Bundesverwaltungsamt

IsyFact-Standard

IsyFact – Tutorial

Version 1.13
25.04.2016

IsyFact Standards

Ansprechpartner:

Referat Z II 2
Bundesverwaltungsamt
E-Mail: isyfact@bva.bund.de
Internet: www.isyfact.de

Dokumentinformationen

Dokumenten-ID:	IsyFact-Tutorial.docx

Inhaltsverzeichnis

1. Zusammenfassung

Das IsyFact-Tutorial gibt praktische Anleitungen, wie die Konzepte der IsyFact in der Softwareentwicklung umzusetzen sind. Es hilft technischen Chefdesignern und Entwicklern einen Überblick über alle wichtigen Aspekte der IsyFact-konformen Anwendungsentwicklung zu erhalten und unterstützt mit Schritt-für-Schritt-Anleitungen bei der Realisierung.

2. Einleitung

Dieses Dokument dient als Einstieg in die technischen Grundlagen der IsyFact. Es enthält in Kurzform die wesentlichen Designvorgaben und beschreibt, wie einzelne Aspekte in Anwendungen umzusetzen sind, die gemäß der Referenzarchitektur der IsyFact gebaut werden. Das Dokument unterstützt damit Entwickler und technische Chefdesigner bei der Konstruktion und Realisierung von Anwendungen einer zur IsyFact konformen Anwendungslandschaft. Im Sinne einer Referenz enthält es jedoch nicht alle Details der zugrunde liegenden Konzepte. Der Fokus liegt darauf, einen schnellen Überblick über die einzelnen Themen zu geben und die wichtigsten Realisierungsaufgaben zusammenfassend darzustellen. An vielen Stellen bezieht sich das Tutorial auf die Beispielimplementierung, die unter [VorlageAnwendung] zu finden ist. Zu beachten ist, dass die Vorlage Anwendung eine Register-Anwendung ist, und damit einen Spezialfall einer Fachanwendung realisiert (siehe [IsyFactEinstieg]). Die in diesem Dokument beschriebenen Konzepte lassen sich jedoch ohne weiteres auf beliebige Fachanwendungen übertragen, bei denen keine Aufteilung in ein Register und eine Geschäftsfanwendung vorliegt.

In den Kapiteln 3 bis 5 werden jeweils einzelne Kernthemen der Anwendungen beschrieben. Kapitel 6 fasst alle Querschnittsthemen zusammen. Jedes Teilkapitel beginnt in der Regel mit der Auflistung der wichtigsten Designvorgaben und einem Klassendiagramm. Es folgt eine Realisierungsanleitung, die alle wesentlichen Schritte zur Umsetzung in Kurzform enthält.

3. Datenhaltung

In diesem Kapitel wird die Implementierung der Datenhaltung einer Anwendung beschrieben. Die Datenhaltung besteht im Wesentlichen aus den Entitätsklassen und den zugehörigen Data Access Objects (DAOs).

3.1. Designvorgaben

- Der eigentliche Datenbankzugriff wird über Java Persistence API (JPA) gekapselt. Als Implementierung für die JPA wird Hibernate verwendet.

- Das Datenbank-Mapping wird über Annotation in den Entitätsklassen festgelegt.

- Der Zugriff auf Entitäten erfolgt immer über DAOs oder durch die Traversierung des persistenten Datenmodells.

- DAOs stellen Zugriffsmethoden (Suchen, Schreiben, Löschen…) für genau einen Entitätstyp bereit.

- Datenbank-Queries werden nach Möglichkeit als Named Queries abgelegt.

- Entitäten werden entweder von genau einer fachlichen Komponente oder querschnittlich von der Komponente „Basisdaten“ verwaltet.

- Der Datenzugriff erfolgt entweder über die Komponenten-Schnittstelle oder im Fall der Basisdaten direkt über die Basisdaten-DAOs.

- Für persistente Entitäten, die über eine Komponentenschnittstelle angeboten werden, werden eigene Schnittstellenobjekte definiert, die per Deep-Copy gefüllt werden. Hierfür wird der Bean-Mapper Dozer verwendet. Innerhalb einer Teilanwendung dürfen persistente Entitäten allerdings über Komponentengrenzen hinweg genutzt werden.

- Die Transaktionssteuerung wird per AOP[1] (Aspect-Oriented Programming) in der Service-Schicht durchgeführt (siehe Kapitel 5.1).

- Der Primärschlüssel einer Entität besteht aus genau einem Attribut. Besteht der fachliche Schlüssel der Entität aus genau einem Attribut, so wird er auch als Primärschlüssel verwendet. Ansonsten wird ein künstlicher technischer Primärschlüssel verwendet.

[1] AOP ist ein Programmierparadigma, das anstrebt, verschiedene logische Aspekte eines Programmes getrennt voneinander zu entwerfen, zu entwickeln und zu testen. Die getrennt entwickelten Aspekte werden dann zur endgültigen Anwendung zusammengefügt.

3.2. Packagestruktur

- Die DAOs- und Entitätsklassen werden im Persistence-Package der entsprechenden Komponente implementiert.

- Für DAOs ist dies
  `<organisation>`[2]`.<domäne>.<system>.persistence.<komponente>.dao.impl`
  Dazu wird für jedes DAO ein Interface in
  `<organisation>.<domäne>.<system>.persistence.<komponente>.dao`
  angelegt

- Entitäten werden in
  `<organisation>.<domäne>.<system>.persistence.<komponente>.entity`
  implementiert

- Die Schnittstellen-Klassen sind Teil der Schnittstelle der zugehörigen Komponente und werden daher im Package der entsprechenden Komponente implementiert:
  `<organisation>.<domäne>.<system>.core.<komponente>.ausgabedaten`

3.3. Klassendesign

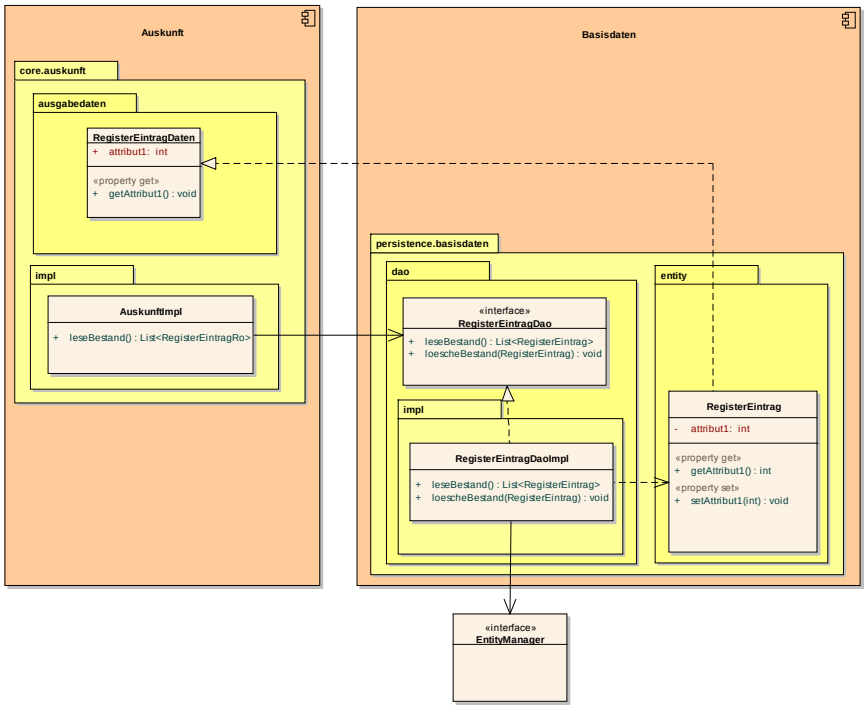

Abbildung 1: Klassendesign für die Datenhaltung.

[2] z.B. `de.bund.bva`

AuskunftImpl	`AuskunftImpl` ist die Implementierung der Komponente Auskunft (Anwendungskern). Die Komponente verwendet die DAOs zum Zugriff auf ihre Daten. Während die DAOs die eigentlichen Entitätstypen anbieten, liefert die Komponente Auskunft nur nicht persistente Schnittstellenobjekte auf ihre Daten.
RegisterEintragDao	DAOs kapseln den Datenbankzugriff für spezifische Entitätstypen. Sie stellen Methoden zum Suchen, Anlegen und Löschen dieser Typen zur Verfügung. Zur Durchführung der Datenbank-Queries benötigt das DAO den `EntityManager`. Dieser wird von Spring bereitgestellt.
RegisterEintrag	Implementierungsklasse für Entitäten vom Typ „`RegisterEintrag`“. Enthält Getter- und Setter-Methoden für alle Attribute.
RegisterEintragDaten	Nicht persistentes Schnittstellenobjekt, das die Daten des RegisterEintrags enthält. Wird als Deep-Copy durch Bean-Mapper erzeugt.

Tabelle 1: Klassenbeschreibung für die Datenhaltung.

Im Normalfall verwenden Komponenten die Daten anderer Komponenten über deren Schnittstelle. Der direkte Zugriff auf die DAOs ist verboten.

Entitäten, die von mehreren Anwendungskomponenten verwendet werden, werden von der „Komponente“ Basisdaten angeboten. Die Komponente Basisdaten ist keine vollständige Komponente, sie bietet keine Schnittstelle an. Ausnahmsweise verwenden Komponenten stattdessen direkt die zugehörigen DAOs der Basisdaten. Dieser Fall ist auch in Abbildung 1 dargestellt. Weitere Details können [IsyFactReferenzarchitektur] entnommen werden.

3.4. Realisierung

Zur Realisierung der Datenhaltung müssen folgende Aktivitäten durchgeführt werden.

3.4.1 Anlegen des Datenbankschemas

Das Datenbankschema muss angelegt werden. Dazu werden die benötigten DDL-Anweisungen in einer Datei abgelegt. Es bietet sich an, einen Ant-Task zur Installation des Schemas anzubieten (siehe [VorlageAnwendung]).

Das initiale Datenmodell kann über das Tool hbm2ddl[3] erzeugt werden. Dieses muss anschließend noch bearbeitet werden.

3.4.2 Einbinden der Bibliotheken

Die benötigten Bibliotheken müssen in die Maven-Konfiguration (`pom.xml`) aufgenommen werden. Sie können Tabelle 2 entnommen werden. Eine passende Beispielkonfiguration findet sich in [VorlageAnwendung][4].

[3] hbm2ddl ist Teil der Hibernate Tools und gehört zum Umfang der Standard-Hibernate Bibliothek. Nutzungsdokumentation unter: http://docs.jboss.org/hibernate/core/3.2/reference/en/html/toolsetguide.html#toolsetguide-s1-5

GroupId	ArtifactId
asm	asm
asm	asm-commons
cglib	cglib-nodep
com.oracle	ojdbc14
commons-dbcp	commons-dbcp
concurrent	concurrent
org.hibernate	hibernate-annotations
org.hibernate	hibernate-entitymanager
org.hibernate	hibernate
org.springframework	spring-orm
de.bund.bva.pliscommon	plis-common

Tabelle 2: Bibliotheken für Datenbankzugriff.

3.4.3 Implementierung der Entitätsklassen und DAOs

Die DAOs, Entitäts-Klassen und Schnittstellen-Klassen müssen implementiert werden. In den Entitätsklassen müssen die Mapping-Informationen für JPA als Annotations eingetragen werden.

Die Named Queries für die DAOs werden in der Konfigurationsdatei „`NamedQueries.hbm.xml`“ (siehe nächstes Kapitel) abgelegt. Die DAOs verwenden einen Schlüssel zur Identifikation einer Query. Für jeden Schlüssel wird eine Konstante in einer Konstantenklasse (z.B. `DatenbankKonstanten`) angelegt.

3.4.4 Anlegen der Konfigurationsdateien

Die in Tabelle 3 genannten Konfigurationsdateien müssen angelegt werden. Details dazu können [DatenzugriffDetailkonzept] entnommen werden. Entsprechende Beispieldateien finden sich in [VorlageAnwendung].

