die vom derzeit etablierten Java Logging

Framework (java.util.logging, logback, log4j) abgeleitet ist. Zur besseren Einheitlichkeit der Protokollierung wurde SL4J durch eine zusätzliche Abstraktionsschicht entkoppelt. Der Kern dieser Abstraktionsschicht ist die LoggerFactory-Klasse für die Instantiierung bestimmter Logger und die ILogger-Schnittstelle zum Erstellen von Protokollausgaben. Bei der Erzeugung von Fehlerausgaben wurde besonderer Wert auf die einheitliche Qualifizierung dieser Fehler gelegt. Daher muss für jede Fehlerklasse eine spezifische Implementierung der INotification-Schnittstelle erstellt werden. Dies garantiert, dass bei Verwendung dieser Fehlerklassen einheitliche Fehlercodes in den openK-Plattformmodulen erzeugt werden. Die Interaktion zwischen den Klassen wird im folgenden Klassendiagramm gezeigt.

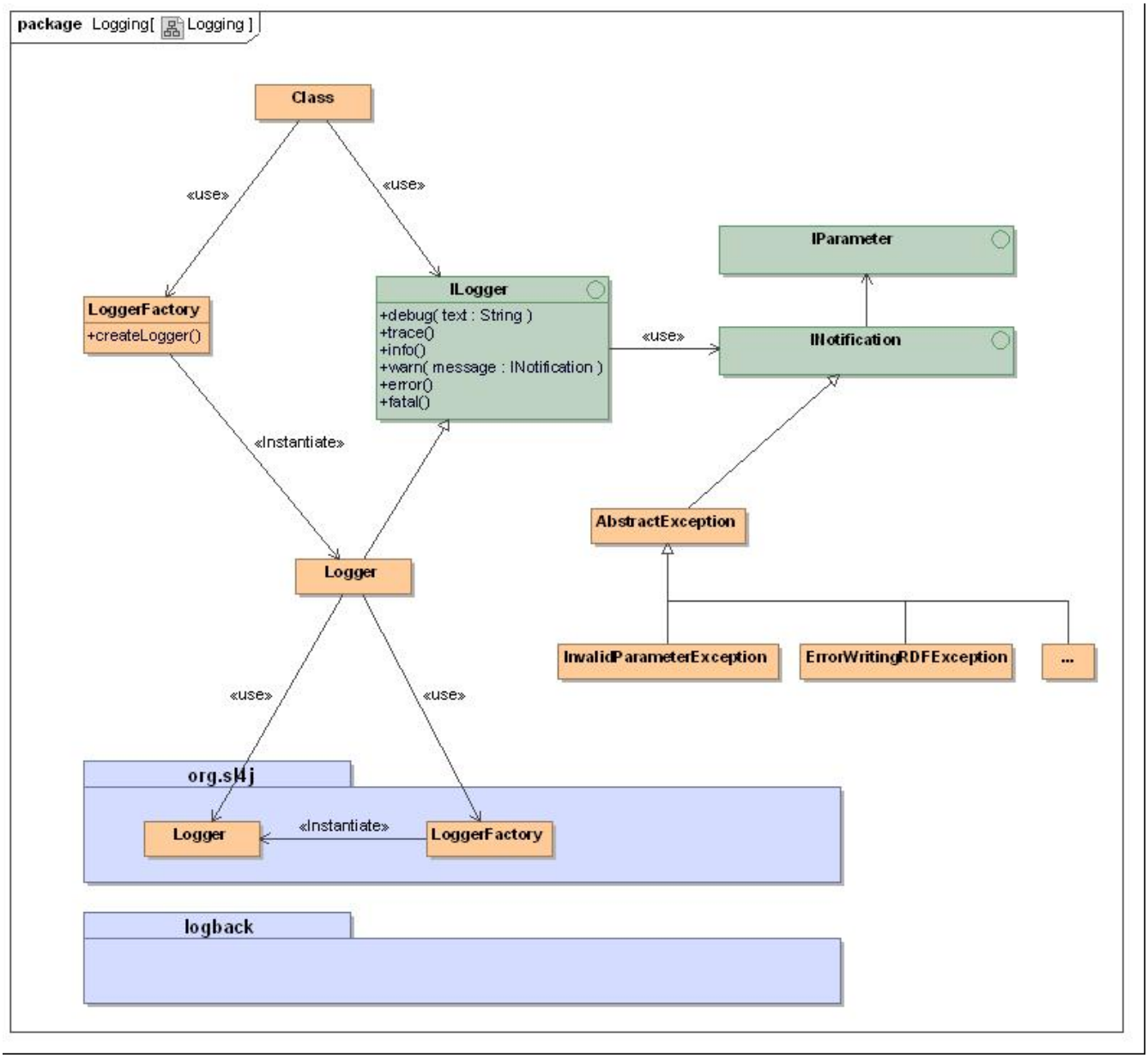

Figure 10. Klassendiagramm Logging

Zur besseren Auswertung von Protokolldateien besteht die Möglichkeit, unterschiedliche Kontextinformationen zu übertragen, so dass zusammengehörige Einträge (z. B. nach Prozess-ID, Anwendungsfall, Benutzer ...) gefiltert werden können.

2.2. Architektur-Referenzen

Für Qualitätsanforderungen siehe

Quality Committee Handbook @ openKONSEQUENZ

Für Architekturanforderungen siehe

Architecture Committee Handbook @ openKONSEQUENZ)

2.3. Modell

Internes Datenmodell von service-service

Die Service-Komponente stellt Basisklassen für die einzelnen fachlichen Services (Dynamic-Topology, Static-Topology, Topology-State, Measurement und Asset-Data) zur Verfügung. Die jeweiligen Datenmodelle sind in den fachlichen Services beschrieben.

Weiterhin stellt die Service-Komponente die REST-Schnittstelle für das Ausführen eines Servcice-Kommandos (POST-command) und REST-Schnittstellen zum abfragen (GET-command) bzw. Laden (POST-command) eines (Referenz-)Modells bereit. Siehe Schnittstellenbeschreibung.

3. HowTo

3.1. Erstellen

Diese Anleitung beschreibt, wie der Source-Code zu einer ausführbaren Jar-Datei gepackt werden kann.

Vorbereitung

- Maven installiert (>= Version 3.3.9)
- Git installiert (>= Version 2.10.1)

Durchführung

1. Clonen des Quellcodes z. B.für domain.xxx

```
