„Fertigstellungsgrad“ der Architekturdokumentation

Analyse des Dokumentationsbedarfs und Priorisierung der Arbeiten

Energie
Energy

„Fertigstellungsgrad“ der Architekturdokumentation oK

Dr. Jürgen Meister

OpenKONSEQUENZ Architecture&Quality Committees Workshop
20. – 21.4.2016 - Dortmund

2 Motivation und Ziele

- Bestandsaufnahme der aktuellen Architekturdokumentation zur Vorbereitung eines Architekturdokuments mit Dokumentation der Architektur- und Entwicklungsvorgaben für oK-Module

- Darstellung und Diskussion des „Fertigstellungsgrads“ der Architekturdokumentation für einzelne Themenbereiche von Arc42
- Identifikation fehlender Themen zur Vorbereitung der Ausschreibungen
- Hinweise für vorhandene (aber hier noch nicht beachtete) Dokumentation
- Hinweise zu Themen, die für anstehende oK-Module besonders wichtig sind
- Vorbereitung eines Action Item-Backlogs für AC/QC

3 Inhalt Arc42

Übersicht der Arc42-Themen (1)

Anforderungsbezogene Informationen
Strukturen des Systems (Sichten, Muster)
Übergreifende (technische) Konzepte
Entwurfsentscheidungen und Risiken

- **1. Einführung und Ziele**
  Aufgabenstellung, Qualitätsziele, eine Kurzfassung der architekturrelevanten Anforderungen (insb. die nichtfunktionalen), Stakeholder.
- **2. Randbedingungen**
  Welche Leitplanken schränken die Entwurfsentscheidungen ein?
- **3. Kontextabgrenzung**
  In welchem fachlichen und/oder technischen Umfeld arbeitet das System?

- **4. Lösungsstrategie**
  Wie funktioniert die Lösung? Was sind die fundamentalen Lösungsansätze?

- **5. Bausteinsicht**
  Die statische Struktur des Systems, der Aufbau aus Implementierungsteilen.

  6. Laufzeitsicht
  Zusammenwirken der Bausteine zur Laufzeit, gezeigt an exemplarischen Abläufen ("Szenarien").

4 Inhalt Arc42

Übersicht der Arc42-Themen (2)

Anforderungsbezogene Informationen
Strukturen des Systems (Sichten, Muster)
Übergreifende (technische) Konzepte
Entwurfsentscheidungen und Risiken

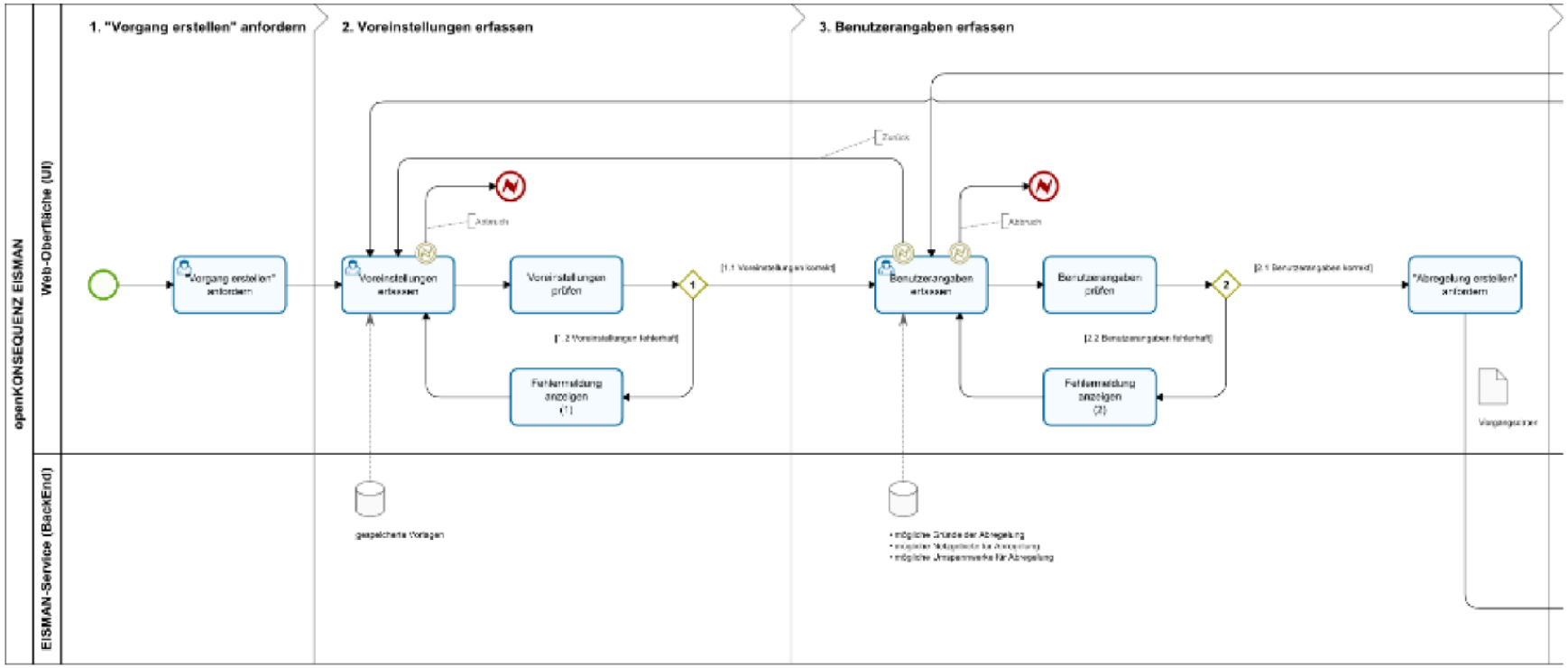
- **7. Verteilungssicht**
  Deployment: Auf welcher Hardware werden die Bausteine betrieben?
- **8. Querschnittliche Konzepte und Muster**
  Wiederkehrende Muster und Strukturen.
  Fachliche Strukturen.
  Querschnittliche, übergreifende Konzepte, Nutzungs- oder Einsatzanleitungen für Technologien. Oftmals projekt-/systemübergreifend verwendbar!
- **9. Entwurfsentscheidungen**
  Zentrale, prägende und wichtige Entscheidungen.
- **10. Qualitätsszenarien**
  Qualitätsbaum sowie dessen Konkretisierung durch Szenarien
- **11. Risiken**
- **12. Glossar**
  Wichtige Begriffe

5 1. Einführung und Ziele

- Aufgabenstellung, Qualitätsziele, eine Kurzfassung der architekturrelevanten Anforderungen (insb. die nichtfunktionalen), Stakeholder

- Annahmen und zu beachtende Aspekte
  - Allgemeine Teile werden für alle Module gelten
  - Ergänzende funktionale und nicht funktionale Anforderungen werden in Lastenheften für die Ausschreibungen dokumentiert werden

6 1. Einführung und Ziele (Status Quo bei Anforderungen)



7 1. Einführung und Ziele

- Status und ToDos
  - Explizite (Kurz-)Dokumentation der allgemeinen Aufgabenstellung und Qualitätszielen fehlt noch
  - Dokumentation für offizielle Architektur- und Entwicklungsvorgaben steht an
  - Prozessdokumentation für Module soll künftig in Lastenheften erfolgen

2. Randbedingungen

- Welche Leitplanken schränken die Entwurfsentscheidungen ein?

- Annahmen und zu beachtende Aspekte
  - Open Source-Lizenzierungsmodell „Eclipse Public License 1.0“
  - Entwicklungsprozesse:
    - Kaufmännisch: Konsortiale Beauftragung
    - Entwicklung: SCRUM
      - Product Owner kommt von oK
      - SCRUM Master vom Software-Hersteller
    - QS und Abnahme: Einzelsprints + 3 Monate Probebetrieb

9 Randbedingungen

Innovation: Product development is driven by a constant innovative progress which covers not only the product features but also the development methods and tools.

Standardization: Usage of standardized data structures (CIM) and technical platform.

Process integrity and Data Exchange: According to the distributed product development exchange of data is a crucial issue for collaboration. Test results have to be passed seamlessly along the process flows between organizations (internal departments or external organizations) and between engineering disciplines. It is therefore a matter of the tools used to use open and standardized interfaces.

Long Term Availability: Software product of openKONSEQUENZ® WG might be run more than 15 years. Development and support have to be ensured.

Legal requirements: Legal and regulatory requirements change frequently and vary from country to country within the energy-industry. openKONSEQUENZ®-Systems should be prepared to adapt to these requirements.

Vendor neutrality: Systems should not have interdependences with proprietary systems.

Security: Critical Infrastructure Operators have statutory duty to supply electric power and water. Consequently, the EWIS software must satisfy strict security requirements, such as confidentiality, availability and integrity of processed information. Furthermore, software development process must conform to rules and best practices of secure software development (Secure Development Lifecycle).

Safety: Being part of the critical infrastructure, the EWIS software must conform to commonly adopted power industry safety standards, and guarantee non-functional properties such as reliability and fault tolerance.

Data integrity: Data integrity requirements are particularly high for SCADA Systems. EWIS-solutions have to meet versatile security requirements to protect data from unauthorized access and modification that does not affect the efficient flow of data within the production process.

Availability: The availability requirement is particularly high for SCADA systems. This has to be considered in system- and software architecture.