Pfad	Datei	Beschreibung
`src/main/resources/config`	`jpa.properties`	Konfiguration der Datenbank-Verbindung
`src/main/resources/resources/spring`	`jpa.xml`	Spring Konfiguration für JPA
`src/main/resources/resources/spring`	`NamedQueries.hbm.xml`	Named

[4] Siehe [VorlageAnwendung]/cd-register/pom.xml

		Queries für DAOs
src/main/resources/resources/spring	hibernate.cfg.xml	Hibernate-Konfiguration
src/main/resources/META-INF	persistence.xml	JPA-Konfiguration

Tabelle 3: Konfigurationsdateien für die Datenhaltung.

3.4.5 Implementierung von Schnittstellen-Klassen

Schnittstellen-Klassen dienen als eine nur Lese-Sicht auf persistente Entitäten. Dieses wird benötigt, wenn Komponenten persistente Entitäten über ihre Schnittstelle herausgeben, um zu verhindern, dass andere Komponenten diese Daten ändern.

Schnittstellen-Klassen enthalten alle Attribute, die auch ihre persistenten Gegenstücke besitzen. Zusätzlich besitzen sie Getter-/Settermethoden für alle Attribute.

Die Schnittstellen-Objekte werden per Deep-Copy mittels des Bean-Mappers Dozer erzeugt und dem Aufrufer außerhalb der Teilanwendung zurückgeliefert. So stehen dem Aufrufer alle Informationen zur Verfügung, es ist ihm aber nicht möglich, Änderungen zu persistieren. Damit ist die Datenhoheit der Komponente gewahrt.

Im Folgenden ist ein beispielhaftes Mapping zu sehen:

```
/** Dozer Bean-Mapper. */
protected Mapper mapper;

// Entität mappen
RegisterEintragDaten daten = mapper.map(registerEintrag,
RegisterEintragDaten.class);
```

4. Fachkomponenten der Anwendung

In diesem Kapitel wird die Realisierung von Fachkomponenten beschrieben.

4.1. Designvorgaben

- Alle Komponenten definieren ihre Schnittstelle über ein Java-Interface. Eine Ausnahme bildet die Komponente Basisdaten. Diese Komponente verwaltet gemeinsam genutzte Daten und bietet keine eigene Schnittstelle an. Der Zugriff erfolgt hier direkt über die DAOs (siehe Kapitel 3.3).

- Komponenten bieten an ihrer Schnittstelle eine Nur-Lese-Sicht auf ihre Daten an. Für jeden Entitätstyp wird eine nicht-persistente Schnittstellenklasse erstellt. Das Komponenten Interface wird von einer Java-Klasse implementiert. Diese Klasse kann die Anwendungsfälle im einfachen Fall direkt implementieren oder an Anwendungsfall-Klassen delegieren.

- Die interne Strukturierung von Komponenten ist nicht im Detail vorgeben. Für fachliche Komponenten wird eine Basisimplementierung in [IsyFactReferenzarchitektur] beschrieben.

4.2. Klassendesign

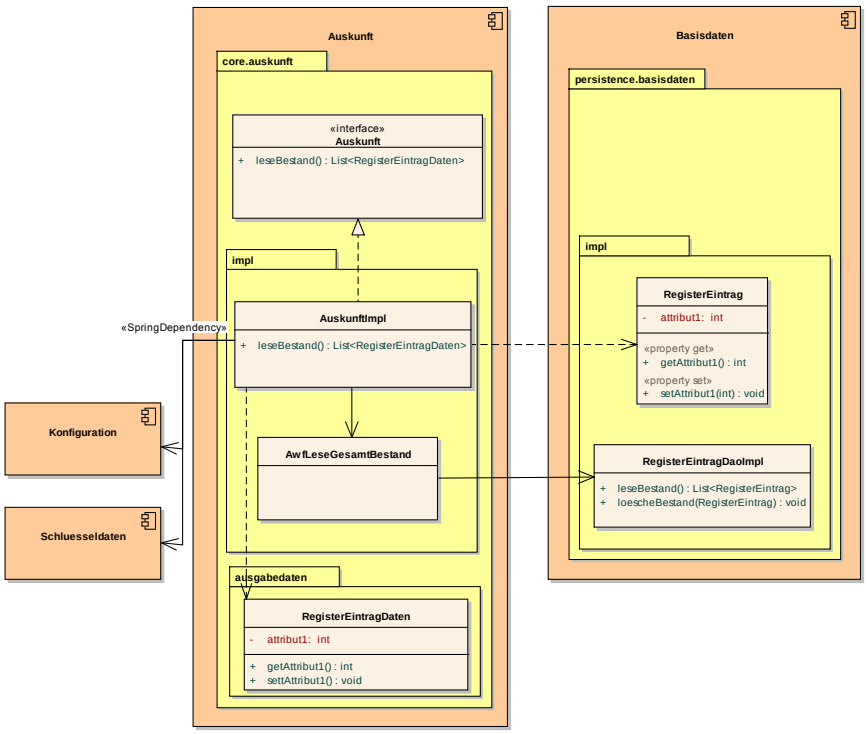

Abbildung 2: Klassendesign für Fachkomponenten.

Auskunft	Interfaces zur Definitionen der Schnittstelle der Komponente „Auskunft“. Zu beachten ist, dass über die Schnittstelle

	keine Entitäten der Komponente herausgegeben werden. Es darf immer nur eine Nur-Lese-Sicht (nicht-persistente Schnittstellen-Objekte) herausgegeben werden. Die Umwandlung der internen (`RegisterEintrag`) auf die externe Sicht erfolgt per Bean-Mapper Dozer.
AuskunftImpl	Implementierung der Komponente „Auskunft“. Diese Klasse wird als Spring-Bean konfiguriert. Weitere benötigte Komponenten (Spring-Beans) werden dieser Komponente per Spring-Dependency-Injection bekannt gemacht. Alle weiteren Klassen der Komponente, z.B. AWF-Klassen werden in der `AuskunftImpl` „normal“ instanziiert, und die benötigten Referenzen übergeben.
AwfLeseGesamtBestand	Beispielklasse zur Implementierung eines Anwendungsfalls. Diese Klassen werden explizit instanziiert, also nicht als Spring-Bean konfiguriert. Falls ein Anwendungsfall weitere Komponenten (Konfiguration, Regelwerk) etc. benötigt, werden diese durch die instanziierende Impl-Klasse übergeben.
RegisterEintrag	Persistente Entität für Register-Einträge.
RegisterEintragDaten	Nur-Lese-Sicht auf Register-Einträge (siehe Kapitel 3.4.5).

Tabelle 4: Klassenbeschreibung für Komponenten.

4.3. Package-Struktur

- Die Realisierung der Komponenten-Schnittstelle erfolgt im Package `<organisation>`[5]`.<domäne>.<system>.core.<komponente>`

- Die Realisierung der Komponenten-Implementierung erfolgt im Package `<organisation>.<domäne>.<system>.core.<komponente>.impl.*`

- Die nicht-persistenten Schnittstellen-Klassen werden im Package `<organisation>.<domäne>.<system>.core.<komponente>.ausgabedaten.*` implementiert.

4.4. Realisierung

- Die Implementierungsklassen und Interfaces der Komponente werden implementiert.

- Die Komponente wird als Spring-Bean in der Spring-Konfiguration „`src/main/resources/resources/spring/komponenten.xml`“ konfiguriert.

[5] z.B. `de.bund.bva`

- Je nach Bedarf wird die Komponente anderen Komponenten per Dependency-Injection bekannt gemacht.

5. Anwendungsnutzung

In diesem Kapitel wird die Realisierung von verschiedenen, technischen Zugangswegen zum Anwendungskern beschrieben, mit Ausnahme des GUI-Zugangs. Das Thema umfasst das Anbieten von internen Service-Schnittstellen per HttpInvoker, das Nutzen derselben und die Nutzung des Anwendungskerns im Rahmen der Batch-Verarbeitung.

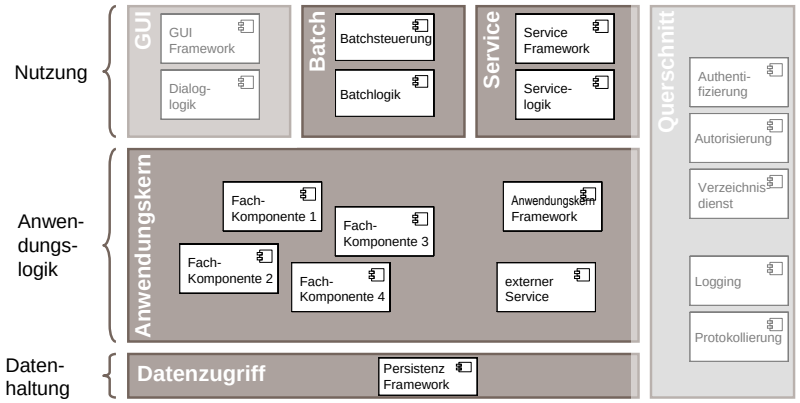

Abbildung 3: Referenzarchitektur eines IT-Systems

5.1. Anbieten von Service-Schnittstellen

Dieser Abschnitt beschreibt die Realisierung von HttpInvoker-Schnittstellen (siehe [Spring]). HttpInvoker-Schnittstellen sind interne Service-Schnittstellen, die innerhalb der Register Plattform durch andere Anwendungen genutzt werden dürfen. Extern verfügbare Services sind durch WebService-Schnittstellen anzubieten, über einen ServiceGateway.

5.1.1 Designvorgaben

- Interne Services werden per Spring-HttpInvoker angeboten.
- Es werden keine Komponenten des Anwendungskern extern verfügbar gemacht: Es wird stets eine eigene Service-Schicht implementiert. Dazu gehört auch die Definition einer Service-Schnittstelle als Java-Interface (RemoteBean).
- Jede Service-Methode erhält einen zusätzlichen Parameter „`AufrufKontext`“. Im Aufrufkontext werden Informationen zum Aufrufer (Name, Behördenkennzeichen, Rollen…) übermittelt. Die Implementierungen verschiedener Aufrufkontext-Transportobjekte sind in der Bibliothek „plis-serviceapi“ enthalten.
- Die Implementierung der Service-Schnittstelle wird in eine Exception-Fassade und die eigentliche Service-Implementierung aufgeteilt.
- In der Service-Schnittstelle werden nur Transport-Exceptions und Transportobjekte verwendet. Die Umwandlung der internen

Exceptions und Entitäten auf Transport-Exceptions und -Objekte erfolgt in der Service-Schicht.

- Listen von Objekten in Ein- und Ausgabeparametern werden als Arrays übertragen. Andere `Collection`-Typen sind nicht erlaubt.

- Beim Kompilieren der Schnittstellenprojekte muss auf die Java-Version geachtet werden. Die Java-Version darf nicht neuer sein, als diejenige des Nutzers. Ggf. muss die Schnittstelle auf eine ältere Version kompiliert werden.

5.1.2 Package-Struktur

- Schnittstellen werden versioniert. Die Versionsnummer wird dreistellig im Package-Namen der Serviceschnittstelle angegeben. Beispiel: Die Version 1.0.0 der Schnittstelle der Komponente Meldung der Anwendung Vorlage-Register, wird in den folgenden Packages implementiert:
  `de.bund.bva.cd.registercd.service.httpinvoker.v1_0_0.*`

- Interfaces, Transport-Exceptions und Transportobjekte werden im Package `<organisation>.<domäne>.<system>.service.httpinvoker.vX_Y_Z` implementiert.[6]

- Die Implementierung der Service-Schnittstelle erfolgt im Package `<organisation>.<domäne>.<system>.service.httpinvoker.vX_Y_Z.impl`.

[6] Das sind genau die Inhalte, die im eigenen Projekt `<system>-httpinvoker-sst` implementiert werden.

