Architecture- & QS-Board – 4. März 2015

openK platform für openKONSEQUENZ

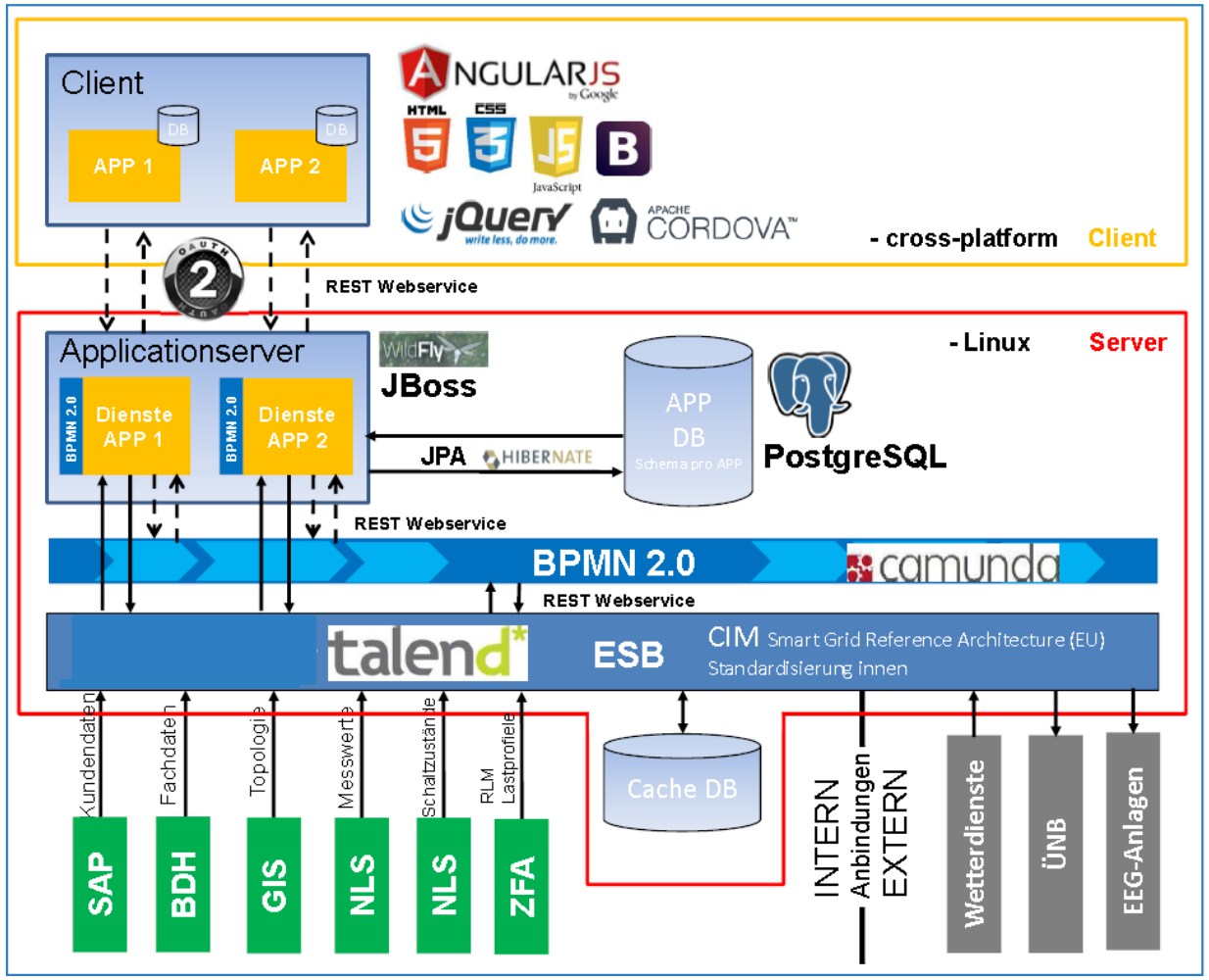
Matthias Rohr & Jan Krüger

open KONSEQUENZ

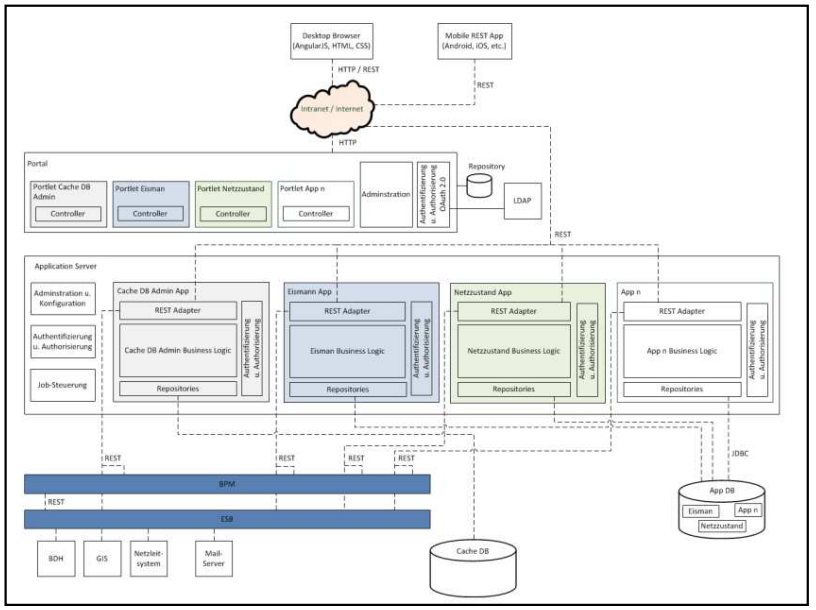
Agenda

1. Vorstellung Architektur – Big Picture
   (Referenzarchitektur => Architektur BTC)
2. Darstellung der Architekturkomponenten und deren Aufgaben
3. CIM
4. Build Process und Maven Projekte
5. Qualitätssicherung
6. Referenz-Plattform Darmstadt & Software-Versionen
7. Domain Driven Design nach Eric Evans
8. Clean Code

