Import und Export von „Process Templates“ als Baustein einer Prozessunterstützung auf Basis von aktiven Dokumenten

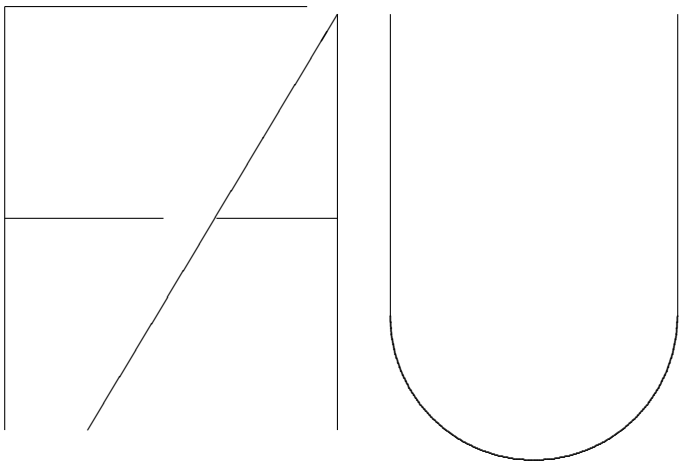
Bachelorarbeit

Patrick Reischl

Lehrstuhl für Informatik 6
(Datenmanagement)

Department Informatik
Technische Fakultät

Friedrich Alexander-
Universität
Erlangen-Nürnberg



Import und Export von „Process Templates“ als Baustein einer Prozessunterstützung auf Basis von aktiven Dokumenten

Bachelorarbeit im Fach Informatik

vorgelegt von

Patrick Reischl

geb. 01.05.1989 in Nürnberg

angefertigt am

Department Informatik
Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Univ.-Prof. Dr.-Ing. habil. Richard Lenz
Dipl.-Inf. Christoph P. Neumann

Beginn der Arbeit: 01.06.2011
Abgabe der Arbeit: 30.10.2011

Erklärung zur Selbständigkeit

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Informatik 6 (Datenmanagement), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Bachelorarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 30.10.2011

_____

(Patrick Reischl)

Kurzfassung

Import und Export von „Process Templates“ als Baustein einer Prozessunterstützung auf Basis von aktiven Dokumenten

Das Projekt α-Flow hat sich die Entwicklung von verteilten, elektronischen Fallakten, genannt α-Docs, zum Ziel gesetzt. α-Docs werden speziell für die Anwendung in institutionsübergreifenden Behandlungsabläufen entworfen. Die Speicherung und Bearbeitung der α-Docs erfolgt dezentral, innerhalb der IT-Infrastruktur der einzelnen Behandlungspartner, wobei die Daten über das Internet synchronisiert werden. Kommt ein neuer Teilnehmer hinzu, benötigt er lediglich eine lokale Kopie der verteilten Fallakte. Da medizinische Dokumente üblicherweise in Papierform vorliegen, wird ein dokumentenorientiertes Paradigma, in Form von „Aktiven Dokumenten“, verfolgt. „Aktive Dokumente“ enthalten neben ihren medizinischen Nutzdaten Metainformationen zum Behandlungsablauf und werden durch ihre aktive Eigenschaften definiert. In α-Flow stellt eine regelbasierte Prozesssteuerungslogik eine dieser Eigenschaften dar.

Bislang muss die Prozessplanung auch bei stets ähnlich ablaufenden Behandlungen für jeden Patienten von neuem betrieben werden. Dies sorgt für einen großen Aufwand bei der Erstellung eines neuen α-Docs und kann zu einer erhöhten Anzahl von Behandlungsfehlern durch falsche Planung führen. Um dem entgegenzutreten, stellt diese Arbeit eine Import-/Exportkomponente für sogenannte „Process Templates“ vor. „Process Templates“ modellieren strukturiert ablaufende Behandlungsprozesse zum Zwecke der Wiederverwendung und des Austauschs von Wissen. Im Rahmen der Import-/Exportvorgänge dürfen die Benutzer individuell entscheiden, wie viele Informationen aus einem „Process Template“ übernommen, beziehungsweise in einem Template gespeichert werden sollen. Die Auswahl erfolgt mithilfe einer graphischen Oberfläche in Form eines Wizards, der den Benutzer durch eine Reihe von Dialogfenstern führt.

Abstract

Import and Export of Process Templates as a component of Process Support based on Active Documents

The project α-Flow aims to develop distributed electronic case files, which are called α-Docs. α-Docs are particularly designed for the application in inter-institutional treatment processes. They are stored and edited in the participants' local IT infrastructure, while the information is synchronized using the Internet. A new participant solely needs to gain access to a copy of the distributed case file. Since medical documents are traditionally available in hard copy, a document-oriented paradigm, with Active Documents in particular, is used. Active Documents contain medical payloads as well as meta information describing the treatment process, and are defined by its active properties. In α-Flow a rule-based action library is one of these properties.

Until now, in the beginning of a new episode, an entire process planning has to be done, even if treatments are very similar to each other. Thus, it takes a lot of effort to create a new α-Doc and the number of treatment errors caused by wrong planning may be high. Therefore, in this thesis an additional component for the import and export of Process Templates is developed. Process Templates model structured treatment processes in order to support their reuse and to provide knowledge exchange. During the import and export of Process Templates, users may individually decide how much template information they want to use and how much information they want to store in a Process Template respectively. The choices are made using a wizard-based Graphical User Interface (GUI), in which the user is guided through a series of dialogs.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

APA	Adornment Prototype Artifact
CRA	Collaboration Resource Artifact
CRC	Class-Responsibility-Collaboration
DOM	Document Object Model
EAV	Entity-Attribute-Value
ER	Entity-Relationship
GUI	Graphical User Interface
HTML	Hypertext Markup Language
ID	Identifier
IDE	Integrated Development Environment
JAXB	Java Architecture for XML Binding
PC	Personal Computer
POJO	Plain Old Java Object
PSA	Process Structure Artifact
SAX	Simple API for XML
UML	Unified Modelling Language
XML	Extended Markup Language
XSLT	XSL Transformation

1 Einleitung

Unsere heutige moderne Gesellschaft ist ohne Computer nicht mehr vorzustellen. Neben den Vorzügen im Alltag profitieren wir insbesondere in der Arbeitswelt von der Rechenleistung und dem nahezu grenzenlosen Gedächtnis von PCs. Gerade in der Gesundheitsbranche gibt es eine Flut von unterschiedlichen Informationen, wobei Patientendaten und Laborergebnisse beispielhaft genannt seien. Man kann davon ausgehen, dass heutzutage jeder niedergelassene Arzt ein Praxisinformationsystem benutzt, in dem die Informationen über seine Patienten gespeichert sind. Es kommt jedoch aufgrund der Spezialisierung vieler Ärzte häufig vor, dass Patienten im Rahmen eines Behandlungsprozesses von mehreren Medizinern an vielen verschiedenen Orten behandelt werden. In diesen Fällen besteht daher zusätzlich die Notwendigkeit eines Datenaustauschs. Dieser erfolgt traditionell papierbasiert, in Form von Überweisungsscheinen und Befundberichten, die entweder vom Patienten von Arzt zu Arzt getragen oder auf dem Postweg gesendet werden müssen. Dadurch existiert meist keine vollständige Sammlung von Behandlungs- und Gesundheitsdaten eines Patienten, obwohl das Fehlen von Patienteninformationen eine der größten Fehlerquellen bei Patientenbehandlungen darstellt [LBC+95]. Aus diesen Gründen sind die Vorteile einer erweiterten Computer-Unterstützung für die dezentrale Behandlung von Patienten besonders groß.

1.1 Motivation

Das Projekt α-Flow verfolgt den Ansatz einer verteilten, elektronischen Fallakte [Haa05] und versucht dabei die genannten Probleme traditioneller Patientenbehandlungen zu beseitigen [NL12]. Diese Fallakten werden α-Docs genannt und werden speziell für die Anwendung in institutionsübergreifenden Behandlungsabläufen entworfen. Aufgrund der Heterogenität der Informationsysteme der verschiedenen Prozessteilnehmer existiert kein zentraler Server, der beispielsweise eine Datenbank anbieten könnte. Die Speicherung und Bearbeitung der α-Docs erfolgt stattdessen auf die autonomen Behandlungspartner verteilt, wobei die Fallakten über das Internet synchronisiert werden. Kommt ein neuer Teilnehmer hinzu, benötigt er lediglich eine lokale Kopie der verteilten Fallakte. Es besteht

kein weiterer Integrationsaufwand, wie beispielsweise bei anderen Ansätzen das Installieren einer neuen Workflow-Engine. Da medizinische Dokumente traditionell in Papierform vorliegen, wird außerdem ein dokumentenorientiertes Paradigma verfolgt [BKL05]. Dabei wird insbesondere das Konzept von Aktiven Dokumenten [LED+99], welche neben ihren medizinischen Nutzdaten auch Information zum Prozess enthalten, umgesetzt. Aktive Dokumente besitzen eine Reihe von aktiven Eigenschaften, wobei im Rahmen von α-Flow besonders das umfassende Regelwerk zur Prozesssteuerung zu erwähnen ist. Das „α“ in α-Flow soll diese aktiven Eigenschaften der α-Docs unterstreichen.

1.2 Ziele der Arbeit

In α-Flow stellen die benötigten medizinischen Dokumente die einzelnen Prozessschritte einer Patientenbehandlung dar. Dabei kann es sich unter anderem um Untersuchungsergebnisse oder Überweisungsscheine handeln. Zu Beginn einer Behandlungsepisode erstellt beispielsweise der Hausarzt einen grundlegenden Behandlungsplan, indem er eine Menge von benötigten Dokumenten bestimmt. Zu jedem anfallenden Dokument müssen dann eine Reihe von Zusatzinformationen eingegeben werden. Dabei handelt es sich zum Beispiel um den Namen des verantwortlichen Mediziners sowie um dessen Institution. Dies sorgt für einen hohen initialen Aufwand bei der der Erstellung eines α-Docs. Da viele Behandlungsepisoden häufig ähnlich ablaufen (beispielsweise ein typischer Brustkrebsdiagnostik-Prozess) wäre es nützlich, vergangene Abläufe auf einfache Art und Weise wiederholen zu können. Eine Wiederholung schließt dabei sowohl die einzelnen Prozessschritte, in Form von benötigten Dokumenten, als auch die Prozessteilnehmer ein. Hierfür kann man das Konzept der sogenannten „Process Templates“ nutzen. Diese modellieren strukturierte Abläufe zum Zwecke der Wiederverwendung und um medizinisches Wissen austauschen zu können.

Das Ziel der Arbeit ist daher die Verwendung von „Process Templates“ in α-Flow zu ermöglichen. Dazu muss eine Import-/Exportkomponente für „Process Templates“ entwickelt werden. Eine Herausforderung besteht darin, dass bislang weder Standards für die Repräsentation von „Process Templates“ noch Leitlinien für deren Wiederverwendung existieren [MZM04]. Deshalb muss eine geeignete Darstellungsform für „Process Templates“ konzipiert werden. Damit der Benutzer bestimmen kann, wie viele Informationen er importieren/exportieren möchte, muss außerdem eine geeignete graphische Benutzeroberfläche gewählt und umgesetzt werden. Diese Oberfläche sollte möglichst intuitiv und übersichtlich gestaltet sein, da die Vorgänge nicht Teil des ärztlichen Routinebetriebs

sind und den Benutzer nicht überfordern dürfen. Um die Vorgaben des Nutzers umsetzen zu können, müssen weiterhin verschiedene Informationsfilter entwickelt werden. Dabei handelt es sich um Filter für Prozessschritte, Prozessteilnehmer, Metainformationen sowie Metainformations-Datentypen.

Das Thema Datenschutz in der Medizin ist, unter anderem im Rahmen der ärztlichen Schweigepflicht, von hoher Bedeutung. Trotzdem wird in dieser Arbeit nicht vertieft auf die Problematik eingegangen. Der Benutzer muss stattdessen selbst sicherstellen, dass der Import und Export sowie insbesondere die Weitergabe von „Process Templates“ rechtlich legitimiert ist.

2 Methodik

Das folgende Kapitel beschreibt die Vorgehensweise bei der Entwicklung der α-Templates-Komponente. Der Aufbau der Arbeit ist an die einzelnen Vorgehensphasen angelehnt, weshalb bei der Beschreibung der Schritte das korrespondierende Kapitel referenziert wird.

Zu Beginn der Arbeit erfolgt zur Einarbeitung in das Themengebiet eine umfassende Literaturrecherche. In Kapitel 3 werden die technischen und wissenschaftlichen Grundlagen des α-Templates-Subsystems beschrieben. Dabei wird zunächst auf das übergeordnete Forschungsprojekt α-Flow eingegangen, wobei der Fokus auf dessen Datenmodell liegt. Im Anschluss daran wird das Konzept des Workflow-Managements sowie die Idee der „Process Templates“, deren Unterstützung das Ziel der Arbeit ist, erarbeitet.

Danach folgt in Kapitel 4 eine Recherche nach verwandten Arbeiten, die für die Konzeption des α-Templates-Subsystems von Interesse sein können. Dabei werden zum einen Arbeiten betrachtet, die sich mit der Verwaltung und der Repräsentation von Prozessschablonen befassen. Zum anderen werden Ideen aus Software-Systemen, die eine Unterstützung von „Process Templates“ anbieten, extrahiert. Des Weiteren wird auf die Idee der Klinischen Pfade eingegangen, da diese eine hohe Verwandtschaft zu Prozessschablonen, unter anderem aufgrund ihres klinischen Anwendungskontexts, besitzen.

In Kapitel 5 wird die Entwicklung eines Konzepts für die α-Templates-Komponente beschrieben. Zunächst werden dazu die Vorteile von Prozessschablonen erklärt und mithilfe eines Anwendungsszenarios unterstrichen. Daraus werden im Anschluss die funktionalen Anforderungen an die α-Templates-Komponente abgeleitet. Den Abschluss des Kapitels bildet die Vorstellung des Lösungskonzepts. Die Gliederung der Beschreibung ist hierbei an die drei Bestandteile der typischen Drei-Schichten-Architektur, welche aus Datenhaltungs-, Logik- und Präsentationsschicht besteht, angelehnt.

Kapitel 6 behandelt schließlich die Umsetzung eines Prototypen, der das zur erarbeitende Konzept umsetzt. Zu Beginn liegt der Fokus auf dem äußeren Verhalten der α-Templates-Komponente, wobei dessen Zusammenspiel mit den weiteren Subsystemen erklärt wird. Dies geschieht mithilfe von CRC-Karten sowie einfachen Unified

Modelling Language (UML)-Sequenzdiagrammen. Im Anschluss daran wird der Software-Feinentwurf geschildert. Dabei wird der Aufbau der α-Templates-Komponente, wiederum entlang der Drei-Schichten-Architektur, erarbeitet. Das innere Verhalten wird mit detaillierten UML-Sequenzdiagrammen modelliert, während die Visualisierung der Software-Bausteine mit UML-Klassendiagrammen erfolgt. Auf eine Evaluation der Performance der α-Templates-Komponente wurde verzichtet, da das Subsystem nicht Teil des ärztlichen Routinebetriebs ist. Aufgrund der Seltenheit von Import-, beziehungsweise Exportvorgängen ist deren Geschwindigkeit daher von nachrangigem Interesse.

In Kapitel 7 wird ein Ausblick auf offene Arbeiten an der α-Templates-Komponente gegeben und eine sinnvolle Nutzung von Prozessschablonen motiviert. Bei den zukünftigen Arbeiten wird unter anderem auf Ideen eingegangen, die sich aus der Literaturrecherche in Kapitel 4 ergeben haben und im Rahmen des α-Templates-Subsystems noch nicht umgesetzt werden.

3 Grundlagen

In diesem Kapitel werden die technischen und wissenschaftlichen Grundlagen der α-Templates-Komponente beschrieben. Dabei wird in 3.1 zunächst ein Grobüberblick über das übergeordnete Projekt α-Flow gegeben. Danach wird in 3.2 der Begriff der „Process Templates“ eingeführt und definiert. Zum Abschluss werden in Abschnitt 3.3 die Grundlagen des Workflow-Managements vorgestellt.

3.1 α-Flow

Dieser Abschnitt erklärt die Grundprinzipien von α-Flow, dessen zugrundeliegendes Datenmodell sowie die Aufgabenverteilung auf die einzelnen Subsysteme. Ausführliche Darstellungen sind in [NL09], [Tod10], [Han10], [TN11] sowie [NL12] zu finden.

3.1.1 Grundlagen

Das Ziel des übergeordneten Projekts α-Flow ist die Entwicklung einer verteilten, elektronischen Fallakte [Haa05]. Das Anwendungsgebiet ist die institutionsübergreifende Patientenbehandlung in heterogenen Informationssystemen. Da im Laufe von Patientenbehandlungen traditionell sehr viele Dokumente anfallen, verfolgt der Workflow-Ansatz ein dokumentenorientiertes Paradigma. Der Workflow wird jeweils durch Hinzufügen neuer Information, zum Beispiel eines Überweisungsscheins, gesteuert. Diese Dokumente müssen neben ihren medizinischen Daten auch vollständige Prozessinformationen, wie beispielsweise die Bezeichnung des Dokuments und den Namen des Arztes, enthalten. α-Docs setzen dabei das Konzept der Aktiven Dokumente um [LED+99]. Diese besitzen eine Reihe von aktiven Eigenschaften, wobei im Rahmen von α-Flow jedes Subsystem eine dieser Eigenschaften darstellt (vergleiche 3.1.3).

Nach dem Einbringen neuer Information muss diese allen Prozessteilnehmern bekannt gemacht werden. Das α-Doc wird daher über das Internet synchronisiert. Nimmt ein neuer Teilnehmer an der Behandlung teil, so benötigt dieser lediglich eine Kopie der verteilten Fallakte, ohne Notwendigkeit weiteren Integrationsaufwands.

3.1.2 Datenmodell

In diesem Abschnitt werden die einzelnen Bausteine des α-Flow-Datenmodells vorgestellt. Ein Behandlungsablauf wird α-Episode genannt und durch ein α-Doc repräsentiert. Das α-Doc ist eine Sammlung von allen Dokumenten, die im Laufe einer Patientenbehandlung entstehen. Die einzelnen Dokumente werden in Form von α-Cards gespeichert, welche in Coordination- und Content-Cards unterteilt sind. Jede α-Card muss einen α-Descriptor und kann eine Payload[1] besitzen. α-Descriptors sammeln Attribute einer α-Card, die sogenannten α-Adornments. Es existiert eine Standardmenge von prozessrelevanten α-Adornments, wie etwa der Name des Patienten (vergleiche 3.1.2). Diese Menge kann jedoch zur Laufzeit erweitert und damit an die spezielle α-Episode angepasst werden. Beispielsweise ist in vielen Behandlungsfällen der Gesundheitszustand des Patienten interessant und kann deshalb als zusätzliches α-Adornment gespeichert werden [Sch11, NSWL11]. Bei Content-Cards entspricht die Payload einem elektronischen Dokument, wie zum Beispiel einem Überweisungsschein oder einem Befundbericht. Aktuell existieren darüber hinaus in jedem α-Doc drei Coordination-Cards, deren Payloads Metainformationsträger des α-Docs darstellen.

Die erste Coordination-Card, das Process Structure Artifact (PSA), enthält eine Sammlung aller α-Card-Identifier (ID)s des α-Docs sowie eine Liste von Beziehungen zwischen einzelnen α-Cards. Eine Beziehung zwischen α-Cards kann erstellt werden, wenn eine α-Card die notwendige Grundlage für eine andere darstellt. Zum Beispiel sollte vor dem Diagnosebericht eines Facharztes ein Überweisungsschein zu diesem existieren.

Die einzelnen Prozessteilnehmer, also beispielsweise die verschiedenen behandelnden Ärzte, werden im Collaboration Resource Artifact (CRA) beschrieben. Zur Synchronisation des α-Docs sind hier für jeden Teilnehmer elektronische Kontaktdaten hinterlegt.

Die letzte Coordination-Card ist das Adornment Prototype Artifact (APA), welches eine Menge verschiedener α-Adornments und deren Standardwerte definiert. Eine Beschreibung der vordefinierten Adornment-Typen findet sich in Abschnitt 3.1.2. Das APA dient als ein Template für alle α-Card-Descriptors. Da sich der Aufbau der α-Descriptors nach dem Entity-Attribute-Value (EAV)-Modell richtet, ist das APA zur Laufzeit veränderbar [Sch11, NSWL11]. In Abbildung 3.1 ist der beschriebene Aufbau eines α-Docs abstrahiert dargestellt. 3.2 modelliert ergänzend den Aufbau der α-Descriptors mithilfe eines Entity-Relationship-Diagramms [Che76].

1 Nutzdaten

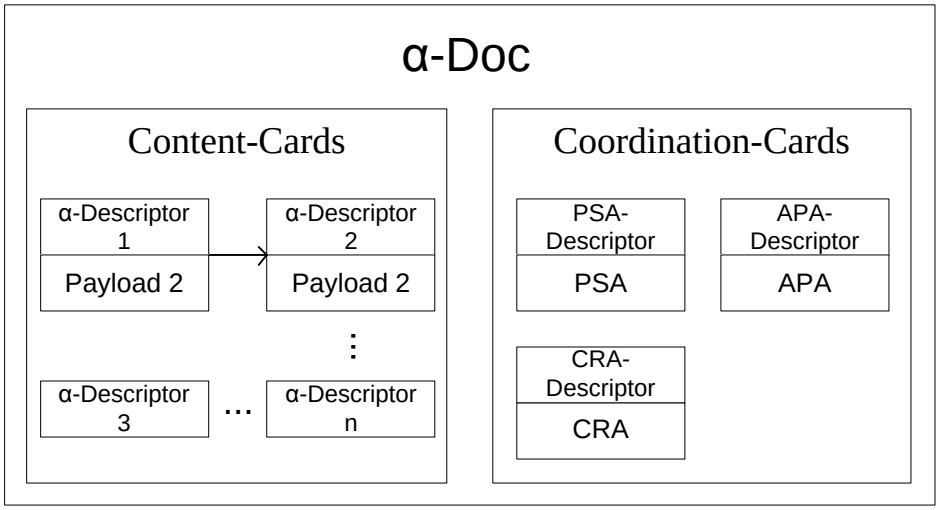


Bild 3.1: Grundlegender Aufbau des α-Docs

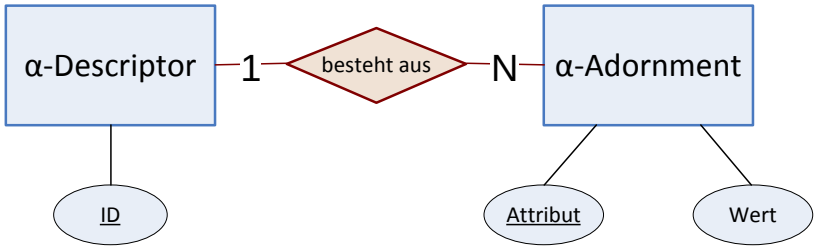


Bild 3.2: Modellierung eines α-Descriptors

Die Grundmenge der α-Adornments

Wie in Bild 3.2 dargestellt, besteht jeder α-Card-Descriptor aus einer Menge von α-Adornments. Dabei existiert eine Reihe von Standard-Adornments, die jeder α-Descriptor besitzt und deren jeweilige Bedeutungen in Tabelle 3.3 erklärt werden.

Name des α-Adornments	Bedeutung
AlphaCard Title	Name der α-Card
Actor ID	Name des Behandelnden
Role ID	Rolle des Behandelnden
Institution ID	Name der Institution
OC ID	Patientenkennung
Visibility	Sichtbarkeit der α-Card unter den Prozessteilnehmern
Validity	Gültigkeit (Abgeschlossenheit) der α-Card
Version	Versionsnummer
Version Control	Sagt aus, ob eine α-Card unter Versionierung steht.
Variant	Variante (Ausprägung) der α-Card
Syn. Payload Type	Datentyp der Payload
Fund. Semantic Type	Semantischer Typ der α-Card (Content- oder Coordination-Card)
AlphaCard Type	Art der α-Card (Dokumentation, Überweisung oder Untersuchungsergebnis)
Due Date	Fälligkeit der α-Card
Deferred	Sagt aus, ob die α-Card verschoben wurde.
Deleted	Sagt aus, ob die α-Card irrelevant geworden ist.
Priority	Priorität der α-Card

Bild 3.3: Die Grundmenge der α-Adornments und deren Bedeutungen

3.1.3 Komponenten

Der folgende Abschnitt beschreibt die wichtigsten Komponenten von α-Flow und deren jeweilige Aufgaben. Die vorgestellten Subkomponenten bilden zusammen die aktiven Eigenschaften des α-Docs und verwirklichen somit die Idee von aktiven Dokumenten [LED+99].

Das in 3.1.2 beschriebene Datenmodell ist in α-Model realisiert. α-Model wird von allen anderen Komponenten benötigt und bildet somit den Grundbaustein von α-Flow. Für den Startvorgang ist das Modul α-Startup zuständig. Es initialisiert die benötigten

Software-Bausteine, die α-Props, den α-VerVarStore, α-Overnet sowie gegebenenfalls den α-Injector.

Das α-Props-Modul implementiert ein zentrales Regelwerk für die verschiedenen Aktivitäten in α-Flow. Die Regeln steuern beispielsweise das Hinzufügen neuer α-Cards und lösen deren Synchronisation über das Netzwerk aus. Der α-VerVarStore sorgt für die Verwaltung der einzelnen α-Cards. Er kontrolliert das Speichern und Laden der α-Cards sowie deren Versionierung. α-Overnet verschickt und empfängt alle relevanten Zustandsänderungen des α-Docs unter den Prozessteilnehmern, die im CRA gespeichert sind. Es sorgt dadurch für die Umsetzung der Datensynchronisation unter den Prozessteilnehmern, welche vom α-Props-Modul initiiert wird.

Der α-Injector fügt einem (Content-)α-Card-Descriptor die Payload hinzu. Eine zentrale Rolle spielt der α-Injector bei der Erstellung des α-Docs. Hierbei übergibt der Benutzer das erste Dokument, beispielsweise ein Anamnesebericht[1], der initialen (Content-)α-Card als Payload, was den Beginn einer α-Episode markiert. Des Weiteren setzt das α-Injector-Subsystem das Hinzufügen neuer Payloads zu α-Card-Descriptors eines bereits visualisierten α-Docs via „Drag and Drop“ um.

Die eingebettete Benutzerschnittstelle ist im α-Editor-Subsystem umgesetzt. Diese Komponente visualisiert die Liste der α-Cards und bietet diverse Formen der Interaktion. Neben der Möglichkeit, α-Cards hinzuzufügen sowie α-Adornments bestehender α-Cards zu editieren, kann man sich hier neue α-Adornment-Typen definieren. Schließlich existiert noch das α-Utils-Modul. Dessen Hauptaufgabe ist die Kapselung von Java Architecture for XML Binding (JAXB)-Funktionalität, also die (De-)Serialisierung der Datensätze (vergleiche 3.1.4). Das beschriebene Zusammenspiel der Komponenten ist in Abbildung 3.4 modelliert. Darin ist zu erkennen, dass das α-Model-Subsystem die Grundlage beinahe aller Subsysteme von α-Flow bildet.

1 Unter Anamnese versteht man die erste Informationssammlung im Rahmen einer Patientenbehandlung. Hierbei werden dem Patienten oder einer Vertrauensperson gezielt Fragen von medizinischem Interesse gestellt.

