Bundesverwaltungsamt

IsyFact-Standard

Detailkonzept Komponente GUI
Nutzungsvorgaben Spring Web Flow / JSF / Facelets

Version 2.26
20.01.2017

Das Detailkonzept zur Komponente Web GUI und die Bibliothek plis-web werden zurzeit grundlegend überarbeitet.
Der vorliegende Stand dieses Dokuments beschreibt noch nicht alle neuen Funktionen von plis-web 4.x.

IsyFact Standards

Ansprechpartner:

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

Dokumentinformationen

Dokumenten-ID:	Detailkonzept_Komponente_Web_GUI.docx

Java Bibliothek / IT-System

Name	Art	Version
isy-web	Bibliothek	siehe isyfact-bom v1.3.6
isy-style	Bibliothek	2.2.x

Inhaltsverzeichnis

1. Management Summary

In diesem Dokument werden Nutzungsvorgaben für die Erstellung einer grafischen Oberfläche in der Portalanwendung der Zielplattform aufgestellt. Sie betreffen die Erstellung der Oberfläche unter Verwendung von Spring Web Flow und JSF, sowie deren initialen Setup. Es wird die grobe Struktur der Oberfläche - beschrieben.

Betroffene Themenbereiche dieser Nutzungsvorgabe sind:

- Definition von Dialogabläufen mit Spring Web Flow
- Einsatz von JSF und Facelets
- Anbindung von Backend Services

2. Einleitung

Im Dokument zur Zielarchitektur ist der Aufbau von Fachanwendungen in fünf Komponenten beschrieben [IsyFactReferenzarchitekturITSystem]. Eine ist die Komponente GUI. Die T-Architektur und die Komponente sind in Abbildung 1 dargestellt.

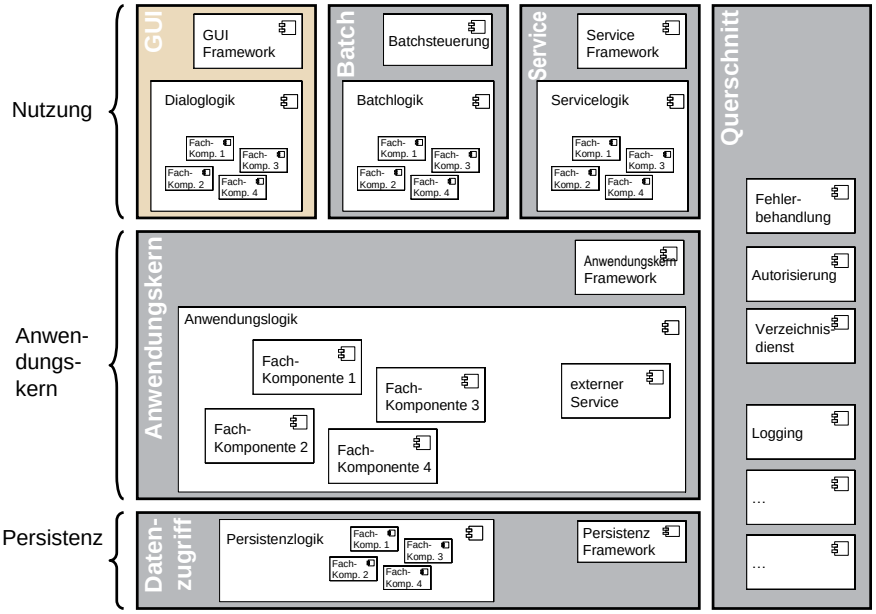

Abbildung 1: T-Architektur von Fachanwendungen

Das vorliegende Dokument enthält die Detailkonzeption der Komponente GUI, die entsprechend [IsyFactReferenzarchitektur] als Web-GUI umgesetzt wird.

Beschrieben werden allgemeine Vorgaben zur Erstellung der GUI einer Anwendung nach IsyFact-Architektur. Das Dokument gibt zusätzlich einen Überblick über die benötigten Basistechnologien Spring Web Flow, JSF und Facelets. Es werden die Randbedingungen und Besonderheiten für deren Einsatz innerhalb einer Anwendungslandschaft, die nach dem IsyFact-Standard gebaut wird, erläutert. Darüber hinaus stellt es die wesentlichen Schritte für den effektiven Aufbau einer geeigneten Projektumgebung dar.

Zielgruppe dieses Dokuments sind Entwickler und Architekten, die die IsyFact-konforme Umsetzung einer Web-GUI angehen. Vorkenntnisse mit den Technologien Spring Web Flow und JSF erleichtern den Einstieg und das Verständnis der in diesem Konzept enthaltenen Vorgaben.

Das Dokument ist in fünf Teile gegliedert:

- Darstellung der Anforderungen (Kapitel 3)
- Überblick über die verwendeten Basistechnologien (Kapitel 4).
- Architektur der Web-GUI und deren Einbettung in die IsyFact-Referenzarchitektur (Kapitel 5).

- Anleitung zur Umsetzung einer Web-GUI (Kapitel 6).
- Erläuterung der wichtigsten Konfigurationseinstellungen (Kapitel 6.4).
- Eine Beschreibung des Spring Web Flow Mechanismus zum Session-Handling (Kapitel 7.8).
- Eine Checkliste für die Qualitätssicherung in der Umsetzung der oben beschriebenen Tätigkeiten (Kapitel 9).

Das vorliegende Dokument bezieht sich auf einen BVA-internen Styleguide, der nicht bereitgestellt wird. Diese Abhängigkeit wird in einer der nachfolgenden Versionen des Dokuments aufgelöst. Die dargestellten Konventionen können jedoch als Vorlage zur Erstellung eines eigenen Styleguides verwendet werden.

Das in diesem Dokument vorgestellte Detailkonzept stellt die allgemeinen Architekturmuster einer Webanwendung im Rahmen der IsyFact dar. Spezifische Angaben zur Gestaltung und zur Umsetzung von Oberflächen (z.B. Nutzung von Widgets) finden sich im Styleguide [Styleguide].

3. Anforderungen

3.1. Allgemeine Anforderungen

Die in diesem Dokument aufgestellten Vorgaben setzen folgende Anforderungen um:

- **Einfachheit der Verwendung von JSF**: Hier werden die Konfiguration und der Einsatz der verwendeten Technologien festgelegt. Dazu zählen:
  - **Einsatz von Facelets**
  - **Einsatz von Tag Libraries**
  - **Fehlerbehandlung**
  - **Anbindung von Hilfesystemen**
- **Einfachheit des Einsatzes von Spring Web Flow:** Die Definition der Dialogschritte und die Abhängigkeiten zu den zugehörigen Daten für die Darstellung wird über Spring Web Flow festgelegt. Hierzu werden folgende Themenbereiche genauer definiert:
  - **Konfiguration der Dialog Abläufe (Flow)**
  - **Anbindung der Backend Services (Spring Beans)**
- **Einfachheit des Session Management:** Hier wird definiert, wie die Behandlung von Session Informationen erfolgen soll und welche wiederverwendbaren Services zur Verfügung gestellt werden.

3.2. Sicherheitsanforderungen

Web-Anwendungen sind besonderen Gefährdungen ausgesetzt. Folgende Anforderungen müssen bei der Entwicklung von Web-Anwendungen berücksichtigt werden:

- Entwickler müssen sich mit den TOP10 Risiken für Web-Anwendungen gemäß OWASP vertraut machen (siehe [OWASP10])
- Vertrauliche Informationen dürfen nicht als GET-Parameter übermittelt werden. Dies verhindert, dass solche Informationen ungewollt in Log-Dateien, Caches usw. gespeichert werden.

4. Verwendete Basistechnologien

Die Übersicht der Basistechnologien soll dem Leser einen einfacheren Einstieg in die angewendeten Frameworks und Technologien ermöglichen, zusätzlich findet sich noch ein Verweis auf die konkret eingesetzten Implementierungen. Hierbei werden auch Architekturprinzipien angesprochen, welche in den Technologien Verwendung finden.

Für detaillierte Informationen über die verwendeten Technologien sei auf entsprechende Literatur verwiesen. [WikiJSF] [SWF] [Spring]

4.1. JSF / Facelets

Nachfolgend soll kurz eine Erläuterung der Begrifflichkeiten vorgenommen werden:

- Java Server Faces (JSF) ist ein Framework für die Entwicklung von Webanwendungen. Es implementiert wie auch weitere Java-Frameworks die MVC-Architektur. Der Schwerpunkt von JSF ist die Bereitstellung grafischer Komponenten wie z.B. Tabellen, Formulare, Kalender, Menüs, Editoren, usw. für die Entwicklung von Webanwendungen. Zentrales Konzept hinter JSF ist die Bereitstellung eines Komponentenbaumes. Aus diesem Baum wird zum Schluss der Bearbeitung die HTML Seite generiert.
  Für JSF kommt die SUN Referenzimplementierung zum Einsatz und wird im Bereich der Komponenten um die Apache Tomahawk Bibliothek erweitert. [SUNRI] [Tomahawk]

- Facelets sind eine alternative View Technologie für JSF, hiermit sind kleine, wieder verwendbare GUI-Komponenten möglich, die durch ein entsprechendes XML-Tag in eine Seite inkludiert werden. Im Gegensatz zu Taglibs erfolgt ein Include der Komponente und kein Aufruf von Java-Code für die Generierung von gerendertem GUI-Code. Hierbei wird der Templating Mechanismus der Facelets verwendet. Facelets bieten vielfältige Möglichkeiten Vorlagenfragmente zu einer Gesamtseite zusammenzusetzen - zum Beispiel um auf jeder Seite eine einheitliche Kopfzeile zu realisieren.

  Weiter können Facelets mit herkömmlichen HTML-Editoren bearbeitet werden und sind somit einfacher zu verstehen und zu editieren. Die Ablage erfolgt hierbei dann in XHTML-Dokumenten. Als Implementierung kommt die SUN Bibliothek für Facelets [Facelets] zum Einsatz.

- Model-View-Controller (MVC, „Modell / Präsentation / Steuerung“) bezeichnet ein Architekturmuster zur Strukturierung von Software-Entwicklung in die drei Einheiten *Datenmodell* (engl. *Model*), *Präsentation* (engl. *View*) und *Programmsteuerung* (engl. *Controller*). Ziel des Musters ist es, einen flexiblen Programmentwurf zu machen, der u. A. eine spätere Änderung oder Erweiterung

erleichtert und eine Wiederverwendung der einzelnen Komponenten ermöglicht. Dieses Muster ist in Abbildung 2 dargestellt.

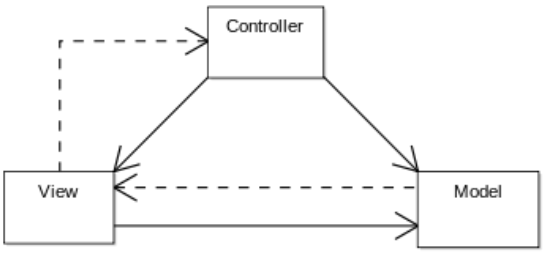

Abbildung 2: MVC Pattern

4.1.1 Bearbeitungsmodell einer JSF Anfrage

Die Spezifikation der Java Server Faces definiert einen sogenannten Lebenszyklus [WikiJSF], den eine JSF-Anwendung mit jedem Aufruf erneut durchläuft. Dieser Lebenszyklus ist in sechs *Phasen* (englisch *Phases*) aufgeteilt.

1. *Restore View* („Sicht wiederherstellen“) wählt anhand der eingehenden Anforderung eine Sicht (*View*) aus und baut den dazu passenden Komponentenbaum bei Bedarf auf.

2. *Apply Request Values* („Anforderungsparameter anwenden“) extrahiert Parameter aus der Anforderung (üblicherweise ein HTTP-Post-Request) und weist sie den passenden JSF-Komponenten zu, beispielsweise Eingabefeldern.

3. *Process Validations* („Validierung ausführen“) überprüft die Gültigkeit der zuvor ermittelten Eingaben. Dazu werden eigene Validator-Objekte verwendet, die den Komponenten in der View-Definition zugewiesen wurden.

4. *Update Model Values* („Modell aktualisieren“) weist den Modellobjekten die zuvor ermittelten Werte zu.

5. *Invoke Application* („Anwendung aufrufen“) ruft durch die Anwendung definierte Methoden auf, beispielsweise wenn ein Button betätigt wurde.

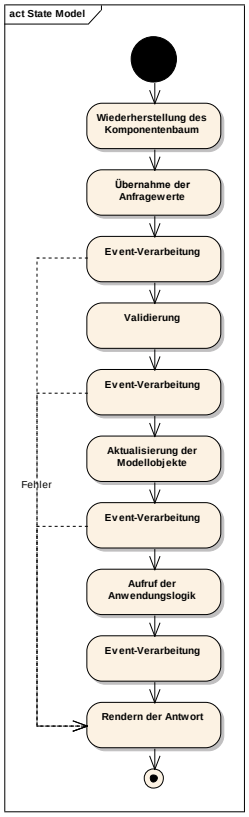

Abbildung 3: Phasenmodell JSF

6. *Render Response* („Antwort wiedergeben“) erzeugt schließlich die Antwort auf die ursprüngliche Anfrage, beispielsweise eine HTML-Seite. Hierzu werden sogenannte *Renderer* aufgerufen, die den View-Komponenten zugeordnet sind.

Treten Fehler auf, oder soll als Antwort beispielsweise eine HTML-Seite aufgerufen werden, die keine JSF-Komponenten enthält, so können einzelne Phasen übersprungen werden.

4.1.2 Datenmodell

Die Daten für die Visualisierung in JSF werden in Model Beans gehalten. Hierfür wird nicht auf den durch JSF zur Verfügung gestellten Mechanismus der Managed Beans zurückgegriffen. Das Datenmodell wird über Spring Web Flow direkt aus Model Beans verfügbar gemacht. Damit ist die Verwaltung des Models unter Kontrolle des Dialogflusses welcher über Spring Web Flow gesteuert wird. Die einem Flow zugeordneten Model Beans werden durch den Flow instanziiert und unterliegen somit dem Flow-Lebenszyklus.

4.1.3 Facelets

Mit den Facelets werden das visuelle Layout und die Controls für die Ansicht im Browser definiert. Durch die Nähe zu HTML sind schnell die notwendigen Ansichten designed und können getestet werden. Das Mapping der Controls auf die Daten geschieht über die in JSF verwendete Expression Language (EL). Mit der EL werden direkt die Attribute des zugehörigen Beans genutzt.

Ein weiterer Vorteil von Facelets ist die Verwendung von Templates. Durch diese ist es möglich, bereits einen zum Styleguide [Styleguide] konformen Rahmen zur Verfügung zu stellen, in welchen die Applikation lediglich durch definierte Einfügungen ihre Inhalte einbetten.

4.1.4 Taglibs

Durch den Einsatz von Facelets ist die direkte Einbettung von Tag Libraries nicht möglich. Vielmehr müssen diese noch separat mit einer Deskription versehen werden, aus welcher die einzelnen Tags ersichtlich sind und ihr Mapping auf die zugehörigen Klassen definiert ist.

Für den Einsatz der myFaces Tomahawk Library wird eine entsprechende Konfiguration zur Verfügung gestellt. Der Einsatz dieser Library unterliegt für den Einsatz in der IsyFact der allgemeinen Einschränkung bei der Verwendung von JavaScript. Siehe auch Kapitel 6.1 Prämissen.

4.2. Spring Web Flow

Spring Web Flow ist ein Framework für die Ablaufsteuerung von Anwendungsfällen innerhalb von Web-Anwendungen. Ein solcher „Flow“ innerhalb von Spring Web Flow ist eine Abfolge zusammenhängender Masken, wie z.B. das Durchlaufen der Schritte zur Registrierung eines neuen Benutzers in einer Web Anwendung.

Ein erklärtes Ziel von Spring Web Flow ist die Unterstützung der Browser-Navigation, die in der Web-Entwicklung immer wieder zu Problemen führt. Das Framework übernimmt dabei die Navigation zwischen den einzelnen Views und stellt darüber hinaus einen eigenen Scope Container für Model Beans zur Verfügung. Dieser erweitert die Web-Anwendung um die folgenden Scopes:

- Flash: Gültig, solange der Flow aktiv ist, jedoch werden Flash Scope Beans nach jedem View-State geleert und dienen somit dazu, Daten zwischen zwei User Events zu transferieren.
- Flow: Steht über die gesamte Laufzeit des Flows zur Verfügung.
- Conversation: Die Lebensdauer ist mit dem Flow Scope identisch, nur stehen Conversation Scope Beans auch in den zugehörigen Subflows zur Verfügung.

Spring Web Flow implementiert im Kern einen finiten Zustandsautomaten, der auf definierte Anfangs- und Endzustände angewiesen ist.

Der Ablauf der logisch zusammenhängenden Views wird in einem so genannten Flow definiert. Innerhalb eines Flows stehen verschiedene States zur Verfügung. Zunächst muss jeder Flow einen Start State und einen End State besitzen. Der Start State definiert den Einstiegspunkt und aktiviert den jeweiligen Flow. Dieser bleibt so lange aktiv, bis ein End State erreicht wird.

Wie bereits erwähnt, setzt sich Spring Web Flow das Ziel, die Browser-Navigation mit „Back-/Forward-Button“ zu unterstützen. Um diese Funktionalität zu gewährleisten, muss die Möglichkeit bestehen, den Zustand einer View zu speichern und wieder abzurufen. Hierfür steht ein so genanntes Repository zur Verfügung, welches die Zustände der einzelnen Views innerhalb eines Flows zwischenspeichert. Dadurch kann man den Zustand jeder View innerhalb eines Flows zu einem beliebigen Zeitpunkt reproduzieren.

