Bundesverwaltungsamt

IsyFact-Standard

Detailkonzept Komponente Anwendungskern

Version 2.15
08.04.2016

IsyFact Standards

„Detailkonzept Komponente Anwendungskern“
des Bundesverwaltungsamts ist lizenziert unter einer
Creative Commons Namensnennung 4.0 International Lizenz.

Ansprechpartner:

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

Dokumentinformationen

Dokumenten-ID:	Detailkonzept_Komponente_Anwendungskern.docx

Inhaltsverzeichnis

1. Einleitung 5

2. Überblick 6

3. Aufbau des Anwendungskerns 7

3.1. Die Fachkomponenten 7
3.2. Das Anwendungskern-Framework 8
3.3. Externe Services 8
3.4. Austausch von Geschäftsobjekten 8

4. Verwendung des Spring-Frameworks 10

4.1. Anforderungen 10
4.2. Konfiguration von Spring 11

4.2.1 Die Konfiguration von Spring-Beans 11
4.2.2 Konfiguration von Spring im Tomcat 16
4.2.3 webAppRootKey konfigurieren 16
4.2.4 Spring ContextLoaderListener konfigurieren 16
4.2.5 Spring DispatcherServlet konfigurieren 16
4.2.6 Spring Watchdog konfigurieren 18

4.3. Die direkte Verwendung des Spring-Frameworks 18

4.3.1 Keine Beans per Namen auslesen 18
4.3.2 Statischer Zugriff auf Bean-Instanzen nur in Sonderfällen 19

4.4. Aspektorientierte Programmierung in Spring 19

4.4.1 AOP für Transaktionssteuerung verwenden 19
4.4.2 AOP für Berechtigungsprüfungen verwenden 20
4.4.3 AOP nicht für das Logging von Exceptions verwenden 20

5. Quellenverzeichnis 21

6. Abbildungsverzeichnis 22

1. Einleitung

Im Anwendungskern werden die fachlichen Funktionen der Anwendung realisiert. Diese Funktionen sind im Anwendungskern genau einmal implementiert und können von unterschiedlichen Komponenten einer Anwendung genutzt werden, z. B. kann eine Meldung in einer Fachanwendung sowohl über die GUI als auch über einen Batch oder einen Serviceaufruf vorgenommen werden. Alle diese Aufrufer nutzen dieselbe Implementierung. Der Anwendungskern wird aus Fachkomponenten und Komponenten zur Nutzung externer Services aufgebaut.

Die Implementierung wird durch die Verwendung eines Anwendungskern-Frameworks unterstützt. Die Hauptaufgabe des Anwendungskern-Frameworks ist es, die Komponenten zu konfigurieren und miteinander bekannt zu machen. Als zentrales Framework für die Implementierung des Anwendungskerns wird das Spring-Framework verwendet.

Neben der Verwendung als Anwendungskern-Framework, bietet das Spring-Framework auch weitere Funktionalität, die die Entwicklung von Anwendungen für die Plattform vereinfacht.

Dieses Dokument richtet sich an Architekten und Entwickler. Neben der Festlegung der Architektur des Anwendungskerns ist die Hauptaufgabe dieses Dokuments, die Verwendung von Spring als Anwendungskern-Framework zu erläutern. Daher werden im Folgenden Spring-Kenntnisse vorausgesetzt.

Für ein Verständnis der in diesem Dokument geforderten Spring-Nutzung ist vor allem das Codebeispiel Vorlage-Anwendung [VorlageAnwendung] zu empfehlen, bestehend aus den beiden Anwendungssystemen Vorlage-Register und Vorlage-Geschäftsanwendung.

2. Überblick

Gemäß den Vorgaben der Referenzarchitektur der IsyFact [IsyFactReferenzarchitektur] basiert die Architektur eines IT-Systems auf der bekannten Drei-Schichten-Architektur. Eine dieser Schichten ist die Anwendungskernschicht mit der Komponente Anwendungskern.

Im Anwendungskern werden die fachlichen Funktionen der Anwendung realisiert. Dazu gehören zum Beispiel Datenvalidierungen oder Funktionen zum Verarbeiten von Geschäftsobjekten, die über die Komponente Datenzugriff geladen wurden. In Richtung Nutzungsschicht bietet der Anwendungskern nur Funktionalitäten an, die allgemein verwendbar sind und die im Prinzip von mehreren Nutzern verwendet werden könnten.

In Abbildung 1 ist die Referenzarchitektur eines IT-Systems noch einmal dargestellt, wobei die Anwendungskernschicht mit der Komponente Anwendungskern hervorgehoben ist.

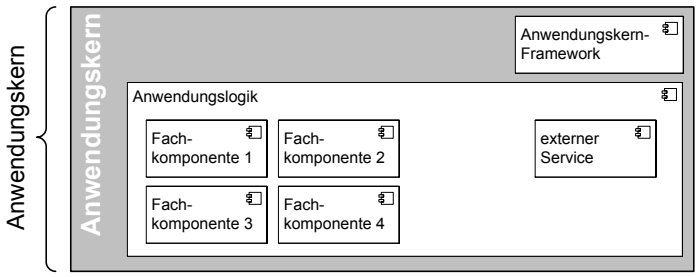

Abbildung 1: Referenzarchitektur eines IT-Systems

In diesem Dokument werden die Vorgaben zum Bau der Komponente Anwendungskern im Detail beschrieben. Als zentrales Framework für die Implementierung des Anwendungskerns wird das Spring-Framework verwendet.

