Bundesverwaltungsamt

IsyFact-Standard

Nutzerdokumentation Sicherheit

Version 1.14
29.08.2016

IsyFact Standards

Bundesverwaltungsamt
Der zentrale Dienstleister des Bundes

„Nutzerdokumentation Sicherheit“
des Bundesverwaltungsamts ist lizenziert unter einer
Creative Commons Namensnennung 4.0 International Lizenz.

Ansprechpartner:

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

Dokumentinformationen

Dokumenten-ID:	Nutzerdokumentation_Sicherheit.docx

Java Bibliothek / IT-System

Name	Art	Version
isy-sicherheit	Bibliothek	1.3.x
plis-sicherheit-cams	Bibliothek	1.4.x
isy-aufrufkontext	Bibliothek	1.3.x

Inhaltsverzeichnis

1. Einleitung

IsyFact-konforme Anwendungen sind gegen nicht autorisierte Nutzung zu schützen. Authentifizierung und Autorisierung finden nach den Vorgaben des Berechtigungskonzepts [Berechtigungskonzept] statt, welches wesentliche Aspekte der Sicherheitsarchitektur beschreibt. Das Berechtigungskonzept muss für jede Anwendungslandschaft individuell erstellt werden und ist daher kein Bestandteil der IsyFact.

1.1. Inhalt dieses Dokuments

Dieses Dokument beschreibt wie die Komponente Sicherheit verwendet wird, um die **Autorisierung innerhalb einer Fachanwendung** umzusetzen. Es wird davon ausgegangen, dass die **Authentifizierung** des Benutzers über eine zentrale Komponente (Access Manager) der Anwendungslandschaft erfolgt, in die die Anwendung eingebettet ist. Nach der Authentifizierung eines Benutzers über den zentralen Access-Manager der Anwendungslandschaft wird dessen Anfrage an die Fachanwendung weitergeleitet und hier – je nach Schutzbedarf und Funktionalität der Anwendung – autorisiert. Dies erfolgt mithilfe der von der IsyFact bereitgestellten Komponente Sicherheit.

1.2. Aufbau und Zweck des Dokuments

Dieses Dokument richtet sich an Entwickler und Chefdesigner, die eine Anwendung gemäß den Vorgaben der IsyFact mit Autorisierungsfunktionalität ausstatten müssen. Es beschreibt, wie und in welchen Teilen einer Anwendung die Autorisierung umzusetzen ist.

Dazu wird zunächst in einem Überblick erläutert, wie die Sicherheitsarchitektur auf Ebene der Anwendungslandschaft und auf Ebene einer einzelnen Anwendung aussieht.

Im Abschnitt „Verwendung von Rollen“ wird erläutert, wie für eine neue Anwendung das Rollenmodell definiert, in das Benutzerverzeichnis eingespielt und für eine Anwendung konfiguriert wird.

Im Hauptkapitel „Autorisierung innerhalb einer Anwendung“ wird auf die Umsetzung der Autorisierung in der Anwendung eingegangen. Dieser Abschnitt erläutert die Verwendung der Komponente Sicherheit innerhalb einer Anwendung.

Im Kapitel 5 werden weitere Vorgaben zur Verwendung der Komponente Sicherheit gemacht. Hier wird der Aufruf von Nachbarsystemen, Behandlung der Korrelations-ID und der Test von Anwendungen erläutert.

1.3. Vorlageanwendung

Viele der in diesem Dokument beschriebenen Vorgaben sind in den Vorlage-Anwendungen [VorlageAnwendung] umgesetzt.

1.4. Prinzipien der Sicherheitsarchitektur

Die Sicherheitsarchitektur einer konkreten Anwendungslandschaft wird zum großen Teil innerhalb des spezifischen Berechtigungskonzepts für diese Landschaft festgelegt. Sie kann daher nicht Teil des vorliegenden Nutzerdokumentation der Sicherheitskomponente sein.

Auf der technischen Ebene macht jedoch die IsyFact-Referenzarchitektur bestimmte Vorgaben für die Sicherheitsarchitektur und für die Prinzipien, nach denen sie aufgebaut ist.

Die konkrete Sicherheitsarchitektur einer Anwendungslandschaft muss nach folgenden Prinzipien gestaltet sein:

- Authentifizierung und grobgranulare Autorisierung an den Außenschnittstellen des Systems (Portal, Service-Gateway, Native-Client-Serviceproxy). Hier finden der Session-Aufbau und die Rollenermittlung durch den Access-Manager statt.

- Grobgranulare Autorisierung vor den GUI-Schnittstellen der Anwendungen, z.B. mittels Servlet-Agent des Access-Managers.

- Feingranulare Autorisierung in den Schnittstellen und im Anwendungskern der Anwendungen über die Komponente Sicherheit.

2. Das rollenbasierte Berechtigungskonzept

Die Vergabe von Rollen ist *das* Mittel der Benutzeradministration, um Benutzer der Anwendungslandschaft mit Berechtigungen auszustatten. Die Vergabe von Rollen an einen Benutzer (menschliche und technische) erfolgt in der Anwendung Benutzerverzeichnis.

Es ist konzeptionell beabsichtigt, dass die Administration per Rollen recht grobgranular erfolgt. Eine administrative Vergabe feingranularer Rechte ist konzeptionell nicht erwünscht. Die individuelle Zuordnung von Rechten zu Benutzern ist daher im Benutzerverzeichnis nicht möglich.

Rechte werden ausschließlich indirekt über Rollen (in denen sie enthalten sind) den Benutzern zugeordnet. Welche Rechte einer Rolle zugeordnet sind, wird innerhalb der Konfiguration einer Anwendung definiert und ist damit Teil der Software.

Ein Beispiel: Bietet eine Anwendung X zwei Dialoge zur Administration von Anwendungseigenschaften, so könnten diese Dialoge über eine Rolle AnwendungX_Administrator abgesichert werden. Innerhalb der Anwendung ist Dialog 1 mit Recht AdministrierenDialog1 und Dialog 2 mit dem Recht AdministrierenDialog2 abgesichert. Im Benutzerverzeichnis kann die Rolle AnwendungX_Administrator an einen Anwender vergeben werden (grobgranular). Innerhalb der Anwendungskonfiguration werden die beiden Rechte konfiguriert und der Rolle AnwendungX_Administrator zugeordnet. Alle Anwender mit der Rolle AnwendungX_Administrator sind somit innerhalb der Anwendung autorisiert die beiden Admin-Dialoge zu verwenden.

Wie Rollen und Rechte spezifiziert, bereitgestellt, konfiguriert und Benutzern zugeordnet werden, wird im Folgenden beschrieben.

2.1. Spezifikation der Rollen

Die Grundlage für die Umsetzung einer angemessenen Absicherung einer Anwendung wird in der Spezifikation der Anwendung gelegt.

Hier werden zunächst in geeigneter Granularität Rechte definiert, die zur Benutzung bestimmter Funktionalität der Anwendung berechtigt.

Diese Rechte werden fachlichen Rollen zugeordnet, die den Benutzern der Anwendung zugeordnet werden können. Dies sind Rollen die entweder pauschal den Zugriff auf die Anwendung erlauben, oder im Sinne einer fachlichen Sachbearbeiter-Rolle die Nutzung ausgewählter Anwendungsfälle ermöglicht. Zu jeder fachlichen Rolle wird festgelegt:

> Name: Der interne Name der Rolle, wie er für die Autorisierung im Access-Manager und innerhalb von Anwendungen zur Überprüfung bereitgestellt wird.
>
> Der Rollenname sollten dem folgenden Schema entsprechen: <Anwendungskürzel>_<Zweck/Funktionalität>[_<Zugang>].

Das Suffix <Zugang> ist optional, sollte jedoch wenn möglich verwendet werden. Folgende Werte sind als <Zugang> sinnvoll:

- Schnittstellennutzer (für technische Rollen, die zur Verwendung einer internen Service-Schnittstelle berechtigt)
- Webservicenutzer (für technische Rollen, die zur Verwendung einer Web-Service-Schnittstelle eines Service-Gateways berechtigt)
- Nutzer (für fachliche Rollen, die zur Nutzung der Oberfläche einer Anwendung berechtigt)
- Servicenutzer (für fachliche Rollen, die zur Verwendung eines Service-Gateways berechtigt)
- Batch (für fachliche Rollen, die zur Ausführung eines Batches zu einer Anwendung berechtigt)
- Launcher (für fachliche Rollen, die zur Ausführung eines internen Tasks einer Anwendung berechtigt)

- Label: Der Name der Rolle, wie sie in der Benutzeroberfläche des Benutzerverzeichnisses anzuzeigen ist. In der Regel identisch mit dem Namen der Rolle. Eine Umbenennung ist nur dann sinnvoll, wenn die Rollenvergabe durch den Administrator dadurch intuitiver wird.

- Typ: Eine Rolle kann fachlich oder technisch sein. Nur fachliche Rollen können administriert werden. Technische Rollen können fachlichen Rollen untergeordnet werden (siehe Untergeordnete Rollen).

- Enthaltene Rechte: Die Ausstattung einer fachlichen Rolle mit Rechten beschreibt den Funktionsumfang, den der Rolleninhaber bei der Nutzung der Anwendung vorfindet.

- Untergeordnete Rollen (optional): Fachliche Rollen können technische Unterrollen besitzen. Wofür wird das benutzt? Berechtigt eine fachliche Rolle für einen Anwendungsfall der die Dienste eines Nachbarsystems (z.B. eines Schlüsselverzeichnisses) verwendet, so wird die von diesem Nachbarsystem zur Verwendung der Serviceschnittstelle definierte technische Rolle der fachlichen Rolle untergeordnet.

