Bundesverwaltungsamt

IsyFact-Standard

Konzept
Überwachung und Konfiguration

Version 2.15
14.07.2016

IsyFact Standards

Ansprechpartner:

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

Dokumentinformationen

Dokumenten-ID:	Konzept_Ueberwachung-Konfiguration.docx

Java Bibliothek / IT-System

Name	Art	Version
isy-ueberwachung	Bibliothek	siehe isyfact-bom v1.3.6
isy-konfiguration (Teil isyfact-base)	Bibliothek	siehe isyfact-bom v1.3.6

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument enthält das technische Feinkonzept zur Überwachung und Konfiguration von Anwendungen gemäß den IsyFact-Standards.

2. Aufbau und Zweck des Dokuments

In diesem Dokument werden die Designvorgaben für die Überwachung und die Handhabung von Konfigurationen für Anwendungen gemäß der IsyFact-Referenzarchitektur vorgestellt [IsyFactReferenzarchitektur]. Gleichzeitig werden konkrete Nutzungskonzepte und Implementierungsvorgaben beschrieben.

Die Realisierung einer einheitlichen Überwachungsschnittstelle erlaubt es dem Betrieb frühzeitig Probleme in der Anwendungslandschaft zu erkennen und darauf zu reagieren. Dadurch können Ausfallzeiten minimiert werden.

Die Vereinheitlichung der Konfiguration erleichtert nicht nur die Entwicklung der Anwendungen, sondern stellt vor allem eine Erleichterung des Betriebs der Anwendungen da.

Alle Vorgaben orientieren sich an den in den jeweiligen Kapiteln aufgeführten Anforderungen und gelten für alle Anwendungen nach IsyFact-Architektur.

3. Überwachung von Anwendungen

In diesem Kapitel werden die Anforderungen an die Überwachung von Anwendungen beschrieben. Der Fokus liegt dabei auf der Schaffung von einheitlichen Administrationsschnittstellen, welche dem Betrieb eine einfache Überwachung der Anwendungen erlauben. Alle Vorgaben sind beim Entwurf und der Realisierung von neuen Anwendungen nach IsyFact-Architektur zu berücksichtigen. Gegebenenfalls können in Systementwürfen Verfeinerungen der hier getroffenen Vorgaben gemacht werden. Weiterhin können die hier gemachten Vorgaben als Basis für die Beschreibung der Überwachungsschnittstelle im Betriebshandbuch der Anwendungssysteme verwendet werden.

3.1. Kurzeinführung in das System-Monitoring mit JMX

Für die Überwachung einer Systemlandschaft und damit auch für die Überwachung der einzelnen Anwendungen werden Open-Source- Produkte wie Nagios[1] oder Icinga[2] empfohlen. Sie umfassen verschiedene Plugins, welche die zu überwachenden Daten sammeln. Diese Daten werden zur manuellen Überwachung auf einer Web-Oberfläche angezeigt. Zusätzlich können Administratoren bei bestimmten Systemzuständen automatisch informiert werden.

Dieser Abschnitt dient als Einstieg in das System-Monitoring und enthält eine kurze Einführung in die Technologie JMX. Die eigentlichen Designvorgaben werden erst im anschließenden Abschnitt beschrieben.

Für die Anbindung der Anwendungen an das Monitoring wird JMX[3] (Java Management Extensions) verwendet. Durch JMX können Java-Anwendungen ihren aktuellen Zustand über eine definierte Schnittstelle bekannt geben bzw. auf administrative Nachrichten reagieren. Dazu nutzt die Anwendung sogenannte MBeans (Managed Beans).

Eine solche MBean definiert eine Administrationsschnittstelle auf eine Ressource der Anwendung. Die Schnittstelle enthält beispielsweise Property-Getter-Methoden, um die Anzahl der eingeloggten Benutzer oder die Anzahl der in der letzten Minute durchgeführten Transaktionen zu ermitteln. Genauso kann eine MBean auch Property-Setter-Methoden anbieten, um den Zustand der von ihr verwalteten Ressource zu ändern. Ein Beispiel hierfür ist das Ändern von Log-Levels. Über Operationen können Aktivitäten angestoßen werden, z.B. das gezielte Aktualisieren von Caches.

Zusätzlich können MBeans Nachrichten (Notifications) empfangen und versenden. Der Mechanismus kann benutzt werden, um die Anwendung darüber zu informieren, dass das Management-System eine kritische Situation erkannt hat. Genauso kann die Anwendung damit das Management-System über die Änderung bestimmter System-Parameter explizit informieren. Da Notifications jedoch nicht zusammen mit allen

[1] Siehe http://www.nagios.org.
[2] Siehe http://www.icinga.org.
[3] Siehe http://java.sun.com/j2se/1.5.0/docs/guide/jmx/index.html.

System-Monitoring-Produkten eingesetzt werden können, soll darauf an dieser Stelle nicht weiter eingegangen werden.

Eine administrierbare Ressource wird von einem JMX-Agenten für externe Management-Anwendungen verfügbar gemacht. Dieser Agent wird im Normalfall in der gleichen virtuellen Maschine ausgeführt wie die von ihm kontrollierte Ressource. Der Agent stellt die Management-Schnittstelle (Management Interface) über einen MBean Server per RMI zur Verfügung.

MBeans sind standardisierte Java-Objekte. Es werden unterschiedliche Typen von MBeans unterschieden.

Standard MBeans sind die einfachste Art von MBeans. Der Zugriff auf die Management-Methoden erfolgt per Konvention. Konkret wird ein einfaches Java-Interface implementiert, welches die entsprechenden Management-Methoden anbietet. Der Name des Management-Interfaces (MI) muss per Konvention gleich dem Namen der MBean mit angehängtem Suffix „MBean“ sein. Das Interface bietet Attribute und Operationen an. Attribute werden über Getter- und Setter-Methoden bereitgestellt. Attribute, für die kein Setter existiert sind Read-Only-Attribute. Sonstige angebotene Methoden bieten sog. Operationen an.

Die Implementierung des Management-Interfaces, also die eigentliche MBean speichert und ermittelt die Daten, welche über das MI bereitgestellt werden.

Dynamische MBeans haben kein explizites Java-Interface. Das Interface wird dynamisch zur Laufzeit definiert. Dazu müssen sie das Interface „DynamicMBean“ implementieren. Dieses Interface enthält Methoden, über welche der JMX-Agent das Management Interface zur Laufzeit ermitteln kann. Zur Beschreibung des Interfaces wird ein MBeanInfo-Objekt verwendet. Dieses Objekt beschreibt die Methoden, deren Parameter und die Attribute des Management Interfaces.

Open MBeans: Während die zuvor genannten MBean-Typen prinzipiell beliebige Parameter und Rückgabe-Werte verwenden können, d.h. auch selbst definierte komplexe Java-Objekte zurückgeliefert werden können, schränken „Open MBeans“ die verwendbaren Datentypen ein. Prinzipiell sind Open MBeans dynamische MBeans, welche ein besonderes MBeanInfo-Objekt zur Beschreibung des MI an den JMX-Agenten liefern. Das MBeanInfo-Objekt muss das Interface OpenMBeanInfo implementieren. Durch die Einschränkung auf definierte Datentypen und die Bereitstellung besonders detaillierter Informationen über das OpenMBeanInfo-Objekt lassen sich Open MBeans besonders gut in beliebige JMX-konforme Management-Systeme einbinden. Insbesondere müssen dem Management-System keine besonderen Java-Klassen zum Datenaustausch bekannt gemacht werden. Die im OpenMBeanInfo-Objekt gelieferten Informationen eigenen sich sehr gut zum Anzeigen des Management-Interfaces in einer GUI, in dem Methoden und Parameter textuell beschrieben werden.

Model MBeans sind ebenfalls dynamische MBeans. Model MBeans eigenen sich vor allem dazu vorhandene Ressourcen managebar zu machen, ohne eine eigene MBean zu implementieren. Model MBeans werden vom MBean Server erzeugt. Die MBean wird der Anwendung durch den JMX-Agenten zur Verfügung gestellt. Die Anwendung konfiguriert die Model MBean, um so bekannt zu machen, wie die Management Informationen zu ermitteln sind.

Die Implementierung eines JMX-Agenten und eines MBean-Servers sind Teil von Java 5 und können durch entsprechende Parameter mit der VM mitgestartet werden. Dadurch steht JMX prinzipiell jeder Java-Anwendung ab J2SE 5.0 zur Verfügung.

3.2. Festlegungen und Ausgrenzungen

Für die Überwachbarkeit der Anwendungen werden die folgenden Festlegungen getroffen:

- Zum Anbieten einer Administrationsschnittstelle wird JMX verwendet
- Die Überwachung der Anwendungen erfolgt über das Nagios JMX-Plugin4.
- Der JMX-Agent muss per RMI erreichbar sein.
- Es werden keine Notifications verwendet. Da das zur Anbindung von Nagios verwendete Nagios-JMX-Plugin keine Notifications unterstützt.
- JMX wird nicht zur Konfiguration von Anwendungen verwendet.
- Zur Erhöhung der Sicherheit in der Betriebsumgebung muss eine Absicherung der RMI-Schnittstelle für JMX per Benutzername und Passwort erfolgen.
- Anwendungen stellen einen definierten Satz von Überwachungsinformationen bereit.

Folgende Punkte sind bewusst nicht Teil dieses Konzeptes:

- Die Konfiguration eines konkreten Produkts zum System-Monitoring ist nicht Teil des Konzeptes.
- Es gibt keine Vorgabe, welcher Typ von MBeans zu verwenden ist. Die Erzeugung der MBeans erfolgt transparent durch Spring und ist nicht konfigurierbar. Sowohl für den Entwickler, als auch für den Betrieb, der die MBeans ausliest, ist der MBean-Typ nicht relevant.