Neben der Verwendung als Anwendungskern-Framework bietet das Spring-Framework auch weitere Funktionalität, die die Entwicklung von Anwendungen für die Plattform vereinfacht. Daher werden in diesem Dokument auch Aspekte des Spring-Frameworks behandelt, die über die Verwendung im Anwendungskern hinausgehen.

3. Aufbau des Anwendungskerns

Der Anwendungskern ist der zentrale Baustein eines IT-Systems. Hier werden die fachlichen Funktionen der Anwendung realisiert. In Abbildung 2 sind drei Aspekte des Anwendungskerns dargestellt. Er besteht aus zwei wesentlichen Bestandteilen: Den Fachkomponenten und dem Anwendungskern-Framework. Weiterhin ist dort dargestellt, dass externe Services als Komponente in den Anwendungskern integriert sind. Diese drei Aspekte werden im Folgenden im Detail beschrieben.

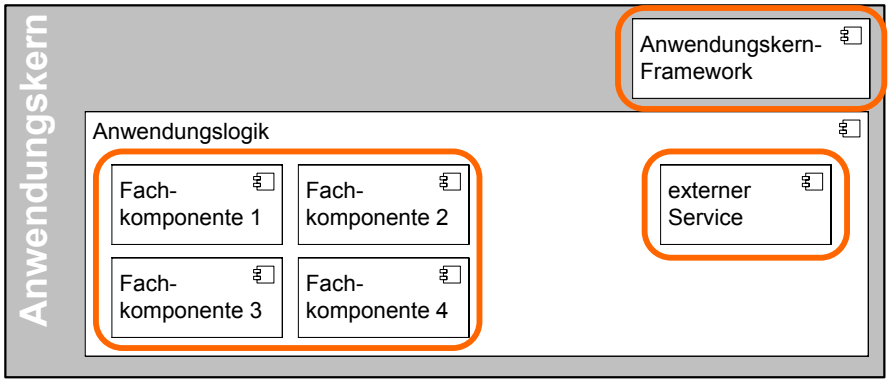

Abbildung 2: Bestandteile des Anwendungskerns

3.1. Die Fachkomponenten

Die Umsetzung einer Komponente aus der fachlichen Architektur erfolgt durch eine Fachkomponente. Damit entsprechen die Fachkomponenten dem fachlichen Komponentenschnitt aus der fachlichen Architektur. Diese Komponenten implementieren weitgehend reine Fachlichkeit und trennen so Anwendungslogik und Technologie. Das ist der Schlüssel für gute Wartbarkeit und einfache Weiterentwickelbarkeit des Anwendungskerns. Diese Struktur findet sich auch in anderen Schichten wieder, z. B. in der Persistenz: dort sind die Fachkomponenten auch gemäß den Komponenten aus der fachlichen Architektur strukturiert. Im Vorlage-Register sind z.B. die Komponenten „Auskunft“, „Meldung“ und „Protokollierung“ Fachkomponenten.

Die Fachkomponenten operieren auf Geschäftsobjekten (Entitäten). Jedes Geschäftsobjekt ist genau einer Komponente zugeordnet, die die Datenhoheit über dieses Geschäftsobjekt besitzt und seinen Lebenszyklus verwaltet. Im Vorlage-Register gibt es z.B. die Entitäten „CD“, „Interpret“ und „Track“.

Es kann aber auch den Fall geben, dass ein Geschäftsobjekt fachlich nicht genau einer Komponente zugeordnet werden kann. Hierfür wird die Komponente „Basisdaten“ definiert, die nur die eine Aufgabe hat, solche Geschäftsobjekte an zentraler Stelle effizient zu verwalten. Auch das Vorlage-Register besitzt die Komponente „Basisdaten“.

3.2. Das Anwendungskern-Framework

Für querschnittliche Funktionalität innerhalb des Anwendungskerns wird das Spring-Framework genutzt. Hauptaufgabe des Frameworks ist es, die Komponenten zu konfigurieren und miteinander bekannt zu machen. Dadurch wird die Trennung zwischen Fachlichkeit und Technik verbessert. Beispiel für querschnittliche Funktionalität ist die deklarative Steuerung von Transaktionen.

Die Vorgaben zur Nutzung und Konfiguration des Spring-Frameworks werden in Kapitel 4 beschrieben.

3.3. Externe Services

Wenn der Anwendungskern fachliche Services benötigt, die von anderen IT-Systemen innerhalb der Plattform angeboten werden, so werden diese Services als Komponente im Anwendungskern abgebildet. Dadurch ist die Funktionalität sauber gekapselt, was die Wartbarkeit erhöht: Wenn der externe Service ausgetauscht werden soll, ist keine Änderung der gesamten Anwendung notwendig – es ist lediglich eine interne Änderung der externen Service-Komponente notwendig. Für andere fachliche Komponenten des Anwendungskerns ist nicht zu unterscheiden, ob es sich beim Aufruf einer Komponentenschnittstelle um eine in dieser Komponente implementierte Funktion oder um einen Serviceaufruf handelt. Komponenten, die externe Services kapseln, sind im Idealfall von außen nicht von fachlichen Komponenten des Anwendungskerns unterscheidbar. Neben der technischen Kommunikation erfolgt durch diese Komponenten auch die Abbildung von Schnittstellendaten und Exceptions auf die Daten der Anwendung.

Im Vorlage-Register werden Services des Schlüsselverzeichnisses der IsyFact-Erweiterungen als externe Services genutzt.