5.1.3 Klassendesign

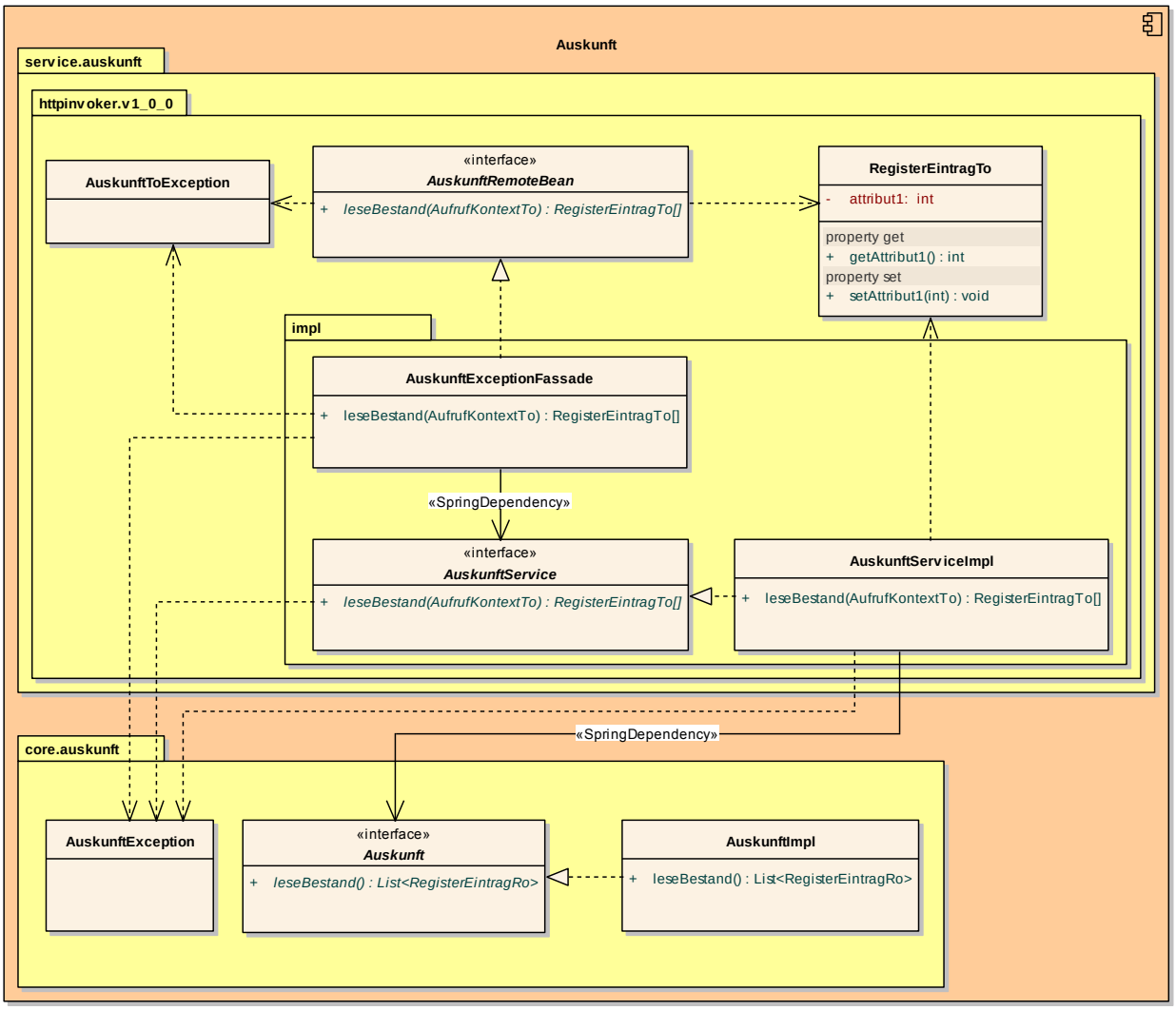

Abbildung 4: Klassendesign für HttpInvoker-Service-Schnittstellen.

AuskunftRemoteBean	Externes Interface für den Zugriff auf die Auskunft-Komponente per HttpInvoker. Bei Nutzung einer Service-Schnittstelle generiert Spring auf Basis dieses Interfaces einen Proxy für den Remote-Zugriff. Die Methoden dieser Komponente verwenden ausschließlich Transportobjekte und -Exceptions. Die Überwachung der Service-Aufrufe (siehe Kapitel 6.7) wird als Aspekt der RemoteBean konfiguriert.
AuskunftToException	Transport-Exception der Auskunft. Jede Komponente darf ausschließlich Transport-Exceptions an ihrer Service-Schnittstelle werfen. Details sind in [FehlerbehandlungKonzept] nachzulesen.
AufrufKontextTo	`AufrufKontext` der Service-Utilities mit den Informationen zum Aufrufer (Name,

	Passwort, Rollen…).
RegisterEintragTo	Transportobjekt für Register-Eintrag-Entitäten.
AuskunftExceptionFassade	Die Klasse `AuskunftExceptionFassade` implementiert das `AuskunftRemoteBean`-Interface. Die Exception-Fassade erhält damit alle Aufrufe der Auskunft-Service-Schnittstelle. Diese werden an die Auskunft-Service-Implementierung (`AuskunftServiceImpl`) delegiert. Die Aufgabe der Exception-Fassaden ist das Exception-Handling und -Mapping durchzuführen. Wichtig ist, einen Catch-Throwable-Block um den Aufruf der `AuskunftService`-Implementierung zu machen, um sicherzustellen, dass alle auftretenden Fehler gefangen werden. Die Implementierung der Fehlerbehandlung wird im Detail in [FehlerbehandlungKonzept] beschrieben. In der Exception-Fassade muss die Correlation-ID aus dem AufrufKontext in den Logging-Kontext gesetzt werden (siehe Kapitel 6.1).
AuskunftService	Internes Interface für den `Auskunft-Service`. Diese Schnittstelle verwendet Transportobjekte aber noch die internen Exceptions. Diese werden erst von der Exception-Fassade auf die eigentlichen Exceptions der `AuskunftRemoteBean` umgewandelt.
AuskunftServiceImpl	Implementierung des `AuskunftService`. In Service-Implementierung müssen die folgenden Aktivitäten durchgeführt werden: • Berechtigungsprüfung • Mappen der eingehenden Daten • Aufrufen des Anwendungskerns (Auskunft) • Mappen der ausgehenden Daten Das Mappen der Daten wird mit Dozer[7] durchgeführt. Dies geschieht automatisiert, ohne dass man Mapping-Informationen hinterlegen muss. Grund hierfür ist die strukturelle Gleichheit der Objekte des Anwendungskerns und der Service-Schicht. Dadurch ist Dozer in der Lage

[7] Dozer: Generischer Java-Bean zu Java-Bean Mapper (http://dozer.sourceforge.net)

	diese Objekte generisch zu übersetzen. In den Service-Implementierungen wird außerdem die Transaktionssteuerung durchgeführt. Diese wird per Spring-AOP über Annotations konfiguriert (siehe Kapitel 3).

Tabelle 5: Klassenbeschreibung für Service-Schnittstellen.

5.1.4 Realisierung

Zur Realisierung einer Service-Schnittstelle müssen einige Aktivitäten ausgeführt werden. Diese werden im Folgenden beschrieben.

5.1.4.1 Anlegen des Schnittstellen Projekts

Das neue Projekt `<system>-httpinvoker-sst` muss angelegt werden. Dazu wird eine neue `pom.xml` angelegt. Wichtig ist, dass darin die Compiler-Version so festgelegt wird, wie es im Dokument [ProduktKatalog] vorgegeben ist.

Das Projekt muss ein Jar erzeugen, das von anderen Systemen zur Nutzung der Service-Schnittstelle benötigt wird. In der Pom-Datei muss konfiguriert werden, dass das Jar in das Verzeichnis `repository-deploy` (Deployment-Repository) deployt wird. Ein Beispiel dafür findet sich in [VorlageAnwendung].

Das Schnittstellen-Projekt erhält dieselbe Group-ID wie das eigentliche Anwendungsprojekt, z.B. `de.bund.bva.cd.registercd`. Die Artifact-ID ist `<system>-httpinvoker-sst`.

5.1.4.2 Realisierung der „externen“ Service-Schnittstelle

Das RemoteBean-Interface, die Transportobjekte und -Exceptions müssen im Schnittstellen-Projekt angelegt werden.

5.1.4.3 Realisierung der Service-Implementierung

Im Projekt der eigentlichen Anwendung müssen die Exception-Fassade, das Service-Interface (z.B. `AuskunftService`) und die Implementierung dieses Interfaces angelegt werden.

Im Rahmen der Implementierung muss ggf. das Dozer-Mapping für die Transformation der Transport- auf die Entitätsobjekte angelegt werden. Dozer wird als Spring-Bean in der Datei „`src/main/resources/resources/spring/querschnitt.xml`“ konfiguriert. Dabei werden die zuvor angelegten Mapping-Dateien in Dozer konfiguriert.

In derselben Konfigurationsdatei werden die Exception-Fassade und die Service-Implementierung als Spring-Beans konfiguriert. Die Exception-Fassade erhält eine Referenz auf die Service-Implementierung per Dependency-Injection. Genauso erhält die Service-Implementierung eine Referenz auf Dozer per Dependency-Injection.

In der Datei „src/main/resources/resources/spring/remoting-servlet.xml“ wird die HttpInvoker-Konfiguration der Service-Schnittstelle durchgeführt. Dazu werden das Remote-Bean-Interface und die zugehörige Implementierung in Form der Exception-Fassade konfiguriert.

In der Datei „src/main/webapp/WEB-INF/web.xml“ muss das Dispatcher-Servlet (org.springframework.web.servlet.DispatcherServlet) eingebunden werden. Als Parameter wird die zuvor angelegte Konfiguration „remoting-servlet.xml“ übergeben.

Für jede Service-Schnittstelle wird ein Servlet-Mapping auf dieses Servlet konfiguriert.

Vor der Exception-Fassade wird mit Hilfe der Annotation @StelltLoggingKontextBereit die mit dem Aufrufkontext mitkommende Correlation-Id für das Logging registriert.

An den Methoden der Service-Implementierung werden die Annotationen @StelltAufrufKontextBereit und @Gesichert gemäß [SicherheitNutzerdok] verwendet, um den Zugriff auf die Service-Methode zu autorisieren.

Beispiele dafür finden sich in [VorlageAnwendung].

5.1.4.4 Konfigurieren der Service-Schnittstelle

Die angebotenen RemoteBeans (Service-Interfaces) werden in der Spring-Konfiguration „/src/main/resources/resources/spring/remoting-servlet.xml“ eingetragen. Der HttpInvoker-Service wird hier als Spring-Bean konfiguriert. Der Bean-Name ist für die URL, unter welcher der Service erreichbar sein wird, wichtig.

In die web.xml der Anwendung muss ein Servlet-Mapping für die URL des Services festgelegt werden. Alle Mappings zeigen dabei auf dasselbe HttpInvoker-Dispatcher-Servlet. Dieses delegiert den Aufruf dann an die zuvor konfigurierte Spring-Bean. Das Dispatching erfolgt an Hand der URL bzw. des Bean-Namens.

5.1.4.5 Einbinden der benötigten Bibliotheken

5.1.4.5.1 Bibliotheken für das Service-Schnittstellen-Projekt

Das Projekt der Service-Schnittstelle benötigt die in Tabelle 6 aufgelisteten Bibliotheken:

GroupId	ArtifactId
de.bund.bva.isyfact	isy-exception-sst
de.bund.bva.pliscommon	plis-serviceapi

Tabelle 6: Bibliotheken für das Anbieten von Service-Schnittstellen.

5.1.4.5.2 Bibliotheken für die Implementierung der Service-Schnittstelle
In die Build-Konfiguration des Hauptprojekts des Anwendungssystems müssen folgende Bibliotheken aufgenommen werden:

GroupId	ArtifactId
<organisation>[8].<domäne>.<system>	<systemname>-httpinvoker-sst[9]
de.bund.bva.pliscommon	plis-serviceapi
de.bund.bva.isyfact	isy-exception-core
org.springframework	spring-web
org.springframework	spring-webmvc
org.springframework	spring-aop
org.springframework	spring-aspects
net.sf.dozer	dozer[10]

Tabelle 7: Bibliotheken für das Anbieten von Service-Schnittstellen.