4.2.1 Flows / Subflows

Ein Flow kann wahlweise als XML-Datei oder mittels der Java-API realisiert werden. Ein Flow besteht in der Regel aus mehreren Zuständen (innerhalb von Web Flow als States bezeichnet), die nacheinander und in Abhängigkeit von der jeweiligen Benutzerinteraktion durchlaufen werden.

Auch die Modularisierung von Flows in kleine Einheiten ist durch so genannte Subflows bzw. Inline-Flows ohne weiteres möglich. Ein Subflow

wird wie jede andere Flow-Definition erstellt. Der Unterschied zu einem normalen Flow liegt lediglich darin, dass der Subflow innerhalb eines Flows aufgerufen wird. Eine Flow-Definition kann beliebig viele Subflows enthalten, welche wiederum weitere Subflows aufrufen können.

4.2.2 Back-Button Handling

Während der Ausführung von Flows werden die Variablen mit einer Zwischenspeicherung in das Repository geschrieben. Hierbei wird immer, wenn ein Flow durch eine User-Interaktion unterbrochen wird, der aktuelle Status gespeichert. Das Repository liefert diesen bei der Fortsetzung des Flows zurück. Der dafür notwendige „Flow-Execution-Key“, der Schlüssel, der zur Identifikation des aktuellen Flow-Status dient, wird hierbei von Spring Web Flow erzeugt.

Dieses Speichern des Status von vorhergehenden Schritten im Flow unterstützt so in Kombination mit einem „Post Redirect Get“ Mechanismus (PRG-Pattern) die Nutzung des Back-Buttons im Browser. Da jeder Request im Flow einen eindeutigen Execution Key an den Server sendet, kann, wenn man im Flow zurückgeht, auch der alte Status zum Zeitpunkt dieses Request angezeigt werden, selbst wenn der Server von dem Click auf den Back-Button selbst nichts mitbekommt.

4.3. Spring Dependency Injection

Das Spring Framework ist ein Java EE Framework. In ihm werden die Bestandteile eines Systems als „Beans“ definiert. Neben seiner Kern-Funktionalität, der Verwaltung, Konfiguration und aspektorientierten Erweiterung von Beans, bietet Spring viele Funktionalitäten, welche die Entwicklung einer Anwendung erleichtern.

Für das Web-GUI findet primär der Basisteil von Spring Verwendung, in welchem der Anwendung über Dependency Injection Beans zur Verfügung gestellt werden [Spring].

4.4. Transaktionsbehandlung

Die GUI-Komponente und der Anwendungskern (AWK) sind Teil derselben Web-Applikation und werden per Spring-Konfiguration miteinander verbunden.

Oft gibt es den Fall, dass über die GUI eine Aktion in einer anderen Anwendung ausgelöst werden soll. Ein Beispiel dafür ist die GUI einer Fachanwendung zur Datenerfassung, wobei die Speicherung der Daten über einen Service einer anderen Fachanwendung implementiert ist. In diesem Fall enthält der Anwendungskern der Fachanwendung zur Datenerfassung nur wenig Funktionalität: in ihm werden die Daten für den Serviceaufruf der nachgelagerten Fachanwendung aufbereitet und der Serviceaufruf selbst durchgeführt. Wichtig in diesem Fall ist, dass es nach Zielarchitektur keine Transaktionen über Serviceaufrufe hinweg gibt.

In diesem Abschnitt wird die Behandlung von Transaktionen innerhalb einer Anwendung beschrieben. Grundregel dabei ist, dass die Komponente GUI die Transaktion steuert. Dabei muss die Komponente GUI die Brücke schlagen zwischen der *fachlichen Transaktion*, die dem Nutzer dargestellt wird und der *technischen Transaktion,* die in der Datenbank abgebildet wird.

Die fachliche Transaktion entspricht einem Dialogablauf. Ein Beispiel dafür: Der Nutzer kann in der Regel über mehrere Masken hinweg Daten eingeben. Abschließend drückt er in ein Dialog den „OK“- bzw. den „Abbrechen“-Button. Für den Nutzer ist klar, dass alle die von ihm eingegebenen Daten im Sinne einer Transaktion behandelt werden müssen, d. h. sie werden entweder vom System komplett übernommen oder komplett verworfen.

Aus technischer Sicht ist die Behandlung dieses Ablaufs etwas komplizierter: Die Daten, die der Nutzer in den verschiedenen Dialogen eingibt, müssen zunächst zwischengespeichert werden, bevor dann bei Betätigung eines Buttons die technische Transaktion in der Datenbank erfolgt. Das Zwischenspeichern der Werte benötigt allerdings ebenfalls technische Transaktionen. Da der Prozess der Web-Anwendung zustandslos ist, muss das Zwischenspeichern ebenfalls in der Datenbank erfolgen. Hier muss die GUI zusätzliche Transaktionen durchführen.

Bei der Spring Web Flow Integration wurde ein Mechanismus verwendet, um die Zwischenwerte und Informationen zum Dialogablauf in der Datenbank abzulegen. Das Zwischenspeichern erfolgt grundsätzlich in einer separaten Transaktion. Somit beeinflussen sich die fachliche Transaktion und die technischen Transaktionen nicht.

Mit den technischen Transaktionen ist es jetzt möglich, „Sitzungen“ abzubilden. Eine Sitzung ist letztendlich die Summe aller Zwischendaten, die der Nutzer eingegeben hat oder die das System selbst erzeugt hat, wie z. B. interne Zustände, Nutzerinformationen, etc. Innerhalb einer Sitzung werden mehrere fachliche Transaktionen durchgeführt.

Das technische Mittel zur Repräsentation einer Sitzung ist zunächst einmal die Session des Servers. Diese Session ist transient. Da der Serverprozess zustandslos ist, muss sie in der Datenbank persistiert werden. Dazu gibt es zwei Alternativen:

- Serialisierung der Session nach Beendigung des Request und Wiederherstellung bei neuerlichem Aufruf
- Speichern des Spring Web Flow State an den durch Spring Web Flow vorgesehenen Hooks

4.5. Die Variante der Session Serialisierung ist zwar einfacher, beinhaltet aber auch eine wesentliche Gefahr. Die Session des Servers wird zum Speichern von verschiedensten Daten genutzt, der Zugriff auf sie ist frei möglich. Dies führt in der Praxis dazu, dass unkontrolliert große Datenmengen in der

Session abgelegt werden. Diese großen Datenmengen lassen sich dann nicht mehr effizient persistieren. Daher wurde diese Option in [IsyFactReferenzarchitektur] ausgeschlossen. Die Details dazu, wie in Spring Web Flow die zu speichernden Daten einer Session ermittelt werden, finden sich in Kapitel 7.8 Konfiguration des Ressource Caching Mechanismus

Möchte man Webressourcen wie Bilder oder CSS-Dateien von Softwareversionen abhängig machen, muss das folgende gemacht werden:

4.5.1 JSF ResourceHandler einrichten

Die Ressourcenlieferung wird durch so genannte *ResourceHandler* verkapselt. Um also den Prozess anpassen zu können, muss man einen eigenen *ResourceHandler* definieren und an die Web Applikation anbinden. Da jedoch das GUI mittels JSF erstellt wird, muss es JSF-spezifisch umgesetzt werden. Der folgende Code ist ein Beispiel so eines Handlers:

```
public class VersionierungResourceHandler extends ResourceHandlerWrapper {


    private ResourceHandler wrapped;
    private SystemVersionBean systemVersionBean;

    public VersionierungResourceHandler(ResourceHandler wrapped) {
        this.wrapped = wrapped;
        if (StatischerKontextInhaber.getApplicationContext() != null) {
            this.systemVersionBean =
                StatischerKontextInhaber.getApplicationContext()
.getBean(SystemVersionBean.class);
        }
    }

    @Override
    public Resource createResource(String resourceName) {
        return createResource(resourceName, null, null);
    }

    @Override
    public Resource createResource(String resourceName, String libraryName) {
        return createResource(resourceName, libraryName, null);
    }

    @Override
    public Resource createResource(String resourceName, String libraryName,
String contentType) {
        final Resource resource = super.createResource(resourceName,
libraryName, contentType);
        if (resource == null) {
            return null;
        }

        return new ResourceWrapper() {

            @Override
            public String getRequestPath() {
                String requestPath = super.getRequestPath();
                String version = "v=" + getSystemVersion();
                if (!requestPath.contains("?")) {
                    return requestPath + "?" + version;
                }
                return requestPath + "&" + version;
            }

            @Override
            public Resource getWrapped() {
                return resource;
            }
        };
    }

    @Override
    public ResourceHandler getWrapped() {
        return this.wrapped;
    }
```

```
    private String getSystemVersion() {
        if (this.systemVersionBean == null &&
StatischerKontextInhaber.getApplicationContext() != null) {
            this.systemVersionBean =
                StatischerKontextInhaber.getApplicationContext()
.getBean(SystemVersionBean.class);
        }
        if (this.systemVersionBean == null) {
            this.systemVersionBean = new SystemVersionBean();
            this.systemVersionBean.setSystemVersion("DEV");
        }
        return this.systemVersionBean.getSystemVersion();
    }
}
```

In der Methode `getRequestPath()` kann man die Pfade der Ressourcen beliebig anpassen und zum Beispiel, wie oben gezeigt, eine Version an sie anhängen.

4.5.2 Konfiguration

Der *ResourceHandler* muss noch mittels Konfiguration an die Web Applikation angebunden werden. Dies macht man in `faces-config.xml` auf die folgende Weise:

```
    <application>
        <resource-handler>pfad.VersionierungResourceHandler<resource-handler/>
    </application>
```

4.5.3 JSF Tags benutzten

Es gibt noch eine letzte Bedingung die man erfüllen muss, damit der JSF *ResourceHandler* die Ressourcen überhaupt behandelt. Es müssen JSF Tags benutzt werden:

- JavaScript: `<h:outputScript>` anstatt `<script>`
- CSS: `<h:outputStylesheet>` anstatt `<link>`

Dank ihnen werden die jeweiligen Ressourcen für unseren *ResourceHandler* sichtbar und werden von ihn nun verwaltet.

Session Behandlung.

4.6. JQuery

JQuery ist ein JavaScript-Framework, das auf einfache Weise JavaScript-Funktionen bereitstellt, die insbesondere auf die grafische Gestaltung einer Oberfläche benötigt werden. Die JavaScript-Datei, die diese Funktionen enthält wird im folgenden JQuery-Bibliothek genannt.

Die Homepage jquery.com bietet die Möglichkeit Module individuell zusammenzustellen, so dass nur die benötigten Funktionen zur Verfügung stehen. Für die IsyFact wurde eine Auswahl der nutzbaren Module zusammengestellt.

Erklärtes Ziel ist es, die Oberfläche durch den Einsatz von JavaScript eleganter nutzbar zu machen. Besonderer Fokus liegt dabei auf den

Sicherheitsaspekten, die eine Aktivierung von JavaScript mit sich bringt. Die Oberfläche muss jedoch auch mit deaktiviertem JavaScript mit Komforteinschränkungen nutzbar sein.

JQuery ist modular aufgebaut. Folgende Module dürfen eingesetzt werden[1]:

- jquery-core (Kernfunktionalität zur DOM-Manipulation)
- jquery-effects (Ein- und Ausblendfunktionalität)
- jqueryui-datepicker (Kalender-Widget)
- jquery-validation-plugin (Datenvalidierung)

Die Module jquery-data und jquery-ajax werden explizit nicht gesetzt, da AJAX-Funktionalität im Hinblick auf die eingesetzte Seitenlogik mit Spring-Web Flow nicht angeboten werden soll.

4.7. Bootstrap

Bootstrap ist ein Open-Source CSS Framework, welches im Web sehr weit verbreitet ist. Es bietet z.B. Funktionalitäten für das Layouting, Scaffolding und kann dynamisch auf die vorgegebene Fenstergröße reagieren (Responsive CSS).

Über eine ergänzende JavaScript Bibliothek (welche selbst wiederum JQuery nutzt), stellt das Framework auch Komponenten wie Navigations-Menüs und Dropdowns zur Verfügung.

[1] Fertige jQuery-Pakete inkl. Stylesheet sollten in [IsyFactJQuery] abgelegt sein. Sie werden jedoch nicht als Teil der IsyFact ausgeliefert.

5. Architektur

Im Folgenden soll eine grobe Übersicht über die Zusammenhänge der Web-GUI-Architektur und deren Einbettung in die Referenzarchitektur der IsyFact gegeben sowie die grundlegende Architektur der GUI-Komponenten erläutert werden.

5.1. Referenzarchitektur einer Fachanwendung

Das Nutzungskonzept für das Web-GUI nimmt Bezug auf die in der Referenzarchitektur vorgegebenen Schichten und Komponenten einer IsyFact-konformen Fachanwendung.

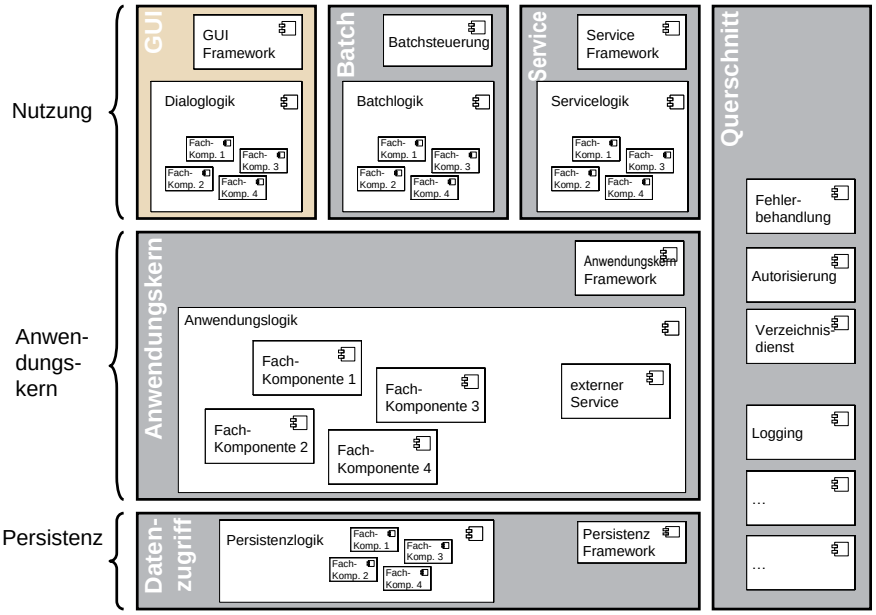

Abbildung 4: Referenzarchitektur einer Fachanwendung

Die Schicht der Nutzung ist eine Erweiterung der klassischen 3-Schichten Architektur, in der die oberste Schicht in GUI, Batch und Service differenziert wird.

Das vorliegende Dokument beschreibt die Ausgestaltung der Komponente „GUI“. Aufgabe der GUI ist es, die Funktionalität der Anwendung für einen menschlichen Nutzer zur Verfügung zu stellen. Dazu stellt sie die benötigten Dialoge und Masken bereit.

Die GUI ist untergliedert in ein GUI-Framework (verwendet wird JSF mit Spring Web Flow) und die Dialoglogik, welche die für den Anwendungsfall notwendigen Anforderungen umsetzt [IsyFactReferenzarchitektur].

5.2. Grundprinzipien der Web-GUI

Die Architektur der GUI ist durch die Eigenschaften der eingesetzten Frameworks JSF und Spring Web Flow weitgehend vorgegeben. Darin sind diese grundlegenden Prinzipien enthalten:

- Nutzung des MVC-Patterns

- Trennung des Dialogs in Dialogsteuerung und Präsentation
- Dialogsteuerung über das Spring Web Flow Framework
- Bildung von gekapselten GUI-Komponenten
- Präsentation über JSF und Facelets
- Verwaltung von Nutzer Sessions über Spring Web Flow
- Interaktive Oberflächenelemente mit JQuery und JQueryUI

5.3. Integration mit dem Framework Spring Web Flow

Abbildung 5 zeigt die Komponenten für die Web-GUI und die Integration mit dem Framework Spring Web Flow.