3.4. Austausch von Geschäftsobjekten

Der Anwendungskern hat die Hoheit über die Geschäftsobjekte des IT-Systems. In [IsyFactReferenzarchitektur] wurde bereits beschrieben, dass zur Strukturierung eines IT-Systems zwei Sichten verwendet werden können:

Technische Sicht: Die technische Sicht unterteilt die IT-Anwendung in Schichten für die jeweils eigene Technologien verwendet werden: Nutzung (GUI, Batch und Service), Anwendungskern und Datenzugriff. Das Ziel ist es, eine Durchmischung von Technologien zu verhindern, so darf z. B. in der GUI kein SQL formuliert werden.

Fachliche Sicht: Die fachliche Sicht beschreibt eine Teilaufgabe, z. B. Meldung, Auskunft oder Fristenkontrolle. Innerhalb dieser Teilaufgabe wird der Anwendungskern über eine GUI, Services oder Batches angesprochen, der wiederum über die Datenzugriffsschicht mit der Datenbank kommunizieren muss. Daher zerlegt die fachliche Sicht das IT-System in fachliche Säulen. Diese fachlichen Säulen werden im Folgenden Teilanwendungen genannt. Das Ziel ist die fachliche

Trennung und Minimierung inhaltlicher Abhängigkeiten. Die Teilanwendungen sollen Fachlichkeit kapseln.

Damit kann eine IT-Anwendung sowohl vertikal in Teilanwendungen (fachliche Sicht) und horizontal in Schichten (technische Sicht) strukturiert werden, wie in Abbildung 3 dargestellt.

Abbildung 3: Schichten und Teilanwendungen

Eine Teilanwendung erstreckt sich über alle technischen Schichten, kapselt aber die Fachlichkeit, zu der gewisse Geschäftsobjekte gehören. Um den Austausch von Objekten innerhalb einer Teilanwendung zwischen den Schichten zu vereinfachen, gleichzeitig aber unterschiedliche Teilanwendungen gegeneinander abzugrenzen, wird für die Verwendung von Objekten in Schnittstellen folgende Regel aufgestellt:

- Zwischen zwei Teilanwendungen dürfen nur Objekte ausgetauscht werden, deren Modifikation keine Auswirkungen auf die liefernde Teilanwendung hat.

Das kann erreicht werden, indem nur Deep-Copies von Objekten an andere Teilanwendungen herausgegeben werden.

Innerhalb einer Teilanwendung dürfen über die Schichten hinweg durchaus änderbare Objekte ausgetauscht werden. Die hierfür bereitgestellte Schnittstelle der Schicht gehört damit aber zur Teilanwendung darf von einer anderen Teilanwendung nicht genutzt werden.

Im Vorlage-Register sind z.B. „Meldung“ und „Auskunft“ Teilanwendungen. Zur Teilanwendung „Meldung“ gehören in der Schicht Nutzung der Service „Meldung“ und der Batch „Meldung“, in der Schicht Anwendungslogik die Fachkomponente „Meldung“ und der Schicht Datenhaltung die Entitäten CD, Name, Interpret und Track mit der Komponente Basisdaten.

4. Verwendung des Spring-Frameworks

Das Spring-Framework ist ein Java EE Framework, welches im Kern sehr verständlich und leicht zu verwenden ist. In ihm werden die Bestandteile eines Systems als „Beans“ definiert. Neben seiner Kern-Funktionalität der Verwaltung, Konfiguration und aspektorientierten Erweiterung von Beans bietet Spring viele Funktionalitäten, welche die Entwicklung einer Anwendung erleichtern sollen. Aber nicht alle Funktionalitäten werden in den Anwendungen der Plattform verwendet.

Verwendet werden die folgenden Spring-Funktionalitäten. Die Festlegungen hierzu werden in den zugehörigen Detailkonzepten beschrieben:

- Die Konfiguration des Loggings über Spring, beschrieben im Konzept [NutzungsvorgabenLogging].
- Die Bereitstellung von JMX Beans über Spring, beschrieben im Konzept [ÜberwachungKonfigKonzept].
- Die Verwendung der `HttpInvoker`-Schnittstellentechnologie, beschrieben im Konzept [ServiceDetailkonzept].
- Die Zugriffe auf ein LDAP erfolgen über Spring-LDAP, beschrieben im Konzept [SpringLDAPNutzungskonzept].
- Die Implementierung von Dialogabläufen erfolgt über Spring-WebFlow, beschrieben im Dokument [WebGUIDetailkonzept].

Alle anderen Spring-Funktionalitäten (Validierung über Spring, Emailing, Thread Pooling, Scripting, Annotations) werden nicht verwendet.

Dieses Kapitel teilt sich in vier Teile:

- Die Auflistung der Anforderungen an die Verwendung des Spring-Frameworks.
- Die Vorgaben für die Konfiguration der Spring-Beans sowie von Spring selbst.
- Die Vorgaben für den direkten Zugriff auf das Spring-Framework in der Anwendung.
- Die Vorgaben für aspektorientierte Programmierung mit Spring.

Nicht beschrieben wird hier der Einsatz von Tools für die Entwicklung von Spring-Anwendungen.

4.1. Anforderungen

Für die Nutzung von Spring gelten folgende Anforderungen:

Reduzierung der Komplexität: Spring soll so eingesetzt werden, dass es die Komplexität der Anwendungen und den Aufwand ihrer Erstellung reduziert. Dies bezieht sich unter anderem auf die Bestandteile von Spring: Spring bietet verschiedene Komponenten, welche getrennt voneinander eingesetzt werden können. Es sollen nur die Kom-

ponenten eingesetzt werden, welche zu geringerer Komplexität und geringerem Entwicklungsaufwand führen.