3.3. Designvorgaben für die Überwachungsschnittstelle

Diese Kapitel beschreibt die zu verwendenden Designmuster und Konventionen für MBeans.

[4] Siehe http://www.nagiosexchange.org/Misc.54.0.html?&tx_netnagext_pi1[p_view]=808&tx_netnagext_pi1[page]=20%3A10.

3.3.1 Namenskonventionen

In diesem Kapitel werden Konventionen für die Benennung der MBeans der Management-Schnittstelle beschrieben. Diese Konventionen beeinflussen direkt die nach außen sichtbare Management-Schnittstelle. Im Anschluss werden die Vorgaben für Benennung der Java-Packages bzw. -Klassen, welche die MBeans realisieren, beschrieben.

3.3.1.1 Benennung der Managed Beans

MBeans werden über sog. „Object Names“ bekannt gemacht. Hierüber kann das Management-System die einzelnen Beans adressieren und unterscheiden.[5] Ein Object Name hat die Form:

```
Domain:Key-Value-List
```

Ein Beispiel für einen gültigen Object Name ist:

```
de.msg.terminfindung:type=TransactionCounter
```

Prinzipiell ist die Domäne ein beliebiger String. Außer „:“ sind alle Zeichen erlaubt.

Die Name-Wert-Paare haben die Form:

```
key1=value,key2=value
```

Wichtig ist, dass hier Leerzeichen nicht ignoriert werden, d.h. obiger String ist nicht identisch zu:

```
key1=value,  key2=value
```

Die Reihenfolge der Parameter ist irrelevant. Sollte ein Wert potentiell Sonderzeichen enthalten (auch Komma oder Gleich sind Sonderzeichen) ist dieser mit Anführungsstrichen zu versehen. Beispiel:

```
roles="11,12,13",users="mherz,fbauer"
```

Neben den zuvor genannten technischen Restriktionen gelten für die Benennung von MBean-Objekten im Rahmen der Referenzarchitektur folgende Regeln:[6]

- Die Domäne hat die Form eines Java-Package-Namens. Die Domäne umfasst den Teil des Package-Namens bis zur Festlegung des Anwendungssystems. Die MBeans der Vorlage-Anwendunghaben demnach die Domäne:
  `de.msg.terminfindung`
- Alle Teile eines Object Names müssen gültige Java-Bezeichner sein. Insbesondere werden keine Umlaute oder Sonderzeichen verwendet.

[5] Object Names dürfen auch Platzhalter enthalten, um mehrere Beans auf einmal zu adressieren.
[6] Weitere Details zur Benennung von MBeans sind in [JMXBestPrac] beschrieben.

- Jede MBean definiert ihren Typ über eine Property „Type=…“. Dieser Typ identifiziert die Art der MBean und die darüber bereitgestellten Informationen. Zusätzlich erhält jede MBean über das Attribut „name“ einen eindeutigen Namen:
  `de.msg.terminfindung:type=BatchMonitor,name="Worker1"`
  `de.msg.terminfindung:type=BatchMonitor,name="Worker2"`

Der Type der MBean entspricht dem von ihr bereitgestellten Management Interface. Die zuvor genannten MBeans vom Typ „`BatchMonitor`“ implementieren das Interface „BatchMonitorMBean“[7] Die implementierende Java-Klasse heißt „`BatchMonitor`“. Daraus folgt offensichtlich, dass alle MBeans mit gleichem Typ dasselbe Management Interface anbieten.

3.3.1.2 Vorgaben für die Package-Struktur

MBeans, welche Informationen über die gesamte Anwendung bereitstellen oder identisch für mehrere Komponenten implementiert sind, werden in folgendem Package abgelegt:

```
<organisation>.<domäne>.<anwendungssystem>.common.jmx
(bspw. de.msg.terminfindung.common.jmx)
```

Diese MBeans sammeln entweder Daten über Komponentengrenzen hinweg oder sind gleichartig für mehrere Komponenten implementiert und werden mehrfach instanziiert.

MBeans, welche die Administration einzelner Anwendungskomponenten betreffen bzw. spezifisch hierfür implementiert wurden, werden im Layer „core“ in einem eigenen Package unterhalb der Anwendungskomponente abgelegt:

```
<organisation>.<domäne>.<anwendungssystem/register>.
core.<komponente>.jmx
(bspw. de.msg.terminfindung.core.verwaltung.jmx)
```

3.3.1.3 Vorgaben für Klassennamen

Der Java-Klassenname der eigentlichen MBean wird mit dem Suffix „MBean“ versehen. Der Name für die Monitor-MBean der Verwaltungskomponente[8] könnte z.B. "`VerwaltungMonitorMBean.java`“ genannt werden.

3.3.2 Entwurf der Anwendungsüberwachung

Dieses Kapitel beschreibt die Anforderungen an den Entwurf der MBeans und des Management-Interfaces.

[7] Für die Umsetzung der MBeans wird Spring verwendet. Das Management-Interface wird deklarativ über Annotations festgelegt und nicht explizit als Java-Interface umgesetzt, siehe Kapitel 3.3.3.
[8] Die Komponenten einer IsyFact-Anwendung werden in [IsyFactReferenzarchitektur] beschrieben.

3.3.2.1 MBeans enthalten keine Anwendungslogik

Das von der Anwendung bzw. einer Anwendungskomponente bereitgestellte Management-Interface wird in einer eigenen MBean-Klasse implementiert. Aspekte der Administration (MBean-Klasse) und der eigentlichen Anwendungslogik (Anwendungsfälle, DAOs…) dürfen nicht vermischt werden. Die MBean-Implementierung kümmert sich nur um die Bereitstellung der Management-Informationen, dafür notwendige Logik wird in den Anwendungskern ausgelagert. Allenfalls werden hier einfache Berechnungen (Durchschnittsbildung, Summierung usw.) durchgeführt.

3.3.2.2 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: Die Beans überprüfen nicht die Einhaltung von Grenzwerten oder Ähnlichem, dies ist Aufgabe des Management-Systems. Eine Implementierung in den MBeans wäre durch den Betrieb nur schwer zu steuern und intransparent. Außerdem können durch die Implementierung komplexer Logik in den MBeans Performance-Engpässe entstehen.

3.3.2.3 Business-Logik ruft Management-Logik

Der Anwendungskern (AWK) und die MBean werden von einer Spring-Factory erzeugt. Der Anwendungskern ruft Methoden der MBean auf (Push-Konzept). Der Kern erhält eine Referenz auf die MBean über die Komponentenfassade durch Spring-Dependency-Injection.

Das Auslesen der Informationen durch die MBean aus dem Anwendungskern (Pull-Konzept) ist zu vermeiden, da Performance-Probleme entstehen können.

3.3.3 Verwendung von Datentypen

Das Management-Interface darf nur die für Open MBeans erlaubten Datentypen für Parameter oder Rückgabewerte verwenden. Im Einzelnen sind dies die folgenden Typen:

```
java.lang.Void                  java.lang.Short
java.lang.Boolean               java.lang.Integer
java.lang.Byte                  java.lang.Long
java.lang.Character             java.lang.Float
java.lang.String                java.lang.Double
java.math.BigDecimal            java.math.BigInteger
java.util.Date                  javax.management.ObjectName
javax.management.openmbean.CompositeData (interface)
javax.management.openmbean.TabularData (interface)[9]
```

Die Methoden des Management Interfaces sollen keine Exceptions werfen. Falls sich dieses nicht vermeiden lässt dürfen nur Standard-Java-Exceptions geworfen werden. Würde das Management-Interface selbst definierte Datentypen oder Exceptions verwenden, müssten diese der Management-Anwendung bekannt gemacht werden.

3.3.4 Vorgaben für bereitgestellte Informationen

In diesem Kapitel werden die Informationen beschrieben, welche jede Anwendung zur Überwachung bereitstellen muss.

Für das Anbieten dieser genormten Management-Schnittstellen wird eine querschnittliche Bibliothek „`plis-überwachung`“ bereitgestellt, welche alle notwendigen MBeans enthält. Die Bibliothek wird in die zu überwachende Anwendung als Jar eingebunden.

3.3.4.1 Allgemeine Überwachungsinformationen

Tabelle 1 zeigt eine Liste der zu überwachenden bzw. anzubietenden Informationen für das Management-Interface. Diese Informationen müssen von allen Anwendungen bereitgestellt werden. Die beiden zuerst genannten MBeans werden standardmäßig von der Java-VM angeboten und müssen nicht selbst implementiert werden. Diese stehen daher prinzipiell auch für Batches zur Verfügung. Ansonsten werden die Vorgaben für das Management-Interface von Batches im Konzept [BatchDetailkonzept] beschrieben.

MBean-Name	
Attribut-Name	**Beschreibung**
java.lang:type=OperatingSystem	
FreePhysicalMemorySize	Liefert den freien physikalischen Arbeitsspeicher in Byte.

[9] Der Datentyp javax.management.openmbean.TabularData kann möglicherweise nicht von jedem JMX-Plugin verarbeitet werden. Er sollte daher nicht in einem Management-Interface verwendet werden.