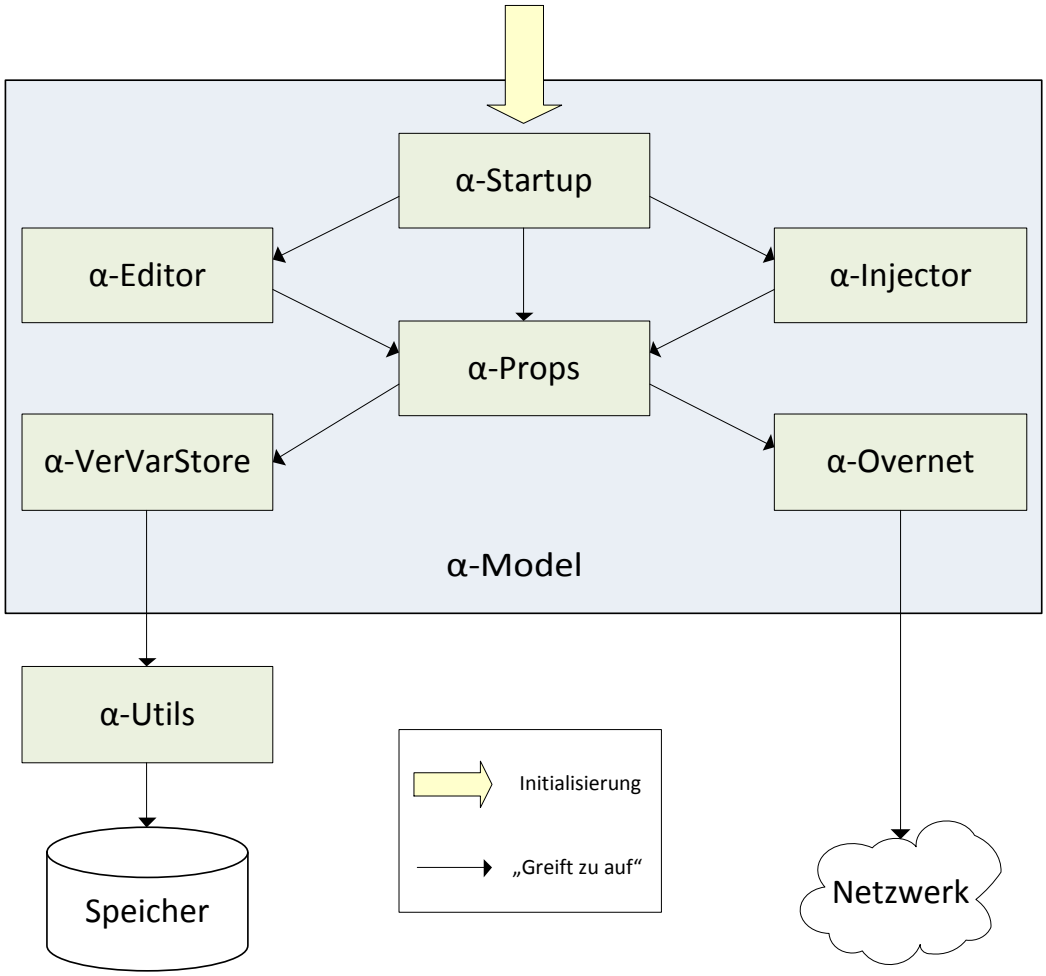


Bild 3.4: Zusammenspiel der wichtigsten Komponenten von α-Flow

3.1.4 Java Architecture for XML Binding

Die JAXB ist eine Programmierschnittstelle für die Extended Markup Language (XML)-Datenbindung in Java. Unter XML-Datenbindung versteht man die Abbildung eines XML-Dokuments auf ein Plain Old Java Object (POJO) und umgekehrt. Die Überführung eines Java-Objekts in das XML-Format wird Marshalling (oder auch Serialisierung), der umgekehrte Weg Unmarshalling (Deserialisierung) genannt. Abbildung 3.5 veranschaulicht diesen Zusammenhang.

Der große Vorteil gegenüber klassischen Ansätzen, wie die Simple API for XML (SAX) oder das Document Object Model (DOM), besteht in der höheren Abstraktionsstufe. Es ist kein zusätzlicher Implementierungsaufwand für das Schreiben von Parsern nötig. Der Programmierer behandelt das spätere XML-Dokument stattdessen wie ein normales

Java-Objekt. Mithilfe von Annotationen im Quelltext wird die spätere Struktur des XML-Dokuments festgelegt. Im Rahmen von α-Flow wird die JAXB für jegliches Speichern und Laden von Daten des α-Models verwendet.

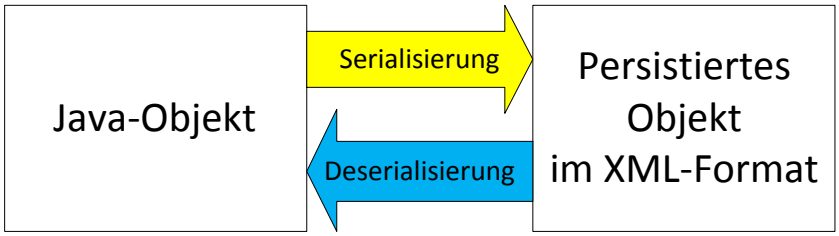


Bild 3.5: XML-Datenbindung mithilfe von JAXB

3.2 „Process Templates“

„Process Templates“, in der Folge auch Prozessschablonen genannt, speichern Abläufe von bestimmten, strukturierten Prozessen in einer abstrahierten Form. Bei Verwendung einer Prozesschablone, muss diese den besonderen Gegebenheiten der jeweiligen Prozessinstanz angepasst werden [DRK97]. „Process Templates“ werden in elektronischer Form gespeichert, was deren automatisierte Verarbeitung erleichtert.

Durch Benutzung von Templates kann die Prozessqualität erheblich steigen, da auf vorhandenes Wissen und Erfahrungen aus früheren Abläufen zurückgegriffen wird. So sinkt beispielsweise die Wahrscheinlichkeit von Fehlern und die Dauer der Ablaufplanung. Mithilfe von Prozessschablonen kann Wissen über Unternehmens- oder Institutionsgrenzen hinweg ausgetauscht werden [LS05].

3.3 Workflow-Management

Bei einem Workflow handelt es sich um eine Sammlung von Aufgaben, die zusammen die Durchführung eines Geschäftsprozesses ermöglichen [GHS95]. Die einzelnen Schritte werden dabei von einer beliebigen Zahl von Akteuren durchgeführt. Die Menge der Akteure kann sowohl Menschen als auch Computer umfassen. Die Reihenfolge der Bearbeitungschritte sowie deren Abhängigkeiten werden ebenfalls in einem Workflow definiert. Des Weiteren beschreibt dieser die Synchronisation zwischen einzelnen Aufgaben und den allgemeinen Informationsfluss.

Andere Autoren definieren einen Workflow analog zu einem Geschäftsprozess. Die erwähnten Zusätze, beispielsweise die Reihenfolge der Bearbeitungsschritte, werden hingegen bereits als erste Form des in der Folge beschriebenen Workflow-Managements betrachtet [vdAvH04]. Im weiteren Lauf der Arbeit werden die Begriffe (Geschäfts-)Prozess und Workflow daher aus Gründen der Vereinfachung synonym verwendet.

Georgakopoulos, Hornick und Sheth [GHS95] verstehen als Aufgaben des Workflow-Managements drei hauptsächliche Aktivitäten. Als erstes muss der umzusetzende Prozess modelliert und in einer Workflow-Spezifikation umgesetzt werden. Je nach gewünschter Abstraktion kann der Workflow in einem anderen Detailgrad spezifiziert werden. Zweitens wird im Rahmen des Workflow-Managements der Geschäftsprozess optimiert. Schließlich muss der Workflow noch auf Basis der erarbeiteten Spezifikation implementiert und automatisiert werden.

Zur Verwaltung und Automatisierung der Schritte werden sogenannte Workflow-Management-Systeme genutzt. Die wichtigsten Hersteller dieser Systeme haben sich dabei zur „Workflow Management Coalition“ zusammengeschlossen. Diese dient der Standardisierung von Komponenten und Schnittstellen, beispielsweise durch das „Workflow Reference Model“ [Hol95].

Im klassischen Workflow- beziehungsweise Geschäftsprozessmanagement ist ein Prozess bereits vor seinem Beginn vollständig bekannt [Swe10]. Der Benutzer soll den Prozess lediglich ausführen, ihn jedoch nicht modifizieren. Im Gegensatz dazu steht in α-Flow das Prozessmodell zu Beginn einer Behandlungsepisode noch nicht fest. Hier kann und muss der Nutzer den Prozess laufend anpassen. Es besteht insbesondere die Möglichkeit ihn aus einer Sammlung von „Process Templates“ zusammenzustellen.

3.4 Zusammenfassung

Im zurückliegenden Kapitel wurden die technischen und wissenschaftlichen Grundlagen der α-Templates-Komponente beschrieben. Zu Beginn wurde in 3.1 das übergeordnete Lehrprojekt α-Flow eingeführt. Dabei lag der Fokus auf dem vorhandenen Datenmodell, da dessen Komponenten im Rahmen der Import-/Exportvorgänge verwendet werden. Um die Funktionsweise von α-Flow zu verstehen, wurden außerdem dessen einzelne Subkomponenten beschrieben. Das Ende des Abschnitts markierte eine Einführung der in α-Flow verwendeten JAXB. Diese vereinfacht den Umgang mit XML-Dateien durch eine Abstraktion des Aufbaus.

Danach wurden in 3.2 „Process Templates“ eingeführt und charakterisiert. Sie modellieren die Abläufe von strukturierten Prozessen zum Zwecke der Wiederverwendung.

Abschließend erfolgte in 3.3 eine Vorstellung der Grundlagen des Workflow-Managements. Workflow-Management umfasst unter anderem Spezifikation, Optimierung und Implementierung von Workflows.

4 Verwandte Arbeiten

In diesem Kapitel werden verwandte Konzepte sowie spezielle Anwendungsprogramme, die Prozessschablonen einsetzen, vorgestellt. Es beginnt in 4.1 mit einem Ansatz einer ontologiebasierten, einheitlichen Speicherung von Prozessschablonen. Danach wird in 4.2 das Konzept von „Process Repositories“ vorgestellt. Abschnitt 4.3 beschreibt ein fallbasiertes Workflow-Modell-Verwaltungssystem. In der Folge wird in 4.4 eine Plattform zur kooperativen Software-Entwicklung mit Unterstützung von Templates vorgestellt. Das Kapitel schließt in 4.5 mit der Beschreibung von Klinischen Pfaden ab.

4.1 Vereinheitlichung von Prozessschablonen

Ein Ansatz, um eine hohe Wiederverwendbarkeit von „Process Templates“ zu erreichen, besteht darin, sie in einer einheitlichen Beschreibungssprache zu modellieren. Dadurch werden sie unabhängig von den Datenmodellen eingesetzter Software, was den Einsatz in verschiedensten Umgebungen ermöglicht. Es muss vom jeweiligen Anwendungsprogramm lediglich eine Schnittstelle zur Prozessschablone implementiert werden.

Lin und Strasunkas haben hierzu eine ontologiebasierte Modellierungsform geschaffen [LS05]. Mithilfe einer Ontologie wird das bestehende Datenmodell in ein einheitliches Template-Format übertragen. Dabei werden beispielsweise Synonyme wie „Client“ und „Customer“ zusammengefasst. Um den Nutzen solcher Templates weiter zu erhöhen, können sie mit Metainformationen, beispielsweise zum Einsatzgebiet, versehen werden. Bei Sammlungen von Prozessschablonen erhöht diese Maßnahme, neben der einheitlichen Struktur, die Wahrscheinlichkeit, eine passende Schablone zu finden.

Für die im Rahmen dieser Arbeit vorgestellten α-Templates existiert jedoch eine feste Einsatzumgebung. Sie werden zunächst nur für die Verwendung in α-Flow konzipiert. Daher wäre es nach aktuellem Stand unnötig, die „Process Templates“ universell zu gestalten. Aufgrund des erleichterten Wissensaustauschs bleibt der Ansatz für die Zukunft trotzdem interessant. Dies gilt insbesondere für den möglichen Aufbau eines „Process Repositories“, dessen Konzept im folgenden Abschnitt erklärt wird.

4.2 „Process Repositories“

Je mehr „Process Templates“ vorhanden sind, desto größer wird der Bedarf nach einer Möglichkeit der Template-Verwaltung. Hierfür können sogenannte „Process Repositories“ verwendet werden, die das Speichern und Wiederauffinden von Prozessschablonen unterstützen. Es existieren bereits diverse Ansätze, wie das „MIT Process Handbook“ [MCH03], die „SAP Business Maps“ [SAP] und die „IBM Patterns for E-Business“ [IBM]. Diese Repositories sind jedoch zumeist auf spezielle Anwendungsgebiete beschränkt [WZJ]. Auch die Erweiterung der Repositories um neue Prozesse ist häufig nur durch die Anbieter selbst möglich. Eine globale Suche in verschiedenen Sammlungen wird durch die unterschiedlichen Abstraktionsstufen in der Prozessmodellierung erschwert.

Daher haben Wohed, Zdravkovic und Johannesson ein Konzept eines universellen „Process Repositories“ entwickelt [WZJ]. Dieses Repository ist frei verfügbar und soll langfristig die Basis einer „Business Process Wikipedia“ bilden. Um das Wiederauffinden von Prozessschablonen zu erleichtern, werden zu den einzelnen Prozessen zusätzliche Metadaten gespeichert, die beispielsweise deren Beziehungen untereinander oder die allgemeinen Ziele des Prozesses umfassen. Den Benutzern können dabei verschiedene Rollen zugewiesen werden. Dadurch kann beispielsweise das Recht, neue Prozessschablonen hinzufügen zu dürfen, verliehen werden.

Sobald in Zukunft eine gewisse Anzahl von Schablonen vorhanden ist, könnte auch in α-Flow ein Repository für „Process Templates“ von Interesse sein. Das Hinzufügen von Metadaten (zum Beispiel Tags) zur Beschreibung von Templates, wäre hierfür nötig, könnte jedoch auch unabhängig davon den Informationsgehalt von Prozessschablonen erhöhen.

4.3 CODAW

Um vorhandenes Fallwissen wiederverwenden sowie neue Workflows unter Zusammenführung von Wissen erstellen zu können, haben Madhusudan, Zhao und Marshall das CODAW[1]-System erarbeitet [MZM04]. Es basiert auf dem Paradigma des fallbasierten Schließens [Kol93] und besteht insgesamt aus dem Wiederauffinden, Wiederverwenden, Anpassen und Verifizieren von Fällen. Bei einem Fall handelt es sich hier um ein in der Vergangenheit aufgetretenes und gelöstes Problem. Die Repräsentation der Fälle

1 Case-Oriented Design Assistant for Workflow Modeling

erfolgt dabei zum einen prozedural, mithilfe einer graphbasierten Prozessdarstellung. Zum anderen werden die Fälle zusätzlich deklarativ, in einer regelbasierten Form, beschrieben. Das Wiederauffinden von Workflow-Modellen erfolgt nach dem Prinzip des „Similarity Floodings“ [MGMR02]. Dabei werden die Strukturen von Prozessgraphen auf Ähnlichkeiten mit dem Benutzerwunsch geprüft. Eine Zusammenführung von Fällen erfolgt nun logikbasiert, wobei die deklarative Beschreibung der Prozesse benötigt wird. Die Workflows sind hierarchisch, aus Schritten und Teilschritten, aufgebaut. Es wird versucht, Vor- und Nachbedingungen des gesamten Workflows sowie dessen einzelner Schritte zu erfüllen. Die Nachbedingung des Workflows kann dabei auch als dessen Ziel bezeichnet werden.

Das vorgestellte Konzept ist insbesondere für eine spätere Entwicklung eines Template-Repositories (vergleiche 4.2) von Interesse. Dabei muss beispielsweise eine Methode zum Wiederauffinden von Prozessschablonen entwickelt werden, die ebenfalls durch eine Suche nach größter Übereinstimmung umgesetzt werden kann. Auch der hierarchische Aufbau von Workflows und Möglichkeiten der (De-)Komposition von Templates (vergleiche 7.10) sind relevant für die α-Templates-Komponente.

4.4 Jazz und „IBM Rational Team Concert“

Jazz ist eine erweiterbare Plattform für die kooperative Software-Entwicklung. Es bietet eine Datenbank, ein Projektarchiv, sowie weitere grundlegende Dienste an. Beispielsweise werden Funktionen zur Benutzerauthentifizierung und -autorisierung, sowie die Persistierung der Anwendungsdaten, realisiert.

„IBM Rational Team Concert“ ergänzt diese Basistechnologie um Bausteine zur Unterstützung des Software-Entwicklungsprozesses[1]. Es umfasst unter anderem Versionskontrolle, ein Projektplanungssystem, eine Dokumentationskomponente und eine Funktion zur automatisierten Einrichtung der Build-Umgebung [Lan10]. Die Benutzung kann einerseits über Plugins für eine Integrated Development Environment (IDE) wie Eclipse oder Visual Studio erfolgen. Andererseits besteht die Möglichkeit über eine Webschnittstelle direkt auf den Jazz-Server zuzugreifen [MLLL10].

Interessant an diesem Software-System ist, dass eine Import-/Exportfunktion für Prozessschablonen vorhanden ist. Mithilfe von Prozessvorlagen können einzelne Projektbereiche konfiguriert werden. Es existiert bereits eine große Sammlung von Prozess-

1 http://www-01.ibm.com/software/rational/products/rtc/

schablonen, beispielsweise für die agile Software-Entwicklung mit Scrum [Sch04]. Die Templates werden grundsätzlich im XML-Format gespeichert. Sie können jedoch zusätzlich Prozessdokumentation, beispielsweise eine Erklärung des Scrum-Prozesses in Form von zip[1]-archivierten Hypertext Markup Language (HTML)-Dokumenten, enthalten. Bereits erstellte Templates können in einem speziellen Editor angepasst werden. Neben der Import-/Exportfunktion existiert zusätzlich die Möglichkeit, ein bereits laufendes Projekt auf Basis einer anderen Prozessschablone zu migrieren. Prozessschablonen der Jazz/"Rational Team Concert"-Umgebung können außerdem in verschiedenen Sprachen vorliegen [IBM09].

Der beschriebene Ansatz kann als eine wertvolle Inspirationsquelle für die α-Templates-Komponente dienen. Ebenso wie bei α-Flow haben die geschilderten Templates eine feste Einsatzumgebung und erweitern diese um zusätzliche Funktionalität.

4.5 Klinische Behandlungspfade

Klinische Behandlungspfade beschreiben die Behandlungsabläufe von unterschiedlichen Patientengruppen im Krankenhaus. Für die jeweiligen Behandlungsschritte werden Terminierung, Inhalte und die Verantwortlichkeitsverteilung festgelegt. Klinische Pfade stellen einen Konsens zwischen den verantwortlichen Medizinern einer Institution dar. Den Verantwortlichen stehen dabei individuelle Abweichungen, innerhalb eines gewissen Handlungskorridors, frei [RHH+03]. Sie dienen hauptsächlich der Kostenkontrolle und der Qualitätsverbesserung im Krankenhaus [LO06].

Pfade basieren häufig auf Medizinischen Leitlinien, unter zusätzlicher Berücksichtigung der jeweiligen Standorteigenschaften und Ressourcen. Diese Einflüsse werden in Abbildung 4.1, welche auf einer Grafik in [LO06] basiert, dargestellt. Leitlinien sind systematisch entwickelte Empfehlungen, die der Entscheidungsunterstützung bei der Patientenbehandlung dienen [FL90]. Sie existieren für viele verschiedene Anwendungsfälle, beispielsweise für das Vorgehen bei einer Sepsis[2][RBB+06].

Genau wie α-Templates beschreiben Klinische Pfade die Abläufe von Patientenbehandlungen. Im Unterschied zu Pfaden, die den Behandlungsablauf innerhalb einer Klinik beschreiben, erfolgt die Behandlung in α-Flow jedoch stark institutionsübergreifend. Außerdem spielt bei α-Templates die Einplanung von Ressourcen bislang keine Rolle.

1 Format zur Datenarchivierung und -komprimierung
2 Blutvergiftung

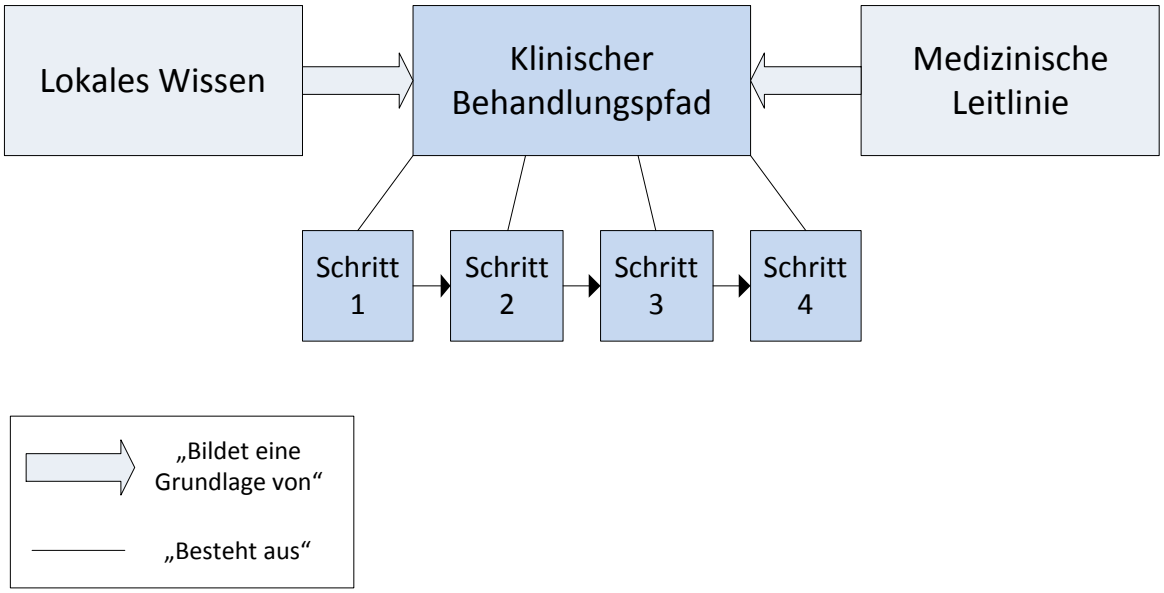


Bild 4.1: Entwurf eines Klinischen Pfades

4.6 Zusammenfassung

In diesem Kapitel wurden einige zur α-Template-Komponente verwandte Arbeiten und Software-Komponenten vorgestellt. Zunächst wurde dabei in 4.1 ein Modell von vereinheitlichten, ontologiebasierten Prozessschablonen vorgestellt, welche unabhängig von eingesetzter Software sind und so in jeder neuen Umgebung wiederverwendet werden können.

Im Anschluss daran folgte in 4.2 die Beschreibung von „Process Repositories“. Diese verwalten eine Sammlung von Workflows, indem sie das Speichern und Wiederauffinden von Workflows unterstützen.

In 4.3 wurde ein Software-System vorgestellt, welches das Konzept des fallbasierten Schließens umsetzt. Das System schlägt dem Benutzer beispielsweise diejenigen Workflow-Modelle vor, die die größte Ähnlichkeit mit dem gewünschten Ablauf haben.

Danach wurde in 4.4 das Software-Entwicklungswerkzeug „IBM Rational Team Concert“ sowie dessen zugrundeliegende Plattform Jazz vorgestellt. Dieses Software-Paket bietet eine Unterstützung von „Process Templates“ an und zeigt damit eine Verwandtschaft zu α-Flow und der α-Templates-Komponente.

Abschließend wurde in 4.5 das Konzept von Klinischen Pfaden vorgestellt. Diese modellieren die Abläufe von Patientenbehandlungen in Kliniken. Aufgrund ihrer Wie-

derverwendbarkeit und ihres Anwendungskontexts ähneln sie den zu erarbeitenden α-Templates und können daher als inhaltliche und konzeptionelle Grundlage dienen.

5 Konzept

Im folgenden Kapitel wird das Konzept der α-Templates-Komponente vorgestellt. Zu Beginn werden im Rahmen des Fachkonzepts in 5.1 der Nutzen der α-Templates-Komponente sowie verschiedene Anwendungsszenarien beschrieben. Aus den Szenarien werden dann innerhalb der Anforderungsanalyse in 5.2 die fachlichen Anforderungen extrahiert. Abschließend wird in 5.3 das Lösungskonzept, das die Benutzeranforderungen umsetzt, vorgestellt.

5.1 Fachkonzept

Im Fachkonzept werden zunächst die Vorteile und der Nutzen der α-Templates-Komponente erklärt. Daraufhin wird die Verwendung anhand eines konkreten Anwendungsszenarios beschrieben. In den folgenden beiden Teilabschnitten werden zwei typische Anwendungsfälle von Prozessschablonen verfeinert betrachtet.

5.1.1 Der Nutzen von α-Templates

Das Ziel der α-Templates-Komponente ist die Möglichkeit, typische Behandlungsabläufe wiederverwendbar zu machen. Ein Behandlungsablauf wird hierfür in Form von Prozessschablonen, sogenannten „Process Templates“, gespeichert. Die Schablonen können daraufhin als Grundlage zukünftiger Patientenbehandlungen verwendet werden. Der Prozess verläuft dokumentenbasiert, weshalb jedes anfallende Dokument (Untersuchungsergebnisse, Überweisungsscheine etc.) einen Prozessschritt darstellt. „Process Templates“ bringen eine Reihe von Vorteilen für alle Beteiligten mit sich.

Der Aufwand bei der Erstellung, beziehungsweise der Erweiterung, eines individuellen Behandlungsplans, wird gesenkt. Man kann auf ein Grundgerüst von Behandlungsschritten aus der Prozessschablone zurückgreifen. Die gesparte Zeit kann für eine bessere Betreuung des Patienten verwendet werden oder auch der Entlastung des medizinischen Personals dienen.

Des Weiteren wird durch die Anwendung von Prozessschablonen die Häufigkeit von Behandlungsfehlern potentiell gesenkt. In der Prozessschablone sind die typischen Maßnahmen einer Behandlung dokumentenbasiert gespeichert. Bei Benutzung des Templates stehen diese dem Benutzer zur Auswahl. Dadurch sinkt die Wahrscheinlichkeit, unnötige oder sogar schädliche Behandlungen, respektive Untersuchungen, anzusetzen. Für den Kostenträger kann dies eine signifikante Einsparung bedeuten. Auch wird der Patient nicht unnötigem Risiko, beispielsweise durch Röntgenstrahlen, ausgesetzt. Stattdessen erfährt er idealerweise die Behandlung, die den größten und schnellsten Erfolg verspricht. Für diesen Punkt wird vorausgesetzt, dass die Prozessschablonen regelmäßig gewartet und dem aktuellen medizinischen Kenntnisstand angepasst werden.

5.1.2 Anwendungsszenario

Analog zu [NL12] soll in diesem Abschnitt mithilfe eines kurzen, fiktiven Anwendungsszenarios die typische Verwendung der α-Templates-Komponente illustriert werden. Außerdem werden typische Komponenten von α-Flow sowie dessen Datenmodell in ihrem Anwendungskontext eingeführt.