Einheitlichkeit der Nutzung: Spring soll in den verschiedenen Anwendungen einheitlich eingesetzt werden. Hierfür sind geeignete Vorgaben für die Nutzung zu verwenden.

Verständlichkeit der Konfiguration: Die Konfiguration der Spring-Komponenten wird in hierarchisch gegliederten XML-Dateien durchgeführt. Diese XML-Dateien sollen übersichtlich und verständlich sein.

Komponentenorientierung wahren: Über Spring sollen Komponenten konfiguriert werden: Es soll nicht möglich sein, direkt auf Implementierungsklassen einer Komponente zuzugreifen.

4.2. Konfiguration von Spring

Das grundlegende Konzept von Spring ist das der Spring-Bean. Die Konfiguration von Spring teilt sich deshalb in zwei Teile: In die Konfiguration der Spring-Beans sowie in die Konfiguration von Spring selbst (innerhalb eines Tomcat Servers). Diese Konfigurationsarten werden in den folgenden Kapiteln beschrieben.

4.2.1 Die Konfiguration von Spring-Beans

Spring ist ein Applikations-Container, welcher sogenannte Spring-Beans instanziiert, per „Dependency Injection“ konfiguriert und bereitstellt. Spring-Beans sind beliebige Java-Klassen. Für diese Klassen kann man benötigte andere Beans oder Konfigurationsparameter konfigurieren, welche der Klasse daraufhin im Konstruktor oder per `set`-Methode übergeben werden. Konfiguriert werden Beans und ihre Abhängigkeiten in XML-Dateien.

Die folgende Abbildung zeigt einen Ausschnitt der für das Vorlage-Register erstellten Beans. Diese werden grün dargestellt und besitzen „referenziert“ Abhängigkeiten zu benötigten anderen Beans.

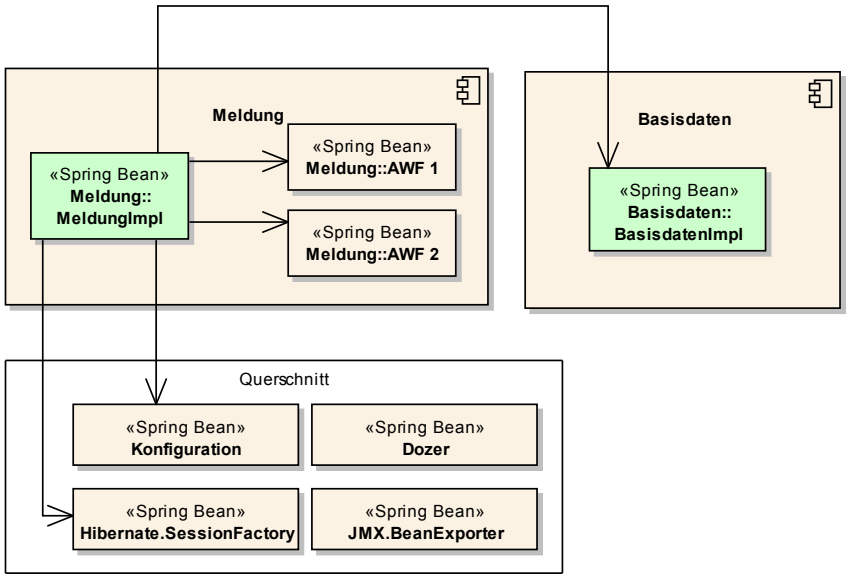

Abbildung 4: Spring-Beans des Vorlage-Registers

Die Abbildung zeigt bereits, dass nicht alle Klassen der Anwendung als Beans konfiguriert werden: Für die Komponenten `Meldung` und `Basisdaten` wird je eine Klasse als Bean konfiguriert, welche die Funktionalität der Komponente bereitstellt. Generell gilt, dass jede zentrale und wichtige Klasse aber als Spring Bean konfiguriert werden sollte.

Für die Modellierung und Konfiguration der Spring-Beans werden im Folgenden Vorgaben aufgestellt.

4.2.1.1 Konfiguration einer Komponente

Anwendungen werden in Form von Komponenten implementiert. Jede Komponente besitzt ein Interface mit den Operationen aller Anwendungsfallklassen der Komponente. Auf eine Komponente wird nur über ihr Interface zugegriffen. Das Interface realisiert damit eine Fassadenfunktion. Dies wird in der folgenden Abbildung dargestellt.

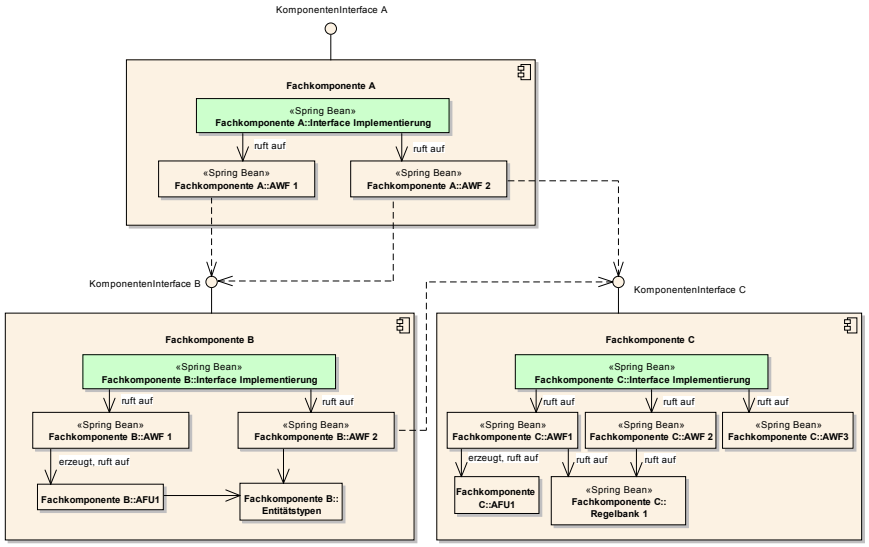

Abbildung 5: Komponenten: Ihre Interfaces und ihre als Beans konfigurierten Implementierungen

Die Komponenten-Fassaden (oder Verwalter-) Klasse implementiert das Komponenten-Interface. In obiger Abbildung ist sie grün dargestellt. Diese Klasse referenziert die weiteren zentralen Klassen der Komponente, insbesondere die Anwendungsfallklassen. Aufrufe werden von der Fassade an die Anwendungsfallklassen weitergegeben.