openKONSEQUENZ Architekturentwurf

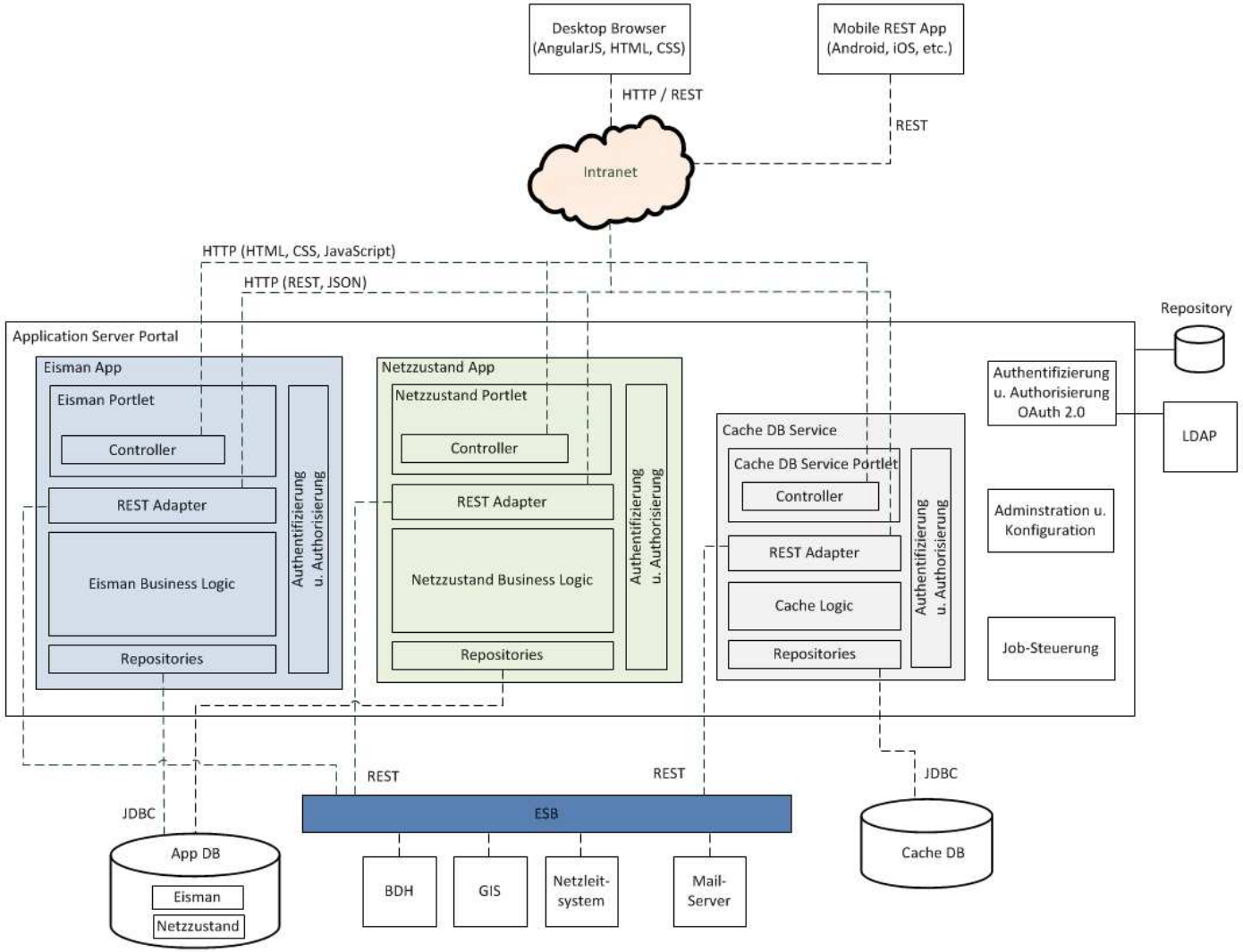


BTC Angebot



open KONSEQUENZ

IT-Architektur



Architekturkomponenten: Portal

Darstellung der Benutzeroberflächen

Implementierung in Portlets (Portlet Specification 2.0 (JSR 286))

- HTML
- JavaSript: AngularJS
- CSS: Bootstrap
- Java: Spring MVC (nur in Controller)

Warum Liferay Portal?

- Open Source Portal mit großer Community
- Delegation an das Portal: Autorisierung/Authentifizierung, Benutzerverwaltung, Menüstruktur, etc.
- Liferay Portal wird als Laufzeitumgebung verwendet, die nützliche Features mitbringt und die Entwicklung beschleunigt sowie den Betrieb vereinfacht.

Architekturkomponenten: AngularJS

JavaScript-Framework für dynamische Web-Anwendungen

Entwicklung von Single-Page Web-Anwendungen
nur **eine** HTML-Seite! der „Rest“ ist JavaScript!

Entwickelt von Google als Open-Source seit 2009

MVC (Model-View-Controller) Framework

open KONSEQUENZ

AngularJS ermöglicht einfachen Code

Code-Reduzierung
DOM-Bearbeitung oder Event-Handling wird von AngularJS übernommen

Dependency Injection
durchgängiges Konzept in AngularJS

Bi-directional Data-Binding
Änderungen in der View werden an das Model gegeben

Directiven
AngularJS weist einem DOM-Element ein bestimmtes Verhalten zu
vergleichbar mit Komponenten

```
app.factory('nominationresource', ['$resource', function ($resource) {
        return $resource('http://localhost:8080/nomination-process/nominationdocumentrestservice/nominationdocuments', {},
            {query: {method: 'GET', responseType: 'json'}}
        );
    }
]);
app.controller('NominationListCtrl', function ($scope, nominationresource) {
    $scope.nominations = nominationresource.query();
});
```