FreeSwapSpaceSize	Liefert den freien Auslagerungsspeicher in Byte.
OpenfileDescriptorCount	Liefert die Anzahl der offenen Datei-Deskriptoren
java.lang:type=Memory	
HeapMemoryUsage.used[10]	Liefert den Java-Heap-Speicherverbrauch in Byte.
<organisation>.<domäne>.<anwendung>:type=StatusMonitor, name=“Status-Ueberwachung“[11]	
LetztePruefungErfolgreich	Gibt an, ob die letzte Prüfung der Anwendung erfolgreich war, oder nicht. Das Ergebnis wird als Boolean angegeben.
ZeitpunktLetztePruefung	Liefert den Zeitpunkt der letzten Prüfung als Java-Date.

Tabelle 1: Standard Überwachungsinformationen

Das Management-Interface für die StatusMonitor-MBean wird von folgender Java-Klasse angeboten:[12]

```
de.bund.bva.pliscommon.ueberwachung.common.jmx.StatusMonitorMBean
```

Die Aktualisierung der Daten erfolgt in der Prüfmethode der Admin-Komponente. Details dazu werden in Kapitel 3.3.5.1 beschrieben.

3.3.4.2 Informationen von Services

Tabelle 2 zeigt eine Liste der zu überwachenden bzw. anzubietenden Informationen für das Management-Interface von Service-Anwendungen. Die in Tabelle 2 aufgelisteten Informationen müssen für jeden Service einzeln angeboten werden.

<organisation>.<domäne>.<anwendung>:type=ServiceStatistik, name=“<Servicename>-Statistik“	
AnzahlAufrufeLetzteMinute	Liefert die Anzahl der in der letzten Minute erfolgten Aufrufe des Services insgesamt.[13]
AnzahlFehlerLetzteMinute	Liefert die Anzahl der in der letzten Minute erfolgten Aufrufe des Services,

[10] Für Attributwerte die zusammengesetzte Werte zurückliefern (CompositeData) wird Attributname und Composite-Key-Name durch Punkt getrennt.
[11] Die Werte dieser MBean werden von dem in Kapitel 3.3.5 beschriebenen Watchdog zur Verfügung gestellt.
[12] Diese Klasse ist Teil der Überwachungsbibliothek.
[13] Die Summe wird alle 60 Sekunden aktualisiert.

	bei denen ein technischer Fehler aufgetreten ist.[13]
DurchschnittsDauerLetzteAufrufe	Liefert die durchschnittliche Bearbeitungsdauer der letzten 10 Aufrufe der Services in Millisekunden (einfacher gleitender Durchschnitt).

Tabelle 2: Standard Überwachungsinformationen für Services

Da das Management-Interface für alle Services identisch ist, wird für die „ServiceStatistik“-MBean eine einzelne Java-Klasse in jeder Anwendung, die Services anbietet, verwendet:[12]

`de.bund.bva.pliscommon.ueberwachung.common.jmx.ServiceStatistikMBean`

Diese Klasse stellt das zuvor beschriebene Management-Interface für eine einzelne Servicekomponente zur Verfügung. Um die Informationen für mehrere Services in einer Anwendung anzubieten, wird die Klasse mehrfach als Spring-Bean konfiguriert und als Management-Bean bekannt gemacht.[14]

Die Aktualisierung der Daten in den Statistik-MBeans wird in der Service-Fassade der jeweiligen Komponente durchgeführt. Dazu wird per Spring ein AOP-Advice für die Service-Methoden der Komponente definiert. Dieses wurde in der Datei „`src/main/resources/resources/ueberwachung.xml`“ der Vorlageanwendung beispielhaft für die Controller durchgeführt:

```
<aop:config>
    <aop:advisor order="1000"
                 advice-ref="erstellungMonitor"
pointcut="target(de.msg.terminfindung.gui.terminfindung.erstellen.ErstellenController)"/>
...
</aop:config>
```

Der Advice bewirkt, dass jeder Aufruf der Klasse „`ErstellenController`“ zu einem Aufruf der Methode „`zaehleAufruf`“ in der Bean „`erstellungMonitor`“ führt. Die Bean „`erstellungMonitor`“ ist die MBean-Instanz für den `ErstellenController`. Eine entsprechende Konfiguration wird also für jeden Service in die Spring-Konfiguration aufgenommen.

Jeder AOP-Advisor mit einem Pointcut erfordert nicht unbeträchtliche Ressourcen des Heaps. Es muss daher darauf geachtet werden, die Advisors für verschiedene Ziele nicht zu kopieren, sondern einen gemeinsamen Advisor mit einem Pointcut zu verwenden, der die verschiedenen Ziele mit „or“ verknüpft, wie im folgenden Beispiel dargestellt:

[14] Für Details siehe Kapitel 3.4.

```
<aop:config>
  <aop:advisor order="1000" advice-ref="erstellungMonitor"
pointcut="target(de.msg.terminfindung.gui.terminfindung.erstellen.ErstellenControl
ler1) or
target(de.msg.terminfindung.gui.terminfindung.erstellen.ErstellenController2)"/>
</aop:config>
```

Alternativ können und sollten Wildcard-Pattern in den Pointcuts angewendet werden, soweit möglich.

Für das Monitoring per AOP werden intern Datenstrukturen gespeichert, die vorhalten, an welchen Stellen Aspekte eingebracht werden. Das kann, wenn eine große Anzahl von Service-Klassen mit vielen Methoden genutzt wird, zu einem großen Speicherverbrauch führen. Dies muss beim Design der Services berücksichtigt werden. Da in Anwendungen pro Komponente in der Regel ein Service angeboten wird und in Anwendung gemäß Referenzarchitektur nur eine eher kleine Anzahl von Komponenten vorhanden ist, stellt dies in der Regel kein Problem dar.

3.3.5 Vorgaben für die Prüfung der Verfügbarkeit

Anwendungen nach IsyFact-Architektur sollen Mechanismen bereitstellen, die es erlauben, die Verfügbarkeit der Anwendung durch eine betriebliche Überwachung zu prüfen. Grundlage dafür ist die Bereitstellung einer Ping- und einer Prüfmethode durch die Anwendung. Zusätzlich wird in jeder Anwendung ein sog. Watchdog implementiert, welcher die Funktionsfähigkeit der Anwendung kontinuierlich prüft und das Ergebnis per JMX bereitstellt.

3.3.5.1 Implementierung von Ping- und Prüfmethoden

Jede Anwendung muss eine Service-Operation anbieten, die es nutzenden Nachbarsystemen erlaubt, die Erreichbarkeit dieses Systems zu prüfen. Bei der Implementierung dieser Ping-Methode sind dabei folgende Vorgaben einzuhalten:[15]

- Die Ping-Methode wird als Service-Methode der Admin-Komponente angeboten.
- Die Ping-Methode verwendet einen String als Parameter und liefert beim Aufruf den übergebenen String zurück. Neben dem String muss zusätzlich ein weiterer String Parameter mit der Korrelations-ID übergeben werden. Aufrufer müssen das Feld Korrelations-ID immer zwingend befüllen.
- Für Systeme, die einen Tomcat verwenden, wird genau eine Ping-Methode pro Webanwendung angeboten.

[15] Aufgrund der Trivialität dieser Methode wird dafür keine querschnittliche Bibliothek angeboten, sondern die Methode explizit in jeder Anwendung implementiert.

- Java-Anwendungen welche dauerhaft laufen und keinen Tomcat verwenden bieten ebenfalls eine einzelne Ping-Methode an.
- Nicht dauerhaft laufende Anwendungen, z.B. Batches bieten keine Prüfmethode an.

Neben der Ping-Methode wird in jeder Anwendung eine Prüfmethode implementiert, welche die Funktionsfähigkeit des Systems überprüft. Die Prüfmethode, wird dabei gemäß den folgenden Anforderungen implementiert:

- Die Prüfmethode wird als Methode der Administrationskomponente implementiert.
- Die Prüfmethode darf keine fachlich relevanten Daten verändern.
- Die Prüfmethode wird nur intern vom Watchdog verwendet und nicht über eine Service-Schnittstelle angeboten werden. Die Prüfmethode darf nicht auf Anforderung per JMX den Status des Systems prüfen. Ansonsten könnten negative Performanceeinflüsse durch sehr häufiges Pollen des Status nicht ausgeschlossen werden (siehe Kapitel 3.7).
- Die Prüfmethode muss zu Beginn eine Korrelations-ID erzeugen, die im Logging-Kontext gesetzt werden muss. Bei jedem Aufruf an ein Nachbarsystem ist diese zu übergeben.
- Die Prüfmethode muss so implementiert werden, dass mindestens folgende Aspekte der Anwendung getestet werden:
  - Verfügbarkeit aller genutzten Nachbarsysteme. Hierzu wird die Ping-Methode dieser Nachbarsysteme aufgerufen. Der Aufruf einer fachlichen Funktion ist nicht gestattet, um fachliche Auswirkungen durch die Watchdog-Funktion zu verhindern.
  - Verfügbarkeit der Datenbank, sofern die Anwendung eine Datenbank verwendet. Dafür wird eine einfache immer gültige SQL-Abfrage definiert und ausgeführt.[16]
  - Verfügbarkeit weiterer genutzter Ressourcen, wie beispielsweise der LDAP-Server oder genutzte FTP-Verzeichnisse. Bei der Prüfung der genutzten Ressourcen ist zu beachten, dass die Prüfmethode sich nicht aufhängt und somit die Prüfung nicht weiterläuft. Dies hätte zur Folge, dass der Watchdog nicht mehr arbeitet und somit die Überwachung der Anwendung fehlschlägt.[17] Um dies zu vermeiden, sollte zur Prüfung der genutzten Ressourcen das Future-Pattern verwendet werden:

```
boolean checkFaceVacs() {
  ExecutorService executor =
    Executors.newCachedThreadPool();

  Future<Boolean> future =
```

[16] Eine Beispielimplementierung kann der Klasse AdminImp.java des Vorlage-Registers entnommen werden.
[17] Als Beispiel sei hier der LDAP-Server genannt. Zur Prüfung des LDAP-Servers wird in der Regel eine Beispielanfrage an den Server gesendet. Ist vor den LDAP-Server ein Loadbalancer geschaltet, so kann es nach einem Fail-Over passieren, dass diese Beispielanfrage endlos läuft.

```
    executor.submit(new Callable<Boolean> () {
      public Boolean call() {
        if (!anwendungXYZ.isAnwendungXYZAlive()) {
          throw new AnwendungXYZNotAvailableException();
        }
        return Boolean.TRUE;
      }
  });

  try {
    Boolean value = future.get(10, TimeUnits.SECONDS);
    return value.booleanValue();
  } catch (Exception e) {
    return false;
  }
}
```

3.3.5.2 Implementierung des Watchdogs

In jeder Anwendung wird ein Watchdog realisiert, welcher in regelmäßigen Abständen die Prüfmethode der Anwendung aufruft. Der Aufruf der Prüfmethode prüft den Status des Systems und aktualisiert das Ergebnis in der MBean.

