Bundesverwaltungsamt

IsyFact-Standard

Grundlagen der internen Servicekommunikation

Version 2.18
01.03.2017

IsyFact Standards

Ansprechpartner:

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

Dokumentinformationen

Dokumenten-ID:	Grundlagen_interne_Servicekommunikation.docx

Inhaltsverzeichnis

1. Einleitung

Die Anwendungen innerhalb einer Plattform kommunizieren in Form von Serviceanfragen miteinander. Das vorliegende Dokument definiert grundlegende Konzepte der Servicekommunikation innerhalb dieser Plattform.

Dazu wird beschrieben, auf welche Art und Weise die Anwendungssysteme der IsyFact miteinander kommunizieren und welche Anforderungen hierfür existieren. Nicht Bestandteil dieses Dokuments sind die IT-Systeme des Service-Gateways. D. h. die Anforderungen und die Architektur der Service-Provider und der Service-Consumer, um Services extern zur Verfügung zu stellen, werden nicht beschrieben. Diese Punkte sind Inhalt des Dokuments [ServiceGatewaySystementwurf].

Das vorliegende Dokument enthält folgende Inhalte:

- Abschnitt 2: Es werden die Grundlagen bzgl. der Kommunikation innerhalb der IsyFact beschrieben. Hierbei gehen wir insbesondere auf technische Aspekte der Kommunikation ein.
- Abschnitt 3: Services müssen versioniert werden. In diesem Abschnitt wird kurz auf die Motivation der Versionierung und die Realisierung der Versionierung eingegangen.
- Abschnitt 4: In diesem Abschnitt beschreiben wir kurz die Anforderungen bzgl. der Verfügbarkeit von Services.

Das Thema Testen wird im vorliegenden Dokument nicht beschrieben.

2. Grundlagen

In diesem Abschnitt beschreiben wir die Grundlagen der Kommunikation innerhalb der Plattform. Zunächst wird in Abschnitt 2.1 die Unterscheidung von internen und externen Services geschärft und zentrale Begriffe eingeführt. Danach wird in Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.** vorgestellt, wie synchrones und asynchrones Verhalten von Services abgebildet wird. In Abschnitt 0 gehen wir dann auf die technische Umsetzung von Services ein.

2.1. Interne und externe Servicekommunikation

Gemäß der Referenzarchitektur (siehe [IsyFactReferenzarchitektur]) kommunizieren Anwendungssysteme auf Basis von Services. Das bedeutet, dass die Anwendungssysteme ihre Fachlichkeit anderen Anwendungssystemen in Form von Services anbieten. Hierbei ist zu unterscheiden, ob ausschließlich Anwendungssysteme innerhalb der Plattform miteinander kommunizieren oder ob die Kommunikation auch Anwendungssysteme einschließt, die außerhalb der Plattform liegen. (siehe Abbildung 1)

Kommunikation zwischen internen Anwendungssystemen: Innerhalb der Plattform basiert die Kommunikation auf Basis des Protokolls HTTP-Invoker: Das aufgerufene Anwendungssystem exportiert seine Fachlichkeit in Form einer HTTP-Invoker-Schnittstelle und das aufrufende Anwendungssystem ruft diese HTTP-Invoker-Schnittstelle direkt auf. Zu beachten ist, dass kein System zur Vermittlung der ausgetauschten Nachrichten wie beispielsweise ein ESB (Enterprise Service Bus) verwendet wird.

Kommunikation mit externen Anwendungssystemen: Die Kommunikation mit externen Anwendungssystemen basiert auf Web-Services. Hierbei muss man unterscheiden, ob ein externes Anwendungssystem ein internes Anwendungssystem aufrufen möchte (1) oder umgekehrt (2):

(1) Durch die Plattform wird externen Anwendungssystemen die Fachlichkeit eines internen Anwendungssystems in Form eines Web-Services zur Verfügung gestellt. Hierbei definiert das interne Anwendungssystem selbst keinen Web-Service. Vielmehr definiert das interne Anwendungssystem wie bei der internen Kommunikation lediglich eine HTTP-Invoker-Schnittstelle. Diese HTTP-Invoker-Schnittstelle wird dann durch ein eigenständiges IT-System als Web-Services exportiert. Dieses IT-System wird als Service-Provider bezeichnet. Für jede HTTP-Invoker-Schnittstelle, die als Web-Services exportiert werden soll, muss ein eigener Service-Provider definiert werden.

(2) Möchte ein internes Anwendungssystem ein externes Anwendungssystem aufrufen, so muss das externe Anwendungssystem einen Web-Service definieren. Ähnlich wie bei

Punkt (1) ruft das interne Anwendungssystem diesen Web-Services nicht direkt auf. Es ruft ein eigenständiges IT-System auf, welches den Web-Service des externen Anwendungssystems als HTTP-Invoker-Schnittstelle in die Anwendungslandschaft importiert. Dieses IT-System wird als Service-Consumer bezeichnet. Das interne Anwendungssystem ruft dann lediglich die HTTP-Invoker-Schnittstelle des Service-Consumers auf. Für das interne Anwendungssystem ist dieser HTTP-Invoker-Aufruf nicht von einem HTTP-Invoker-Aufruf zu einem anderen internen Anwendungssystems zu unterscheiden. Für jeden Web-Service, der in die Anwendungslandschaft importiert werden soll, muss ein eigener Service-Consumer definiert werden.

Die Gesamtheit aller Service-Provider und Service-Consumer wird als Service-Gateway bezeichnet. Der Service-Gateway stellt somit die zentrale Schnittstelle der IsyFact zur Außenwelt dar.