$ git clone https://git.eclipse.org/r/openk-platform/org.eclipse.openk.domain.xxx
```

xxx = Anwendungsfall z. B. measurement

1. Wechseln in den Ordner

```
$ cd org.eclipse.openk.domain.xxx/
```

2. Bauen des Projekts

```
$ mvn clean install
```

Nach dem Bauen kann das JAR-Archiv aus dem Unterordner `\dev\service\target` entnommen werden.

3.2. Konfiguration

Vorbereitung

- Projekt erfolgreich mit Maven gebaut (siehe Erstellen)
- *.jar aus dem target-Ordner des Service-Moduls kopiert

Ordnerstruktur erstellen

Jeder Service kommt in ein eigenes Verzeichnis. Darin enthalten ist die JAR-Datei des jeweilen Service und die Konfigurationsdateien (Properties-Datei, Logger-XML-Datei und eventuell eine Service-Konfigurationsdatei). Für die Log-Dateien wird ein eigenes Unterverzeichnis angelegt (siehe logging.path in der Properties-Datei).

Am Beispiel Measurement-Service (Windows Betriebssystem) schematisch dargestellt:

```
|-> log (Verzeichnis)
  -> mock-up-measurement-service.log
  -> mock-up-measurement-service-2018-12-03.0.log
  -> ...weitere rollierende log-Dateien
-> measurement-service-<version>.jar (Service-JAR-Datei)
-> measurement-service.properties (Properties-Datei)
-> logger-configuration.xml (Logger-XML-Datei)
-> measurement-service.config (service-config-Datei)
-> start-measurement-service.cmd (CMD-Datei zum starten)
```

Services mit einer Service-Konfigurationsdatei:

- asset-data
- measurement
- dynamic-topology
- topology-state

Service properties Beispiel (Measurement-Service)

```
# server properties
server.port=9704

# logger properties (org.slf4j.Logger)

## path of the logger-configuration.xml file
logging.config=./src/main/resources/logger-configuration.xml

## name of the log-file
logging.file=measurement-service

## path for the log-file
logging.path=${java.io.tmpdir}/logs

## custom log-levels
## logging.level.root is defined in logging.config-file, but can be overridden here
# logging.level.root=INFO

## custom log-levels (base path is root, other paths override root settings)
logging.level.com.btc=DEBUG
logging.level.org.eclipse.openk=DEBUG
logging.level.org.hibernate=WARN
logging.level.org.springframework=INFO
logging.level.org.apache=WARN
logging.level.org.eclipse.persistence=WARN
```