5.2. Nutzen von Service-Schnittstellen

Dieser Abschnitt beschreibt, wie Service-Schnittstellen genutzt, d.h. aufgerufen werden können.

5.2.1 Designvorgaben

Die genutzte Schnittstelle soll vom eigenen Anwendungskern entkoppelt werden. D.h. im eigenen Anwendungskern werden keine Exceptions oder Transportobjekte der genutzten Schnittstelle verwendet. Dazu wird ein Wrapper um die Schnittstelle implementiert.

[8] z.B. `de.bund.bva`
[9] Das fügt das Schnittstellen-Projekt als Bibliothek hinzu.
[10] Die verwendete Version von Dozer ist dem Produktkatalog zu entnehmen

5.2.2 Klassendesign

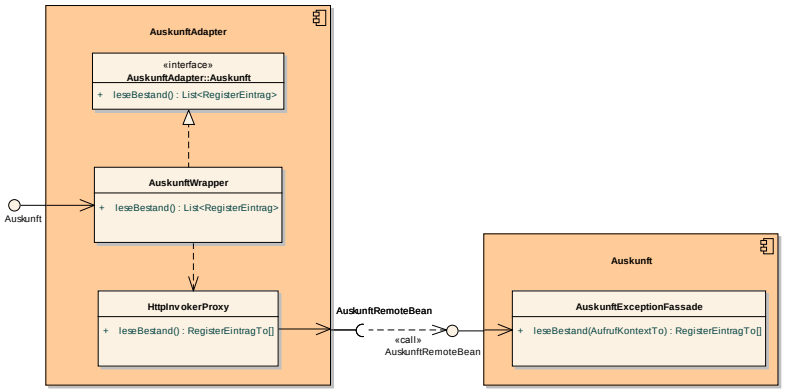

Abbildung 5: Beispiel für die Implementierung eines Client-Adapters.

Zum Zugriff auf den Service wird im Normalfall ein Adapter im Client implementiert (`AuskunftAdapter`). Dieser Adapter entkoppelt den Anwendungskern des Clients vom Service.

Im Adapter wird im Wesentlichen eine Wrapper-Klasse (`AuskunftWrapper`) implementiert. Diese führt das Mapping der Exceptions und der Daten durch. Der Adapter implementiert im Beispiel ein eigenes „Auskunft“-Interface für die Nutzung durch die Client-Anwendung.

Für die Remote Zugriffe wird der `HttpInvokerProxy` benutzt. Dieser wird automatisch von Spring erzeugt. Er bietet das RemoteBean-Interface auf Client-Seite an und sorgt dafür, dass Aufrufe per HttpInvoker an den Service weitergereicht werden.

5.2.3 Realisierung

5.2.3.1 Einbinden der Schnittstellen-Bibliothek

Zur Realisierung wird die Bibliothek mit der zu nutzenden Service-Schnittstelle benötigt. Dieses befindet sich im Deployment-Repository der zu nutzenden Anwendung. Das Jar wird in das eigene Projekt-Repository kopiert und via Maven eingebunden.

Zusätzlich müssen die HttpInvoker-Bibliotheken von Spring eingebunden werden. Eine vollständige Liste zeigt Tabelle 8.

GroupId	**ArtifactId**
<organisation>.<domäne>.<system>	<systemname>-httpinvoker-sst
de.bund.bva.pliscommon	plis-serviceapi
org.springframework	spring-web
org.springframework	spring-aop
org.springframework	spring-aspects

Tabelle 8: Bibliotheken für Service-Nutzung.

5.2.3.2 Durchführen der Konfiguration

Spring erzeugt anhand des Service-Interfaces HttpInvoker-Proxies, die den eigentlichen HttpInvoker-Aufruf durchführen. Diese Proxies werden als Spring-Bean konfiguriert:

```
<bean id="xxxRemoteBean"
class="org.springframework.remoting.httpinvoker.HttpInvokerPr
oxyFactoryBean">
  <property name="serviceUrl" value="${service.xxx.url}"/>
  <property name="serviceInterface"
value="de.bund.bva….xxxRemoteBean"/>
</bean>
```

Auf der Bean können alle Methoden des Interfaces „`serviceInterface`“ aufgerufen werden, der Aufruf erfolgt dann automatisch per HttpInvoker gegen das in „`serviceUrl`“ konfigurierte Ziel-System. [11]

5.2.3.3 Erweiterung um die Aufrufwiederholung mittels Service Utilities

Die in Kapitel 5.2.3.2 aufgeführte Konfiguration eines aufzurufenden Dienstes kann durch die Verwendung einer Aufruf-Wiederholungsimplementierung erweitert werden, so dass Aufrufe bei Timeouts wiederholt werden. Dies ist nur notwendig, sofern eine Aufrufwiederholung eine Anforderung an die Anwendung ist. Für die Aufruf-Wiederholung ist lediglich die Spring-Konfiguration des Proxies anzupassen:

```
<bean id=" xxxRemoteBean "
class="org.springframework.remoting.httpinvoker.HttpInvokerPr
oxyFactoryBean">
 <property name="serviceInterface"
value="de.bund.bva….xxxRemoteBean " />
 <property name="serviceUrl" value="${service.xxx.url}" />
 <property name="httpInvokerRequestExecutor"
ref="xxxRequestExecutor"/>
</bean>

<bean id="xxxRequestExecutor"
class="de.bund.bva.pliscommon.serviceapi.core.httpinvoker.Tim
eoutWiederholungHttpInvokerRequestExecutor">
    <property name="anzahlWiederholungen"
value="${xxx.service.wiederholungen}" />
    <property name="timeout" value="$xxx.service.timeout}" />
</bean>
```

Auf der Bean (`xxxRequestExecutor`) sind die „`anzahlWiederholungen`“ und der „`timeout`“ konfiguriert. Dieser `RequestExecutor` erweitert den Standard-`RequestExecutor` von Spring um die Möglichkeit Timeouts zu definieren und eine konfigurierte Anzahl an Aufruf-Wiederholungen durchzuführen. Dieser `RequestExecutor` ist der Spring-HttpInvokerProxyFactoryBean bekannt zu machen. Alle Unterschiede zur Konfiguration ohne die Verwendung der

[11] Die URL wird als betriebliche Konfiguration in eine Property-Datei ausgelagert und durch den PropertyPlacerHolder von Spring ersetzt.

Service-Api für Aufruf-Wiederholungen und Timeouts sind fett markiert im Code-Beispiel.

5.2.3.4 Implementierung des Wrappers

Zur Entkopplung des eigenen Anwendungskerns von der Schnittstelle wird ein Wrapper für die Schnittstelle implementiert. Der Wrapper führt das Mapping der internen Datenobjekte auf die Transportobjekte durch. Dieses kann bei Bedarf mit Dozer gemacht werden.

Zusätzlich führt der Wrapper das Exception-Handling durch. Der Wrapper kann auftretende Exceptions in eigene Exceptions umwandeln (Exception-Chaining) oder explizit behandeln.

5.3. Batch-Verarbeitung

In diesem Kapitel wird die Implementierung von Batches zu einer Anwendung beschrieben.

5.3.1 Designvorgaben

- Die Batch-Verarbeitung verwendet den Anwendungskern der zugehörigen Anwendung. Der Anwendungskern ist Teil des Batch-Deployments, d.h. der Code ist sowohl Teil der Server-Anwendung als auch der Batch-Anwendung in Bezug auf Deploymenteinheiten.

- Zur Realisierung der Batchlogik wird eine Batch-Ausführungs-Bean implementiert.

- Falls für die Verarbeitung im Batch eigene Fachlogik benötigt wird, ist diese trotzdem den Register-Komponenten hinzuzufügen.

- Im Rahmen der Initialisierung hat die Ausführungs-Bean unter anderem die Aufgabe, die Konsistenz und Korrektheit der Eingabedaten zu prüfen.

- Falls die zu verarbeitenden Sätze eines Batches das Ergebnis einer Datenbank-Query sind, ist in der Initialisierung die Query über eine Fachkomponente abzusetzen. Diese Query soll die (fachlichen) Schlüssel von Entitäten, nicht Entitäten selbst auslesen.

- Die Batches sind möglichst robust zu konstruieren: Falls auf ein fachliches Problem in der Ausführungs-Bean reagiert werden kann, sollte dies getan werden.

- Batches erzeugen ein Ausführungsprotokoll. Der Batchrahmen, die Steuerungsimplementierung, die jeden Batch und dessen Arbeitsschritte steuert, stellt die notwendige Implementierung bereit. Die Ausführungs-Bean übermittelt dem Batchrahmen Status-Informationen für das Protokoll.

- Batches verwenden einen (konfigurierten) technischen Benutzer, um sich vor Start der fachlichen Verarbeitung am Access-Manager der Plattform zu authentifizieren.

- Alle Batches zu einer Anwendung werden als eigenständige Deployment-Einheit ausgeliefert.

5.3.2 Klassendesign

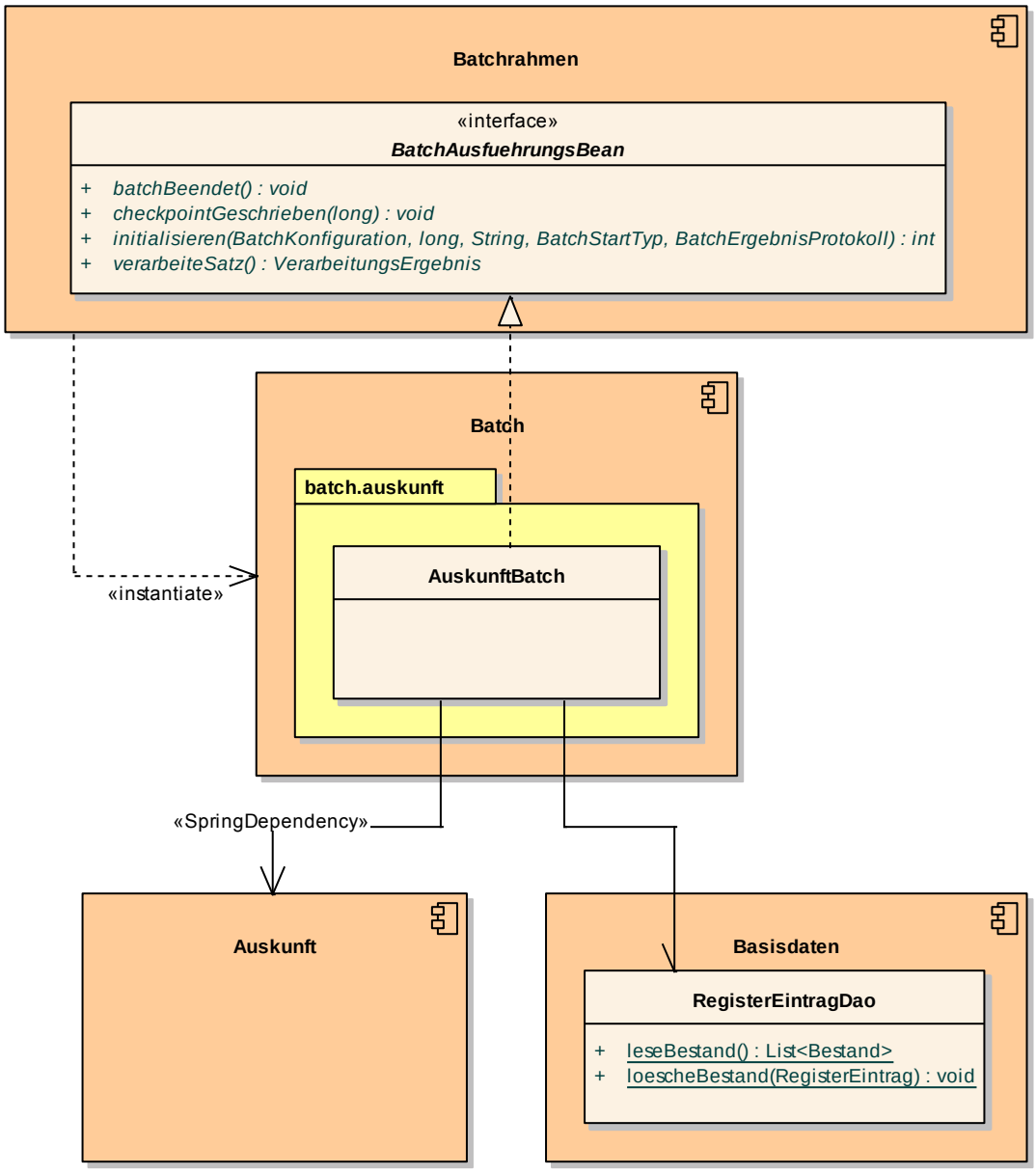

Abbildung 6: Klassendesign eines Batches.