Dr. Müller, ein angesehener Gynäkologe aus Nürnberg, ist stets auf der Suche nach modernen Methoden und Techniken, die seine Arbeit verbessern. Seit einiger Zeit benutzt er deshalb in seiner Praxis α-Docs als elektronische Fallakten. Aus Erfahrung weiß er, dass der Behandlungsprozess bei einem Brustkrebsverdacht meist sehr ähnlich abläuft. Das Behandlungs-Team besteht dabei häufig aus dem Radiologen Dr. Meyer, ebenfalls aus Nürnberg, sowie Medizinern des Brustkrebszentrums in Erlangen. Mit diesen Kollegen hat Dr. Müller bislang sehr erfolgreich zusammengearbeitet, wodurch sich ein gutes Vertrauensverhältnis entwickelt hat. Nun möchte der Arzt auch von den Vorteilen der α-Templates profitieren.

Erstellung einer Prozessschablone

Zur Erstellung eines „Process Templates“ (α-Template) öffnet er die elektronische Fallakte (α-Doc) eines abgeschlossenen, möglichst beispielhaften, Brustkrebsdiagnostik-Prozesses in einer graphischen Oberfläche (α-Editor). Als erstes klickt er auf die Export-Schaltfläche und bestimmt den gewünschten Speicherort des α-Templates. Dann kann er alle Dokumente (α-Cards) auswählen, die in zukünftigen Behandlungen voraussichtlich wieder benötigt werden. Bei diesen Dokumenten kann es sich beispielsweise um Untersuchungsergebnisse oder um Überweisungsscheine handeln. α-Cards bestehen aus dem eigentlichen

Dokument (Payload) sowie aus einer Sammlung von Metadaten (α-Descriptor). Die einzelnen Informationen werden α-Adornments genannt und bilden in ihrer Gesamtheit den α-Descriptor. Der Gynäkologe kann für jedes Dokument separat entscheiden, welche Metadaten er im α-Template speichern möchte. Metadaten sind beispielsweise der Name der α-Card (z.B. Anamnesebericht), die Rolle der behandelnden Person (Gynäkologe) und die konkrete Person, der Akteur, der die Maßnahme durchführt (Dr. Müller). Je nach Art der zukünftigen Anwendung kann Dr. Müller mehr oder weniger α-Adornments in das α-Template übernehmen (siehe 5.1.3). Da man sich in einem α-Doc auch eigene α-Adornment-Typen definieren kann, wählt der Arzt unter diesen ebenfalls die Wiederverwendbaren aus. Dr. Müller möchte zum Beispiel bei seinen Patienten stets den Gesundheitszustand dokumentiert haben [Sch11]. Auch die beteiligten Prozessteilnehmer, welche sowohl die Ärzte als auch den Patienten umfassen, können nun bei Bedarf exportiert werden (vergleiche 5.1.3). Da der Gynäkologe, wie eingangs erwähnt, gerne mit dem Radiologen Dr. Meyer sowie den Ärzten des Brustkrebszentrums Erlangen zusammenarbeitet, wählt er diese für den Export aus. Schließlich ist die Prozessschablone, das α-Template, fertiggestellt und kann eingesetzt werden. Kommt nun eine neue Patientin mit Verdacht auf Brustkrebs in die Praxis des Gynäkologen, kann Dr. Müller die Prozessschablone auf zweierlei Art anwenden.

Erstellung eines neuen α-Docs auf Basis einer Prozessschablone

Die erste Möglichkeit besteht darin, das α-Template per „Drag and Drop“ auf ein spezielles Desktop-Symbol zu ziehen. Dies löst die Erstellung eines neuen α-Docs aus. Im Anwendungsszenario in [NL12] wird ein neues α-Doc stets auf diese Art und Weise erstellt. Dabei wird das erste anfallende Dokument auf das erwähnte Desktop-Symbol gezogen und bildet die Payload der initialen (Content-)α-Card. Im Gegensatz dazu basiert das neue α-Doc nun auf den Informationen der Prozessschablone.

Analog zum Exportieren der Schablone trifft Dr. Müller per Mausklick einige Entscheidungen. Er wählt zunächst aus, welche α-Cards und welche jeweiligen α-Adornments in der neuen Behandlungsepisode benötigt werden. Des Weiteren bestimmt er, welche α-Adornment-Typen er aus dem α-Template im Rahmen der Behandlung zur Verfügung gestellt haben will. Der Arzt wählt nun noch die Prozessteilnehmer aus dem α-Template, die an der neuen Behandlungsepisode teilnehmen sollen. Abschließend ergänzt der Arzt die behandlungsspezifischen Daten und die neue α-Episode kann beginnen.

Ergänzung eines bestehenden α-Docs um Informationen einer Prozessschablone

Möchte Dr. Müller kein neues α-Doc erstellen, sondern nur Informationen aus dem α-Template hinzufügen, kann er sich für die zweite Möglichkeit des Imports entscheiden. Dafür klickt er im α-Editor auf die Import-Schaltfläche und wählt die Prozessschablone im Dateisystem aus. Er arbeitet die gleichen weiteren Schritte ab und erhält ein α-Doc mit zusammengeführten Mengen von α-Cards, α-Adornment-Typen und Prozessteilnehmern. Nach Ergänzung der behandlungsspezifischen Daten ist die Grundlage für eine erfolgreiche α-Episode geschaffen.

5.1.3 Granularitätsstufen bei der Template-Erstellung

Bei der Erstellung von α-Templates kann man diese für verschiedene Einsatzzwecke optimieren. Je nach gewünschter Art der Wiederverwendung kann die Speicherung einer unterschiedlichen Anzahl von Informationen sinnvoll sein. Im folgenden werden daher beispielhaft die Erstellung eines allgemeinen Behandlungsschemas und die Schaffung einer Prozessschablone für ein gleichbleibendes Behandlungsumfeld motiviert. In diesen beiden Szenarien geht es hauptsächlich um die Frage, ob und in welchem Umfang Prozessteilnehmer exportiert werden sollen. Jeder Prozessteilnehmer hat eine bestimmte Rolle (z.B. Gynäkologe) und eine Akteurbezeichnung (z.B. Dr. Müller). Potentiell kann der Arzt den Umfang der zu exportierenden Teilnehmerinformationen nun auf verschiedene Art und Weise bestimmen. Zum einen könnte je α-Card eine bestimmte Stufe (beispielsweise ein vollständiger Export von Rolle und Akteur) ausgewählt werden. Zum anderen könnte die Auswahl auch je Prozessteilnehmer erfolgen und schließlich auf alle α-Cards angewendet werden. Im Rahmen dieser Arbeit wird die Umsetzung der ersten Variante beschrieben.

Generalisierte α-Templates

Mithilfe von α-Templates ist es möglich, ein allgemeines Behandlungsschema in Form einer Prozessschablone zu speichern. Hierbei wird das Template so allgemein wie möglich gehalten und auf die eigentlichen Behandlungsschritte reduziert. Man kann diese Form eines α-Templates weitergeben und in jeder neuen Umgebung sowie bei jedem neuen Patienten wiederverwenden. Auch aus datenschutzrechtlichen Gründen kann diese Art von α-Templates sinnvoll sein, da keinerlei Patienteninformationen enthalten sind. Um dies zu erreichen, wird der Benutzer in diesem Fall auf den Export jeglicher α-Adornments, die behandlungs- oder patientenspezifische Daten beinhalten, verzichten. Außerdem wird er

die Daten der Prozessteilnehmer nicht in das α-Template übernehmen. Die Informationen der Schablone beschränken sich nur auf die verschiedenen anfallenden Dokumente, stellen also eine Art universelle Arbeitsliste[1] dar. Die Zuständigkeiten bleiben unverteilt. Eine Illustration am Beispiel der Brustkrebsdiagnostik findet sich in 5.1. Sie basiert auf den Beschreibungen in [NL10].

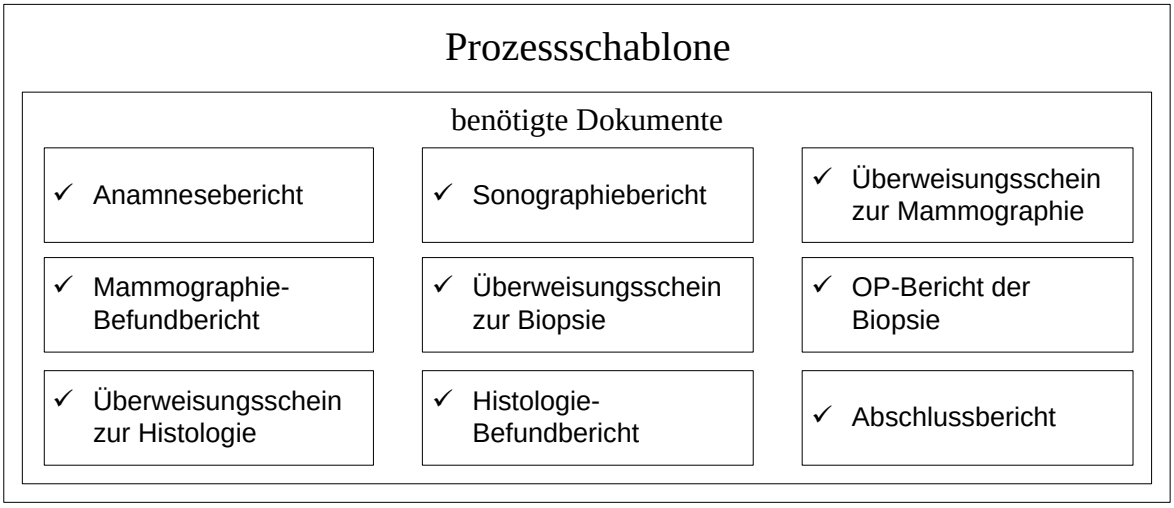


Bild 5.1: Allgemeines Modell einer Prozessschablone für die Brustkrebsdiagnostik

α-Templates für Behandlungen in gleichbleibendem Umfeld

Im vorigen Abschnitt wurde der Anwendungszweck eines generalisierten α-Templates beschrieben. Es kommt in der Praxis jedoch häufig vor, dass bestimmte Behandlungspartner bevorzugt zusammenarbeiten. Der aus der „User Story“ (5.1.3) bekannte Dr. Müller hat beispielsweise gute Erfahrungen mit dem Radiologen Dr. Meyer und dessen Pflegepersonal gemacht. Bei einer allgemein gehaltenen Prozessschablone wäre in diesen Fällen unnötiger und fehlerträchtiger Ergänzungsaufwand nötig. Stattdessen kann dieses Szenario bei der Erstellung des α-Templates berücksichtigt werden. Die Kontaktinformationen des gesamten Behandlungsteams können in der Prozessschablone abgelegt werden. In diesem Fall sind außerdem die Rollen (Gynäkologe, Radiologe) und die Akteure (Dr. Müller, Dr. Meyer) bekannt, die die jeweiligen Handlungen in der Regel durchführen. Diese Informationen werden als α-Adornments der jeweiligen α-Cards gespeichert und ins Template übernommen. Die obige Beispielschablone wird in Abbildung 5.2 um diese Daten ergänzt.

1 Sammlung aller Aufgaben eines Teilnehmers oder einer Teilnehmergruppe

5.1.4 Fazit

In den vorigen Abschnitten wurde die Verwendung von α-Templates motiviert und mithilfe verschiedener Anwendungsszenarien beschrieben. Selbstverständlich sind noch viele weitere Nutzungsmöglichkeiten denkbar. Beispielsweise könnte man die gespeicherten Instanzinformationen auch auf die Rolle (Gynäkologe, Radiologe etc.) beschränken, ohne einen konkreten Arzt zu benennen. Im Allgemeinen lassen sich die vorhandenen Daten der α-Templates stets nachträglich reduzieren. Besteht ein Patient zum Beispiel auf einen bisher unbekannten Arzt, wird lediglich auf den Import etwaig vorhandener Akteurinformationen in den α-Cards sowie des Prozessteilnehmers allgemein verzichtet.

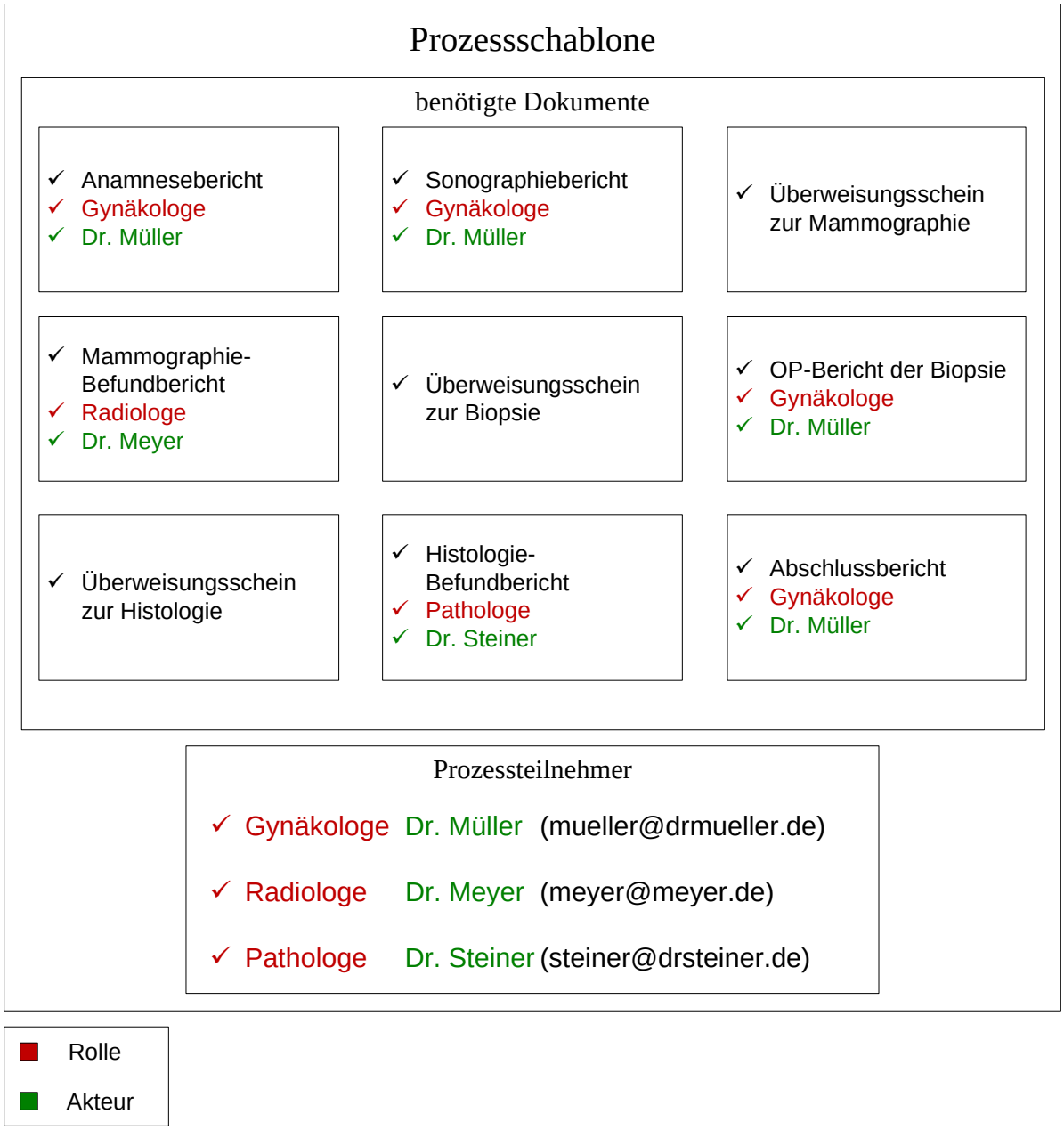


Bild 5.2: Erweiterte Modellierung eines „Process Templates“ für die Brustkrebsdiagnostik, inklusive Rollen und Akteure

5.2 Anforderungsanalyse

Im folgenden Abschnitt werden die fachlichen Anforderungen an α-Templates beschrieben. Zunächst wird die Notwendigkeit aufgezeigt, das grundsätzliche Domänenmodell an die Anforderungen von α-Templates anzupassen. Eine Erweiterung der Benutzeroberfläche und damit des α-Editors wird ebenso benötigt. Danach folgt die Motivation der Erarbeitung von Filtermechanismen für die verschiedenen Datentypen. Abschließend werden noch zusätzliche Veränderungen am bestehenden System beschrieben.

5.2.1 Anpassung des Domänenmodells

Zur Persistierung der α-Templates sollte ein neuer Datentyp konzipiert werden. Beim Export wird das α-Template in dieser Form in einer Datei gespeichert. Das α-Template muss so aufgebaut sein, dass alle Informationen enthalten sind, die beim Import grundlegend benötigt werden.

5.2.2 Erweiterung der Benutzeroberfläche

Damit der Benutzer α-Templates im geöffneten α-Editor importieren und exportieren kann, muss der Editor ergänzt werden. Die einzelnen Anforderungen werden im Folgenden aufgezeigt.

Buttons für den Import und Export für α-Templates - Um den Import- sowie den Exportvorgang von α-Templates starten zu können, muss der α-Editor um Schaltflächen ergänzt werden. Durch Klicken der Buttons soll der jeweilige Vorgang initiiert werden.

GUI für den Import und Export von α-Templates Sowohl beim Import als auch beim Export von α-Templates muss der Benutzer mehrere Entscheidungen treffen. Beim Exportvorgang bestimmt er, welche Daten im Template konserviert werden. Im Laufe des Imports entscheidet er, welche Informationen des Templates er verwenden möchte. Um diese Auswahlentscheidungen treffen zu können, muss ihm eine neue graphische Benutzerschnittstelle zur Verfügung gestellt werden. Die Vorgänge müssen dabei zu jedem Zeitpunkt mithilfe einer geeigneten Schaltfläche beendet werden können.

5.2.3 Verschiedene Möglichkeiten des Imports

Um die in 5.1.2 beschriebenen Anwendungsszenarien umsetzen zu können, müssen zwei verschiedene Formen des Imports konzipiert werden. Zum einen muss die Möglichkeit bestehen, ein α-Template via „Drag and Drop“ auf ein Desktop-Symbol ziehen zu können, um die Erstellung eines neuen α-Docs auf Basis der Schablone zu initiieren. Diese Form des Imports wird in der Folge als „Drag and Drop“-Import bezeichnet. Zum anderen sollte der Benutzer die Informationen einer Prozessschablone in ein bereits bestehendes α-Doc integrieren können. Dieser Vorgang wird im weiteren Verlauf der Arbeit Merge-Import genannt.

5.2.4 Filterung der α-Adornments von α-Cards

Jede α-Card besteht aus einer Menge von α-Adornments. Beispiele sind der Name der α-Card (Anamnesebericht), die Rolle des Behandelnden (Gynäkologe) oder der konkrete Akteur (Dr. Müller). Wie bereits in 5.1.3 motiviert, soll der Arzt je nach Grad der Generalisierung eine andere Menge von α-Adornments in das α-Template aufnehmen können. Neben dem Namen der α-Card und den Informationen zu den Prozessteilnehmern enthalten α-Cards noch weitere Adornments, deren möglicher Export in 5.3.4.4 diskutiert wird. Beim Import kann wiederum entschieden werden, wie viele der vorhandenen Daten des α-Templates genutzt werden sollen. Diese Auswahl wird grundsätzlich für jede α-Card separat getroffen. Zur Erleichterung soll jedoch eine Standardauswahl definiert werden können, die bei ausgewählten α-Cards berücksichtigt wird.

5.2.5 Konzeption von Filtermechanismen für die Prozessartefakte

Die Notwendigkeit, die drei zentralen Systemartefakte (PSA, CRA und APA) filtern zu können, wird im folgendem Abschnitt aufgezeigt. Dabei wird stets auf den Spezialfall des α-Template-Imports in ein bereits vorhandenes α-Doc eingegangen.

Filterung des PSAs - Das PSA ist eine Liste aller α-Cards, also aller Dokumente, die im Rahmen einer α-Episode anfallen. Sowohl beim Import als auch beim Export darf der Benutzer entscheiden, welche α-Cards im α-Doc respektive im α-Template vorhanden sein sollen. Um diese Auswahl zu ermöglichen, muss ein Filter für das PSA realisiert werden.

Einen Sonderfall stellt der Import eines α-Templates in ein bereits vorhandenes α-Doc dar. Hier existiert sowohl das PSA des α-Docs als auch das des α-Templates. Im fertigen α-Doc dürfen keine duplizierten α-Cards vorliegen. Daher muss ein geeignetes Verschmelzungsverfahren entwickelt werden, das Duplikate erkennen und eliminieren kann.

Filterung des CRAs - Das CRA enthält die elektronischen Kontaktdaten aller Prozessteilnehmer. Auch hier gibt es bei Import und Export die Wahlmöglichkeit, welche Teilnehmer dem α-Template beziehungsweise dem α-Doc hinzugefügt werden sollen. Deshalb besteht wiederum die Notwendigkeit ein geeignetes Filter für das CRA zu entwerfen.

Analog zum PSA ist ein Spezialfall das Importieren eines α-Templates in ein vorhandenes α-Doc. Die selben Akteure können in beiden CRAs vorhanden sein und müssen demzufolge verschmolzen werden.

Filterung des APAs - Das APA besteht zunächst aus den prozessrelevanten α-Adornment-Typen, die in jedem α-Doc vorhanden sein müssen. Neben diesen „generischen“ kann das APA auch selbst definierte α-Adornment-Typen enthalten. Ob diese in das α-Template aufgenommen beziehungsweise ob diese aus dem α-Template in das neue α-Doc importiert werden, muss der Benutzer wiederum entscheiden können. Deshalb wird ein Filter für „nicht-generische“ α-Adornment-Typen benötigt. Ein Verschmelzungsmechanismus für gleichnamige α-Adornments im Rahmen des eingangs erwähnten Spezialfalls ist wiederum notwendig.

5.2.6 Sonstige Anpassungen

Identifikationsmöglichkeit für den α-Template-Dokumenttyp - Durch Ziehen eines Dokuments auf den Alph-O-Matic-Injector[1] mittels „Drag and Drop“ kann der Benutzer ein neues α-Doc erstellen. Ist dieses Dokument ein α-Template, wird man in Zukunft eine neue α-Episode auf Basis des α-Templates beginnen können. Um das Dokument als α-Template erkennen und von einer normalen Payload unterscheiden zu können, muss ein geeignetes Identifikationsverfahren konzipiert werden. Das Verfahren sollte dabei die α-Injector-Komponente ergänzen, da diese für die Umsetzung der „Drag and Drop“-Funktionalität zuständig ist.

1 Ein „Java Archive“ (JAR), das die Ausführungslogik von α-Flow enthält

5.3 Lösungskonzept

Der folgende Abschnitt stellt das Lösungskonzept von α-Templates vor. Die Beschreibung des Entwurfs erfolgt verteilt auf die Komponenten der typischen Drei-Schichten-Architektur.

5.3.1 Speicherungsform für α-Templates

Die Motivation, α-Templates abspeichern zu können, erfolgte in 5.2.1. Eine Methode hierfür wird im folgenden Abschnitt aufgezeigt.

In Abschnitt 3.1.4 wurde die JAXB-Bibliothek vorgestellt, welche im Rahmen von α-Flow zur XML-Datenbindung verwendet wird. Die Vorteile von XML als Speicherungsform liegen in der Strukturiertheit der Daten, der Plattformunabhängigkeit und der freien Nutzbarkeit des Standards. Da die α-Templates-Komponente eine Erweiterung von α-Flow ist, profitiert sie gleichermaßen von diesen Punkten. Wie in 5.3.4 beschrieben, arbeitet die α-Template-Komponente analog zum Restprojekt mit POJOs. Deshalb ist es sinnvoll auch die XML-Datenbindung von α-Templates mithilfe von JAXB zu realisieren. Abbildung 5.3 beschreibt die Verwendung von JAXB bei α-Templates.

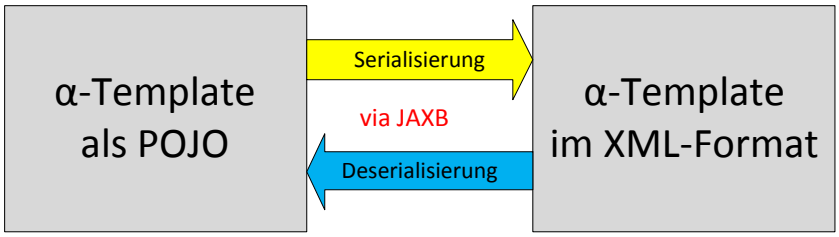


Bild 5.3: XML-Datenbindung eines α-Templates

5.3.1.1 Aufbau von α-Templates

In α-Templates müssen gemäß 5.2.1 alle Informationen enthalten sein, die für einen späteren Import benötigt werden. Um welche Daten es sich hierbei handelt, wird in diesem Abschnitt erklärt.

Die Informationen eines kompletten α-Docs umfassen zunächst alle Systemartefakte - das PSA, das CRA und das APA. Außerdem sind alle α-Cards mitsamt ihrer α-Descriptors und Payloads Informationsträger. Da Payloads (von Content-Cards) rein behandlungsspezifische Daten umfassen, dürfen diese nicht in einem α-Template gespeichert werden. Die restlichen genannten Datentypen, die drei Systemartefakte und die Grundmenge der

α-Card-Descriptors, können dagegen in „Process Templates“ aufgenommen werden und sind daher Teil der Template-Architektur.

α-Cards sind Sammlungen von α-Adornments, welche jeweils einzelne Informationen, wie den Namen der α-Card oder den Namen des Behandelnden, enthalten. Zur Redundanzminimierung werden nur diejenigen α-Adornments gespeichert, deren Wert vom Standardwert im exportierten APA abweicht. Der Ansatz, nur abweichende Werte bestimmter Datensätze zu speichern, wird Delta-Kodierung genannt [MDFK97].

Für alle Bausteine des α-Flow-Datenmodells existieren bereits Objektimplementierungen in Java. Für die α-Templates muss also lediglich ein neues Wurzelobjekt erstellt werden, das alle Systemartefakte sowie alle α-Card-Descriptors kapselt. Der Aufbau des Templates ist in 5.4 skizziert. Selbstverständlich obliegen die jeweiligen Objekte noch Filterungen, was jedoch nichts an der Gesamtarchitektur des α-Templates ändert.

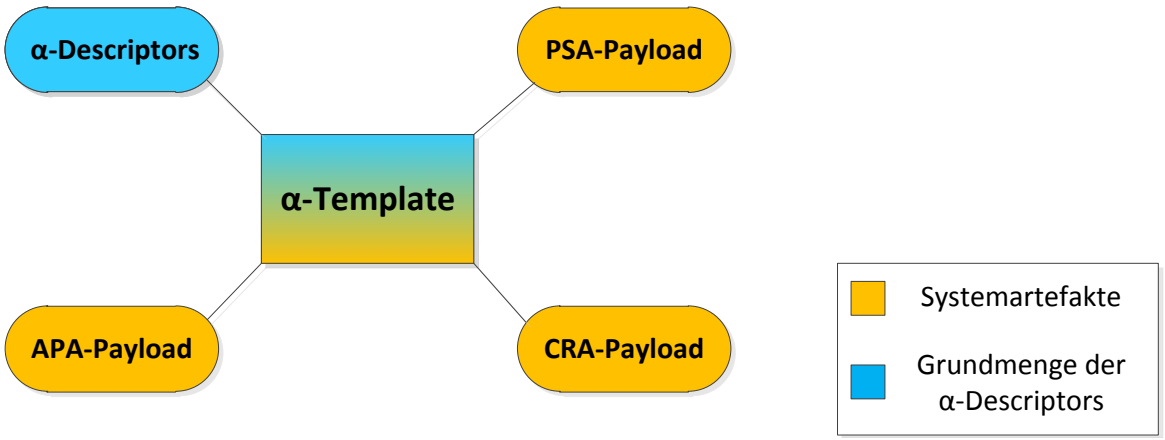


Bild 5.4: Architektur eines α-Templates

5.3.2 Validierung der α-Templates

In 5.2.6 wurde die Notwendigkeit begründet, α-Templates als solche identifizieren zu können. Der Anwendungsfall ist in Abbildung 5.5 dargestellt.