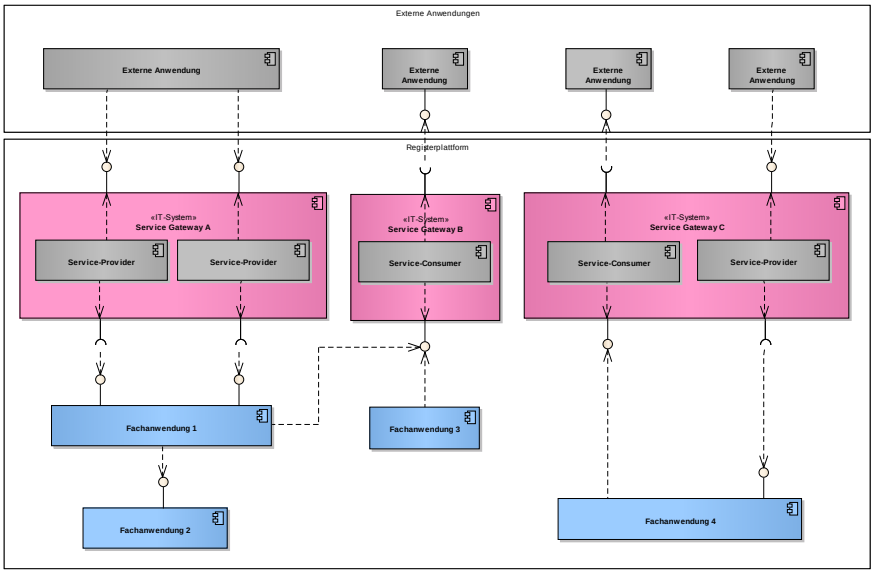

Abbildung 1: Überblick Kommunikation in der IsyFact

Wird ein Service von einer externen Anwendung angeboten, wird er als „externer Service“ bezeichnet. Ein Service-Consumer macht diesen „externen Service“ als „inneren Service“ der Plattform verfügbar. Wird ein Service von einer internen Anwendung angeboten, so ist das ebenfalls ein „Innerer Service“. Wenn ein Service-Provider diesen „Inneren Service“ einer Anwendung außerhalb der Plattform zugänglich macht, ist dies ein „Äußerer Service“ der Plattform. Dies ist in Abbildung 2 dargestellt. Die Unterscheidung zwischen „Innere“ und „Äußere“ ist analog für die Begriffe „Request“ und „Response“ zu verwenden.

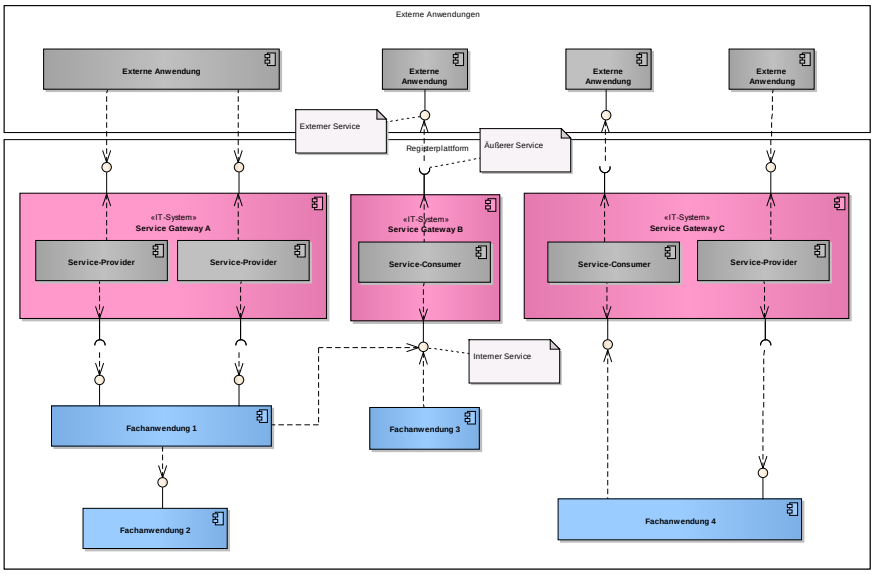

Abbildung 2: Externe, äußere und innere Services

2.2. Synchrone Kommunikation

Die interne Kommunikation erfolgt grundsätzlich synchron, d. h. Aufrufe werden direkt an den Empfänger gesendet und bearbeitet. Synchrone Service-Aufrufe sind in der Regel eine vergleichsweise zeitintensive Operation. Häufig ist es sinnvoll, Service-Aufrufe nach Möglichkeit einzusparen. Das Sparen von Aufrufen kann jedoch auch Nachteile in Bezug auf Wartbarkeit bedeuten, wenn beispielsweise Redundanzen oder komplexe Caches implementiert werden müssen. Die Abwägung darüber trifft der Chefdesigner während des Systementwurfs.

2.3. Asynchrone Kommunikation

Falls eine länger dauernde Verarbeitung eine direkte Rückmeldung unmöglich macht, erfolgt die Rückmeldung über ein Callback. Zur Abfederung von Lastspitzen kann ein Spooling über eine Queue eingesetzt werden.

2.3.1 Queueing

Hierfür wird in der Fachanwendung für jedes asynchron aufzurufende Anwendungssystem eine eigene Queue angelegt. Jede Queue enthält automatisch eine Standard-Exception-Queue, die ausschließlich Oracle-intern verwendet wird. Die Fachanwendung erzeugt Nachrichten, die anschließend mit JAXB in XML umgewandelt und mit Spring-JMS in die passende Queue geschrieben werden. Fremdsysteme können die Benachrichtigungen aus der Queue auslesen und verarbeiten.

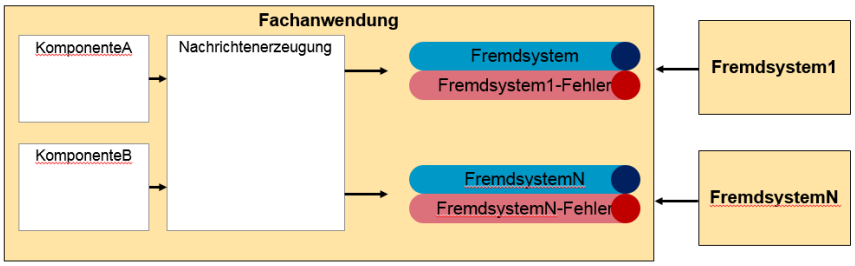

Abbildung 3: Schematische Darstellung der Queues

Die Fremdsysteme erhalten explizite Datenbank-Berechtigungen auf ihre Queue zum Dequeuen. Somit ist sichergestellt, dass kein System auf fremde Queues zugreifen kann und dass keine Benachrichtigungen erneut eingestellt werden können. Da die Queue als Bestandteil der Fachanwendung angesehen wird und die Daten somit in der Datenhoheit der Fachanwendung liegen, sind Datenbank-Benutzer und -Rechte durch das Anwendungssystem und nicht durch das die Queue nutzende Fremdsystem zu vergeben.