Ein Beispiel dafür ist die Klasse `MeldungImpl` des Vorlage-Registers.

Eine Komponente sollte durchgängig über Spring konfiguriert werden. Alle relevanten und zentralen Klassen werden daher als Spring Beans konfiguriert. Das umfasst vor allem aber nicht ausschließlich die Fassade und zugehörige Anwendungsfallklassen. Bei Anwendungsfunktionsklassen oder Hilfsklassen ist je nach Relevanz zu entscheiden, ob diese als eigene Spring Beans definiert werden. Im Zweifel sollte die Konfiguration über Spring bevorzugt werden. Wenn eine Klasse nur an einer Stelle genutzt wird, kann sie als Kompromiss auch als anonyme Spring Bean definiert werden. Sind Klassen nicht von relevanter Bedeutung, so können sie beim Erzeugen der Spring Bean programmatisch erzeugt werden.

4.2.1.2 Querschnittsdienste als Beans konfigurieren

Querschnittsdienste (etwa für JMX, für Nachrichten, für die Versendung von Mails) sind als Beans zu konfigurieren. Dies gilt unabhängig davon, ob die Querschnittsdienste statisch verfügbar sind oder über Dependency Injection gesetzt werden (siehe Kapitel 4.3.2).

Ebenfalls über Beans durchzuführen ist die Konfiguration diverser Frameworks, z.B. Hibernate. Die Konfiguration dieser Frameworks wird in ihren Nutzungsvorgaben beschrieben.

4.2.1.3 Kein Autowiring verwenden

Die Verwendung von Autowiring ist verboten, da es das Verständnis der Zusammenhänge und die Fehlersuche erschwert.

Über den Autowiring-Mechanismus von Spring ist es möglich, die `set`-Methoden von Beans während der Initialisierung von Spring beim Start der Anwendung nicht nur anhand der in den XML-Konfigurationsdateien abgelegten Konfiguration aufzurufen, sondern auch implizit über Namenskonventionen. Das muss aber explizit in Spring konfiguriert werden.

4.2.1.4 Beans standardmäßig als Singletons definieren

Beans können entweder als Singletons mit nur einer Instanz, mit einer Instanz pro Aufruf oder mit einer Instanz pro Abhängigkeit (Prototype) erzeugt werden. Die Komponenten-Beans einer Anwendung sollen zustandslos sein und werden als Singleton-Beans erzeugt. Wo technisch erforderlich, können auch andere Scopes verwendet werden.

4.2.1.5 Spring-Konfiguration hierarchisch aufbauen

Die Spring-Konfiguration soll hierarchisch aufgebaut werden. Das bedeutet, dass zunächst je Komponente der Anwendung eine Spring-Konfiguration erstellt wird. Hier werden alle Konfigurationen eingetragen, die diese eine Komponente betreffen.

Konfigurationen, die die gesamte Anwendung betreffen werden ebenfalls in einer oder mehreren Konfigurationsdateien abgelegt. Dabei sollen logisch zusammengehörende Konfigurationen auch in einer Datei abgelegt werden.

Für die gesamte Anwendung wird eine Konfigurationsdatei application.xml erstellt, die nur alle von der Anwendung benötigten Konfigurationsdateien über `import`-Tags zusammenfasst.

4.2.1.6 Konfigurationsparameter über Property-Objekte konfigurieren

Die Konfiguration der Anwendung wird (wie im Detailkonzept Überwachung und Konfiguration [ÜberwachungKonfigKonzept] beschrieben) über ein Property-Objekt durchgeführt. Dieses Property-Objekt enthält sämtliche Konfigurationsparameter der Anwendung. Das Setzen einzelner Konfigurationsparameter über `set`-Methoden ist nur in Ausnahmefällen erlaubt: Es ist stets das Property-Objekt zu übergeben.

Ein Beispiel dafür ist die Konfigurations-Bean `Konfiguration` in der Datei `querschnitt.xml` des Vorlage-Registers. Diese Bean wird den Komponenten in der Datei `komponenten.xml` per Dependency Injection übergeben.

Abweichungen für technische Beans sind erlaubt.

4.2.1.7 Abhängigkeiten von Komponenten über set-Methoden konfigurieren

Benötigte Beans können entweder über `set`-Methoden oder über Konstruktor-Parameter injiziert werden. Falls eine benötigte Bean nicht zum Zeitpunkt der Objekterzeugung vorhanden sein muss, sollte sie über eine `set`-Methode und nicht über einen Konstruktor-Parameter konfiguriert

werden. Grund dafür ist, dass bei Konstruktor-Parametern eine zusätzliche Reihenfolge-Abhängigkeit erzeugt wird.