- Sichtbarkeit der Rolle: die Sichtbarkeit der Rollen bei der Rollenzuordnung an Anwender, externe Systeme und interne Systeme kann eingeschränkt werden, um die Administration zu vereinfachen.

- Beschreibung der Rolle: Eine kurze Beschreibung der Rolle in einer fachlichen Sprache, die für Dienstleister und Administration verständlich ist, erleichtert die Nutzung der Rolle.

Bietet die Anwendung Funktionalität über Service-Schnittstellen an, so ist die Nutzung jeder Service-Schnittstelle zumindest durch eine technische Rolle abzusichern. Diese Rollen werden nicht direkt an Benutzer vergeben, sondern fachlichen Rollen anderer Anwendungen untergeordnet.

Wenn die Anwendung fachliche oder technische Batches enthält, dann müssen für diese Batches in der Spezifikation entsprechende „interne Systeme“ definiert werden. Die Systemnamen sollten dem folgenden Schema entsprechen: <Anwendungskürzel>_BAT_<Batchname in CamelCase>. Für jedes interne System müssen die fachlichen Rollen definiert werden, die diesem System zugeordnet sind.

2.2. Konfiguration von Rollen in der Anwendung

Für jede fachliche oder technische Rolle, die innerhalb einer Anwendung zum Zwecke der Autorisierung eines Benutzers ausgewertet werden soll, müssen in der Anwendung Anwendungsrollen konfiguriert und die erforderlichen Rechte zugeordnet werden. Dies erfolgt in der Datei /resources/sicherheit/rollenrechte.xml der Anwendung. (Die zugehörige Schema-Datei RollenRechteSchema.xsd findet sich im Anhang 7.2.)

```
<tns:Anwendung AnwendungsId="AnwendungsId"
    xmlns:tns="http://www.example.org/RollenRechteSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/RollenRechteSchema
RollenRechteSchema.xsd ">

    <!-- Definition von in der Anwendung verwendeten Rechten -->
    <tns:rechte>
        <tns:rechtId Id="Recht_A" />
    </tns:rechte>

    <tns:rechte>
        <tns:rechtId Id="Recht_B" />
    </tns:rechte>

    <!-- Definition von in der Anwendung verwendeten Rollen -->
    <tns:rollen RolleId="Rolle_A">
        <tns:rechtId Id="Recht_A" />
    </tns:rollen>

    <tns:rollen RolleId="Rolle_AB">
        <tns:rechtId Id="Recht_A" />
        <tns:rechtId Id="Recht_B"/>
    </tns:rollen>
</tns:Anwendung>
```

Abbildung 1: Definition von Rollen und Rechten

Die Konfiguration muss die folgenden Anforderungen erfüllen:

- Es sind alle in der Anwendung überprüften Rechte definiert. Wird in der Anwendung ein Recht überprüft, das hier nicht definiert ist, so wird ein Fehler erzeugt. Die Konfiguration gibt also verlässlich Auskunft darüber, welche Rechte in der Anwendung überprüft werden.

- Es sind alle den Benutzern der Anwendung angebotenen Rollen definiert. Die Konfiguration gibt also verlässlich Auskunft darüber, welche Rollen durch die Anwendung angeboten werden. Innerhalb jeder Rolle werden gemäß Spezifikation die zugeordneten Rechte festgelegt. Rollen können überlappende Teilmengen von Rechten enthalten.

Die Komponente Sicherheit ermöglicht innerhalb der Anwendung eine Autorisierung nur auf Basis von Rechten, nicht von Rollen. Jeder Rolle muss also zumindest ein Recht zugeordnet werden, um eine Autorisierung durchführen zu können. Werden im Lebenszyklus der Anwendung weitere Rollen (für neu hinzukommende Akteure) definiert (und mit einer neuen Kombination von Rechten ausgestattet), so muss die Anwendung dafür nicht umprogrammiert werden.

2.3. Bereitstellen von Rollen im Benutzerverzeichnis

Damit Rollen auch an Benutzer im Benutzerverzeichnis vergeben werden können, müssen die Rollen in das Benutzerverzeichnis eingespielt werden.

Dies erfolgt für Testsysteme und das Produktivsystem (im Rahmen von Inbetriebnahmen) per Import-Batch, bei dem ein Excel-Dokument mit zu importierenden Rollen in das Benutzerverzeichnis geladen wird. Der Import ist der einzige Weg, um Rollen hinzuzufügen.

Mit dem Batch ist es möglich Rollen hinzuzufügen (Add), zu verändern (Upd) oder zu löschen (Del).

Das Excel-Dokument zum Einspielen von Rollen benötigt ein Arbeitsblatt „Rollen“ und hat folgendes Format:

Aktion	Rollenlabel	Rollenname	Rollentyp	Unterrollen
Add	BNVZ_Rolle1	BNVZ_Rolle1	fachlich	BHVZ_Schnittstellennutzer
Upd	BNVZ_Rolle2	BNVZ_Rolle2	fachlich	BHVZ_Schnittstellennutzer, SVZ_Schnittstellennutzer
Del		BNVZ_Rolle3		

Abbildung 2: Einspielen von Rollen in das Benutzerverzeichnis (Beispiel)

Zusätzlich zu den oben abgebildeten Spalten enthält das Excel-Dokument die folgenden weiteren Spalten, deren Inhalt bereits in Kapitel 2.1 beschrieben wurde:

- **Rollenbeschreibung**
- **SichtbarkeitAnwender**
- **SichtbarkeitSystemeExtern**
- **SichtbarkeitSystemeIntern**

Nach der letzten zu importierenden Zeile des Arbeitsblatts sollte eine Zeile mit der Aktion „End“ eingefügt werden. Dies verbessert die Performance beim Import. Bei allen anderen Werten in der Spalte „Aktion“ wird die Zeile ignoriert.

Zusätzlich kann das Excel-Dokument ein weiteres Tabellenblatt „Systeme“ enthalten. Dieses folgt demselben Schema zum Hinzufügen (Add), Ändern (Upd) und Löschen (Del) von Einträgen. Es dient dazu, die Systeme (interne und ggf. auch externe) zu pflegen, wie in Kapitel 2.1 beschrieben. Systeme sind im Benutzerverzeichnis spezialisierte Benutzer. Sie haben daher alle Attribute der Entitäten ETY_Benutzer, ETY_System und ETY_Rollenträger.

Das Tabellenblatt „Systeme“ enthält die folgenden Spalten, deren Inhalte in den Attributbeschreibungen im Datenmodell des Benutzerverzeichnisses erklärt werden:

- **Aktion**: „Add“, „Upd“, „Del“ oder „End“
- **InterneKennung**
- **Name**
- **Status**: „gueltig“ oder „ungueltig“
- **BHKNZ**: Kennzeichen der Organisation (Behördenkennzeichen) durch die das System genutzt wird.
- **Anbieter**
- **Intern**: „true“ oder „false“
- **PasswortPlain**: Passwort im Klartext, wird beim Import verschlüsselt
- **Anlagedatum**
- **PasswortLaeuftAb**: „true“ oder „false“ (für Systeme sinnvollerweise „false“)
- **PasswortLetzteAenderung**
- **PasswortMussGeaendertWerden**: „true“ oder „false“
- **Beschreibung**
- **LetzteAenderung**
- **LetzteAenderungDurch**
- **RollenDirekt**: Die kommagetrennte Liste der direkt zugeordneten fachlichen Rollen des Systems

Motivation für die Updatefunktion

Die Löschung einer bereits verwendeten Rolle kann große Auswirkungen auf die Rollenzuordnung des Benutzerbestandes haben, da diese allen besitzenden Benutzern weggenommen werden muss. Eine nachträgliche hinzugefügte Ersatzrolle müsste dann manuell administrativ den Benutzern wieder zugeordnet werden. Das ist nicht praktikabel. Daher wird für eine Aktualisierung einer Rolle das Ändern einer Rolle (Upd) angeboten. Die Rollendefinition wird dabei verändert, während die Rolle allen Benutzern und Nutzergruppen zugeordnet bleibt.

Einschränkungen für den Rollenimport

Folgende Einschränkungen bestehen beim Import von Rollen:

- Erzeugen einer Rolle:
    - Der Name der Rolle darf noch nicht vergeben sein.

  - Eine fachliche Rolle darf nur technische Unterrollen haben. Im Excel-Dokument referenzierte Unterrollen müssen im Datenbestand bereits bekannt sein, bzw. im Excel-Dokument weiter oben stehen.

- Ändern einer Rolle
  - Der Typ der Rolle (fachlich, technisch) kann nicht geändert werden.
  - Der (neue) Name der Rolle darf nicht bereits an eine andere Rolle vergeben sein.
- Löschen der Rolle: Handelt es sich um eine technische Rolle, so darf diese Rolle zum Zeitpunkt der Löschung nicht mehr in einer anderen Rolle als Unterrolle verwendet werden. Die fachliche Rolle ist zunächst explizit zu löschen.

2.4. Verwaltung des Rollen-Masters

Die Rollendefinition der Anwendungslandschaft (in Form eines Excel-Dokuments) ist ein zentral zu verwaltendes Dokument, welches zur Befüllung von Testumgebungen verwendet wird. Es repräsentiert den insgesamt verfügbaren Rollenvorrat über alle Anwendungssysteme. Das Dokument trägt den Namen **Rollen-Master**.