Der Watchdog wird mit Spring realisiert. Dazu ist die folgende Konfiguration notwendig:[18]

```
<bean id="administration"
class="de.msg.terminfindung.gui.administration.TerminfindungWatchdog">
  <property name="watchdogMBean" ref="watchdogMonitor"/>
  <property name="konfiguration" ref="konfiguration"/>
  <property name="entityManager" ref="entityManagerFactoryBean"/>
  <property name="executor">
    <bean
class="org.springframework.scheduling.concurrent.ThreadPoolExecutorFactoryBean">
      <property name="queueCapacity" value="${admin.watchdog.threadpool.size}"/>
</bean>

<task:scheduled-tasks scheduler="scheduler">
  <task:scheduled ref="administration" method="pruefeSystem"
                  fixed-delay="${admin.watchdog.interval}" initial-delay="5000"/>
</task:scheduled-tasks>

<task:scheduler id="scheduler" pool-size="${tasks.threadpool.size}"/>
```

Die vorgestellte Konfiguration erzeugt einen TimerTask, welcher in regelmäßigen Abständen die Prüfmethode „pruefeSystem“ der Admin-Bean („admininistration“) aufruft. Die Komponente Administration wird durch Ableiten der Klasse WatchdogImpl aus plis-überwachung implementiert.

[18] Ein Beispiel findet sich in den Dateien „ueberwachung.xml und „tasks.xml“ unter „src/main/resources/resources“ der Vorlageanwendung.

```
public class TerminfindungWatchdog extends WatchdogImpl implements InitializingBean {

  ...

  @Override
  public void afterPropertiesSet() throws Exception {
     addPruefung("Datenbank", new Callable<Boolean>() {
       @Override
       public Boolean call() throws Exception {
         final String watchdogQuery =
           konfiguration.getAsString(CONF_ADMIN_WATCHDOG_VALIDATION_QUERY);

           entityManager.createNativeQuery(watchdogQuery).getSingleResult();
           return true;
         }
    });
  }

  ...
```

Für den Betrieb wird ein Konfigurationsparameter

`admin.watchdog.interval`

in die betriebliche Konfigurationsdatei aufgenommen, über den sich das Prüfintervall konfigurieren lässt.

3.4. Implementierung der JMX-Schnittstelle mit Spring

Alle MBeans werden als Spring-Beans konfiguriert und über den MBeanExporter von Spring exportiert. Dazu ist der folgende Abschnitt in die Spring-Konfiguration aufzunehmen:[19]

```
<bean id="mBeanExporter"
 class="org.springframework.jmx.export.MBeanExporter">
  <property name="assembler" ref="assembler" />
  <property name="autodetect" value="false" />
  <property name="beans">
    <map>
      <entry
       key="de.msg.terminfindung:type=DemoBean"
       value-ref="demoBean" />
    </map>
  </property>
</bean>
```

Dabei enthält das Attribut „`key`“ einen Object Name und „`value-ref`“ den Namen eines Spring-Beans. Durch diese Konfiguration wird der MBeanExporter initialisiert und angewiesen, die Spring-Bean „`demoBean`“ unter dem Object Name „`de.msg.terminfindung:type=DemoBean`“ zu exportieren.

Die „`demoBean`“ wird wie eine gewöhnliche Spring-Bean definiert, muss jedoch, zur Einhaltung der Initialisierungsreihenfolge, vom „`mBeanExporter`“ abhängen:

[19] Siehe „src/main/resources/resources/ueberwachung.xml“ in der Vorlageanwendung.

```
<bean id="basisdatenStatistik" class="DemoBean" depends-
on="mBeanExporter"/>
```

Das Management-Interface der MBean wird nicht als Java-Interface implementiert, sondern dynamisch von Spring an Hand von Annotations ermittelt. Dazu wird ein MBeanInfoAssembler wie folgt konfiguriert:

```
<bean id="assembler"
 class="….jmx.export.assembler.MetadataMBeanInfoAssembler">
  <property name="attributeSource" ref="jmxAttributeSource"/>
</bean>
<bean id="jmxAttributeSource"
 class="….jmx.export.annotation.AnnotationJmxAttributeSource"/>
```

Im Java-Code der MBean-Klasse werden alle via JMX zu exportierenden Methoden, d.h. die Methoden, die Teil des Management-Interfaces werden sollen, mit der Annotation `@ManagedAttribute` bzw. `@ManagedOperation` versehen:

```
@ManagedAttribute (description = "Liefert die Dauer der letzten
Suchabfrage in ms.")
public long getLetzteSuchdauer() {…}
```

Zusätzlich muss die MBean-Klasse mit der Annotation `@ManagedResource` versehen:

```
@ManagedResource(
        description = "Diese MBean liefert Statistiken…")
public class BasisdatenStatistikMBean {…}
```

3.5. Konfigurieren der Laufzeit-Umgebung für JMX

Damit die MBeans per JMX erreichbar sind, muss der JMX-Agent der Java-VM aktiviert werden.

Die Java VM enthält seit Version 5 einen JMX-Agenten.[20] Dieser kann durch den VM-Parameter

```
-Dcom.sun.management.jmxremote
```

gestartet werden. Diese Einstellung erlaubt jedoch keinen Zugriff per RMI. Die Aktivierung von RMI wird im folgenden Abschnitt erläutert.[21]

[20] Insbesondere können so auch Standalone-Java-Anwendungen direkt per JMX administriert werden.
[21] Weitere mögliche Parameter sind [JMXParam] zu entnehmen.

3.5.1 Aktivierung von RMI

Um Anwendungen per JMX administrieren zu können, muss der JMX-Agent der VM gestartet werden. Dazu müssen folgende VM-Parameter gesetzt werden:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=%RMI_PORT%
-Dcom.sun.management.jmxremote.ssl=false
```

Zusätzlich muss das Passwort für den RMI-Zugriff in der Datei „`%JAVA_HOME/lib/management/jmxremote.password`“ gesetzt werden.[22] Eine Vorlage für die Datei befindet sich als „`jmxremote.password.template`“ im selben Verzeichnis.

Die oben angegebenen Parameter werden für Webanwendungen in das Startskript von Tomcat (z.B. startup.sh) aufgenommen:[23]

```
CATALINA_OPTS='-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=%RMI_PORT% \
-Dcom.sun.management.jmxremote.ssl=false'
```

Für Batches (siehe [BatchDetailkonzept]) werden die VM-Parameter in die ausgelieferten Startskripte integriert.

3.5.2 Abschalten der Authentifizierung für die Entwicklung

In Entwicklungsumgebungen kann die Authentifizierung abgeschaltet werden. Dazu sind die JMX Parameter wie folgt zu ändern:

```
-Dcom.sun.management.jmxremote.port=%RMI_PORT%
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```

3.6. Testen der Überwachungsschnittstelle

In diesem Kapitel wird beschrieben, wie auf MBeans manuell mit der Anwendung JConsole oder zu automatisierten Tests aus JUnit heraus zugegriffen werden kann.

3.6.1 Zugriff über JConsole

Für manuelle Tests der JMX-Schnittstelle kann die Anwendung „JConsole“ verwendet werden. Sie ist Bestandteil des JDK. JConsole erlaubt sowohl Verbindungen zu lokalen VMs, als auch Remote-Verbindung per RMI. Nachdem Start der Anwendung kann entweder eine lokale VM mit aktiviertem JMX-Agenten direkt ausgewählt werden oder auf dem Register „Remote“ die RMI-Verbindungsparameter eingegeben werden.

[22] Das Passwort gilt so für alle Anwendungen, die diese VM benutzen. Soll eine Konfiguration pro Anwendung durchgeführt werden, so können beim VM-Start weitere Parameter für anwendungsspezifische Konfigurationen mitgegeben werden. Details dazu können [JMXParam] entnommen werden.

[23] Die durch „\“ angedeuteten Zeilenumbrüche sind Layout-bedingt und in der Konfiguration nicht einzugeben.

Die Anwendung selbst ist in mehrere Reiter aufgeteilt (siehe Abbildung 1). Auf den Reitern „Summary“, „Memory“, „Threads“, „Classes“, und „VM“ werden die von jeder VM standardmäßig bereitgestellten JMX-Informationen angezeigt. Selbst definierte MBeans werden in einer Baum-Ansicht auf dem Reiter „MBeans“ dargestellt.