Erstellung

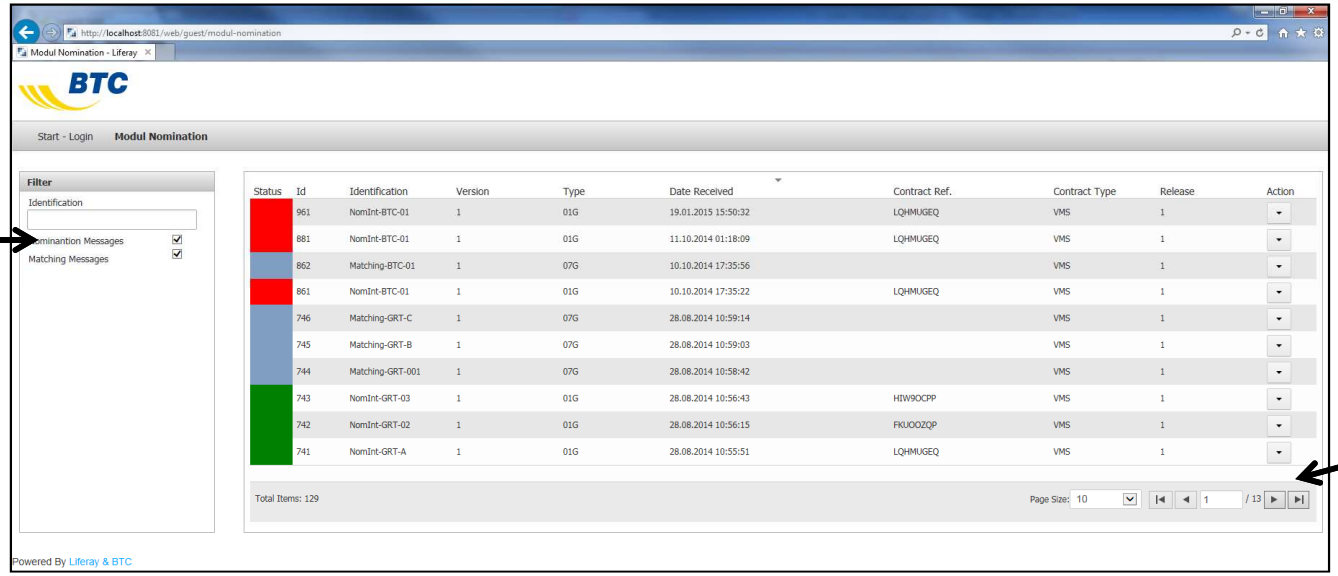
Verwendung

```
<div ng-app="app" ng-controller="NominationDocumentGridController">

    <div id="sidebar-container" class="sidebar-container">
        <div class="input-group">
            <label for="filter-option-identification">Identification</label>
            <input id="filter-option-identification" type="text" class="form-control" placeholder="" ng-model="filterOptions.filterText">
            <input type="submit" value="Filter anwenden"/>
        </div>
    </div>

    <div class="content-container">
        <div class="gridStyle" ng-grid="gridOptions"></div>
    </div>

</div>
```

einfacher, wiederverwendbarer
& modularer Code

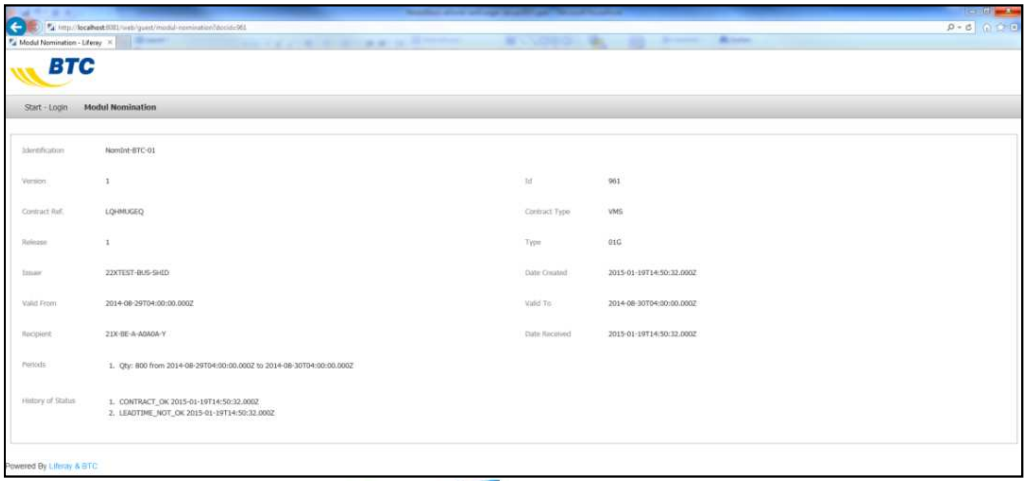
open KONSEQUENZ

AngularJS – less talk more music



Filter

Paging



2 x HTML 30 bzw. 130 Zeilen
1 x JavaScript 170 Zeilen

open KONSEQUENZ

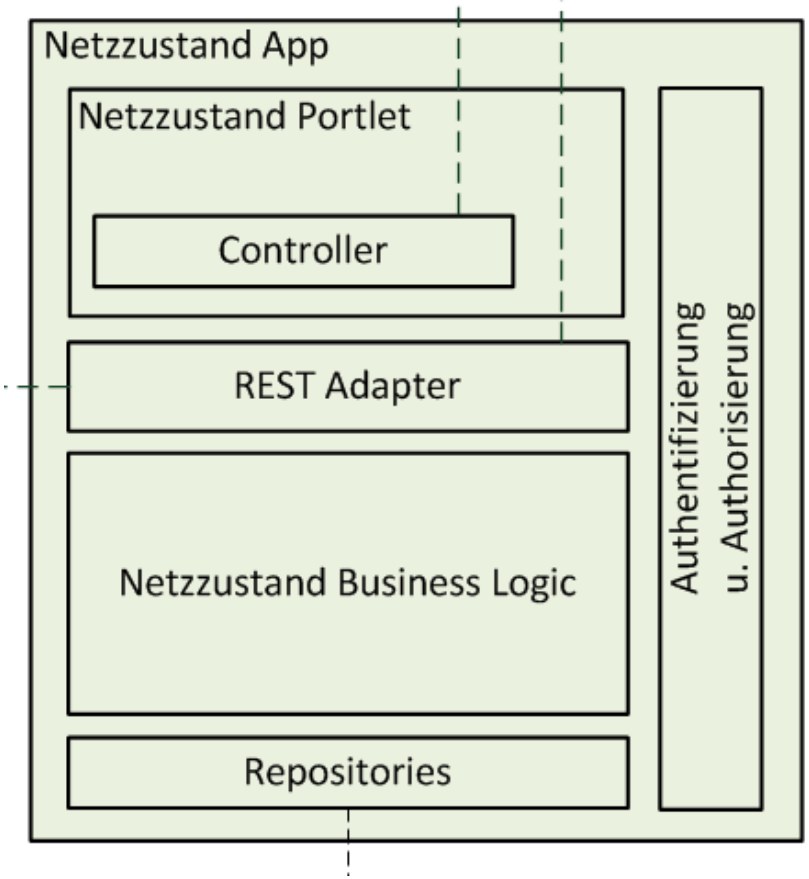
Application Server & Apps

Application Server

- Zentrale Komponente für die Business Logic
- Laufzeitumgebung für Apps & Portlets
- Job-Steuerung
- Authentifizierung u. Autorisierung