Der Queue-Name erhält eine einstellige Versionsnummer. Damit ist es möglich bei inkompatiblen Schnittstellenänderungen mehrere Versionen der Benachrichtigungsschnittstelle anzubieten. Ändert sich die Signatur der Benachrichtigungsschnittstelle, dann wird eine neue Queue in der bestehenden Queue-Tabelle angelegt und der Versionsanteil hochgezählt. Über die betriebliche Konfiguration kann gesteuert werden, in welche Queue-Version die Daten en-/dequeued werden. Dieses Vorgehen hat den Vorteil, dass die Fachanwendung von den Fremdsystemen stark entkoppelt ist. Die Fachanwendung kann für einen gewissen Zeitraum immer sowohl die alte Queue-Version als auch die neue Queue-Version bedienen – es wird jedoch immer nur eine Queue befüllt.

Die Umsetzung erfolgt mit dem Spring JMS Integration Framework. Zusätzlich wird die Register-Factory-Bibliothek *plis-jmsutil* zur Überwachung verwendet. Die Benachrichtigungen werden ohne *expiration time* enqueued. D.h. eine automatische Löschung von alten, nicht abgeholten Benachrichtigungen wird nicht durchgeführt.

2.3.2 Vorgaben für Fremdsysteme

Die Fremdsysteme dürfen sich nicht auf die Reihenfolge der Nachrichten in der Queue verlassen. Anhand des Zeitstempels (*JMSTimestamp*) kann jedoch eine Reihenfolge ermittelt werden. Das Feld *JMSTimestamp* wird beim Schreiben in die Queue automatisch von der Datenbank gesetzt.

Bei fachlichen Fehlern muss eine entsprechende Benachrichtigung (z.B. Logging) vorgenommen werden; Bei technischen Fehlern muss eine erneute Verarbeitung ermöglicht werden z.B. indem die Nachricht in eine eigene „Fehlerqueue“ enqueued wird. Die Oracle-AQ-Exception-Queue ist nicht zu verwenden.

Nachrichten enthalten als JMS-Properties den Aufrufkontext mit den Rollen, jedoch ohne die Korrelations-ID. Der Aufrufkontext muss von dem

aufrufenden System geprüft werden. Das Feld *JMSCorrelationId* wird mit der Korrelations-ID befüllt.

2.3.3 Nachrichteninhalt

Eine JMS-Message besteht aus (siehe Abbildung 4: Aufbau der JMS-Message)

- dem Message Header
- den Message Properties
- dem Message Body

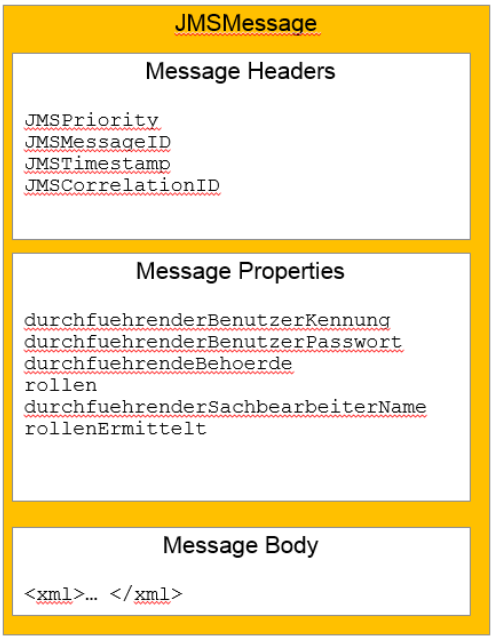

Abbildung 4: Aufbau der JMS-Message

Über die Message-Properties wird der Aufrufkontext abgebildet, da es sich um die Kommunikation zwischen zwei Systemen handelt und wie eine Außenschnittstelle zu betrachten ist.

Der Message-Body enthält die Benachrichtigung im XML-Format. Für jedes Fremdsystem wird eine gesonderte XSD-Datenbeschreibung bereitgestellt, so dass jedes System eigene Nachrichten in seinem Format erhalten kann. Damit haben bspw. Änderungen der Nachrichten von Fremdsystem1 keine Auswirkungen auf die Nachrichten von Fremdsystem2.

2.4. Technologie

Interne Anwendungssysteme kommunizieren miteinander über das durch das Spring-Framework definierte HTTP-Invoker-Protokoll. Das heißt, interne Anwendungssysteme stellen ihre Services innerhalb der IsyFact über eine HTTP-Invoker-Schnittstelle bereit. Da HTTP-Invoker auf serialisierten Java-Objekten basiert, können innerhalb der IsyFact ausschließlich Java-basierte Anwendungssysteme miteinander kommunizieren.[1] Um eine möglichst lose

[1] Fachanwendungen sind entsprechend der Referenzarchitektur alle auf Basis von Java zu erstellen. Somit ist sichergestellt, dass alle Anwendungssysteme über HTTP-

Kopplung der Anwendungssysteme zu erreichen, wurden folgende Festlegungen getroffen:

- **Es werden keine Komponenten des Anwendungskerns extern verfügbar gemacht.** Es wird stets eine explizite Schnittstellen-Bean (RemoteBean-Schnittstelle) als HTTP-Invoker-Schnittstelle implementiert.
- **Es werden keine Datenbank-Entitäten verfügbar gemacht**. Jegliche über HTTP-Invoker-Aufrufe zu transportierende Objekte sind Transportobjekte. Diese Transportobjekte sind im Client-HttpInvoker-Wrapper bzw. in der Server-HttpInvoker-Bean zu befüllen (siehe Abbildung 5).
- **Es werden keine Exceptions des Anwendungskerns geworfen.** Stattdessen werden möglichst grobe Exceptions geworfen, welche nur von der Schnittstelle verwendet werden.