Service Konfiguration Beispiel (Measurement-Service)

```
 # Measurments, V4
 RP|measurements|4|application-xml.scheme=http
 RP|measurements|4|application-xml.host=localhost ①
 RP|measurements|4|application-xml.port=9706 ②
 RP|measurements|4|application-xml.base-path=source-system/electricity/measurement

 # MeasurmentPoints, V4
 RP|measurment-points|4|application-xml.scheme=http
 RP|measurment-points|4|application-xml.host=localhost ①
 RP|measurment-points|4|application-xml.port=9706 ②
 RP|measurment-points|4|application-xml.base-path=source-system/electricity
/measurement

<1> Hostadresse (z. B. <IP> oder localhost)
<2> Port über den zugegriffen wird
```

- Schnittstellen und Parameter: measurement-service.config

Start Datei (CMD-Datei) Beispiel (Measurement-Service)

```
@echo off
rem Starten des Measurement-Service mit benutzerspezifischen Properties-Datei
java -jar "measurement-service-<version>.jar" --spring.config.location=measurement-
service.properties
@echo.
```

3.3. Starten

Vorbereitung

- Konfigurationen erstellt (siehe Konfiguration)
- abhängige Services sind gestartet
- benötigte Daten sind vorhanden (Testmodelle, etc.)

Service starten am Beispiel Measurement-Service

```
$ java -jar measurement-service-<'version'>.jar ①
```

① Befehl in Kommandozeile aufrufen

Überschreiben der Voreinstellungen

Dies kann durch Argumente erfolgen, die den Befehl erweitern.

```
--spring.config.location=<Pfad zur *.properties> ①
--logging.config=<Pfad zur logger konfig *.xml> ②
```

① enthält die Konfiguration für den Dienst

② enthält die Konfiguration für das Logging

3.4. Code

Vorbereitung

- IDE Eclipse (>= Version 4.7)
  - Checkstyle Plugin (>= Version 8.8.0)
  - Maven Plugin (>= Version 3.3.9)
- org.eclipse.openk.domain.measurement ausgecheckt (siehe Erstellen)

3.4.1. Projekt in Eclipse importieren

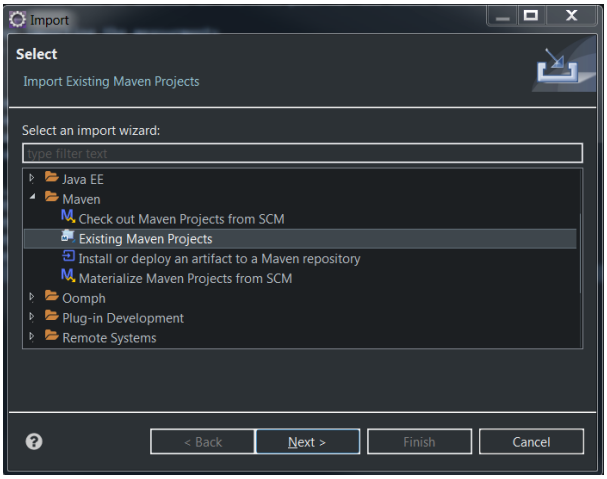

Figure 11. Maven import

- Projekt über die Importfunktion in Eclipse importieren (File > import)

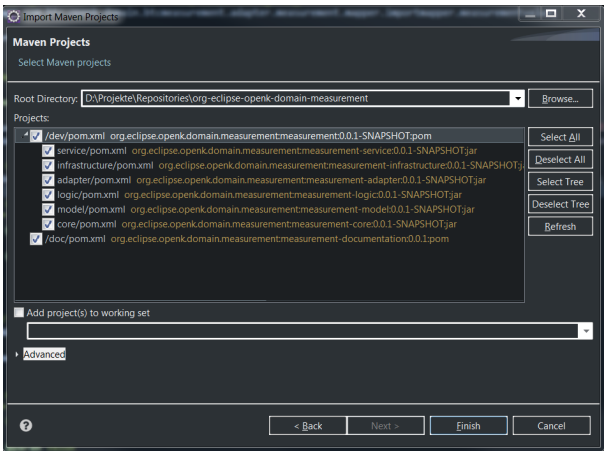

Figure 12. Projektauswahl

1. Angabe des lokalen Ablageortes
2. Module auswählen
3. Import durchführen (finish)

3.4.2. Entwicklung

Erzeugen eines Feature-branchs in dem Änderugen vorgenommen werden.