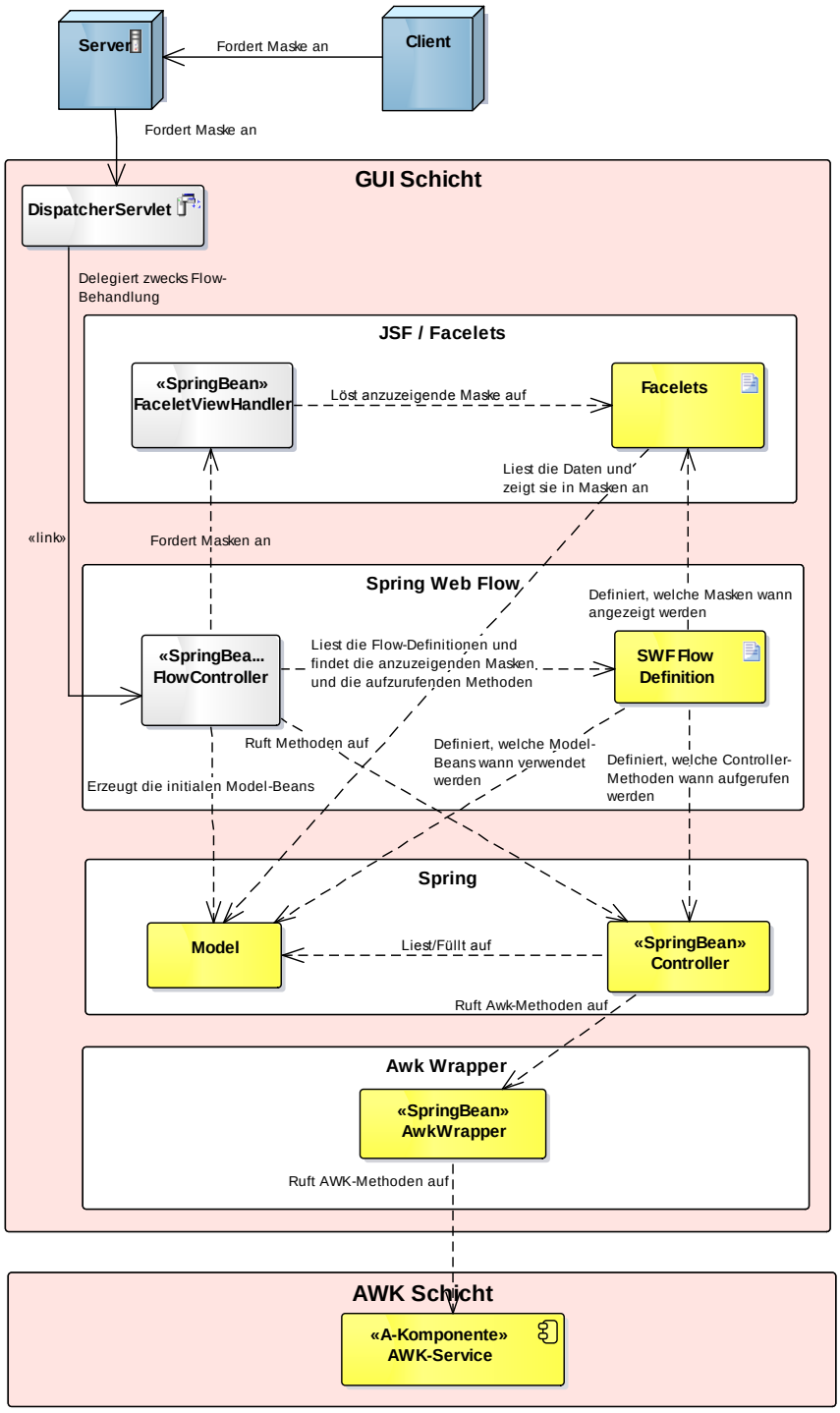

Abbildung 5: Integration von Spring bzw. Spring-Web-Flow

Durch den Programmierer einer GUI sind die gelb hervorgehobenen Teile bereitzustellen (die anderen Bestandteile werden durch das Framework bereitgestellt).

- Konfiguration des Dialogablaufs als Flow in Form von XML-Dateien
- Model und Controller-Beans zur Datenhaltung und für GUI-Logik

- Visualisierung durch Facelets in XHTML-Dateien

5.4. GUI-Komponenten

Zur Strukturierung von Masken und zugehöriger Funktionalität verwenden wir ein einheitliches Muster zum Aufbau von GUI-Komponenten. Neben der Anwendung des MVC-Pattern mittels der oben beschriebenen Web-Technologien (Spring Web Flow, JSF) definiert es zusätzlich Regeln, die eine Kapselung -, einen einheitlichen Aufbau und eine einheitliche Interaktion von GUI-Komponenten ermöglicht.

Die Fachkomponenten einer Anwendung ergeben sich aus der Systemspezifikation. Diese werden auf der Ebene Persistenz, Anwendungskern und GUI implementiert (siehe Abbildung 4). Auf Ebene der GUI sprechen wir von GUI-Komponenten (siehe Abbildung 6).

Die GUI-Komponenten umfassen für jeden Dialog eine Subkomponente. Jeder Dialog aus der Systemspezifikation ist also ebenfalls eine eigene Komponente.

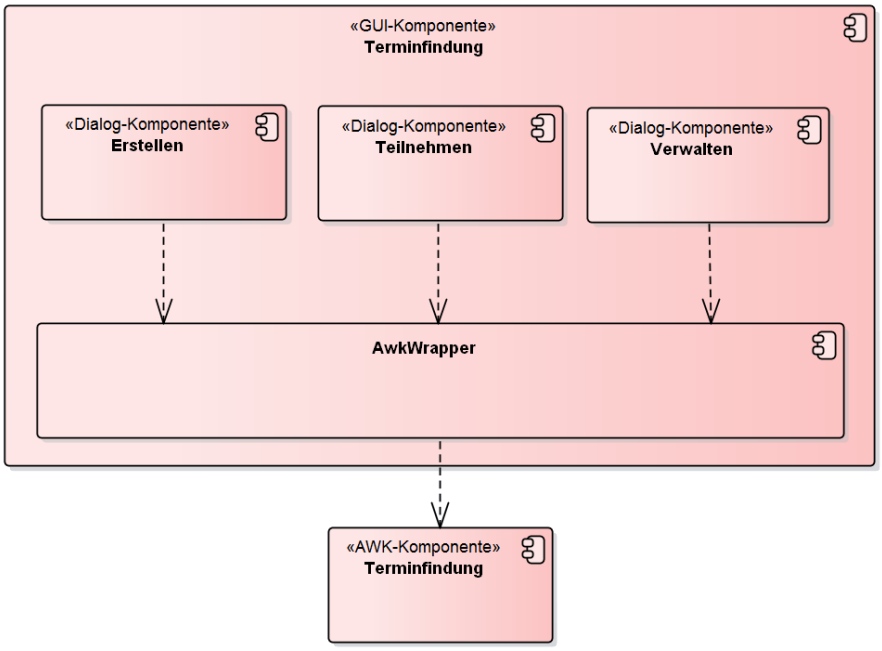

Abbildung 6: Komposition von Dialogen zu GUI-Komponenten

Die Dialog-Komponenten einer GUI-Komponente können einen gemeinsamen AWK-Wrapper und in ihren Modellen gemeinsame Datenobjekte verwenden. Trotzdem sind die Dialog-Komponenten zu kapseln, d.h. Controller- und Model Beans dürfen nicht gemeinsam verwendet werden (siehe Abbildung 7).

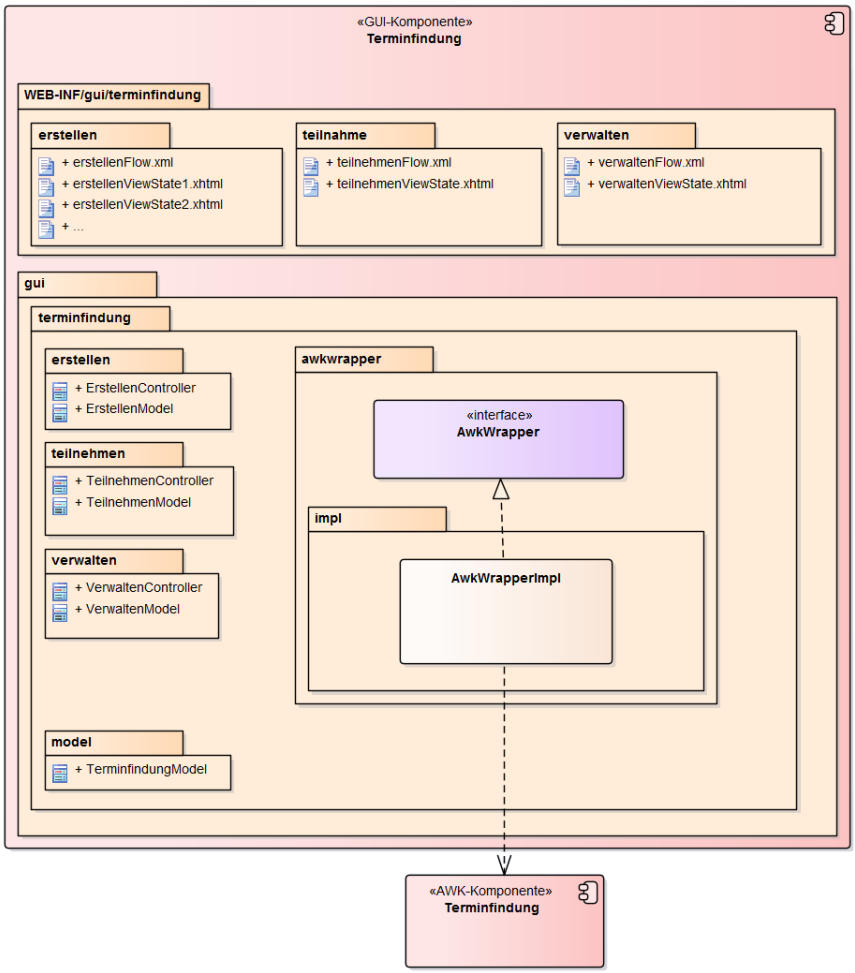

Abbildung 7: Innensicht einer GUI-Komponente mit ihren GUI-Sub-Komponenten

Zentral ist also die Forderung, dass die Elemente jeder GUI-Komponente (Flow, Controller, View und Model) in definierter Weise ausschließlich untereinander kommunizieren und Zugriffe auf Elemente anderer Komponenten unterbleiben. Abbildung 8 zeigt die Abhängigkeitsbeziehungen innerhalb einer GUI-(Sub-)Komponente.

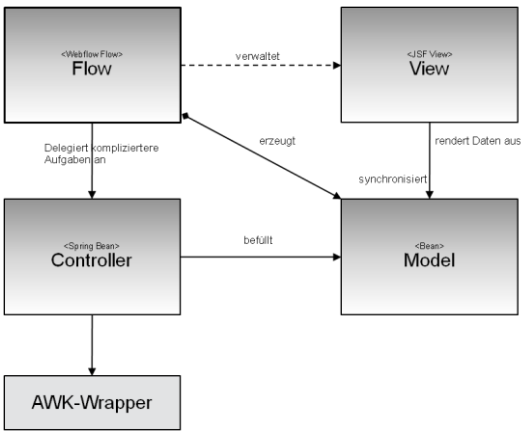

Abbildung 8: Kommunikation von View, Controller und Model innerhalb einer GUI-Komponente

5.4.1 Flow als zentraler Controller

Jede GUI-Komponente wird durch einen Flow beschrieben. Dieser definiert das Zustandsmodell der Komponente und hat die Funktion des zentralen Controllers für diese Komponente. Er erfüllt die folgenden Aufgaben:

- Erzeugung und Verwaltung eines (ggf. auch mehrerer) Model Beans
- Definition des Flow-Ablaufs in Form eines Zustandsautomaten mit Zuständen und Zustandsübergängen (Flow, Subflows, Decision-States, Action-States, Event-Handlers)
- Anbinden des Views
- Steuerung der Verarbeitung im Rahmen von Zustandsübergängen

Der Flow-Aufbau wird so gestaltet, dass im Flow alle Zustände, Zustandsübergänge sowie Aufrufe von Verarbeitungslogik zentral gebündelt werden und Ablauf und Verhalten des Flows für den Entwickler klar nachvollziehbar sind.

Der Flow wird als XML-Datei im Ordner der Komponente hinterlegt.

5.4.2 Controller-Bean

Das Controller-Bean ist ein vom Komponenten-Flow aufzurufendes **zustandsloses** Spring Bean, welches Änderungen an den Daten des Models vornimmt oder diese aufbereitet bzw. Services des Anwendungskern-Wrappers aufruft. Das Model Bean wird dem Controller mit jedem Aufruf übergeben.

Architekturkonvention: Die Implementierung des Controllers ist zustandslos und stellt nur Methoden bereit.

Das Controller-Bean wird im Spring IoC-Container mit Singleton Scope erzeugt und konfiguriert.

Das Controller-Bean wird vom Flow per Expression-Language aufgerufen. In bestimmten Fällen (siehe 5.4.4) wird ein Controller-Bean auch in einer Action (oder ActionListener) des Komponenten-View aufgerufen.

5.4.3 Model Bean

Das Model Bean ist ein Datenobjekt (einfaches POJO) und hält die Daten einer GUI-Komponente. Es hat keine Abhängigkeiten zu View, Controller oder Anwendungskern und enthält im Regelfall keine Logik. Das Model Bean wird durch den Flow erzeugt (durch Definition einer Web Flow-Variablen) und ist somit automatisch im View sichtbar.

```
<flow ...">
    <!--  Erzeuge das Model zur Benutzung durch diesen Flow. -->
    <var name="erstellenModel"
     class="de.msg.terminfindung.gui.terminfindung.erstellen.ErstellenModel" />
```

Abbildung 9: Erzeugung einer Model-Instanz im Flow

Der View liest die Daten zur Präsentation der Webseite aus dem Model Bean. Dies können Informationen zur Ansicht aber auch änderbare Formularinhalte sein. Werden Formularinhalte in Form eines Post-Requests auf den Server gesendet, so sorgt JSF eigenständig dafür, dass die Formularinhalte in das Model Bean rückübertragen werden.

Da das Model Bean durch den Flow erzeugt wird und Flow Scope besitzt, wird es automatisch mit in die Session-Persistierung einbezogen. Dazu muss das Model das Interface Serializable implementieren. Die Daten des Models werden bei den Dialogschritten eines Flow zwischen Client (Browser) und Server transparent für den Entwickler abgeglichen.

Das Model Bean ist nicht mit den JPA-Datenobjekten verbunden. Das Schreiben in das Model bewirkt also zunächst keine Änderung in der Datenbank. Die Persistenz fachlicher Datenobjekte wird über das Controller-Bean ausgelöst, welches über Methodenaufrufe des Anwendungskern-Wrappers fachliche Daten persistiert.

Die Verwendung von Model Beans wird im Verlauf dieses Dokuments noch genauer beschrieben.

5.4.4 View

Der Komponenten-View präsentiert die Daten der Anwendung in Form von generierten HTML-Seiten. Dazu werden ein oder mehrere Facelets verwendet, die mittels JSF-HTML-Tags auf das Model Bean der Komponente zugreifen, um die Daten in den View einzubinden. Da das Model Bean seine Datenzugriffsmethoden nach dem Bean-Standard (get/set/is) anbietet, kann mittels Value-Expressions *#{teilnehmenModel.terminfindung.tage}* direkt auf Eigenschaften des Model Beans und enthaltener Objekte zugegriffen werden. Ein View kann auch auf mehrere zum Flow gehörende Model Beans zugreifen.

Im View können Actions definiert sein (z.B. Submit durch einen Command-Button). Dabei werden nur Action-Tokens (String, der die Aktion benennt) übergeben, die dann im Flow entgegengenommen werden und dann Methodenaufrufe auf dem Controller auslösen.

Achitekturkonvention. Aus einer Action des Views sollte i.d.R. immer ein Zustandstoken zur Steuerung von Transitionen im Flow erzeugt werden. Dies ist vor allem bei Maskenübergängen und fachlichen Aktionen zu verwenden. Beispiel: Suche in einem Formular, Öffnen der Detailansicht.

Aktionen, welche zur Steuerung der Darstellung innerhalb einer Maske verwendet werden, müssen nicht zwingend eine Transition auslösen. In diesen Fällen darf der Controller direkt aufgerufen werden. Beispiel: Selektion eines Elements und darauf basierende Anpassung der Maske.

Die Erstellung von Views wird im Verlauf dieses Dokuments noch genauer beschrieben.

5.4.5 JQuery

Die oben beschriebene JQuery-Bibliothek wird im Seitenrahmen eingebunden. Sollen nun in einem View interaktive Elemente aktiviert werden, wird eine JavaScript-Datei mit dem Namen des Views benötigt und am Seitenende (Ende des Templates) eingebunden.

```
<script type="text/javascript"
  src="#{facesContext.externalContext.requestContextPath}/js/vorgangSuchen.js">
</script>
```

Abbildung 10: Einbindung View-spezifischer JS-Dateien

Diese Datei enthält die benötigten JavaScript-Befehle zum Erzeugen von UI-Elementen oder zum Binden von Events an bestehende Fragmente. Von inline-JavaScript ist in jedem Fall abzusehen.

Beim Einbinden sind niemals relative Pfade zu verwenden, um die Same-Origin-Policy zu forcieren. Zusätzlich sorgt, das script-Tag dafür, dass im Fall von deaktiviertem JavaScript kein Fehler auftritt und die XHTML-Konformität erhalten bleibt.

Folgende Abbildung zeigt ein Beispiel für eine Java-Script-Datei „vorgangSuchen.js“, welche ein GUI-Element mit der ID „Geburtsdatum“ fokussiert:

```
(function(){

  $('#Geburtsdatum').focus();

})()
```

Abbildung 11: Beispiel für eine JavaScript-Datei

5.4.6 Zugriff auf Anwendungskern

In einer GUI-Komponente werden grundsätzlich keine Klassen des AWK verwendet. Stattdessen wird vom Controller-Bean der Komponente (und nur von diesem) auf den zur GUI-Komponente gehörenden Anwendungskern-Wrapper zugegriffen, der den Anwendungskern aufruft. In den Models der GUI-Komponenten werden eigene Datentypen und nicht die des Anwendungskerns verwendet. Die Aufgabe des AWK-Wrappers ist die Daten vom Anwendungskern in die der GUI zu mappen.

Die Transaktionssteuerung findet im AWK-Wrapper per Annotationen an der Wrapperklasse statt.

```
@Transactional(rollbackFor = Throwable.class, propagation=Propagation.REQUIRED)
public class AwkWrapperImpl implements AwkWrapper {
```

Abbildung 12: Deklaration des Transaktionsverhaltens am AWK-Wrapper

Damit die Persistierung funktioniert, müssen die AWK-Wrapper-Beans im selben Spring-Applikationskontext wie der Anwendungskern definiert werden, damit der Transaktionskontext aus der Hibernate-Konfiguration nutzbar ist.