2.4.1 Definition der Schnittstelle

Die durch ein Anwendungssystem definierte HTTP-Invoker-Schnittstelle, d. h. die RemoteBean-Schnittstelle inklusive aller direkt und indirekt verwendeten Transportobjekte und Exceptions ist Teil des Anwendungssystems. Das bedeutet, dass die Quelldateien der RemoteBean-Schnittstelle, der Transportobjekte und der Exceptions im Anwendungssystem definiert werden. Das Anwendungssystem wird daher auch als definierendes Anwendungssystem der HTTP-Invoker-Schnittstelle bezeichnet. Damit andere Anwendungssysteme die HTTP-Invoker-Schnittstelle aufrufen können, muss das definierende Anwendungssystem die Schnittstelle in Form einer Bibliothek (JAR-Datei) zur Verfügung stellen. Diese JAR-Datei muss dann von den aufrufenden Anwendungssystemen eingebunden werden.

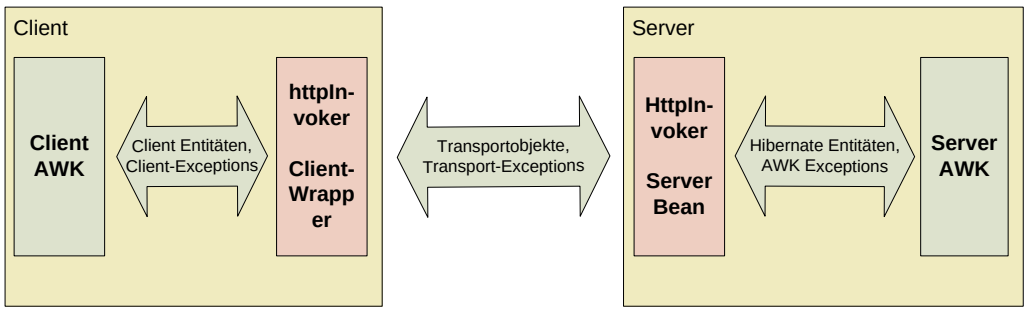

Abbildung 5: Kapselung der Aufrufe von HttpInvoker Beans

2.4.2 Parameter der Schnittstelle

Jede Methode der RemoteBean-Schnittstelle muss als ersten Parameter ein Objekt der Klasse `AufrufKontextTo` bzw. `ClientAufrufKontextTo` verwenden. Dieser Parameter dient dazu, Meta-Informationen zum jeweiligen Aufruf zu übergeben. Daneben enthält die Schnittstelle natürlich noch weitere, fachliche Parameter, die frei definiert werden können.

Invoker miteinander kommunizieren können. Für Fremdsysteme sind andere Kommunikationsarten zulässig.

Die Verwendung von Parametern in einer Schnittstelle ist im folgenden Beispiel dargestellt.

```
public NachrichtenlisteTo holeNachrichten(

         AufrufKontextTo kontext,

         NachrichtenanfrageTo anfrage)

throws BusinessToException, TechnicalToException;
```

Im Folgenden werden die beiden Klassen `AufrufKontextTo` und `ClientAufrufKontextTo` näher beschrieben.

AufrufKontextTo: Die Klasse `AufrufKontextTo` wird für HTTP-Invoker-Schnittstellen verwendet, die durch Fachanwendungen definiert werden und nicht durch Service-Consumer. Die Klasse kapselt die Informationen, mit denen die Fachanwendung aufgerufen wurde:

- **Behörde:** Das Behördenkennzeichen der aufrufenden Behörde
- **Kennung:** Die Kennung des aufrufenden Benutzers oder des aufrufenden Fremdprogramms
- **Kennwort:** Das Passwort des aufrufenden Benutzers oder des aufrufenden Fremdprogramms
- **Rollen:** Die Rollen des aufrufenden Benutzers oder des aufrufenden Fremdprogramms
- **Correlation-ID:** Die ID, um den Service-Aufruf eindeutig zu identifizieren

ClientAufrufKontextTo: Die Klasse `ClientAufrufKontextTo` wird für HTTP-Invoker-Schnittstellen verwendet, die durch Service-Consumer definiert werden. Im Gegensatz zu `AufrufKontextTo` kapselt diese Klasse die Informationen, um sich bei einem externen Service zu authentifizieren und zu autorisieren:

- **Kennung:** Die Kennung mit der der externe Service aufgerufen wird
- **Kennwort:** Das Passwort mit der der externe Service aufgerufen wird
- **Zertifikat:** Das Zertifikat, um sich beim externen Service zu authentifizieren
- **Zertifikat-Kennwort:** Das Passwort des Zertifikats für die Authentifizierung

Sowohl `AufrufKontextTo` als auch `ClientAufrufKontextTo` sind in der Bibliothek `plis-serviceutil` definiert. Das heißt zur vollständigen Definition der HTTP-Invoker-Schnittstelle ist immer auch diese Bibliothek einzubinden.

2.4.3 Zugriff auf Querschnittssysteme

Gemäß der Referenzarchitektur (siehe [IsyFactReferenzarchitektur]) ist der Zugriff von Service-Gateways auf Querschnittssysteme erlaubt.

3. Versionierung

In diesem Abschnitt gehen wir auf die Versionierung von Services ein. In Abschnitt 3.1 geben wir die Motivation für die Versionierung an und in den Abschnitten 3.2 und 3.3 gehen wir auf die Realisierung der Versionierung in Java ein. In Abschnitt 3.4 führen wir einige Vorgaben für den Parallelbetrieb von Service-Versionen auf und in Abschnitt 3.5 gehen wir schließlich kurz auf die Grenzen der Versionierung ein.