Änderungen am Rollenmodell im Rahmen von Wartungsarbeiten werden in dieses Dokument übertragen. Zum Einspielen einer Rollenänderung in ein produktiv- oder Testsystem wird jedoch ein passendes **Rollen-Delta** (ebenfalls Excel) verwendet, welches nach einer Inbetriebnahme gelöscht wird.

Die Koordination der Änderungen am Rollen-Master obliegt dem Release-Verantwortlichen.

2.5. Releases und Rollen-Deltas

Für jedes Release, welches Änderungen an dem Rollenbestand der Anwendungslandschaft vornimmt, werden ein oder mehrere Rollen-Deltas aufbauend auf dem Rollen-Master erstellt, die das Rollenmodell vom Ist-Zustand in den Soll-Zustand überführen. Die Rollen-Deltas werden in den Sourcen des zugehörigen IT-Systems im Verzeichnis `/src/main/skripte/sicherheit` abgelegt

Diese Rollen-Deltas werden auf Testumgebungen im Rahmen der Integrationstests getestet und mit dem Release ausgeliefert. Die Reihenfolge, in der sie eingespielt werden müssen, wird im Releaseletter für die Rollendeltas definiert. Sie hängt von den Abhängigkeiten der Systeme ab, die in den Releaselettern der jeweiligen Systeme beschrieben sind.

3. Einbindung der Komponente Sicherheit

Die Komponente Sicherheit ist eine Querschnittkomponente der IsyFact-Referenzarchitektur [IsyFactReferenzarchitektur]. Diese Komponente ist von jeder konformen Anwendung zur Autorisierung von Zugriffen und Vorgängen zu verwenden.

Für ein korrektes Funktionieren benötigt die Komponente Sicherheit die Komponente AufrufKontextVerwalter, deren Verwendung ebenfalls in diesem Kapitel erläutert wird.

3.1. Sicherheitsarchitektur einer Anwendung

3.1.1 Ziele

Die Ausgestaltung der Sicherheitsmechanismen für IsyFact-konforme Anwendungen hat das Ziel, die Autorisierung von Zugriffen auf IT-Systeme einer Anwendungslandschaft systematisch, einheitlich und einfach umzusetzen.

- Die **Systematik** und Vollständigkeit der Berechtigungsprüfungen wird dadurch erreicht, dass Berechtigungsprüfungen in den IT-Systemen an definierten Stellen und auf identische Weise stattfinden.

- Die **Einheitlichkeit** wird durch Bereitstellung der Komponente Sicherheit und Nutzungsvorgaben gewährleistet, die von allen IT-Systemen der Anwendungslandschaft zu verwenden sind. Berechtigungsprüfungen erfolgen innerhalb einer Anwendung immer über die Komponente Sicherheit.

- Die **Einfachheit** der Nutzung der Komponente Sicherheit wird durch weitgehende Transparenz bei der Initialisierung, kompakte Schnittstellen und deklarative (z.B. per Annotation) statt programmatischer Implementierung erreicht.

3.1.2 Prämissen

Aus den im Abschnitt 1.4 beschriebenen Prinzipien leiten sich die folgenden Randbedingungen für die Umsetzung der Berechtigungsprüfung innerhalb einer Anwendung ab:

- Anfragen die am Dialog einer Anwendung eingehen, sind immer bereits durch den Access-Manager authentifiziert. Der http-Header der Anfrage enthält die Identifikation des Benutzers und dessen Rollen. Die Informationen aus diesem Header werden in die Anwendung als AufrufKontext übernommen.

- Anfragen die an einer Service-Schnittstelle einer Anwendung eingehen, sind immer bereits durch den Access-Manager authentifiziert. Das mit der Anfrage an eine Anwendung als Parameter übergebene AufrufKontextTo enthält die Identifikation

des Benutzers und dessen Rollen und wird in der Anwendung als AufrufKontext verwendet.

- Prozesse, die unabhängig von eingehenden Anfragen (über GUI und Service) durch eine Anwendung gestartet werden, müssen zunächst einen (meist technischen) Benutzer gegen den Access-Manager authentifizieren, dessen Rollen ermitteln und diese Informationen als AufrufKontext in der Anwendung hinterlegen.

- Einem innerhalb der Logik- und Verarbeitungszone einer Anwendung übergebenen AufrufKontext kann vertraut und ohne erneute Rückfrage an den Access-Manager verwendet werden.

3.1.3 Software-Architektur

Die folgende Abbildung zeigt den logischen Aufbau für die Authentifizierung und für die Bereitstellung von Berechtigungsinformationen an die Komponenten einer Anwendung.

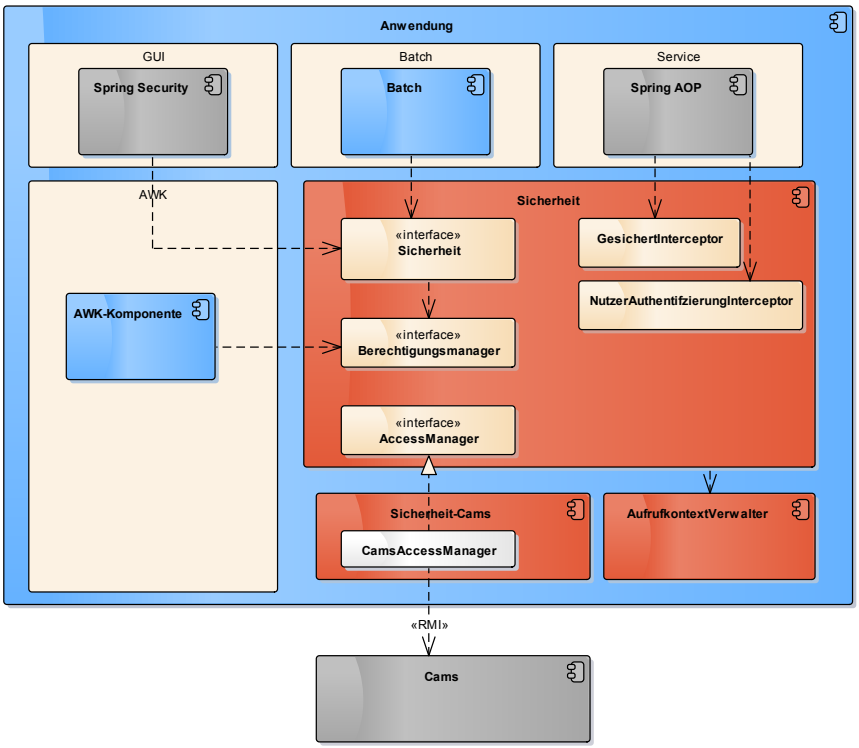

Abbildung 3: Software-Architektur der Berechtigungsprüfung

Im Folgenden werden die Aufgaben und grobe Funktionsweise der Komponenten für die Autorisierung von Anfragen in einer Fachanwendung erläutert.

Die Komponente AufrufKontextVerwalter stellt für eine laufende Anfrage Kontextinformationen zur Anfrage bereit, die in einem AufrufKontext hinterlegt werden. Das sind insbesondere die mit der Anfrage über die Außenschnittstelle eingehenden Informationen zum Benutzer und dessen Rollen, die Korrelations-ID und anwendungsspezifisch ggf. weitere Informationen. Die Komponente bringt Hilfsmittel zur transparenten Nutzung des AufrufKontextVerwalters mit. So wird kann über den StelltaufrufkontextBereitInterceptor der AufrufKontext bei Serviceaufrufen transparent über Spring AOP gesetzt werden. Weiterhin wird der Aufrufkontext durch die Komponente Sicherheit im Rahmen der Authentifizierung automatisch befüllt. Nach Initialisierung des AufrufKontextVerwalters für eine laufende Anfrage kann die Anwendung fortan transparent mit den im AufrufKontextVerwalter hinterlegten Benutzerinformationen arbeiten (ohne deren Herkunft zu kennen) und damit auch weitere Nachbarsysteme aufrufen.

Die Komponente Sicherheit bietet folgende Funktionen:

- für Service-Aufrufe werden Interceptoren angeboten, welche über Spring AOP eine deklarative Berechtigungsprüfung ermöglichen.

- für den Kontext der Anfrage stellt die Komponente einen Berechtigungsmanager zur Verfügung, der die Rollen des anfragenden Benutzers kennt. Die Informationen zum anfragenden Benutzer werden – falls vorhanden – aus dem AufrufKontextVerwalter entnommen. Die Fachkomponenten einer Anwendung nutzen den Berechtigungsmanager für spezielle Berechtigungsprüfungen, die nicht deklarativ über Annotationen erfolgen.

- Benutzer können anhand der übergebenen Benutzerkennung (und Passwort) authentifiziert werden. Dazu wird der Access-Manager angesprochen. Die gewonnenen Informationen werden im AufrufKontextVerwalter hinterlegt.

- Der AccessManager kann für verschiedene Berechtigungsquellen implementiert werden. Mit Sicherheit-CAMS wird eine Implementierung für den CAMS angeboten.

Die Authentifizierung und Autorisierung von Web-Zugriffen wird über Spring-Security durchgeführt. Die Integration von Spring-Security und Sicherheit werden in [WebGUIDetailkonzept] beschrieben.

3.2. Außensicht der Komponente Sicherheit

Im Folgenden wird die Schnittstelle der Komponente Sicherheit beschrieben.