5.4.7 Schnittstellen zwischen Komponenten

Ein Grundprinzip der Architektur der GUI-Komponenten ist die Kapselung aller Komponenten. Ein View oder Controller einer GUI-Komponente darf daher nicht auf das Model (oder das Controller-Bean) einer anderen GUI-Komponente zugreifen. Der Austausch von Informationen erfolgt stattdessen über Input/Output-Elemente im Flow, die aus dem Model einer GUI-Komponente gelesen oder geschrieben werden.

Ist ein Subflow B mit Daten aus dem aufrufenden Flow A zu versorgen, so bekommt dieser nicht das Model Bean A, sondern eine Kopie eines einzelnen Objekts (kann auch eine Datenstruktur, aber niemals das gesamte Model A sein) aus A übergeben. Es ist wichtig, dass eine Kopie übergeben wird, damit Flow B nicht Teile des Model Beans A absichtlich oder versehentlich ändert.

Besteht Bedarf, dass ein Subflow B an den aufrufenden Flow A Daten zurückgibt, so erfolgt dies über ein Output-Element. Hier gilt analog, dass nicht das gesamte Model Bean B, sondern lediglich Einzelwerte (Kopie) übergeben werden.

Das folgende Beispiel zeigt wie ein Flow an einen Subflow Parameter übergibt und von diesem einen Ausgabewert empfängt.

```
<subflow-state id="loeschenViewState" subflow="loeschenFlow">
    <input name="terminfindung"
           value="verwaltenController.kopiereTerminfindungModel()"/>
    <output name="loeschenTerminfindung"/>
    <transition on="finished" to="verwaltenViewState">
        <evaluate
          expression="verwaltenModel.setTerminfindung(loeschenTerminfindung)"/>
    </transition>
</subflow-state>
```

Abbildung 13: Informationsaustausch zwischen Flows – aufrufender Flow

Innerhalb des Subflows werden übergebene Parameter entgegengenommen und verarbeitet. Im Endzustand wird ein Rückgabewert zurückgegeben.

```
<input name="terminfindung"
type="de.msg.terminfindung.gui.terminfindung.model.TerminfindungModel"/>

<on-start>
    <evaluate expression="loeschenModel.setTerminfindung(terminfindung)"/>
</on-start>

<view-state id="loeschenViewState">
    <on-entry>
        <evaluate
          expression="loeschenController.setzeAuswahlZurueck(loeschenModel)"/>
    </on-entry>
    <transition on="cancel" to="finished"/>
    <transition on="delete" to="loeschenViewState">
      <evaluate
        expression="loeschenController.loescheZeitraeume(loeschenModel)"/>
    </transition>
</view-state>

<end-state id="finished">
    <output name="loeschenTerminfindung"
            value="loeschenModel.getTerminfindung()"/>
</end-state>
```

Abbildung 14: Informationsaustausch zwischen Flows – aufgerufener Flow

Zur Datenübergabe können auch mehrere Input und mehrere Output-Elemente verwendet werden.

Für die Steuerung des Vorgabelayouts (z.B. Menüleiste, Linksnavigation) sowie der Nutzung von vorgegebenen Funktionen (z.B. Validierung) werden auch querschnittliche Controller mit zugehörigen Models verwendet. Die Instanziierung übernimmt dabei ein übergeordneter Parent-Flow. So kann z.B. die Seitentoolbar ausgeblendet oder ein Quicklink hinzugefügt werden.

Der Aufruf dieser Controller zur Steuerung des Verhaltens ist erlaubt. Auf die Controller kann per Spring zugegriffen werden. Welche Controller im Detail für das Vorgabelayout verfügbar sind, wird in [Styleguide] aufgelistet.

5.4.8 Packaging und Namenskonventionen

Ein nicht zu vernachlässigender Aspekt zur Komponentenbildung ist die Paketierung, durch die zu einer Komponente gehörende Elemente gruppiert abgelegt werden. Alle Elemente werden in einem Paket mit einheitlichem Paketnamen abgelegt.

Für die Namenskonvention zu Java-Klassen und Paketen wird hier auf das Dokument [Programmierkonventionen] verwiesen. Zusätzlich gelten die folgenden Konventionen:

- Jede GUI-Komponente hat einen Namen. Die Namen richten sich nach den fachlichen Komponenten bzw. Dialogen.
- Das Paket, in dem die GUI-Komponente abgelegt wird, trägt den vollständig kleingeschriebenen Namen der GUI-(Sub-)Komponente (z.B. `erstellen`). Jede GUI-Komponente nutzt zwei Ablageorte:
    - java/de/…/gui/terminfindung/erstellen/… für Java-Klassen
    - WEB-INF/gui/terminfindung/erstellen/… für Flows und Views
- Model Bean-Klassen tragen den Namen der GUI-Komponente und enden auf Model (z.B. `ErstellenModel`).
- Controller Bean-Klassen tragen den Namen der GUI-Komponente und enden auf Controller (z.B. `ErstellenController`).
- Flows tragen den Namen der GUI-Komponente und enden auf Flow.xml (z.B. `erstelllenFlow.xml`).
- Der Main-View, der dem Flow-View-State zugeordnet ist endet auf `ViewState` (z.B. `erstellenViewState.xhtml`). Besteht der Flow aus mehreren View-States, so wird eine Schritt-Nummer angehängt (z.B. `erstellenViewState1.xhtml`).
- Alle weiteren für den View verwendeten Facelets tragen den Namen der Komponente und eine Charakterisierung des Facelets (z.B. `erstellenFormular.xhtml`). Auch hier ist eine Schrittnummer anzuhängen, wenn der Flow mehrere View-States enthält (z.B. `erstellenFormular1.xhtml`).
- Die bei einem View-State verwendete JavaScript-Datei trägt den Namen des View-States (z.B. `erstellenFormular1.js`). Gibt es View-übergreifende Funktionalität kann diese in eine wiederverwendbare JavaScript-Datei ausgelagert werden (z.B. `erstellenForumlar.js`).

Im folgenden Abschnitt ist die Benennung der Elemente auch noch einmal in Form der Projektdateistruktur nachvollziehbar dargestellt.

5.4.9 Projekt-Verzeichnis einer Fachanwendung mit GUI

Nachfolgend ist der Verzeichnisbaum der Beispiel-Implementierung (insbesondere die GUI-Komponente `Erstellen`) dargestellt, in dem zu sehen ist, wie die Elemente der GUI im Dateisystem abgelegt werden.

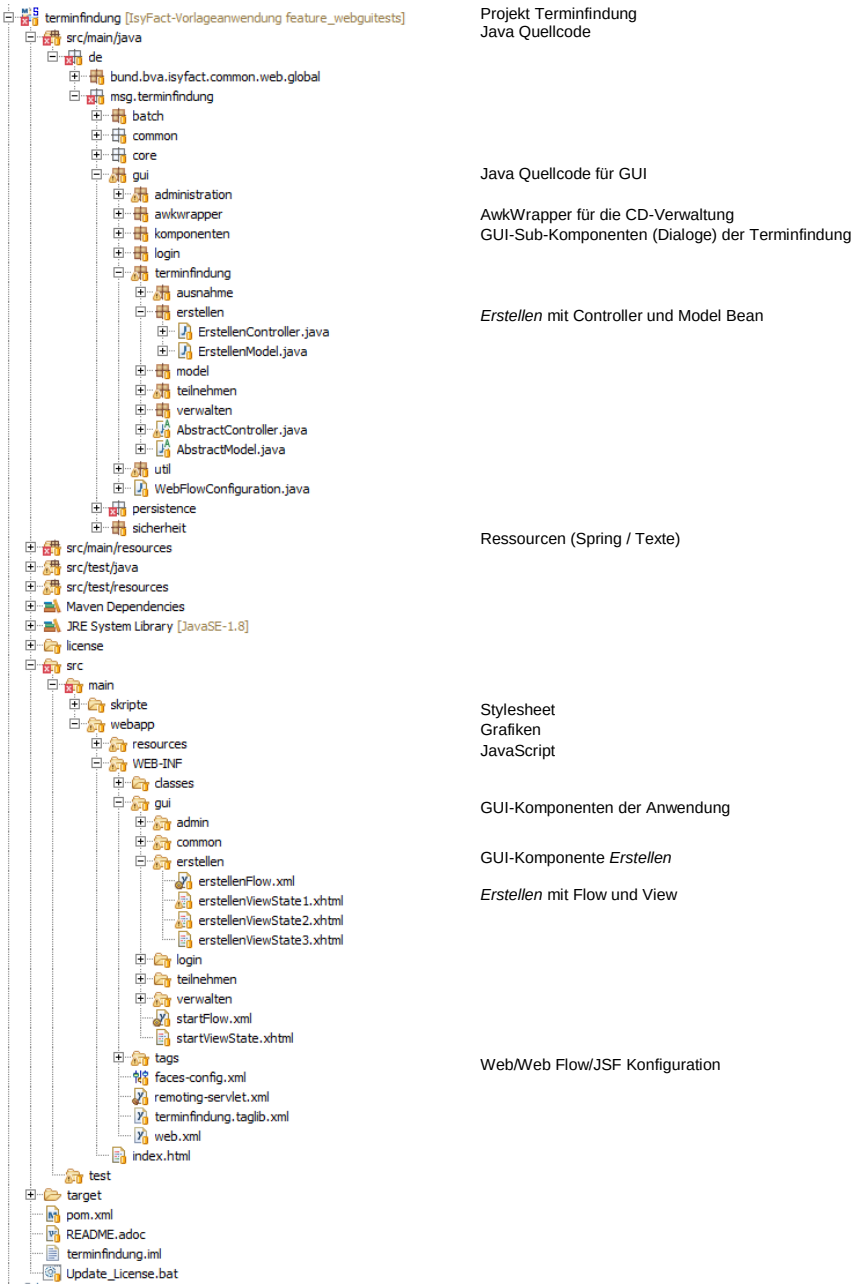

Abbildung 15: Verzeichnisstruktur am Beispiel Erstellen

6. Umsetzen der Web-GUI einer IsyFact-Anwendung

6.1. Prämissen

Für die Erstellung von Weboberflächen existieren Vorgaben in Form eines Styleguides [Styleguide]. Hierüber werden Anforderungen an die Gestaltung und Nutzbarkeit von Oberflächen, sowie technische Aspekte, wie die Verwendung von Javascript oder zu unterstützende Browser festgelegt.

6.2. Erstellung einer GUI-Komponente

6.2.1 Der Flow

Für eine GUI-Komponente wird zunächst die Definition des Flow als XML-Datei erstellt. Spring Web Flow sucht und findet den Flow selbständig und nimmt ihn in die Flow-Registry auf. Zunächst ein exemplarisches Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-webflow-2.4.xsd"
      parent="basisFlow">

    <var name="teilnehmenModel"
         class="de.msg.terminfindung.gui.terminfindung.teilnehmen.TeilnehmenModel" />
    <var name="tfRef" class="java.lang.String"/>

    <view-state id="teilnehmenViewState">
        <on-entry>
            <set name="tfRef" value="requestParameters.tfref" type="java.util.UUID" />
            <evaluate expression="tfNumberHolder.updateRefIfNotNull(tfRef)"/>
            <evaluate expression="teilnehmenController.initialisiereModel(teilnehmenModel)" />
        </on-entry>

        <transition on="save" to="teilnehmenViewState">
            <evaluate expression="teilnehmenController.speichereTeilnehmer(teilnehmenModel)"/>
        </transition>

        <transition on="saveAllFalse" to="teilnehmenViewState">
            <evaluate
                expression="teilnehmenController.speichereTeilnehmerAllFalse(teilnehmenModel)"/>
        </transition>
    </view-state>
</flow>
```

Abbildung 16: Ein vollständiger Flow am Beispiel teilnehmenFlow.xml

Charakteristisch sind hier die folgenden Elemente:

- Flow-Tag mit Deklaration aller verwendeten Taglibs und Namespaces, sowie die Einbindung der anwendungsübergreifend einheitlichen `parent-Flow`-Definition (enthält global gültige Regeln, Fehler-Handler und Layoutkonfigurationen).

- Definition des Models unter Angabe der Model-Bean-Klasse als Flow-Variable

- Es wird ein `onStart`-Handler für den Flow definiert. Das Model sollte immer über die standardisierte Initialisierungsmethode beim Starten des Flows initialisiert werden. Weiterhin können weitere, spezifische Methoden aufgerufen werden, um z.B. Eingabeparameter in das Model einzuarbeiten.

- Ein View-State, welcher den Hauptzustand des Flow definiert. Der Name des View-State verknüpft die Komponente auch mit dem gleichnamigen View (z.B. `teilnehmenViewState.xhtml`), der automatisch beim Rendern aufgerufen wird.

- Gegebenenfalls weitere View-States, falls die Komponente mehrere Views (z.B. aufeinanderfolgende Eingabemasken zu einem zu erfassenden Datentyp) verwaltet.

- Für jeden View-State kann ein `onEntry`-Handler definiert werden.

- Für jeden View-State werden die ausgehenden Transitionen im Sinne eines Zustandsautomaten definiert. Für jede Transition wird ein Zielzustand festgelegt. Diese können sein:

    - Der eigene View-State (zur Aktualisierung des Views).

    - Ein untergeordneter SubFlow (führt zur Anzeige einer anderen GUI-Komponente). Nach Ausführung des Subflow kehrt die Anwendung in den aktuellen Flow zurück.

    - Ein Action-State oder ein Decision-State, in denen der Flow entweder Aktionen (z.B. Aufrufe des Anwendungskerns) oder Entscheidungen zum weiteren Flow-Ablauf trifft.

    - Keine Angabe eines Zielzustands: Dadurch verbleibt der Flow im aktuellen View-State. Dieser wird aktualisiert. Der vorherige Zustand kann NICHT mehr über den Browser-Back-Button erreicht werden. Für AJAX-Aufrufe, welche nur einen bestimmten Teil der Seite aktualisieren sollen, darf kein Zielzustand angegeben werden.

  Für jede Transition kann hinterlegt werden, ob die Browser-Historie (für Back Button Handling) zurückgesetzt werden soll. Kommt man also nach Anzeigen einer Trefferliste über das Löschen eines Eintrages wieder zur Trefferliste, so sollte die Möglichkeit der Bereinigung der Historie genutzt werden.

- Beim Aufruf von Subflows müssen meist Parameter übergeben werden. Dazu wird ein Input-Tag verwendet, welches ein Schlüssel/Wertpaar an den Subflow übergibt. Im Subflow wird der Parameter über ein Input-Tag entgegengenommen und steht dann als Flow-Variable zur Verfügung und sollte im `onStart`-Handler per Controller in das Model übernommen werden.

  Wichtig: Um die Kapselung der GUI-Komponenten zu bewahren ist es wichtig, dass GUI-Komponenten ihre Parameter immer über ein Input-Tag erhalten und nicht frei auf fremde Models und Controller zugreifen.

Wichtig: Bei der Übergabe eines Parameters (z.B. Liste) ist immer eine Kopie der Datenstruktur zu übergeben, damit Änderungen an der Datenstruktur durch einen Subflow sich nicht auf den aufrufenden Flow auswirken. In diesem Sinne ist auch verboten, ein ganzes Model-Bean zu übergeben. Müssen mehrere Informationen übergeben werden, so können natürlich auch mehrere Input-Parameter verwendet werden.

Auch die Rückgabe von Out-Parametern ist über ein Output-Tag möglich. Es gelten die gleichen Richtlinien wie bei Input-Parametern.

- Ein Decision-State namens `_aktualisieren`, der für technische Zwecke u.a. bei der Fehlerbehandlung verwendet wird.

- An nahezu allen Stellen der Flow-Definition ist der Aufruf von Spring-Beans per `evaluate`-Tag möglich. Hier wird die Java-Expression-Language verwendet (beachte: es handelt sich hier nicht um die deutlich leistungsfähigere Spring Expression Language). Genutzt wird die Möglichkeit ausschließlich zum Aufruf des zustandslosen Controller-Beans – meist unter Bereitstellung des Model-Beans. Das Ergebnis kann in einer neuen Flow-Variablen hinterlegt werden (was wir aber im Allgemeinen nicht nutzen, da diese Werte im Model-Bean hinterlegt werden sollten).

Eine Flow-Definition bietet weitaus mehr Möglichkeiten, die aber im Regelfall nicht benötigt und daher hier nicht erläutert werden.

6.2.2 Der Controller

Jede GUI-Komponente verfügt über ein Controller-Bean. Dieses ist der „verlängerte Arm“ des Flow, denn im Flow kann und soll nicht programmiert werden. Jegliche zu programmierende GUI-Logik wird im Controller in zustandslosen Methoden bereitgestellt. Typische Methoden im Controller sind:

- Methoden zur Initialisierung des Models.

- Methoden zum Aufruf des Anwendungskerns.

- Methoden zur Aufbereitung von Daten des Models bevor diese gerendert werden.

Das Controller-Bean wird in der Spring Konfiguration als einfaches Spring-Bean definiert und ist somit im Flow automatisch sichtbar und nutzbar.

Das Controller-Bean muss von AbstractGuiController erben.

Da das Controller Bean zustandslos ist, muss im Regelfall bei jedem Aufruf das Model mitgeliefert werden.

Der Controller kann auch eine eigene Fehlerbehandlung enthalten, im seltenen Fall auch selbst Meldungen in den `FlowRequestContext` schreiben, die dann als Fehler- oder Hinweismeldung ausgegeben werden.

Eine Rückgabe von Zielzuständen zur Steuerung des Flow in Methoden des Controllers ist zu vermeiden. Sinnvoll ist die Rückgabe eines Ergebnistokens (Erfolg oder Fehler), um dann im Flow den Zielzustand festzulegen und dann anzusteuern. Solche Entscheidungen können im Flow auch per Action- oder Decision-State umgesetzt werden, wobei im Controller eine Methode `is…` mit Rückgabewert `boolean` verwendet wird.

Die häufig gesehene Umsetzung von einfachen `geheZu`-Methoden des Controllers, die lediglich einen Rückgabewert aus einer Konstanten zurückliefern, erbringt keinen Mehrwert. Der Wert kann auch direkt in der Flow-Definition festgelegt werden.

6.2.3 Das Model

6.2.3.1 Bereitstellung eines Models

Wie unter 6.2.1 beschrieben, wird das Model immer durch einen Flow instanziiert und verwaltet. Ein Model für eine Maske muss von AbstractMaskenModel erben.

6.2.3.2 Befüllen eines Models

Wie unter 6.2.2 beschrieben, wird das Model durch den Controller bei Bedarf initialisiert und mit Daten aus dem Anwendungskern befüllt. Auch die Aufbereitung von Daten des Models kann durch den Controller erfolgen (alternativ über View-Konverter). Das Konvertieren von Model-Inhalten durch Logik im Model-Bean soll möglichst vermieden werden. Insbesondere ist Logik zu vermeiden, bei der Fehler auftreten können. Ein Model soll vor dem Rendern möglichst alle anzuzeigenden Daten passgenau für das Rendering vorhalten.

6.2.3.3 Abgleichen eines Models

Der Abgleich des Models mit dem View (nach Submit einer Maske) erfolgt automatisch durch JSF. Alle Seiteninhalte, die beim Rendern aus einem Model gelesen wurden, werden nach dem Submit wieder in das Model rückübertragen und stehen dann zur weiteren Verarbeitung für den Controller oder erneutes Rendering zur Verfügung.

6.2.3.4 Speichern der Daten eines Models

Speichern der Session-Daten

Das Model-Bean wird vom Flow im Flow Scope gehalten. Daher wird die Datenstruktur zwischen den einzelnen Dialogschritten in der Session persistiert (Conversation-Persistierung siehe 4.2).

Für die Ablage im Flow Scope werden die Daten in serialisierter Form abgelegt. Daher muss das Model-Bean das Interface Serializable implementieren.

Größere Datenmengen beeinträchtigen die Performance der Anwendung zur Laufzeit. Umso mehr Daten im Model enthalten sind, desto aufwändiger ist die Session-Persistierung. Daher ist die Menge an gehaltenen Daten auf das Notwendige zu beschränken. Model-Members, die nur temporär während der Datenaufbereitung befüllt werden, sollten dringend als „transient“ markiert werden, um diese aus der automatischen Persistierung auszunehmen.

Hinweis: Die Session-Persistierung erfolgt neben den Dialogschritten zusätzlich auch beim Redirect im Rahmen der Anwendung des Post Redirect Get-Pattern (siehe 4.2.2).

Speichern der fachlichen Daten
Die nach einem Submit im Model-Bean gespeicherten (und vom Anwender ggf. veränderten) Daten werden nach Validierung in die fachlichen Tabellen der Datenbank übernommen. Dies erfolgt immer durch einen Controller, der die Daten aus dem Model an den Anwendungskern-Wrapper übergibt.

6.2.4 Der View

6.2.4.1 Definition des View-State

Jeder View-State in der Flow-Definition der GUI-Komponente ist mit einer eigenen Maske, dem View verknüpft. Spring Web Flow steuert das Rendering des Views.

Der View einer GUI Komponente ist nach unserem Umsetzungsmuster wie folgt aufgebaut: Eine Facelet-Datei dient dazu, die im Seitentemplate (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**) inkludierte Definition der Seitenbereiche des Inhaltsbereiches zu definieren. Das `<ui:composition>` Tag referenziert hierbei das Seitentemplate, in welchem die über `<ui:define>` definierten Teile eingebunden werden. Die verschiedenen Teile des Seitentemplates finden sich in [Styleguide].

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:sf="http://www.springframework.org/tags/faces"
        template="/WEB-INF/gui/common/layout/applikation.xhtml">

        <ui:define name="title"><h:outputText
value="#{msg.MEL_CdErwerben}"/></ui:define>

        <ui:define name="content1">
                <ui:include src="/WEB-
INF/gui/cderwerb/cdErwerbUebersicht1.xhtml" />
        </ui:define>

        <ui:define name="content2">
                <ui:include src="/WEB-
INF/gui/cderwerb/cdErwerbFormular1.xhtml" />
        </ui:define>

        <ui:define name="buttonzeile">
                <ui:include src="/WEB-
INF/gui/cderwerb/cdErwerbButtonzeile1.xhtml" />
        </ui:define>
</ui:composition>
```