3.1. Motivation

Die Notwendigkeit Services in mehreren Versionen anbieten zu können, ist bedingt durch die Vielzahl an Service-Nutzern, die bei Änderung an einem Service nicht alle zeitgleich auf die neue Version eines Service umschalten können. Daher ist es notwendig, dass in einem Übergangszeitraum mehrere Versionen eines Service parallel betrieben werden können.

Die Versionierung wird auf der Ebene von Services, nicht Service-Operationen ausgeführt, da diese Ebene von ihrer Granularität zu den üblichen fachlichen Änderungen passt.[2] Es kann vorkommen, dass in *einem* Systemrelease neue Versionen von *mehreren* Services ausgeliefert werden.

3.2. Architektur

In der Fachanwendung wird pro Service-Version eine eigne Service-Schnittstelle angeboten. Die Services verwenden alle denselben Anwendungskern. Die für die Versionierung notwendigen Transformationen sollen in der Service-Schicht der Anwendung durchgeführt werden, z.B. das Einfügen eines Standardwerts für neu hinzugefügte Attribute. In komplexen Fällen kann es auch notwendig sein, den Anwendungskern zu erweitern und die Versionierung dort zu behandeln. Die Entscheidung dafür obliegt dem Chefarchitekten.

Externe Services werden durch Service Gateways bereitgestellt. Die Versionierung eines Services muss also auch auf Ebene des Service Gateways durchgeführt werden. Ein Service Gateway ist ein rein technischer Protokoll-Wandler, der z. B. SOAP auf HttpInvoker konvertiert. Im Service Gateway erfolgt daher immer nur ein einfaches Mapping auf der Service-Schnittstelle der angebundenen Fachanwendungen. Der Ausgleich der Versionsunterschiede soll ausschließlich in der Fachanwendung und nicht im Service Gateway erfolgen. Es ist möglich pro Service Version ein eigenes Service Gateway zu erstellen (siehe Abbildung 6).

[2] Für die HTTP-Invoker-Schnittstelle heißt das, dass die komplette RemoteBean-Schnittstelle versioniert wird und nicht die einzelnen Methoden der RemoteBean-Schnittstelle.

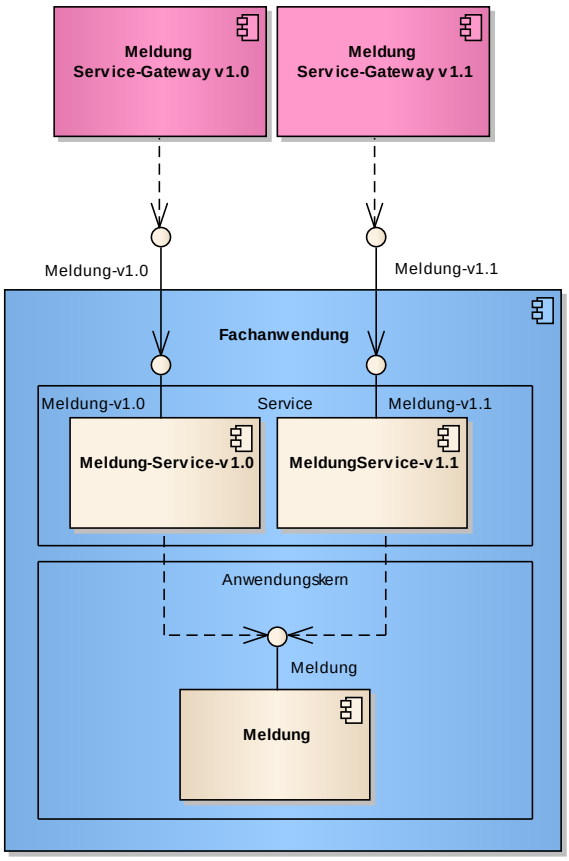

Abbildung 6: Architektur versionierter Services

3.3. Vorgehen

Das Vorgehen zur Versionierung soll an folgendem Beispiel illustriert werden:

3.3.1 Einfachster Fall: Kompatible Erweiterung eines Services

Eine Fachanwendung stellt einen Service bereit, mit dem Personendaten gemeldet werden können. Parameter dieser Meldung sind Vor- und Nachname sowie das Geburtsdatum. Dazu gibt es einen Meldungs-Service in der Version 1.0. Dieser wird in der Service-Schicht der Fachanwendung implementiert. Ab einem Stichtag soll zusätzlich noch das Geschlecht gemeldet werden. Im bisherigen Datenbestand wird dieses neue Attribut auf den Wert „unbekannt“ gesetzt. Der bestehende Service wird um dieses Attribut erweitert und erhält die Versionsnummer 1.1. Anwendungskern und Datenzugriffsschicht müssen ebenfalls erweitert werden. Aus Gründen der Rückwärtskompatibilität soll aber weiterhin die Version 1.0 des Service angeboten werden. Dazu wird ein neuer Service innerhalb der Serviceschicht implementiert, der die Meldung entgegennimmt, das fehlende Attribut mit dem Wert „unbekannt“ ergänzt und dann den Anwendungskern aufruft.

Werden die beiden Services durch ein Service Gateway nach außen verfügbar gemacht, existieren dort zwei parallele Mappings auf die jeweiligen Services der Fachanwendung. Innerhalb des Service Gateways

existiert keine Fachlogik, d. h. die Abbildung von Version 1.0 auf 1.1 findet erst in der Fachanwendung statt.

3.3.2 Komplexerer Fall: Inkompatible Veränderung eines Services

In einem komplexeren Fall kann es passieren, dass die Service-Schnittstelle einer Anwendung komplett umgestaltet wird, so dass die Aufrufe nicht mehr einfach aufeinander abgebildet werden können. Wird in so einem Fall ein neuer Service eingeführt, während der alte Service noch verfügbar bleiben muss, müssen die inkompatiblen Verarbeitungslogiken im Anwendungskern parallel erhalten bleiben. Auch hier enthält der Service Gateway keine Fachlogik.