Abbildung 6 zeigt eine beispielhafte Implementierung eines Batches, der die Komponenten „Auskunft“ und „Basisdaten“ verwendet. Im Normalfall erhält die Batch-Bean (`AuskunftBatch`) eine Referenz auf die Komponenten des Anwendungskerns per Spring-Dependency. Für die

Komponente Basisdaten erfolgt der Zugriff wie immer mittels statischer Aufrufe der DAOs.

Der Batchrahmen definiert das Interface „`BatchAusfuehrungsBean`“. Dieses dient der Steuerung des Batches durch den Batchrahmen. Es muss von der Batch-Ausführungs-Bean implementiert werden. Der Batchrahmen sorgt auch für die Initialisierung und Ausführung des Batches.

Der Batchrahmen übernimmt die Transaktionssteuerung. Die Transaktionssteuerung im Batch sieht vor, mehrere Arbeitsschritte in einer Transaktion abzuwickeln. Die Größe der Transaktion (Commit-Rate) wird über den Batchrahmen konfiguriert.

5.3.3 Realisierung

5.3.3.1 Einbinden der Bibliothek

Zur Realisierung von Batches muss die in Tabelle 9 aufgelistete Bibliothek eingebunden werden.

GroupId	ArtifactId
de.bund.bva.pliscommon	plis-batchrahmen

Tabelle 9: Bibliotheken für die Realisierung von Batches.

5.3.3.2 Implementierung der Batch-Logik

Die Batch-Logik wird implementiert, in dem eine Batch-Bean im Package `de.bund.bva.<domäne>.<anwendung>.batch` implementiert wird. Für die Realisierung ist es notwendig, dass die Batch-Bean das Interface `de.bund.bva.pliscommon.batchrahmen.batch.rahmen.BatchAusfuehrungsBean` aus der Bibliothek plis-batchrahmen implementiert.

Der Batchrahmen ruft als erstes die Methode „initialisieren“ auf. Dabei werden alle zur Initialisierung benötigten Informationen übergeben. Details dazu werden im JavaDoc der Methode beschrieben.

Der Parameter „`BatchErgebnisProtokoll`“ enthält eine Referenz auf ein Protokollobjekt, welches der Batch verwendet, um Protokoll-Meldungen und Statistiken an den Batchrahmen zu übergeben.

5.3.3.3 Konfiguration des Batches und Batchrahmens

Für jeden Batch muss eine Property-Datei in „`/src/main/resources/resources/batch`“ angelegt werden. In dieser statischen Konfiguration werden unter anderem die Batch-ID und die Transaktionssteuerung konfiguriert. Eine Beschreibung der Parameter ist in [BatchDetailkonzept] enthalten.

Die betriebliche Konfiguration des Batches ist identisch zu derjenigen der zugehörigen Anwendung. Auch Parameter, die nur für den Batch benötigt werden, werden in die betriebliche Konfiguration der Fachanwendung aufgenommen.

5.3.3.4 Spring-Konfiguration anlegen

Für den Batchrahmen werden die in Tabelle 10 aufgelisteten Konfigurationsdateien benötigt.

Pfad	Datei	Beschreibung
`src/main/resources/resources/batch/rahmen`	`Batchrahmen.xml`	Hier werden die Spring-Beans des Batchrahmens definiert. Zusätzlich muss für jeden existierenden Batch die Ausführungs-Bean als Spring-Bean definiert werden.

Tabelle 10: Springkonfiguration für den Batchrahmen.

Der Batch verwendet den Anwendungskern der eigentlichen Anwendung. Dafür wird eine Spring-Konfiguration benötigt. Als Basis können die Spring-Konfigurationen der Anwendung verwendet werden. Diese werden anschließend wie in Tabelle 11 beschrieben angepasst:

Pfad	Datei	Beschreibung
`src/main/resources/resources/batch/register`	`jpa.xml`	In der Bean `EntityManagerFactory` muss der Parameter „`PersistenceUnitName`“ so umkonfiguriert werden, dass die `PersitenceUnit` für den Batch (siehe `persistence.xml`) verwendet wird.
	`komponenten.xml`	Bei der Bean „`AufrufKontext`“ muss der Scope „`request`“ entfernt werden. Alternativ dazu kann über die Klasse `SimpleRequestContextAttributes` aus isy-util auch der Request-Kontext für den Batch nachgebildet werden.
	`querschnitt.xml`	Die Beans für die JMX-Überwachungs-schnittstelle werden entfernt. Die Timer für Watchdog und Systemprüfung werden entfernt.
	`hibernate.cfg.xml`	Die Klassen BatchStatus und BatchKon-figurationsparameter werden über `<mapping resource="resources/plis-batchrahmen/hibernate/hibernate-mapping.xml"/>` eingebunden.
`src/main/resources/META-INF`	`persistence.xml`	Hier muss die spezielle Hibernate-konfiguration (`hibernate.cfg.xml`) als neue Persistence-Unit eingetragen werden.

Tabelle 11: Springkonfiguration für den Anwendungskern für Batches.

Bei geeigneter Aufteilung der Spring-Konfiguration kann auch die Konfiguration der Anwendung direkt verwendet werden. Dazu müssen

Timertasks und JMX-Beans in eigene Spring-Konfigurationen ausgliedert werden.

5.3.3.5 Konfiguration des Batch-Deployments

Für das Deployment des Batches wird ein neues Maven-Projekt `<system>-batch` angelegt. Dieses hat die Aufgabe das Deployment-Paket für den Batch zusammenzustellen.

Dazu wird eine neue `pom.xml` angelegt, die als Ziel-Typ ein Jar mit allen Dateien des Batches erzeugt. Zusätzlich können in diesem Projekt Shell-Skripte und ähnliches für den Batch abgelegt werden. Ein Beispiel ist in [VorlageAnwendung] enthalten.

Das Batch-Projekt enthält keinen Java-Code. Die Batch-Beans liegen im normalen Anwendungsprojekt.

6. Querschnitt

In diesem Kapitel wird die Umsetzung querschnittlicher Aspekte beschrieben.

6.1. Logging

In diesem Abschnitt wird beschrieben, wie das Logging umzusetzen und zu konfigurieren ist.

6.1.1 Designvorgaben

- Für Logging wird Log4j verwendet.
- Es wird ein Debug-, Info- und ein Error-Log geführt. Die Zuordnung der Log-Levels auf diese Log-Arten wird in [LoggingKonzept] definiert. Ebenso welche Informationen mit welchem Log-Level ausgeben werden sollen.
- Für das Logging wird das im Rahmen der IsyFact erstellte Single-Line-Layout verwendet.
- In jeder Log-Meldung ist eine Correlation-ID mitzuloggen. Diese identifiziert den Aufruf über die Anwendungslandschaft hinweg.
- Log-Konfigurationen müssen zur Laufzeit änderbar sein.

6.1.2 Realisierung

6.1.2.1 Implementierung von Log-Ausgaben

Log-Ausgaben können an beliebigen Stellen im Code erzeugt werden. Dazu wird in jeder Klasse ein eigener Logger erzeugt:

```
public static final Logger LOG =
Logger.getLogger(XXX.class);
```

Wichtig ist, in der Exception-Fassade an der Service-Schnittstelle (siehe Kapitel 5.1.3) die Correlation-ID zu setzen:

```
@StelltLoggingKontextBereit
public int cdErworben(AufrufKontext kontext, …)
…
```

6.1.2.2 Einbinden der Bibliotheken

Um die Logging Funktionen in der eigenen Anwendung nutzen zu können müssen die in Tabelle 12 aufgelisteten Bibliotheken eingebunden werden.

GroupId	ArtifactId
de.bund.bva.isyfact	isy-logging

Tabelle 12: Bibliotheken für das Logging.

Dadurch wird Log4j automatisch in die Anwendung integriert.

6.1.2.3 Anlegen der Konfiguration

In „`/src/main/resources/config/`“ muss die Datei log4j.properties angelegt werden. Diese definiert, wohin Log-Ausgaben geschrieben werden und wie das Layout dafür ist. Für das Layout wird die Klasse `de.bund.bva.pliscommon.logging.common.layout.SingleLinePatternLayout` aus „isy-logging“ verwendet.

In der web.xml muss der Pfad für die Log-Konfiguration und das Polling darauf konfiguriert werden. Dazu müssen die Parameter `log4jConfigLocation` und `log4jRefreshInterval` als Context-Parameter definiert werden.

Ebenfalls in der `web.xml` muss die Klasse `org.springframework.web.util.Log4jConfigListener` als Servlet-Listener eingebunden werden.

6.2. Konfiguration

In diesem Kapitel wird die Verarbeitung von Konfigurationen in Anwendungen beschrieben.

6.2.1 Designvorgaben

- Für die Konfiguration werden betriebliche, statische und Benutzerkonfigurationen unterschieden. Eine Definition und Kriterien zur Typisierung können in [ÜberwachungKonfigKonzept] nachgelesen werden.
- Betriebliche Konfigurationen werden als Properties-Datei in „`/src/main/resources/config`“ abgelegt.
- Statische Konfigurationen werden als Datei in „`/src/main/resources/resources`“ abgelegt.
- Benutzerkonfigurationen werden in der Datenbank abgelegt.
- Betriebliche Konfigurationen können in Ausnahmefällen zur Laufzeit aktualisiert werden.
- Für das Laden von betrieblichen Konfigurationen wird die Bibliothek „isy-konfiguration“ verwendet.

6.2.2 Realisierung

6.2.2.1.1 Einbinden der Bibliotheken

Die in Tabelle 13 aufgeführte Konfigurationsbibliothek muss eingebunden werden.

GroupId	ArtifactId
de.bund.bva.isyfact	isy-konfiguration

Tabelle 13: Bibliotheken für die Konfiguration.

Die Konfigurationsbibliothek ermöglicht den typsicheren Zugriff auf Konfigurationsparameter in Property-Dateien. Außerdem implementiert sie einen Polling-Mechanismus, der dazu genutzt werden kann Konfigurationsänderungen zur Laufzeit bekannt zu machen.

6.2.2.2 Auslesen von Konfigurationsparametern in der Anwendung

Zum Zugriff auf die in den betrieblichen Konfigurationsdateien abgelegten Parametern aus der Anwendung heraus wird die Klasse „`de.bund.bva.pliscommon.konfiguration.common.impl.ReloadablePropertyKonfiguration`“ aus isy-konfiguration verwendet. Diese wird als Spring-Bean konfiguriert und erhält im Konstruktor eine Liste aller betrieblichen Property-Dateien. Komponenten, welche Zugriffe auf Parameter benötigen, erhalten eine Referenz auf diese Bean und können über die angebotenen Getter-Methoden die Konfigurationsparameter auslesen.

Für die Namen der Konfigurationsparameter wird eine abstrakte Klasse „`KonfigurationSchluessel`“ angelegt, welche alle Parameternamen als String-Konstanten enthält.

Im Ausnahmefall können Parameter zur Laufzeit geändert werden. Solche Parameter werden vorzugsweise nicht in Instanzvariablen gehalten, sondern bei jeder Benutzung ausgelesen. Alternativ kann der Eventlistener-Mechanismus der Konfigurationsbibliothek verwendet werden.