Apps

- Portlet
- REST-Adapter
- Business Logic
- Repositories
- Authentifizierung & Authorisierung



open KONSEQUENZ

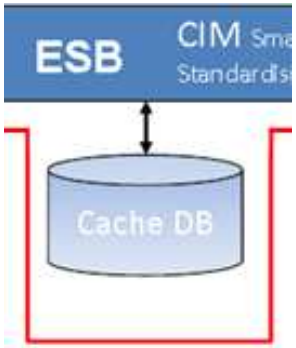
Cache DB Service

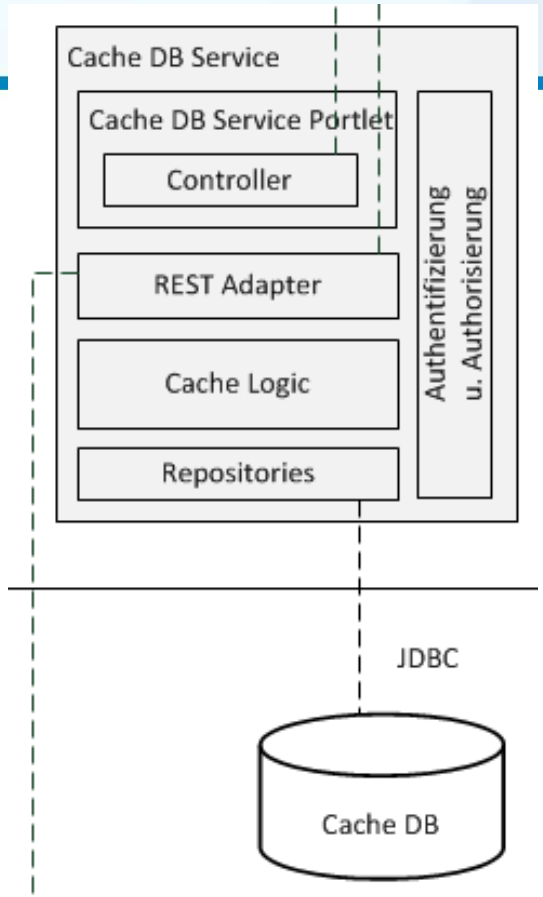
Cache DB

- Datenhaltung von Dritt-Systemen mit zu *geringer Verfügbarkeit*

Cache DB Service

- Nur der Cache DB Service schreibt Daten in die Cache DB
- Der Cache DB Service stellt via REST die Daten der Cache DB den anderen Komponenten zur Verfügung

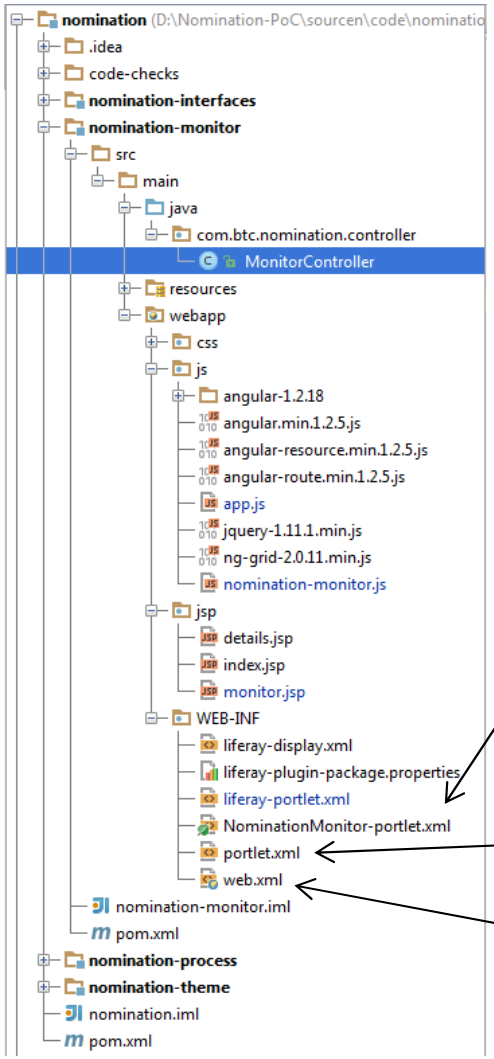




- Initiale- und Delta-Befüllung der Cache DB
- Aufruf von REST-Sevices, die BDH, GIS, etc. zur Verfügung stellen.
- Benutzeroberfläche für Administration u. Monitoring

open KONSEQUENZ

Entwicklung Spring Portlets mit Spring MVC
Portlet Specification 2.0 (JSR 286)



IDE Projektstruktur

Spring Konfiguration
web application context file: [portlet_name]-portlet.xml

```
<portlet>
  <portlet-name>NominationMonitor</portlet-name>
  <portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>view</portlet-mode>
  </supports>
  <portlet-info>
    <title>ift Document Center</title>
  </portlet-info>
</portlet>
```

Konfiguration Spring MVC

```
<servlet-class>org.springframework.web.servlet.ViewRendererServlet</servlet-class>
```


open KONSEQUENZ

REST – mehr als eine Technologie

Keine Technologie!