Man kann sowohl gewöhnliche Dokumente als auch Prozessschablonen mittels „Drag and Drop“ auf den Alph-O-Matic-Injector ziehen. Handelt es sich um keine XML-Datei, so kann sie gar keine Prozessschablone darstellen. Ist die Datei jedoch in XML gespeichert und repräsentiert ein α-Template, so ist der Importvorgang zu starten, welcher auf Basis der Prozessschablone ein neues α-Doc erzeugt. Ansonsten wird wie üblich lediglich ein neues α-Doc erstellt, bei dem das mittels „Drag and Drop“ übergebene Dokument die Payload der initialen α-Card bildet.

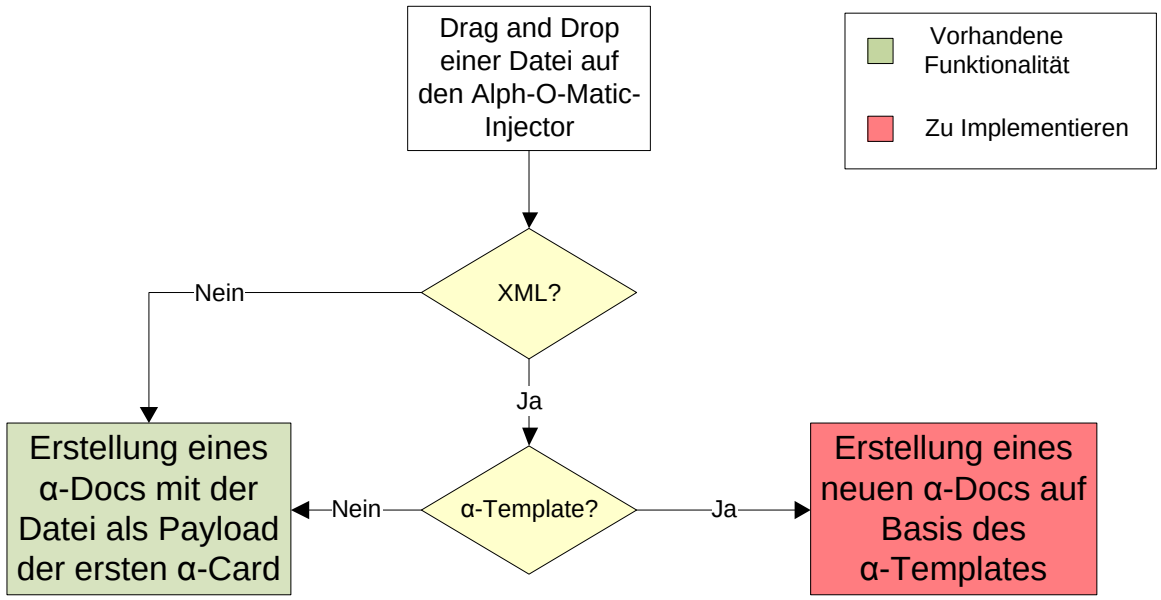


Bild 5.5: Neuer Ablauf der „Drag and Drop“-Injection

Um nun ein α-Template erkennen zu können, muss bei allen XML-Dateien eine Validierung stattfinden. Diese erfolgt mithilfe von JAXB, welches beim Versuch zu deserialisieren das α-Template von anderen Datentypen unterscheiden kann. Scheitert die Deserialisierung, so handelt es sich nicht um ein α-Template. Die Datei wird stattdessen wie eine gewöhnliche Payload behandelt. Gelingt die Deserialisierung, so ist zumindest die Korrektheit des Wurzelelements sichergestellt. Eine Erweiterung der Validierung wird in 7.2 diskutiert.

Alternativ könnte man das potentielle α-Template ohne Hilfe von JAXB manuell mit DOM oder SAX parsen. Anschließend würde der Name des Wurzelelements sowie möglicherweise weiterer Elemente abgeglichen werden. Neben dem komplizierteren Verarbeiten von XML müsste man hier jedoch die Namen der Elemente hart kodiert ablegen. Bei Validierung mittels JAXB wird hingegen automatisch das aktuelle Datenschema betrachtet.

5.3.3 Besondere Strategien für den Merge-Import

In 5.2.1 wurden bereits die speziellen Anforderungen an den Merge-Import beschrieben. Im Gegensatz zum „Drag and Drop“-Import und zum Export müssen dabei gleichartige Datenstrukturen zusammengeführt werden. Es existieren sowohl die Datensätze des

α-Templates als auch die des bestehenden α-Docs. Aus diesem Grund müssen geeignete Verschmelzungsverfahren entwickelt werden.

Der gewählte Lösungsansatz besteht darin, dass beim Merge-Import lediglich neue Informationen hinzugefügt werden. Ersetzungen finden hingegen zu keinem Zeitpunkt statt. Bestehen Konflikte, so wird stets die alte Version beibehalten. Die Erkennung der Konflikte basiert je nach Datentyp auf unterschiedlichen Faktoren.

Bei der Verschmelzung der PSAs werden hierzu die jeweiligen α-Card-IDs[1] verglichen. Ist der ID einer α-Card bereits im α-Doc vorhanden, darf diese nicht importiert werden. Diese Einschränkung ist sinnvoll, da es sich um die selben α-Cards handelt und die Version des α-Templates höchstwahrscheinlich älter ist. Änderungen an α-Cards des α-Docs werden grundsätzlich im Netzwerk bekannt gemacht. Ein α-Template bleibt dagegen auf dem Informationsstand des Exportzeitpunktes. In den meisten Fällen würden also durch die Ersetzung von α-Cards Änderungen verloren gehen. Der Nutzen von α-Templates besteht jedoch im Hinzufügen neuer Information.

Des Weiteren werden bei der Zusammenführung der CRAs die (eindeutigen) Namen der Behandelnden verglichen. Im Konfliktfall bleiben erneut die jeweils alten Informationen erhalten. Es ist wiederum davon auszugehen, dass bereits eingetragene Kontaktinformationen aktueller sind als die des α-Templates. Neben den Behandelnden kann das CRA der Schablone auch Patientendaten enthalten. Auf deren Import wird jedoch vollständig verzichtet, da beim Merge-Import die α-Episode bereits begonnen hat. Bei der Erstellung des α-Docs wurde demzufolge bereits der Name des Patienten bestimmt.

Beim Verschmelzen der APAs werden die Namen der eigenen α-Adornment-Typen verglichen. Sind zwei Namen identisch, so bleibt erneut die alte Version bestehen. In diesem Punkt wäre eine Auswahlmöglichkeit zwischen altem und neuem Adornment-Typ am ehesten denkbar. Es ist beispielsweise möglich, dass in einem Datentyp des α-Templates die Gesundheitszustände eines Patienten feingranularer definiert wurden. Trotzdem wird auf die Umsetzung dieser Idee zunächst verzichtet. Dies verringert den Import-Aufwand für den Benutzer und folgt der einheitlichen, eingangs erwähnten Grundstrategie auf Ersetzungen zu verzichten.

1 Bezeichner

5.3.4 Entwurf einer Filter-Pipeline

In diesem Abschnitt wird ein Lösungsansatz für die benötigten Filterungen (siehe 5.2.5) erarbeitet. Zunächst wird die Filterungsumgebung, anschließend die Reihenfolge der Schritte und das zugrundeliegende Konzept beschrieben.

5.3.4.1 Filterungsumgebung

Die Filterungen werden direkt an Java-Objekten (POJOs) und deren Datenstrukturen vorgenommen. Eine andere Möglichkeit könnte darin bestehen, beim Export die Java-Objekte zunächst mit JAXB in XML-Form zu überführen. Anschließend wäre der Prozess der Templatisierung mittels XSL Transformation (XSLT) durchzuführen. Analog dazu würden die XML-Transformationen beim Import vor der Deserialisierung der XML-Dateien stattfinden. Jedoch wird im Gesamtprojekt JAXB gerade deswegen verwendet, um den Einarbeitungsaufwand in XML zu minimieren (vgl. 3.1.4). Daher ist es sinnvoller, POJOs direkt zu filtern.

5.3.4.2 Reihenfolge der Filterungen

Für die verschiedenen Filterungen muss eine Reihenfolge bestimmt werden. Dabei existieren einige Abhängigkeiten, die berücksichtigt werden müssen. So sollten beispielsweise stets APA, PSA und CRA vor den α-Adornments-Mengen gefiltert werden. Täte man dies nicht, würde der Benutzer möglicherweise vor unnötigen Entscheidungen stehen. Bei zu später PSA-Filterung könnte gefragt werden, ob ein α-Adornment einer α-Card zu exportieren ist, wobei die α-Card selbst gar nicht wiederverwendet werden soll.

Ebenso müssen APA und CRA vor den einzelnen α-Adornments der α-Cards gefiltert werden, da diese beiden die Menge der exportierbaren α-Adornments einschränken. Wird zum Beispiel ein bestimmter α-Adornment-Typ gar nicht übernommen, sollte der Benutzer die jeweiligen α-Adornment-Instanzen der α-Cards auch nicht auswählen können.

Des Weiteren darf der Name eines bestimmten Arztes nicht in Form eines α-Adornments übernommen werden, wenn dieser Arzt an einer neuen Behandlungsepisode gar nicht mehr teilnimmt. Innerhalb der Artefakt-Filter ist die Reihenfolge schließlich beliebig.

5.3.4.3 Filterungskonzept

Für die Filterungen (siehe 5.2.5) wird nun das Ablaufkonzept vorgestellt. Es basiert auf dem Architekturmuster „Pipes And Filters“ [BMR+96]. Beim klassischen „Pipes And

Filters“ wird ein Datensatz in einer Reihe von Filterungsschritten bearbeitet. Die Ausgabe des vorigen Filters ist dabei jeweils die Eingabe des nächsten. Das Architekturmuster wird auch bei vielen weiteren Problemstellungen angewendet. Ein Beispiel hierfür wäre das Übersetzen von Programmcode, wobei Parsen, Analysieren, Optimieren ausgewählte Filterungsschritte darstellen.

Im Fall der vorgestellten α-Templates handelt es sich prinzipiell um verschiedene Datenstrukturen, die in den jeweiligen Schritten gefiltert werden. Der Begriff Pipeline trifft die Funktionsweise also nicht exakt.

Dennoch kann man die Menge aller vorhandenen Informationen als einen einzelnen, großen Datensatz verstehen, der die Filter-Pipeline durchläuft. Beim Exportvorgang wäre dies vergleichbar mit einem Klon des α-Docs, der schrittweise gefiltert in ein α-Template transformiert wird. Analog dazu würde beim „Drag and Drop“-Import die Gesamtinformation aus der Prozessschablone bestehen. Beim Merge-Import käme ein Klon des bereits existierenden α-Docs, in welches die Prozessschablone importiert wird, hinzu. Das Architekturmuster „Pipes And Filters“ dient also in jedem Fall als geeignete Grundlage. Abbildung 5.6 beschreibt den konzipierten Ablauf der einzelnen Filterungsschritte. Dabei sind die in Abschnitt 5.3.4.2 beschriebenen Abhängigkeiten berücksichtigt.

Es gilt dabei weiterhin zu beachten, dass die Filterungsschritte zweigeteilt ablaufen. Zunächst wird die jeweilige Filtermaske nach Benutzerwunsch erstellt. Anschließend wird diese direkt auf die korrespondierende Datenstruktur angewendet. Am Beispiel des PSAs erläutert Zeichnung 5.7 diesen Umstand. Aufgrund der direkten Anwendung kann der gesamte Filterungsprozess nicht durch einen atomaren Umschaltvorgang nach Abschluss der Benutzereingaben umgesetzt werden. Daher ist die Wiederholbarkeit von einzelnen vorherigen Schritten nicht auf triviale Weise umsetzbar. Dafür müssten partielle Rollback[1]-Mechanismen implementiert werden, die den Stand von Datenstrukturen vor bestimmten Filterungsschritten wiederherstellen können. Eine mögliche spätere Umsetzung wird in Abschnitt 7.3 diskutiert.

1 Zurücksetzen von Verarbeitungsschritten

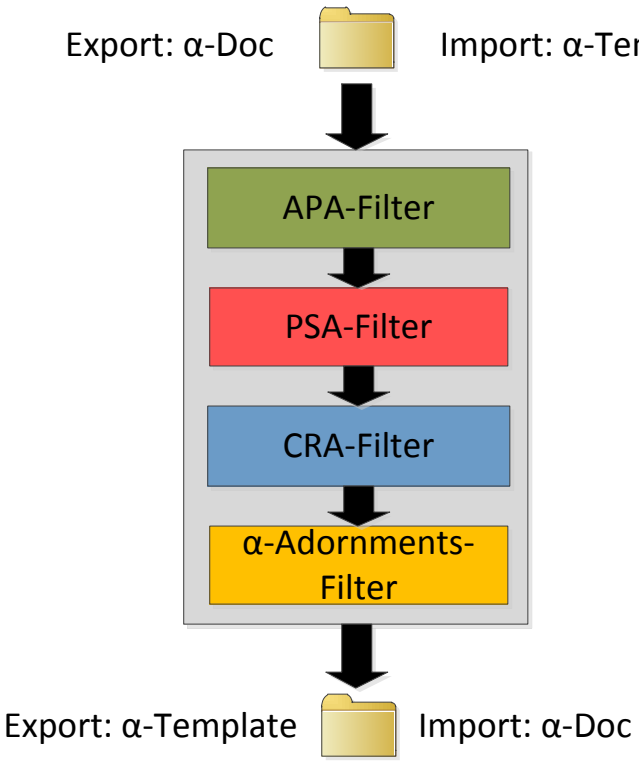


Bild 5.6: Der Filterungsablauf

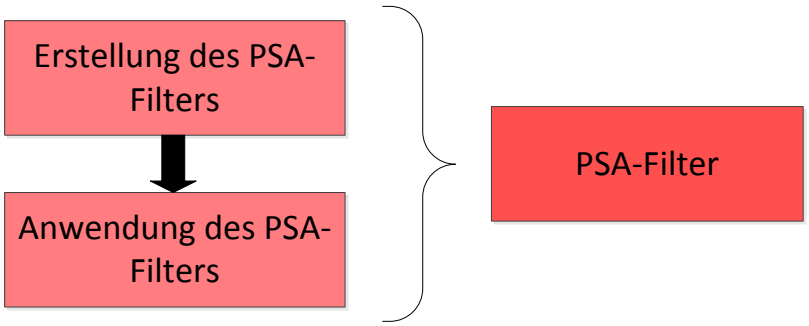


Bild 5.7: Die zwei Teile des PSA-Filterungsprozesses

5.3.4.4 Das α-Adornments-Filter

Den aufwendigsten Schritt stellt das α-Adornments-Filter dar, welcher deshalb in der Folge differenziert dargestellt wird. Hier muss für jede α-Card die Menge an α-Adornments einzeln ausgewählt werden. Um den Vorgang zu erleichtern und die Fehleranfälligkeit zu senken, kann zu Beginn eine Standardauswahl getroffen werden. Danach wird bestimmt, für welche α-Cards die Standardauswahl gilt, beziehungsweise welche separat behandelt

werden sollen. Beim Exportvorgang stellt sich außerdem die Frage, welche α-Adornments der Benutzer überhaupt exportieren darf. In Abschnitt 3.1.2 wurde die Menge der Standard-Adornments vorgestellt. Es gibt dort sowohl α-Adornments, die fundamentale Prozessrelevanz besitzen, als auch Adornments mit keinerlei Wiederverwendungswert. Erstere müssen exportiert werden, wohingegen letztere nicht übernommen werden dürfen. Bei allen anderen α-Adornments entscheidet der Benutzer, ob sie exportiert werden.

Die hierfür konzipierte Einteilung findet sich in Abbildung 5.8. Der Name der α-Card (AlphaCard Title) muss immer exportiert werden, da jede α-Card einen eigenen Prozessschritt darstellt und der Name zur Unterscheidung unerlässlich ist. Des Weiteren bleibt die Art der α-Card (AlphaCard Type) immer gleich. Ein Röntgenbericht ist beispielsweise in jeder α-Episode ein Untersuchungsergebnis und kein Überweisungsschein. Ebenso müssen mithilfe des semantischen Typs der α-Card (Fund. Semantic Type) Content- von Coordination-Cards unterschieden werden können.

Nie exportiert werden hingegen diejenigen Adornments, die rein behandlungsspezifische Daten enthalten. Dies umfasst die Sichtbarkeit (Visibility), Gültigkeit (Validity), Versionsnummer (Version), Variante (Variant), den Datentyp der Payload (Syn. Payload Type) und den Fälligkeitstermin (Due Date) der α-Card.

Alle anderen α-Adornments können exportiert werden, falls dies vom Benutzer so gewünscht wird. Je nach Grad der Abstraktion (vergleiche Abschnitt 5.1.3) können Name (Actor ID) sowie Rolle des Behandelnden (Role ID), der Name der Institution (Institution ID) und der des Patienten (OC ID) in die Prozessschablone übernommen werden. Ebenso besteht die Möglichkeit bei exportierten α-Cards die Attribute Gelöscht (Deleted), Verschoben (Deferred) sowie die Priorität (Priority) und den Status der Versionskontrolle (Version Control) zu speichern. Dies betrifft außerdem alle eigenen Adornment-Typen, da diese erst zur Laufzeit bekannt und damit nicht klassifizierbar sind.

Werden immer exportiert	Benutzer entscheidet über Export	Werden nie exportiert
Name der α-Card	Name des Behandelnden	Sichtbarkeit
Art der α-Card (Alpha-Card Type)	Rolle des Behandelnden	Gültigkeit
Semantischer Typ der α-Card (Fund. Semantic Type)	Name der Institution	Versionsnummer
	Name des Patienten	Variante
	Gelöscht	Datentyp der Payload
	Verschoben	Fälligkeitstermin
	Priorität	
	Versionskontrolle	
	Alle eigenen Adornments	

Bild 5.8: Klassifikation der α-Adornments

5.3.5 Erstellung einer Benutzeroberfläche auf Basis von Wizards

Im folgenden Abschnitt wird eine graphische Benutzerschnittstelle für die Erstellung von α-Templates entworfen. Beim Import-, wie auch beim Exportvorgang, können die Auswahlmöglichkeiten von vorher getroffenen Entscheidungen abhängen (vergleiche Abschnitt 5.3.4.2). Beispielsweise muss man die α-Adornments einer α-Card nicht filtern, falls die α-Card gar nicht importiert beziehungsweise exportiert werden soll. Deshalb ist es nicht möglich, den jeweiligen Vorgang in einem einzigen statischen Fenster durchzuführen. Die Oberfläche wird stattdessen als Wizard[1] gestaltet, in welchem der Benutzer eine Reihe von Dialogen abarbeitet. Eine Dialogführung ist gut geeignet, da die Benutzer bei dem seltenen Prozess des Imports und Exports von Prozessschablonen unterstützt werden. Das Wizard-Konzept ist beispielsweise bei Installationsmechanismen weit verbreitet. Alternativen wären die Einbettung der Auswahlmöglichkeiten in den normalen α-Editor sowie die Dialogführung via Popups. Was gegen diese Ansätze und für Wizard-Fenster spricht, wird im Folgenden genauer beschrieben.

Gegen Popups wurde sich aus ästhetischen sowie aus ergonomischen Gründen entschieden. Es ist optisch ansprechender, den Prozess in einem einzigen Fenster stattfinden

1 Assistent

zu lassen. Der Wizard hat eine feste Größe und eine einheitliche Struktur. Durch die „Weiter“-Buttons wird die Abfolge und die Zusammengehörigkeit der Bearbeitungsschritte betont. Klassische Popup-Fenster werden hingegen durch Klicken von „OK“ oder „Abbrechen“ einzeln verlassen.

Man könnte die Dialoge nun stattdessen in das Hauptfenster des α-Editors einbinden. Jedoch spiegelt ein separater Wizard das Aufsetzen der α-Templates-Komponente auf das grundlegende System besser wider. Das Importieren/Exportieren von α-Templates wird nur selten ausgeführt und ist nicht Teil des alltäglichen Behandlungsprozesses. Durch Button-Klick im laufenden α-Editor wird ein separater Vorgang mit klarem Anfang und Ende initiiert. Beim Erstellen eines neuen α-Docs mithilfe eines α-Templates ist außerdem noch kein geöffneter α-Editor vorhanden. Deshalb besteht in diesem Fall die Notwendigkeit ein neues Fenster zu öffnen. Durchgehende Verwendung von Wizard-Fenstern vereinheitlicht also die Benutzung der α-Templates-Komponente. Es ist in der Praxis außerdem üblich, eine Dialogführung in einem eigenen, abgetrennten Fenster durchzuführen.

Der Entwurf für ein typisches Wizard-Fenster ist in 5.9 zu sehen. Er zeigt den Aufbau des Dialogs während einer der Auswahlentscheidungen, die bei den Filterungsprozessen auftreten. Beispielsweise muss er in einem Schritt die Menge der zu exportierenden α-Cards wählen. Dann kann er in diesem Fall die gewünschten α-Cards, welche unter der Entscheidungsfrage aufgelistet sind, markieren. Es ist keine aufwendigere Filterkonfiguration, beispielsweise durch logische Ausdrücke, notwendig. Abschließend bestätigt der Benutzer die Auswahl durch Klicken des „Weiter“-Buttons oder er beendet den Prozess mithilfe von „Abbrechen“. Ein „Zurück“-Button wurde im Rahmen der α-Templates-Komponente aufgrund der in 5.3.4.3 festgestellten Nicht-Atomarität des Gesamtfilterungsprozesses nicht implementiert. Zur Wiederholung von vorigen Filterungsschritten würde die Notwendigkeit bestehen, nicht-triviale Rollback-Funktionalitäten umzusetzen. Für die Diskussion einer späteren Implementierung sei wiederum auf 7.3 verwiesen.

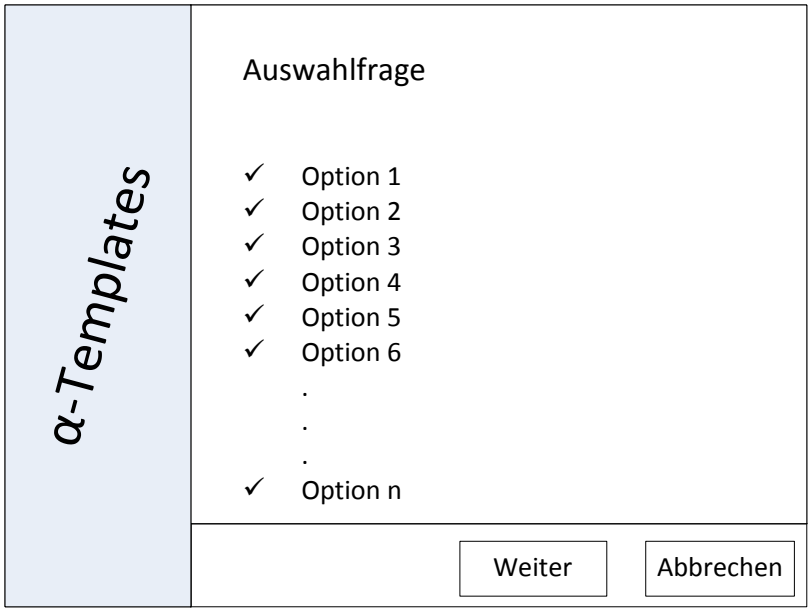


Bild 5.9: Entwurf eines Wizard-Fensters

5.3.5.1 Die Reihe der Wizard-Dialogfenster

In vorigem Abschnitt wurde sich auf das Konzept einer Wizard-Fensterreihe festgelegt. Die einzelnen Fenster und deren Abfolge werden in der Folge genauer vorgestellt.

Den Rahmen bilden ein Begrüßungsfenster und ein Fenster zur Präsentation des Ergebnisses. Das Begrüßungsfenster markiert stets den Beginn, das Ergebnisfenster das Ende des Import- beziehungsweise Exportvorgangs. Falls der Vorgang erfolgreich abgeschlossen wurde, wird ein positives Ergebnis präsentiert. Bricht der Benutzer hingegen den Prozess ab oder tritt ein Fehler auf, so wird zum nächstmöglichen Zeitpunkt das Scheitern des Vorgangs gemeldet.

Nach dem Begrüßungsbildschirm folgt ein Fenster zur Bestimmung des Speicherorts des zu importierenden respektive des neu zu exportierenden α-Templates. Eine Ausnahme bildet hierbei der „Drag and Drop“-Import, bei dem der Pfad des α-Templates nicht erneut festgestellt werden muss. Für die Erstellung der drei Artefakt-Filter (APA-, PSA- und CRA-Filter) stehen danach jeweils einzelne Dialogfenster zur Verfügung. Deren typisches Design wurde bereits mithilfe von Abbildung 5.9 gezeigt.

Falls α-Cards zum Import oder Export gewählt wurden, folgen nun die Wizard-Fenster des α-Adornment-Filters (vergleiche Abschnitt 5.3.4.4). Im ersten Dialog kann sich der Benutzer eine Standardmenge an α-Adornments definieren. Diese Standardmenge kann im zweiten Fenster für die im PSA-Filter gewählten α-Cards verwendet werden. Für die

restlichen α-Cards werden bei Bedarf weitere einzelne Fenster zur jeweiligen Bestimmung der α-Adornment-Menge präsentiert.

Die skizzierte Dialogreihe wird in Abbildung 5.10 veranschaulicht. Die schwarzen Pfeile kennzeichnen dabei den „Happy Path“[1]. Bei den roten Pfeilen kann es sich um Fehlerfälle beziehungsweise um einen möglichen Abbruch durch den Benutzer handeln. Die Gruppierung mit gestricheltem Rand stellt die Reihe der Dialogfenster des α-Adornments-Filters dar.