Abbildung 17: Beispiel Facelet cdErwerbViewState.xhtml

Die Seitenbereiche werden im `ViewState`-Facelet wiederum durch Inklusion auf kleinere Facelets umgesetzt. Das heißt jeder Seitenbereich (Formular, Buttonzeile) wird per Konvention in einer separaten Datei gepflegt. Innerhalb dieser Facelets wird im Normalfall nichts mehr inkludiert.

```
<div class="buttonzeile"
        xmlns:h="http://java.sun.com/jsf/html">
        <h:commandButton value="#{msg.MEL_Weiter}" action="weiter"
                styleClass="btn suchen"></h:commandButton>
        <h:commandButton value="#{msg.MEL_Abbrechen}" action="abbrechen"
styleClass="btn suchen"></h:commandButton>
</div>
```

Abbildung 18: Beispiel-Facelet cdErwerbButtonzeile1.xhtml

6.2.4.2 Rendern einer Maske mit den Daten eines Models

Der Zugriff auf die Daten des Models erfolgt in den Facelets über die Common Expression Language (EL) [CommonEL]. Das Model ist im View sichtbar, da es im Flow als Flow-Variable deklariert wurde. Das Model ist zum Zeitpunkt des Renderns bereits mit Daten befüllt, da im `onEntry`-Handler des Flows der Controller die Befüllung des Models vorgenommen hat.

```
<isy:formInput reference="name" value="#{erstellenModel.name}" label="Titel der
Veranstaltung" required="true"/>
```

Abbildung 19: Datenzugriff im Facelet

Es ist wichtig zu verstehen, dass durch das Rendern der Daten aus dem Model eine Bindung der Model-Property mit dem GUI-Element (im Beispiel ein Form-Input-Feld) hergestellt wird, welches nach Submit des Webformulars automatisch durch Spring Web Flow in das Model zurücksynchronisiert wird. Damit das funktioniert ist eine eindeutige HTML-ID zu vergeben. Die HTML Elemente erhalten entsprechend ihres Inhaltes den Bezeichner des zugehörigen Attributes. Würde keine ID vergeben, so würde JSF selbständig eine dynamische ID vergeben. Das erschwert jedoch den automatischen Test der Oberfläche.

```
<h:selectOneMenu id="vonZeit" value="#{tag.vonZeitraum}" converter="calCon">
                value="#{erstellenModel.alleZeitraeume}" var="von"
                itemValue="#{von}" itemLabel="#{von}"/>
```

Abbildung 20: HTML ID Vergabe

Werden versehentlich IDs mehrfach verwendet, so sind Fehler bei der Datenübernahme wahrscheinlich.

6.2.4.3 Auslösen von Aktivitäten in Facelets

Die Auslösung von Aktion erfolgt über die Nutzung des `action`-Attributes der verwendeten GUI-Komponenten. Hier wird ein Token verwendet, welches auch im Flow bekannt ist und die Transition so steuert, dass der Controller die Daten über den Anwendungskern persistiert.

```
<isy:button action="back" value="Zurück"/>
<isy:button action="continue" value="Weiter"/>
```

Abbildung 21: Beispiele für Actions

6.2.4.4 Datenkonvertierung für Darstellung und Eingabe

Für die formatierte Darstellung von Daten können JSF-Konverter zur Konvertierung aus der Ansicht ins typisierte Datenmodell, wie auch zur Umwandlung aus dem Datenmodell in die Ansicht verwendet werden. Hier bieten sich JSF-Konverter an, die jedoch nur mit Einschränkungen verwendet werden können, da diese bei der Konvertierung „freier Eingaben“ nicht mit Fehleingaben umgehen können.

```
<h:inputText id="datum" value="#{erstellenModel.newDate}">
        <f:convertDateTime type="date" />
</h:inputText>
```

Abbildung 22: Umwandlung eines Datums mittels Konverter

Wenn die Validierung in einem JSF-Konverter stattfindet, werden die Daten in einem Fehlerfall nicht ins Modell geschrieben. Dies führt dazu, dass das Formular zurückgesetzt wird, weil die Seite wegen des PRG-Patterns mit einem GET-Request mit dem alten Modell neu geladen wird. Die ungültigen Eingaben gehen also zusammen mit allen anderen Änderungen im Modell verloren. Standard-JSF-Konverter sind also faktisch nicht nutzbar.

Ein geeigneter Konverter muss auch ungültige Daten ins Modell schreiben können. Wenn dies aufgrund der Nutzung spezieller Datentypen (wie z.B. `Date`) nicht möglich ist, muss im View-Model der Datentyp `String` verwendet werden. Die Konvertierung findet in diesem Fall nicht durch einen Konverter statt, sondern erst nach oder während der Validierung.

Oft ist es notwendig, im Model Schlüssel eines Schlüsselverzeichnisses zu verwenden. Dieser sollte in der Regel in der Maske nicht als Schlüssel, sondern in einer verständlichen Form dargestellt werden. Hier bietet sich der Einsatz eines eigenen Konverters an, der mittels Schlüssel-Wert-Mapping die Umwandlung je nach Verarbeitungsrichtung leistet. Analog gilt dies auch für Booleans und Aufzählungstypen.

```
<h:outputText value="#{cdAblageDatenBackBean.interpretMaennlich}">
        <f:converter converterId="geschlechtsTypConverter"/>
</h:outputText>
```

Abbildung 23: Umwandlung eines Aufzählungstyps mittels Konverter

Im Hinblick auf aktiviertes JavaScript darf bei `outputText` niemals das Attribut `escape` auf `false` gesetzt werden.

6.2.4.5 Interaktive Elemente mit JQuery

JQuery ist ein mächtiges Framework zur DOM-Manipulation. Entsprechend vorsichtig und gezielt sollte der Einsatz gewählt werden. In der Regel bieten die Komponenten des Styleguides [Styleguide] Zugriff auf Visualisierungsformen mittels JavaScript (z.B. Kalenderwidget, Tags, Panels). Für bestimme Zusatzanforderungen (z.B. bedingtes Deaktivieren eines Felds, weitere GUI-Verschönerungen) kann es jedoch notwendig sein zusätzliches JavaScript einzubinden.

Die View-spezifische Funktionalität wird in einer eigenen JavaScript-Datei umgesetzt. Dabei sind grundsätzlich folgende Regeln zu beachten:

Regel	**Begründung / Beispiel**

`eval()` darf nicht verwendet werden	Die Verwendung von `eval()` stellt ein Sicherheitsrisiko da. Es besteht die Gefahr, dass Werte aus Request-Parametern ohne ausreichende Prüfung als Code ausgeführt werden. Beispiel: `var requestValue = getParameterValueForParameter("searchString");` `eval(requestValue)` Dies ermöglicht es jeden beliebigen JavaScript Code per Injektion auf einem Client ausführen zu lassen. Ein Angreifer könnte dann Code auf dem Server wie folgt einschleusen: *www.mySite.de?searchString=alert(‚hallo‘)*
`setTimeout()` darf nicht in der Variante aufgerufen werden, die den Code in einer Zeichenfolge enthält. Stattdessen muss ein `function()`–Parameter übergeben werden: `setTimeout(` `  function() {` `  …` `  }` `, 100)`	Die verbotenen Varianten von `setTimeout()` stellen ebenfalls ein Sicherheitsproblem da. Die Angriffsmöglichkeit ist jedoch weniger offensichtlich. Beispiel: `setTimeout("callSomeSpecialFunktion(searchString)", 100);` Ein Angreifer könnte nun eine URL wie folgt aufrufen: *www.mySite.de?searchString=5);alert(‚hallo‘* Der Inhalt der Variable „searchString“ wird im searchString ersetzt, so dass folgender Code ausgeführt würde: `setTimeout("callSomeSpecialFunktion(5);alert("hallo"), 100);` Der Angreifer hätte es also geschafft die Ausführung der Funktion `alert("hallo")` auf dem Server zu veranlassen.
Sofern keine Wiederverwendung möglich ist, ist von der Definition benannter Funktionen abzusehen und anonyme Funktionen einzusetzen. Dies betrifft im speziellen das Event-Binding	Definierte JavaScript-Funktionen sind in der Regel im globalen Variablen-Kontext gültig. Würde für jede Callback-Funktion eine eigene Funktion definiert, würde das den Speicher unnötig belasten. Weiterhin verschlechtert sich die Lesbarkeit. Gerade bei Callbacks ist es nützlich wenn direkt ersichtlich ist, was passiert wenn der Callback aufgerufen wird. Weiterhin können anonyme Funktionen auf Variablen der umgebenen Funktion zugreifen, was die Implementierung vereinfacht:

	`var einWert=5;` `setTimeout(` `  function {` `  alert(4 + einWert);` `  }` `, 100);`
Benannte Funktionen sollten in einem Namespace deklariert werden, die eine Zuordnung zu einem View erkennen lässt: `var ns_<view> = {` `  foo : function()` `{ … }` `}`	Wie bereits beschrieben gelten Funktionen häufig im globalen Kontext. Funktionsnamen können wie Variablen durch redundante Deklaration leicht versehentlich überschrieben werden. Dann gilt immer die letzte Definition. Die Verwendung des View-Names als „Namespace“ vermeidet, Funktionen aus einem anderen View versehentlich zu „überschreiben“.
Jede JavaScript-Datei beginnt mit *`(function(){`* und endet mit `})()` Die Deklaration von Wiederverwendbaren Funktion- bzw. Namespace-Definitionen müssen außerhalb dieses Blocks erfolgen.	Mit diesem Konstrukt wird verhindert, dass (versehentlich) neue Funktions- und Variablen-Definitionen Elemente aus dem globalen Kontext überschreiben.
Inline-JavaScript ist zu vermeiden.	Es gibt Fälle in denen JavaScript „inline“ technisch bedingt direkt in der XHTML-View-Definition implementiert werden muss. Hier besteht die Gefahr, dass der JS-Script-Code schlecht strukturiert und auf zu viele Dateien verteilt wird. Zudem ist JavaScript-Code in XHTML-Dateien unerwartet und wird bei der Analyse der Anwendung schnell übersehen. Insgesamt wird hierdurch die Verständlichkeit und Wartbarkeit der Anwendung verschlechtert.
Der DOM-Zugriff mit der $-Funktion sollte stets über die Id oder Klasse eines DOM-Knotens erfolgen und nicht über die Knotenhierarchie des DOMs.	`$(„#eineBildID“);  // GUT` `$(„div span a img“);// SCHLECHT` Die Gefahr bei letzterer Variante ist die Fehleranfälligkeit auf Änderungen in der DOM-Struktur. Wird z.B. ein weiteres DIV eingefügt, greift die Funktion ggf. nicht und die Anwendung arbeitet fehlerhaft.
Event-Binding erfolgt im JavaScript-Code und nicht in den on<Event>-Attributen des HTML-Elementes.	Das Event-Binding in den on<Event>-Attributen erzeugt Inline-JavaScript, dass stets zu vermeiden ist (s.o.)

Im JavaScript-Code dürfen Request- oder URL-Parameter nur nach ausreichendem encodieren und escapen verwendet werden. Gleiches gilt für den Einsatz von Server-Parametern bzw. Model-Attributen.	Auch hier besteht ein Sicherheitsrisiko. Wird Beispielsweise ein Suchstring wie folgt in die Seite eingebunden: `<title>${searchStringFromRequest}</title>` Ein Angreifer könnte dann folgende URL aufrufen: *www.mySite.de?searchString=<script>alert(‚halloWelt‘)</script>* Der übergebene JavaScript-Block würde dann auf dem Server ausgeführt. Das Escapen „zerstört“ die spitzen Klammern und Hochkommata, so dass kein Code ausgeführt wird.

6.2.4.6 Clientseitige Validierung von Eingaben

Zur clientseitigen Validierung von Eingaben wird das jQuery Validation Plugin verwendet. Die Markierung von Pflichtfeldern und die Definition von eigenen Regeln und Hinweistexten ist unter http://blogs.fau.de/webworking/2011/05/13/tutorial-zur-eingabevalidierung-von-formularen-mit-hilfe-von-jquery/ beschrieben. Der Aufruf erfolgt dabei innerhalb der View-spezifischen JavaScript-Datei über die ID des Formulars: `$("#formular").validate();`

Da JavaScript deaktivierbar und manipulierbar ist, müssen grundsätzlich alle Validierungen auch serverseitig erfolgen.

6.2.4.7 Serverseitige Validierung von Eingaben

Die Validierung und Prüfung der in der GUI erfassten Daten soll entweder vollständig durch die GUI oder aber vollständig im Anwendungskern durchgeführt werden. Die Validierung in der GUI ist dabei bevorzugt. In diesem Falle wird der Validierungsmechanismus von Spring Web Flow verwendet. (Abschnitt 5.10 Validating a model in [SWF]).

Die JSF-Validatoren oder JSF-Konverter sollten für die Validierung aus den im Abschnitt 6.2.4.4 genannten Gründen nicht verwendet werden.

Sind im Datenmodell Datentyp wie Datum, Zeit oder Zeitpunkt enthalten, und diese auch durch den Benutzer der GUI frei eingebbar, so ist es am einfachsten, im Model ein String-Feld zu verwenden und die Konvertierung und Validierung im selbst programmierten Spring Web Flow-Validator durchzuführen.