Falls eine Bean erkennen muss, dass alle von ihr benötigten Beans konfiguriert wurden, hat sie das Interface `InitializingBean` sowie die Methode `afterPropertiesSet()` zu implementieren: Diese Methode wird erst aufgerufen, wenn alle Konfigurationen durchgeführt wurden.

In einigen Fällen müssen benötigte Beans bei der Instanzerzeugung vorliegen. Ein Beispiel dafür ist die Property-Bean mit der Anwendungskonfiguration: Sie wird ggf. benötigt, um bei fehlenden Konfigurationsparametern im Konstruktor eine Exception zu werfen und damit den gesamten Spring-Konfigurationsprozess zu stoppen.

Falls ein oder mehr Parameter einer Bean als Konstruktor-Parameter konfiguriert werden müssen, sind die anderen Parameter trotzdem über `set`-Methoden zu injizieren.

4.2.1.8 Required-Annotation verwenden

Um Fehler in der Spring-Konfiguration frühzeitig erkennen zu können, sollen zwingend erforderliche Dependencies mit der Annotation @Required versehen werden.

4.2.1.9 Explizite Abhängigkeits-Konfiguration bei Nachrichten

Falls man innerhalb des Konstruktors eine andere Bean benötigt, so konfiguriert man diese per Konstruktor-Argument (siehe Kapitel 4.2.1.7). In einigen Fällen wird auf Beans aber nicht über derartige Dependency Injection, sondern über statische Methoden zugegriffen (siehe Kapitel 4.3.2). Hierzu gehören im Vorlage-Register die Protokoll-Bean sowie die ResourceBundle-Bean.

Auf Beans kann auch statisch über eine Bean-Holderklasse[1] zugegriffen werden. Eine Bean-Holderklasse wird als Spring-Bean mit einer Abhängigkeit definiert. Beim Aufruf der `set`-Methode für die abhängige Bean wird diese in eine statische Variable gespeichert. Falls in einem Bean-Konstruktor oder einer `set`-Methode statisch auf eine Holder-Klasse zugegriffen wird, so kennt Spring die Abhängigkeit zur Bean der Holder-Klasse nicht. Es muss sichergestellt werden, dass die Holder-Klasse bereits befüllt wurde. Hierzu ist ein `depends-on` Attribut in die Bean-Konfiguration einzufügen. Diese muss den Bean-Namen der benutzten Holder-Klassen enthalten.

Für die Beans Template-Holder und Protokoll ist dies nicht notwendig: Für ihre Verwendung muss eine Transaktion geöffnet worden sein, und der Transaktionsmanager besitzt bereits eine `depends-on` Beziehung zu

[1] Holder-Klasse stellen Beans über statische Methoden bereit. Diese sind nur im Ausnahmefall zu verwenden. Im Vorlage-Register gibt es drei Holder-Klassen: `JpaTemplateHolder`, `ProtokollHolder` und `SpringContextHolder`.

diesen Beans. Lediglich für die Verwendung von Resource-Bundles im Konstruktor ist eine Abhängigkeit zu konfigurieren.

4.2.2 Konfiguration von Spring im Tomcat

Eine Anwendung der IsyFact ist technisch gesehen eine Tomcat Webanwendung. Der Spring-Kontext muss beim Start dieser Webanwendung instanziiert werden, damit alle benötigte Funktionalität (die HttpInvoker-Beans, die JMX-Beans, die Watchdog-Bean, die gesamte verwendete Anwendungslogik etc.) bereitgestellt wird. Hierfür stellt Spring Funktionalitäten bereit, die in der `web.xml` konfiguriert werden müssen. Diese werden in den folgenden Abschnitten beschrieben. Eine exemplarische `web.xml` findet sich im Vorlage-Register.

4.2.3 webAppRootKey konfigurieren

Der Kontextparameter `webAppRootKey` muss auf den eindeutigen Namen der Webanwendung gesetzt werden, bspw.:

```
<context-param>
    <param-name>webAppRootKey</param-name>
    <param-value>CD-Register</param-value>
</context-param>
```

Hintergrund: Spring speichert standardmäßig den Pfad zum Wurzelverzeichnis der Webanwendung im Webserver in der Kontextvariable `webapp.root`. Wenn mehrere Anwendungen gleichzeitig in einem Tomcat betrieben werden (bspw. in einer Entwicklungsumgebung), wird dieser Parameter durch die verschiedenen Anwendungen überschrieben. Dies kann zu ungewünschten Seiteneffekten führen. Ist der Kontextparameter `webAppRootKey` wie im obigen Beispiel gesetzt, wird der Pfad statt im Parameter `webapp.root` im Parameter `CD-Register` abgelegt. Da jede Webanwendung einen eindeutigen Namen besitzt, und damit einen eigenen Kontextparameter verwendet, wird das Überschreiben vermieden.

4.2.4 Spring ContextLoaderListener konfigurieren

Damit der Spring-Kontext beim Start des Tomcat geladen wird, ist `org.springframework.web.context.ContextLoaderListener` als Listener zu konfigurieren. Ihr sind über den Konfigurationsparameter `contextConfigLocation` die Spring-Konfigurationsdateien bekannt zu machen.

4.2.5 Spring DispatcherServlet konfigurieren

Falls eine Anwendung über eine HttpInvoker-Bean zugreifbar sein soll, muss im Tomcat das Spring `DispatcherServlet` konfiguriert werden. Das Dispatcher-Servlet nimmt Aufrufe entgegen und leitet sie an die Beans weiter, welche sich für die entsprechende URL registriert haben.