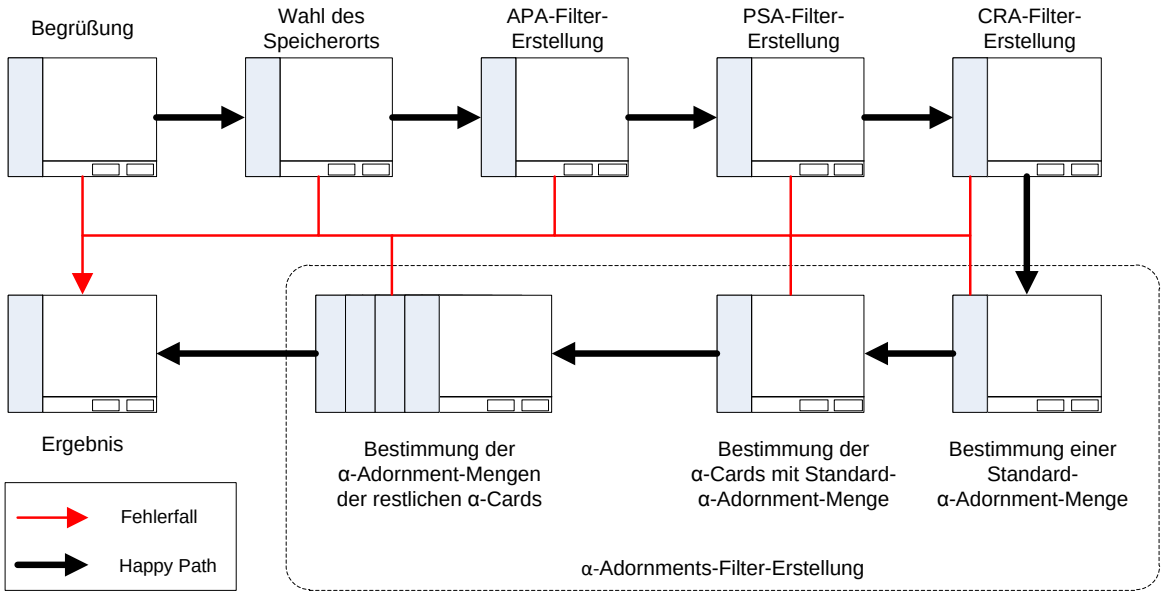


Bild 5.10: Die Wizard-Dialogreihe beim Import-/Exportvorgang

5.4 Zusammenfassung

Zu Beginn des Kapitels wurde in 5.1 das Fachkonzept der α-Templates-Komponente vorgestellt. Bei den Vorteilen der Benutzung von α-Templates wurde vor allem die Reduktion von Kosten, Zeit und Aufwand aufgeführt. Im anschließenden Anwendungsszenario wurde ein typisches Beispiel eines Import-/Exportvorgangs beschrieben und es wurden die Komponenten des α-Flow-Datenmodells in derem Kontext eingeführt. Anschließend wurden Strategien zur sinnvollen Verwendung und Erstellung von α-Templates vorgestellt. Dabei handelt es sich um generalisierte, stets wiederverwendbare Schablonen sowie um solche, die Informationen zu ihrer speziellen Behandlungsumgebung tragen.

1 normale, erwünschte Dialog-Ablaufsteuerung

In 5.2 wurden in der Folge die Anforderungen aus den Anwendungsszenarien extrahiert. Es wurde die Notwendigkeit von Änderungen des Datenmodells und der Benutzeroberfläche sowie der Konzeption von Filtermechanismen beschrieben.

Zum Abschluss folgte in 5.3 die Vorstellung eines auf den Anforderungen basierenden Lösungskonzepts für die α-Templates-Komponente. Zunächst wurde der Aufbau und die Speicherungsweise der α-Templates erarbeitet. Diese bestehen aus den Prozessartefakten des α-Docs sowie der Grundmenge der α-Card-Descriptors und werden mittels JAXB (de-)serialisiert. Als nächstes wurde die Notwendigkeit einer Validierung von α-Templates aufgeführt, um dem Benutzer einen „Drag and Drop“-Import von Prozessschablonen zu ermöglichen. Danach folgte die Konzeption einer Filterungspipeline, welche auf dem Entwurfsmuster „Pipes and Filters“ basiert. Dabei wurde der Ablauf der einzelnen Filterungsschritte beschrieben und POJOs als die zu filternde Datenform gewählt. Als letztes wurde das Konzept der Wizard-Fensterreihe motiviert und vorgestellt. Im Gegensatz zu Popup-Fenstern sind diese benutzerfreundlicher und optisch ansprechender. Außerdem wurden in diesem Abschnitt Beschreibungen von Aufbau, Zuständigkeiten und Abfolge der einzelnen Wizard-Fenster angebracht.

6 Prototypische Umsetzung

In diesem Kapitel wird die Prototypische Umsetzung der α-Templates-Komponente vorgestellt. Zu Beginn wird in 6.1 das äußere Verhalten der Komponente dargestellt. Dabei werden die Interaktionen mit den weiteren Subsystemen von α-Flow und damit die Einbettung der α-Templates-Komponente in den Gesamtkontext beschrieben. Danach folgt in 6.2 der eigentliche Software-Entwurf. Darin wird der technische Aufbau eines α-Templates sowie die Gestaltung der Wizard-Oberfläche und dessen einzelner Fenster vorgestellt. Die weiteren Teile des Entwurfs umfassen die Realisierung der erarbeiteten Filter und der Basisklassen, die die Import- und Exportvorgänge steuern.

6.1 Äußeres Verhalten der α-Templates-Komponente

Der folgende Abschnitt gibt einen ersten Überblick über den Aufbau der α-Templates-Komponente. Es wird vor allem dessen Zusammenspiel mit den restlichen Subsystemen von α-Flow beschrieben. Hierzu werden erstmals die drei Hauptklassen der Import-/Exportvorgänge eingeführt. Deren interne Verhaltensweisen werden jedoch erst in den folgenden Abschnitten vertieft.

Die Beziehungen zu anderen Klassen von α-Flow sind mithilfe von CRC-Karten [BC89] sowie einfachen Sequenzdiagrammen dargestellt. Eine CRC-Karte enthält als Überschrift den Namen der zu beschreibenden Klasse, auf der linken Seite dessen Aufgaben und rechts die Namen der Klassen, mit denen sie hauptsächlich interagiert. Abbildung 6.1 zeigt den Aufbau einer solchen Karte.

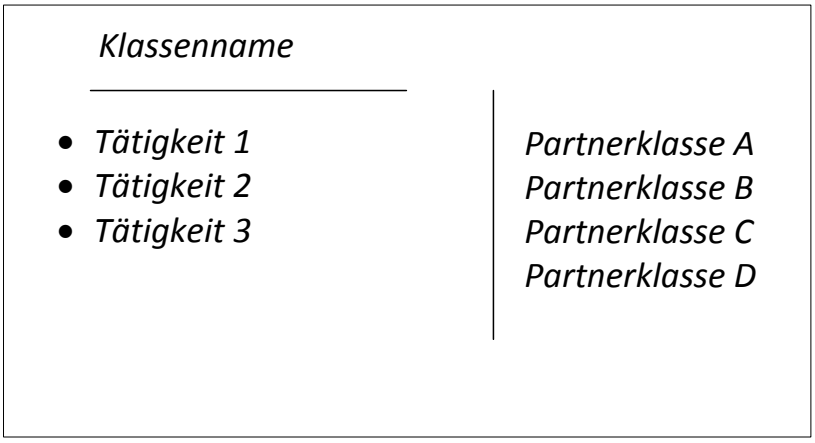


Bild 6.1: Prototypischer Aufbau einer CRC-Karte

6.1.1 Merge-Import

Den äußeren Rahmen des Merge-Imports bildet die Klasse *ImportHandler*. Die CRC-Karte des *ImportHandler*s findet sich in Abbildung 6.2.

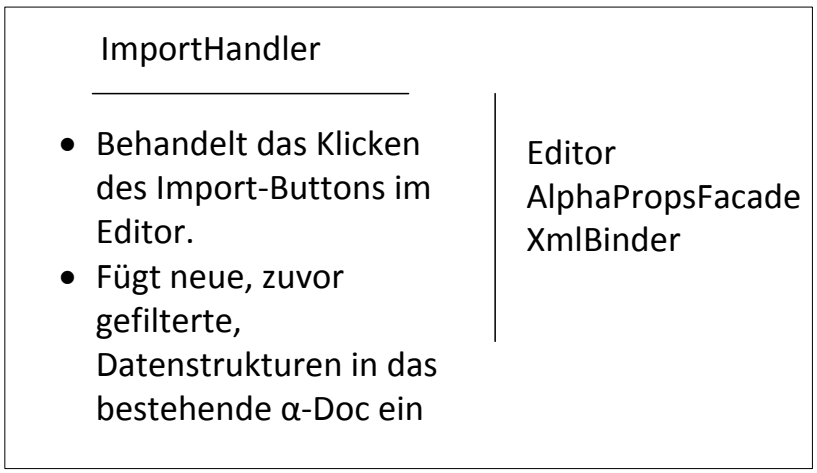


Bild 6.2: Verantwortlichkeiten des *ImportHandler*s

Der *ImportHandler* wird durch das Klicken des Import-Buttons im α-Editor erstellt und initiiert. Zunächst kann der *ImportHandler* die bestehenden Datenstrukturen des α-Docs mithilfe der *AlphaPropsFacade* einlesen. Danach werden unter Zuhilfenahme des *XmlBinder*s aus dem α-Utils-Modul die Datenstrukturen des „Process Templates“ deserialisiert. Die eingelesenen Daten werden nun unter Verwendung der verschiedenen Filterklassen zusammengeführt. Eine genauere Beschreibung dieser Klassen und des inneren Verhaltens des *ImportHandler*s folgt im Lauf dieses Kapitels. Abschließend

müssen die neuen Informationen noch mithilfe der *AlphaPropsFacade* in das α-Doc eingefügt werden. Die *AlphaPropsFacade* kapselt das Abspeichern der neuen α-Cards, das Hinzufügen neuer Prozessteilnehmer und die Bekanntmachung neuer α-Adornment-Typen. In diesem Punkt verläuft der Import intern analog zum normalen Hinzufügen von Informationen über den α-Editor. Im Sequenzdiagramm 6.3 sind die beschriebenen wichtigsten Interaktionen mit anderen α-Flow-Komponenten illustriert.

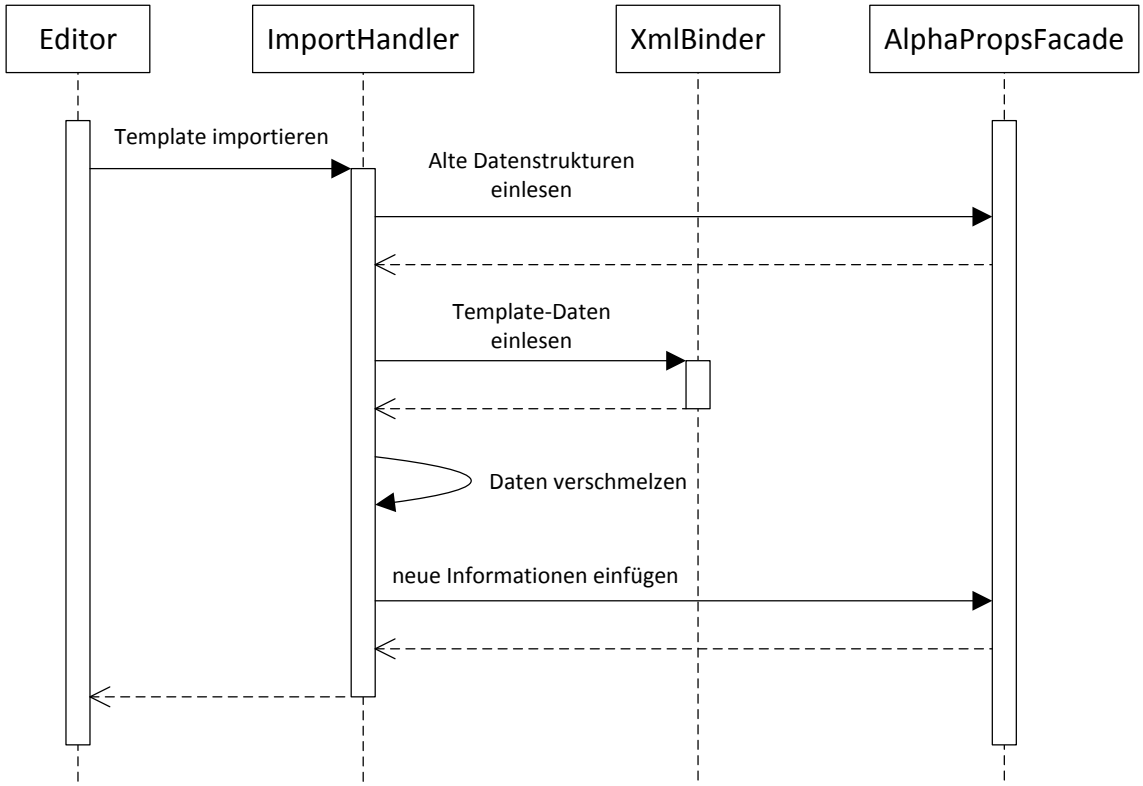


Bild 6.3: Skizze der Interaktion des *ImportHandler*s mit anderen α-Flow-Komponenten

6.1.2 „Drag and Drop"-Import

Beim „Drag and Drop"-Import wird ein neues α-Doc auf Basis eines α-Templates erstellt. Diese Form des Imports wird durch das Ziehen eines „Process Templates" auf den Alph-O-Matic-Injector[1] initiiert. Die Hauptklasse bildet dabei der *TemplateInjector*. Abbildung 6.4 stellt eine CRC-Karte dieser Klasse dar.

1 Ein „Java Archive" (JAR), das die Ausführungslogik von α-Flow enthält

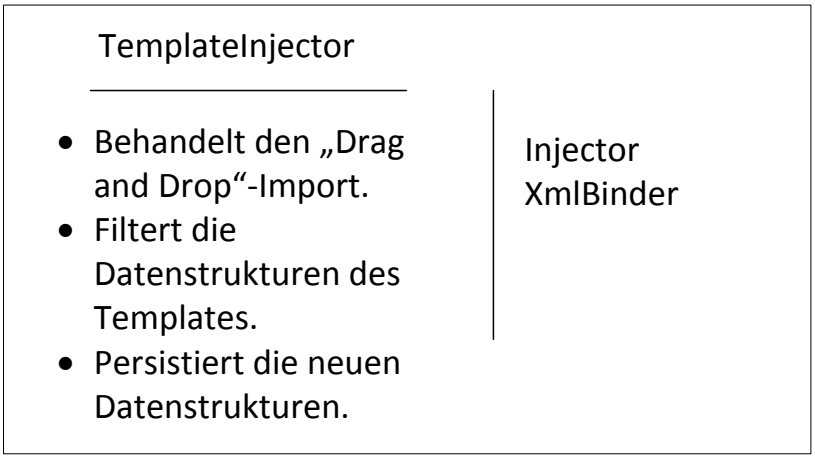


Bild 6.4: Verantwortlichkeiten des *TemplateInjector*s

Nachdem ein Dokument auf den Alph-O-Matic-Injector gezogen wurde, wird der *Injector* gestartet und beginnt damit ein neues α-Doc zu erstellen. Erkennt der *Injector* das Dokument als α-Template, erstellt er einen neuen *TemplateInjector* und überträgt diesem seine Aufgabe. Die Daten des Templates werden nun mithilfe der verschiedenen Filterklassen auf den gewünschten Umfang reduziert. Zum Schluss müssen diese Daten, analog zur normalen Erstellung eines α-Docs, persistiert werden. Da zu diesem Zeitpunkt keine *AlphaPropsFacade* existiert, muss dies manuell durchgeführt werden. Wie bei der (De-)Serialisierung von α-Templates wird dafür der *XmlBinder* benutzt. Die Interaktion mit anderen Klassen von α-Flow ist in Bild 6.5 visualisiert.

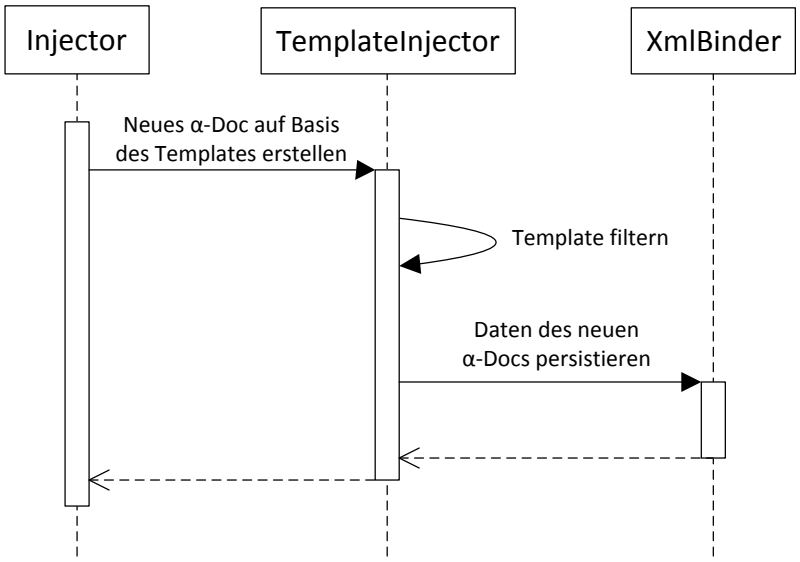


Bild 6.5: Skizze der Interaktion des *TemplateInjector*s mit anderen α-Flow-Komponenten

6.1.3 Exportfunktion

Die Exportfunktion generiert aus den bestehenden Informationen des geöffneten α-Docs ein „Process Template“. Klickt der Benutzer auf den Export-Button, wird ein neuer *ExportHandler* erstellt. Dieser koordiniert in der Folge den Exportvorgang. Die Verantwortlichkeiten und Interaktionspartner des *ExportHandler*s finden sich in Abbildung 6.6.

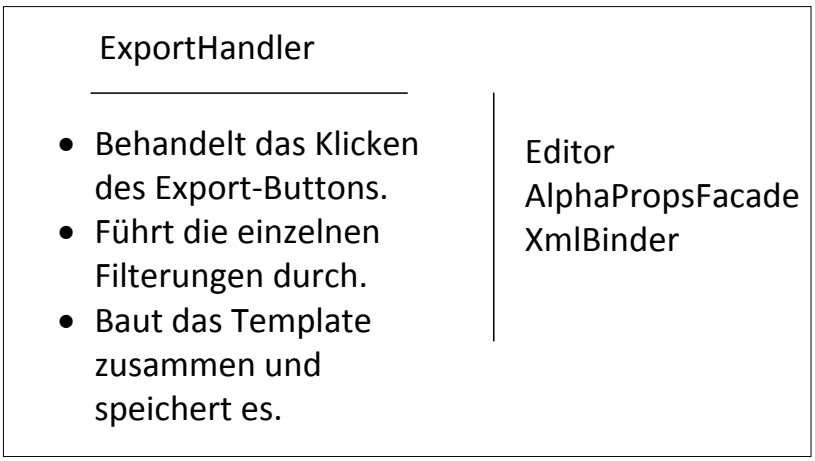


Bild 6.6: Verantwortlichkeiten des *ExportHandler*s

Zunächst werden die bestehenden Daten des α-Docs unter Zuhilfenahme der *AlphaPropsFacade* abgefragt. Kopien dieser Datenstrukturen werden anschließend intern gefiltert. Der Benutzer entscheidet dabei, wie umfangreich die Prozessschablone werden soll. Sind die Filterungen abgeschlossen, wird ein neues α-Template erstellt und mit der gefilterten Datenmenge gefüllt. Der *XmlBinder* sorgt anschließend für die Speicherung des Templates im Dateisystem. Abbildung 6.7 veranschaulicht die Interaktion des *ExportHandler*s mit weiteren α-Flow-Subsystemen.

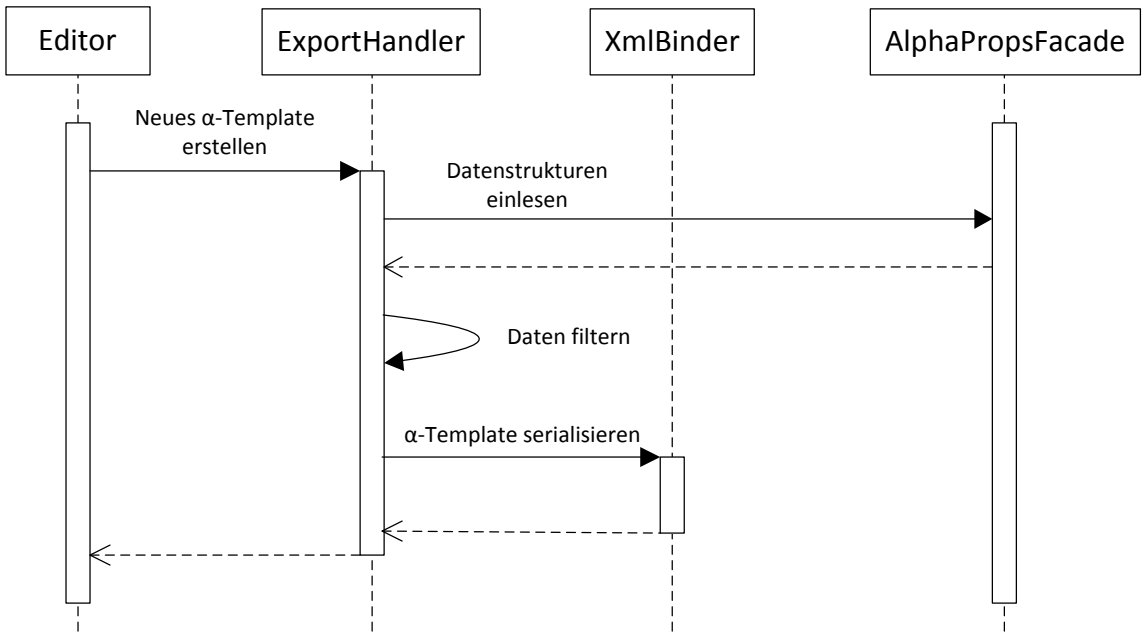


Bild 6.7: Skizze der Interaktion des *ExportHandler*s mit anderen α-Flow-Komponenten

6.2 Software-Entwurf

Der nächste Abschnitt beschreibt den Software-Entwurf der α-Templates-Komponente. Dabei werden der Datentyp von α-Templates und die Umsetzung des Wizards sowie dessen Fenster vorgestellt. Außerdem folgen die Entwürfe der einzelnen Filterklassen sowie der Basisklassen, die den Rahmen der Import- und Exportvorgänge bilden.

6.2.1 Technischer Aufbau von α-Templates

In Abbildung 6.8 wird die Klasse *AlphaTemplate* modelliert, welche die erarbeitete α-Template-Architektur umsetzt. Wie in 5.3.1.1 beschrieben, benötigt der Benutzer bei späterem Import unter anderem die drei Systemartefakte, die *PSAPayload*, die *APAPayload* und die *CRAPayload*. Des Weiteren halten die „Process Templates“ die zusätzlich notwendigen α-Card-Descriptors in Form einer Liste bereit. Im referenzierten Abschnitt wurde außerdem erklärt, dass nur vom Standardwert abweichende α-Adornments in Prozessschablonen gespeichert werden. Damit der Benutzer stets alle α-Adornments zur Verfügung hat, müssen die fehlenden Daten beim Import wieder ergänzt werden können.

Bei einem der Standard-α-Adornment-Typen handelt es sich um den Namen des Behandelnden. Wird nun ein Arzt exportiert und später nicht importiert, muss der Wert

des Adornments bei allen betreffenden α-Cards geändert werden. Da die Behandelnden nur ihre eigenen α-Cards editieren können, wäre ansonsten eine Anpassung dieser α-Cards nie mehr möglich. Daher werden dem importierenden Benutzer die jeweiligen α-Cards zugeteilt. Dieser kann nach dem Import wiederum andere Prozessteilnehmer als Behandelnde bestimmen.

Beide Funktionalitäten werden sowohl beim „Drag and Drop“-Import als auch beim Merge-Import benötigt. Daher bietet sich die von beiden genutzte *AlphaTemplate*-Klasse als Ort der Implementierung an. Die Methode *completeAdornments* realisiert schließlich die beschriebene Funktionalität. Nachdem alle fehlenden α-Adornments ergänzt wurden, wird geprüft, ob der Name des Behandelnden in der Teilnehmerliste vorhanden ist. Bei negativem Ergebnis wird er stattdessen durch den Namen des Benutzers ersetzt.

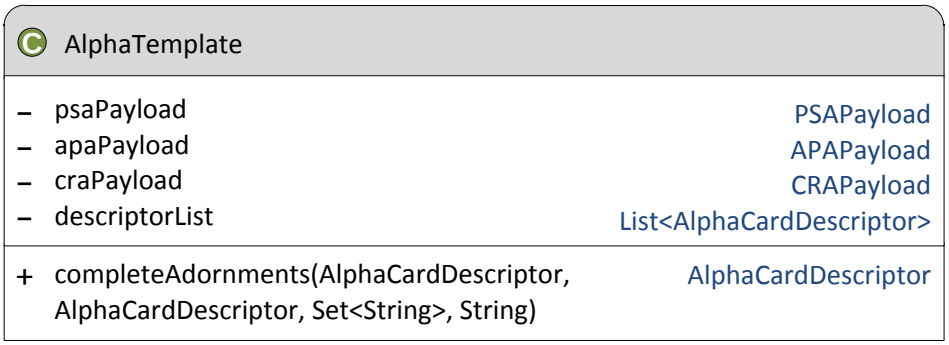


Bild 6.8: Technischer Aufbau eines α-Templates

6.2.2 Umsetzung der Wizard-Oberfläche

6.2.2.1 Aufbau der Wizard-Klasse

Der grundsätzliche Aufbau des Wizards wird durch die Klasse *TemplateWizard*, welche in Abbildung 6.9 modelliert ist, bestimmt. Das Oberflächenkonzept sowie ein prototypisches Design wurden bereits in 5.3.5 vorgestellt. Den äußeren Rahmen der Oberfläche bildet ein *JDialog*, welcher alle Komponenten des Wizards enthält. Die Wizard-Fenster sind in vier feste Teilbereiche eingeteilt, die jeweils durch ein *JPanel* repräsentiert werden. Dabei existieren Teilbereiche für Überschrift, Auswahlfelder, Buttons und eine simple Illustration zur Verschönerung der Oberfläche. Bei den Auswahlfeldern handelt es sich um *JCheckBox*-Objekte, die beispielsweise den Benutzer nach der Verwendung bestimmter α-Cards fragen. Das Button-Panel besteht aus einem „Weiter“- und einem „Abbrechen“-

Button. Auf einen „Zurück“-Button wurde aus Komplexitätsgründen zunächst verzichtet, wobei dessen mögliche spätere Implementierung in 7.3 diskutiert wird.

Das Überschrifts-Panel und das Panel der Auswahlfelder sind diejenigen Komponenten, die einem ständigen äußerlichen Wandel unterstehen. Da Überschriften außerdem unterschiedlich lang sein können, wäre es sinnvoll den Übergang zum Auswahl-Panel fließend zu gestalten. Daher werden die zwei erwähnten Panels zu einem gemeinsamen Haupt-Panel zusammengefasst. Die Überschrift des Wizards wird bei fast jedem Fenster angepasst. Deshalb muss sie als Feld des *TemplateWizard*s inklusive Setter-Methode realisiert werden. Gleiches gilt für die verwendeten Buttons, deren Reaktion auf Klicks im Laufe der Import-/Exportvorgänge angepasst werden muss. Neben den typischen Gettern und Settern enthält die Klasse lediglich eine Initialisierungsmethode. Darin werden die beschriebenen Panels erstellt und das Layout des Wizard-Fensters festgelegt.