Eine Architektur für die Entwicklung von verteilten Systemen

REST Architektur Prinzipien

Addressability
Alle Objekte und Ressourcen sind über einen Unique Identifier (URI) adressierbar
scheme:*//host:port/path?queryString#fragment*

Uniform, Constrained Interface
zur Verfügung stehen eine definierte Menge von HTTP-Operationen:

GET: read-only	PUT: insert or update
DELETE: delete	POST: modify service in unique way
HEAD: get but only response code	OPTIONS: get Meta-Data and capability of service

⇒hohe Wiedererkennbarkeit, einfaches Programmiermodell, optimale Interoperability

⇒Zustandslose Kommunikation und damit hohe Skalierbarkeit

REST ist protokoll-unabhängig
(nicht nur REST over HTTP, bzw. SOAP ohne Header)

REST – die Implementierung

Umsetzung mit Spring via @RestController

```java
@RestController
public class NominationDocumentRestController {

    @Inject
    private NominationDocumentService nominationDocumentService;

    @RequestMapping(value = "/nomination/document", method = RequestMethod.GET)
    public Page<NominationDocument> listDocuments(Pageable pageable, @RequestParam(value = "q", required = false) String query) {
```

Spring Data REST

Spring Data REST exportiert Spring Data Repositories als REST Ressourcen
=> Standardisierter Weg die Operationen auf den Entity-Objekten als REST Services zur Verfügung zu stellen

Security-Aspekte der REST-Service

Verwendung REST in AngularJS mit Module ngResource

```javascript
app.factory('nominationresource', ['$resource', function ($resource) {
    return $resource('http://localhost:8080/nomination-rest/nominationdocuments', {},
        {query: {method: 'GET', responseType: 'json'}}
    );
}]);
```


open KONSEQUENZ

Objekt-Relationales Mapping mit JPA Eclipselink

JPA: Java Persistence API

- Bestandteil J2EE vs. proprietäre Lösung
- JPA 2.0 Referenzimplementierung von Eclipselink (nicht Hibernate!)

Vorteil:

- Standard bei BTC, alle Entwickler kennen JPA
- Testbarkeit

Repository bzw. DAO

```
@Repository
public class PruefberichtRepositoryJpa extends
                AbstractImportDbRepositoryJpa<Long, Pruefbericht> {

    public PruefberichtRepositoryJpa() {
        super(Pruefbericht.class);
    }

    public List<String> getKundenMailAdressen(Long id) {
        Query q = entityManager.createNamedQuery("Pruefbericht.getKundenById");
        q.setParameter("id", id);
        return q.getResultList();
    }
…
```

Model

```
@Entity(name = "Pruefbericht")
@NamedQueries({
        @NamedQuery(name = "Pruefbericht.getPruefberichtListByKundeId",
                query = "select pb from Pruefbericht pb where pb.kunde.id = :id")
})
@Table(name = "PRUEFBERICHTE")
@Access(value = AccessType.FIELD)
public class Pruefbericht extends AbstractBaseEntity {

    @NotNull
    @Column(name = "DATUM_ERSTELLUNG")
    private Timestamp erstellungsDatum;

    @NotNull
    @Column(name = "PRUEFBERICHT_STATUS")
    private int status;
```

Bean Validation →

open KONSEQUENZ

BPMN & ESB

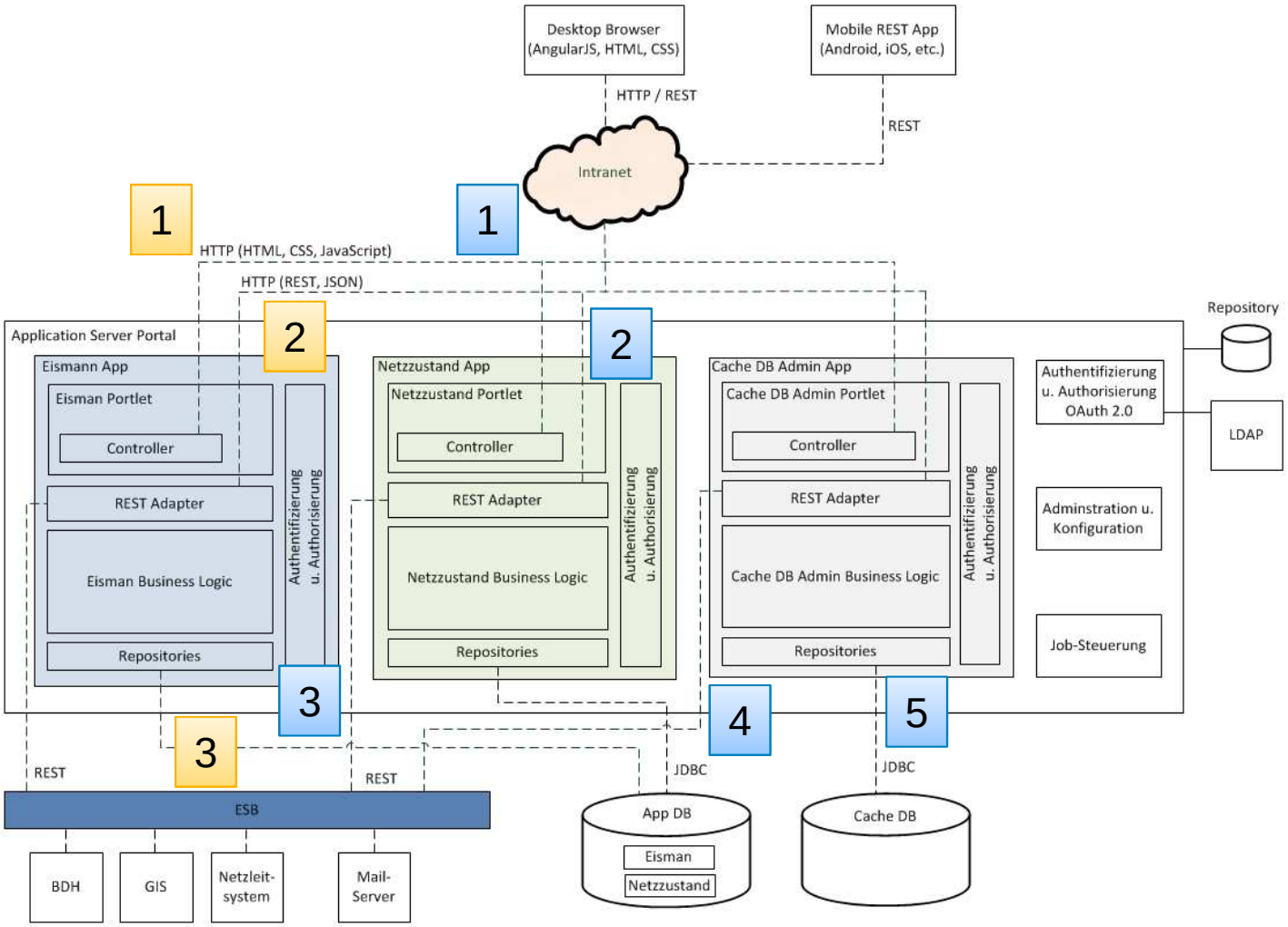
Integrationsplattform als zentrale Drehscheibe (Hub) für den Datenaustausch zw. IT-Systemen.
IT-Systeme dürfen NUR unter Verwendung der Integrationsplattform miteinander kommunizieren!