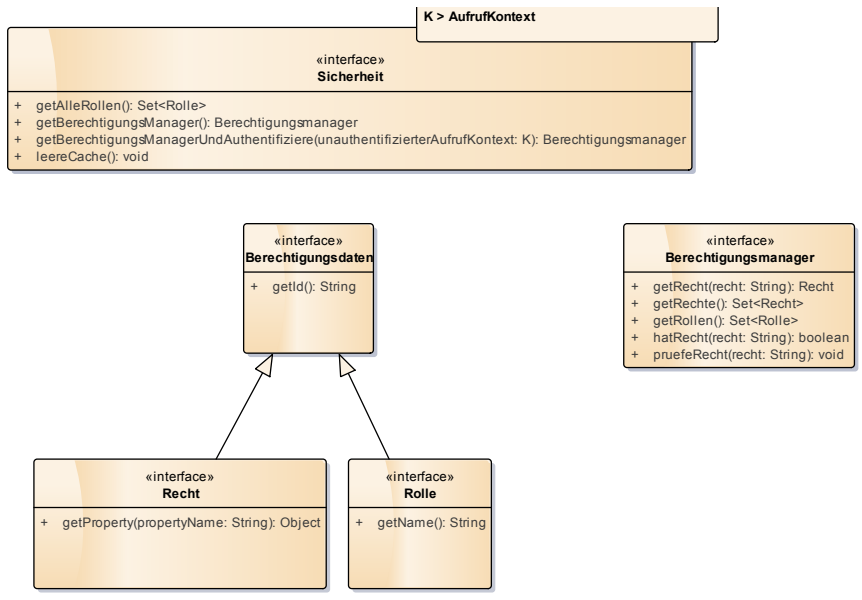

Abbildung 4: Außensicht der Komponente „Sicherheit“

Die Außensicht der Komponente enthält folgende Objekte:

Zentrales Interface für den Zugriff auf Rollen und Rechte eines Benutzers ist `Berechtigungsmanager`. Instanzen des Berechtigungsmanagers zur Autorisierung einer Anfrage werden über `Sicherheit` erzeugt.

Vom Berechtigungsmanager werden die Interfaces `Recht` und `Rolle` verwendet. Rollen werden über das Benutzerverzeichnis Benutzern zugewiesen. Berechtigungen sind anwendungsspezifisch und an Rollen gebunden. Diese Zuordnung erfolgt über die Konfigurationsdatei rollenrechte.xml.

Die Implementierung des zu verwendenden Aufrufkontexts richtet sich nach dem verwendeten AccessManager.

3.3. Erzeugen eines Berechtigungsmanagers

Die Komponente erzeugt für einen angegebenen Benutzer eine Instanz des Berechtigungsmanagers. Der Berechtigungsmanager ist ein Container für die Berechtigungsinformationen des Benutzers, also für Rechte und Rollen. Die Instanz des Berechtigungsmanagers kann auf mehrere Arten erzeugt werden:

```
getBerechtigungsManager()
```

Die Komponente Sicherheit ermittelt die Informationen zum Benutzer und zugehörige Rollen aus dem `AufrufKontextVerwalter` der Anwendung. Diese Methode soll verwendet werden, wenn der Benutzer bereits in der Informations- und Dienstezone authentifiziert wurde und die Benutzerinformationen (inklusive Rollen) in der Anwendung im `AufrufKontextVerwalter` vorliegen.

```
getBerechtigungsManagerUndAuthentifiziere
    (AufrufKontext unauthentifzierterAufrufkontext)
```

Die Komponente Sicherheit authentifiziert zunächst den Benutzer durch eine Anfrage am Access-Manager, dabei werden die zur Authentifizierung benötigten Informationen über ein AufrufKontext-Objekt übergeben. Für CAMS sind dies das Zertifikat (alternativ ZertifikatDn), Kennung und Passwort. Der AccessManager ermittelt dann die Rollen des Benutzers und diese werden im AufrufKontextVerwalter der Anwendung hinterlegt. Aus Sicht des CAMS erfolgt anschließend erfolgt sofort der Logout.

3.4. Verwendung des Berechtigungsmanagers

Der Berechtigungsmanager wird selten direkt im Programmcode verwendet, da die meisten Berechtigungsprüfungen deklarativ per Annotationen oder Webflow-Tags (siehe Abschnitt 4) umgesetzt werden.

Trotzdem kann eine Berechtigungsprüfung auch ausprogrammiert werden. Dafür stellte die Komponente Sicherheit den Berechtigungsmanager mit folgender Schnittstelle bereit:

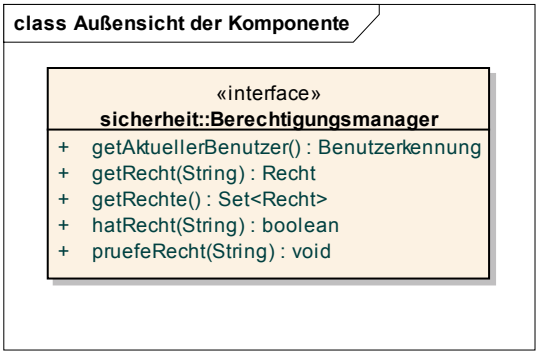

Abbildung 5: Schnittstelle des Berechtigungsmanagers

Die Klasse Berechtigungsmanager bietet die folgenden Methoden an:

getAktuellerBenutzer

Liefert die Benutzerkennung des Benutzers (Login und Behörden-/Organisationskennzeichen) für den der Berechtigungsmanager erzeugt wurde.

getRechte

Diese Methode liefert eine Liste aller Rechte des Benutzers.

getRecht

Diese Methode liefert zu einer ID das zugehörige Recht, falls der Benutzer es besitzt.

hatRecht

Diese Methode prüft, ob der Benutzer das angegebene Recht hat.

pruefeRecht

Diese Methode prüft, ob der Benutzer das angegebene Recht hat und löste eine AutorisierungFehlgeschlagenException aus, wenn das nicht der Fall ist.

3.5. Prüfung der Erreichbarkeit des Access Managers (Ping)

Die Komponente Sicherheit bietet über das Bean SicherheitAdmin die Möglichkeit die Verfügbarkeit des Nachbarsystems Access-Manager zu prüfen.

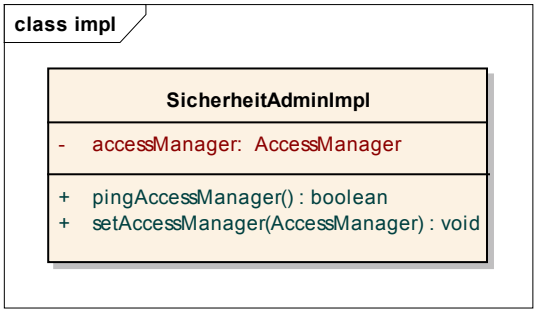

Abbildung 6: Schnittstelle von SicherheitAdmin

pingAccessManager

Es wird ein Ping gegen den Access-Manager durchgeführt, um dessen Erreichbarkeit zu prüfen. Diese Methode kann von nutzenden Anwendungen in die Watchdog-Test-Methode der Überwachung eingebunden werden.

3.6. Konfiguration

Zum Einbinden der Sicherheitskomponente und des AufrufKontextVerwalters sind wenige Konfigurationen erforderlich.

3.6.1 Maven-Konfiguration

Die Komponente Sicherheit wird per Maven-Dependency in das Projekt eingebunden:

```
<dependency>
    <groupId>de.bund.bva.pliscommon</groupId>
    <artifactId>plis-sicherheit-cams</artifactId>
    <version><aktuelle Version der Komponente Sicherheit></version>
</dependency>
```

Abbildung 7: Konfiguration für Maven

3.6.2 Spring-Konfiguration

Die benötigten Beans für Sicherheit, AufrufKontext und angebotener Annotationen werden in der separaten Spring-Konfigurationsdatei */resources/spring/querschnitt/sicherheit.xml* konfiguriert.

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:sec="http://www.springframework.org/schema/security"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-
3.2.xsd http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
  <!--  @StelltAufrufKontextBereit Annotation einschalten  -->
  <bean id="aufrufKontextInterceptor"
class="de.bund.bva.pliscommon.aufrufkontext.service.StelltAufrufKonte
xtBereitInterceptor">
    <property name="aufrufKontextVerwalter"
ref="aufrufKontextVerwalter"/>
    <property name="aufrufKontextFactory"
ref="aufrufKontextFactory"/>
  </bean>
  <aop:config>
    <aop:pointcut id="aufrufKontextPointcut"
expression="@annotation(de.bund.bva.pliscommon.aufrufkontext.service.
StelltAufrufKontextBereit) ||
@within(de.bund.bva.pliscommon.aufrufkontext.service.StelltAufrufKont
extBereit)"/>
    <aop:advisor pointcut-ref="aufrufKontextPointcut" advice-
ref="aufrufKontextInterceptor"/>
  </aop:config>
  <!--  @NutzerAuthentifizierung Annotation einschalten  -->
  <bean id="nutzerAuthentifizierungInterceptor"
class="de.bund.bva.pliscommon.sicherheit.annotation.NutzerAuthentifiz
ierungInterceptor">
    <property name="aufrufKontextVerwalter"
ref="aufrufKontextVerwalter"/>
    <property name="konfiguration" ref="konfiguration"/>
    <property name="sicherheit" ref="sicherheit"/>
  </bean>
  <aop:config>
    <aop:pointcut id="nutzerAuthentifizierungPointcut"
expression="@annotation(de.bund.bva.pliscommon.sicherheit.annotation.
NutzerAuthentifizierung) ||
@within(de.bund.bva.pliscommon.sicherheit.annotation.NutzerAuthentifi
zierung)"/>
    <aop:advisor pointcut-ref="nutzerAuthentifizierungPointcut"
advice-ref="nutzerAuthentifizierungInterceptor"/>
  </aop:config>
  <!--  @Gesichert Annotation einschalten  -->
```