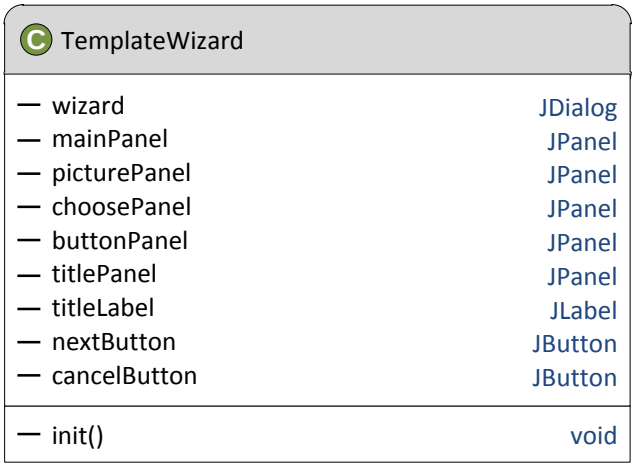


Bild 6.9: Aufbau des *TemplateWizard*s

6.2.2.2 Visualisierung der Auswahlfenster

Im vorigen Abschnitt wurde der technische Aufbau der Wizard-Fenster beschrieben. Zur Erstellung der jeweiligen Artefaktfilter muss der Wizard stets neu mit Auswahldaten gefüllt, visualisiert und ausgewertet werden. Aus dem Grund werden für die drei Artefakte Visualisierungsklassen erstellt. Diese implementieren dabei spezielle Schnittstellen, was der Entkopplung von Ausführungslogik und GUI[1] dient. Das Oberflächenkonzept kann somit auf einfache Art und Weise ausgetauscht werden, ohne alle Methodenaufrufe in der

1 graphischer Benutzeroberfläche

Logikschicht anpassen zu müssen. Das zugrundeliegende Entwurfsmuster wird Strategie genannt [GHJV95].

Da das innere Verhalten der Klassen sehr ähnlich ist, wird zunächst die Visualisierung der α-Card-Abfragen beispielhaft beschrieben. Anschließend werden die Besonderheiten der anderen Klassen hervorgehoben.

Visualisierung der α-Card-Abfragen

Die Visualisierung und anschließende Auswertung der α-Card-Abfragen erfolgt in der Klasse *WizardAlphaCardNamesPresenter*. Neben dem *TemplateWizard* wird eine *HashMap<AlphaCardDescriptor, String>*, welche später die α-Card-Filtermaske darstellt, gespeichert. Die Aufgaben werden auf drei verschiedene Methoden verteilt.

Die Hauptmethode *createAlphaCardMap* ruft zunächst *rebuildMainPanel* auf, was den Neuaufbau des Haupt-Panels initiiert. Es wird die Überschrift neu gesetzt sowie ein neues GUI-Layout bestimmt. Anschließend werden dem Wizard *JCheckBox*es, welche jeweils den Namen einer übergebenen α-Card visualisieren, hinzugefügt. Als nächstes werden in der Hauptmethode die *ActionListener*s für die jeweiligen Buttons hinzugefügt respektive erneuert. Nun wird der Wizard sichtbar gemacht, was aufgrund dessen standardmäßiger Modalität die Ausführung des Programms unterbricht. Sobald der Benutzer einen Button geklickt hat, wird der registrierte *ActionListener* ausgeführt. In jedem Fall wird der Wizard nun wieder unsichtbar gemacht, um die Weiterführung des Programms zu ermöglichen. Der „Weiter“-Button wird zusätzlich durch den Aufruf von *evaluateCheckBoxes* behandelt. In *evaluateCheckBoxes* wird für jede visualisierte α-Card die korrespondierende *JCheckBox* ausgewertet. Das Ergebnis wird in einer *HashMap<AlphaCardDescriptor, Boolean>* gespeichert und stellt die α-Card-Filtermaske dar.

Der Aufbau der Klasse ist in Bild 6.10 dargestellt. Die Interaktion mit den restlichen Komponenten ist mithilfe von 6.11 modelliert. Die rot dargestellten Schritte werden in *rebuildMainPanel* durchgeführt. Der blaue Teil erfolgt durch *createAlphaCardMap*, der grüne durch *evaluateCheckBoxes*.

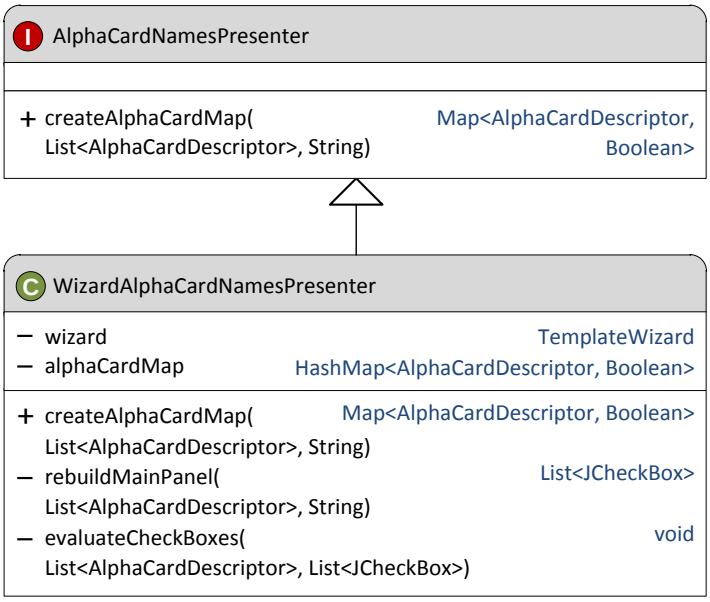


Bild 6.10: Aufbau des *WizardAlphaCardNamesPresenter*s

Visualisierung der α-Adornment-Abfragen

Die Visualisierung und Auswertung der α-Adornment-Abfragen wird durch die Klasse *WizardAdornmentNamesPresenter* gesteuert. Wie erwähnt gleicht dessen Aufbau und Verhalten stark dem *WizardAlphaCardNamesPresenter*. Der einzige wichtige Unterschied besteht darin, dass lediglich mit den Namen der α-Adornments in Form von Strings gearbeitet wird. Diese Möglichkeit hätte bei den α-Cards ebenso bestanden. Anders als bei den Adornments existiert für die α-Card-Descriptors jedoch schon eine Grundsammlung. Bei der Erstellung einer neuen α-Adornments-Sammlung in den Filtern kann sich zur Kopplungsminimierung auf deren Namen beschränkt werden. Die α-Adornments-Filtermaske wird folglich durch eine *HashMap<String, Boolean>* repräsentiert. Der Aufbau der Klasse findet sich in Bild 6.12.

Visualisierung der Prozessteilnehmer-Abfragen

Die Klasse *WizardContributorNamesPresenter* ist für die Präsentation und Auswertung der Prozessteilnehmer-Abfragen zuständig. Diese ähnelt wiederum stark den beiden vorigen Klassen, mit der zusätzlichen Herausforderung, zwei verschiedene Datentypen visualisieren zu müssen. Neben den Behandelnden muss bei Export und „Drag and Drop“-Import auch der Patient präsentiert werden können. Aus diesem Grund speichert die Klasse zusätzlich ein *ObjectUnderConsideration*.

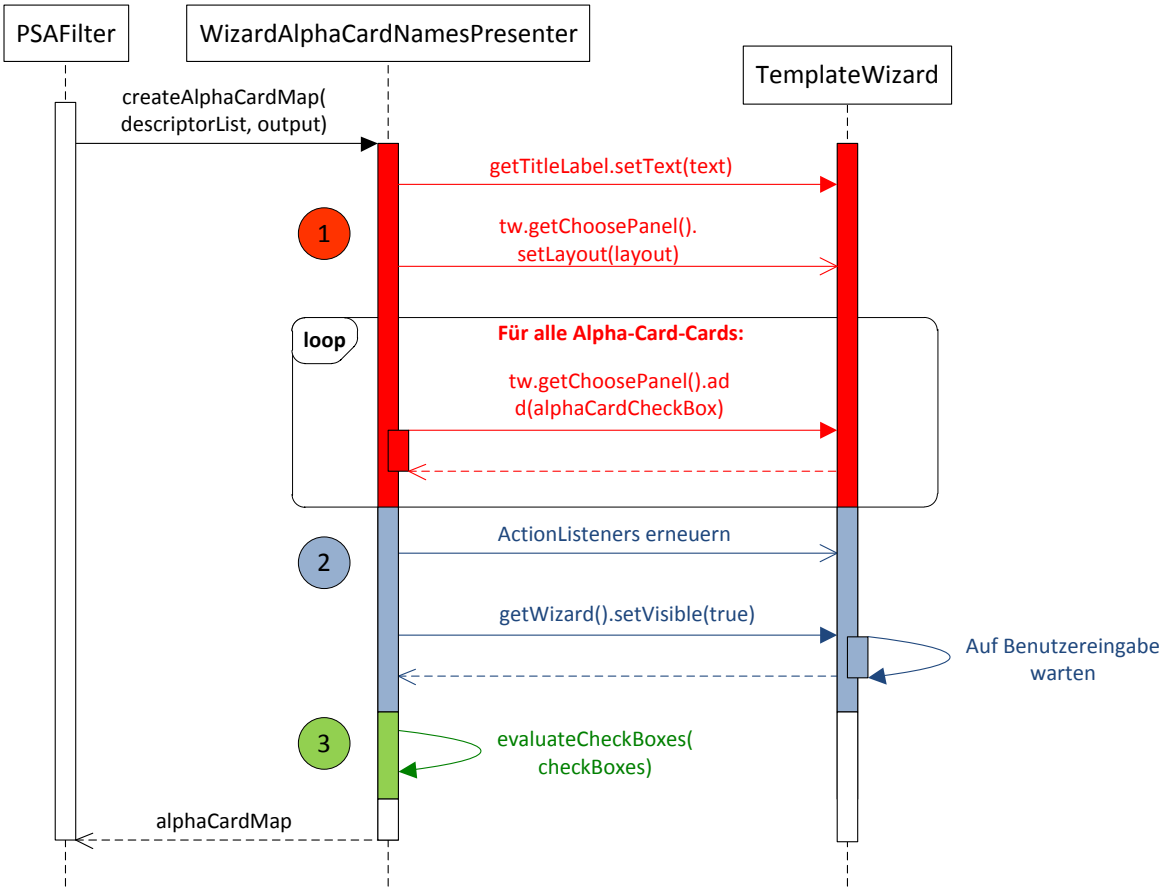


Bild 6.11: Verhalten des *WizardAlphaCardNamesPresenters*

In der *rebuildMainPanel*-Methode müssen Patient und Behandelnde visuell getrennt dargestellt werden. Hierzu werden vor den jeweiligen Teilnehmer-*JCheckBox*en *JLabel*s zum Auswahl-Panel hinzugefügt, welche als Unterüberschriften fungieren. Die Prozessteilnehmer-Filtermaske wird durch eine *HashMap<Participant, Boolean>* repräsentiert. Bei der Auswertung der Benutzereingaben wird das ObjectUnderConsideration nun als <null, Boolean>-Tupel gespeichert. Dies dient der einfachen Unterscheidung von anderen Prozessteilnehmern. Alternativ hätte man einen neuen Datentyp entwerfen können, der neben der beschriebenen *HashMap* das *Boolean* getrennt speichert. Abbildung 6.13 zeigt ein Diagramm der beschriebenen Klasse.

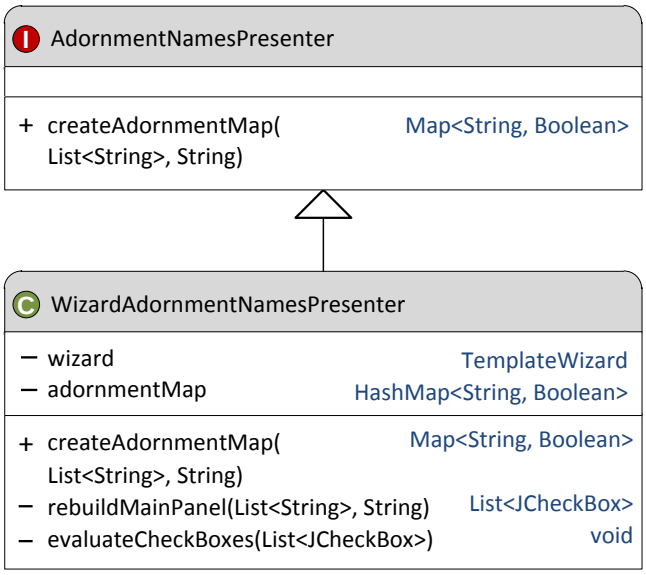


Bild 6.12: Aufbau des *WizardAdornmentNamesPresenter*s

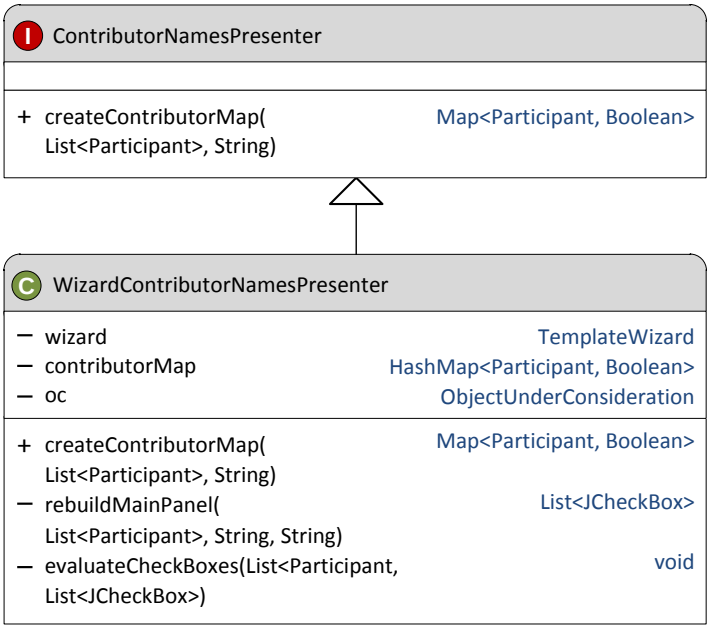


Bild 6.13: Aufbau des *WizardContributorNamesPresenter*s

6.2.2.3 Anzeige von Nachrichten im Wizard

Sowohl beim Begrüßungsfenster als auch bei der Benachrichtigung über das Ergebnis des Vorgangs müssen dem Benutzer kurze Nachrichten angezeigt werden. Diese Funktio-

nalität wird in der Klasse *WizardMessagePresenter* realisiert, welche die Schnittstelle *MessagePresenter* implementiert.

Neben dem stets benötigten *TemplateWizard*-Objekt wird im *WizardMessagePresenter* außerdem eine boolesche Variable, die die Beschriftung des „Weiter“-Buttons steuert, gespeichert. Beim Willkommensbildschirm sollte der Button mit „Start“, beim Abschlussfenster mit „Finish“ beschriftet sein. Die gesamte Visualisierung wird in der *print*-Methode umgesetzt. Dort wird das Haupt-Panel neu aufgebaut, die Überschrift gesetzt sowie die zu präsentierende Nachricht in Form eines *JLabel*s erstellt. Nun wird je nach Status der booleschen Variable der Text des „Weiter“-Buttons neu gesetzt und der Abbruch-Button deaktiviert. Zudem wird die Behandlung des „Weiter“-Buttons neu definiert. Dabei werden neben der Unsichtbarmachung des Wizards lediglich der Button-Text zurückgesetzt und der „Abbrechen“-Button reaktiviert. Nachdem der Wizard schließlich sichtbar gemacht wird, blockiert dieser wiederum solange, bis das Klicken des „Weiter“-Buttons die eben beschriebenen Aktionen auslöst. Ein Modell des *WizardMessagePresenter*s findet sich in Abbildung 6.14.

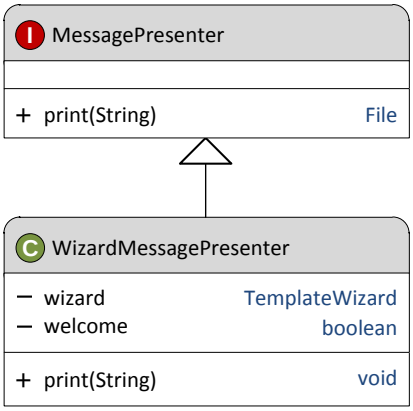


Bild 6.14: Aufbau des *WizardMessagePresenter*s

6.2.2.4 Auswahl eines Speicherorts innerhalb des Wizard-Fensters

Beim Export von α-Templates sowie beim Merge-Import darf der Benutzer einen Speicherort im System wählen. Während dies beim Exportvorgang der künftige Speicherplatz der Schablone ist, wählt man beim Merge-Import das bestehende α-Template im Dateisystem aus. Die Umsetzung erfolgt in der Klasse *WizardFileChooser*, die auf einer festgelegten Schnittstelle, dem *AlphaFileChooser*, basiert.

Neben dem Wizard und einer Referenz auf die gewählte Datei verwalten Objekte dieser Klasse wiederum eine Beschriftung des „Weiter“-Buttons. Falls ein bestehendes Template

gewählt wird, trägt der Button die Aufschrift „Open“. Bei der Abspeicherung eines neuen „Process Templates“ lautet die Beschriftung „Save“.

Das Wählen eines Speicherorts findet in der Methode *choose* statt. Zunächst wird ein *JFileChooser* erstellt, welcher es dem Benutzer erlaubt, einen Eintrag des Dateisystems visuell auszuwählen. Danach wird das Haupt-Panel neu aufgebaut sowie Überschrift und Beschriftung des „Weiter“-Buttons gesetzt. Weiterhin wird der eben erstellte *JFileChooser* in das Auswahl-Panel eingebettet. Anschließend müssen die Behandlungsmethoden für beide Buttons erneuert werden. In beiden Fällen wird der Wizard zur Programmfortsetzung unsichtbar gemacht und der Text des „Weiter“-Buttons auf den Standardwert zurückgesetzt. Durch Klicken des „Weiter“-Buttons wird zusätzlich die gerade im *JFileChooser* ausgewählte Datei gespeichert. Das Ende der Methode markiert erneut das Sichtbarmachen des Wizards, was die Fortführung des Vorgangs bis zum nächsten Button-Klick anhält. In Bild 6.15 ist der beschriebene Aufbau der Klasse dargestellt.

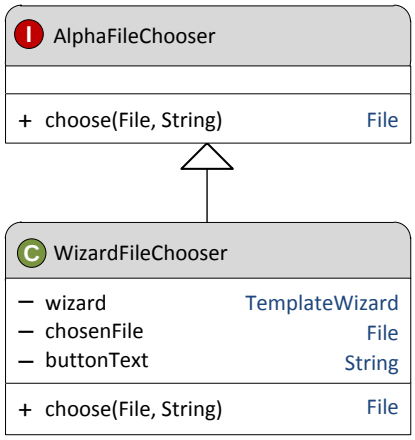


Bild 6.15: Aufbau des *WizardFileChoosers*

6.2.3 Realisierung der Filterklassen

In diesem Abschnitt wird die Umsetzung der in 5.3.4 beschriebenen Filter-Pipeline beschrieben. Für jedes Filter existiert dabei eine korrespondierende Klasse, die die gewünschte Funktionalität implementiert. Die Artefaktfilter besitzen dabei stets eine Filter- und eine Verschmelzungsmethode, wobei letztere nur im Falle des Merge-Imports Anwendung findet (vergleiche 5.3.3). Wie bereits in 6.2.2.2 erwähnt, handelt es sich bei den im Folgenden erwähnten Filtermasken stets um *HashMaps*, deren Wahrheitswerte über die Verwendung der jeweiligen Information entscheiden.

6.2.3.1 APA-Filter

Möchte der Benutzer eigene α-Adornment-Typen importieren oder exportieren, so wird dies durch den *APAFilter* gesteuert. Bei Export und „Drag and Drop“-Import bildet die von außen zugängliche *filterAPA*-Methode den Rahmen. Als Vorarbeit muss die Menge der Adornment-Typen, unter denen der Benutzer wählen darf, mithilfe der *getNewAdornmentNames*-Methode bestimmt werden. Nun wird diese Menge einem *AdornmentNamesPresenter* übergeben, der basierend auf den Auswahlentscheidungen des Benutzers die APA-Filtermaske generiert. Abschließend wird dieses Filter in der Methode *applyFilter* angewendet. Dabei werden aus der ursprünglich geklonten APA-Payload diejenigen Adornment-Typen gelöscht, die vom Nutzer nicht gewählt wurden.

Der beschriebene Ablauf ist in Abbildung 6.16 am Beispiel des Exportvorgangs dargestellt. Dabei ist zur Verdeutlichung der beiden Hauptschritte die Erstellung der Filtermaske blau, die Anwendung dieser rot markiert.

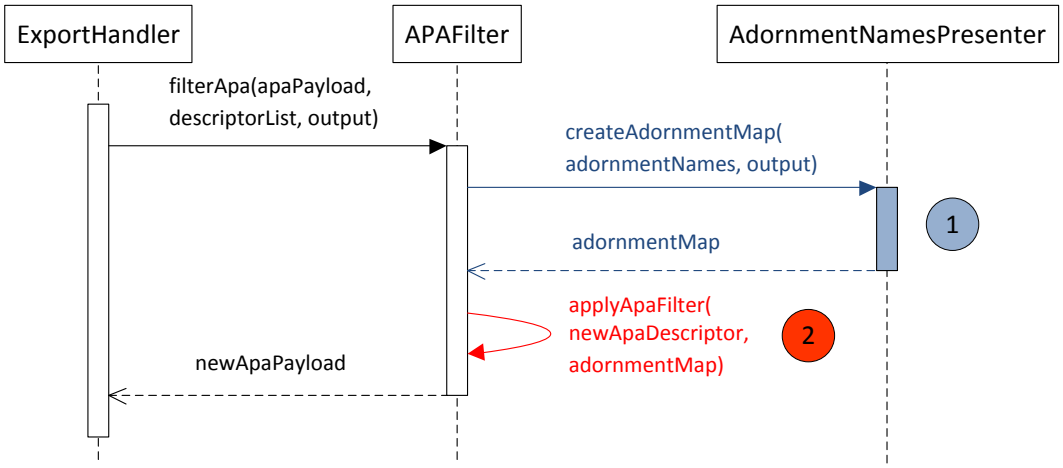


Bild 6.16: Ablauf der reinen APA-Filterung

Importiert der Benutzer ein α-Template in ein bestehendes α-Doc, so wird die APA-Filterung durch einen Aufruf von *mergeAPA* des *APAFilter*s initiiert. Dort werden zunächst die APAs des α-Docs und der Prozessschablone mithilfe von *createUnitedApa* zusammengeführt. Dabei wird die in 5.3.3 erarbeitete Strategie angewendet. Anschließend wird die bereits beschriebene *filterApa*-Methode zur Filterung der Gesamtmenge wiederverwendet. Der einzige Unterschied besteht darin, dass die *getNewAdornmentNames*-Methode nur noch *neue* eigene α-Adornment-Typen zurückliefert.

Abbildung 6.17 zeigt die beschriebene Funktionalität. Neben den bekannten Markierungen aus Bild 6.16 ist nun zusätzlich ein dritter Prozessschritt farblich dargestellt.

Dabei handelt es sich um die grün gekennzeichnete Vereinigung der beiden APAs. Der Aufbau der in diesem Abschnitt beschriebenen *APAFilter*-Klasse ist des Weiteren in 6.18 modelliert.

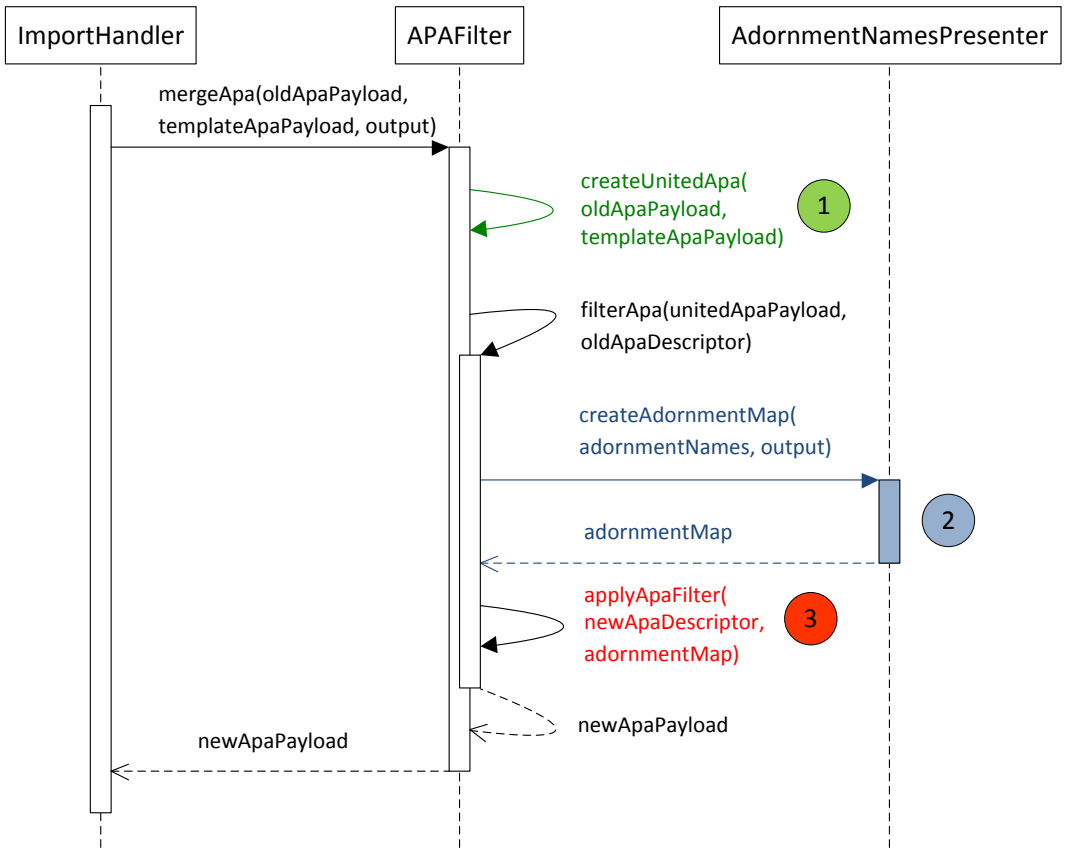


Bild 6.17: Ablauf der APA-Verschmelzung

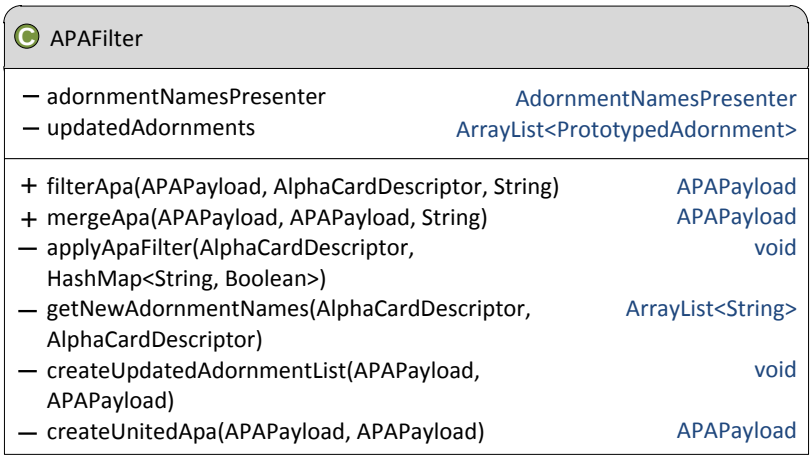


Bild 6.18: Aufbau des *APAFilter*s

6.2.3.2 PSA-Filter

Möchte der Benutzer Behandlungsschritte in Form von α-Cards wiederverwenden, so ist eine Filterung des PSAs sowie der Grundmenge der α-Card-Descriptors notwendig. Die umsetzende Klasse ist dabei der *PSAFilter*, welcher erneut leicht unterschiedliche Funktionalität für reine Filterung und Verschmelzung bietet.