Ein Beispiel für die Konfiguration des Dispatcher-Servlets findet sich in der Datei `web.xml` des Vorlage-Registers:

```
<servlet>
  <servlet-name>remoting</servlet-name>
  <servlet-
class>org.springframework.web.servlet.DispatcherServlet</serv
let-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/classes/resources/spring/remoting-servlet.xml
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>remoting</servlet-name>
  <url-pattern>/CdRegister</url-pattern>
</servlet-mapping>
```

Für das Dispatcher-Servlet muss eine eigene Spring-XML-Datei als Konfiguration bereitgestellt werden (hier Datei `remoting-servlet.xml`). Die Beans dieser Datei werden als eigener Anwendungskontext bereitgestellt. Es gibt somit zwei Spring-Kontexte:

- Der durch den Spring-Listener erzeugte Kontext: Dies ist der „eigentliche“ Kontext.
- Der Kontext des Dispatcher-Servlets: Dieser Kontext ist ein „Kind“ Kontext des eigentlichen Kontextes. Ihm sind alle im eigentlichen Kontext definierten Beans bekannt.

Der Zugriff aus dem DispatcherServlet-Springkontext auf den eigentlichen Kontext geschieht somit direkt über die Definition von Bean-Referenzen im Dispatcher-Kontext. Die fachliche Verarbeitung eines HttpInvoker-Aufrufs soll im eigentlichen Kontext durchgeführt werden. Deshalb soll in der `HttpInvokerServiceExporter` Bean konfigurierte Service-Bean im eigentlichen Anwendungskontext konfiguriert sein.

Dies wird in der folgenden Anwendung illustriert.

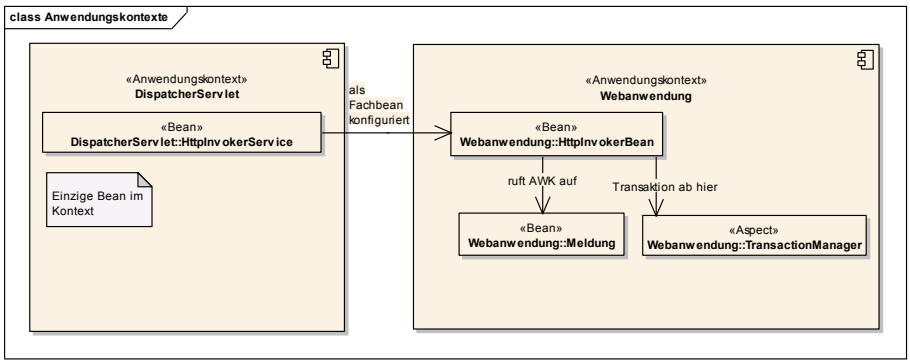

Abbildung 6: Aufruf-Weiterleitung eines HttpInvoker Aufrufs

Eine detailliertere Beschreibung der Umsetzung von HttpInvoker-Schnittstellen findet sich im Detailkonzept Komponente Service (siehe [ServiceDetailkonzept]).

4.2.6 Spring Watchdog konfigurieren

Jede Anwendung implementiert einen Watchdog, der die Verfügbarkeit der Anwendung überprüft. Der Watchdog ist ein Timer, welcher in regelmäßigen Abständen eine Prüfung durchführt. Der Watchdog-Timer wird dabei als Spring-Bean konfiguriert, so dass keine eigene Timer-Implementierung notwendig ist. Weitere Details dazu und eine Beispielkonfiguration sind im Konzept [ÜberwachungKonfigKonzept] enthalten.

4.3. Die direkte Verwendung des Spring-Frameworks

Neben der Konfiguration der Beans enthält eine Anwendung kaum Abhängigkeiten von Spring: Da die benötigten Objekte per Dependency Injection konfiguriert werden, müssen diese nach ihrer Konfiguration lediglich verwendet werden.

In einigen Fällen existieren jedoch weitere Abhängigkeiten von Spring:

Für die Bereitstellung von JMX-Beans wird der Spring-MbeanExporter Mechanismus verwendet (siehe [ÜberwachungKonfigKonzept]).

Bei HttpInvoker Server-Beans muss explizit eine Bean des eigentlichen Anwendungskontextes aufgerufen werden (siehe [ServiceDetailkonzept] und Kapitel 4.2.5).

Beans werden per Namen aus dem Anwendungskontext ausgelesen.

Beans werden nicht per Dependency Injection, sondern über statische Methoden bereitgestellt. Hierüber entstehen Abhängigkeiten zur Instanziierungsreihenfolge der Beans durch Spring.

Für die noch nicht in anderen Konzepten besprochenen (letzten beiden) Themen werden im Folgenden Vorgaben aufgestellt.

4.3.1 Keine Beans per Namen auslesen

Über den Anwendungskontext könnten Beans explizit per Namen ausgelesen werden. Dies ist mit einer Ausnahme verboten: Die Namen von Beans sollen nicht im Anwendungscode verwendet werden. Die Ausnahme gilt für den Zugriff von einem Anwendungskontext auf einen anderen (in Zusammenhang mit dem DispatcherServlet). In diesem Fall ist ein explizites Auslesen nicht zu vermeiden. Auszulesen ist in diesem Fall keine AWK-Komponente, sondern eine weitere Schnittstellen-Bean, welche nur für diesen Zweck verwendet wird (siehe Kapitel 4.2.5).

Auch wenn eine Bean statisch zugreifbar gemacht werden soll (siehe Kapitel 4.3.2), soll sie nicht per Namen aus dem Kontext gelesen werden. Dies ist nicht notwendig: Als Beispiel betrachte man die Klassen `ProtokollHolder` und `JpaTemplateHolder` des Vorlage-Registers.