Der Baum enthält alle registrierten MBeans der VM strukturiert nach ihrem Object Name. Wird eine MBean gewählt erscheinen auf der rechten Seite die vom Management-Interface angebotenen Attribute, Operationen und Notifications. Auf der Seite „Info“ werden allgemeine Informationen zur MBean angezeigt. Änderbare Attribute werden blau dargestellt. Zum Ändern werden die Werte direkt in die Tabelle auf der Seite „Attributes“ eingetragen. Durch Doppelklick auf ein Read-Only Attribut wird für diesen Wert ein Diagramm dargestellt.

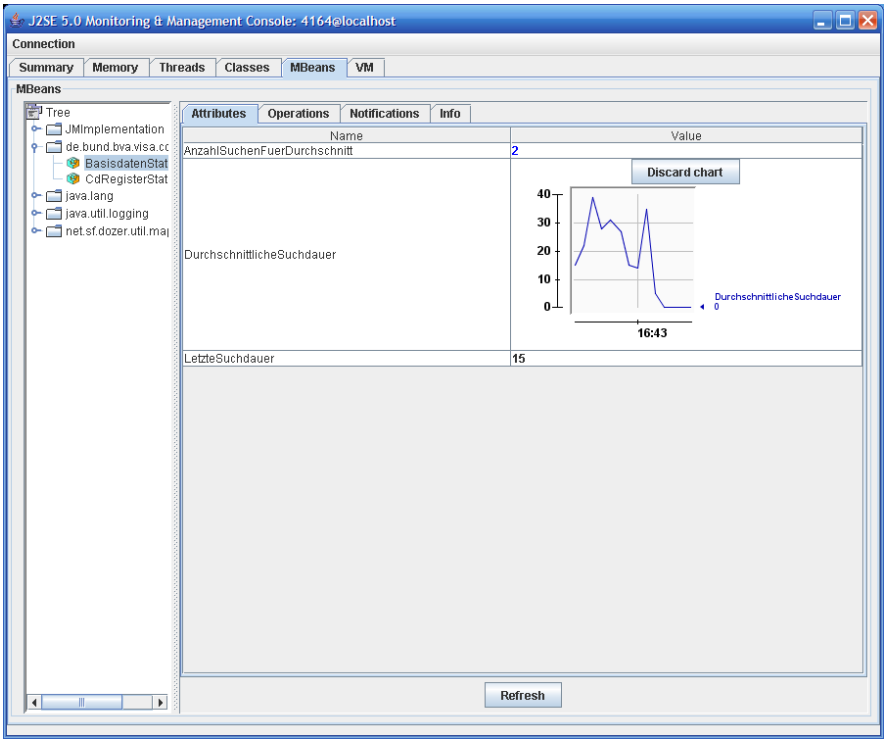

Abbildung 1: JMX-Konsole „JConsole“

3.6.2 Automatisierte JUnit-Tests

Das per JMX bereitgestellte Management-Interface kann sehr einfach per JUnit getestet werden. Der Zugriff auf eine MBean erfolgt dabei nach folgendem Muster:[24]

```
// Holen des MBeanServers
ArrayList<MBeanServer> mBeanServerList =
MBeanServerFactory.findMBeanServer(null);
MBeanServer mBeanServer = mBeanServerList.get(0);
```

[24] Ein implementierter Test findet sich in der Vorlageanwendung in TestJmxUeberwachung.testJmxUeberwachung().

```
// Lesen der gewünschten Information per JMX
Hashtable<String, String> table = new Hashtable<>();
table.put("type", "ServiceStatistik");
table.put("name", "\"Erstellung-Statistik\"");
ObjectName testObjectName = new ObjectName("de.bund.bva.isyfact.terminfindung",
table);
String testAttributeName = "DurchschnittsDauerLetzteAufrufe";
String result = mBeanServer.getAttribute(testObjectName, testAttributeName)
                           .toString();

// Auswerten des Ergebnisses
assertEquals("0", result);

// Einen Anwendungsfall ausführen
erstellenController.initialisiereModel(new ErstellenModel());
result = mBeanServer.getAttribute(testObjectName,
testAttributeName).toString();
assertNotEquals("0", result);
```

3.7. Performance

Die im Konzept beschriebenen Überwachungsfunktionen dürfen keinen relevanten negativen Einfluss auf die Performance der Anwendung haben. Dazu sind neben der Einhaltung der in Kapitel 3.3.2 beschriebenen Vorgaben noch einige grundlegende Regeln zu beachten:

- Der Aufruf der in Kapitel 3.3.5.1 beschriebenen Prüfmethode durch den Watchdog darf in nicht zu kurzen Abständen erfolgen. Ein sinnvoller Richtwert hierfür sind 5 Sekunden.
- Da nicht auszuschließen ist, dass ein Überwachungswerkzeug sehr häufig Informationen aus MBeans abruft, darf das Abrufen von Management-Informationen aus MBeans keine zeitaufwändigen Aktionen im Anwendungssystem veranlassen.
- Bei der Bereitstellung weiterer Überwachungsinformationen ist darauf zu achten, dass die Ermittlung der Kennzahlen keinen relevanten negativen Einfluss auf die Anwendungs-Performance hat. Insbesondere dürfen keine fachlichen Funktionen des Anwendungskerns aufgerufen werden.
- MBeans werden in der Regel von mehreren Threads genutzt. Die Synchronisierung der MBean-Methoden ist möglichst effizient zu gestallten.

3.8. Anwendungen deaktivierbar machen

Für die Durchführung von Updates beim Deployment ist es notwendig, einzelne „Knoten“ eines Anwendungsclusters aus dem Loadbalancing herauszunehmen, so dass dieser Knoten keine Anfragen mehr vom Loadbalancer zugeteilt bekommt.

3.8.1 Beschreibung des Loadbalancer-Servlets

Zur Realisierung dieser Anforderung wird als Teil jeder Webanwendung ein sog. Loadbalancer-Servlet ausgeliefert. Das Servlet prüft beim Aufrufen seiner URL, ob eine IsAlive-Datei im Konfigurationsverzeichnis (siehe

[DeploymentKonzept]) vorhanden ist. Ist eine solche Datei vorhanden, liefert das Servlet den HTTP-Statuscode HTTP OK (200) zurück. Falls keine IsAlive-Datei gefunden wird liefert das Servlet den Code HTTP FORBIDDEN (403) zurück.

Der Loadbalancer prüft in regelmäßigen Abständen die URL des Servlets und nimmt die entsprechende für die Anwendung den entsprechenden Server aus dem Loadbalancing heraus, falls kein HTTP OK gelesen wird. Zu beachten ist, dass auf einem Server prinzipiell mehrere verschiedene Anwendung laufen können. Der Loadbalancer muss daher so konfiguriert werden, dass auf dem Server nur die betreffende Anwendung deaktiviert wird, zu der das Loadbalancer-Servlet gehört. Alle anderen Anwendungen auf dem entsprechenden Server müssen weiterhin bedient werden.

3.8.2 Integration des Loadbalancer-Servlets

Das Loadbalancing-Servlet ist als Teil der Bibliothek PLIS-Überwachung in der Klasse `de.bund.bva.pliscommon.ueberwachung.service.loadbalancer.LoadbalancerServlet` implementiert. Zur Nutzung muss in die `web.xml` jeder Webanwendung folgender Abschnitt aufgenommen werden:

```
<servlet>
  <servlet-name>loadbalancer</servlet-name>
  <servlet-class>

de.bund.bva.pliscommon.ueberwachung.service.loadbalancer.LoadbalancerServl
et
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>loadbalancer</servlet-name>
  <url-pattern>/Loadbalancer</url-pattern>
</servlet-mapping>
```

Standardmäßig verwendet das Servlet die Datei `/WEB-INF/classes/config/isAlive`[25] als IsAlive-Datei. Die zu suchende Datei kann bei Bedarf durch den Servlet-Parameter „`isAliveFileLocation`“ in der `web.xml` geändert werden.

3.8.3 Nutzung des Loadbalancing-Servlets

Durch die oben beschriebene Konfiguration kann die gewünschte Verfügbarkeit der Anwendung über die URL `http://<serverurl>/<anwendungsname>/Loadbalancer` abgefragt werden.

Zur Steuerung des Loadbalancing-Servlets muss die IsAlive-Datei im Konfigurationsverzeichnis der Anwendung angelegt bzw. entfernt werden.

[25] Nach dem Deployment entspricht dies der Datei /etc/<anwendungsname>/isAlive.

Der Standardname für die IsAlive-Datei ist `/etc/<anwendungsname>/isAlive`. Dieses kann der Betrieb bei Bedarf über ein Shell-Skript automatisieren.[26]

Die Verwendung des Servlets im Rahmen des Deployments wird in [DeploymentKonzept] beschrieben.

[26] Ein Beispiel für ein solches Skript findet sich in [VorlageAnwendung]/cd-register/src/main/skripte/lb_tomcat.sh.

4. Vorgaben für Konfigurationen

In diesem Kapitel wird die Handhabung von Konfigurationen für Anwendungen der IsyFact-Architektur beschrieben. Dazu gehören Vorgaben für die Ablage von Konfigurationsdateien und Implementierungshinweise zum Lesen der Konfigurationen. Außerdem werden Besonderheiten für die Konfiguration der eingesetzten Bibliotheken und Frameworks beschrieben.

Für das Verständnis ist es wichtig die Konfigurationsparameter von den konkreten Parameterwerten für diese Konfigurationsparameter zu unterscheiden. Erstere sind fest von der Anwendungsimplementierung vorgegeben. Die Anwendung legt z.B. fest, dass es einen Parameter „`datenbank.kennwort`“ zur Festlegung des Datenbankkennworts gibt. Parameterwerte meinen die Einstellungen für diese Parameter und werden z.B. vom Betrieb konfiguriert. Als Konfiguration wird die Menge aller Konfigurationsparameter einer Anwendung verstanden.