2. Randbedingungen

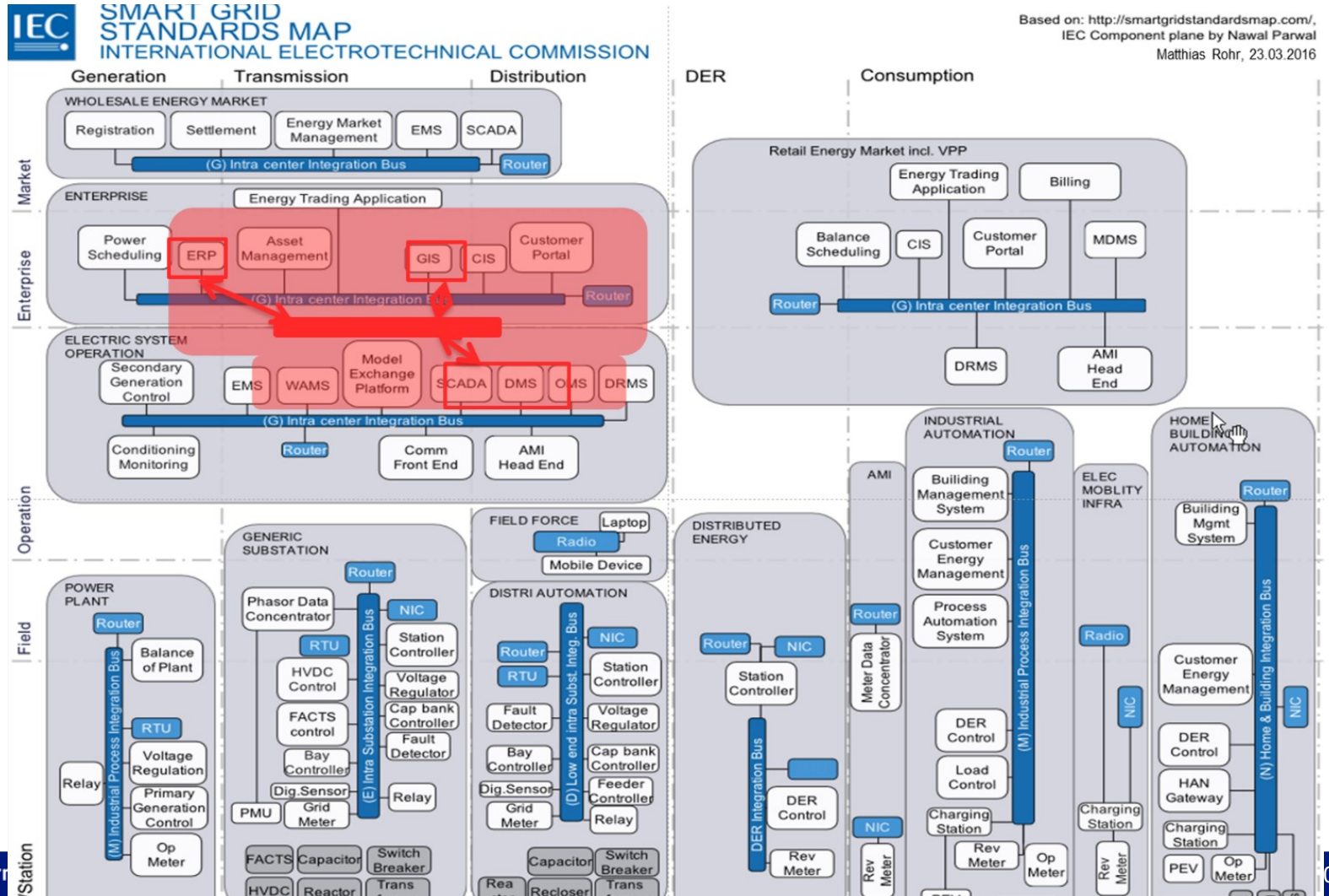
- Status
  - Grundlegende Ideen auf dem Auszug auf der vorheriger Folie dokumentiert
- ToDos
  - Es müssen noch kritische nicht-funktionale Anforderungen zu Prozessen, Laufzeitumgebung, etc. kompakt aufbereitet werden

Es soll zwischen Vorgaben und Empfehlungen unterschieden werden.

3. Kontextabgrenzung

- In welchem fachlichen und/oder technischen Umfeld arbeitet das System?
- Annahmen und zu beachtende Aspekte
  - Ergänzende Komponente für NLS oft mit Kopplung mit IT der Verteilnetzbetreiber

12 3. Kontextabgrenzung



3. Kontextabgrenzung

- Status
  - Grundlegende Abgrenzung vorhanden
- ToDos
  - Kontextabgrenzung openK -> allgemeine Architekturvorgaben
  - Kontextabgrenzung Modul -> Architekturdokumentation des Moduls (In Projekten)

4. Lösungsstrategie

- Wie funktioniert die Lösung? Was sind die fundamentalen Lösungsansätze?

- Annahmen und zu beachtende Aspekte
  - Grundlegende Konzepte werden in Abschnitten „8. Querschnittliche Konzepte und Muster“ und „9. Entwurfsentscheidungen“
  - Detaillierte Dokumentation ist kein Bestandteil für allgemeine (modulübergreifende) Architekturdokumentation
  - Lösungsstrategie je Modul -> Architekturdokumentation des Moduls

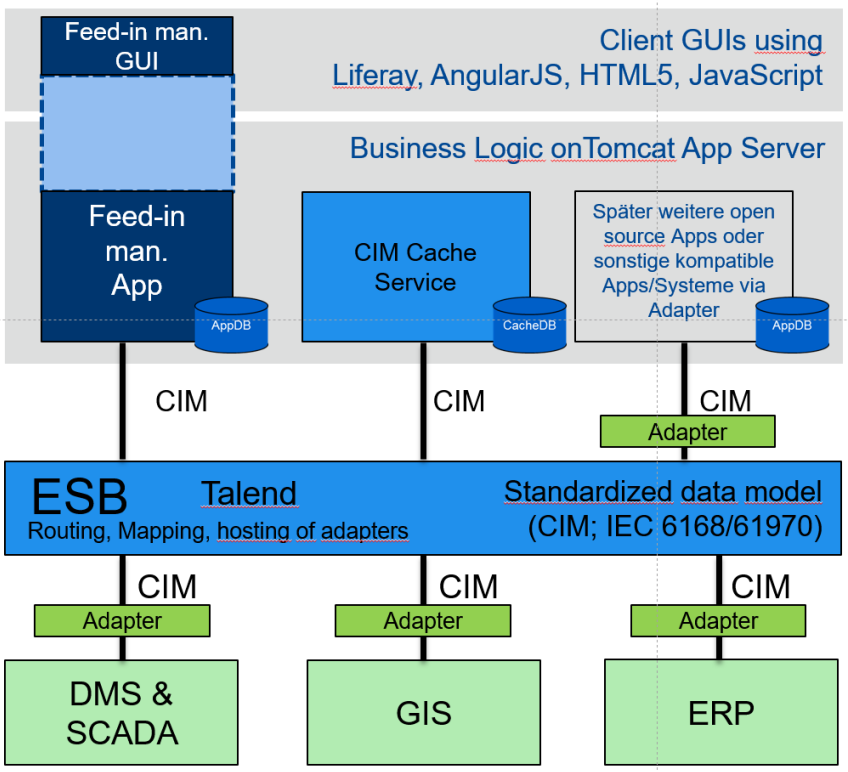
- Status und ToDos
  - Für AC/QC keine ToDos

5. Bausteinsicht

- Die statische Struktur des Systems, der Aufbau aus Implementierungsteilen (UML -> ~~Paket~~ Komponenten- und Klassendiagramme)
- Annahmen und zu beachtende Aspekte
  - Dokumentation 1. Detaillierungsebene als UML ~~Paket~~Komponenten-diagramme Vorgaben durch ACQC
    - Fachliche Module und externe Systeme
    - 1. Detaillierungsebene (technische und fachliche Module) für oK- Allgemein sollte vom AC/QC gesteuert werden
    - Hier auch die fachlichen Schnittstellen zwischen Modulen definieren
  - Dokumentation 2. Detailierungsebene als UML ~~Paket~~ Komponenten- und Klassendiagramme durch Modulentwickler
    - Überblick über Packages und internen Schnittstellen
    - Detailentwurf „komplizierter“ Module als UML ~~Paket~~ Komponenten- und Klassendiagram
    - Detaildokumentation als JavaDoc Detaildokumentation in Architektur- und Qualitätsteil Trennen. Dies ist nötig hinsichtlich Nutzung und vor allem Wartung der Module
    - Datenbankmodell als ER- oder UML-Klassendiagramm
  - Trennung zwischen technischen und fachlichen Packages muss explizit gemacht werden

5. Bausteinsicht – aktueller Stand

High-level architecture 1/2



Architectural rules:

- *Apps interact only via ESB and CIM interfaces (web services, rest)*
- *Each app may store its own entities in a private app-DB (➔ no data exchange using DB)*
- *Communication between apps and legacy systems via ESB*
- *CIM-based communication (IEC 61968, IEC 61970)*
- *Special features of ESB (and other middleware) are not used to prevent vendor-lock-in*
- *Apps are hosted in the openKONS. App-Server (Tomcat) ➔ Java*