6.2.4.8 Darstellung von Fehlern

Das folgende JSF-Tag `message` kommt zum Einsatz, um einen Fehler für ein bestimmtes Feld anzuzeigen:

```
<h:message for="isbn" showDetail="false" errorClass="error"/>
```

Abbildung 24: Beispiel aus Facelet

Das obige JSF Tag markiert das Feld, das die JSF-ID „isbn“ hat, als fehlerhaft, wenn im JSF-Context eine Fehlernachricht für die JSF-ID „isbn“ geschrieben wurde.

Für die Darstellung aller Fehlermeldungen kommt das JSF Tag `messages` zum Einsatz. Hierdurch werden alle Fehler, unabhängig von ihren JSF-IDs, in einer Liste dargestellt:

```
<h:messages/>
```

Abbildung 25: Darstellung von Fehlermeldungen

Die Darstellung von Fehlern und Validierungsnachrichten wird auch im Styleguide [Styleguide] beschrieben.

6.2.4.9 Verwendung von JSF Widgets

Für die Arbeit mit JSF werden einige Komponenten/Widgets bereits vorab zur Verfügung gestellt. Die Widgets sind alle als JSF Composite Components realisiert. Dadurch ist eine einfachere Wartung möglich, da die Komponenten vollständig in XHTML definiert sind und ein Grundverständnis von JSF genügt, um Anpassungen vorzunehmen. Spezielle Renderer oder Java-Klassen werden nicht benötigt. Details hierzu sind im Styleguide [Stylguide] zu finden.

Für die Verwendung der Tags muss in den XHTMLs folgender Namespace eingebunden werden:

```
xmlns:isy="http://java.sun.com/jsf/composite/isyfact"
```

6.2.4.10 Einsatz von Action Listenern

Action Listener können dazu verwendet werden, um auf das Klicken eines Buttons oder Links innerhalb einer Seite zu reagieren.

Auf Grund einer Eigenart von JSF in Zusammenhang mit dem Partial-State-Saving muss unbedingt darauf geachtet werden, dass die Komponente (Button/Link), an die der Action Listener gebunden ist und nicht durch den Klick ausgeblendet wird. Andernfalls führt dies zu Problemen mit dem Loadbalancing. Hintergrund ist, dass JSF durch das Partial-State-Saving den Maskenzustand teilweise in der Serversession ablegt. Werden die Anfragen an die Webanwendung durch den Loadbalancer an verschiedene Server verteilt, kann dies daher dazu führen, dass JSF eine Exception wirft, weil der Action Listener der ausgeblendeten Komponente nicht gefunden werden konnte.

6.2.4.11 Parameter mit Button/Link übergeben

JSF bietet mehrere Möglichkeiten, einen Parameter in Abhängigkeit eines geklickten Buttons oder Links an die Webanwendung zu übergeben. Dies ist beispielsweise dann notwendig, wenn auf einer Maske mehrere Elemente angezeigt werden, zu denen jeweils ein eigener „bearbeiten“-Button existiert. In diesem Fall muss es möglich sein zu erkennen, welcher Button zu welchem Element geklickt worden ist.

Die hierfür in JSF 2.x vorgesehenen Lösung mit `f:param` erfordert den Einsatz von JavaScript und kann daher Probleme in der Abwärtskompatibilität hervorbringen (z.B. wenn kein JavaScript aktiviert ist). Die Umsetzung sollte daher in der Regel mit einem Action Listener stattfinden (siehe Abbildung 26):

```
<h:commandLink id="bearbeite_SV_#{sachverhalt.id}" value="#{msg.MEL_Bearbeiten}"
actionListener="#{listener.waehleSachverhalt}">
    <f:attribute name="sachverhaltId" value="#{sachverhalt.id}" />
    ...
</h:commandLink>
```

Abbildung 26: Verwendung eines Action Listeners (View)

```
FacesContext.getCurrentInstance().getExternalContext()
.getRequestParameterMap().get("sachverhaltId");
```

Abbildung 27: Auswertung von Request Attributen

Dabei kann in der Methode `waehleSachverhalt`, der Wert des Attributs aus der `RequestParameterMap` des `FacesContextes` gelesen werden (siehe Abbildung 27). Beim Einsatz des Action Listeners muss darauf geachtet werden, dass die Komponente (Button/Link), an die der Action Listener gebunden ist, nicht durch den Klick ausgeblendet wird (vgl. Abschnitt 6.2.4.10).

Als Alternative zum Einsatz eines Action Listeners, kann die ID des Buttons/Links parametrisiert und im Controller ausgewertet werden. Die Parametrisierung der ID wird ebenfalls in Abbildung 26 dargestellt. Die Auswertung ist in diesem Fall aufwendiger, da alle Attribute der `RequestParameterMap` durchlaufen werden müssen, bis ein Parameter gefunden wurde, dessen ID mit „bearbeite_SV_“ beginnt. Vorteil der Lösung ist jedoch, dass Probleme mit dem Einsatz von Action Listenern damit umgangen werden.

6.3. Allgemeines

6.3.1 Festlegung der Startseite

Für den initialen Zugriff auf die Applikation wird in der `web.xml` der Startpunkt für den Dialogablauf definiert. Dieses geschieht durch den Eintrag in der `<welcome-file-list>` auf ein Index File, in welchem der Redirect auf den Web Flow steht. Dieses ist notwendig, weil eine Angabe der Flow Engine im `web.xml` für Welcome Files nicht möglich ist.

```
<welcome-file-list>
      <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

Abbildung 28: Ausschnitt web.xml

Der Auszug aus der Datei `index.html` sieht folgendermaßen aus:

```
<html>
    <head>
        <meta http-equiv="Refresh"
              content="0; URL= app/startFlow">
```

```
    </head>
</html>
```

Abbildung 29: Ausschnitt index.html

6.3.2 Der allgemeine Seitenrahmen

Der Aufbau und die Einbindung des allgemeinen Seitenrahmens ist im Styleguide [Styleguide] beschrieben.

6.3.3 Behandlung von Fehlern

In diesem Kapitel wird die Behandlung von Fehlern beschrieben. Dabei sind folgende Arten von Fehlern zu unterscheiden:

- Validierungsfehler
- Exceptions innerhalb des Dialogablaufs (AWK)
- Exceptions innerhalb des Dialogablaufs (GUI)
- Exceptions außerhalb des Dialogablaufs
- Exceptions innerhalb des Clients

6.3.3.1 Umgang mit Validierungsfehlern

Um Validierungsfehler innerhalb der Masken bei den fehlerhaften Feldern darzustellen, besitzt jede Formularkomponente ein „reference“ Attribut (siehe [Styleguide]). Dadurch können Validierungsfehler den entsprechenden Feldern zugeordnet werden.

Die eigentliche Validierung der Eingabedaten sollte im Anwendungskern erfolgen (z.B. durch Nutzung der plis-validation).

In bestimmten Fällen kann es auch notwendig sein zusätzliche Validierungsprüfungen in der GUI Schicht bereitzustellen (z.B. wenn die GUI je nach Eingabe unterschiedliche AWK-Aufrufe durchführt). Diese sollten durch einen evaluate-Aufruf über Webflow vor dem eigentlichen Zustandsübergang durchgeführt werden. Die Nutzung des Spring Webflow Validierungsmechanismus ist grundsätzlich möglich, bringt jedoch auch Nachteile mit sich, da der Aufruf der Validierung auf einer Namenskonvention von Viewstate und Transition beruht und bei Namensanpassungen sehr fehleranfällig ist. Durch den Aufruf der Validierung mit evaluate, wird die Validierung daher explizit und sichtbar definiert.

Bei der Übermittlung von Werten in JSF müssen die Vorgaben aus 6.2.4.4 beachtet werden: Die Eingabe von ungültigen Werten (z.B. ungültiges Datum, Buchstaben in einem Zahlenfeld) muss grundsätzlich möglich sein. Ggf. müssen entsprechende Datentypen und JSF-Converters erzeugt werden. Die Validierung (im AWK oder in der GUI) sollte die Fehleingabe dann feststellen. Die Validierung über JSF ist derzeit nicht vorgesehen (keine Unterstützung durch den Styleguide).

6.3.3.2 Behandlung von Exceptions innerhalb der Verarbeitung im AWK

Innerhalb der Verarbeitung von Dialogaktionen können Exceptions auftreten. Sofern diese nicht ohnehin behandelt werden, müssen sie innerhalb der GUI-Schicht behandelt werden. Der jeweilige Controller muss dabei durch eine Exception-Fassade (try/catch) sicherstellen, dass auftretende Fehler nicht in die Dialogsteuerung weitergegeben werden. Der Styleguide [Styleguide] bietet dazu Hilfsmethoden zum Loggen und zur Anzeige von Fehlern an. Das dort implementierte Exception Handling unterscheidet fachliche und technische Exceptions. Während fachlicher Fehler mit einer möglichst aussagekräftigen Fehlermeldung in der Oberfläche angezeigt werden sollen, soll für technische Fehler nur eine allgemeine Fehlermeldung angezeigt werden.

Ausgabe für fachliche Exceptions
im Error-LOG und der GUI: **Fehler-ID + Fehlernachricht + UUID**

Weil die Fehlertexte der fachlichen Fehler in der GUI angezeigt werden, ist die Verwendung von spezialisierten Exceptions anzuraten, die Fehlertexte enthalten, die für die Anzeige in der GUI geeignet sind. Hierzu finden sich weitere Informationen im Dokument [FehlerbehandlungKonzept].

Ausgabe für technische Exceptions
im Error-LOG: **Fehler-ID + Fehlernachricht + UUID**

in der GUI: **„Es ist ein technischer Fehler aufgetreten (Fehler-ID). Bitte versuchen Sie es später noch einmal (Referenzcode: UUID).**

In der Oberfläche wird also für technische Fehler immer ein fester Standardtext, zusammen mit dem „echten“ Fehlercode ausgegeben. Nur das Error-Log enthält die „echte“ Fehlermeldung. Dadurch wird verhindert, dass interne oder nur für die Systementwicklung oder den Betrieb relevante Meldungen nach außen getragen werden.

6.3.3.3 Behandlung von Exceptions innerhalb des Dialogablaufs

Exceptions, welche während der Ausführung von Logik der GUI-Schicht (z.B. Datenaufbereitung, Rendering) entstehen, werden immer als technische Ausnahmefehler angesehen. Für diese wird eine standardisierte Fehlerseite mit der in Punkt 6.3.3.3 angegebenen Fehlermeldung für technische Fehler ausgegeben.

Um die entsprechende Behandlung solcher Fehler zu ermöglichen, müssen folgende Schritte durchgeführt werden

- Für alle Flows muss ein globaler Übergang auf die Fehlerseite eingerichtet werden (`<transition on-exception="java.lang.Exception" to="fehler" />`)
- In der `faces-config.xml` muss der JSF Exception Handler konfiguriert werden:
  `<factory>`

```
            <exception-handler-
            factory>de.bund.bva.isyfact.common.web.exce
            ption.web.JsfExceptionHandlerFactory</excep
            tion-handler-factory>
        </factory>
```

- Die Fehlerseite (errorFlow) muss zur Verfügung gestellt werden.

Bei Nutzung der PLIS-Web entfallen diese Schritte, da dies automatisch passiert.

6.3.3.4 Behandlung von Exceptions außerhalb des Dialogablaufs

Es gibt Exceptions, die außerhalb des “normalen“ Dialogablaufs auftreten. Beispiele dazu sind fehlende Flow-Definitionen, abgelaufene Sessions und Autorisierungsfehler. Solche Fehler werden im Rahmen der HTTP-Request-Bearbeitung durch einen `ExceptionResolver` behandelt. Hierfür stellt Spring Web Flow einen Handler zur Verfügung, über welchen ein Mapping der Fehler auf spezielle Fehlerseiten möglich ist.

Es gibt keine anwendungsspezifischen Fehlerseiten. Alle IsyFact-Anwendungen verwenden eine einheitliche Fehlerseite.

Die Fehlerseite gibt aus Sicherheitsgründen keine Informationen über die Art des Fehlers preis. Die eigentliche Fehlerursache kann nur aus den Server-Logs ermittelt werden.

Die Fehlerbehandlung wird in der Spring Konfiguration konfiguriert:

```
<!-- Exception handlers for web flow -->
<bean id="simpleMappingExceptionResolver"
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
   <property name="defaultErrorView" value="errors/error" />
   <property name="exceptionMappings">
      <props>
         <!-- Thrown when no flow definition was found during a
               lookup operation by a flow locator. -->
         <prop key="org.springframework.Web Flow.execution.repository.
NoSuchFlowExecutionException">errors/flowException</prop>

         <!-- This might occur if the conversation ended, expired,
               or was otherwise invalidated, but a client view still
               references it. -->
         <prop key="org.springframework.Web Flow.conversation.
NoSuchConversationException">errors/convoException</prop>
      </props>
   </property>
</bean>
```

Abbildung 30: Ausschnitt aus config.xml

6.3.3.5 Behandlung von Exceptions innerhalb des Clients

Durch die Nutzung von AJAX entstehen clientseitige HTTP-Requests, welche durch die JSF-JavaScript Bibliothek gesteuert werden. Auftretende Fehler (z.B. Server nicht erreichbar, Serverfehler) werden daher nicht durch den Browser direkt, sondern durch die JS Bibliothek behandelt.

6.4. Sicherheit

Die Absicherung von Masken erfolgt auf Dialogablauf-Ebene. Die Berechtigungsprüfung verwendet die Sicherheits-Komponente.

Gekapselt werden die Bestandteile der Sicherheits-Komponente hinter Spring Security. Durch den Einsatz von Spring Security ist es einfacher eine Integration der Berechtigungen in die Spring Umgebung, insbesondere Spring Web Flow, zu erreichen.

6.4.1 Autorisierung

Für die Absicherung von Dialogabläufen wird innerhalb der Spring Web Flow Konfiguration auf dem `<view-state>` ein `<secured>` Tag mit dem geforderten Recht gesetzt. Die verwendeten Rechte müssen durch die Rollenrechte-Konfiguration (siehe [SicherheitNutzerdok]) für den eingeloggten Benutzer entsprechend gesetzt werden. Durch die Verwendung des Tags wird vor dem Anzeigen des `<view-state>` eine Überprüfung der Berechtigung des anfragenden Benutzers durchgeführt. Fehlt dem Benutzer diese, wird die Fehlerseite angezeigt.

```
<view-state id="erstellenViewState1" model="erstellenModel">
   <secured attributes="Erstellen"/>
   <transition on="continue" to="validiereStammdaten"/>
   <transition on="add" to="erstellenViewState1">
   ...
</view-state>
```

Abbildung 31: Ausschnitt aus Flow erstellenFlow.xml

```
<view-state id="verwaltenViewState">
   <secured attributes="Verwalten"/>
   <transition on="abschliessen" o="abschliessenViewState"/>
   <transition on="loeschen" to="loeschenViewState" />
   ...
</view-state>
```

Abbildung 32: Ausschnitt aus Flow verwaltenFlow.xml

6.4.2 Berechtigungsabhängige Darstellung in Masken

Um eine berechtigungsabhängige Darstellung von Maskenelementen zu realisieren, muss die Anwendung einen Controller bereitstellen, mit dessen Hilfe über nicht parametrisierte Methoden Anfragen nach den benötigten Rechten gestellt werden können. Dieser Controller ist dafür zuständig, den Rechte-Schlüssel aufzulösen und die Anfrage an die Komponente Sicherheit zu delegieren. Beantwortet werden die Anfragen stets mit einem einfachen Booleschen Wert.

Mit Hilfe dieses Controllers lassen sich einzelne Maskenelemente, aber auch Gruppen von Maskenelementen berechtigungsabhängig ausblenden. Die JSF-Komponenten bietet hierfür das `rendered`-Attribut (siehe Abbildung 33). Beantwortet der `BerechtigungsController` den Methoden-Aufruf von `getBenutzerDarfAdministrieren()` mit `false`, so wird der Button nicht dargestellt.

```
<t:commandButton […]
rendered="#{berechtigungsController.benutzerDarfAdministrieren}"/>
```

Abbildung 33: JSF-Komponente berechtigungsabhängig ausblenden

Sollen mehrere oder nicht-JSF Komponenten berechtigungsabhängig ausgeblendet werden, kann das Fragment-Tag der Facelets-Bibliothek

verwendet werden. Dieses kann eine beliebige Anzahl an weiteren Maskenelementen umschließen und bietet ebenfalls das `rendered`-Attribut, welches sich dann auf alle enthaltenen Komponenten auswirkt (siehe Abbildung 34).