Pilotphase beinhaltet nur zustandslose Prozesse, in denen primär das Mapping zw. Datenstrukturen implementiert ist.

Pilotphase setzt nur ESB ein, in dem möglichst keine Business Logic implementiert ist.

ESB *spricht* das kanonische – auf CIM basierende - Datenmodell

IT-Architektur Pilotprojekt



Darstellung Abschaltempfehlung

1. Request Browser
2. REST-Request Browser
3. JDBC-Request an App DB Schema Eismann

Darstellung View Netzzustand

1. Request Browser
2. REST-Request Browser *wenn nicht in Netzzustand App verfügbar*
3. *REST-Request an ESB*
4. *REST Request an Cache DB Admin App*
5. *JDBC-Request an Cache DB*

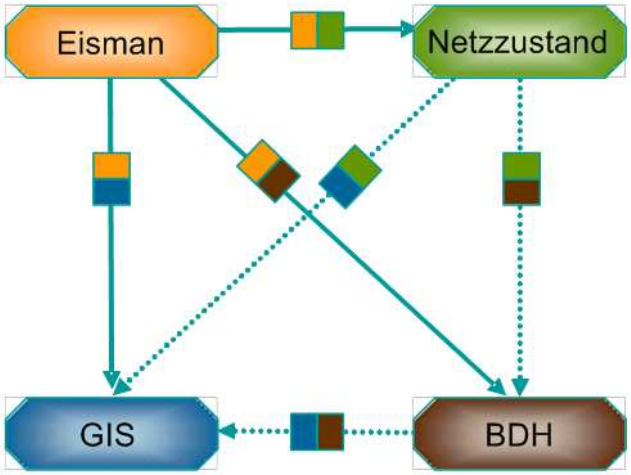
open KONSEQUENZ

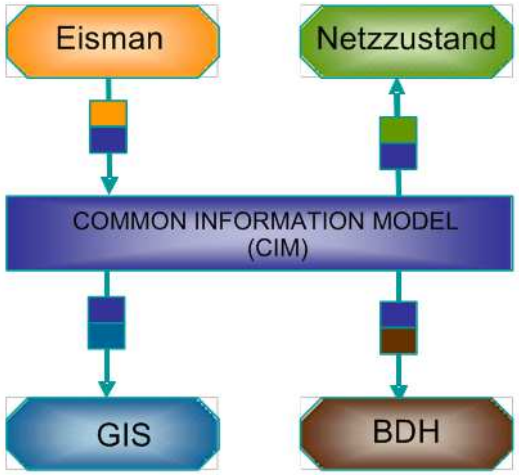
CIM als Standard für den Datenaustausch

Das Common Information Model (CIM, IEC 61970 u. IEC 61968) wird als Datenmodell für den Informationsaustausch zwischen

- den Apps untereinander
- den Apps mit Quellsystemen wie GIS, BDH/ERP, und GIS genutzt

CIM ist die gemeinsame Sprache auf dem ESB.



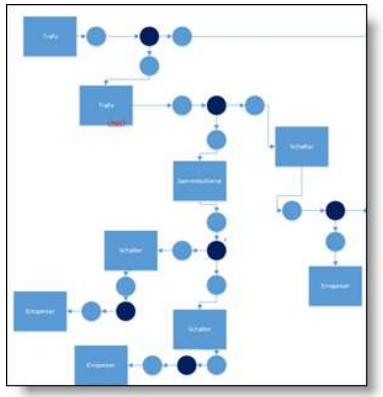


open KONSEQUENZ

Vereinfachte Sicht auf CIM fürs Projekt

- CIM ist ein herstellerunabhängiger Standard, der als Konvention zwischen verschiedenen Softwarelieferanten, Softwaredienstleistern und Ihren Nutzern (z.B. VNBs) genutzt werden kann.
- CIM definiert Entitäten (z.B. Generator, Switch, PowerTransformer) inkl. Attribute und Beziehungen
- Diese Entitäten, Attributen, Beziehungen sind nützlich als
  1. Konvention für Nachrichten zwischen Systemen (z.B. Meldungen, Messwerte)
  2. Konvention für Topologien
  - (CIM ist nützlich sowohl als Spezifikation, wie auch für den Aufbau einer konkreter Nachricht bzw. eines Topologie-Austauschartefakts)
- CIM ist insgesamt umfangreich und mächtig – daher kann es für bestimmte Zwecke (z.B. Einspeisemanagement) sinnvoll sein eine Teilmenge zu benennen (als CIM Profil) und zusätzliche Konventionen zu definieren. Mehrere solcher Anwendungsfälle u. Profile lassen sich auch zusammenfassen.
- Praktische Vorteile von CIM als hersteller- und technologieunabhängiger Standard
  - Verbessert Integrierbarkeit und Interoperabilität
  - Erhöht Flexibilität und reduziert Vendor-Lock-in
  - Verbessert Zuverlässigkeit (weniger Schnittstellenmissverständnisse)