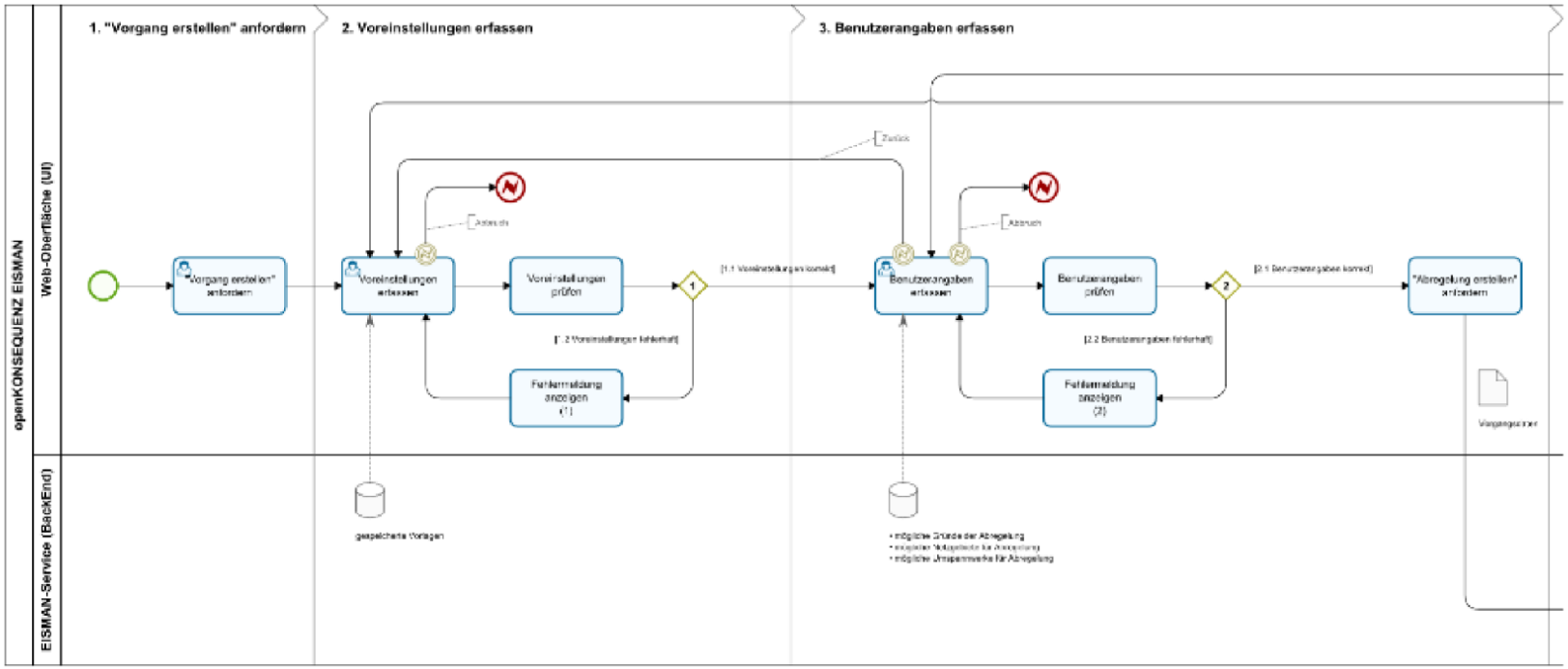
5. Bausteinsicht

- Status
  - Marchitectures vorhanden (auch mehrere Varianten)
- ToDos
  - 1. Detaillierungsebene zur Vorbereitung der Ausschreibungen nachdokumentieren, um den Kontext für die geplanten Module festzulegen
  - Provides- und Requires-Schnittstellen für geplante Module als 1. Draft (sofern es nicht bereits „fertige“ Requires-Schnittstellen gibt) festlegen

6. Laufzeitsicht

- Zusammenwirken der Bausteine zur Laufzeit, gezeigt an exemplarischen Abläufen ("Szenarien")
- Annahmen und zu beachtende Aspekte
  - Nicht-triviale Abläufe in der Anwendungslogik und Abläufe, die mehrere Module betreffen werden als UML Sequenz- oder Kollaborationsdiagramme modelliert
  - Modulübergreifende Szenarien
    - Technische Dokumentation der Prozesse als Sequenzdiagramm, das die Aufreihenfolgen zwischen Anwendern, Modulen und externen Systemen dokumentiert
    - Matching der Schnittstellen der Module bzgl. Der Funktions- oder Service-Aufrufe qualitätssichern
  - Modulinterne Szenarien
    - Dokumentation nicht-trivialer Prozesse als Sequenzdiagramme mit Klassen als „Aktoren“
  - UML-Sequenzdiagramm (Kollaborationsdiagramme) definieren Funktions-, Kommunikations- und Datenflüsse für ein Szenario

19 Laufzeitsicht



Offen ist das Zusammenspiel der Module in der Architektur zur Ausführung des obigen Prozesses

20 Laufzeitsicht

3. Betrachtete Szenarien

Für die Untersuchung und Bewertung der Architektur wird ein typisches Nutzungsszenario betrachtet:

I. Der Benutzer sieht das gerade aktualisierte Netzzustandsbild.
II. Auf Grund einer Situation im Netz schlägt das System einen Schaltantrag vor.
III. Der Benutzer bearbeitet den Schaltantrag und markiert ihn zur Übertragung.
IV. Das System stellt den Schaltantrag auf der Leitwarte zur Verfügung.
V. Der Schaltantrag wird in der Leitstelle ausgeführt.
VI. Über die zyklische Aktualisierung des Netzzustandsbilds ist die Veränderung beim Benutzer sichtbar.

Grundsätzlich ist dies eine „Analyse-Aktion-Kontrolle“ Funktion des Systems. Ausgehend von einer analytischen Sicht auf den Zustand der tatsächlichen Welt, wird eine Aktion ausgelöst und deren Wirkung kontrolliert. Kritisch ist in solchen Interaktionen im Allgemeinen das Duo „Aktion-Kontrolle“, so auch hier. Abbildung 4 veranschaulicht an Hand des Szenarios exemplarisch, welche Datenflüsse zwischen einzelnen Elementen des Systems erforderlich sind.

21 Laufzeitsicht

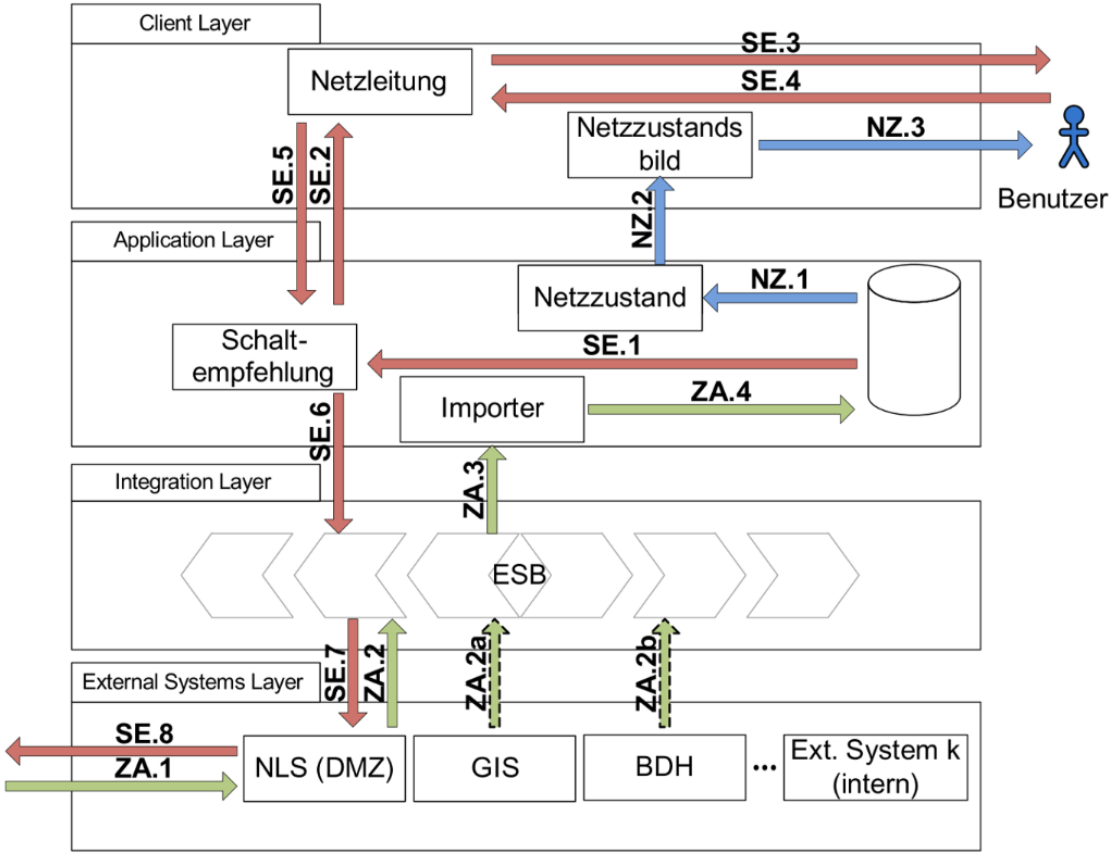


Abbildung 4: Datenfluss für das betrachtete Szenario

6. Laufzeitsicht

- Status
  - Fachliche Prozesse „nur“ als BPMN Diagramme dokumentiert
  - Technische Prozesse „nicht“ als UML dokumentiert

- ToDos
  - Modulunabhängige „technische“ Prozesse (z.B. Nutzung CIM Cache, Authentifizierung und Autorisierung) mit UML Mitteln nachdokumentieren

7. Verteilungssicht

- Deployment: Auf welcher Hardware und Infrastruktursoftware werden die Bausteine betrieben?