4.1. Festlegungen und Ausgrenzungen

- Das Konfigurationskonzept betrifft alle von den IsyFact-Anwendungen verwendeten Konfigurationen. Dazu gehören sowohl vom Betrieb zu pflegende Konfigurationsdateien, als auch statische Konfigurationen, die z.B. das Layout von Dialog-Masken beschreiben (Ressource-Dateien). Wenn im Folgenden von Konfigurationen gesprochen wird, sind sowohl Konfigurationen im eigentlichen Sinne, als auch Ressourcen gemeint.
- Nicht zu den hier erfassten Konfigurationen gehört die Konfiguration der Basis-Software, z.B. des Tomcat.
- Anwendungen müssen im Normalfall für Konfigurationsänderungen neu gestartet werden. Ausnahmen hiervon bedürfen besonderer technischer und organisatorischer Maßnahmen. Details dazu werden in Kapitel 4.4 beschrieben.
- Anwendungen werden im Cluster betrieben und verfügen nicht über ein gemeinsames Datei-System. Datei-basierte Konfigurationen müssen daher für alle Knoten eines Clusters einzeln gepflegt werden.
- Umgebungsspezifische Parameter, z.B. Datenbank-URL und Passwort, sind alleine durch den Betrieb zu pflegen.
- Das Konfigurationskonzept berücksichtigt sowohl die Konfiguration für die Entwicklungsumgebung als auch geeignete Vorgehensweisen für die Konfiguration der Produktionsumgebung.
- Jede Fachanwendung enthält einen Konfigurationsparameter zur Deaktivierung der Schreibzugriffe (Meldungen), der z. B. bei Durchführung längerer Datenmigrationen verwendet wird (siehe Abschnitt 4.5.5). Das Auslesen dieses Parameters wird so realisiert, dass er zur Laufzeit umkonfiguriert werden kann (siehe Abschnitt 4.4).
- Alle Anwendungssysteme und Batches, die schreibend auf eine andere Fachanwendung zugreifen, müssen auf die Nichtverfügbarkeit dieser

Komponente vorbereitet sein. Entweder können jene Systeme vorübergehend heruntergefahren sein, oder bestimmte Funktionen, z.B. der Schreibzugriff, können über Konfigurationsparameter deaktiviert sein. Systeme, bei denen Funktionen deaktiviert sind, zeigen Benutzern frühzeitig einen Hinweis an, welche Funktionen nicht zur Verfügung stehen.

Ob eine Anwendung einen Konfigurationsparameter erhält oder heruntergefahren werden kann, muss für jede Anwendung, abhängig von deren Verfügbarkeitsanforderung, individuell entschieden werden.

Als weitere Rahmenbedingung gilt, dass während der Entwicklung die für die Produktion relevanten Werte der Konfigurationsparameter nicht bekannt sind.

4.2. Typisierung und Handhabung von Konfigurationen

Eine Einordnung der Konfigurationen ist für das Deployment und den Betrieb einer Anwendung notwendig. Nur so ist sichergestellt, dass z.B. Parameterwerte für die Produktion nicht schon während des Bauens der Anwendung bekannt sein müssen. Außerdem wird gewährleistet, dass die jeweilige verantwortliche Personengruppe einen leichten Zugriff auf „ihre“ Konfigurationsparameter erhält.

Das wichtigste Kriterium ist, ob die betreffende Konfiguration für alle Umgebungen (Ziel-Systeme) identisch ist, und die Fragestellung, von wem die Konfiguration angepasst wird. Grundsätzlich kommen dafür Entwickler, der Betrieb oder die Fachabteilung in Frage.

Konfigurationen lassen sich wie in Tabelle 3 dargestellt typisieren:

Die Spalte „Pflegeverantwortung“ gibt an, wer die entsprechenden Konfigurationen pflegt. So wird beispielsweise die Spring-Konfiguration ausschließlich von den Anwendungsentwicklern bearbeitet. Die pflegende Gruppe muss aber nicht zwangsläufig die Inhalte des entsprechenden Konfigurationstyps bestimmen. So werden Validierungsregeln maßgeblich durch eine Fachabteilung inhaltlich vorgegeben werden. Trotzdem ist die Konfiguration statisch, d.h. sie ist schon zur Entwicklungszeit bekannt und auch nach der Installation nicht mehr veränderbar.

Der Spalte „Umgebungsabhängigkeit erlaubt“ lässt sich entnehmen, ob der entsprechende Konfigurationstyp für eine bestimmte Umgebung (d.h. Testumgebungen, Produktionsumgebung) spezifische Teile enthalten darf. So dürfen von Entwicklern zu pflegende Konfigurationen niemals umgebungsabhängig sein. Wäre dies der Fall müsste, beispielsweise ein Entwickler das Kennwort der Produktionsdatenbank kennen.

Die Spalte „Erlaubte Zugriffsart der Anwendung“ gibt an, ob der entsprechende Konfigurationstyp von der Anwendung nur gelesen oder auch geschrieben werden darf. Die wenigsten Konfigurationen sollten durch die Anwendung selbst geschrieben werden. Lediglich Benutzerkonfigurationen werden typischerweise zur Laufzeit der

Anwendung dynamisch geändert. Diese müssen in der Datenbank gespeichert werden.

Die letzte Spalte der Tabelle gibt die bevorzugte Art für die Speicherung der entsprechenden Konfiguration an. Der Ablageort ist für die Paketierung der Anwendung (Build) und das Deployment wichtig.[27] So müssen betriebliche Konfigurationen leicht durch den Betrieb zugänglich und änderbar sein. Daher werden diese in einem separaten Ordner „config“ in Form von einfachen Property-Dateien abgelegt. Statische Konfigurationen sind bereits zum Build-Zeitpunkt bekannt und können als Ressourcen mit der Anwendung verpackt werden. Hier kommen häufig auch komplexere, XML basierte Konfigurationsdateien zum Einsatz. Da Benutzer-Konfigurationen durch die Anwendung geschrieben werden, dürfen diese nicht im Datei-System abgelegt werden. Ansonsten wäre eine gesonderte Synchronisierung dieser Dateien notwendig, wenn die Anwendung im Cluster betrieben wird.

Konfigurationstyp	**Statische Konfiguration**	**Betriebliche Konfiguration**	**Benutzer-Konfiguration**
Pflegeverantwortung	Entwickler	Betrieb	Fachabteilung oder Administratoren
Beispiel	Spring-Konfiguration	Datenbank-Benutzer und -Kennwort	Dialog-Einstellungen
Umgebungs-abhängigkeit erlaubt	nein	ja	nein
Erlaubte Zugriffsarten der Anwendung	nur lesend	nur lesend	lesend und schreibend
Speicherung[28]	Als Datei im Resources-Ordner	Als Property-Datei im Config-Ordner	In der Datenbank

Tabelle 3: Typisierung von Konfigurationen nach Zielgruppen

[27] Für Details siehe Kapitel 4.3.1.
[28] Der Speicherort der Ordner „Resources“ und „Config“ wird in Kapitel 4.3.1 beschrieben.

4.3. Vorgaben für die Ablage und Verwendung von Konfigurationen

Dieses Kapitel enthält die Vorgaben wo Konfigurationen abgelegt und wie diese verwendet werden.

4.3.1 Datei-basierte Konfigurationen

Konfigurationsparameter, die nicht durch die Anwendung geschrieben werden, sollen in Dateien und nicht in der Datenbank gespeichert werden. Als Format kommen bevorzugt Property-Dateien zum Einsatz.

Alle datei-basierten Konfigurationen werden im Klassenpfad abgelegt. Dazu werden die Ordner „`config`“ und „`resources`“ verwendet. Alle Konfigurationen werden in der Entwicklung unterhalb von „`src/main/resources`“ abgelegt.[29] Dort werden Unterordner wie folgt angelegt:

- **Resources-Ordner:** In „`src/main/resources/resources/`“ liegen die statischen Konfigurationen. Zur Strukturierung sollen hier Unterordner für gleichartige Konfigurationen angelegt werden (z.B. spring, hibernate). Falls sich eine Konfiguration explizit auf eine Java-Klasse bezieht (z.B. Dialog-Beschreibungen) wird eine Verzeichnisstruktur analog zur Package-Struktur angelegt und die Konfiguration dort abgelegt, z.B.:
  `src/main/resources/resources/de/msg/terminfindung/gui/verwaltung/eingabe-dialog.xml`
- **Config-Ordner:** In „`src/main/resources/config/`“ liegen alle betrieblichen Konfigurationen.

4.3.1.1 Namenskonventionen für Konfigurationsparameter

Für die Benennung von Konfigurationsparametern werden Zeichenketten ohne Sonderzeichen verwendet. Parameternamen bestehen aus mehreren Teilen, welche durch Punkte getrennt werden. Die Teile werden mit dem unspezifischsten Begriff beginnend sortiert aufgeschrieben:

```
datenbank.kennwort
datenbank.benutzername
```

So entsteht eine Hierarchie von Parameternamen (alle mit „datenbank“ beginnenden Parameter beziehen sich auf die Datenbankkonfiguration).

Komponentenspezifische Parameter beginnen mit dem Namen der Komponente, die sie konfigurieren.

```
verwaltung.regelwerk.regelpfad=…
```

[29] Die Ablage der Konfigurationsdateien zur Laufzeit wird in Kapitel 4.3.1.4 beschrieben.