```
  <bean id="gesichertInterceptor"
class="de.bund.bva.pliscommon.sicherheit.annotation.GesichertIntercep
tor">
    <property name="sicherheit" ref="sicherheit"/>
    <property name="sicherheitAttributeSource">
      <bean
class="de.bund.bva.pliscommon.sicherheit.annotation.AnnotationSicherh
eitAttributeSource"/>
    </property>
  </bean>
  <aop:config>
    <aop:pointcut id="gesichertPointcut"
expression="@annotation(de.bund.bva.pliscommon.sicherheit.annotation.
Gesichert) ||
@within(de.bund.bva.pliscommon.sicherheit.annotation.Gesichert)"/>
    <aop:advisor pointcut-ref="gesichertPointcut" advice-
ref="gesichertInterceptor"/>
  </aop:config>
  <!--  @StelltLoggingKontextBereit Annotation einschalten  -->
  <bean id="stelltLoggingKontextBereitInterceptor"
class="de.bund.bva.pliscommon.aufrufkontext.service.StelltLoggingKont
extBereitInterceptor"></bean>
  <aop:config>
    <aop:pointcut id="stelltLoggingKontextBereitPointcut"
expression="@annotation(de.bund.bva.pliscommon.aufrufkontext.service.
StelltLoggingKontextBereit) ||
@within(de.bund.bva.pliscommon.aufrufkontext.service.StelltLoggingKon
textBereit)"/>
    <aop:advisor pointcut-ref="stelltLoggingKontextBereitPointcut"
advice-ref="stelltLoggingKontextBereitInterceptor"/>
  </aop:config>
  <!--  Factory zum Erzeugen neuer Aufruf-Kontexte  -->
  <bean id="aufrufKontextFactory"
class="de.bund.bva.pliscommon.aufrufkontext.impl.AufrufKontextFactory
Impl">
    <property name="aufrufKontextKlasse"
value="de.bund.bva.pliscommon.aufrufkontext.impl.AufrufKontextImpl"/>
  </bean>
  <!--
  AufrufKontextVerwalter definieren (jeder Request hat einen eigenen
Kontext (-Verwalter))
  -->
  <bean id="aufrufKontextVerwalter" scope="thread"
class="de.bund.bva.pliscommon.aufrufkontext.impl.AufrufKontextVerwalt
erImpl">
    <aop:scoped-proxy/>
  </bean>
  <!--

========================================================================
Über diese Bean wird die Komponente
  Sicherheit Einsatzbereit gemacht
========================================================================
  -->
  <bean id="sicherheit"
class="de.bund.bva.pliscommon.sicherheit.impl.SicherheitImpl">
    <property name="rollenRechteDateiPfad"
value="/resources/sicherheit/rollenrechte.xml"/>
    <property name="aufrufKontextVerwalter"
ref="aufrufKontextVerwalter"/>
    <property name="accessManager" ref="camsAccessManager"/>
    <property name="konfiguration" ref="konfiguration"/>
    <property name="aufrufKontextFactory"
ref="aufrufKontextFactory"/>
  </bean>
  <!--  Zur Überwachung der Verfügbarkeit des Cams  -->
  <bean id="sicherheitAdmin"
class="de.bund.bva.pliscommon.sicherheit.impl.SicherheitAdminImpl">
    <property name="accessManager" ref="camsAccessManager"/>
  </bean>
  <!--

========================================================================
Konfiguration der CAMS Implementierung
  des AccessManagers
========================================================================
```

```
  -->
  <bean id="camsAccessManager"
class="de.bund.bva.pliscommon.sicherheit.cams.CamsAccessManagerImpl"
depends-on="konfiguration">
    <constructor-arg index="0">
      <value>classpath:/config/cams-webagent.conf</value>
    </constructor-arg>
    <constructor-arg index="1">
      <ref bean="konfiguration"/>
    </constructor-arg>
  </bean>
</beans>
```

Abbildung 8: Konfiguration für Spring

Die Property `cacheKonfiguration` der Bean `sicherheit` beinhaltet den Pfad zu einer Cache-Konfiguration. Dieser Parameter ist optional. Der Client besitzt schon eine Standard-Konfiguration für den Cache. Muss die Konfiguration der Authentifizierungs-Caches angepasst werden, so wird diese Cache-Konfiguration benötigt. Der Standard-Wert für die Lebensdauer von Cache-Elemente liegt bei 5 Minuten, die maximale Anzahl an Elementen in einem Schlüsselcache Cache liegt bei `1000`. Die Standard-Konfiguration befindet sich im Anhang. Die Werte für die Lebensdauer von Cache-Elementen und die Anzahl der Elemente kann über die betriebliche Konfiguration angepasst werden (siehe 3.6.5).

Die Spring-Konfiguration der Komponente Sicherheit wird in die allgemeine Spring-Konfiguration der Querschnittsfunktionalität der Anwendung in */resources/spring/querschnitt.xml* eingebunden.

```
  […]
      <import resource="querschnitt/sicherheit.xml" />
  […]
```

Abbildung 9: Konfiguration Sicherheit in querschnitt.xml

3.6.3 Scope des AufrufKontextVerwalters

Der AufrufKontextVerwalter ist in einem geeigneten Spring-Scope zu halten. Welcher Scope geeignet ist, ist abhängig von der Art der Verarbeitung:

- für Anfragen über die GUI oder über eine Service-Schnittstelle wird der Request-Scope (alternativ Thread-Scope) verwendet

```
<!–definiere AufrufKontextVerwalter; jeder Request hat einen eigenen -->
<bean id="aufrufKontextVerwalter" scope="request" class="..." >
```

- für Batches wird ein Singleton (alternativ Thread-Scope) verwendet

```
<!–definiere AufrufKontextVerwalter für Batches als Singleton -->
<bean id="aufrufKontextVerwalter" class="..." >
```

- für Workflow-Threads innerhalb einer Fachanwendung wird der Thread-Scope verwendet.

```
<!–definiere AufrufKontextVerwalter für interne Threads mit ThreadScope -->
<bean id="aufrufKontextVerwalter" scope="thread" class="..." >
```

Ab Spring 3 kann auch generell der Thread-Scope verwendet werden.

3.6.4 Konfiguration für den Access Manager

Die Komponente Sicherheit verwendet eine Client-Bibliothek des Access-Managers, um einen Benutzer gegen den Access-Manager-Service zu authentifizieren.

Wenn als Access-Manager CAMS verwendet wird, ist eine Konfiguration in der betrieblichen Konfigurationsdatei */config/cams-webagent.conf* erforderlich. Da diese Konfiguration in der Regel langjährig unverändert verwendet wird und lediglich die Aufrufadresse anzupassen ist, wird auf die Darstellung der Konfiguration hier verzichtet.

3.6.5 Konfiguration des Caches

Um den Access Manager zu entlasten, kann ein Cache eingeschaltet werden. Die Konfiguration des Caches erfolgt über die betriebliche Konfiguration. Hierzu sind folgende Parameter zu setzen:

```
# Zeit in Sekunden, die ein Eintrag im Cache verweilt, bis er als ungültig #
markiert wird.
# Time-To-Live 0 deaktiviert den Cache.
sic.caching.ttl=300

# Maximale Anzahl an Elementen, die im Cache vorgehalten werden. Die Elemente,
# die am
# längsten nicht verwendet wurden, werden aus dem Cache entfernt.
# Der Cache hat eine unbegrenzte Größe, wenn der Wert auf 0 gesetzt wird.
sic.caching.max.elements=10000
```

Wird der Parameter `sic.caching.ttl` auf 0 gesetzt oder nicht konfiguriert, ist der Cache deaktiviert. Wird der Parameter `sic.caching.max.elements` auf 0 gesetzt, werden beliebig viele Elemente im Cache vorgehalten.

4. Autorisierung innerhalb einer Anwendung

Vor Durchführung von Autorisierungsprüfungen bei der Bearbeitung einer Anfrage ist einmalig für die laufende Anfrage der AufrufKontextVerwalter mit dem AufrufKontext zu füllen, damit die Komponente Sicherheit während der Bearbeitung der Anfrage korrekt funktioniert. Sowohl für die Entgegennahme des AufrufKontextes, als auch für die eigentliche Autorisierung werden mit den Komponenten Sicherheit und Aufrufkontext einige Hilfsmittel angeboten.

Durch Verwendung dieser Hilfsmittel lässt sich die Autorisierung weitgehend deklarativ und transparent abwickeln. Autorisierung wird zum Querschnittsaspekt und weitgehend aus dem Programmcode eliminiert, und fachliche Schnittstellen werden von Parametern befreit.

Befüllen des AufrufKontextVerwalters

Der AufrufKontext ist aus der eingehenden Anfrage auszulesen (so früh wie möglich, bevorzugt noch vor dem Aufruf der Schnittstellenmethode) und im AufrufKontextVerwalter zu registrieren, damit nachfolgend die Komponente Sicherheit verwendet werden kann. Dies soll transparent für den Entwickler mit geringem Aufwand erfolgen.

Dazu stellt die Bibliothek isy-aufrufkontext Hilfsmittel (Annotationen, Filter) bereit, die diese Aufgabe transparent für den Entwickler durchführen:

- Für Aufrufe der GUI wird ein Processing-Filter aus Spring-Security verwendet (siehe 4.1).
- Für Aufrufe von Service-Schnittstellen wird eine Annotation an den Service-Methoden verwendet (siehe 4.2).
- Die Autorisierung eines Batch-Benutzers und die Befüllung des AufrufKontextVerwalters erfolgt über den Batchrahmen (siehe 4.3).

Durchführung der Autorisierungsprüfung

Nach Übernehmen des AufrufKontext in den AufrufKontextVerwalter erfolgt die Autorisierungsprüfung dann über die Komponente Sicherheit.

Für die Autorisierungsprüfung stellt die isy-sicherheit neben dem Berechtigungsmanager (erlaubt Autorisierung per API-Aufruf) zusätzliche Hilfsmittel (Annotationen, Tags zur Verwendung in Flow-Definitionen) bereit, mit denen die Autorisierungsprüfung deklarativ erfolgen kann:

- Für Aufrufe der GUI wird das *Secured*-Tag zur Autorisierung verwendet (siehe 4.1).
- Für Aufrufe der Service-Schnittstellen wird eine Annotation @Gesichert zur Autorisierung verwendet (siehe 4.2).
- Für Batchläufe ist im Batchrahmen bereit eine Autorisierung eines konfigurierten Benutzers enthalten (siehe 4.3).

- Für Autorisierungen im Anwendungskern kann die Annotation @Gesichert oder direkt der Berechtigungsmanager verwendet werden.

4.1. Autorisierung in der GUI

Hier wird erklärt, wie der Dialog einer Geschäftsanwendung mit Hilfe der Komponente Sicherheit abgesichert wird. Die Umsetzung der Autorisierungsprüfung in der GUI ist in [WebGUIDetailkonzept] beschrieben.

4.1.1 Implementierung im Webflow

Ein Webflow (oder auch ausgewählte Zustände und Transitionen) können durch das <secured/> Tag gesichert werden. Hier kann geprüft werden, ob ein Benutzer ein oder mehrere erforderliche Recht(e), aufgrund der ihm zugeordneten Rolle besitzt. Beispiel:

```
[…]
<flow>
    <secured attributes="lesen" />
[…]
```

Abbildung 10: Absichern eines Flow

Die Nutzung des <secured> Tags setzt voraus, dass vor Verarbeitung des Requests der Aufrufkontext im AufrufKontextVerwalter registriert wird.

Aufrufe der GUI erfolgen per http. Der Access-Manager stellt am Webserver vor Weiterleitung der Anfrage an eine Fachanwendung sicher, dass nur authentifizierte Anfragen an der GUI der Anwendung eingehen. Mit der Authentifizierung des Anwenders fügt der Access-Manager http-Header in die Anfrage ein. Diese Header enthalten neben Informationen der Identifikation des anfragenden Benutzers auch die Gesamtheit seiner Rollen. Die Informationen aus dem Http-Header werden in den AufrufKontext übertragen.

Die Bibliothek plis-sicherheit-cams enthält einen *CamsAuthentication-ProcessingFilter*, der den Aufrufkontext bei jeder Anfrage an die GUI aus dem http-Header ermittelt und damit den AufrufKontextVerwalter befüllt. Einmal konfiguriert, erfolgt die Erstellung des Aufrufkontexts transparent für sämtliche Aufrufe der GUI.

4.1.2 Konfiguration

Folgende Konfiguration in */resources/spring/querschnitt/sicherheit.xml* ist erforderlich:

```
    <sec:http entry-point-ref="camsPreAuthenticatedProcessingFilterEntryPoint"
              create-session="never" />

    <sec:authentication-manager alias="authenticationManager" />

    <bean id="camsPreAuthenticatedProcessingFilterEntryPoint" class="org.spring
framework.security.ui.preauth.PreAuthenticatedProcessingFilterEntryPoint" />
<!-- Für Tests ohne Cams das folgende Bean auskommentieren. -->
    <bean id="camsProcessingFilter" class="de.bund.bva.pliscommon.aufrufkontext
.http.CamsAuthenticationProcessingFilter">
        <sec:custom-filter position="PRE_AUTH_FILTER" />
        <property name="authenticationManager" ref="authenticationManager" />
        <property name="aufrufKontextVerwalter" ref="aufrufKontextVerwalter" />
```

```
        <property name="httpHeaderParser">
      <bean
class="de.bund.bva.pliscommon.sicherheit.cams.web.HttpHeaderParser"/>
        </property>
    </bean>

    <bean id="accessDecisionManager"
class="de.bund.bva.pliscommon.sicherheit.web.DelegatingAccessDecisionManager">
        <property name="sicherheit" ref="sicherheit" />
    </bean>
    <!-- ========================================================================
    custom authentication provider setzen um Sicherheits Komponente
    für Rollen Ermittlung zu verwenden
    ======================================================================== -->

    <bean id="webflowAuthenticationProvider" class="de.bund.bva.pliscommon.
aufrufkontext.http.WebFlowAuthenticationProvider">
        <sec:custom-authentication-provider />
    </bean>

    <!--========================================================================
    Dummy User Service
    ======================================================================== -->
    <sec:user-service>
        <sec:user authorities="ROLE" name="name" password="password"
disabled="true" />
    </sec:user-service>
```

Abbildung 11: Konfiguration für Autorisierung in der GUI

4.2. Autorisierung an einer Service-Schnittstelle

Hier wird erklärt, wie eine Service-Schnittstelle einer Anwendung mit Hilfe der Komponente Sicherheit abgesichert wird.

Die Umsetzung der Autorisierungsprüfung in angebotenen Schnittstellen ist in [Services] beschrieben.

4.2.1 Implementierung

Methoden der Service-Implementierung werden durch die Annotation @Gesichert gesichert. Die Annotation hat das Feld Recht, mit der ein oder mehrere geforderte Rechte konfiguriert werden können. Der Benutzer muss alle geforderten Rechte besitzen, damit ihm Zugang gewährt wird.

Um die Prüfung der Rechte vornehmen zu können, müssen zuvor Informationen über den aufrufenden Benutzer im AufrufKontext hinterlegt werden. Im Gegensatz zu Aufrufen der GUI, werden bei Aufrufen von Serviceschnittstellen (über http-Invoker) keine http-Header zum Transport des Aufrufkontextes verwendet. Stattdessen wird entsprechend der Vorgaben der IsyFact an Service-Schnittstellen der Aufrufkontext als Schnittstellenparameter AufrufKontextTo mit jeder Serviceanfrage übergeben. Im Regelfall enthält dieses AufrufKontextTo bereits die Informationen zum anfragenden Benutzer und dessen Rollen.

Ist dies der Fall kann die Annotation @StelltAufrufKontextBereit verwendet werden, um die bereitgestellten Informationen in den AufrufKontextVerwalter zu übertragen.

Nachfolgend ist es möglich mittels der Annotation @Gesichert die Berechtigungsprüfung durchzuführen.

Die Annotationen nutzen Spring-AOP. Daher sind die Annotationen nur an public-Methoden von Spring-Beans funktionsfähig. Für Http-Invoker-

Schnittstellen sind hier die ServiceMethoden der <Service>Impl (unterhalb der ExceptionFassade mit den Annotationen auszustatten.

Folgendes Beispiel zeigt die Implementierung einer Service-Methode, für die der Aufrufkontext automatisch in den AufrufKontextVerwalter übernommen wird und anschließend die Autorisierung gegen die Komponente Sicherheit durchgeführt wird.

```
@StelltAufrufKontextBereit
@Gesichert("Recht_A", "Recht_B")
public void gesichertDurch_RechtAundB(AufrufKontextTo kontext, ...){
   //…
}
```

Abbildung 12: Absichern einer Servicemethode

Erfüllt ein Aufrufer nicht die Forderungen der Annotation @Gesichert, so wird die AutorisierungFehlgeschlagenException geworfen. Da eine fehlende Berechtigung nicht behandelt werden kann und dies letztlich auf einen Konfigurationsfehler in einer anderen Anwendung zurückzuführen ist (oder einen Angriff), wird nicht empfohlen, diese Ausnahme spezifisch zu behandeln. Das normale Fehlerhandling greift.

4.3. Autorisierung eines Batches

Hier wird erklärt, wie ein Batchlauf einer Anwendung mit Hilfe der Komponente Sicherheit abgesichert wird. Die Integration der T-Komponente Sicherheit ist im Batchrahmen ab Version 1.1 enthalten.

Ein Batch wird innerhalb der Anwendungslandschaft gestartet. Dazu verwendet der Batch die Benutzerkennung eines Systems (Benutzer), die in seiner Startkonfiguration (als Konfigurationsdatei) hinterlegt und im Benutzerverzeichnis vorhanden ist.

Damit der Benutzer bei der Ausführung des Batches authentifiziert wird, muss die Methode *initialisieren*, die durch das BatchAusfuehrungsBean vorgegeben wird, mittels der Annotation @Gesichert geschützt werden. Der Access-Manager authentifiziert den Benutzer anhand der angegebenen Benutzerkennung und des Passwortes, registriert alle Informationen zum Benutzer im AufrufKontextVerwalter und schließt die Session des Benutzers im Access-Manager.

Da alle Informationen zum Benutzer als Aufrufkontext im AufrufKontextVerwalter hinterlegt wurden, kann die Anwendung jederzeit auf diese Kontextinformationen zugreifen, also im Zuge des Batchlaufes Berechtigungsprüfungen vornehmen oder auch Services von Nachbarsystemen (vergleiche Kapitel 5.1) unter Bereitstellung des Aufrufkontextes aufrufen.

In Ausnahmefällen ist es auch möglich, einen Batch zu implementieren, der ohne Benutzer laufen soll. Dies ist nur möglich, wenn bei Aufrufen des Anwendungskerns keine Autorisierungsprüfungen stattfinden und auch keine Nachbarsystemschnittstellen aufgerufen werden.

4.3.1 Implementierung

Der Batchrahmen fordert vom BatchAusführungsBean die Implementierung der Methode: getAuthenticationCredentials(), in der dem Batchrahmen bekanntgegeben wird mit welchem Benutzer der Batch laufen soll.

In der Regel wird der Benutzer aus der Konfiguration gelesen, wie im folgenden Beispiel veranschaulicht wird:

```
@Override
public BatchAuthenticationCredentials getAuthenticationCredentials(
                    BatchKonfiguration konfiguration) {

    BatchAuthenticationCredentials auth = new BatchAuthentifizierung();

    //Wenn der Benutzer nicht auf der Kommandozeile angegeben wird, lade
      die Daten aus der betrieblichen Konfigurationsdatei
    String bhknz = this.behoerdenVerzeichnisKonfiguration.
     getAsString(KonfigurationSchluessel.CONF_BATCH_BEHOERDE, null);
    String benutzer = this.behoerdenVerzeichnisKonfiguration.
     getAsString(KonfigurationSchluessel.CONF_BATCH_BENUTZER, null);
    String passwort = this.behoerdenVerzeichnisKonfiguration.
     getAsString(KonfigurationSchluessel.CONF_BATCH_PASSWORT, null);

    //Lade Nutzer aus Kommandozeile / Batchkonfiguration
    auth.setBehoerdenkennzeichen(konfiguration.getAsString(
     KonfigurationSchluessel.CONF_BATCH_BEHOERDE, bhknz));
    auth.setBenutzerkennung(konfiguration.getAsString(
     KonfigurationSchluessel.CONF_BATCH_BENUTZER, benutzer));
    auth.setPasswort(konfiguration.getAsString(
     KonfigurationSchluessel.CONF_BATCH_PASSWORT, passwort));

    return auth;
}
```

Abbildung 13: Implementierungsbeispiel für die Versorgung des Batchrahmens mit Benutzerdaten

Soll der Batch ohne Benutzer laufen, so ist null zurückzugeben.

4.3.2 Konfiguration

Für eine korrekte Funktion wird in der Anwendung die Komponente Sicherheit konfiguriert (vergleiche 3.6).

Zusätzlich wird der Benutzer, über den der Batch laufen soll, wie folgt konfiguriert:

```
# Benutzerkennung für den Batch Anschriftenverzeichniserzeugung
batch.anschriftenverzeichnis.benutzerkennung = BHVZ_BAT_Anschriftenverzeichnis
batch.anschriftenverzeichnis.benutzerpasswort = changePassword
batch.anschriftenverzeichnis.benutzerbehoerde = 42
```

Abbildung 14: Konfiguration des Batchbenutzers

Damit eine Authentifizierung über den Access-Manager möglich ist, sind einige zusätzliche Konfigurationen für die Komponente Sicherheit erforderlich:

```
# Konfiguration fuer den CAMS-Zugriff zur Autorisierung von Benutzern
sic.camsagent.login.config.entry=applicationplatform
sic.camsagent.login.medium=applicationplatform
sic.camsagent.security.domain=applicationplatform
sic.camsagent.nutzertyp=unbekannt
```

Abbildung 15: Konfiguration der plis-sicherheit-cams für Batches

Hier wird festgelegt, dass ein Aufrufer gegen die im Access-Manager konfigurierte Security-Domain applicationplatform authentifiziert wird.

4.4. Autorisierung im Service-Gateway

Der Serviceanbieter eines Service-Gateways [ServiceGatewaySystementwurf] verwendet die T-Komponente Sicherheit, um den Benutzer einer eingehenden Anfrage zu authentifizieren und zu autorisieren. Sie verwendet dazu die Bibliothek plis-sgw-util, in der isy-sicherheit eingebunden ist.

4.4.1 Autorisierung einer Anfrage an ein Service-Gateway

Ein Service eines Service-Gateways wird aufgerufen. Mit der Anfrage wird die Benutzerkennung des anfragenden Benutzers übermittelt.

Die im Service-Gateway verwendete Komponente Serviceanbieter nutzt die Komponente Sicherheit, um den anfragenden Benutzer bereits in der Informations- und Dienstezone zu authentifizieren. Der Access-Manager authentifiziert den Benutzer anhand des Zertifikats, der angegebenen Benutzerkennung und des Passwortes, registriert alle Informationen zum Benutzer im AufrufKontextVerwalter und schließt die Session des Benutzers im Access-Manager wieder.

Beim Weiterleiten der Anfrage an den Service der Fachanwendung wird der AufrufKontext an die aufzurufende Schnittstelle der Anwendung übergeben (vergleiche Kapitel 5.1).

4.4.2 Konfiguration

Für eine korrekte Funktion wird im Service-Gateway die Komponente Sicherheit konfiguriert (vergleiche 3.6).

Damit eine Authentifizierung über den Access-Manager möglich ist, sind einige zusätzliche Konfigurationen für die Komponente Sicherheit erforderlich:

```
# Konfiguration fuer den CAMS-Zugriff zur Autorisierung von Benutzern
sic.camsagent.login.config.entry=servicegateway
sic.camsagent.login.medium=servicegateway
sic.camsagent.security.domain=servicegateway
sic.camsagent.nutzertyp=unbekannt
```

Abbildung 16: Konfiguration der plis-sicherheit-cams für Service-Gateways

Hier wird festgelegt, dass ein Aufrufer gegen die Security-Domain des Access-Managers servicegateway authentifiziert wird.

4.5. Autorisierung im Anwendungskern

Im Regelfall wird die Autorisierung einer Anfrage an den Schnittstellen der Anwendung durchgeführt. Es ist jedoch bei Bedarf auch möglich, Prüfungen innerhalb des Anwendungskerns durchzuführen.

4.5.1 Autorisierung an Methoden des Anwendungskerns

Im Anwendungskern kann durch Verwendung der Annotation @Gesichert an Methoden des Anwendungskerns deklarativ festgelegt werden, wo eine Berechtigungsprüfung erfolgen soll. Hier wird ausgewertet, welche Rollen und/oder Rechte der Benutzer haben muss, damit der Zugriff gewährt wird. Die Anwendung fragt bei der Komponente Sicherheit die Rollen und Rechte des Benutzers ab und autorisiert die Anfrage.

Voraussetzung für das Funktionieren der Annotation @Gesichert ist, dass im Rahmen des Aufrufs der Anwendung der AufrufKontextVerwalter bereits gefüllt wurde.

4.5.2 Autorisierung in Methoden des Anwendungskerns

Prüfungen können auch ausprogrammiert werden. Dabei wird die Komponente Sicherheit verwendet. Hierzu wird das Bean Sicherheit in die Komponente des Anwendungskerns injiziert und der Berechtigungsmanager über die Methode getBerechtigungsManager() beschafft. Der Berechtigungsmanager stellt die Methoden getRechte(), getRecht(), hatRecht() und pruefeRecht() bereit, mit denen die gewünschte Prüfung vorgenommen werden kann.

4.5.3 Autorisierung innerhalb des Regelwerk

Werden Regelwerke verwendet, so sind hier häufig Prüfungen zur Sichtbarkeit, Melde- und Auskunftsrechten, sowie von Primärdaten abhängigen Rechten umzusetzen. Hierbei handelt es sich meist nicht um eine Autorisierung gegen Rollen, sondern um datenbezogene Prüfungen (z.B. Prüfung der Behördengruppe des Benutzers). Für diese Fälle gibt es keine Vorgaben, wie das zu erfolgen hat. Sie können zum Beispiel innerhalb des Regelwerks als Regeln hinterlegt werden.

Soll innerhalb des Regelwerks auf Rollen geprüft werden, so ist eine individuelle Lösung unter Nutzung der Komponente Sicherheit möglich.

4.5.4 Asynchrone Prozesse

Einige Anwendungen (z.B. Nachrichtenempfang an einem Email-Service-Gateway) verwenden asynchrone Prozesse, bzw. starten Prozesse ereignis- oder zeitgesteuert. Diesen Prozessen geht im Moment der Bearbeitung keine Benutzeranfrage voraus. Daher kann die Bearbeitung im Regelfall nicht mit dem Aufrufkontext eines anfragenden Benutzers durchgeführt werden. Stattdessen wird zum Start des Prozesses ein hinreichend berechtigter Benutzer (System) verwendet. Es ist ein analoger Ablauf wie beim Start von Batches (siehe 4.3) umzusetzen. Es ist eine analoge Konfiguration wie bei Batches (siehe 4.3.2) vorzunehmen.

5. Weitere Konzepte

5.1. Aufrufen von Nachbarsystemen

So wie eine Anwendung bei einem Aufruf erwartet, einen gültigen, vollständigen Aufrufkontext vorzufinden, erwartet dies auch ein Nachbarsystem, welches von der eigenen Anwendung aufgerufen wird. Das aufrufende System muss daher einen Aufrufkontext mitliefern. Im Regelfall soll dabei der Aufrufkontext der originären Anfrage verwendet und unverändert weitergeleitet werden.

Zum Aufruf des Nachbarsystems werden entweder die mit dem Nachbarsystem bereit gestellten Service-Client-Bibliotheken oder direkt die RemoteBean der aufzurufenden Schnittstelle verwendet.

Wenn ein Service über eine spezifische Client-Implementierung (z.B. den Schlüsselverzeichnis-Client) verwendet wird, so enthält diese (konform zum Berechtigungskonzept) bereits die Logik zur Weiterleitung des Aufrufkontextes.

Wenn der Service direkt über Spring HttpInvoker und das zugehörige RemoteBean-Interface aufgerufen wird, so ist die Weiterleitung des AufrufKontextes unter Nutzung des AufrufKontextVerwalters (per Spring injizieren) manuell zu programmieren. Hierbei ist es wichtig, dass immer ein AufrufKontextTo-Objekt der Service-API verwendet wird, da nur dieses bei einem HttpInvoker-Aufruf im Server korrekt deserialisiert werden kann.