- Annahmen und zu beachtende Aspekte
  - Es werden zwei Deployment- und Verteilungsszenarien für die Systemarchitektur unterschieden
    - Minimal-Umgebung =? openK-Referenzumgebung
    - Standard-Umgebung gemäß dem Vorschlag von Develop Group
  - Verteilungs- und Deployment-Verfahren für oK-Module müssen für alle Module vereinheitlicht sein

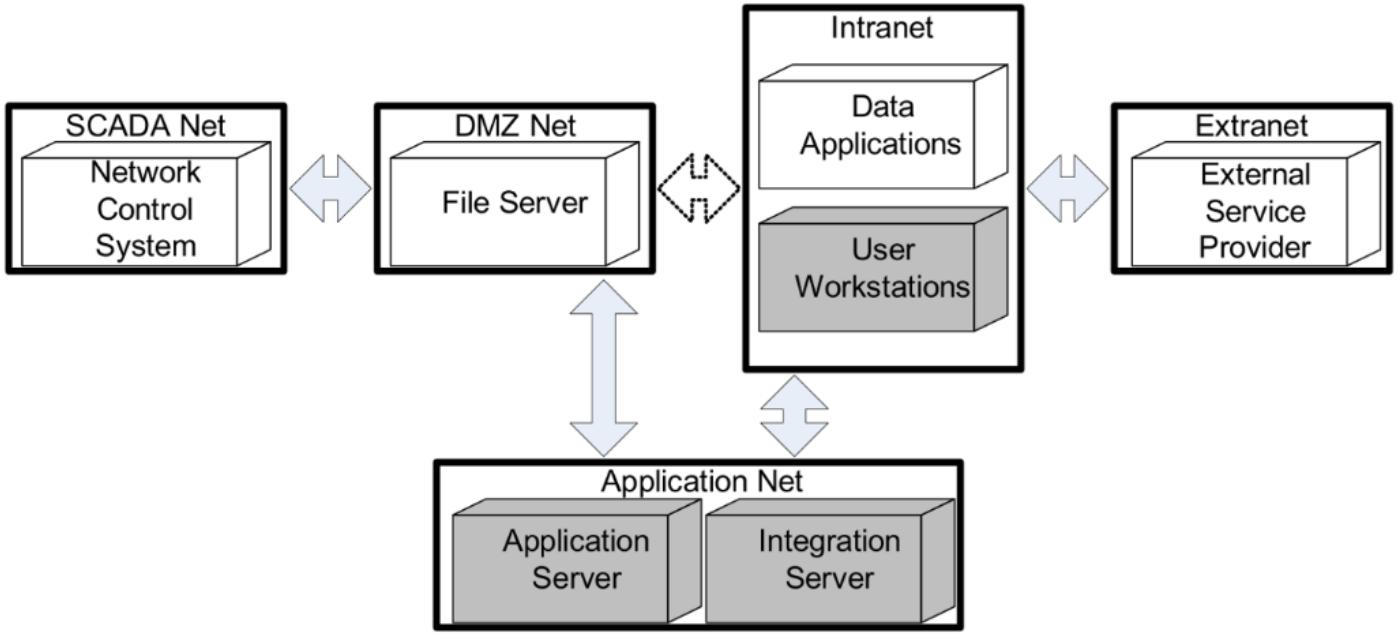
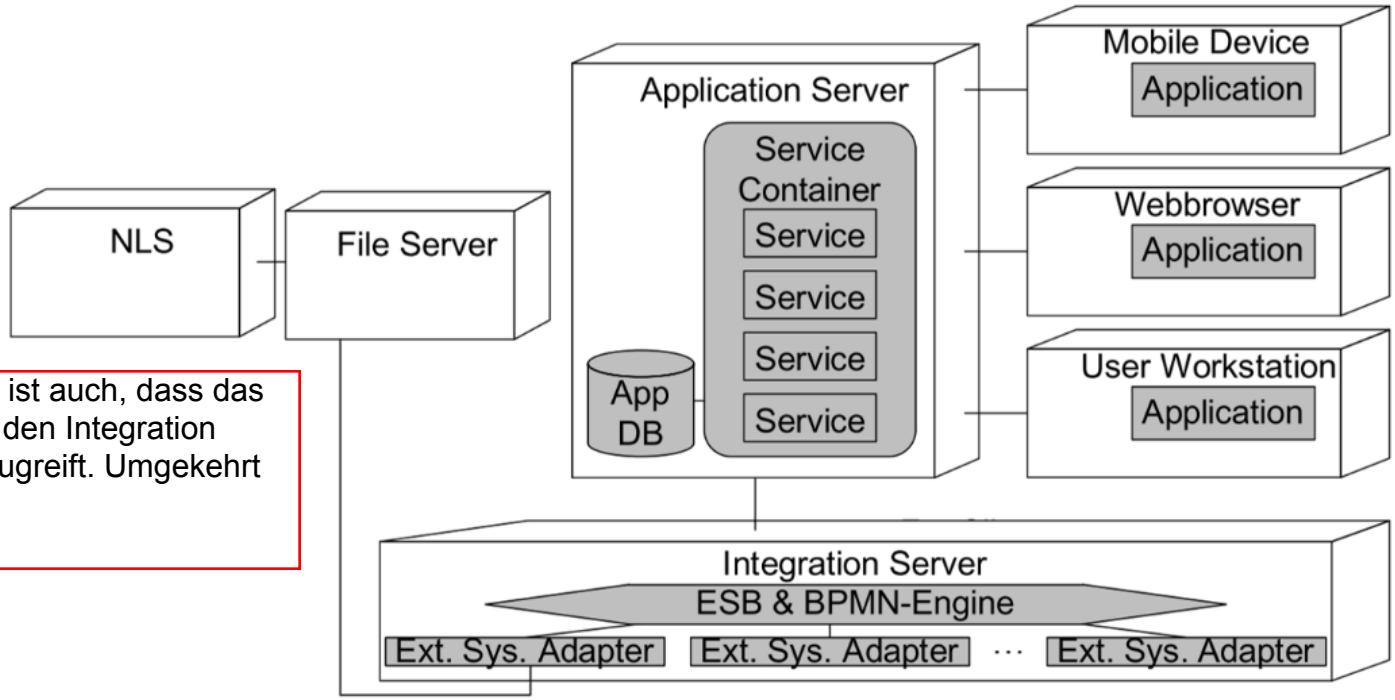


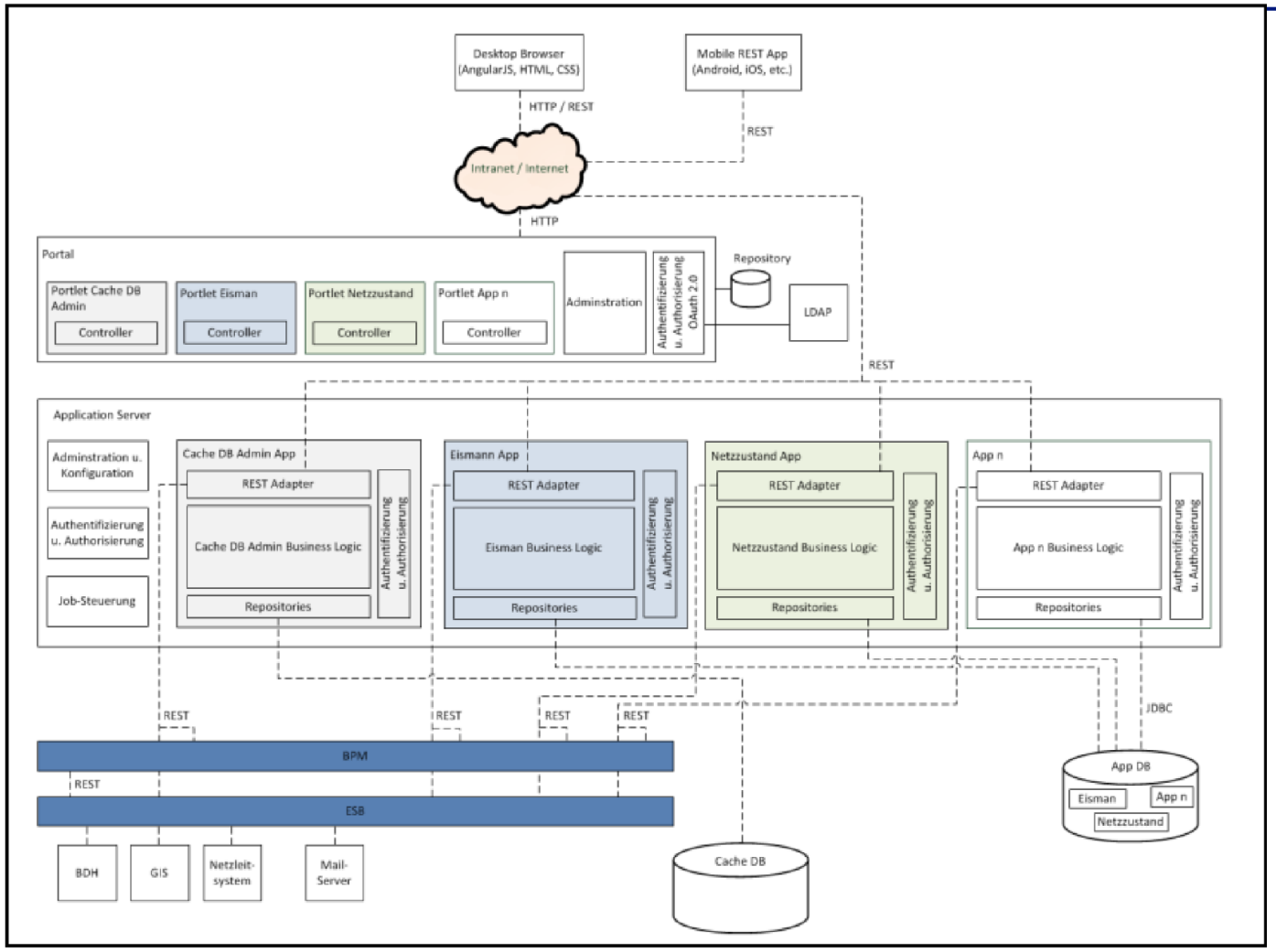
Abbildung 2: Netzwerktopologie

Dies ist keine MicroService-Architektur.