Attribute zu einem MeterReading



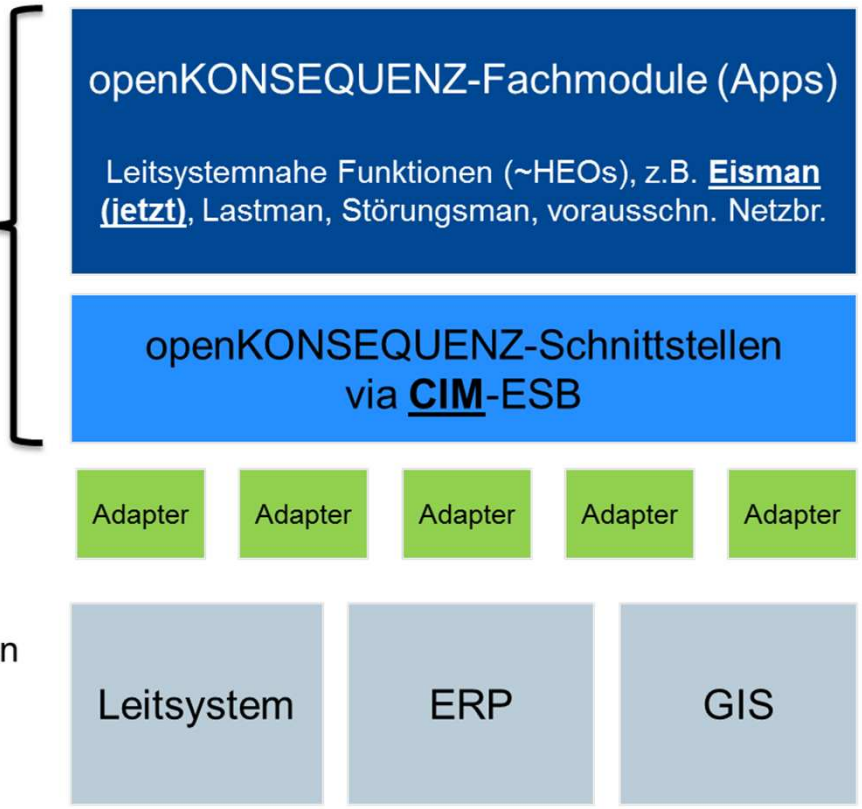
Objektmodell einer konkreten Topologie

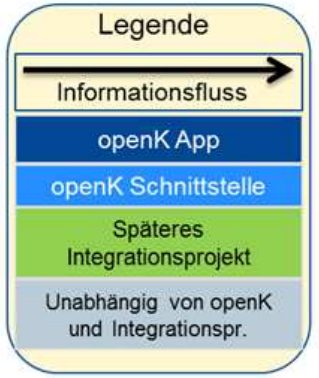
open KONSEQUENZ

Topologie-Modell für openKONSEQUENZ

Aktueller Auftrag:
- Eisman-App & CIM-Schnittstellen u. ESB
- **Keine Integration/Adapter**
- **Alles als open source**
- **Chance für Industriestandard**

Schnittstellen:
- Auf Basis CIM sein
- Fachlich:
  - Nachrichten/Informationslisten (z.B. Liste von Einspeiser)
  - **Topologiemodell (neue Fokus KW7)**

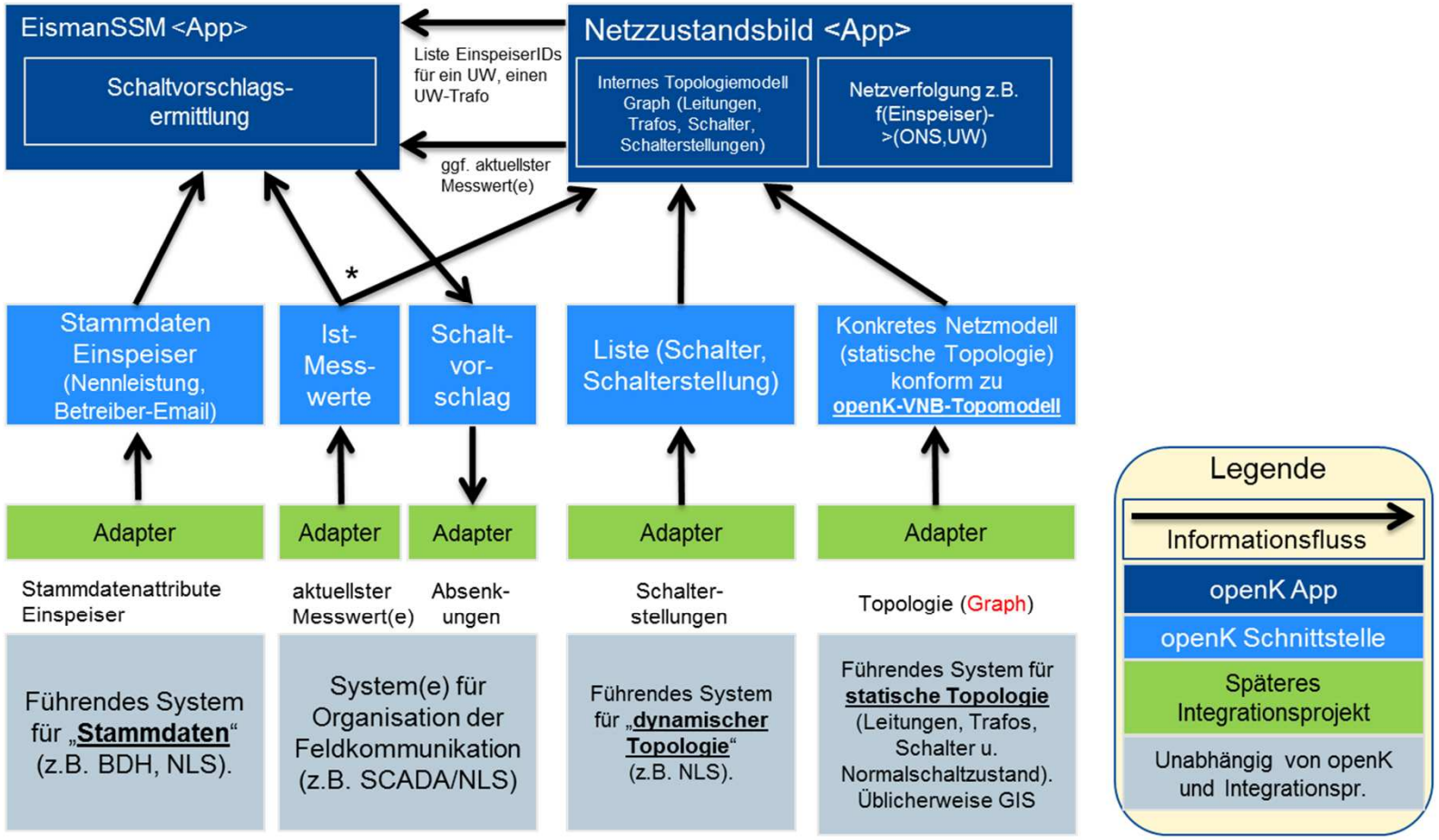




- Es soll jetzt ein Topologiemodell spezifiziert werden, worauf sich Adapter-Stakeholder grob einstellen können.
- Idee: 1. Schritt Richtung Profil für VNB-Topo-Austausch: Erlaubte CIM Klassen, Beziehungen, Attribute, Beispiel

open KONSEQUENZ

Informationsfluss und Schnittstellen im Überblick
(Unvollständige Auswahl, Stand Architektur 25.02.2015 – 1. Iteration)