Im Übrigen sind möglichst aussagekräftige Bezeichner zu verwenden. Die Sprache sollte deutsch sein, sofern es sich nicht um feststehende englische Begriffe handelt (z.B. „Session“).

4.3.1.2 Implementierungsvorgaben für Property-Dateien

Zur Bereitstellung der Konfigurationsparameter in der Anwendung wird die Bibliothek PLIS-Konfiguration verwendet.

Die Konfigurations-Bibliothek stellt Interfaces und Implementierungen für das Laden von Property-Dateien und das typsichere Auslesen von Konfigurationsparametern bereit. Die Konfiguration wird der Anwendung als querschnittliche Spring-Bean (im Folgenden Konfigurations-Bean genannt) bereitgestellt:

```
<bean id="konfiguration"
class="de.bund.bva.pliscommon.konfiguration.common.impl.ReloadableProperty
Konfiguration">
    <constructor-arg>
        <list>
            <value>/config/terminfindung.properties</value>
            <value>/config/jpa.properties</value>
        </list>
    </constructor-arg>
</bean>
```

Die Klasse `…konfiguration.common.impl.ReloadablePropertyKonfiguration` stellt über das Interface `de.bund.bva.pliscommon.konfiguration.common` einen typsicheren Zugriff auf die Konfigurationsparameter zur Verfügung.

Alle Parameter aus den in der Liste aufgeführten Property-Dateien werden der Anwendung als eine gemeinsame Sicht aller Konfigurationsparameter zur Verfügung gestellt. Sind Parameter in mehreren Dateien aufgeführt, so überschreiben Werte aus Dateien, die in der Liste hinten stehen, solche von Dateien, die zuvor aufgelistet wurden.

Die Konfigurations-Bean wird den Komponentenfassaden (z.B. MeldungImpl) per Spring-Dependecy-Injection bereitgestellt. Dazu muss die Komponente einen Setter für das Interface anbieten:

```
public void setKonfiguration(Konfiguration konfiguration)
```

Für den Fall, dass sehr viele Konfigurationsparameter benötigt werden, können komponentenspezifische Konfigurations-Beans verwendet werden.

Der Zugriff auf einzelne Konfigurationsparameter erfolgt dann über Methoden des Konfigurations-Interfaces, z.B.

```
String url = konfiguration.getAsString
(KonfigurationSchluessel.SERVICE_URL);
```

Das Interface bietet für verschiedene Datentypen (String, Integer, Long, Double und Boolean) jeweils typsichere Zugriffsmethoden an. Für jeden Datentyp wird zusätzlich eine Methode angeboten, welche die Übergabe eines Default-Werts ermöglicht. Dieser wird verwendet, falls der Konfigurationsparameter nicht in der Konfigurationsdatei vorhanden ist. Wird die Variante ohne Default-Wert aufgerufen und ein Konfigurationswert nicht vorhanden sein, wird eine Exception geworfen.

4.3.1.3 Handhabung von Default-Werten

Default-Werte können entweder explizit im Java-Code oder in eigenen Property-Dateien im Verzeichnis „`resources/default-config/`“ abgelegt werden. Werte, für die kein Default-Wert in Java hinterlegt ist, werden in eine Property-Datei in „`resources/default-config`“ übernommen, sofern sie nicht umgebungsabhängig sind. Typischerweise existiert für jede Default-Property-Datei eine gleichnamige Datei im Config-Verzeichnis, in welcher der Betrieb die Default-Werte überschreiben kann.

Die Ablage von Properties in den Dateien in „`resources/default-config`“ hat so zum großen Teil einen Dokumentationscharakter. Hier ist auf einen Blick ersichtlich, welche Parameter existieren und welches die Default-Werte sind. Die Ablage von Default-Werten in den Java-Klassen ist nur dann zu verwenden, wenn die Anwendung auch ohne eine entsprechende explizite Konfiguration lauffähig sein soll und das Fehlen der Einstellung keinesfalls einen Fehler erzeugen soll.

Für systemabhängige Werte dürfen keine Default-Werte hinterlegt werden. Wichtig ist, dass die Anwendung die Existenz dieser Werte (z.B. URLs zu genutzten Services) bereits bei der Initialisierung prüft. Dadurch wird vermieden, dass das Fehlen von Einstellungen erst bei späteren Zugriffen erkannt wird.

Konfigurationsparameter, die nicht zur Laufzeit änderbar sind (siehe Kapitel 4.4), können zur Performance-Optimierung in Instanzvariablen gehalten werden. Dabei kann auch gleich das Vorhandensein der Einstellung geprüft werden.[30] D.h. die Komponente liest bereits bei Ihrer Initialisierung den Parameterwert aus und speichert ihn in einer Instanzvariablen.

4.3.1.4 Deployment von Konfigurationsdateien

Für das Deployment von Konfigurationen ist zu beachten, dass der Resources-Ordner und der Config-Ordner in den Klassenpfad der Anwendung kopiert werden. Der Ordner „`config`“ muss nach dem Deployment ungepackt auf dem Dateisystem liegen, er darf z.B. nicht in ein Jar verpackt werden.

Der Inhalt des Resources-Ordners wird beim Deployment in das Verzeichnis „`/classes/resources`“ kopiert. Der Config-Ordner wird beim Deployment aus der eigentliche Anwendung herausgezogen und der Inhalt in „`/etc/<Anwendungsname>`“ abgelegt. Zusätzlich wird ein symbolischer

[30] Ein Beispiel findet sich in de.bund.bva.cd.registercd.core.meldung.impl.AwfCdErworben.

Link von „/classes/config“ auf „/etc/<Anwendungsname>/“ angelegt, so dass auch diese Inhalte Teil des Klassenpfads der Anwendung sind. Details dazu können dem Konzept [DeploymentKonzept] entnommen werden.

In einigen Fällen wird die für die Entwicklung benötigte Konfiguration von der Release-Version abweichen. Für jede Konfiguration aus „config“ kann es eine Entwicklungs- und genau eine Release-Variante geben: Es werden keine umgebungsabhängigen Varianten in den Sourcen abgelegt. Für die Release-Varianten wird in der Entwicklung ein Unterordner:

```
src/main/resources/config/release
```

angelegt. Beim Bauen des Release-Pakets werden alle Dateien aus dem Release-Unterordner in den übergeordneten Ordner verschoben und der Release-Ordner gelöscht. Der Config-Ordner enthält dann die Release-Konfigurationen. Diese werden wie oben beschrieben deployt.

Beim Deployment einer Anwendung werden alle auf dem Zielsystem liegenden Dateien des Resources-Ordners überschrieben. Die Dateien aus dem config-Ordner werden beim Deployment nicht überschrieben. Neue Parameter müssen dem Betrieb mitgeteilt werden. Für nicht systemabhängige Werte wird ein Default entweder im Java-Code oder in einer Property-Datei aus `resources/default-config` ausgeliefert. Für systemabhängige Werte existiert kein Default, diese werden aber bereits während der Initialisierung der Anwendung geprüft (siehe Kapitel 4.3.1.3). Fehlende Einstellungen werden so beim Programmstart erkannt.

4.3.2 Datenbank-basierte Konfigurationen

Konfigurationen, welche durch die Anwendung geschrieben werden, sind in der Datenbank abzulegen. Die Tabellen hierfür sind Teil des Datenmodells der Anwendung. Der Zugriff erfolgt genau wie der auf die übrigen Entitätstypen. Für weitere Details siehe [DatenzugriffDetailkonzept].

4.4. Konfigurationsänderungen zur Laufzeit

Betriebliche Konfigurationen werden in Dateien gespeichert und nur beim Starten der Anwendung geladen. Im Normalfall werden Konfigurationsparameter beim Start der Anwendung ausgelesen und in Instanzvariablen gehalten.

Sollen Konfigurationsparameter zur Laufzeit änderbar sein, müssen besondere Vorkehrungen getroffen werden:

- Konfigurationsänderungen gelten nicht zeitgleich für den gesamten Cluster, es muss daher ausgeschlossen werden, dass kurzzeitige Konfigurationsunterschiede zwischen den einzelnen Knoten zu fachlichen oder technischen Inkonsistenzen führen.
- Konfigurationsparameter, für die Änderungen zur Laufzeit vorgesehen sind, werden im Betriebshandbuch gesondert ausgewiesen.

- Solche Konfigurationsparameter werden vorzugsweise nicht in Instanz-Variablen gehalten, sondern bei jeder Verwendung aus der Konfigurations-Bean ausgelesen.
- Falls aufwändige Initialisierungen bei Konfigurationsänderungen durchgeführt werden müssen, kann die entsprechende Komponente sich als Listener bei der Konfigurations-Bean registrieren und so aktiv über Konfigurationsänderungen informiert werden (siehe Kapitel 4.4.2).

4.4.1 Konfigurationsdateien auf Änderungen prüfen

Damit die Anwendung Änderungen an betrieblichen Konfigurationsdateien erfährt, wird ein Polling auf die betrieblichen Konfigurationsdateien durchgeführt. Dazu implementiert die Konfigurations-Klasse `….pliscommon.konfiguration.common.impl.ReloadablePropertyKonfiguration` das Interface `….pliscommon.konfiguration.common.ReloadableKonfiguration`.

Die vom Interface deklarierte Methode

```
public boolean checkAndUpdate();
```

sorgt beim Aufruf dafür, dass alle Konfigurationsdateien auf Änderungen geprüft und bei Bedarf neu geladen werden. Änderungen werden an Hand des Änderungszeitstempels der Dateien festgestellt.