Denkbar ist auch, dass das NLS auf den Integration Server zugreift. Umgekehrt nicht.

Abbildung 3: Einsatz der Komponenten


Desktop Browser (AngularJS, HTML, CSS)
Mobile REST App (Android, iOS, etc.)
HTTP / REST
REST
Intranet / Internet
HTTP
Portal
Portlet Cache DB Admin
Controller
Portlet Eisman
Controller
Portlet Netzzustand
Controller
Portlet App n
Controller
Adminstration
Authentifizierung u. Authorisierung OAuth 2.0
Repository
LDAP
REST
Application Server
Adminstration u. Konfiguration
Authentifizierung u. Authorisierung
Job-Steuerung
Cache DB Admin App
REST Adapter
Cache DB Admin Business Logic
Repositories
Authentifizierung u. Authorisierung
Eismann App
REST Adapter
Eisman Business Logic
Repositories
Authentifizierung u. Authorisierung
Netzzustand App
REST Adapter
Netzzustand Business Logic
Repositories
Authentifizierung u. Authorisierung
App n
REST Adapter
App n Business Logic
Repositories
Authentifizierung u. Authorisierung
REST
REST
REST
REST
JDBC
BPM
REST
ESB
BDH
GIS
Netzleit-system
Mail-Server
Cache DB
App DB
Eisman
App n
Netzzustand

7. Verteilungssicht

- Status
  - Vorgaben für Standard-Umgebung können aus dem Gutachten der Develop Group übernommen werden

- ToDos
  - Abgleich der Standard-Umgebung mit der Referenzarchitektur machen und dokumentieren
  - Hardware-Vorgaben an echte Rechnerknoten resp. Anforderungen Virtualisierungsinfrastruktur und VMs für Test- und Produktivumgebungen festlegen
  - Verfahren und Tools für Verteilung- und Deployment der Module müssen nachdokumentiert werden.

Abnahmeplattform ist zu definieren und Minimalkonfiguration anzugeben.

8. Querschnittliche Konzepte und Muster

- Wiederkehrende Muster und Strukturen. Fachliche Strukturen. Querschnittliche, übergreifende Konzepte, Nutzungs- oder Einsatzanleitungen für Technologien. Oftmals projekt-/systemübergreifend verwendbar!

- Annahmen und zu beachtende Aspekte
  - Für openKONSEQUENZ der wichtigste Abschnitt in der Architekturdokumentation
  - Festlegung des „Technologiesoftware“-Stacks
  - Styleguides UI
  - Grundsätzliche Regelungen (Entwurfsmuster) für die Entwicklung wartbarer Software in Anlehnung an Quasar von Prof. Siedersleben festlegen
    - Wartungsdauer > 15 Jahre
    - Technologiesoftware (App-Server, …) < 5 Jahre
    - Trennung von Technologie- und Anwendungscode anstreben
  - Entwurfsmuster für Authentifizierung und Autorisierung
  - Vorgaben für CIM Nutzung und Profilierung
  - Etc.

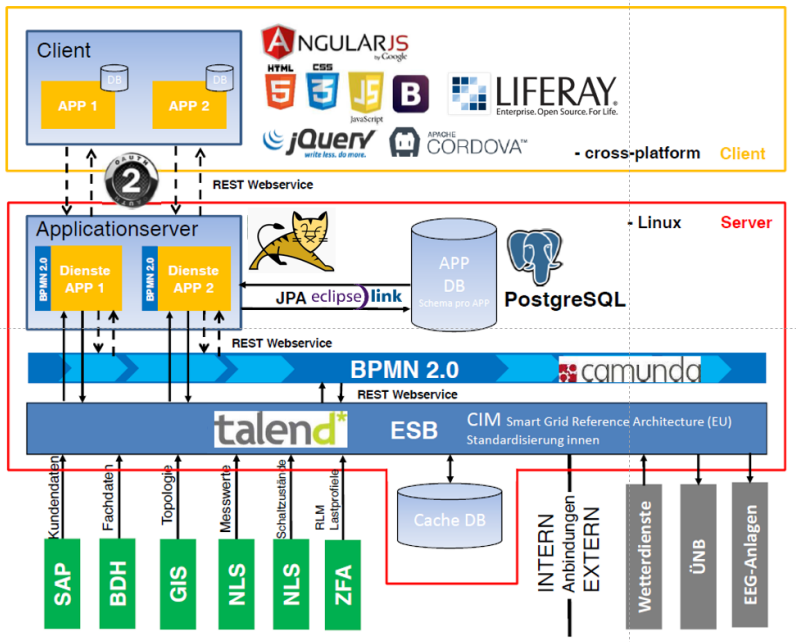
8. Querschnittliche Konzepte und Muster

Architectural rules:

- *Apps interact only via ESB and CIM interfaces (web services, rest)*
- *Each app may store its own entities in a private app-DB (➔ no data exchange using DB)*
- *Communication between apps and legacy systems via ESB*
- *CIM-based communication (IEC 61968, IEC 61970)*
- *Special features of ESB (and other middleware) are not used to prevent vendor-lock-in*
- *Apps are hosted in the openKONS. App-Server (Tomcat) ➔ Java*

8. Querschnittliche Konzepte und Muster

Architecture and technology



- The initial architecture & technology stack were preselected. The first project refined it.

- **Portal: Liferay**
- **ESB: Talend**
- **ORM: Eclipselink**
- **App-Serv.: Tomcat**

(Other DBMSs, ESBs, Java-App-Server are possible)

This can change:
- Arch. & Qual. Committee
- Eclipse os licence review

Not addressed for first apps:
- Cordova, Camunda

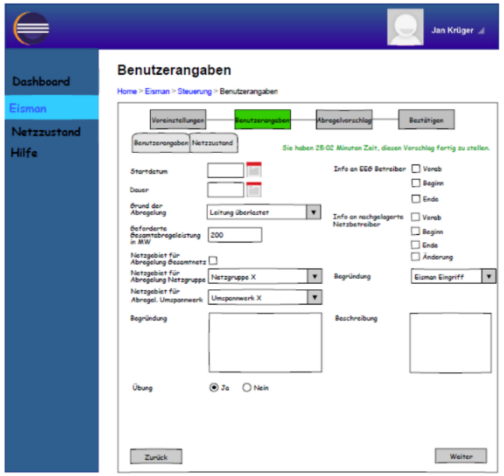
➔ Architecture on next slide

Entwicklung Uis - Vorgehen

Entwurf der Uis durch die BTC in Abstimmung mit Product Owner

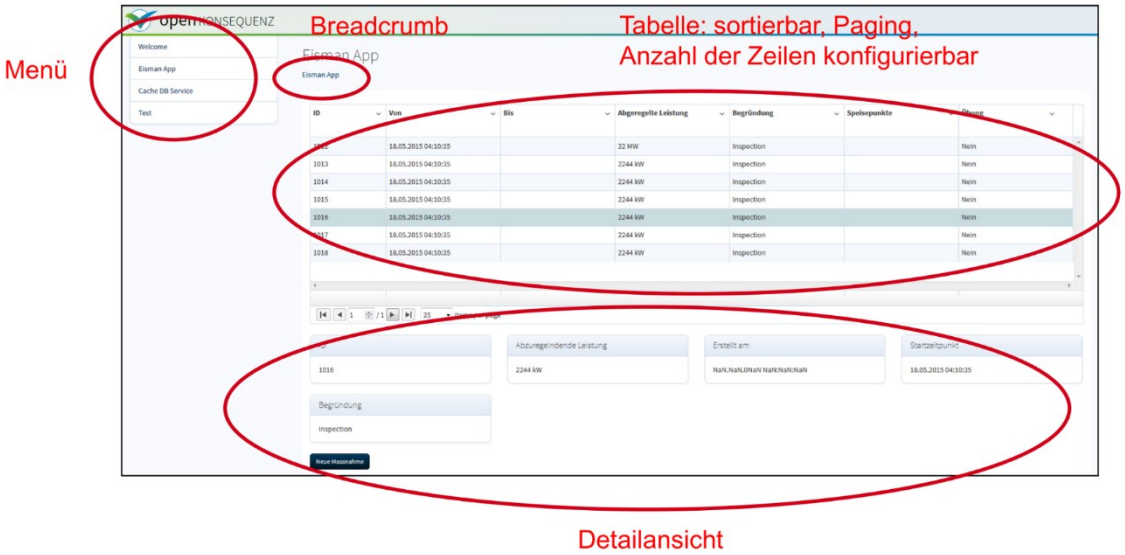
Entwicklung Design durch die Firma Minnemedia

- Styleguide
- HTML-Seiten
  (inkl. Bootstrap und Javascript Artefakte)



32 Querschnittliche Konzepte und Muster

Implementierung UIs



Build Process als Maven Projekt

TestNG: Ausführung in IDE u. maven


```
@TransactionConfiguration(defaultRollback = false)
@Transactional(value = "importDbTransactionManager", rollbackFor = Throwable.class)
@TestExecutionListeners(TransactionalTestExecutionListener.class)
@ContextConfiguration(locations = {"classpath:spring-persistence-test.xml"})
public abstract class AbstractImportDbTransactionalBaseTest extends AbstractTransactionalBaseTest {
    @PersistenceContext(unitName = "importDbEntityManagerFactory")
    protected EntityManager entityManager;
}

public abstract class AbstractTransactionalBaseTest extends AbstractTestNGSpringContextTests {
…
}
```

Integrationstest „gegen“ Datenbank und Tests für das Verarbeiten von REST-Services (JSON)

ToDo: Continuous Build

Checkstyle

8. Querschnittliche Konzepte und Muster

- Status
  - Viele der Konzepte durch den Piloten bereits erprobt und implizit festgeschrieben
  - Einheitliche und zentrale Dokumentation fehlt

- ToDos
  - Nachdokumentieren
  - Backlog des Architecture Committee mit offenen Actions Items aus diesem Workshop „auffüllen“
  - Vorgaben für Konfigurations- und Versionsmanagement dokumentieren
  - Vorgaben für Paketierung sowie Deployment- und Inbetriebnahme-Unterstützung festlegen

9. Entwurfsentscheidungen

- Entwurfsentscheidungen: Zentrale, prägende und wichtige Entscheidungen.