Damit die Konfiguration periodisch auf Änderungen überwacht wird, muss ein Timer erzeugt werden. Dieses erfolgt ebenfalls über eine Spring-Timer-Task. Details dazu werden in [ÜberwachungKonfigKonzept] beschrieben. Ein entsprechendes Beispiel ist in [VorlageAnwendung] umgesetzt.

6.2.2.3 Konfigurationsparameter in Spring-Konfigurationen

Betriebliche Konfigurationsparameter, z.B. für die Datenbankverbindung, dürfen nicht in der Spring-Konfiguration abgelegt werden. Diese werden über den Property-Replace-Mechanismus von Spring in betriebliche Property-Dateien ausgelagert. Dazu wird die Bean „`org.springframework.beans.factory.config.PropertyPlaceholderConfigurer`“ in die Spring-Konfiguration aufgenommen. Dieser wird die Liste der betrieblichen Konfigurationsdateien gegeben. Betriebliche Parameter können so als Variablen in der Spring-Konfiguration angegeben werden. Spring sorgt für eine Ersetzung der Parameter beim Anwendungsstart. Details dazu können in [Spring] nachgelesen werden.

6.3. Fehlerbehandlung

In diesem Kapitel wird beschrieben, wie die Fehlerbehandlung durchzuführen ist.

6.3.1 Designvorgaben

- In jeder Anwendung bzw. Bibliothek wird eine eigene Exception-Hierarchie angelegt.
- Für Anwendungs-Exceptions wird die oberste Exception dieser Hierarchie von den in der Bibliothek „plis-exception“ enthaltenen Exception-Klassen abgeleitet. Diese Ober-Exceptions sind als abstrakt zu kennzeichnen.
- Für Exceptions in selbst entwickelten Bibliotheken werden nicht die Exception-Klassen aus „plis-exception“ verwendet. Die zugrundeliegenden Designprinzipien sind jedoch identisch umzusetzen. So wird für jede Bibliothek eine abstrakte Ober-Exception angelegt. Diese sorgt für das Laden der Nachrichten, erbt aber direkt von einer der `java.lang.Exception` bzw `java.lang.RuntimeException`.
- Fehlertexte werden in Resource-Bundles ausgelagert und über eine Fehler-ID indentifiziert. Die Schlüssel der Fehler-IDs werden in einer Konstantenklasse zusammengefasst.
- Exceptions werden grundsätzlich nur zur Signalisierung abnormer Ergebnisse bzw. Situationen eingesetzt.
- Exceptions sind in der Regel zu behandeln und zu loggen. Ist es nicht möglich die Exception zu behandeln, muss sie an den Aufrufer weitergegeben werden. Die Exception wird im Fall eines Weiterwerfens nicht geloggt.
- Nur Exceptions in Methodensignaturen verwenden, die auch vorkommen können.
- Bei der Behandlung von Fehlern ist ein geordneter Systemzustand herzustellen, z. B. das Schließen der Datenbankverbindung über einen `finally`-Block.
- Fehler sollten möglichst früh erkannt werden und zu entsprechenden Ausnahmen führen.
- Interne Exceptions dürfen in der Service-Schnittstelle nicht vorkommen.
- Catch-Blöcke dienen der Fehlerbehandlung und dürfen nicht als `else`-Zweige genutzt werden.
- Keine leeren Catch-Blöcke.

- Das destruktive Wrappen einer Exception zerstört den StackTrace und ist nur für Exceptions an den Außen-Schnittstellen sinnvoll. Destruktiv gewrappte Exceptions sind in jedem Fall vor dem Wrappen zu loggen.

Weitere Hinweise für die richtige Behandlung von Fehlern sind in [FehlerbehandlungKonzept] enthalten.

6.3.2 Paketstruktur

Exceptions die querschnittlich, also von mehreren Komponenten genutzt werden, werden im Paket:

```
<organisation>[12].<domäne>.<anwendung>.common.exception
```

implementiert. Komponentenspezifische Exceptions, also solche die nur von einer einzigen Komponente genutzt werden, gehören in das Paket:

```
<organisation>.<domäne>.<anwendung>.core.<komponente>
```

6.3.3 Realisierung

Die Exception-Bibliothek „PLIS-Exception“ ist in zwei Teile aufgeteilt:

- isy-exception-core und
- isy-exception-sst

Das Core-Paket enthält anwendungsinterne Exception-Klassen und Hilfsklassen für das Exception-Mapping. Im sst-Projekt sind die Klassen für die Transport-Exceptions enthalten. Wenn das Core-Paket eingebunden wird, wird über Maven automatisch das Schnittstellen-Projekt mit eingebunden. Die explizite Einbindung von „isy-exception-sst“ sollte dann entfernt werden.

Die Core-Bibliothek wird im Wesentlichen im Anwendungskern bzw. der Service-Schnittstellen-Implementierung benötigt (siehe Kapitel 5.1.4.5.2). Für Service-Schnittstellen werden lediglich die Transport-Exceptions aus „isy-exception-sst“ benötigt (siehe Kapitel 5.1.4.5.1).

6.3.3.1 Einbinden der Bibliothek

Zur Realisierung der Fehlerbehandlung und Implementierung von Exceptions müssen die in Tabelle 14 aufgelisteten Bibliotheken eingebunden werden.

GroupId	ArtifactId
de.bund.bva.isyfact	isy-exception-core

[12] z.B. `bva.bund.de`

GroupId	ArtifactId
de.bund-bva.isyfact	isy-util

Tabelle 14: Bibliotheken für die Fehlerbehandlung.

„isy-exception-core“ enthält abstrakte Exception-Klassen die in Anwendungen zu verwenden sind. „isy-util“ enthält Hilfsklassen zum Laden von Fehlertexten.

6.3.3.2 Anlegen der Exception-Klassen

In jeder Anwendung wird für jede Exception-Art (technisch, fachlich) eine eigene Oberklasse angelegt. Diese erbt von der entsprechenden Klasse aus „plis-exception“. Zum Laden der Fehlertexte wird das Interface „FehlertextProvider“ aus derselben Bibliothek verwendet. In „isy-util“ ist die Implementierung „`MessageSourceFehlertextProvider`“ enthalten. Diese unterstützt das Laden von Fehlertexten aus einer Spring-Message-Source. Ein Beispiel für die Verwendung ist in [VorlageAnwendung] enthalten.

6.3.3.3 Fehlerbehandlung an der Anwendungsschnittstelle

Fehler sind entweder zu behandeln und zu loggen oder weiterzuwerfen. Es muss jedoch sichergestellt werden, dass interne Fehler der Anwendung nicht über die System-Schnittstelle (siehe Kapitel 5.1) geworfen werden. Dazu wird in der Exception-Fassade eine explizite Fehlerbehandlung mit einem Catch-Throwable-Block durchgeführt.

Alle Exceptions der Anwendungen werden hier in Transport-Exceptions umgewandelt. Dazu wird das im Folgenden beschrieben Muster verwendet.

Es wird ein Catch-Block für alle auftretenden eigenen Exceptions angelegt. In jedem Catch-Block wird die Exception geloggt und über `PlisExceptionMapper.mapException` in eine passende Transport-Exception umgewandelt. Als letztes wird ein Catch-Throwable-Block eingefügt.

Hier wird für die aufgetretene Exception über `PlisExceptionMapper.createToException` eine neue Transport-Exception erzeugt. Zur Ermittlung der Fehler-ID wird eine Klasse AusnahmeIdUtil angelegt. Diese implementiert eine statische Methode `getAusnahmeId`, die zu einer übergebenen Exception eine passende Fehler-ID ermittelt. Vor dem Werfen der so erzeugten Exception über die Schnittstelle wird ein Log-Eintrag erzeugt.

Beim Umwandeln der internen Exceptions in Transport-Exceptions wird der Stack-Trace der internen Exceptions verworfen.

Ein Beispiel hierfür ist wieder in [VorlageAnwendung] enthalten.

6.4. Protokollierung

In diesem Kapitel wird beschrieben, wie eine fachliche Protokollierung umzusetzen ist.

6.4.1 Designvorgaben

- Protokolleinträge und Daten des Anwendungsfalls (Meldung, Auskunft…) werden innerhalb derselben Transaktion geschrieben.
- Protokoll-Tabellen werden im Datenbank-Schema der Anwendung abgelegt.
- Binärdaten werden nicht protokolliert. Es wird lediglich eine Referenz auf den entsprechenden Datensatz gespeichert.
- Referenzierte Binärdaten, dürfen erst dann physikalisch gelöscht werden, wenn auch der Protokolldatensatz entfernt wird. Bis dahin werden die Binärdaten mit einem Lösch-Flag versehen und stehen nur noch für die Protokollrecherche zur Verfügung, nicht mehr in der eigentlichen Anwendung. Dies kann über ein eigenes Feld in den Protokoll-Daten und einem Foreign-Key-Constraint in der Datenbank sichergestellt werden.

6.4.2 Realisierung

6.4.2.1 Einbinden der Bibliothek

Zur Realisierung der Protokollierung muss die in Tabelle 15 aufgelisteten Bibliothek eingebunden werden.

GroupId	ArtifactId
de.bund.bva.pliscommon	plis-protokollierung

Tabelle 15: Bibliotheken für die Protokollierung.

Die Bibliothek enthält Basis-Entitäten, von denen eigene Protokollentitäten erben müssen.

6.4.2.2 Implementierung der Protokollierungskomponente

Die Protokollierungskomponente wird analog zu den übrigen fachlichen Komponenten entwickelt. Es wird keine Basisimplementierung dafür vorgeben.

Es müssen eigene Entitätsklassen für die Protokolleinträge angelegt werden. Diese müssen von der abstrakten Entitätsklasse aus „plis-protokollierung“ erben.

Ein Klassendiagram ist in [ProtokollierungKonzept] enthalten.

Die Protokollierungskomponente benötigt in der Regel Kontextinformationen über den Aufrufer. Dazu wird in der Anwendung eine AufrufKontext-Komponente (siehe Kapitel 6.5) eingebunden.

6.5. Aufrufkontextverwaltung

Einige Komponenten der Anwendung, z.B. die Protokollierung oder die Autorisierung benötigen Kontextinformationen über den Aufrufer. Damit

diese nicht durch die gesamte Anwendung gereicht werden müssen, kann in der Anwendung ein AufrufKontextVerwalter verwendet werden.

6.5.1 Designvorgaben

- Die Komponente wird so implementiert, dass sie spezifische Informationen über den Aufruf-Kontext speichern kann (z.B. Name des aufrufenden Benutzers).
- Die Komponente kann in einer Anwendung so erweitert werden, dass sie beliebige weitere Kontext-Informationen aufnehmen kann.

6.5.2 Realisierung

6.5.2.1 Einbinden der Bibliothek

Zur Realisierung des AufrufKontext müssen die in Tabelle 17 aufgelisteten Bibliotheken eingebunden werden.

GroupId	ArtifactId
de.bund.bva.isyfact	isy-aufrufkontext

Tabelle 16: Bibliotheken für die Verwaltung des AufrufKontext.

Die Bibliothek isy-aufrufkontext enthält die Komponente `AufrufKontextVerwalter`, welcher den benutzerspezifischen Aufrufkontext im Thread-Scope (alternativ Request-Scope) hält.

6.5.2.2 Konfiguration der Bibliothek

Die Komponente `AufrufKontextVerwalter` wird als Spring-Bean konfiguriert. Dabei wird festgelegt, dass Spring eine neue Instanz für jeden Thread (alternativ Request) anlegen soll:

```
<bean id="aufrufKontextVerwalter" scope="request"
class="de.bund.bva.pliscommon.aufrufkontext.impl.AufrufKontextVerwalterImpl">
    <aop:scoped-proxy />
</bean>
```

Der vom AufrufKontextVerwalter verwaltete AufrufKontext wird beim Aufruf der Anwendung in der Service-Schnittstelle oder im DialogController der GUI gesetzt und steht fortan, während der Verarbeitung des Requests, in der gesamten Anwendung zur Verfügung.