Die zuvor genannte Methode muss regelmäßig aufgerufen werden. Dazu wird per Spring ein Timer erzeugt, der die Methode der Konfigurations-Bean aufruft:

```
<bean id="konfigurationPollingTask"
class="org.springframework.scheduling.timer.ScheduledTimerTask"
    scope="singleton">
    <!-- wait 5 seconds before starting repeated execution -->
    <property name="delay" value="5000" />
    <property name="period" value="${konfiguration.reload.interval}" />
    <property name="timerTask">
        <bean
class="org.springframework.scheduling.timer.MethodInvokingTimerTaskFactory
Bean">
            <property name="targetObject" ref="konfiguration" />
            <property name="targetMethod" value="checkAndUpdate" />
        </bean>
    </property>
</bean>
```

Dieser Task wird der aus Kapitel 3.3.5.2 bekannten TimerFactory hinzugefügt:

```
<bean id="timerFactory"
class="org.springframework.scheduling.timer.TimerFactoryBean">
    <property name="scheduledTimerTasks">
        <list>
```

```
            …
            <ref bean="konfigurationPollingTask" />
        </list>
    </property>
</bean>
```

Die beschriebene Spring-Konfiguration ist in der Vorlage-Anwendung beispielhaft umgesetzt und kann in [VorlageAnwendung]/cd-register/src/main/resources/resources/spring/querschnitt.xml angesehen werden.

4.4.2 Reagieren auf Konfigurationsänderungen

Wie zu Beginn des Abschnittes beschrieben, sollen Konfigurationsparameter, für die Änderungen zur Laufzeit zugelassen sind, vorzugsweise bei jeder Verwendung aus der Konfigurations-Bean ausgelesen werden. Somit wird automatisch immer der aktuelle Wert verwendet.

In einigen Fällen sind auf Grund von Konfigurationsänderungen jedoch aufwändige Initialisierungen notwendig. Ein Beispiel hierfür wäre das Reinitialisieren von Connection-Pools, wenn eine URL geändert wurde. In solchen Ausnahmefällen kann daher das im Folgenden beschriebene und von der Konfigurationsbibliothek realisierte Listener-Pattern angewandt werden.

Um bei Änderungen informiert zu werden, registriert sich die betroffene Komponente als Listener bei der Konfigurations-Bean. Dafür bietet das von der Konfigurations-Bean implementierte Interface `….pliscommon.konfiguration.common.ReloadableKonfiguration` die Methode an:

```
public void addKonfigurationChangeListener
(KonfigurationChangeListener listener);
```

Die Komponente muss ihrerseits das Interface `….pliscommon.konfiguration.common.KonfigurationChangelistener` implementieren und die zuvor genannte Methode aufrufen. Als Parameter wird die eigene Instanz übergeben.

Bei Änderung der Konfiguration ruft die Konfigurationsbibliothek nun automatisch die Methode

```
public void onKonfigurationChanged(Set<String>
changedKeys);
```

der registrierten Listener auf.[31] Als Aufruf-Parameter werden die Schlüssel aller geänderten Konfigurationsparameter übergeben. Ein Beispiel dafür

[31] Zu beachten ist, dass der Methoden-Aufruf im Thread-Kontext des Timers, der die Konfigurationsdateien überwacht, erfolgt.

findet sich in der Vorlage-Anwendung in der Klasse `…registercd.core.admin.impl.AdminImpl`.

4.5. Spezielle Konfigurationen

In diesem Kapitel werden einige spezielle Aspekte der Konfiguration bzw. Konfigurierbarkeit der entwickelten Anwendungssysteme beschrieben. Dazu gehört z.B. die Konfiguration von Frameworks oder die Konfiguration des Tomcat-Kontexts.

Eine detaillierte Beschreibung der Konfiguration einzelner Frameworks findet sich in deren jeweiliger Dokumentation.

4.5.1 Spring-Konfiguration

Spring wird über eine Reihe von XML-Dateien konfiguriert. Da Spring als Container zur Konfiguration einer Reihe anderer Frameworks verwendet wird (z.B. Hibernate), wird hierüber letztendlich auch die Konfiguration dieser Frameworks durchgeführt. Die Spring-Konfigurationen werden im Resources-Ordner abgelegt:

```
src/main/resources/resources/spring/
```

Spring unterstützt es, über einen Platzhalter-Mechanismus einzelne Parameter aus den XML-Dateien in Property-Dateien auszulagern. Dieser Mechanismus wird benutzt, um einzelne Properties durch den Betrieb konfigurierbar zu machen. Dazu wird eine Property-Datei im Config-Ordner abgelegt, welche von Spring gelesen wird. Die Inhalte dieser Datei werden in die XML-Konfiguration integriert. Nähere Details zu dem Platzhalter-Mechanismus sind in [AnwendungskernDetailkonzept] nachzulesen.

4.5.2 Log4j-Konfiguration

Die eigentliche Log4j-Konfiguration wird in [LoggingKonzept] beschrieben. Aus Sicht des Konfigurationskonzeptes wird lediglich vorgegeben, die Konfigurationsdatei log4j.properties als betriebliche Konfiguration im Config-Ordner abzulegen.

Die Log4-Konfiguration wird nicht über die Konfigurationsbibliothek verwaltet, sondern direkt von der Log4j-Bibliothek verwendet. Damit Log-Level zur Laufzeit änderbar sind, wird der von Log4j bereitgestellte Listener verwendet. Details dazu sind im Konzept [LoggingKonzept] nachzulesen.

4.5.3 Web-Kontext-Konfiguration (web.xml)

Die Web-Kontext-Konfiguration `web.xml` ist eine statische Konfiguration. Hierin dürfen keine betrieblichen Parameter aufgenommen werden. Die Datei kann aus technischen Gründen nicht im Resources-Ordner abgelegt werden. Sie wird daher direkt im Ordner `WEB-INF/` abgelegt.

4.5.4 Tomcat-Kontext-Konfiguration (context.xml)

Tomcat verwaltet für jede Webanwendung eine eigene Konfiguration. In dieser Datei werden in einem <context>-Tag spezielle Einstellungen für

diese Webanwendung konfiguriert. Wird keine Datei ausgeliefert gelten die Standardeinstellungen von Tomcat.

Für das Deployment (siehe [DeploymentKonzept]) ist es notwendig, dass symbolische Links im Dateisystem aufgelöst werden. Dazu wird eine Datei (`META-INF/context.xml`) mit jeder Webanwendung ausgeliefert. Für die Vorlage-Anwendung hat die Datei folgenden Inhalt:

```
<Context path="/cd-register" allowLinking="true" />
```

Zur Vereinfachung des Deployments wird die Datei nicht im Tomcat-Installationsverzeichnis abgelegt, sondern als Datei „context.xml“ im Verzeichnis „META-INF“ der Webanwendung abgelegt. Dort wird die Datei automatisch von Tomcat gefunden und verwendet.

4.5.5 Umsetzen des Auskunftsmodus

In jeder Fachanwendung muss ein Auskunftsmodus implementiert werden. In diesem Modus dürfen keine bestandsverändernden Aktionen möglich sein. Rein lesende Operationen sind weiterhin erlaubt.

Der Modus wird genutzt, um während längerer Datenmigrationen Änderungen der Bestandsdaten zu unterbinden, während gleichzeitig Auskünfte möglich sind.

Die Anforderung wird durch Einfügen eines Konfigurationsparameter „anwendung.auskunftsmodus.aktiviert“ in die betriebliche Konfigurationsdatei umgesetzt.[32] Die Option kann die Ausprägungen „false“ (Alle Funktionen sind aktiv) und „true“ (Schreibzugriffe sind deaktiviert) annehmen.

In den Komponenten, die Schreibzugriffe implementieren, wird vor dem Aufruf der Anwendungsfälle geprüft, ob der Parameter auf „true“ gesetzt ist. Ist dies der Fall, wird eine technische Exception vom Typ „`KomponenteDeaktiviertException`“ geworfen.

Im Falle eines aktivierten Auskunftsmodus sollten die Nutzer einer grafischen Benutzeroberfläche frühzeitig informiert werden – insbesondere nicht erst durch die technische Exception, nachdem sie alle Daten erfasst und abgeschickt haben. Je nach Aufbau der Dialoge kann hierbei ein Hinweis auf den Masken dargestellt werden, oder ganze Dialoge durch einen Hinweisdialog ersetzt werden.

Bevor eine Anwendung in den Auskunftsmodus versetzt werden kann, sind alle nutzenden Anwendungen ebenfalls in den Auskunftsmodus zu versetzen. Zum Wiederherstellen des vollen Funktionsumfangs wird in umgekehrter Reihenfolge vorgegangen, d.h. schreibende Zugriffe werden zunächst in den genutzten Anwendungen wieder erlaubt und danach die nutzenden Anwendungen umgestellt.

[32] Im Vorlage-Register ist dies die Datei src/main/resources/config/cd-register.properties.

5. 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 .

[DeploymentKonzept]

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

[IsyFactReferenzarchitektur]

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

[JMXBestPrac]

Java Management Extensions (JMX) - Best Practices
http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/best-practices.jsp .

[JMXParam]

Monitoring and Management Using JMX
http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html .

[JMXPatterns]

Design patterns for JMX and application manageability - A guide for Developers
http://devresource.hp.com/drc/resources/jmxds/index.jsp.

[LoggingKonzept]

Konzept Logging
20_Bausteine\Logging\Konzept_Logging.pdf.

[VorlageAnwendung]

Beispielimplementierung „Vorlage-Anwendung“
Wird auf Anfrage bereitgestellt.

6. Abbildungsverzeichnis

7. Tabellenverzeichnis