- Annahmen und zu beachtende Aspekte
  - Wichtige Entwurfsentscheidungen, z.B.
    - Einführung von Service-Komponenten für mehr als ein Modul
    - Auswahl einer Technologie, die mehr als ein Modul betrifft
    - Müssen dokumentiert und mit AC/QC abgestimmt werden
  - Dokumentation der Entwurfsentscheidungen dient dem besseren Verständnis bestehender Architektur und soll „not invented here“ Tendenzen vermeiden
  - In der Machbarkeitsuntersuchung und aktuellen Pilotprojekt sind eine Reihe von Entwurfsentscheidungen getroffen, die nachdokumentiert werden müssen

9. Entwurfsentscheidungen

Technologie:

- Back-End: Java, JPA, Spring, Postgres, …
- Schnittstellen:
  - Front-End ⇔ UI: REST, JSON
  - Im-/Export: CIM (über ESB)
- UI (webbasiert): AngularJS, Bootstrap, …
- Build-Tool: Maven

BTC

Feed-in man.-App + GUI:

- Bus.Log. for creating a RES curtailment suggestion
- Focus on TSO call for grid curtailment
- Web-based user interface
- open source
- Uses CIM Cache Service as integrated topology data model
  (e.g., which RES are currently connected to which substations)

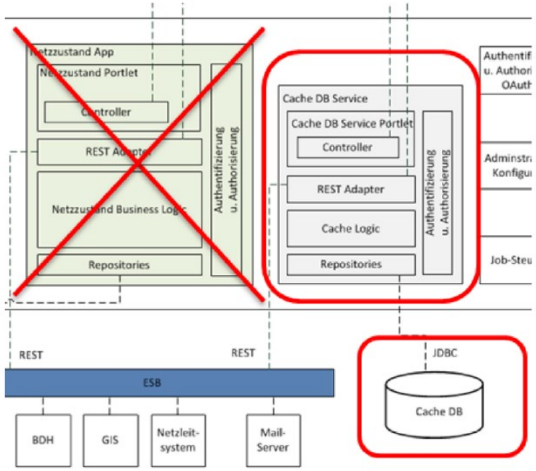
CIM Cache Service:

- Uses CIM-based integrated datamodel for topology, (current measurements), current grid status, RES master data
- Currently focused on demands of the first apps (switches, lines, substations, transformers, renewables, …)
- The CIM Cache is part of the openK platform – it should be used and extended by the following applications
- The cache compensates reliability and performance issues of connected systems

3

9. Entwurfsentscheidungen

Anpassungen Architktur
Notwendigkeit Netzzustand und Funktionserweiterung Cache DB



Aufgaben Cache DB

- Kompensation der geringeren Verfügbarkeiten der Dritt-Systeme
- Erhöhung der Performance bzgl. Datenabfragen der Dritt-Systeme

Cache DB beinhaltet mit CIM ein integriertes Datenmodell, das verschiedene Sichten auf die Daten ermöglicht

- Topologie
- Messwerte/Istwerte
- Stammdaten

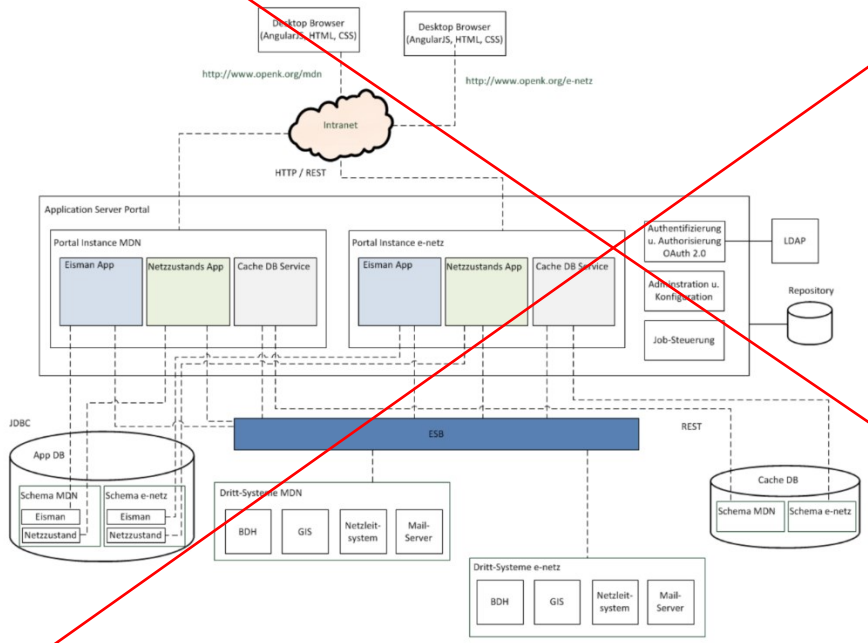
Cache DB Service übernimmt Aufgaben der Netzzustand App und ist damit deutlich mehr als ein Cache

20. Mai 2015 6

open KONSEQUENZ

9. Entwurfsentscheidungen

Mandantenfähigkeit mittels Portaleigenschaften



Erstellung einer 2. Portal-Instanz mit einer „eigenen“ URL

Gleiche Portlets in jeder Portal-Instanz

Portlet-Instanzen verwenden eigene DB-Schemas

Ein ESB der verschiedene Endpoints für die Portlet-Instanzen anbietet

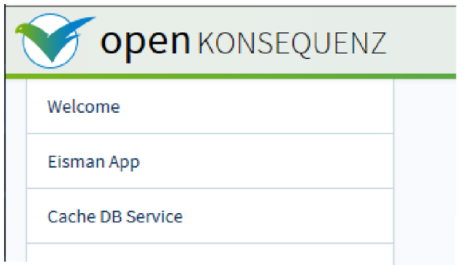
20. Mai 2015 7

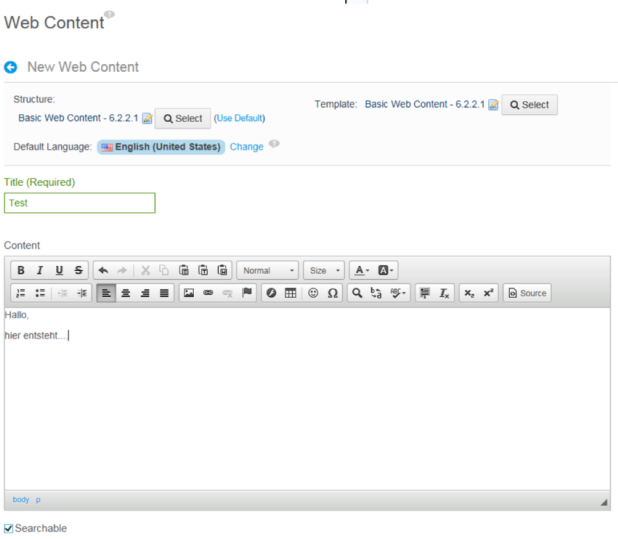
open KONSEQUENZ

9. Entwurfsentscheidungen

Aufgaben Portal

- Dynamisches Menü, das mit Portalmitteln administriert wird
- Mandantenfähigkeit mittels Portal-Instanzen
- Mail-Template mit Liferay-CMS erstellen
- Autorisierung & Authentifizierung
- Verwendung „wie ein Application Server“





9. Entwurfsentscheidungen

Objekt-Relationales Mapping mit JPA
Inheritance Strategy

Inheritance Strategies beeinflussen Performance und Wartbarkeit!