```
AufrufKontext aufrufKontext =
aufrufKontextVerwalter.getAufrufKontext();
if (aufrufKontext == null) {
   throw new ...Exception(...);
}
remoteBean.methodeX((AufrufKontextTo) dozer.map(aufrufKontext,
                      AufrufKontextTo.class, weitere, parameter);
```

Abbildung 17: Weiterleitung des Aufrufkontextes beim Serviceaufruf

Das Beispiel enthält folgende Schritte:

- Der AufrufKontext der Anwendung (dies ist meist eine anwendungsspezifische Implementierung des AufrufKontextes) wird vom AufrufKontextVerwalter abgerufen.
- Es wird ein Dozer-Default-Mapper verwendet, um das für Serviceaufrufe zu verwendende AufrufKontextTo der Service-API zu erstellen.
- Das erzeugte AufrufKontextTo der Service-API wird an die Schnittstelle des Service-Client übergeben.

5.2. Verwendung der Korrelations-ID

Der AufrufKontext enthält auch die Korrelations-ID, die während der Bearbeitung einer Anfrage in der Anwendungslandschaft alle zusammengehörenden Log-Ausgaben einer Anfrage durch eine gemeinsame Id kennzeichnet – auch wenn die Anfrage mehrere Teilsysteme der

Anwendungslandschaft verwendet. Dies erfordert eine Entgegennahme, Verwendung und Weiterleitung einer Korrelations-ID über alle Service-Aufrufe hinweg. Details zum Aufbau der Korrelations-ID sind in [NutzungsvorgabenLogging] beschrieben.

Eine Korrelations-ID wird entweder bereits im eingehenden Aufruf mitgeliefert, oder muss neu erzeugt werden.

5.2.1 Entgegennahme der Korrelations-ID in der GUI

Die Neuerzeugung und Registrierung der Korrelations-ID im MDC erfolgt automatisch innerhalb des HttpHeaderParsers der Bibliothek plis-sicherheit-cams.

5.2.2 Entgegennahme der Korrelations-ID an einer Service-SST

Die Entgegennahme (ggf. Neuerzeugung) und Registrierung der Korrelations-ID im MDC erfolgt an der Exception-Fassade der Http-Invoker-Schnittstelle über die Annotation @StelltLoggingKontextBereit aus der Bibliothek plis-serviceapi-core.

5.2.3 Ablage und Weiterleitung der Korrelations-ID

Die Korrelations-ID wird für das Logging im MDC abgelegt. Gleichzeitig wird diese jedoch auch im vom AufrufKontextVerwalter verwalteten AufrufKontext hinterlegt. Ist dies der Fall, so wird die Korrelations-ID beim Aufruf des RemoteBeans des Nachbarsystems als Bestandteil des AufrufKontextes automatisch weitergeleitet.

5.3. Verwenden anwendungsspezifischer Aufrufkontexte

Gelegentlich möchte eine Anwendung zusätzliche Informationen im AufrufKontextVerwalter ablegen. Hierzu lässt sich eine anwendungsspezifische Spezialisierung des *de.bund.bva.pliscommon.kontext.AufrufKontextImpl* verwenden, der den AufrufKontext um zusätzliche Attribute erweitert.

```
    <!-- Erstellt den anwendungsspezifischen AufrufKontext -->
    <bean id="aufrufKontextFactory" class="de.bund.bva.pliscommon.
aufrufkontext.impl. AufrufKontextFactoryImpl">
        <property name="aufrufKontextKlasse" value="de.bund.bva.pliscommon.
aufrufkontext.impl.<Anwendungsspezifisch>AufrufKontextImpl" />
    </bean>
```

Abbildung 18: Verwendung eines anwendungsspezifischen AufrufKontext

Die zusätzlichen Attribute können bereits beim automatischen Befüllen des AufrufKontextVerwalters gesetzt werden. So wäre es beispielsweise möglich, bei der Annahme eines Requests den Anfragezeitpunkt im Aufrufkontext festzuhalten. Dazu kann eine anwendungsspezifische AufrufKontextFactory verwendet werden.

5.4. Entwickeln und Testen ohne Access-Manager-Service

Hier wird erläutert, wie ein konfigurierbarer Stub des Aufruf-KontextVerwalters verwendet werden kann, um für eine in Entwicklung oder Test befindliche Anwendung das Vorliegen eines Aufrufkontextes (mit

Informationen zu einem Benutzer und dessen Rollen) zu simulieren, obwohl bei einer Anfrage gar kein Aufrufkontext übergeben wurde.

Mit Hilfe dieses Stubs kann auf die Verwendung eines Access Managers in der Software-Entwicklungsumgebung verzichtet werden. Der Stub gibt bei jeder Anfrage die statisch konfigurierten Benutzerdaten zurück. Die Konfiguration erfolgt in */resources/spring/querschnitt/sicherheit.xml*.

```
<!-- Verwende Stub Implementierung für die Entwicklung -->
<bean id="aufrufKontextVerwalter" class="de.bund.bva.pliscommon.
aufrufkontext.stub.AufrufKontextVerwalterStub" scope="request">
  <aop:scoped-proxy/>
  <property name="rollen">
    <list>
       <value><Rolle1, die der Benutzer im Test haben soll></value>
       <value><Rolle2, die der Benutzer im Test haben soll></value>
    </list>
  </property>
  <property name="durchfuehrendeBehoerde" value="42" />
  <property name="aufrufKontextFactory" ref="aufrufKontextFactory" />
</bean>
```

Abbildung 19: Konfiguration Aufrufkontext-Stub

Neben der gezeigten Eigenschaft „Rollen“ und durchführende Behörde lassen sich auch alle weiteren Eigenschaften des Benutzers konfigurieren.

6. Quellenverzeichnis

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

[Berechtigungskonzept]
Berechtigungskonzept
Muss projektspezifisch erstellt werden.

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

[NutzungsvorgabenLogging]
Nutzungsvorgaben Logging
20_Bausteine\Logging\Nutzungsvorgaben_Logging.pdf.

[ServiceGatewaySystementwurf]
Systemdokumentation Service-Gateway
20_Bausteine/Service-Gateway/Systemdokumentation_Service-Gateway.pdf .

[ServicekommunikationKonzept]
Grundlagen der Servicekommunikation innerhalb der Plattform
10_Blaupausen\Integrationsplattform\Grundlagen_interne_Servicekommunikation.pdf .

[Services]
Detailkonzept Komponente Service
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Service.pdf .

[VorlageAnwendung]
Beispielimplementierung „Vorlage-Anwendung“
Wird auf Anfrage bereitgestellt.

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

7. Abbildungsverzeichnis

Anhang

7.1. Konfigurationsdateien

7.1.1 Standard-Cache-Konfiguration

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xsi:noNamespaceSchemaLocation="ehcache.xsd"
    updateCheck="false" monitoring="autodetect" dynamicConfig="true">

    <!--
        Pflicht: Standard-Einstellungen fuer Caches.
        Diese Einstellungen werden verwendet,
        sofern Caches programmatisch erzeugt werden mit Hilfe von
        CacheManager.add(String cacheName).

        Der defaultCache hat den impliziten Namen "default",
        welcher einer reservierter
        cache-Name ist.
    -->

    <defaultCache
        maxEntriesLocalHeap="1000"
        eternal="false"
        timeToLiveSeconds="300"
        timeToIdleSeconds="0"
        overflowToDisk="false"
        memoryStoreEvictionPolicy="LRU" />

    <cache
        name="authentifizierungen"
        maxEntriesLocalHeap="1000"
        eternal="false"
        timeToLiveSeconds="300"
        timeToIdleSeconds="0"
        overflowToDisk="false"
        memoryStoreEvictionPolicy="LRU" />
</ehcache>
```

7.2. Rollen-Rechte-Schema

Im Folgenden ist der Inhalt der Datei `RollenRechteSchema.xsd` wiedergegeben, die das Format der Konfigurationsdatei für Rollen und Rechte (`rollenrechte.xml`) festlegt.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/RollenRechteSchema"
  xmlns:tns="http://www.example.org/RollenRechteSchema"
  elementFormDefault="qualified">

  <include schemaLocation=""></include>

  <complexType name="Rolle">
    <sequence>
      <element name="rechtId" type="tns:RechtId"
               minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="RolleId" type="string"
                 use="required"></attribute>
  </complexType>

  <complexType name="Recht">
    <sequence>
      <element name="rechtId" type="tns:RechtId"
               minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

  <element name="Anwendung" type="tns:Anwendung"></element>

  <complexType name="Anwendung">
    <sequence>
      <element name="rollen" type="tns:Rolle" minOccurs="1"
               maxOccurs="unbounded">
      </element>
      <element name="rechte" type="tns:Recht" minOccurs="0"
               maxOccurs="unbounded"></element>
    </sequence>
    <attribute name="AnwendungsId" type="string"
               use="required"></attribute>
  </complexType>

  <complexType name="RechtId">
    <attribute name="Id" type="string" use="required"/>
  </complexType>
</schema>
```