Komponenten, die diese Informationen benötigen, erhalten dazu einfach eine Referenz auf den AufrufKontextVerwalter per Dependency Injection.

Zur Entgegennahme des AufrufKontextes an der Service-Schnittstelle kann eine Annotation `@StelltAufrufKontextBereit` verwendet werden. Die Konfiguration ist in [SicherheitNutzerdok] beschrieben.

6.6. Authentifizierung und Autorisierung

Dieses Kapitel beschreibt die Realisierung der Authentifizierung und Autorisierung von Anfragen.

6.6.1 Designvorgaben

- Die Authentifizierung von Anfragen wird im Servicegateway und im Portal mit Hilfe des Access Managers durchgeführt.
- Prozesse, die innerhalb der Register Plattform starten (z.B. Timertasks, Batches) verwenden einen technischen Benutzer und authentifizieren diesen selbständig gegen den Access Manager.
- Die Berechtigungsprüfung ist in der Anwendung deklarativ zu definieren bzw. zu programmieren.
- Eine erste Berechtigungsprüfung erfolgt in der Service-Schnittstelle oder im Web-GUI-Dialogcontroller jeder Anwendung. Es wird geprüft, ob der Aufrufer den Service oder den Dialog überhaupt verwenden darf.
- In jeder Service-Methode wird ein Parameter `AufrufKontext` mit den Daten des aufrufenden Benutzers übermittelt. Dieser Parameter wird im `AufrufKontextVerwalter` hinterlegt und beim Aufruf weiterer Nachbarsysteme durchgereicht.
- In der Web-GUI wird ein vom Access-Manager bereitgestellter http-Header mit den Daten des aufrufenden Benutzers entgegengenommen und in einen `AufrufKontext` gewandelt. Dieser Parameter wird im `AufrufKontextVerwalter` hinterlegt und beim Aufruf weiterer Nachbarsysteme durchgereicht.

6.6.2 Realisierung

6.6.2.1 Einbinden der Bibliothek

Zur Realisierung der Autorisierung müssen die in Tabelle 17 aufgelisteten Bibliotheken eingebunden werden.

GroupId	**ArtifactId**
de.bund.bva.isyfact	isy-sicherheit
de.bund.bva.pliscommon	plis-serviceapi

Tabelle 17: Bibliotheken für die Autorisierung.

Die Bibliothek „isy-sicherheit“ enthält die Komponenten `Sicherheit` und `BerechtigungsManager`.

Die Bibliothek „plis-service-api“ enthält das Transportobjekt `AufrufKontextTo`, das zur Übermittlung der Authentifizierungsdaten über Schnittstellenaufrufe benutzt wird.

Zusätzlich wird als Abhängigkeit der `AufrufKontextVerwalter` (siehe Kapitel 6.5) benötigt, der die Informationen zum Aufrufer kennt.

6.6.2.2 Konfiguration der Sicherheitskomponente

Die Komponente Sicherheit wird als Spring-Bean in der Datei „`src/main/resources/resources/spring/querschnitt/sicherheit.xml`“ konfiguriert.

Die einer Rolle zugeordneten Rechte werden in der Datei „`/src/main/resources/resources/rollenrechte.xml`“ konfiguriert.

6.6.2.3 Prüfen der Berechtigung

Die Berechtigungsprüfung erfolgt in der Regel vor der fachlichen Verarbeitung in der Service-Schnittstelle oder im Dialog-Controller einer Anwendung. Dies erfolgt über Annotationen oder im Webflow (siehe [SicherheitNutzerdok]). Es kann auch jederzeit auf das Bean Sicherheit zugegriffen werden, um einen Berechtigungsmanager zu verwenden.

```
Berechtigungsmanager manager =
sicherheit.getBerechtigungsManager();

manager.pruefeRecht(RechteSchluessel.RECHT_MELDEN);
```

Über die Methoden des Berechtigungsmanagers (z.B. `hatRecht`, `pruefeRecht`) kann die Anwendung die Autorisierung durchführen.

6.7. Überwachung

In diesem Abschnitt wird beschrieben, wie die Überwachung einer Anwendung realisiert wird.

6.7.1 Designvorgaben

- Die Überwachungsschnittstelle wird per JMX angeboten.
- Die MBeans werden gemäß der Namenskonvention aus [ÜberwachungKonfigKonzept] benannt.
- MBeans enthalten keine Anwendungslogik. Keinesfalls darf fachliche Logik in MBeans implementiert werden. Allenfalls werden hier einfache Berechnungen (Durchschnittsbildung, Summierung usw.) durchgeführt.
- MBeans enthalten keine Management-Logik. Die `MBeans` sind einfache Datencontainer für Management-Informationen. `MBeans` sind dazu da, einem übergeordneten Management-System die zur Administration notwendigen Informationen zu liefern. Insbesondere wird in den MBeans keine Überwachungslogik implementiert.

- Business-Logik ruft Management-Logik. Der Anwendungskern und die MBean werden von einer Spring-Factory erzeugt. Der Anwendungskern ruft Methoden der MBean auf (Push-Konzept).

- Das Management-Interface darf nur die für Open MBeans erlaubten Datentypen für Parameter oder Rückgabewerte verwenden.[13]

- Die von Anwendungen bereitzustellenden Informationen sind in [ÜberwachungKonfigKonzept] aufgeführt.

- Jede Anwendung muss eine Service-Operation anbieten, die es nutzenden Nachbarsystemen erlaubt, die Erreichbarkeit dieses Systems zu prüfen (`Ping`-Methode).

- In jeder Anwendung wird ein Watchdog realisiert, welcher in regelmäßigen Abständen den Status des Systems prüft. Dazu ruft er eine Prüf-Methode der Anwendung auf. Der Aufruf der Prüf-Methode prüft den Status des Systems und aktualisiert das Ergebnis in der MBean.

- Die Prüf-Methode darf nur intern von einem Watchdog aufgerufen werden. Sie darf weder als Service-Methode, noch per JMX von „außen“ aufrufbar sein.

[13] Eine vollständige Liste dieser Typen ist in [UEBERWKONF] enthalten.

6.7.2 Klassendesign

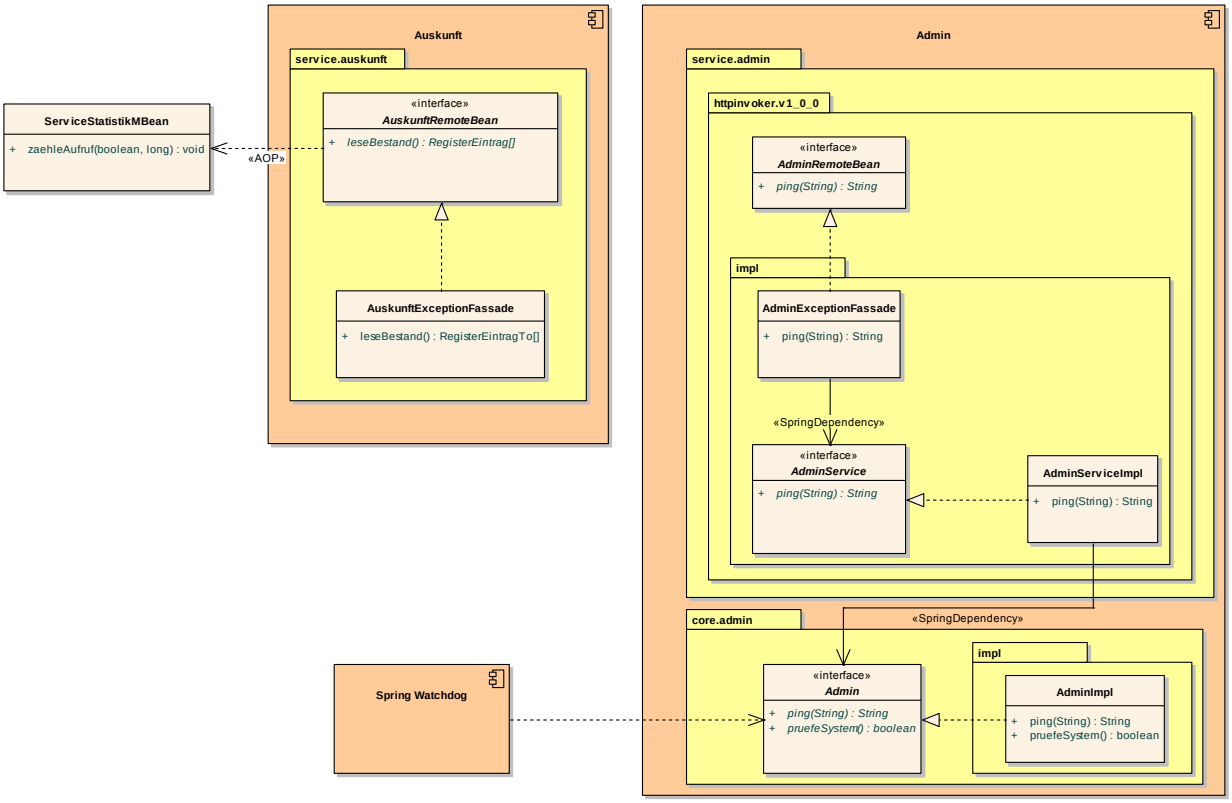

Abbildung 7: Klassendesign für die Überwachung

Abbildung 7 zeigt das Klassendesign für die Anwendungsüberwachung. Sie besteht zum einen aus der Service-Statistik, welche über die Klasse `ServiceStatistikMBean` angeboten wird. Diese wird per AOP beim Aufrufen einer Methode der `RemoteBean` aktualisiert (siehe Kapitel 5.1.3).

Zum anderen wird für die Überwachung eine `Ping`-Methode implementiert und als Service-Methode in der Admin-Komponente angeboten. Die Prüf-Methode wird in `AdminImpl` implementiert. Diese darf auf keinen Fall als Service-Methode angeboten werden.

6.7.3 Realisierung

6.7.3.1 Einbinden der Bibliothek

Zur Realisierung der Überwachung muss die in Tabelle 18 aufgelistete Bibliothek eingebunden werden.

GroupId	ArtifactId
de.bund.bva.pliscommon	plis-ueberwachung

Tabelle 18: Bibliothek für die Überwachung.

Die Bibliothek enthält `MBeans`, welche das von Anwendungen bereitzustellende Management-Interface implementieren.

6.7.3.2 Konfiguration der Überwachungsschnittstelle

Zum Anbieten der Service-Statistik sind alle benötigten Implementierungen in „plis-ueberwachung“ enthalten. Es muss lediglich die Spring-Konfiguration durchgeführt werden. Diese besteht aus zwei Teilen, welche in [ÜberwachungKonfigKonzept] im Detail beschrieben sind:

- Anbieten der MBeans über den Spring-MBean-Exporter.
- Anbinden der Zähl-Methode an den Anwendungskern durch einen AOP-Advice. Dieser Advice wird so konfiguriert, dass bei jedem Aufruf einer Methode der RemoteBean (siehe Kapitel 5.1.3) die Zähl-Methode der Statistik-MBean aufgerufen wird.

6.7.3.3 Implementierung der Ping- und Prüf-Methoden

Um die Verfügbarkeit bzw. Erreichbarkeit eines Systems automatisiert überprüfen zu können, muss eine Ping- und eine Prüf-Methode in der Komponente „Administration“ implementiert werden.

Die Ping-Methode wird per HttpInvoker als Service-Methode angeboten (siehe Kapitel 5.1). Die Implementierung besteht darin, einfach den übergebenen String zurückzugeben.

Die Prüf-Methode darf nicht als Service-Methode angeboten, sondern muss intern von einem Watchdog regelmäßig aufgerufen werden. Die Prüf-Methode muss für jedes System individuell implementiert werden. Als Grundsatz soll darin die Verfügbarkeit jedes Nachbarsystems und die aller genutzter Ressourcen (z.B. Datenbank) geprüft werden.

6.7.3.4 Konfiguration des Watchdogs

Der Watchdog wird per Spring konfiguriert - es ist keine Implementierung erforderlich. Eine Anleitung dafür ist in [ÜberwachungKonfigKonzept] enthalten.