3.4.3. Änderungen committen

Änderungen über Eclispe E-Git committen und veröffentlichen (push).

3.5. Unit-Test

Es existieren umfangreiche UNIT-Tests (JUnit), insbesondere in der Logic-Schicht und bei den Mappern (Adapter-Schicht) in allen Service-Komponenten. Diese können durch die Wahl eines geeigneten Maven-Goals z. B.

```
$ mvn clean test ①
```

① Befehl in Kommandozeile aufrufen

gestartet werden.

Ausführliche (Integrations-)Tests sind im Kapitel Test beschrieben.

4. Test

4.1. Einleitung

Innerhalb des Projektes CIM-Cache 2.0 - Stufe 2 werden neue Funktionalitäten entwickelt, die gemäß QC-Handbook, im Rahmen der internen Qualitätssicherung getestet werden. Die Architektur des CIM-Cache sieht für die Kommunikation zu den Quellsystemen und zur Anwendungssoftware RESTful Webservices vor. Die REST-Schnittstellen sind spezifiziert und dokumentiert im YAML-Format über das openKONSEQUENZ Wiki verfügbar und sind über SWAGGER einsehbar.

Der CIM-Cache 2.0 - Stufe 2 stellt keine Benutzungsoberfläche sondern ausschließlich RESTful Webservices zur Verfügung. Die REST-Schnittstellen werden Positivtests auf Grundlage des internen Referenzmodells unterzogen. Das Referenzmodell ist im openKONSEQUENZ Wiki dokumentiert.

4.2. Testkonzept

Für das Projekt CIM-Cache 2.0 - Stufe 2 werden alle erweiterten Funktionalitäten mithilfe von Positivtests auf deren Funktionalität und Korrektheit hin überprüft. Diese Testfälle werden in Testdrehbüchern spezifiziert und anschließend in der QA-Umgebung durchgeführt. Dabei wird überprüft, ob die Ergebnisse der Schnittstellenaufrufe dem Referenzmodell und der Schnittstellenspezifikation entsprechen. Die Qualitätssicherung findet in mehreren Stufen statt. Zunächst folgt nach der abgeschlossenen Implementierung neuer Features ein initialer Test. Ist dieser erfolgreich, kann das Feature erfolgreich abgeschlossen werden. Wird hingegen ein Fehler festgestellt, wird die Implementierung dieses Features fortgesetzt. Dieser Vorgang wird so lange durchgeführt bis alle Tests eines Features erfolgreich sind.

4.3. Testdurchführung

Zur Durchführung der Positivtests aller neu implementierten Funktionalitäten des CIM-Cache 2.0 - Stufe 2, werden diese über das Internet aufgerufen. Aufrufe von GET-Schnittstellen können über einen Browser aufgerufen werden. Aufrufe von POST-Schnittstellen werden über einen REST-Client ausgeführt. Die Ergebnisse der Schnittstellen-Aufrufe werden anschließend mit den, in YAML spezifizierten, Schnittstellendefinitionen auf Korrektheit abgeglichen. Neben der korrekten Struktur des Ergebnisses werden ebenfalls die Daten und Verknüpfungen des Referenzmodells auf deren Richtigkeit hin überprüft. Zur Dokumentation werden alle relevanten Daten in einem Testprotoll festgehalten. Relevante Daten sind:

- Datum des Tests

- Getestete Buildversion
- Getestete Anforderung
- Schnittstellenaufruf mit Testdaten
- Link zur Schnittstellendefinition
- Erwartetes Testergebnis
- Ergebnis des Schnittstellentests
- Ggfs. Dokumentation von Fehlern

Sowohl die Testspezifikationen als auch die Dokumentation der durchgeführten Tests in Form eines Testprotokolls sind in einem Dokument zusammengefasst. Das verwendete Dateiformat ist ods, dieses kann als Open Document Format mit lizenskostenfreier Software eingesehen werden.

5. Anhang

5.1. Get Started

siehe HowTos

Erstellen

Code

Konfiguration

Starten

Tests

5.2. Variations

Abweichungen zur im Qualitätshandbuch (QC-Handbook) vorgegebenen Struktur werden hier beschrieben.

Die einzelnen fachlichen Komponeten (Dynamic-Topology, Static-Topology, Topology-State, Measurement und Asset-Data) sind alle exakt in der vom Qualitätshandbuch vorgegebenen Struktur gehalten.

Siehe Architektur