Bei der Implementierung ist zu beachten, dass die Versionsnummer aus dem Packagenamen auch in die Implementierung übernommen wird.

3.4. Parallelbetrieb

Es wird empfohlen, so wenige Service-Versionen wie möglich parallel produktiv zu betreiben. Die Motivation zum Parallelbetrieb verschiedener Versionen ist lediglich dem Umstand geschuldet, den Aufrufern den nötigen Zeitrahmen zum Umschalten auf die jeweils neue Version zu geben. Es wird empfohlen alte Versionen nach 6 Monaten abzuschalten. Dies ist organisatorisch zu lösen.

3.5. Grenzen

Eine Versionierung ist nur dann sinnvoll, wenn kleine Änderungen an der Schnittstelle zwischen den Versionen auftreten. Für den Fall, dass sich die Schnittstelle sowohl syntaktisch als auch semantisch grundlegend ändert, würde eine Versionierung der Schnittstelle im schlimmsten Falle für jede Schnittstellen-Version einen eigenen Anwendungskern erfordern. Die Kosten hierfür stehen in den meisten Fällen nicht im Verhältnis zum Nutzen und eine Versionierung ist in solchen Fällen zu vermeiden.

4. Verfügbarkeit

In diesem Abschnitt definieren wir kurz die Anforderungen bzgl. der Verfügbarkeit von Services in der Plattform (Abschnitt 4.1), geben dann einige Ursachen für die Nichtverfügbarkeit von Service an (Abschnitt 4.2) und listen schließlich einige Maßnahmen auf, um die definierten Anforderungen bzgl. Verfügbarkeit zu erreichen (Abschnitt 4.3).

4.1. Anforderungen

Folgende Anforderungen existieren bzgl. der Verfügbarkeit der Services:

Hohe Verfügbarkeit: Die Fachanwendungen der Plattform müssen eine hohe Verfügbarkeit aufweisen.

Die Berechnung der Verfügbarkeit einer Anwendung ist komplex. In die Berechnung fließen unter anderem betriebliche Aspekte wie Hardwareverfügbarkeit ein, während Wartungsfenster herausgerechnet werden. Weiter könnte man Verfügbarkeit auf der Ebene von angebotenen Services und nicht von IT-Systemen betrachten. Von der Seite der Software ist zu beachten, dass sich in einer serviceorientierten Systemlandschaft die Ausfallwahrscheinlichkeiten multiplizieren, wenn Systeme einander Aufrufen. Dies wird im folgenden vereinfachten Szenario gezeigt.

Für das Szenario gehen wir im Folgenden davon aus, dass die Fachanwendungen gemeinsam eine Gesamtverfügbarkeit von 98% aufweisen sollen. Hierbei ist zu beachten, dass Fachanwendungen in der Regel andere Anwendungen und Querschnittssysteme aufrufen, um Anfragen zu beantworten. Die Gesamtverfügbarkeit sinkt dadurch ab, da zur erfolgreichen Bearbeitung einer Anfrage alle Systeme zeitgleich verfügbar sein müssen. Im Szenario wird für alle Systeme ein Richtwert für die Verfügbarkeit von 99,7% angenommen. Tabelle 1 zeigt eine Beispiel-Rechnung (die Gesamtverfügbarkeit ergibt sich aus dem Produkt der Einzelverfügbarkeiten). Durch eine Verfügbarkeit von 99,7% pro System kann im Beispiel also eine Gesamtverfügbarkeit von über 98% erreicht werden.

Eine Berechnung der Gesamtverfügbarkeit nach dem Schema von Tabelle 1 muss für jede Fachanwendung einzeln durchgeführt werden. Dabei müssen die berechneten oder gemessenen Verfügbarkeiten aller Systeme zugrunde gelegt werden, die die Fachanwendung aufruft.

Schnelles Antwortzeitverhalten im Fehlerfall: Die Nichtverfügbarkeit von Services ist ein Ausnahmefall, auf den angemessen reagiert werden muss: Sollte ein Service nicht verfügbar sein, ist es wichtig, dass die aufrufende Anwendung zügig eine Fehlermeldung erhält. Speziell bei Online-Anwendungen ist der schnelle Erhalt einer Fehlermeldung notwendig. Der Nutzer soll auch im Fehlerfall eine gewohnt schnelle Antwort vom System erhalten. Die genaue Definition des Zeitrahmens, in dem die Fehlermeldung über die Nichtverfügbarkeit beim Aufrufer

eintreffen muss, ist anwendungsspezifisch. Die Definition ist dementsprechend durch die jeweiligen Aufrufer vorzunehmen.

System	Verfügbarkeit
Service-Gateway Cluster	99,7%
Fachanwendung 1	99,7%
Fachanwendung 2	99,7%
Fachanwendung 3	99,7%
Querschnittskomponente 1	99,7%
Oracle Cluster	99,7%
Gesamtverfügbarkeit	**98,21%**

Tabelle 1: Beispiel Verfügbarkeits-Rechnung
(Gesamtverfügbarkeit ist das Produkt der Einzelverfügbarkeiten)

4.2. Ursachen für Nichtverfügbarkeit

Die möglichen Ursachen für Nichtverfügbarkeit sind unter anderem:

Deployment einer Anwendung: Bei einem Re-Deployment einer Anwendung kommt es zu einer geplanten Auszeit.

Überlastung während Lastspitzen: Im Tagesverlauf variiert die Last, die ein System verarbeiten muss. Manche Systeme antworten bei Lastspitzen zu langsam.

Ausfall von Hard- oder Software: Auf einem Knoten eines Anwendungsclusters ist eine Störung durch einen Hardware- oder Softwareausfall aufgetreten. Der nicht funktionierende Knoten ist dadurch temporär nicht verfügbar, wodurch die verbleibenden Knoten die Last des ausgefallenen Knotens mitverarbeiten müssen.