Export und „Drag and Drop“-Import starten den jeweiligen Vorgang mit einem Aufruf von *filterPSA*. Zu Beginn werden dabei in *createListOfContentCards* aus der Menge der übergebenen Descriptors die Coordination-Card-Descriptors entfernt. Dann wird diese Menge einem *AlphaCardNamesPresenter* übergeben, der die PSA-Filtermaske gemäß Nutzerwunsch erstellt. Diese Maske muss schließlich in *applyPsaFilter* angewendet werden. Dazu werden die nicht gewählten α-Cards aus dem zu filternden PSA sowie alle α-Card-Beziehungen, in denen diese α-Cards vorhanden sind, entfernt. Nach anschließender Beendigung der *filterPsa*-Funktion muss zusätzlich die Grundmenge der α-Card-Descriptors gefiltert werden. Dies geschieht in der Methode *adaptDescriptorsToPsa* des *PSAFilter*s, welche diejenigen Descriptors löscht, die nicht im neu erstellten PSA vorhanden sind.

Bild 6.19 verdeutlicht wiederum den beschriebenen Ablauf mithilfe von farbigen Markierungen der wichtigsten Filterungsschritte. Die Erstellung der Filtermaske ist blau, die Anwendung der Maske rot dargestellt.

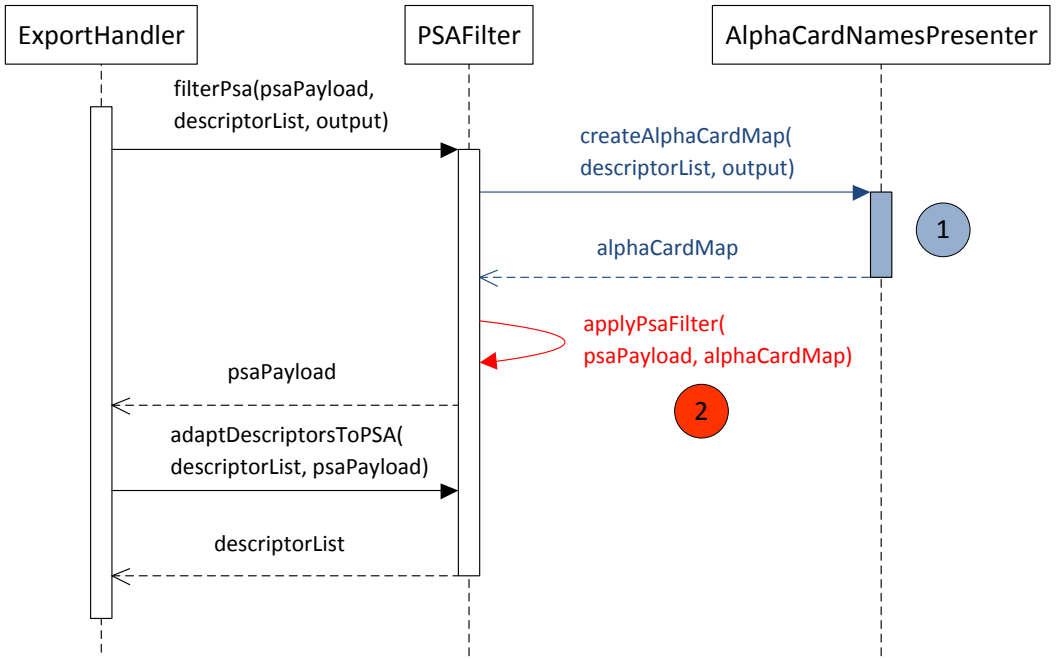


Bild 6.19: Ablauf der reinen PSA-Filterung

Entscheidet sich der Benutzer für einen Merge-Import, so werden die PSAs in der *mergePSA*-Methode des *PSAFilter*s zusammengeführt. Zunächst wird intern mithilfe von *createListOfNewDescriptors* die Menge der bisher nicht vorhandenen α-Cards bestimmt. Anschließend ermittelt ein *AlphaCardNamesPresenter* die Menge der zu importierenden α-Cards. *createMergedPsa* erstellt dann ein neues vereinigtes PSA als Ergebnis des Verschmelzungsprozesses. Zur bestehenden PSA-Payload des α-Docs werden alle gewählten α-Cards hinzufügt. Das gleiche gilt für alle α-Card-Beziehungen des Templates, bei denen beide α-Cards importiert werden.

Nachdem in *mergePsa* ein neues PSA bestimmt wurde, muss nun die Menge der hinzuzufügenden α-Card-Descriptors daraus abgeleitet werden. Dies geschieht in der Methode *createListOfAdditionalCards* des *PSAFilter*s. Das eigentliche Hinzufügen der α-Card-Descriptors wird dann im weiteren Verlauf des Merge-Imports durchgeführt (vergleiche 6.2.4.1).

Der beschriebene Ablauf ist in Bild 6.20 modelliert. Zusätzlich zur bekannten Farbgebung ist hierbei die Bestimmung der noch nicht im α-Doc vorhandenen Descriptors grün markiert. Der Aufbau des beschriebenen *PSAFilter*s findet sich in Abbildung 6.21.

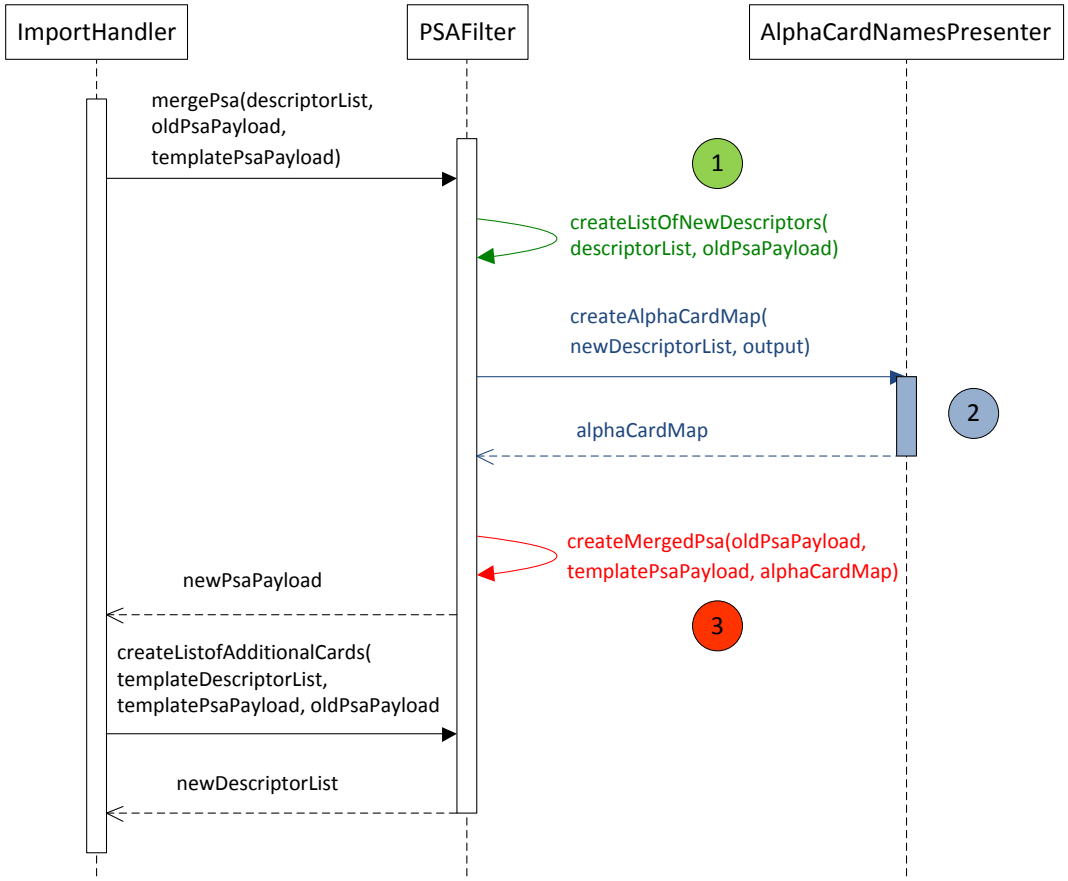


Bild 6.20: Ablauf der PSA-Verschmelzung

6.2.3.3 CRA-Filter

Das letzte Artefakt, das für einen erfolgreichen Import-/Exportvorgang des Benutzers gefiltert werden muss, ist das CRA. Die gewünschte Funktionalität wird dabei von der Klasse *CRAFilter* angeboten.

Eine reine Filterung für den Export sowie den „Drag and Drop“-Import kann durch einen Aufruf der Methode *filterCra* durchgeführt werden. Zu Beginn wird mithilfe eines *ContributorNamesPresenter*s die Menge der zu verwendenden Prozessteilnehmer festgestellt. Die erstellte Filtermaske wird dann in der Methode *applyCraFilter* angewendet. Darin werden alle Namen der gewählten Prozessteilnehmer der neuen CRA-Payload hinzugefügt. Wie bereits in 6.2.2.2 erwähnt, wird der Patientenname getrennt betrachtet. Zur Unterscheidung von den Behandelnden ist dessen *Key* in der *HashMap<Participant, Boolean>* „null“. Der zugehörige Wahrheitswert signalisiert, ob der Patientenname verwendet werden soll.

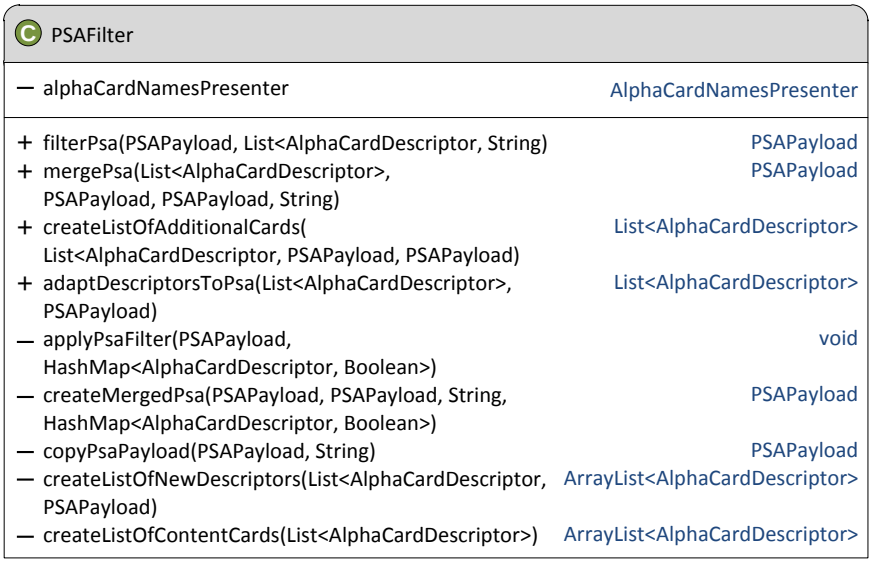


Bild 6.21: Aufbau des *PSAFilter*s

Ein Sequenzdiagramm der beschriebenen reinen CRA-Filterung ist in Bild 6.22 modelliert. Es enthält erneut Markierungen der wichtigsten Prozessschritte. Die Erstellung der Filtermaske ist in blauer, die Anwendung dieser in roter Farbe dargestellt.

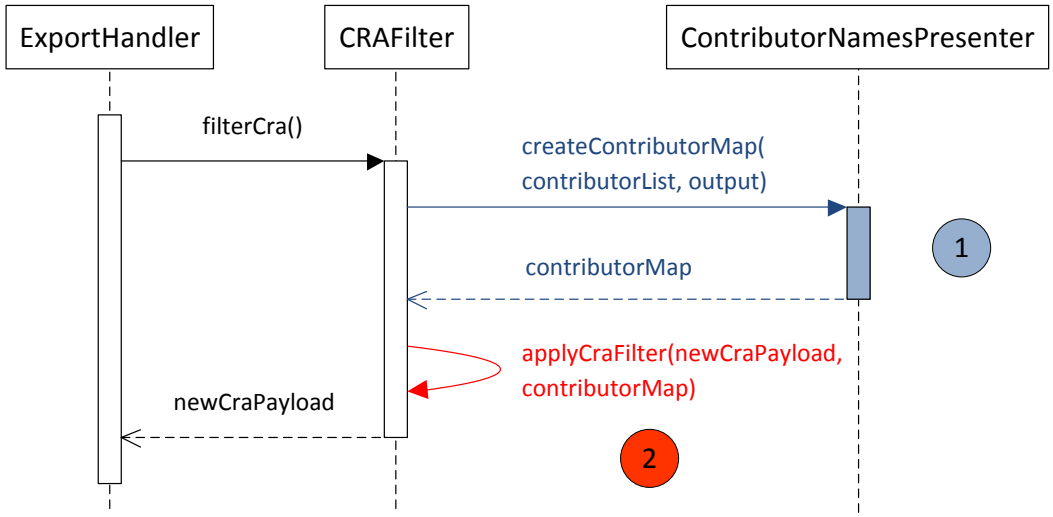


Bild 6.22: Ablauf der reinen CRA-Filterung

Beim Merge-Import wird wiederum ergänzende Funktionalität benötigt, welche in der Methode *mergeCra* implementiert ist. Zunächst wird in *createSetOfNewParticipants* die Menge derjenigen Prozessteilnehmer bestimmt, die im Template, aber nicht im α-Doc vorhanden sind. Diese wird im Anschluss einem *ContributorNamesPresenter* übergeben,

der nach Benutzerwunsch eine CRA-Filtermaske erstellt. In *createMergedCraPayload* wird diese Maske zur Erstellung einer verschmolzenen CRA-Payload verwendet. Darin werden zu der Menge der bestehenden Prozessteilnehmer die neu gewählten hinzugefügt. Eine besondere Behandlung der Patientendaten ist nicht nötig, da diese beim Merge-Import nicht verändert werden (vergleiche 5.3.3).

6.23 zeigt eine Abbildung des CRA-Verschmelzungsablaufs. Als weiterer Hauptschritt ist dabei die Bestimmung der noch nicht vorhandenen Prozessteilnehmer grün gekennzeichnet. Der Aufbau der *CRAFilter*-Klasse wird durch Abbildung 6.24 beschrieben.

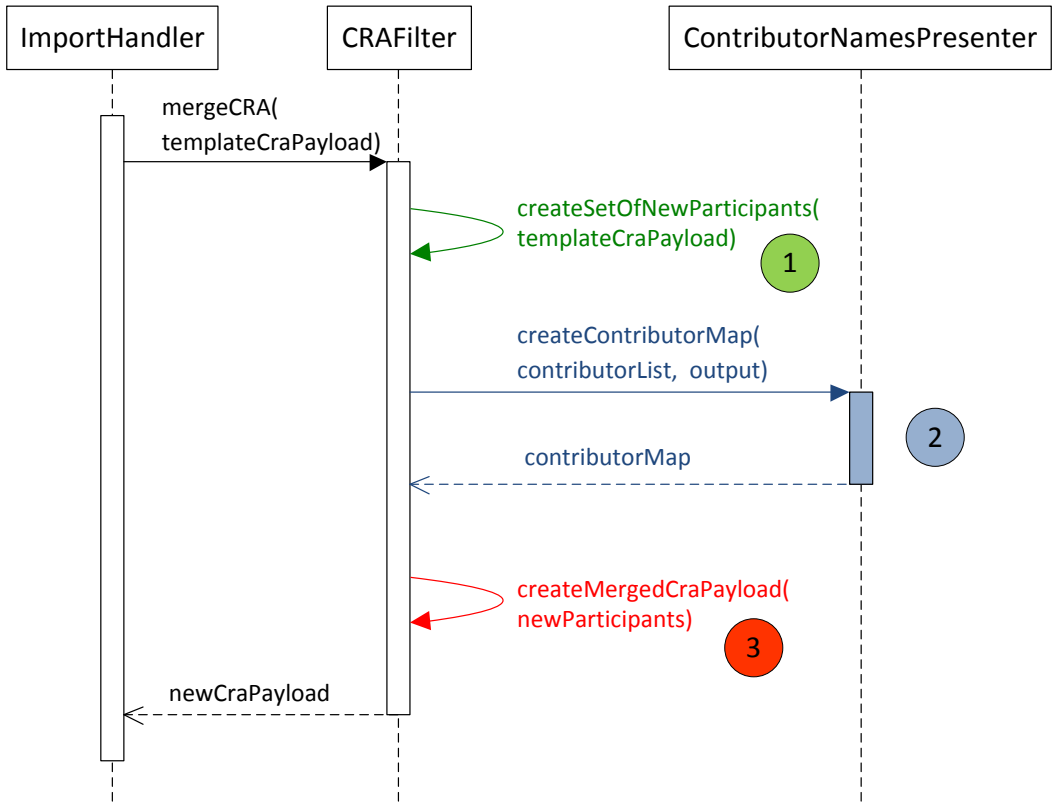


Bild 6.23: Ablauf der CRA-Verschmelzung

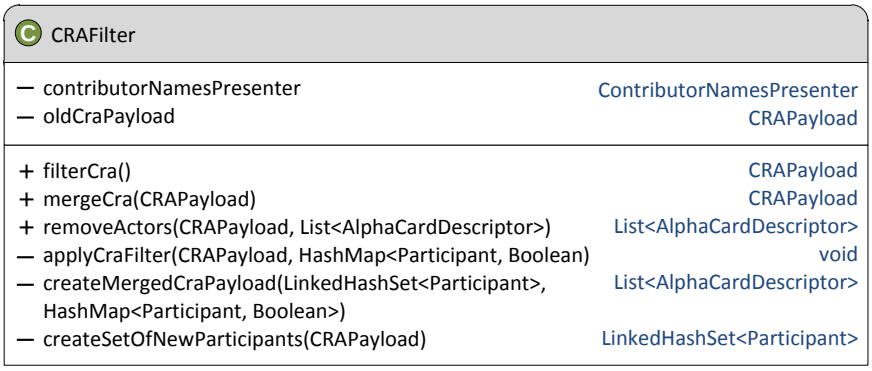


Bild 6.24: Aufbau des *CRAFilter*s

6.2.3.4 α-Adornments-Filter

Als letzten Filterungsschritt darf der Benutzer entscheiden, welche α-Adornments er bei jeder α-Card exportieren beziehungsweise importieren möchte (vergleiche 5.3.4.4). Diese Funktionalität wird im *AdornmentsFilter* umgesetzt. Im Gegensatz zu den Artefaktfiltern unterscheidet sich hier der Ablauf des Merge-Imports nicht von den anderen Vorgängen. Der äußere Zugriff erfolgt stets über die Methode *filterAdornments*, welche man erneut in die typischen zwei Schritte einteilen kann. Zunächst wird eine Filtermaske erstellt, welche dann im zweiten Schritt auf die Menge der α-Cards angewendet wird. Anders als bei den vorigen Schritten ist die in *createAdornmentSets* durchgeführte Erstellung der Maske jedoch umfangreicher. Insgesamt handelt es sich bei der Filtermaske um eine Liste von α-Adornment-Mengen, wobei für jede α-Card eine Adornment-Sammlung existiert.

Zunächst wird in der Methode *createSingleAdornmentSet* eine Menge von Standard-Adornments mithilfe eines *AdornmentNamesPresenter*s generiert. Dabei wird die in 5.3.4.4 konzipierte Aufteilung der Wiederverwendbarkeitsmöglichkeiten der einzelnen α-Adornments berücksichtigt. Im nächsten Schritt bestimmt ein *AlphaCardNamesPresenter* gemäß Benutzerwunsch für welche α-Cards diese Standardmenge angewendet werden soll. Über die zurückgegebene *HashMap* muss nun iteriert werden. Bei jedem Schlüssel-wert „true“ wird die Standard-Adornment-Menge zur Liste hinzugefügt. Ist der Wert „false“, so muss die zu ergänzende Adornment-Menge durch einen erneuten Aufruf von *createSingleAdornmentSet* bestimmt werden. In dieser Methode wird wiederum durch einen *AdormentNamesPresenter* eine Menge von zu verwendenden α-Adornments ermittelt.

Die fertiggestellte Filtermaske kann nun der Methode *applyAdornmentsFilter* zur Anwendung übergeben werden. Diese erstellt für jede α-Card die vom Benutzer ge-

wünschten Adornment-Mengen, indem es die nicht gewählten α-Adornments aus den α-Card-Descriptors entfernt.

Abbildung 6.25 skizziert das beschriebene Verhalten des *AdornmentFilter*s. Neben dem rot markierten Anwenden der Filtermaske ist die Erstellung dieser in drei farblich getrennten Stufen dargestellt. Das Erstellen der Standard-Adornment-Menge ist blau gekennzeichnet, das der Menge der α-Cards, für die dieser Standard gilt, grün. Braun ist schließlich die Bestimmung der zu verwendenden Adornment-Mengen der weiteren α-Cards markiert. Zusätzlich dazu ist der Aufbau der beschriebenen Klasse in 6.26 modelliert.

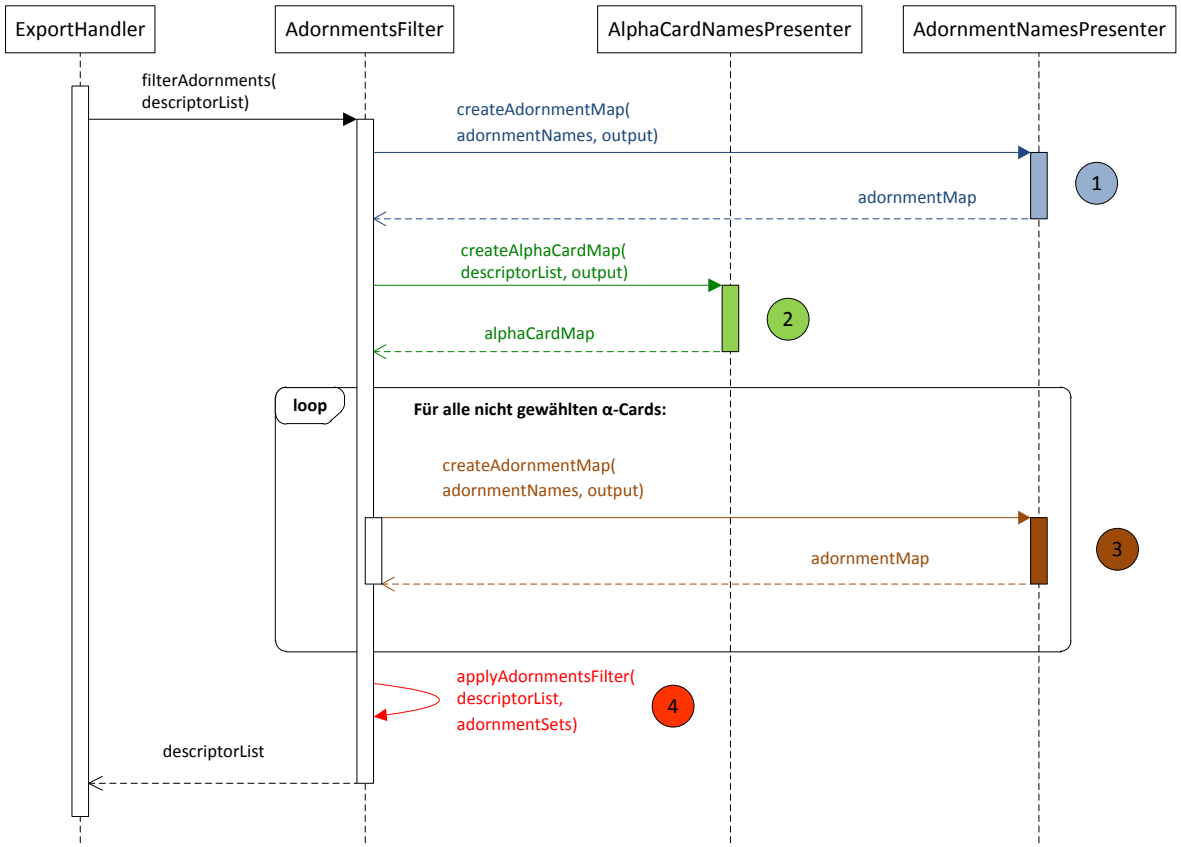


Bild 6.25: Ablauf der Adornments-Filterung

6.2.4 Inneres Verhalten der Basisklassen

In Abschnitt 6.1 wurde das äußere Verhalten der α-Templates-Komponente beschrieben. Dabei wurden bereits die drei Basisklassen vorgestellt, deren interne Abläufe und Zuständigkeiten im Folgenden genauer erklärt werden.

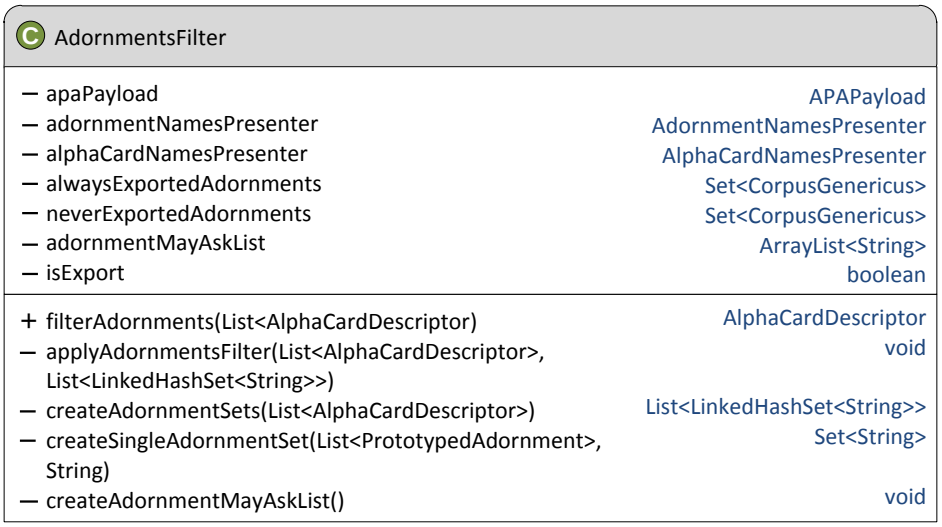


Bild 6.26: Aufbau des *AdornmentFilter*s

6.2.4.1 ImportHandler

Möchte der Benutzer ein „Process Template“ zu einem bestehenden α-Doc hinzufügen, so wird diese Funktionalität durch den *ImportHandler* gesteuert. Die von außen zugängliche Hauptmethode ist dabei *importTemplate*. Dort wird zunächst der Wizard erstellt und nacheinander der Willkommensbildschirm und das Speicherauswahlfenster angezeigt. Nun wird versucht, die gewählte Datei zu einem α-Template zu deserialisieren. Im Erfolgsfall werden nun die „Merge-Methoden“ der in 6.2.3 beschriebenen Filterklassen aufgerufen. Da eine PSA lediglich α-Card-IDs enthält, muss außerdem die Menge der neuen α-Card-Descriptors ermittelt werden. Dies geschieht durch den Aufruf von *createListOfAdditionalAlphaCards* des *PSAFilter*s.

Anschließend müssen die neu zu verwendenden Daten noch bekannt gemacht werden. Dafür werden zunächst die neuen Adornment-Typen den bereits im α-Doc vorhandenen α-Card-Descriptors hinzugefügt (*updateOldDescriptors*) und die neue APA persistiert. Dann können in *addNewParticipants* die neuen Prozessteilnehmer mittels vorhandener Funktionalität der *AlphaPropsFacade* zum α-Doc hinzugefügt werden. Außerdem müssen die α-Card-Descriptors vervollständigt (vergleiche 6.2.1) und wiederum mithilfe der *AlphaPropsFacade* dem α-Doc übergeben werden. Nun wird das Ergebnis des Imports im Wizard präsentiert und der Vorgang ist abgeschlossen. Das Klassendiagramm zum *ImportHandler* findet sich in Bild 6.27.