6.8. LDAP-Zugriff

In diesem Abschnitt wird beschrieben, wie LDAP-Zugriffe in einer Anwendung realisiert werden. Dies kann notwendig sein, wenn ein Zugriff auf Daten notwendig ist, die noch nicht über eine querschnittliche Bibliothek oder einen Querschnittsdienst (z.B. Schlüsselverzeichnis) angeboten werden.

Für die Realisierung der LDAP-Zugriffe wird Spring-LDAP verwendet. Daher muss die in Tabelle 19 aufgelistete Bibliothek eingebunden werden, sofern diese noch nicht wegen vorhandener Dependencies durch Maven geladen wird.

GroupId	**ArtifactId**
org.springframework	spring-ldap

Tabelle 19: Bibliothek für Spring-LDAP.

6.8.1 Spring Konfiguration

In der Spring-Konfigurationsdatei müssen drei Einträge für die Nutzung von Spring LDAP gesetzt werden:

```
<bean id="contextSource"
class="org.springframework.ldap.pool.factory.PoolingContextSource">
  <property name="contextSource">
    <bean class="org.springframework.ldap.core.support.LdapContextSource">
      <property name="url" value="${ldap.url}" />
      <property name="userDn" value="${ldap.userdn}" />
      <property name="password" value="${ldap.password}" />
      <property name="base" value="${ldap.basedn}" />
      <property name="pooled" value="false" />
    </bean>
  </property>
  <property name="dirContextValidator">
    <bean
class="org.springframework.ldap.pool.validation.DefaultDirContextValidator"
/>
  </property>
  <property name="maxActive" value="${ldap.maxActive}" />
  <property name="maxTotal" value="${ldap.maxTotal}" />
  <property name="maxIdle" value="${ldap.maxIdle}" />
  <property name="minIdle" value="${ldap.minIdle}" />
  <property name="maxWait" value="${ldap.maxWait}" />
  <property name="whenExhaustedAction" value="${ldap.whenExhaustedAction}" />
  <property name="testOnReturn" value="${ldap.testOnReturn}" />
  <property name="testOnBorrow" value="${ldap.testOnBorrow}" />
  <property name="testWhileIdle" value="${ldap.testWhileIdle}" />
  <property name="timeBetweenEvictionRunsMillis"
value="${ldap.timeBetweenEvictionRunsMillis}" />
  <property name="numTestsPerEvictionRun"
value="${ldap.numTestsPerEvictionRun}" />
  <property name="minEvictableIdleTimeMillis"
value="${ldap.minEvictableIdleTimeMillis}" />
</bean>

<bean id="ldapTemplate" class="org.springframework.ldap.core.LdapTemplate">
  <constructor-arg ref="contextSource" />
</bean>

<bean id="ldapTemplateHolder"
        class="bva.bund.de.testdurchstich.springldap.LdapTemplateHolder">
  <property name="ldapTemplate" ref="ldapTemplate" />
</bean>
```

In der Bean vom Typ „`LdapContextSource`“ werden die zum Zugriff auf den LDAP benötigten Parameter definiert. Diese Bean wird so konfiguriert, dass sie kein Pooling durchführt (`pooled = false`). Andernfalls würde der LDAP-Pool des JDKs verwendet, welcher keine Prüfung von Verbindungen erlaubt und somit nach einem Failover des LDAPs defekte Verbindungen im Pool behält.

Anstelle des JDK-Pools wird die Implementierung von Spring verwendet. Dazu wird die `LdapContextSource`-Bean durch eine `PoolingContext`-Source-Bean gekapselt. Letztere führt das Pooling der LDAP-Verbindungen durch. In dieser Bean wird folglich auch der Pool

konfiguriert, insbesondere das Prüfen der Verbindungen vor deren Verwendung (`testOnBorrow = true`).

Die Bean „`ldapTemplate`“ definiert die Klasse, die den Zugriff auf den LDAP kapselt, sie benötigt nur die Bean „`contextSource`“ als Parameter.

Da gemäß *Nutzungsvorgaben Spring* [AnwendungskernDetailkonzept] Abschnitt 4.1.5 keine Beans für DAO-Objekte definiert werden dürfen, muss die Bean `LdapTemplate` in eine statische Klasse gesetzt werden, den `LdapTemplateHolder`. Dieser bietet das `LdapTemplate` dann über statische Methoden den jeweiligen DAO-Klassen (siehe Kapitel 6.8.2) an.

6.8.2 Realisierung

LDAP-Zugriffe sind keine eigene Bibliothek, daher wird im Folgenden eine DAO-Klasse vorgestellt, welche einen lesenden und schreibenden Zugriff auf einen LDAP zeigt. Der hier gezeigte Code umfasst das Auslesen der Rollen eines Benutzers sowie das Anlegen eines neuen Anwenders. Es wird exemplarisch gezeigt, wie über das `LdapTemplate` Suchen und Einfügen in den LDAP funktioniert.

6.8.2.1 Auslesen von Rollen

```
public List<String> getRollen(String uid, String
orgknz) {
    AndFilter filter = new AndFilter();
    filter.and(new EqualsFilter("uid", uid));
    filter.and(new EqualsFilter("orgknz", orgknz));

    List alleTreffer =
        LdapTemplateHolder.getLdapTemplate().
            search(DistinguishedName.EMPTY_PATH,
filter.encode(), new RollenContextMapper());
    if (alleTreffer == null || alleTreffer.size() == 0)
{
        throw new MyPlisTechnicalRuntimeException("Kein
Benutzer gefunden");
    }
    return (List<String>)alleTreffer.get(0);
}

private static class RollenContextMapper extends
AbstractContextMapper {
    public Object doMapFromContext(DirContextOperations
ctx) {
        List<String> ergebnis = new
ArrayList<String>();
        String[] rollen =
ctx.getStringAttributes("rollen");
        for (String rolle : rollen) {
            ergebnis.add(rolle);
        }
        return ergebnis;
    }
}
```

Aufgerufen wird in diesem Beispiel die obere Methode mit `uid` (User-ID) und `orgknz` (Organisationskennzeichen) eines Anwenders, womit ein Anwender eindeutig identifiziert ist.

In den ersten drei Zeilen wird die Suchbedingung definiert, wobei „`uid`“ und „`orgknz`“ die Namen der entsprechenden Felder im LDAP sind.

In dem Block dahinter wird über den `LdapTemplateHolder` das `LdapTemplate` geholt, und auf diesem die Methode `search` aufgerufen. Dieser Methode wird zuerst ein einschränkender Pfad übergeben, dann die Suchbedingung und danach die Abbildungsregel für das Ergebnis. Als einschränkender Pfad wird eine Konstante für den leeren Pfad übergeben, die Suchbedingung haben wir definiert und als Abbildungsregel wird eine neue Instanz von `RollenContextMapper` verwendet. Das Ergebnis der Suche wird dann zurückgegeben. Falls es zu keinem Treffer gekommen ist, wird eine Exception geworfen.

Die Klasse `RollenContextMapper` definiert das Abbilden von LDAP-Attributen auf Java-Objekte. Die Methode `doMapFromContext` wird einmal für jeden gefundenen Treffer aufgerufen, der übergebene Context enthält alle Werte des Treffers und zusätzliche Metainformationen. In unserer Klasse werden alle Rollen (Inhalt des LDAP-Attributes „`rollen`“) des Benutzers ausgelesen und als Liste zurückgegeben.

Zusammengefasst sucht diese Methode einen Benutzer, der durch seinen Anmeldenamen und sein Behörden-/Organisationskennzeichen identifiziert wird, und gibt die Rollen des Benutzers als Liste von Strings zurück.

6.8.2.2 Speichern eines Anwenders

Als Beispiel zum Speichern wird hier das Neuanlegen eines Anwenders gezeigt.

Die Klasse Anwender ist ein reines Transportobjekt mit Getter- und Setter-Methoden und wird nicht weiter erläutert.

```
public void speicherAnwender(Anwender anwender) {
    Name dn = buildDn(anwender);
    DirContextAdapter adapter = new
DirContextAdapter(dn);
    adapter.setAttributeValues("objectclass", new
String[] {"top",
        "person", "organizationalperson", "anwender"});
    adapter.setAttributeValue("cn",
anwender.getBenutzerName());
    adapter.setAttributeValue("sn",
anwender.getNachName());
    adapter.setAttributeValue("orgknz",
anwender.getOrgknz());
    adapter.setAttributeValues("rollen",
anwender.getRollen());
```

```
    adapter.setAttributeValue("uid",
anwender.getUid());
    adapter.setAttributeValue("passwort",
"InitialPasswort");
    adapter.setAttributeValue("status", "gueltig");
    LdapTemplateHolder.getLdapTemplate().bind(dn,
adapter, null);
}

private Name buildDn(Anwender anwender) {
    DistinguishedName name = new DistinguishedName();
    name.add("o", anwender.getOrganisation());
    name.add("ou", anwender.getBehoerde());
    name.add("cn", anwender.getBenutzerName());
    return name;
}
```

In der ersten Zeile der Methode wird die Methode `buildDn` aufgerufen, die den Distinguished-Name des Objektes zusammenbaut. Der Distinguished-Name dient zur eindeutigen Identifizierung eines Anwenders, sein Aufbau ist vom Schema des LDAP abhängig.

In den weiteren Zeilen wird ein Context-Adapter mit den Werten des Anwenders befüllt, wobei jeweils angegeben werden muss, welches LDAP-Attribut mit welchem Wert befüllt wird. Bei der Befüllung muss darauf geachtet werden, dass alle Pflichtattribute der angegebenen Objektklassen gesetzt werden, das Attribut „`objectclass`“ ist immer Pflicht.

In der letzten Zeile der Methode wird wiederum das `LdapTemplate` aufgerufen und mit der Methode `bind` ein neuer Eintrag im LDAP angelegt. Als erster Parameter wird der `DN` des Eintrags mitgeliefert, in den Parametern zwei und drei werden alle zu setzenden Attribute übergeben, entweder als Context oder als Sammlung von Attributen.

7. Quellenverzeichnis

[AnwendungskernDetailkonzept]
Detailkonzept der Komponente Anwendungskern
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Anwendungskern.pdf.

[BatchDetailkonzept]
Detailkonzept Komponente Batch
10_Blaupausen/technische_Architektur/Detailkonzept_Komponente_Batch.pdf.

[DatenzugriffDetailkonzept]
Detailkonzept Komponente Datenzugriff
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Datenzugriff.pdf.

[FehlerbehandlungKonzept]
Konzept Fehlerbehandlung
20_Bausteine\Fehlerbehandlung\Konzept_Fehlerbehandlung.pdf.

[IsyFactEinstieg]
IsyFact-Einstieg
00_Allgemein\IsyFact-Einstieg.pdf.

[IsyFactReferenzarchitektur]
IsyFact – Referenzarchitektur
00_Allgemein\IsyFact-Referenzarchitektur.pdf.

[LoggingKonzept]
Konzept Logging
20_Bausteine\Logging\Konzept_Logging.pdf.

[ProduktKatalog]
IsyFact Produktkatalog
00_Allgemein\Produktkatalog.pdf.

[ProtokollierungKonzept]
Konzept Protokollierung
20_Bausteine\Protokollierung_Protokollrecherche\Konzept_Protokollierung.pdf.

[SicherheitNutzerdok]
Nutzerdokumentation Sicherheit
20_Bausteine\Sicherheitskomponente\Nutzerdokumentation_Sicherheit.pdf.

[Spring]
Spring Framework Reference Documentation
http://docs.spring.io/spring-framework/docs/3.1.x/spring-framework-reference/html/.

[ÜberwachungKonfigKonzept]
Konzept Überwachung und Konfiguration
20_Bausteine\Ueberwachung_Konfiguration\Konzept_Ueberwachung-Konfiguration.pdf.

[VorlageAnwendung]

Beispielimplementierung „Vorlage-Anwendung“
Wird auf Anfrage bereitgestellt.

8. Abbildungsverzeichnis

9. Tabellenverzeichnis