Umschaltzeit bei Hard- oder Softwareausfall: Bei Ausfall von Hard- oder Software sorgt ein Loadbalancer dafür, dass alle Anfragen nur an die noch funktionierenden Knoten weitergeleitet werden. In dem kurzen Zeitraum, bis der Loadbalancer einen Server-Knoten als ausgefallen markiert („Umschaltzeit“), kommt es jedoch zur Nichtverfügbarkeit von Services. In diesem Zeitraum werden Anfragen nicht beantwortet die noch an den ausgefallenen Knoten geleitet werden.[3]

Batchläufe: Wenn lang laufende Batches in Fachanwendungen durchgeführt werden, dürfen in dieser Zeit keine Meldungen gemacht

[3] Die Regeln, nach denen der Loadbalancer entscheidet wann ein Server-Knoten nicht mehr verfügbar ist, können üblicher Weise konfiguriert werden. Beispielsweise kann ein Loadbalancer alle paar Sekunden per Script („Health-Check“) überprüfen, ob ein Server-Knoten noch verfügbar ist. Erst nach einer festgelegten Anzahl fehlgeschlagener fachlicher Anfragen und negativem Health-Check leitet dann der Loadbalancer keine Anfragen mehr an diesen Knoten. Unabhängig von der Konfiguration kann es trotz Loadbalancer und Anwendungscluster zu wenigen nicht beantworteten Anfragen und somit zu einer Nichtverfügbarkeit kommen.

werden. So werden Dateninkonsistenzen vermieden. Meldungsaufrufe sind in dieser Zeit nicht verfügbar und werden von der Fachanwendung nicht beantwortet.

Retries des Loadbalancers: Tritt ein Ausfall von Hard- oder Software auf (vgl. Ursache 2), bekommt der Loadbalancer beim Weiterleiten einer Anfrage an einen ausgefallenen Knoten ein Timeout. Loadbalancer können so konfiguriert werden, dass sie in diesem Fall die gleiche Anfrage an einen noch funktionierenden Knoten weiterleiten und nicht sofort eine Fehlermeldung an den Aufrufer zurückgeben. Für den Aufrufer hat der Service dadurch eine längere Antwortzeit. Der Aufrufer hat keine Möglichkeit dieses Timeout/Retry-Verhalten des Loadbalancers zu beeinflussen und auf seine Bedürfnisse anzupassen. Die lange Antwortzeit kann auf Seiten des Aufrufers leicht zu einem Timeout führen.

Verschlimmerung von Nichtverfügbarkeiten: Die aufrufende Anwendung reagiert nicht angemessen auf eine Nichtverfügbarkeit eines Service. Beispiele:

- Der Client versucht Retries, obwohl der Service-Aufruf aus fachlicher Sicht entfallen könnte (optionaler Aufruf).
- Die fachliche Verarbeitung wird nicht rechtzeitig abgebrochen, obwohl ein verpflichtender Service-Aufruf bereits fehlgeschlagen ist.
- Die Bearbeitung der Anfrage dauert bekanntermaßen beim Service-Anbieter sehr lange. Der Aufrufer hat einen sehr knappen Timeout gesetzt und schickt Aufrufwiederholungen. Dies verschlimmert die Antwortzeiten der Service-Aufrufe und führt eventuell zu Duplikaten beim Service-Anbieter.

Eine weitere bekannte Ursache für Nichtverfügbarkeit ist die Umgebungskonfiguration, Firewall-Verbindungen nach einer definierten Zeit automatisch zu schließen. Zustandsbehaftete Verbindungen wie sie bei LDAP- und Datenbank-Clients eingesetzt werden, sind von dieser Restriktion betroffen. Diese Clients müssen vorsehen, dass Sie eine von der Firewall geschlossene Verbindung erkennen und wieder neu aufbauen. Dieses Thema wird in den entsprechenden Nutzungskonzepten wie [DatenzugriffDetailkonzept] und [SpringLDAPNutzungskonzept] behandelt.

Die im vorliegenden Dokument beschriebene Service-Kommunikation über HTTP-Invoker setzt als Transportprotokoll durchgängig HTTP ein. HTTP ist ein zustandsloses Protokoll und baut bei jeder Anfrage eine neue Verbindung zwischen Client und Server auf. Der in HTTP 1.1 angebotene Mechanismus, mehrere Anfragen über eine TCP-Verbindung zu transportieren, wird für HTTP-Invoker-Aufrufe nicht eingesetzt.[4]

[4] Es ist zu beachten, dass dies nur für HTTP-Invoker-Aufrufe gilt. Für andere Aufrufe (z.B. Web-Service-Aufrufe) kann es beispielsweise sein, dass eine TCP-Verbindung wiederverwendet wird. In einem solchen Fall müssen die TCP-Verbindungen ähnlich wie die LDAP-Verbindungen vor ihrer Verwendung validiert werden.

4.3. Maßnahmen

In diesem Abschnitt beschreiben wir, welche Maßnahmen ergriffen werden können, um die in Abschnitt 4.1 aufgeführten Anforderungen an die Verfügbarkeit zu gewährleisten:

Anwendungscluster mit Loadbalancer: Die TI-Architektur der IsyFact setzt die hohen Verfügbarkeitsanforderungen durch Clustering der Applikations- und Datenbankserver um. Anwendungen werden redundant auf mehr als einem Server installiert. Kommt es zu einem Hard- oder Softwareausfall (siehe Abschnitt 4.2) auf einem Server-Knoten, so werden alle Anfragen von einem vorgeschalteten Loadbalancer auf einen anderen Server-Knoten umgeleitet. Durch die Redundanz wird die Verfügbarkeit von Services bei auftretenden Hard- oder Softwareausfällen erhöht. Trotzdem kann es auch hier noch zu Nichtverfügbarkeit kommen.