Single-Table Strategy

- eine Tabelle für mehrere Klassen wobei Spalten nur von einer bestimmten Klasse verwendet werden
- DiscriminatorValue definiert die Klasse
- EquipmentContainer: Substation und VoltageLevel

Joined-Table Strategy

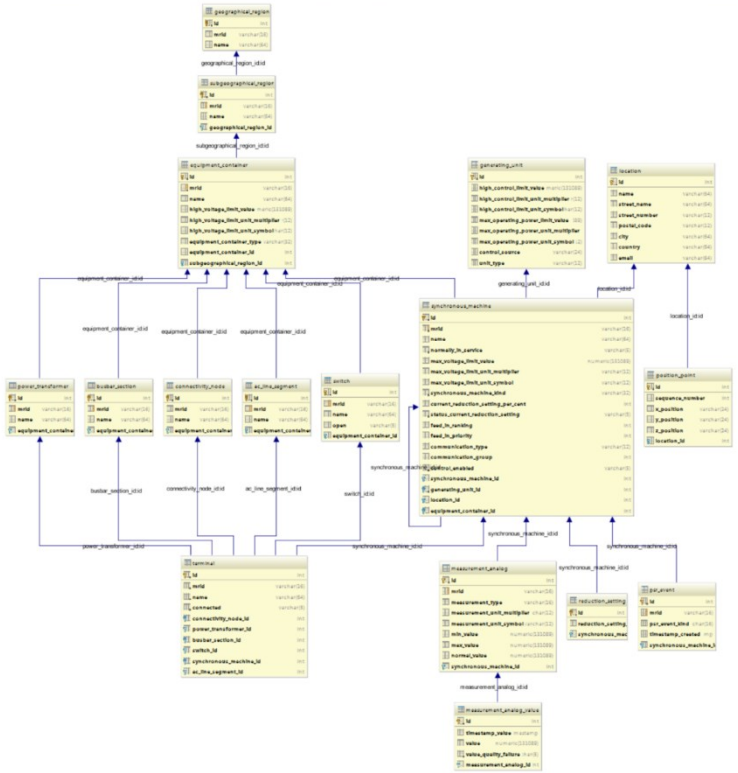
- Jede Klasse der Hierarchie wird auf eine Tabelle „gemapped“, die dann mittels Joins verbunden sind
- kann zu „teuren“ Joins führen
- nicht angewendet in openk platform

Table-Per-Concrete-Class Strategy

- nur konkrete Klassen „erhalten“ eine Tabelle
- Anwendung in Equipment zu Switch, SynchronousMachine, AcLineSegment, etc

9. Entwurfsentscheidungen

Datenbank Modell



Kommentar BTC:
Cache-DB ist erstmal nur für Eisman umgesetzt. Hier sollten sich Hersteller noch einmal zusammensetzen.

Umsetzung der CIM-Klassen in ein relationales Datenbankmodell unter Verwendung der „Inheritance Strategies“

Anforderungen an Datenmodell:

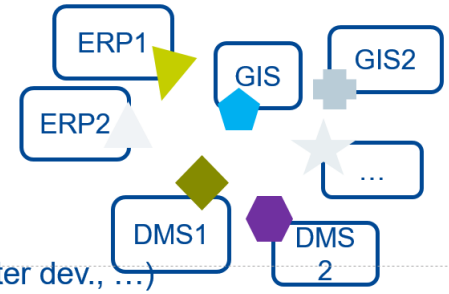
- Performance
- Wartbarkeit
- Erweiterbarkeit

9. Entwurfsentscheidungen

Standards for interoperability

Common system landscape of an utility:

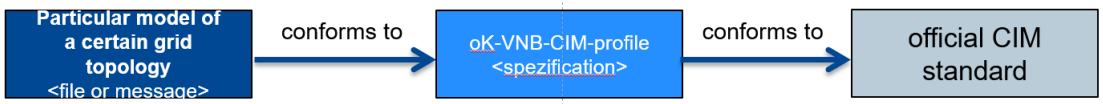
- Heterogeneous data models in relatively closed legacy systems
- Proprietary interfaces



➔ **High integration barriers and –costs slow down grid operators and software supplies**
(redundant data management, inconsistencies, adapter dev., …)

Approach:

- **open interfaces / data model based on CIM standard**
  (e.g., standardized semantics ➔ less misunderstandings; no proprietary knowledge, …)
- **Possibly development of shared DSO profiles comparable to entso-e profiles**



9. Entwurfsentscheidungen

Vom Blickwinkel der Systemarchitektur sind hinsichtlich der ausführbaren Programme folgende Aspekte zu beachten:

- **Unversehrtheit und Korrektheit von Basissoftware und Laufzeitumgebungen**: Durch die Wahl von Open Source-Produkten als Basis der Systemarchitektur können diese inspiziert werden, bevor sie in die jeweiligen Binärformate übersetzt werden. Wichtiger ist aber die Kontrollfunktion der Open Source Communities hinsichtlich der Korrektheit. Zusätzlich werden häufig Prüfsummen von den Herstellern angegeben, um die Unversehrtheit festzustellen.
- **Unversehrtheit und Korrektheit der Dienste**: Durch die Systemarchitektur muss festgelegt werden, dass Dienste nur nach eingehender Prüfung in die Laufzeitumgebung aufgenommen werden dürfen. Zur Prüfung der Korrektheit bieten sich Reviews und/oder Abnahmetests an. Dabei ist allerdings nur eine Prüfung der Dienste gegen die Spezifikation möglich, diese muss also gesondert geprüft werden. Ein so überprüfter Dienst kann digital signiert werden und somit seine Unverfälschtheit sichergestellt werden.
- **Unversehrtheit und Korrektheit der Endanwendungen**: Die Integrität von Endanwendungen kann über ähnliche Mechanismen gewährleistet werden wie bei den Diensten, durch eine Korrektheitsprüfung, gefolgt von der digitalen Signierung der Anwendung.

Grundsätzlich ist für Programme *vor deren Ausführung* sicherzustellen, dass sie korrekt sind. Über eine geeignete Testumgebung kann die Korrektheit auch untermauert, aber nicht abschließend bewiesen werden. Zur Laufzeit muss lediglich deren Unverfälschtheit geprüft werden.

Hinsichtlich der Daten sind folgende Aspekte zu beachten:

- **Vollständigkeit von Datensätzen**: Ein Aspekt der Korrektheit von Daten ist ihre Vollständigkeit. Um diese beurteilen zu können, wird ein Datenschema benötigt, das vorgibt, welche Informationen in einem Datensatz vorliegen müssen.
- **Konsistenz von Datensätzen**: Ein weiterer Aspekt der Korrektheit von Daten ist die Stimmigkeit eines Datensatzes. Um diese prüfen zu können, wird zusätzlich zu dem Datenschema noch eine Menge von Konsistenzregeln benötigt. Diese Regeln beziehen sich auf die Abhängigkeiten von Informationen in

einem Datensatz oder auch auf die Beziehung zwischen mehreren Datensätzen, beispielsweise auf die unverfälschte Reihenfolge.
- **Unversehrtheit von Daten**: Sind Vollständigkeit und Konsistenz eines Datensatzes geprüft, so muss deren Unversehrtheit auf dem Transportweg gesichert werden. Hier kommen, wie bei den ausführbaren Programmen auch, digitale Signaturen zum Einsatz.

Für Daten ist also während der Laufzeit an allen Punkten, an denen Daten verarbeitet werden, *vor der jeweiligen Berechnung* die Korrektheit zu prüfen, wohingegen die Unverfälschtheit *nur an Übergabepunkten* kontrolliert werden muss.

9. Entwurfsentscheidungen

Im Kontext der Integrität müssen zwei Aspekte gesondert betrachtet werden: Die Integrität wird durch fehlerhafte Verarbeitungsschritte verletzt, beispielsweise geht die Integrität eines Datensatzes durch fehlerhafte Berechnungen eines Programms verloren. Derartige Integritätsverletzungen können durch Prüfung auf Basis eines Datenschemas und durch Validierungsregeln aufgedeckt, aber nicht vollständig ausgeschlossen werden. Zudem können in einer Laufzeitumgebung auch unterschiedliche Varianten eines Dienstes parallel ausgeführt werden, um Ergebnisse vergleichen und damit verifizieren zu können (sofern man unabhängige Implementierungen der Dienste zur Verfügung hat).

Davon getrennt zu betrachten ist die absichtliche Verletzung der Datenintegrität durch Angreifer. Hier werden entweder Programme oder Daten verfälscht, um eine fehlerhafte Funktion hervorzurufen. Die Erkennung solcher Verletzungen lässt sich durch den Einsatz geeigneter Prüfsummenverfahren verbessern, siehe dazu auch [5].

Grundsätzlich sollte bei hohen Integritätsanforderungen jede Systemkomponente eigene Prüfungen vornehmen. Zusätzlich müssen die Daten an allen Übergabepunkten digital signiert werden. Bei Betrachtung des Anwendungsszenarios aus Abschnitt 3 zeigt sich, dass dementsprechend viele Prüf- und Signaturschritte notwendig sind. Diesem Aufwand kann man durch zwei Mechanismen begegnen. Zum einen bildet man mehrere Integritätsklassen unter den Systemfunktionen und verhindert so, dass tatsächlich für alle Systemfunktionen der größtmögliche Aufwand betrieben wird. Zum anderen kann man die Dienstspezifikationen entsprechend dem „Design by Contract“ Verfahren erweitern und damit die aufrufenden Komponenten dazu zwingen, bestimmte Integritätsbedingungen selbst zu erfüllen. Beide Mechanismen können in der Systemarchitektur festgelegt werden. Dabei stellen die einzelnen Systemkomponenten die Stellen dar, an denen Konsistenzprüfungen durchgeführt werden, die Netzwerktopologie bzw. die Interaktionen stellen die Übergabepunkte und somit die erforderlichen Signaturschritte dar.