* Weg der Messwertkommunikation noch in Klärung

open KONSEQUENZ

Spezifikation für Topologie-Modelle (Stand 25.2.2015)

- Beschreibt ein konkretes Verteilnetz als Graph

- Wird z.B. von einem GIS-Adapter/Service täglich aktuell erzeugt
- Enthält einen Netzgraph bestehend aus u.a. Leistungen, Trafos, Schalterstellungen, Einspeisern, Normal-schaltzustand

- Dies spezifiziert **wie** eine Topologie beschrieben ist

- Ist Spezifikation für jemanden der z.B. einen GIS-Adapter schreiben will.
- Spezifikation ist eine Teilmenge von Entitäten, Beziehungen, Attribute aus CIM-Standard zzgl. Beispielnetz
- Ggf. ist Spezifikation Word-Datei + UML-Modell

- Stellt sehr viele Entitäten und Semantik bereit

- Nur kleine Teilmenge relevant

open KONSEQUENZ

Entwicklungsprozess mit Maven

maven

liferay-maven-plugin für Deployment

```
<plugin>
  <groupId>com.liferay.maven.plugins</groupId>
  <artifactId>liferay-maven-plugin</artifactId>
  <version>${liferay.version}</version>
  <configuration>
    <autoDeployDir>${liferay.auto.deploy.dir}</autoDeployDir>
    <liferayVersion>${liferay.version}</liferayVersion>
    <pluginType>portlet</pluginType>
  </configuration>
</plugin>

mvn clean package liferay:deploy
```

maven archetype für Liferay

```
mvn archetype:generate
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Nomination - Master ............................... SUCCESS [  4.417 s]
[INFO] Nomination - Nomination Interfaces ................ SUCCESS [ 12.467 s]
[INFO] Nomination - Nomination Process ................... SUCCESS [ 36.575 s]
[INFO] Nomination - Nomination Monitor ................... SUCCESS [  4.586 s]
[INFO] Nomination - Nomination Theme ..................... SUCCESS [01:11 min]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:09 min
[INFO] Finished at: 2014-09-11T21:34:29+01:00
[INFO] Final Memory: 66M/398M
[INFO] ------------------------------------------------------------------------
```


open KONSEQUENZ

Qualitätssicherung QS

Git als Quellcodeverwaltungssystem

- alle Anpassungen des Codes sind nachvollziehbar
- Releases und Tags => reproduzierbare und definierte Projektergebnisse

Bug-Tracking mit Bugzilla

- Fehler und deren Bearbeitungshistorie sind dokumentiert

Coding Standards

- Eclipse Development Conventions and Guidelines (http://wiki.eclipse.org/Development_Conventions_and_Guidelines)

Kontinuierliche Messung der Testabdeckung

- hohe Testabdeckung > 50% Abdeckung der „Entscheidungen und Verzweigungen“ werden angestrebt

Testkonzept => strukturiertes und dokumentiertes Testvorgehen

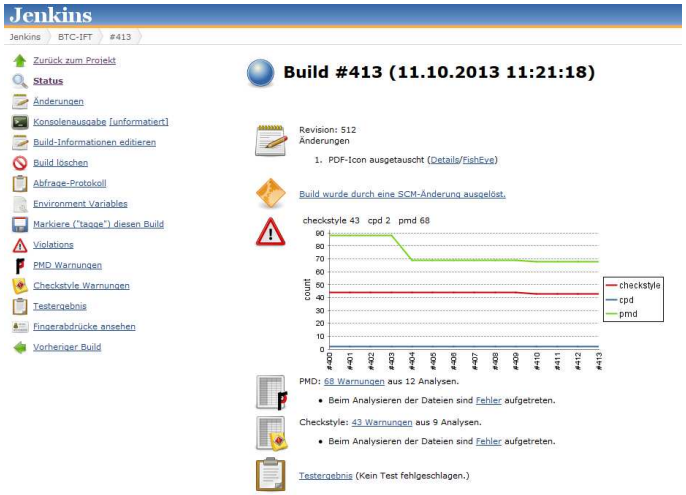
- Erstellung eines Testkonzepts unter Mitwirkung des Konsortiums bzw. Product Owner
- Erstellung von Testfällen
- Abnahmetest

QS Continous Build
Jenkins, TestNG, Sonar, PMD & Checkstyle

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 {
…
}
```

Continuous Build



Checkstyle

Referenz-Plattform

Aufbau der Referenzplattform durch e-netz Südhessen

Installation der Komponenten nach Vorgaben der Referenz-Architektur

- Liferay V. 6.2 BOSS Wildfly Application Server
- Postgres DB V. 9.4.1
- Talend ESB V. 5.6.1
- JDK V. 1.7
- Spring V. 4.1

Deployment der Entwicklungsartefakte

Domain Driven Design (DDD) by Eric Evans

DDD ist ein Design Pattern => Vorgabe für die Entwicklung

Basis sind Fachdomänen, die projektweit verwendet werden und ein gemeinsames und einheitliches Verständnis für die Fachlichkeit des Projekts unterstützen.

Ubiquitous Language für Modelle, Sourcen, Tests, …

Domain Objects:

- Identität
- Lebenszyklus
- Implementierung von Geschäftslogik

Value Objects

- keine Identität, kein Lebenszyklus, z.B. Messwert
- keine Geschäftslogik ~ „dumme Container“

Process Objects

- Implementierung übergeordneter Prozesse
- besitzen keinen „eigenen“ Zustand
- verwenden Domain- und Value Objects

Clean Code

Robert Martin führt den Begriff in der Software-Technik ein

Als ***Clean*** werden Sourcecode, Algorithmen, Konzepte, Dokumente, etc. bezeichnet, die ***intuitiv verständlich*** sind.

The Total Cost of Owning a Mess!

Je größer der Mess, je geringer die Produktivität!

Re-Factoring in dem Projekt ermöglichen und planen

Clean Code beschreibt eine Vielzahl von Maßnahmen:

- Boy Scout Rule: Leave the campground cleaner than you found it.
- Meaningful Names
- Functions (do one thing, have no side effects, …)
- etc.

Diskussion & Fragen?