Knotenweises Deployment: In Abschnitt 4.2 wurde als Ursache für Nichtverfügbarkeit eine geplante Wartungsarbeit beschrieben. Im Clusterbetrieb besteht die Möglichkeit, diese Knoten für Knoten auszuführen. Bevor das Deployment auf einem Knoten ausgeführt wird, wird dem Loadbalancer mitgeteilt dass der Knoten nicht mehr verfügbar ist. Während des Deployments des Knotens verarbeiten die restlichen Knoten alle ankommenden Anfragen. Nach Abschluss des Deployments des Knotens wird dem Loadbalancer mitgeteilt, dass der Knoten wieder zur Verfügung steht. Dann kann das Deployment des nächsten Knotens nach dem gleichen Schema erfolgen. Dadurch können Services im Zeitraum von Wartungsarbeiten voll verfügbar gehalten werden. Dieser „Web-Off-Mechanismus“ wird in [DeploymentKonzept] im Detail beschreiben.

Time-To-Live: Ein Service-Aufruf ist nur für eine bestimmte Zeit gültig. Diese Zeitspanne wird als Time-To-Live (TTL) bezeichnet. Der Aufrufer definiert die TTL und legt so fest, wie lange er bei einem Aufruf auf eine Antwort wartet. Hierdurch wird eine schnelle Antwortzeit gewährleistet.

Aufrufwiederholung: In Abschnitt 4.2 wurde als eine Ursache dargestellt, dass die Retries des Loadbalancers zu einer Erhöhung der Antwortzeit führen können. Loadbalancer innerhalb der Plattform sind deshalb so zu konfigurieren, dass fehlgeschlagene Anfragen nicht an andere Knoten weitergeleitet werden. Eine Wiederholung von Aufrufen ist ausschließlich vom Aufrufer auszuführen. So kann der Aufrufer je nach Fachlichkeit entscheiden, bei welchen Anfragen Wiederholungen Sinn machen.

Achtung: Grundsätzlich sind Retries nur mit größter Vorsicht anzuwenden. Hierfür gibt es mehrere Gründe:

Ruft ein Client einen Service auf und erhält einen technischen Fehler, so kann der Client anhand des technischen Fehlers in der Regel nicht einwandfrei erkennen, ob seine Anfrage nicht doch auf dem Server

erfolgreich verarbeitet wurde. Beispielsweise kann durch einen Netzwerkausfall zwar die Netzwerkverbindung zum Server abgebrochen sein, das hindert den Server aber nicht daran, eine bereits in Verarbeitung befindliche Service-Anfrage weiterzuverarbeiten. In einem solchen Fall würde ein automatischer Retry dazu führen, dass ein und dieselbe Service-Anfrage zweimal ausgeführt würde. Dies kann bei nicht-idempotenten Service-Operationen fatale Auswirkungen haben (z. B. Löschen von falschen Daten).

Eine automatische Aufrufwiederholung kann im Falle einer echten Nichtverfügbarkeit zu einer erhöhten Netzwerklast führen und so die Nichtverfügbarkeit auch anderer Anwendungen in der Anwendungslandschaft erhöhen. Die Situation wird daher durch die Aufrufwiederholung deutlich verschlechtert.

Insbesondere bei einem Timeout eines TTL ist jedoch ein Retry mit großer Vorsicht zu genießen, da nicht klar ist, ob die Service-Anfrage nicht doch durch den Server bearbeitet wird. In einem solchen Fall führt eine Aufrufwiederholung zu einer erhöhten Last auf dem Server und kann im schlechtesten Fall zu einer echten Nichtverfügbarkeit des Services bzw. des kompletten Servers führen.

Empfehlung: In Anbetracht der potentiellen Probleme der Aufrufwiederholung und der Tatsache, dass eine Aufrufwiederholung nur für idempotente Service-Operationen überhaupt zulässig ist, sollte von einer automatischen Aufrufwiederholung als Maßnahme zur Erhöhung der Verfügbarkeit in der Regel abgesehen werden. Ausgenommen davon sind Aufrufe, bei denen nur Daten gelesen werden, z.B. die Aufrufe

- für Suchen im Suchverfahren,
- zur Abfrage von Verzeichnissen, wie Schlüsselverzeichnis, Benutzerverzeichnis oder Behördenverzeichnis

Hierfür soll grundsätzlich eine Wiederholung durchgeführt werden. Diese ist wie folgt zu konfigurieren:

- Pause zwischen den Retries: 500 ms
- Maximale Anzahl von Retries: 3
- Timeout für Anfragen: 2 s (10 s für Suchverfahren)

Die Parameter sollen betrieblich konfigurierbar gemacht werden.

Deaktivierung von Services: Aufgrund von Wartungsaktivitäten oder Batches (z.B. einer Datenmigration) in einer Fachanwendung kann es vorkommen, dass die Meldungskomponente einer Fachanwendung vorübergehend deaktiviert wird. Andere Services wie z. B. eine Auskunft können während dieser Zeit regulär ausgeführt werden. Während der Meldungs-Service deaktiviert ist, wird dem Aufrufer eine entsprechende Fehlermeldung zurückgesendet. Da die Anforderung

besteht, auch andere Services vorübergehend deaktivieren zu können, werden generell alle Services deaktivierbar gemacht.

5. Quellenverzeichnis

[DatenzugriffDetailkonzept]

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

[DeploymentKonzept]

Konzept Deployment für IsyFact-Anwendungen
30_Plattform\Konzept_Deployment.pdf.

[IsyFactReferenzarchitektur]

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

[ServiceGatewaySystementwurf]

Systemdokumentation Service-Gateway
20_Bausteine/Service-Gateway/Systemdokumentation_Service-Gateway.pdf.

[SpringLDAPNutzungskonzept]

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

6. Abkürzungsverzeichnis

ADB/JAXB	Axis Data Binding bzw. Java Architecture for XML Binding: Diese XML-Bindings werden im Service-Gateway eingesetzt um XML-Nachrichten in Java-Objekte umzuwandeln.
ESB	Enterprise Service Bus
TO	Transport-Objekt: Die Entitäten des Anwendungskerns werden nicht direkt nach Außen gegeben. Sie werden zuvor in Transport-Objekte gewandelt die dann an den Aufrufer gegeben werden.
TTL	Time-To-Live

7. Abbildungsverzeichnis

8. Tabellenverzeichnis