- Die Endanwendungen sind von den Serverdiensten entkoppelt und ermöglichen so definierte Prüf- und Übergabepunkte.
- Das NLS als kritisches externes System ist durch Netzwerktopologie und logische Architektur vor Zugriffen geschützt, die nicht durch Integritätsmechanismen abgesichert sind.
- Eine eigene Integrationsschicht kann die Datenintegrität prüfen.
- Die Systemfunktionen sind in einzelne Dienste zerlegt, die jeweils für sich geprüft und signiert werden können.
- Die Laufzeitumgebung kann Endanwendungen, ankommende Daten und auch Dienste gegen Signaturen prüfen.
- Durch die Systemarchitektur wird das in [3] genannte Prinzip der mehrstufigen Absicherung („Defence in Depth“) umgesetzt, die Integrität von Daten wird auf jeder Hierarchieebene des Systems unabhängig von den anderen Schichten geprüft.
- Durchgängiger Einsatz von digitalen Signaturen setzt die von den Normen geforderte Absicherung gegen Verfälschung von Daten durch.
- Das ebenfalls in [3] genannte Prinzip der Minimierung von Informationszugang („Need to Know“) wird von der Systemarchitektur bei Verwendung eines standardisierten Datenmodells mit zugehörigen Zugriffsregeln umgesetzt.

Zu beachten ist:

- Ein gemeinsames Datenmodell auf Basis des CIM [6] muss erarbeitet und der Architektur zu Grunde gelegt werden. Damit erst können strukturelle Integritätsprüfungen zentral umgesetzt werden.
- Auf Basis des Datenmodells sollten Verfahren zur semantischen Integritätsprüfung entworfen und in der Architektur verankert werden.
- Die Spezifikationen der einzelnen Systemfunktionen sollten durch ein Architekturboard geprüft werden. So können auch wirksame Reviews und Tests durchgeführt werden.
- Die Qualifikation der Produkte sollte in einem unabhängigen Gremium „IT-Board“ institutionalisiert oder ausgelagert (siehe z.B. [7]) werden.
- Aus Aufwandsgründen ist es empfehlenswert, in der endgültigen Architektur unterschiedliche Klassen der Integrität vorzusehen. Beispielsweise ist die Datenintegrität der Schaltempfehlungen unbedingt einzuhalten, die von Prognosedaten nicht zwingend.
- Bei der Auswahl der Verfahren zur Integritätssicherung müssen Laufzeitaufwände an Hand einer realitätsnahen Testumgebung geprüft werden.
- Zur Prüfung der Korrektheit wird eine dem realen Einsatz ähnliche Testumgebung benötigt.

9. Entwurfsentscheidungen

Schutz der Kommunikationskanäle: Immer wenn Daten transferiert bzw. Dienste angeboten werden muss gewährleistet sein, dass die Kommunikation im System sowie zwischen Akteur und System vor dem Zugriff Dritter geschützt ist.

Die Systemarchitektur muss für alle drei Aspekte geeignete Mechanismen vorhalten. Zunächst ist erforderlich, dass eine Interaktion ausschließlich über einen Authentifizierungsdienst begonnen werden kann. Typischerweise werden hierzu in den Endanwendungen Mechanismen zur Identitätsabfrage verankert. Die serverseitigen Systemkomponenten kontrollieren die angegebenen Daten. Da ein Szenario der hier diskutierten Systemarchitektur den Einsatz einer Systeminstanz für mehrere Organisationen vorsieht muss allerdings auch eine Authentifizierung der Dienste untereinander und der Dienste gegenüber der Integrationsschicht vorgesehen werden. Insofern müssen Authentifizierungsdienste für alle aktiven Komponenten (Akteure, Dienste, extern angebundene Systeme) vorgesehen werden und an allen Interaktionspunkten (Endanwendungen, Application Server, Integration Server) geprüft werden.

Hinsichtlich der Autorisierung muss durch die Systemarchitektur ein Regelsystem vorgesehen werden, mit dem sich Zugriffsberechtigungen pro Interaktionspartner definieren lassen. Diese Berechtigungen müssen vor jeder Dienstanfrage bzw. jedem Datentransfer abgefragt werden (und der Interaktionspartner muss authentifiziert sein). Dabei unterscheidet man Zugriffe auf Dienste, auf Datentypen und auf Dateninhalte. Zugriffsschutz auf Dienste und auf Datentypen gehört in der Regel zum Funktionsumfang von Middleware. Zugriffsschutz auf Inhaltsebene bedeutet z.B. bei einer generellen Zugriffserlaubnis auf Kundendaten diejenigen Kundendaten zu sperren, denen nicht der zugreifende Akteur als Betreuer zugewiesen ist. Solche Funktionen müssen entweder als spezielle Anforderungen bei der Auswahl der Produkte berücksichtigt werden, oder die entsprechenden Dienste müssen spezifiziert und im Rahmen der Realisierung der Architektur implementiert werden.

9. Entwurfsentscheidungen

Hinsichtlich der Autorisierung muss durch die Systemarchitektur ein Regelsystem vorgesehen werden, mit dem sich Zugriffsberechtigungen pro Interaktionspartner definieren lassen. Diese Berechtigungen müssen vor jeder Dienstanfrage bzw. jedem Datentransfer abgefragt werden (und der Interaktionspartner muss authentifiziert sein). Dabei unterscheidet man Zugriffe auf Dienste, auf Datentypen und auf Dateninhalte. Zugriffsschutz auf Dienste und auf Datentypen gehört in der Regel zum Funktionsumfang von Middleware. Zugriffsschutz auf Inhaltsebene bedeutet z.B. bei einer generellen Zugriffserlaubnis auf Kundendaten diejenigen Kundendaten zu sperren, denen nicht der zugreifende Akteur als Betreuer zugewiesen ist. Solche Funktionen müssen entweder als spezielle Anforderungen bei der Auswahl der Produkte berücksichtigt werden, oder die entsprechenden Dienste müssen spezifiziert und im Rahmen der Realisierung der Architektur implementiert werden.

10. Offene Punkte

Folgende offenen Punkte lassen sich aus dieser Analyse zusammenfassend hervorheben. Alle davon sollten im Rahmen einer prototypischen Implementierung untersucht werden.

1. Die Anforderungen an die Systemarchitektur müssen detailliert werden.
2. Es ist nur eine grobe Abschätzung der zu behandelnden Datenmengen verfügbar. Hinsichtlich der Daten, die an Endanwendungen transferiert werden müssen gibt es noch keine Abschätzung.
3. Unbekannt ist die Laufzeitkomplexität der Algorithmen zur Datenaufbereitung. Damit lassen sich nicht ohne weiteres Anforderungen an Hardware bzw. Laufzeitumgebungen ableiten.
4. Ein durchgängiges Safety- und Security-Konzept ist für das Gesamtsystem noch nicht aufgestellt. Dazu müsste untersucht werden, welche Teile der Systemarchitektur welche Zusagen garantieren (z.B. könnte der Application Server Authentifizierung und Autorisierung garantieren).
5. Die Interoperabilität der unterschiedlichen Produkte miteinander und die bei der Zusammenarbeit anfallenden Protokolldaten sind noch unbekannt. Insbesondere ist auch der Verbindungsprotokolloverhead unbekannt, der ggf. für erhebliche zusätzliche zu transferierende Daten sorgt.
6. Eine Strategie für die Persistenz, für Backup-Speicherungen und für das Protokollieren von Normalbetrieb sowie von Fehlerfällen muss ausgearbeitet werden.
7. Eine Strategie für ein planmäßiges Austauschen von Software-Komponenten im laufenden Betrieb (Applikations-Updates) muss erarbeitet und getestet werden.

9. Entwurfsentscheidungen

- Status
  - Eine Reihe der Entwurfsentscheidungen ist bereits implizit gemacht und umgesetzt worden
  - Architekturanalyse von Herr Jung hat zusätzlich eine Reihe weiterer Empfehlungen ausgesprochen

- ToDos
  - Review und „einfache“ Nachdokumentation der wichtigsten Entwurfsentscheidungen
  - Offene Architekturfragen im Hinblick auf die Relevanz für die geplanten Module priorisieren und Rahmenbedingungen festlegen

10. Qualitätsszenarien und Teststrategie

- 10. Qualitätsszenarien und Teststrategie: Qualitätsbaum sowie dessen Konkretisierung durch Szenarien

- Annahmen und zu beachtende Aspekte
  - Qualitätsanforderungen müssen als Qualitätszenarien dokumentiert werden
  - Neben den Sprint-Reviews sollte eine (teil-)formalisierte QS eingeführt werden (Wiederholbarkeit der Tests)
    - Testskripte und Testdrehbücher für Abnahmen von Releases (gerne vollautomatisiert), Testdatensätze
    - Inbetriebnahmetests

10. Qualitätsszenarien und Teststrategie

Qualitätssicherungsmaßnahmen:

- automatische Style-Checks
- Code-Reviews
- Unit-Tests
- automatischer Build-Prozess
- klassische SW-Test im Rahmen von QS-Maßnahmen
- zeitnahe Tests durch Kunden

10. Qualitätsszenarien und Teststrategie

- Status
  - Manuelles Testen durch Produktowner
  - Unit Tests im Ermessen der Entwickler

- ToDos
  - Minimal-Set an Test- und Abnahmeregeln dokumentieren
  - Zusammen mit Anforderungen gleich die „Abnahmekriterien“ festlegen
  - Ggf. „einfache“ Kennzahlen für UnitTesting festlegen

54 Zusammenfassung

- Es gibt viel zu tun ☺
- Pragmatische und kompakte Dokumentation ist weniger „umständlich“ als es scheint.

- Feedback zu den einzelnen Themen wird protokolliert und bei der Ausarbeitung der Architekturdokumentation berücksichtigt

- Offene Punkte werden im AC/QC Backlog gesammelt und priorisiert. Review und Freigabe der Priorisierung erfolgt im Umlaufverfahren