```
<ui:fragment
rendered="#{berechtigungsController.benutzerDarfAnwenderAdministrieren}">
        […] weitere Komponenten […]
</ui:fragment>
```

Abbildung 34: GUI-Komponenten berechtigungsabhängig ausblenden

6.4.3 Vermeidung von Sicherheitslücken bei aktiviertem JavaScript

Ist JavaScript in einem Browser aktiviert, eröffnet dies gewisse Risiken bei der Verarbeitung datenschutzbedenklicher Informationen. Folgende Maßnahmen reduzieren jedoch das Risiko möglicher Attacken für Cross-site-Scripting (XSS):

- **Verwendung von Standardbrowsern**: Die gängigen Browser befolgen festgelegte Sicherheitsrichtlinien, die nur schwer und vorsätzlich deaktiviert werden können. Diese Standardeinstellungen erschweren XSS und sind gerade für den folgenden Punkt unerlässlich.

- **Übertragung aller Inhalte per HTTPS zum Browser**: Werden Webinhalte per HTTPS zum Client übertragen, ist ein unerwünschtes Datenauslesen per JavaScript-Injection oder IFrame-Injection verhindert, da Browser JavaScript-Code nur dann auf einen domain-fremden DOM zugreifen lassen, sofern dieser nicht sicher übertragen wurde.

- **Keine Verwendung von Request-Variablen in offenem JS**: Werden Request-Parameter, z.B. als Teile eines Formulars, direkt in offenem JavaScript (`eval([var])` oder `setTimeout([var])`) weiterverwendet, so können Angreifer manipulierte Parameter für DOM-based-XSS nutzen, d.h. es werden JavaScript Befehle als Parameter übergeben, die Inhalte verändern, auslesen oder in einen falschen Kontext setzen.

- **Encodierung von Request-Variablen im DOM**: Werden Request-Variablen auf einer Seite dargestellt, so sind diese XML-encodiert einzubinden (siehe Element `outputText` in Kapitel 6.2.4.4). Somit wird verhindert, dass ein Angreifer ein ungewünschtes Script-Tag übergibt.

6.5. Druck von Masken

Für den Druck von Masken wird ein eigenes Stylesheet verwendet. In diesem Stylesheet kann die Anzeige für den Druck optimiert aufbereitet werden, z.B. in dem das Menü ausgeblendet wird.

Für die Druckvorschau einer Seite sollten die benutzten Widgets so aufbereitet werden, dass alle Inhalte auf der Seite angezeigt werden (z.B. bei Tabs alle Tabs anzeigen).

Der Styleguide [Styleguide] stellt eine entsprechende Druckansicht zur Verfügung.

6.6. Temporäre Binärdaten

Für manche Anwendungsfälle in Webanwendungen wird ein Speicherort für temporäre Binärdaten benötigt. Dies ist z.B. notwendig, um dem Nutzer die Binärdaten wieder zur Verfügung zu stellen, bevor diese in einem Datenbestand gespeichert wurden (z.B. hochgeladenes Lichtbild anzeigen).

Für (temporäre) Binärdaten kann der IsyFact-Binärdatenservice genutzt werden. Dieser bietet die Möglichkeit Binärdaten entgegenzunehmen und einen Cleanup-Timertask, der veraltete Binärdaten automatisch wieder aufräumt. Somit ist keine Verwendung von Datenbankmitteln notwendig.

7. Konfiguration

Die allgemeinen Konfigurationen enthalten die anwendungsunabhängig benötigten Einstellungen um eine Web-GUI nach diesem GUI-Konzept einzurichten.

7.1. Übersicht Konfigurationsdateien

In diesem Kapitel soll kurz ein Überblick gegeben werden, an welchen Stellen und in welchen Dateien Konfigurationen vorgenommen werden. Konkrete Beispiele finden sich im der Vorlage-Anwendung und den nachfolgenden Beispielen. Im letzten Abschnitt wird aufgezeigt, welche Konfigurationen durchgeführt werden müssen, sofern die PLIS-Web vollständig verwendet wird.

7.1.1 web.xml

Die Datei `web.xml` ist der Web Deployment Deskriptor. Dieser beschreibt die Teile der Web Applikation, welche für die Applikation die Schnittstellen nach außen darstellen. Die `web.xml` enthält die notwendigen Konfigurationen, um die Teilsysteme, (JSF, Spring Web Flow, Security) zu aktivieren. Die dazu notwendigen Einträge finden sich in den zugehörigen Abschnitten.

7.1.2 application.xml

In der Datei `application.xml` sind alle zentralen Spring Konfigurationen enthalten. Hier wird das Spring Basissystem mit seinen Konfigurationen festgelegt. Für die Persistierung von Spring Beans kommt JPA mit Hibernate zum Einsatz. [Spring] [JPA] [DatenzugriffDetailkonzept]

7.1.3 webflow.xml

Die Datei `webflow.xml` enthält die Spring Konfigurationen für die GUI-Frameworks (Webflow, JSF, Spring MVC).

7.1.4 isy-sicherheit-web.xml

Die Datei `isy-sicherheit-web.xml` beinhaltet die Konfiguration für die Spring Security Definition. Hier wird konfiguriert, wie die Sicherheitskomponente an Spring Security angeschlossen wird.

7.1.5 faces-config.xml

In der Datei `faces-config.xml` sind die notwendigen Konfigurationen für Java Server Faces enthalten

7.2. Basis-Konfiguration JSF

Dieses Kapitel fasst alle notwendigen Basiskonfigurationen für die Verwendung von JSF zusammen.

7.2.1 Verwendung von Facelets (webflow.xml)

Für die Verwendung von Facelets ist es notwendig, in der Spring Konfiguration ein Mapping zwischen der Datei-Erweiterung und den Facelets herzustellen.

```
<!-- Maps logical view names to Facelet templates (e.g. 'search' to '/WEB-
INF/search.xhtml' -->
<bean id="faceletsViewResolver"
        class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass"
                value="org.springframework.faces.mvc.JsfView" />
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".xhtml" />
</bean>
```

Abbildung 35: Ausschnitt config.xml

7.2.2 Abschalten der Ausgabe von Kommentaren

Die Ausgabe von HTML-Kommentaren, die in der Flow-Definition und im View zur Dokumentation der Software eingebettet wurden, wird abgeschaltet.

7.2.3 Konfiguration von Konvertern

Um den gewünschten Konverter für die Verwendung bekannt zu machen, muss dieser in der `faces-config.xml` erfasst werden.

```
<converter>
    <converter-for-class>java.util.Date</converter-for-class>
    <converter-class>de.msg.terminfindung.gui.util.DateConverter</converter-
class>
</converter>
```

Abbildung 36: Ausschnitt aus faces-config.xml

7.2.4 Aktivieren von „Partial State Saving“

Das mit JSF 2.0 eingeführte Feature „Partial State Saving“ muss aktiviert bleiben. Hintergrund ist, dass JSF 2.0 ansonsten im Zusammenhang mit der Replikation der Session die IDs für JSF-Komponenten doppelt vergibt und es dadurch zu Fehlern in der Anwendung kommt.

7.2.5 Erkennung von JavaScript Unterstützung

Je nach Browser und JavaScript Aktivierungsstatus muss die Anwendung Widgets verschieden darstellen. Hierzu existiert ein Filter, welcher in die web.xml eingebunden wird. Die Parameter müssen ggf. angepasst werden.

```
<!-- Filter zur Initialisierung der Applikation (JavaScript De-/Aktiviert,
...). -->
    <filter>
        <filter-name>applicationInitialisierungFilter</filter-name>
        <filter-
class>de.bund.bva.isyfact.common.web.servlet.filter.ApplicationInitialisierungF
ilter</filter-class>

        <!-- Optionaler Parameter: Der Parameter "urlsToSkip" dient zur
Aufnahme von Url-Pfaden, relativ zum ApplicationContext-Pfad,
            die von der Filterung ausgenommen werden. Mehrere Url-Pfade sind
kommasepatiert anzugeben. Es ist pro
```

```
            Url ein fuehrendes
            "/" anzugeben. -->
        <init-param>
            <param-name>urlsToSkip</param-name>
            <param-value>/app/resources</param-value>
        </init-param>

        <!-- Plicht-Parameter: Der Parameter "urlApplicationInitialisierung"
enthaelt die Url zur Application-Initialisierungsseite.
            Es ist ein fuehrendes "/" anzugeben. -->
        <init-param>
            <param-name>urlApplicationInitialisierung</param-name>
            <param-
value>/app/common/init/applicationInitialisierung.xhtml</param-value>
        </init-param>
    </filter>
```

Abbildung 37: Ausschnitt web.xml

7.3. Basis-Konfiguration Web Flow

Dieser Abschnitt beschreibt die Basiskonfiguration für die Verwendung von Spring Web Flow.

7.3.1 Erweiterung für Servlets im web.xml

Damit die Funktionalitäten von Spring Web Flow aufgerufen werden, muss das SWF Dispatcher Servlet registriert werden, welches die weitere Behandlung an den SWF Kern abgibt.

```
<servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/config.xml</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/app/*</url-pattern>
</servlet-mapping>
```

Abbildung 38: Ausschnitt web.xml

7.4. Konfiguration der Navigation

Nachfolgend werden die notwendigen Konfigurationen beschrieben, um die konkreten Dialog Flows zu definieren, und wo die entsprechenden Anpassungen vorgenommen werden müssen.

7.4.1 JSF Flow Builder Service

Der JSF Flow Builder Service initialisiert aus der Dialog-Ablaufbeschreibung die konkreten Dialoge. Zusätzlich kann hier noch der Expression Parser für die EL-Implementierung definiert werden. Durch die Definition des EL-Parsers kann die konkrete EL-Implementierung festgelegt werden. Standardmäßig sollte hier keine spezifische Konfiguration vorgenommen werden.

```
<!-- Configures the Spring Web Flow JSF integration -->
<faces:flow-builder-services id="facesFlowBuilderServices" expression-
parser="expressionParser"/>
```

Abbildung 39: Ausschnitt aus Web Flow.xml

7.4.2 Ablage der Konfiguration

Innerhalb der Spring-Konfiguration wird für den Web Flow die „`flow-registry`“ konfiguriert. Hier wird die Ablage der Konfigurations-Files definiert.

```
<!-- The registry of executable flow definitions -->
<webflow:flow-registry id="flowRegistry"
        flow-builder-services="facesFlowBuilderServices">
        <webflow:flow-location-pattern value="/WEB-INF/gui/flows/**/*Flow.xml"
/>
</webflow:flow-registry>
```

Abbildung 40: Ausschnitt aus Web Flow.xml

7.4.3 Fehlerbehandlung innerhalb der Verarbeitung der GUI

Für die Konfiguration der in Abschnitt 6.3.3.2 erwähnten Fehlerbehandlung müssen die entsprechenden Schritte aus dem Kapitel durchgeführt werden.

7.5. Konfiguration Logging

Für das Logging kommt Log4j zum Einsatz. Die notwendigen Konfigurationen und weitere Details sind in [LoggingKonzept] beschrieben.

Benötigte Libraries: `de.bund.bva.pliscommon.`**`isy-logging`**

Die Ablage der Log4J Konfiguration erfolgt in der Datei `web.xml`.

```
    <!-- LOG4J Konfiguration
        Angabe des Speicherorts der log4j Konfiguration
        Wenn nicht angegeben, greift die Standardinitialisierung:
        Konfiguration im Classpath.
        Aufgrund des Deployoments liegt die Konfiguration aber unter
          /classes/config/log4j.properties
        Details siehe http://static.springframework.org/spring/docs
          /2.5.x/api/org/springframework/web/util/Log4jWebConfigurer.html
    -->
    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>classpath:/config/log4j.properties</param-value>
    </context-param>
```

Abbildung 41: Logging Konfiguration (Ausschnitt web.xml)

Um Log-Einträge zu schreiben, bedient man sich im Sourcecode der bereitgestellten Methoden der Apache Log4J Implementierung.

7.6. Konfiguration Security

7.6.1 Setup web.xml

Für den Einsatz von Spring Security ist ein Servlet Filter notwendig, in welchem die initialen Abhandlungen der Berechtigungen erfolgen.

Hier wird die Ablage der Spring Security Konfiguration definiert.

```
<!-- Spring Security filter, context parameter -->
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:resources/spring/applicationContext-security.xml
        </param-value>
</context-param>
```

Abbildung 42: Ausschnitt aus web.xml

Der Security Filter und die davon betroffenen URL-Patterns werden definiert. Im Falle eines Einsatzes zur Absicherung der Dialogabläufe sollten alle Zugriffe abgesichert werden.

```
        <!-- Spring Security filter -->
        <filter>
                <filter-name>springSecurityFilterChain</filter-name>
                <filter-class>
                        org.springframework.web.filter.DelegatingFilterProxy
                </filter-class>
        </filter>

        <filter-mapping>
                <filter-name>springSecurityFilterChain</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
```

Abbildung 43: Ausschnitt aus web.xml

Der Security Filter benötigt einen `ContextLoaderListener`, welcher wie folgt konfiguriert ist.

```
        <!-- Context Listener for the spring filter -->
        <listener>
                <listener-class>
                        org.springframework.web.context.ContextLoaderListener
                </listener-class>
        </listener>
```

Abbildung 44: Ausschnitt aus web.xml

7.6.2 Spring Security Konfiguration

Die nachfolgende Spring Konfiguration erfolgt in der Datei `isy-sicherheit-web.xml`, welche durch den Security Filter initial geladen wird.

Für die Konfiguration finden die nachfolgenden Namespaces Verwendung.

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-2.0.1.xsd">
```

Abbildung 45: Ausschnitt aus isy-sicherheit-web.xml

Die Definition der HTTP-Konfiguration erfolgt in dem XML-Tag `<http>`.

```
    <http auto-config="true">
        <form-login login-page="/login.jsp" />
    </http>
```

Abbildung 46: Ausschnitt aus applicationContext-security.xml

Durch Spring Security wird die Verwendung von Annotationen bei der Berechtigungsprüfung ermöglicht. Diese wird wie folgt aktiviert.

```
    <global-method-security secured-annotations="enabled" />
```

Abbildung 47: Ausschnitt aus applicationContext-security.xml

Die Konfiguration von Spring Security setzt die Definition eines User Services voraus. Dieses kann nach aktuellem Stand nicht umgangen werden. Daher wird ein Dummy Service definiert, welcher einen inaktiven Benutzer enthält. Die Abfrage auf die realen Benutzer erfolgt im Authentication Provider, welcher im folgenden Kapitel beschrieben ist.

```
<user-service>
   <user authorities="ROLE" name="name" password="password"
     disabled="true" />
</user-service>
```

Abbildung 48: Ausschnitt aus isy-sicherheit-web.xml

Für die Anbindung des Berechtigungsmanagers an Spring Security findet die Möglichkeit zur Erstellung eines “Custom Authentication Providers” Anwendung. In diesem Authentication Provider wird bei Bedarf überprüft ob der Nutzer die notwendigen Berechtigungen besitzt, bzw. es wird für die weitere Verarbeitung in Spring Security der notwendige Kontext aufgebaut.

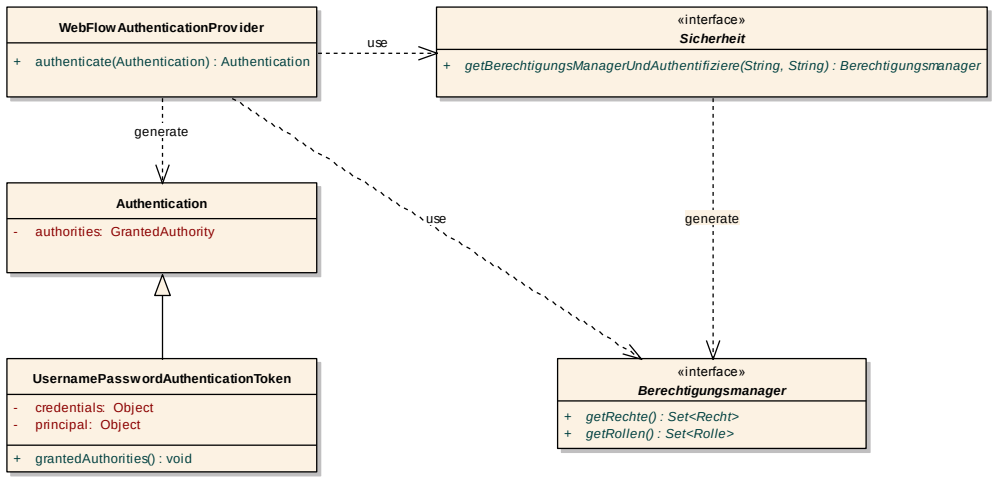

Abbildung 49: Klassendiagramm Anbindung Sicherheitskomponente

Das Sequenzdiagramm stellt den Zugriff des Authentication Providers auf die Sicherheitskomponente dar. Hierbei werden die Rollen eines Benutzers über den Berechtigungsmanager gelesen und in einen Spring Security konformen Token geschrieben. Dieser Token findet dann bei der Autorisierung einzelner Benutzerinteraktion durch Spring Security Verwendung.

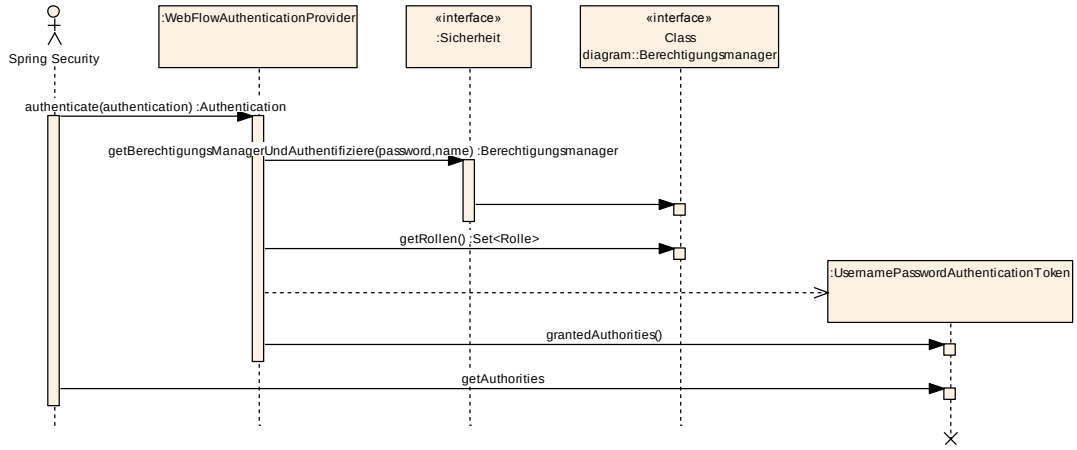

Abbildung 50: Sequenzdiagramm Zugriff auf Sicherheitskomponente

7.6.2.1.1 Konfiguration Sicherheitskomponente

Die Konfiguration der Sicherheitskomponente über eine properties-Datei ist im Konzept [SicherheitNutzerdok] beschrieben.