4.3.2 Statischer Zugriff auf Bean-Instanzen nur in Sonderfällen

Üblicherweise werden Spring-Beans per Dependency-Injection an die Beans weitergegeben, welche sie verwenden. In einigen Fällen muss eine Bean jedoch von sehr vielen Klassen verwendet werden: Die entsprechende Bean müsste in sämtlichen Komponenten-Beans injiziert und von dort in das komplette Objektgeflecht der Komponente (unter anderem an sämtliche Anwendungsfallklassen) weitergeleitet werden.

In einem solchen Fall reduziert es die Komplexität der Anwendung, wenn auf die Konfiguration und Weiterleitung verzichtet wird, indem die Bean-Instanz über eine statische Methode zugreifbar gemacht wird. Dies entspricht nicht den Konventionen des Spring-Frameworks und ist nur in Ausnahmefällen erlaubt.

Die entsprechenden Hilfsklassen werden als *Holder* bezeichnet. Sie besitzen eine statisches Attribut mit statischen `get`- und `set`-Methoden, die per Spring mit dem zur Verfügung zu stellenden Bean initialisiert werden.

4.4. Aspektorientierte Programmierung in Spring

Es ist möglich, für Spring-Beans Funktionalität in Form von Aspekten zu definieren. Ihr Einsatz kann über „PointCuts“ konfiguriert werden. Pointcuts definieren (etwa über reguläre Ausdrücke) Klassen und Methoden, welche um den Aspekt erweitert werden.

Zu intensive Nutzung kann leicht zu einem schwer durchschaubaren Programmfluss führen. In folgenden Bereichen soll AOP genutzt werden

- für die Steuerung von Transaktionen,
- für die Überwachung,
- für die Berechtigungsprüfung.

- Nicht benutzt werden soll AOP
  - für die Fehlerbehandlung.

- Die Verwendung von AOP für andere Bereiche ist nur in begründeten Ausnahmefällen erlaubt.

4.4.1 AOP für Transaktionssteuerung verwenden

Für die Steuerung von Transaktionen ist Spring-AOP einzusetzen: Es ist die Standard Spring-Transaktions-Advice zu verwenden. Ihre Verwendung wird in den Hibernate Nutzungsvorgaben (siehe [DatenzugriffDetailkonzept]) beschrieben. Zusammengefasst gilt:

- Instrumentiert werden alle Methoden der Anwendungs-Schnittstellen.

- Für jeden Request wird eine Transaktion gestartet.

- Falls keine Ausnahme auftritt, wird die Transaktion fortgeschrieben, sonst zurückgerollt.

4.4.2 AOP für Berechtigungsprüfungen verwenden

Die Berechtigungsprüfung wird über AOP mit den von der T-Komponente „Sicherheit“ angebotenen Annotationen umgesetzt (siehe [SicherheitNutzerdok]).

4.4.3 AOP nicht für das Logging von Exceptions verwenden

Sämtliche im System geworfenen und nicht behandelten Ausnahmen müssen inklusive ihrer Stacktraces geloggt werden. Geloggt wird dies in den Methoden der Schnittstellen-Beans. Hierfür soll Spring-AOP nicht verwendet werden.

Es wäre z.B. bei HttpInvoker-Schnittstellen nicht sinnvoll, über AOP die Exceptions der Schnittstellen-Bean-Methoden zu loggen. Es würden nicht alle relevanten Informationen geloggt werden:

Eine HttpInvoker-Bean fängt alle Ausnahmen und wirft statt der erhaltenen Exceptions eigene Schnittstellen-Exceptions, welche den Stacktrace der erhaltenen Exception nicht enthalten.

5. Quellenverzeichnis

[DatenzugriffDetailkonzept]

Detailkonzept Komponente Datenzugriff
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Datenzugriff.pdf.

[IsyFactReferenzarchitektur]

IsyFact – Referenzarchitektur
00_Allgemein\IsyFact-Referenzarchitektur.pdf.

[LoggingKonzept]

Konzept Logging
20_Bausteine\Logging\Konzept_Logging.pdf.

[NutzungsvorgabenLogging]

Nutzungsvorgaben Logging
20_Bausteine\Logging\Nutzungsvorgaben_Logging.pdf.

[ServiceDetailkonzept]

Detailkonzept Komponente Service
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Service.pdf.

[SicherheitNutzerdok]

Nutzerdokumentation Sicherheit
20_Bausteine\Sicherheitskomponente\Nutzerdokumentation_Sicherheit.pdf.

[SpringLDAPNutzungskonzept]

Nutzungskonzept Spring-LDAP
20_Bausteine\LDAP-Zugriffe\Nutzungskonzept_Spring_LDAP.pdf.

[ÜberwachungKonfigKonzept]

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

[VorlageAnwendung]

Beispielimplementierung „Vorlage-Anwendung“
Wird auf Anfrage bereitgestellt.

[WebGUIDetailkonzept]

Detailkonzept Komponente Web-GUI
10_Blaupausen/technische_Architektur/Detailkonzept_Komponente_Web_GUI.pdf.

6. Abbildungsverzeichnis

Abbildung 1: Referenzarchitektur eines IT-Systems 6

Abbildung 2: Bestandteile des Anwendungskerns 7

Abbildung 3: Schichten und Teilanwendungen 9

Abbildung 4: Spring-Beans des Vorlage-Registers 12

Abbildung 5: Komponenten: Ihre Interfaces und ihre als Beans konfigurierten Implementierungen.....13

Abbildung 6: Aufruf-Weiterleitung eines HttpInvoker Aufrufs.....17