7.6.2.1.2 Konfiguration WebFlowAuthenticationProvider

Der `WebFlowAuthenticationProvider` erhält durch die Konfiguration die Instanz der Bean „`sicherheit`“ injected und wird als „`custom-authentication-provider`“ Spring Security bekannt gemacht.

```
<!-- ========================================================================
custom authentication provider setzen um Sicherheits Komponente für Rollen
Ermittlung zu verwenden
    ======================================================================== -->
<beans:bean id="Web FlowAuthenticationProvider"
   class="de.bund.bva.pliscommon.plisweb.Web Flow.security.Web
FlowAuthenticationProvider">
   <custom-authentication-provider />
   <beans:property name="sicherheit" ref="sicherheit" />
</beans:bean>
```

Abbildung 51: Ausschnitt aus applicationContext-security.xml

7.6.3 Setup Spring Config (webflow.xml)

Hier muss ein Listener registriert werden, über welchen die Security Bedürfnisse geprüft werden.

```
<bean name="flowExecutionSecurityListener"
   class="org.springframework.Web Flow.security.SecurityFlowExecutionListener">
</bean>

<bean name="flowExecutionListenerLoaderFactory"
   class="org.springframework.Web
Flow.config.FlowExecutionListenerLoaderFactoryBean">
   <property name="listeners">
      <map>
         <entry key-ref="flowExecutionSecurityListener"
                value="cd-register"></entry>
         <entry key-ref="jpaFlowExecutionListener"
                value="cd-register"></entry>
         <entry key-ref="flowExecutionBreadcrumbListener"
                value="cd-register"></entry>
      </map>
   </property>
</bean>

<bean name="flowExecutionFactory"
   class="org.springframework.Web Flow.engine.impl.FlowExecutionImplFactory">
   <property name="executionKeyFactory"
      ref="flowExecutionRepository"/>
```

```
    <property name="executionListenerLoader"
        ref="flowExecutionListenerLoaderFactory"/>
</bean>
```

Abbildung 52: Ausschnitt aus Web Flow.xml

7.7. Konfiguration einer Anwendung unter Nutzung der PLIS-Web

Bei Einsatz der PLIS-Web werden viele Konfigurationen bereits vorgenommen. Die Anwendung muss nur noch wenige Einstellungen vornehmen. Hinweise zur Einbindung befinden sich auch im Styleguide [Styleguide].

7.8. Konfiguration des Ressource Caching Mechanismus

Möchte man Webressourcen wie Bilder oder CSS-Dateien von Softwareversionen abhängig machen, muss das folgende gemacht werden:

7.8.1 JSF ResourceHandler einrichten

Die Ressourcenlieferung wird durch so genannte *ResourceHandler* verkapselt. Um also den Prozess anpassen zu können, muss man einen eigenen *ResourceHandler* definieren und an die Web Applikation anbinden. Da jedoch das GUI mittels JSF erstellt wird, muss es JSF-spezifisch umgesetzt werden. Der folgende Code ist ein Beispiel so eines Handlers:

```
public class VersionierungResourceHandler extends ResourceHandlerWrapper {


    private ResourceHandler wrapped;
    private SystemVersionBean systemVersionBean;

    public VersionierungResourceHandler(ResourceHandler wrapped) {
        this.wrapped = wrapped;
        if (StatischerKontextInhaber.getApplicationContext() != null) {
            this.systemVersionBean =
                StatischerKontextInhaber.getApplicationContext()
.getBean(SystemVersionBean.class);
        }
    }

    @Override
    public Resource createResource(String resourceName) {
        return createResource(resourceName, null, null);
    }

    @Override
    public Resource createResource(String resourceName, String libraryName) {
        return createResource(resourceName, libraryName, null);
    }

    @Override
    public Resource createResource(String resourceName, String libraryName,
String contentType) {
        final Resource resource = super.createResource(resourceName,
libraryName, contentType);
        if (resource == null) {
            return null;
        }

        return new ResourceWrapper() {

            @Override
            public String getRequestPath() {
                String requestPath = super.getRequestPath();
                String version = "v=" + getSystemVersion();
                if (!requestPath.contains("?")) {
                    return requestPath + "?" + version;
                }
                return requestPath + "&" + version;
            }
```

```
            @Override
            public Resource getWrapped() {
                return resource;
            }
        };
    }

    @Override
    public ResourceHandler getWrapped() {
        return this.wrapped;
    }

    private String getSystemVersion() {
        if (this.systemVersionBean == null &&
StatischerKontextInhaber.getApplicationContext() != null) {
            this.systemVersionBean =
                StatischerKontextInhaber.getApplicationContext()
.getBean(SystemVersionBean.class);
        }
        if (this.systemVersionBean == null) {
            this.systemVersionBean = new SystemVersionBean();
            this.systemVersionBean.setSystemVersion("DEV");
        }
        return this.systemVersionBean.getSystemVersion();
    }
}
```

In der Methode `getRequestPath()` kann man die Pfade der Ressourcen beliebig anpassen und zum Beispiel, wie oben gezeigt, eine Version an sie anhängen.

7.8.2 Konfiguration

Der *ResourceHandler* muss noch mittels Konfiguration an die Web Applikation angebunden werden. Dies macht man in `faces-config.xml` auf die folgende Weise:

```
    <application>
       <resource-handler>pfad.VersionierungResourceHandler<resource-handler/>
    </application>
```

7.8.3 JSF Tags benutzten

Es gibt noch eine letzte Bedingung die man erfüllen muss, damit der JSF *ResourceHandler* die Ressourcen überhaupt behandelt. Es müssen JSF Tags benutzt werden:

- JavaScript: `<h:outputScript>` anstatt `<script>`
- CSS: `<h:outputStylesheet>` anstatt `<link>`

Dank ihnen werden die jeweiligen Ressourcen für unseren *ResourceHandler* sichtbar und werden von ihn nun verwaltet.

8. Session Behandlung

Die nachfolgenden Kapitel beschäftigen sich mit der Behandlung der Session Informationen, welche in Spring Web Flow anfallen. Hierunter fallen alle Daten, die für die Dialogabläufe benötigt werden:

- Komponentenbaum der Dialogansicht, dieser beinhaltet die Dialogelemente und die Information über die Bindung an die Backing Beans.
- Den Flow Container, in welchem die Backing Beans während dem Dialogfluss vorgehalten werden.
- Die Conversation, welche eine Benutzer Interaktion beinhaltet, bündelt die beiden vorangegangenen Angaben.

Standardmäßig wird diese Information in der HTTP Session abgelegt und wieder hergestellt. Nach IsyFact-Zielarchitektur erfolgt diese Speicherung in der Datenbank. Diese Funktionalität übernimmt der Tomcat. Eine Anpassung der Anwendung ist nicht erforderlich.

8.1.1 Vergabe von Cookies / Session IDs

Für die Identifikation abgelegter Conversations von Spring Web Flow wird auf die Daten im Cookie zurückgegriffen. Der Cookie wird einmalig definiert und ist durch die Session ID vorgegeben.

Die Vergabe der Session IDs erfolgt über die Standardmechanismen innerhalb des Tomcat. Bei der Anforderung einer neuen Session innerhalb der Applikation, was für den Benutzer nicht sichtbar ist, wird die neue ID vergeben und an die Session gebunden. Die Nutzung von Cookies ist zwingend erforderlich, ansonsten können die Webanwendungen nicht genutzt werden.

8.1.2 Session Zugriff

Für die Arbeit mit Spring Web Flow ist es notwendig, die in der Session notwendigen Daten der Conversation für jeden Schritt bereitzustellen und nach jedem Schritt abzulegen. Hierfür werden die Daten in der Datenbank persistiert. Dies erfolgt über einen eigenen Session-Manager im Tomcat (siehe [Tomcat]). Es handelt sich dabei um eine Erweiterung der IsyFact. Wenn diese Erweiterung nicht eingesetzt wird, muss entweder die Session-Persistierung auf andere Weise durchgeführt werden, oder es muss sichergestellt werden, dass die Requests eines Benutzers immer auf die gleiche Tomcat-Instanz gehen (Sticky Sessions). Die Session-Daten müssen dabei möglichst klein gehalten werden, um die Performance der Anwendung nicht zu verschlechtern. Die Größe der Session wird maßgeblich durch die „im Webflow“ gespeicherten Model-Daten bestimmt. Daher muss darauf geachtet werden nur unbedingt notwendige Daten im Model zu halten.

8.1.3 Migration zu Tomcat-Session-Persistierung (plis-web-2.3.x)

Vor der plis-web-Version 2.3.0 erfolgte die Persistierung der Session-Daten in der Datenbank auf Ebene der Anwendung durch eine Erweiterung von Spring-Web-Flow. Dafür mussten Klassen eingebunden werden, die von plis-web zur Verfügung gestellt wurden. In plis-web-Version 2.3.0 sind diese Klassen als deprecated markiert, weil die Session-Persistierung nun transparent vom Tomcat durchgeführt wird (siehe [Tomcat]). Solange die Tomcat Erweiterung zur automatischen Persistierung der Session nicht eingesetzt wird, müssen die als deprecated markierten Klassen weiter verwendet werden. Die folgenden Schritte fassen die Unterschiede und Migrationsschritte von einer früheren plis-web-Version zu plis-web-2.3.0:

1. Die Anwendung muss innerhalb eines Tomcat laufen, der -gemäß dem Konzept [Tomcat] betrieben wird. Der JDBC-Sessionmanager muss wie dort beschrieben aktiviert und konfiguriert werden.

2. Die Definition der Bean `DbBasedConversationContainer-StoreService` muss von dem Spring-Kontext der Anwendung entfernt werden.

3. Die Bean, welche für das Aufräumen der Datenbank-Sessions zuständig ist `(jobExecutorStoreClean)` und die zugehörige Timer-Factory muss von dem Spring-Kontext der Anwendung entfernt werden, weil das Aufräumen der Datenbank-Sessions auch vom Tomcat verwaltet wird.

4. Die Bean DbBasedBindingConversationManager muss im Spring-Kontext der Anwendung mit der Spring-Web-Flow-Klasse `SessionBindingConversationManager` ersetzt werden.

5. Für Anwendungen, die Breadcrumbs unterstützen, kann die Bean `BreadcrumbAwareDbBasedFlowExecutionRepository` mit der neuen plis-web-Klasse `BreadcrumbAwareFlowExecution-Repository` ersetzt werden. Für Anwendungen ohne Breadcrumbs kann die Bean `DbBasedFlowExecution-Repository` mit der Spring-Web-Flow Klasse `DefaultFlow-ExecutionRepository` ersetzt werden.

6. Die Datenbank-Tabelle für die Sessions `(WEBFLOW-CONVERSATIONCONTAINER)` muss entfernt werden. Sie wird mit einer neuen Datenbank-Tabelle ersetzt, die im [Tomcat] beschrieben ist.

8.1.4 Transaktionssteuerung

Alle Zugriffe auf die Datenbank müssen in einer Transaktion verpackt werden. Die Transaktionssteuerung wird dabei vom AWK-Wrapper durch Annotation (@Transactional) der Klasse übernommen.

Aus Sicht der GUI-Schicht bedeutet dies, dass jeder Aufruf des AWKs unmittelbar eine Änderung der Daten zufolge hat. Um eine parallele

Aktualisierung eines Datensatzes zu verhindern, können die entsprechenden Versionsnummern der JPA-Entiäten verwendet werden, indem diese mit in die GUI-Entitäten geladen werden.

9. Checkliste zur QS

Die Checkliste für die QS stellt Prüfpunkte zur Verfügung, anhand derer wichtige Kriterien zur Umsetzung der Oberfläche überprüft werden können.

9.1. ID Vergabe JSF

Für automatisierte Tests ist es notwendig, für Eingabefelder und Controls eine konkrete und in der Seite eindeutige ID zu vergeben.

Wird für eine Komponente keine ID vorgegeben, so erzeugt JSF die IDs dynamisch. Ein Test über ein GUI-Testwerkzeug wird somit erschwert.

9.2. Verwendung der HTTP-Session

Aufgrund der Vorgabe eines zustandslosen Servers ist nicht sichergestellt, dass der Anwendung beim nächsten Zugriff auf einen Server die HTTP-Session zur Verfügung steht. Daher ist es notwendig die Masken und Abläufe so zu entwerfen, das hierauf verzichtet werden kann.

9.3. Nutzung Model Beans

Jeder Flow muss eine (oder mehrere) Model Bean hinterlegt haben, in welcher die Daten für die Maske vorgehalten werden.

9.4. Transaktionsbehandlung

Bei mehrschrittigen Datenerfassungen muss die Behandlung der Transaktion mit einem besonderen Augenmerk behandelt werden.

9.5. Einhaltung Styleguide

Die Applikation muss auf die Vorgaben des Styleguides überprüft werden. Der Styleguide befindet sich unter [Styleguide].

9.6. Flow Konfiguration

Für jede Maske sollte der Flow in einer eigenständigen Konfiguration hinterlegt sein, damit kann jede Maske separat ausgetauscht und gewartet werden. Jeder Flow sollte entsprechend dem Layout von dem vorgegebenen Parent-Flow erben.

9.7. Optimierung JSF Design

Aufgrund der automatischen Ablage eines JSF Komponenten-Baumes in der Conversation und damit in der Datenbank, ist es notwendig sich über die Größe des Komponenten-Baumes Gedanken zu machen.

Begrenzung der real verwendeten Komponenten auf die Ein und Ausgabe Elemente. Statische Texte bevorzugt nur im HTML verwenden.

9.8. Optimierung Snapshots

Für die Benutzung der Breadcrumb-Navigation und auch des Back-Buttons ist es notwendig, die Anzahl der Snapshots eines Dialogflusses zu erhöhen. Dieses sollte in Abhängigkeit der Anforderungen an die konkrete Anwendung erfolgen.

9.9. Festlegung der Texte für die Titel

Es sollte ein Standard-Titel und Präfix für die Anwendung angelegt werden. Für jede Seite sollte ein Breadcrumb angelegt werden.

9.10. Festlegung der Hilfeseiten

Für alle Masken sollte eine zugehörige Hilfe verfügbar sein, und der Aufruf auf die Seite muss überprüft werden.

9.11. Fehlerbehandlung

Alle Exceptions werden an einer definierten Stelle behandelt. Für technische Fehler werden keine Details, sondern eine generische Fehlermeldung angezeigt.

9.12. Flow und Masken Security

Der Zugriff auf Masken und Dialogelemente muss anhand der Systemspezifikation überprüft werden.

9.13. Regeln bei der JavaScript-Programmierung

Die unter 6.2.4.5 und 6.4.3 beschriebenen Regeln müssen überprüft werden.

9.14. Regeln der sicheren Softwareentwicklung

Bei der Entwicklung von Web-Anwendungen müssen in besonderem Maße Sicherheitsaspekte berücksichtigt werden. Die entsprechenden Regeln sind in [OWASP10] beschrieben.

10. Quellenverzeichnis

[CommonEL]
Common Expression Language
http://commons.apache.org/el/.

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

[Facelets]
Facelets
https://facelets.dev.java.net/.

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

[IsyFactJQuery]
Paketierte JQuery-Dateien für IsyFact-Anwendungen
60_Software\Bibliotheken\web-gui.

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

[IsyFactReferenzarchitekturITSystem]
IsyFact – Referenzarchitektur für IT-Systeme
00_Allgemein\IsyFact-Referenzarchitektur-IT-System.pdf.

[JPA]
Java Persistence API
http://java.sun.com/javaee/overview/faq/persistence.jsp.

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

[OWASP10]
OWASP Top 10 Project
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project .

[Programmierkonventionen]
Java Programmierkonventionen
40_Methodik\30_Implementierung\Java-Programmierkonventionen.pdf .

[Quasar]
J. Siedersleben, Moderne Softwarearchitektur - Umsichtig planen, robust bauen mit Quasar, d.punkt.verlag, 2004.

[Services]

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

[SicherheitNutzerdok]

Nutzerdokumentation Sicherheit
20_Bausteine\Sicherheitskomponente\Nutzerdokumentation_Sicherheit.pdf .

[Spring]

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

[Styleguide]

Styleguide
Muss projektspezifisch erstellt werden.

[SUNRI]

SUN Referenzimplementierung JSF
http://java.sun.com/javaee/javaserverfaces/.

[SWF]

Spring Web Flow Dokumentation
http://static.springsource.org/spring-webflow/docs/2.3.x/reference/html/ .

[Tomahawk]

Apache JSF Komponentenbibliothek
http://myfaces.apache.org/tomahawk/index.html.

[TomahawkDataScroller]

Die Dokumentation der Tomahawk "DataScroller" Komponente
http://myfaces.apache.org/tomahawk-project/tomahawk20/tagdoc/t_dataScroller.html .

[Tomcat]

Nutzungskonzept Apache Tomcat
30_Plattform\interne_Arbeitsunterlagen\Nutzungskonzept_Apache_Tomcat.docx .

[ÜberwachungKonfigKonzept]

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

[VorlageAnwendung]

Beispielimplementierung „Vorlage-Anwendung“
Wird auf Anfrage bereitgestellt.

[WikiJSF]

JavaServer Faces
http://de.wikipedia.org/wiki/JavaServer_Faces.

11. Abkürzungsverzeichnis

BVA	Bundesverwaltungsamt
SWF	Spring Web Flow

12. Abbildungsverzeichnis