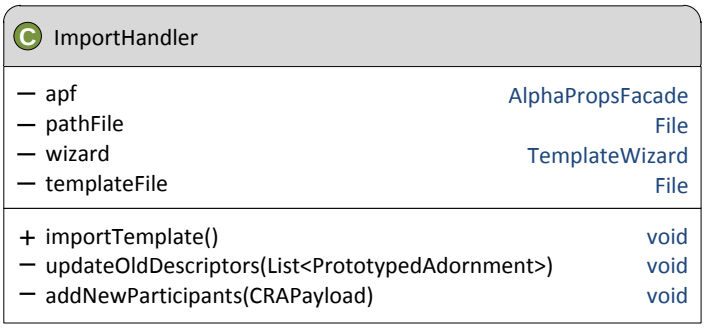


Bild 6.27: Aufbau des *ImportHandler*s

6.2.4.2 TemplateInjector

Entscheidet sich der Benutzer für die Erstellung eines neuen, α-Template-basierten, α-Docs, bildet der *TemplateInjector* den Rahmen der Umsetzung. Die Implementierung erfolgt dabei in der Methode *createDocFromTemplate*. Diese präsentiert zu Beginn wiederum den Willkommensbildschirm mithilfe des zuvor erstellten Wizards. Das Dateiauswahlfenster wird hier nicht benötigt, da beim „Drag and Drop“ von Dateien deren Pfad automatisch als Argument übergeben wird. Nachdem das α-Template erfolgreich deserialisiert wurde, werden, analog zur normalen Erstellung eines α-Docs, Episodeninformationen eingelesen und die Verwaltungsstrukturen des α-Docs erstellt. Anschließend finden die typischen Filterungen statt, wobei jeweils die Filtermethoden (im Gegensatz zu den Merge-Methoden) gewählt werden. Wie im *ImportHandler* ist bei der Filterung der α-Cards zusätzlich eine Anpassung der Gesamtmenge der α-Descriptors nötig. Dies wird in der Methode *adaptDescriptorsToPsa* des *PSAFilter*s umgesetzt. Nach der erneuten Vervollständigung der α-Card-Descriptors (vergleiche 6.2.1) werden diese und alle anderen Daten des α-Docs serialisiert. Danach kann das Ergebnis des Importvorgangs im Wizard angezeigt und das Programm geschlossen werden.

Das Modell des *TemplateInjector*s ist in Abbildung 6.28 gegeben. Da die Funktionalität der normalen Erstellung eines α-Docs mit initialem Payload-Dokument ähnelt und Teile davon wiederverwendet werden können, ist die Klasse vom dafür zuständigen *Injector* abgeleitet.

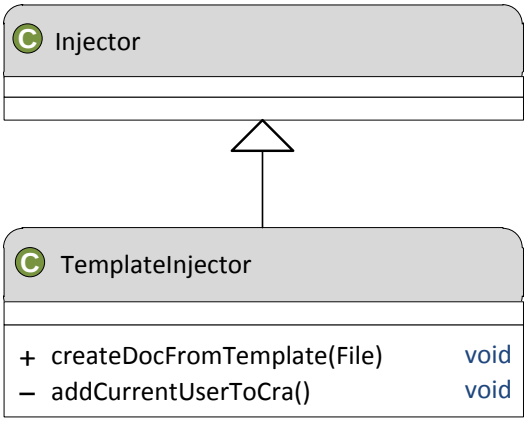


Bild 6.28: Aufbau des *TemplateInjector*s

6.2.4.3 ExportHandler

Ein gewünschter Export eines „Process Templates“ wird durch den *ExportHandler* koordiniert. Innerhalb der Klasse bildet die *exportTemplate*-Methode den Funktionalitätsrahmen. Sie erstellt zu Beginn den Wizard und initiiert darin den Begrüßungsbildschirm sowie die Wahl eines Speicherorts für die zu erstellende α-Template-Datei. Nun wird der typische Filterungszyklus aus 5.3.4 durchlaufen, wobei wie im *TemplateInjector* die jeweiligen Filtermethoden (im Gegensatz zu den Merge-Methoden) verwendet werden. Die zusätzliche Filterung der α-Card-Descriptor-Grundmenge wird, ebenso analog zum „Drag and Drop“-Import, mithilfe von *adaptDescriptorsToPsa* des *PSAFilter*s durchgeführt.

Beim Export ist außerdem auf einige Besonderheiten zu achten. Es müssen zu Beginn vollständige echte Kopien der zu filternden α-Card-Descriptors erstellt werden. Ansonsten würden die Daten des grundlegenden α-Docs miteditiert werden. Dies geschieht in den Methoden *clonePsaWithoutEpisodeId* und *cloneDescriptorsWithoutEpisodeId*. Da in diesen Methoden bereits über alle α-Cards iteriert werden muss, werden hier auch die enthaltenen Episoden-IDs entfernt. Es handelt sich dabei um Instanzinformationen, die nicht exportiert werden dürfen. Des Weiteren dürfen je nach Ergebnis der CRA-Filterung bestimmte α-Adornment-Werte nicht ins α-Template übernommen werden. Dabei handelt es sich um den Namen des Patienten sowie anderer Prozessteilnehmer, falls der Benutzer diese nicht exportieren möchte (vergleiche 5.3.4.4).

Abschließend wird ein neues α-Template-Objekt instantiiert und mit den gefilterten Daten gefüllt. Das fertige Template kann nun am anfangs gewählten Ort gespeichert und das Ergebnis des Exportvorgangs dem Benutzer angezeigt werden. Der Aufbau des *ExportHandler*s findet sich in Abbildung 6.29.

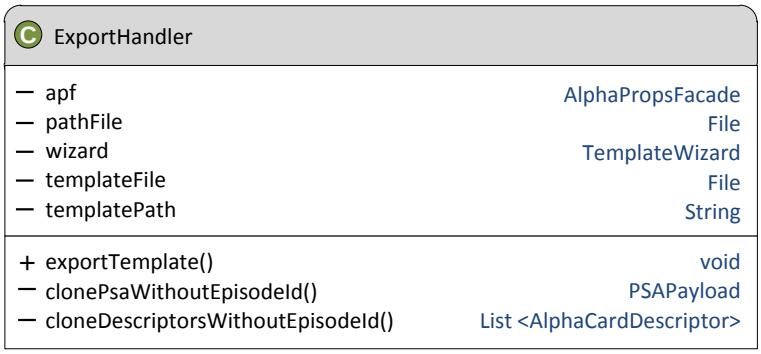


Bild 6.29: Aufbau des *ExportHandler*s

6.3 Zusammenfassung

In diesem Kapitel wurde die Prototypische Umsetzung der α-Templates-Komponente erläutert. 6.1 handelte zunächst vom äußeren Verhalten der Komponente und dessen Einbettung in das Gesamtsystem. Dazu wurden die drei Basisklassen für Export, „Drag and Drop“- und Merge-Import eingeführt und deren Interaktionen mit den weiteren Subsystemen skizziert. *ExportHandler* und *ImportHandler* werden bei einem visualisierten α-Doc im Laufe des normalen Betriebs initiiert und kommunizieren daher mit dem *Editor* sowie der *AlphaPropsFacade*. Der *TemplateInjector* wird dagegen bei der Erstellung eines neuen α-Docs ausgeführt und interagiert stattdessen mit dem *Injector*.

Im Anschluss daran wurde in 6.2 der eigentliche Software-Entwurf vorgestellt. Zu Beginn wurde in 6.2.1 der technische Aufbau eines α-Template-Datentyps beschrieben. Dieser stellt im Wesentlichen eine Komposition der einzelnen Prozessartefakte und der Grundmenge der α-Card-Descriptors dar.

Danach folgte in 6.2.2 die Umsetzung des Wizards und der einzelnen Wizard-Fenster. Der Wizard besteht aus einer Reihe von *JPanel*s, wobei vor allem Überschrifts- und Auswahl-Panel einem ständigem Wandel unterstehen. In den Auswahlfenstern wird eine Reihe von Daten (beispielsweise eine Sammlung von α-Card-Namen) in Form von *JCheckBox*es zum Auswahl-Panel hinzugefügt. Nach der Bestätigung der Auswahlentscheidungen werden die *JCheckBox*es ausgewertet und die Ergebnisse in einer *HashMap* gespeichert. Im Nachrichtenfenster wird lediglich eine Meldung angezeigt, während im Speicherauswahlfenster ein *JFileChooser* eingebettet ist, der die Auswahl eines Speicherorts im System ermöglicht.

Anschließend wurde in 6.2.3 die Umsetzung der Artefaktfilter sowie des α-Adornments-Filters beschrieben. In der Beschreibung der Artefaktfilter wurde stets zwischen einer

einfachen Datenreduzierung bei Export sowie „Drag and Drop“-Import und einer Verschmelzung von Daten beim Merge-Import unterschieden. Bei allen Filterklassen lassen sich zwei Teilschritte identifizieren. Dabei handelt es sich um die Erstellung einer Filtermaske auf Basis von Benutzereingaben und um die Anwendung dieser Maske auf eine bestehende Datenstruktur.

Zum Schluss wurde in 6.2.4 das innere Verhalten der Basisklassen beschrieben. Neben der Initialisierung des Wizards und dem Aufruf der einzelnen Filter haben diese noch weitere besondere Aufgaben. Der *ImportHandler* muss die neu zu verwendenden Daten mittels vorhandener Funktionalität der *AlphaPropsFacade* zum bestehenden α-Doc hinzufügen. Der *TemplateInjector* muss dagegen die gesamte Ordnerstruktur des α-Docs erstellen und dessen Daten serialisieren. Die spezielle Aufgabe des *ExportHandler*s ist die Erstellung eines neuen α-Templates und das Hinzufügen der gewählten Daten.

7 Ausblick

In diesem Kapitel wird die Verwendung von „Process Templates“ kritisch analysiert und ein Ausblick auf zukünftige Arbeiten gegeben. Dabei wird zunächst in Abschnitt 7.1 auf sinnvolle Einschränkungen in der Nutzung von Prozessschablonen hingewiesen. In den folgenden Abschnitten werden Vorschläge für die weitere Verbesserung der α-Templates-Komponente vorgestellt und diskutiert.

7.1 Maßvoller Umgang mit Prozessschablonen

„Eine Schablone schaffen, das ist Genie.“
(Charles Baudelaire)

Prozessschablonen können einen entscheidenden Beitrag zur effektiven wie auch effizienten Patientenbehandlung liefern. Sie können das Risiko von Behandlungsfehlern sowie die Kosten für alle beteiligten Parteien reduzieren. Dennoch werden Ärzte die Nutzung der Templates auf sinnvolle Anwendungsfälle beschränken. Mediziner wissen aus Erfahrung, dass nicht alle Behandlungsepisoden musterhaft verlaufen. Je unterschiedlicher vergangene Abläufe waren, desto geringer ist der potentielle Wiederverwendungswert einer Prozessschablone. In diesem Fall wird der Arzt wohl auf die Erstellung eines Templates verzichten. Sowohl bei der Erstellung als auch bei der späteren Verwendung der α-Templates wäre der jeweilige Aufwand für ihn zu hoch. Auch bei seltenen Krankheiten, deren Behandlung möglicherweise sehr strukturiert abläuft, wird der Benutzer vor dem Export eine Kosten-Nutzen-Analyse durchführen.

Vor Anwendung der Prozessschablonen sollten außerdem erneut Vor- und Nachteile abgewogen werden. Der Benutzer kann für jede Behandlungsepisode separat entscheiden, ob die Verwendung einer Prozessschablone sinnvoll ist. Auf die gründliche Konzeption eines individuellen Behandlungsplans wird er jedoch in keinem Fall verzichten. α-Templates können hierbei zwar eine wichtige Hilfestellung bieten, der Arzt muss trotzdem jeden Behandlungsschritt auf seine Sinnhaftigkeit überprüfen.

7.2 Vollständige Validierung der α-Templates

In Abschnitt 5.3.2 wurde die grundlegende Validierung der α-Templates beschrieben. Diese automatische Validierung von JAXB funktioniert jedoch nur sehr rudimentär. Es wird lediglich der Name des Wurzelelements auf Korrektheit geprüft. In den allermeisten Fällen wird dieses Procedere ausreichen. Trotzdem wäre es praktisch, einen größeren Teil des α-Templates untersuchen lassen zu können. Beispielsweise könnte man prüfen, ob die drei Systemartefakte sowie die Menge von α-Descriptors grundsätzlich vorhanden sind, da ansonsten der Import scheitern würde. Außerdem könnte man sicherstellen, dass die gespeicherten α-Cards jeweils alle fundamentalen, prozessrelevanten (vergleiche 5.3.4.4) α-Adornments enthalten. Dazu würde man sich ein XML-Schema definieren, welches die genannten Anforderungen an die Prozessschablone festlegt. JAXB bietet dann die Möglichkeit an, gegen das XML-Schema zu validieren.

7.3 Implementierung eines „Zurück“-Buttons

In den Abschnitten 5.3.4.3 und 5.3.5 wurde der Verzicht auf die Implementierung einer „Zurück“-Funktionalität erklärt. Nachdem der Benutzer alle Entscheidungen im Rahmen eines Filterungsschritts getroffen hat, wird dieser direkt durchgeführt. Ein transaktionales Umschalten nach dem Sammeln aller benötigten Informationen ist mit dem erarbeiteten Konzept nicht möglich. Der Grund dafür ist, dass Inhalte und sogar die Existenz von später im Prozess angezeigten Wizard-Fenstern von vorigen Auswahlentscheidungen abhängen können, weshalb deren direkte Verarbeitung notwendig ist. Beispielsweise muss für eine nicht exportierte α-Card die Menge der zu exportierenden α-Adornments nicht mehr bestimmt werden. Würde man nun die Benutzung eines „Zurück“-Buttons erlauben, so müssten umfangreiche Rollback-Mechanismen für jeden Filterungsschritt konzipiert werden.

Einen Sonderfall stellt jedoch das α-Adornments-Filter dar. Anders als bei den Artefaktfiltern erstreckt sich die Erstellung der Filtermaske hier über mehrere Wizard-Fenster (vergleiche 6.2.3.4). Erst nach Bestätigung der letzten Auswahl wird die eigentliche Filterung vollzogen. Bei den Fenstern des α-Adornments-Filters könnte daher das Klicken eines „Zurück“-Buttons mit einem Neustart dieser Filtermaskenerstellung verknüpft werden.

7.4 Vollständiger Verzicht auf den α-Card-Identifier (α-Card-ID) in „Process Templates“

Ein α-Card-ID ist ein eindeutiges Tupel aus der Behandlungsnummer und der Nummer der α-Card. Die Behandlungsnummer wird nicht im α-Template gespeichert. Die Nummern der α-Cards werden dagegen zum jetzigen Zeitpunkt mitexportiert und bei beiden Formen des Imports wiederverwendet. Identifikationsnummern werden bei α-Templates vor allem deshalb benötigt, um Beziehungen zwischen α-Cards (vergleiche 3.1.2) speichern zu können. Die Benutzung der alten Nummern vereinfacht den Export und erleichtert die Erkennung von duplizierten α-Cards beim Merge-Import. Da vollständige α-Card-IDs zusätzlich eine eindeutige Behandlungsnummer enthalten, bleibt das Zuordnen der α-Cards zur richtigen α-Episode gesichert.

Trotzdem stellt die Nummer der α-Card eigentlich eine spezifische Instanzinformation dar, welche nicht wiederverwendet werden sollte. Stattdessen könnte man die exportierten α-Cards mit einer eigenen Nummerierung versehen. Beim Import sollte in Zukunft also ein komplett neuer ID erstellt werden. Auch die erwähnten Probleme beim Merge-Import (5.3.3) treten in der Praxis selten auf. Behandlungsteams werden nur in wenigen Fällen die gleiche α-Card exportieren und später wieder importieren.

7.5 Editiermöglichkeit beim Export

Beim Export ist es bisher möglich, bestehende Informationen des α-Docs entweder unverändert zu übernehmen oder sie jeweils wegzulassen. Man muss jedoch davon ausgehen, nicht immer eine perfekt beispielhafte α-Episode als Grundlage der Prozessschablone zu haben. Beispielsweise könnte eine einzelne Teiluntersuchung bei einem anderen Arzt als üblich durchgeführt worden sein. Im bestehenden α-Doc kann man diese Information nicht anpassen, da man sonst die Behandlungsdokumentation verfälschen würde. Somit kann in diesem Fall der Name des Behandelnden nicht exportiert werden.

Weiterhin ist es denkbar, ein Template in verschiedenen Sprachen schaffen zu wollen (vergleiche 4.4). α-Docs sind zwar in englischer Sprache konzipiert, die Benennung der Untersuchungen erfolgt jedoch wahrscheinlich in Landessprache.

Der Ersteller des α-Templates besitzt in der Regel nur für eine beschränkte Anzahl von α-Cards Schreibrechte. Bei allen anderen α-Cards kann er somit auch keine Tippfehler ausbessern.

Aus diesen Gründen ist es sinnvoll, eine Editiermöglichkeit für α-Card-Adornments während des Exports zu schaffen. Hierdurch könnte man die Prozessschablonen optimieren und sie mit höherem Informationsgehalt versehen.

7.6 Definition von Abstraktionsstufen bei der Templatisierung

In 5.1.3 wurden bereits Strategien zur sinnvollen Erstellung von α-Templates vorgestellt. Dabei handelte es sich um generalisierte Schablonen, die stets wiederverwendet werden können, sowie um Templates, die auf ihre spezielle Behandlungsumgebung zugeschnitten sind. Um Zeit zu sparen, wäre es nützlich, eine automatische Umsetzung dieser beiden Anwendungsfälle anzubieten. Der Benutzer könnte dann anfangs entscheiden, ob er den Export manuell oder (teilweise) automatisiert durchführen möchte. Der Ersteller könnte, analog zum Adornments-Filter aus 5.3.4.4, eine Menge vom α-Cards wählen, bei der eine bestimmte Templatisierungstrategie, beispielsweise die Schaffung einer Schablone für ein gleichbleibendes Behandlungsumfeld (5.1.3), angewendet wird.

Auch eine „Alles Exportieren“-Funktion ist in diesem Kontext denkbar. Jedoch sollten α-Templates grundsätzlich möglichst sorgfältig erstellt werden, da nur so alle Vorteile genutzt werden können. Beispielsweise kann das Entfernen von überflüssigen Informationen beim Importvorgang die erwünschte Aufwandsreduktion durch Schablonen zunichte machen.

7.7 Export-Adornment

Im Rahmen von α-Flow ist es möglich, eigene α-Adornment-Typen zu definieren. Eine nützliche Anwendung im Kontext von α-Templates wäre die Erstellung eines „Template-Candidate“-Adornments mit booleschem Wertebereich. Man könnte nun beispielsweise bei der Erstellung einer α-Card dieses Flag setzen. Hierdurch würde man andeuten, dass diese α-Card ein Kandidat für einen späteren Template-Export ist.

Intern besteht außerdem die Möglichkeit, allen Datentypen des α-Flow-Datenmodells α-Adornments zu geben. Dies umfasst neben den α-Cards alle Systemartefakte sowie sogar die Adornments selbst. Man könnte damit nach einem durchgeführten Exportvorgang alle verwendeten Daten als „exportiert“ markieren. Bei einem weiteren Exportvorgang kann diese Auswahl dann erneut vorgeschlagen werden.

In der Praxis wird der Template-Export selten auftreten. Deshalb ist es fraglich, ob sich die manuelle Führung eines „Export“-Adornments lohnt. Die Markierung der Daten nach Exportvorgängen würde für die Benutzer hingegen keinen zusätzlichen Aufwand bedeuten. Stellt man kleinere Fehler an einem α-Template fest, könnte der abermalige Export deutlich erleichtert werden. Das umfangreiche Hinzufügen von α-Adornments könnte jedoch den Exportvorgang erheblich verzögern, da diese in der Regel unter allen Prozessteilnehmern synchronisiert werden müssten. Außerdem könnte wiederum ein Problem mit fehlenden Schreibrechten bestehen und möglicherweise die Episodendokumentation verfälscht werden.

7.8 Aufzeigen der Konflikte beim Merge-Import

In 5.3.3 wurde der Lösungsansatz für die speziellen Herausforderungen des Merge-Imports beschrieben. Beim Merge-Import werden die Datenstrukturen des α-Templates mit den vorhandenen zusammengeführt. Dabei haben bestehende Daten im Konfliktfall automatisch Vorrang. Diese Strategie vereinfacht in den meisten Fällen den Importvorgang.

In bereits erwähnten Ausnahmefällen (vergleiche wiederum 5.3.3) kann es jedoch dazu kommen, dass die Mächtigkeit der α-Templates eingeschränkt wird. Daher wäre es denkbar, Konfliktfälle beim Merge-Import aufzuzeigen und dem Benutzer eine Wahlmöglichkeit zu überlassen.

Hierfür müssten geeignete Darstellungskonzepte entwickelt werden. Gerade die Visualisierung von konkurrierenden α-Cards könnte dabei aufgrund der hohen Zahl an α-Adornments eine Herausforderung darstellen. Selbst wenn nur abweichende α-Adornments präsentiert werden, könnte dies die kompakte Wizard-Oberfläche überladen. Eine Beschränkung auf wenige charakterisierende α-Adornments wäre möglich, würde jedoch abermals die Nutzbarkeit der α-Template-Informationen limitieren.

Konflikte unter Prozessteilnehmern und unter α-Adornment-Typen könnten dahingegen relativ leicht visualisiert werden. Daher bleibt insbesondere bei diesen Datenstrukturen die Umsetzung der Idee ein Thema für die Zukunft.

7.9 Vorlagen für Payloads

Eine (Content-)α-Card besteht stets aus einem α-Card-Descriptor und kann darüber hinaus eine Payload umfassen. Im Rahmen der vorgestellten Import-/Exportfunktion für „Process Templates“ werden bislang nur die α-Card-Descriptors von Content-Cards

exportiert. Da es sich bei dem Inhalt von Content-Payloads um Instanzdaten handelt, sind diese behandlungsspezifisch und können in ihrer Gesamtheit nicht wiederverwendet werden. Es wäre jedoch für die Zukunft denkbar, typische Strukturen von Payload-Dokumenten zu definieren, um Payload-Vorlagen zu α-Templates hinzufügen zu können. Beispielsweise könnte ein Arzt bei der Anamnese dem Patienten stets ähnliche oder sogar die gleichen Fragen stellen. Diese Fragen mit ihren zugehörigen Antwortfeldern sowie einem Freitextfeld könnten in einer Payload-Vorlage gespeichert und dem α-Template beim Export hinzugefügt werden.

7.10 Beziehungen zwischen α-Templates und α-Docs

Innerhalb von PSAs ist es möglich, Beziehungen zwischen einzelnen α-Cards zu definieren (vergleiche Abschnitt 3.1.2). Auf höherer Ebene könnte nun ein ähnliches Konzept zwischen verschiedenen α-Templates oder zwischen α-Templates und α-Docs entwickelt werden. Voraussetzung dafür ist die Definition eines eindeutigen Template-IDs. Ein Beispiel für eine spezifizierbare Verwandtschaft einzelner Prozessschablonen wäre eine „ist Variante von“-Beziehung. Auch eine „ist Teilmenge von“-Beziehung wäre sinnvoll, falls einzelne Unterprozesse in einer Behandlung identifizierbar sind. Diese Subprozesse könnten dann feingranular exportiert werden, um dem Benutzer eine Wahl zwischen möglichst individueller Prozesskomposition sowie einem vollständigen Import zu lassen. Zwischen α-Templates und α-Docs könnte darüber hinaus eine „basiert auf“-Relation definiert werden, die den Ursprung einer Prozessschablone festhält.

8 Zusammenfassung der Ergebnisse

Im heutigen Informationszeitalter ist die Nutzung von Computern zur Unterstützung von Patientenbehandlungen nicht mehr wegzudenken. Die im Rahmen von α-Flow entwickelte verteilte, elektronische Fallakte bietet Funktionalität für institutionsübergreifende, dokumentenbasierte Behandlungsabläufe. Prozessschablonen können nun den Aufwand der Prozessplanung im Rahmen von α-Flow verringern und dabei die Häufigkeit von Fehlern senken. In dieser Arbeit wurde daher ein Verfahren zum Import und Export von „Process Templates“ konzipiert sowie dessen Umsetzbarkeit mithilfe eines Prototyps bewiesen.

In den ersten Kapiteln wurden die wissenschaftlichen und technischen Grundlagen der α-Template-Komponente erarbeitet. Dabei wurde festgestellt, dass insbesondere Repräsentation und Verwaltung der Menge von „Process Templates“ Gegenstand aktueller Forschung in diesem Themengebiet sind und daher bei der zukünftigen Weiterentwicklung der α-Templates-Komponente eine Rolle spielen.

Im Anschluss daran wurde das Konzept des α-Templates-Subsystems geschildert. Aus dem vorgestellten Anwendungsszenario wurden zunächst die drei Kernfunktionalitäten abgeleitet. Der Benutzer kann zum einen mittels Button-Klick im α-Editor ein α-Template auf Basis des aktuellen α-Docs exportieren, wobei er den Umfang der gespeicherten Informationen bestimmen darf. Zum anderen kann er ein α-Template sowohl mittels Buttonklick im α-Editor als auch mittels „Drag and Drop“ auf eine spezielle Datei nach seinen individuellen Vorgaben importieren. Im ersten Fall, dem sogenannten Merge-Import, werden die Daten des bestehenden α-Docs mit denen des α-Templates zusammengeführt, wobei bei Konflikten stets die alte Information erhalten bleibt. Im zweiten Fall wird ein neues α-Doc auf Basis der Prozessschablone erstellt. Die Import- und Exportvorgänge werden mithilfe eines Wizards umgesetzt. Dieser führt den Benutzer durch eine Reihe von Dialogfenstern, in denen er die Menge der zu exportierenden, respektive importierenden Daten bestimmen kann. In den Wizard-Fenstern werden dann die erwähnten Auswahlentscheidungen jeweils für α-Cards, Prozessteilnehmer, α-Adornment-Typen und die α-Adornment-Mengen aller α-Cards getroffen. Die Datenstrukturen werden dann den Entscheidungen entsprechend gefiltert und können beim Export in das α-Template,

beziehungsweise beim Import in das α-Doc, übernommen werden. Da der Inhalt späterer Wizard-Fenster von vorherigen Entscheidungen abhängen kann, werden die Filterungen jeweils unmittelbar nach der Benutzerauswahl durchgeführt.

Als nächstes wurde ein Prototyp beschrieben, der das erarbeitete Konzept umsetzt. Jede der erwähnten Kernfunktionalitäten wird in einer eigenen Rahmenklasse implementiert. Diese verwenden intern die gleichen Filterklassen, führen darüber hinaus jedoch funktionalitätsspezifische Operationen, wie die Erstellung einer neuen α-Doc-Ordnerstruktur beim „Drag and Drop“-Import oder den Aufbau eines α-Template-Objekts beim Export, durch.

Im Ausblick wurden schließlich die Einschränkungen des α-Templates-Prototyps und offene Arbeiten an diesem aufgezeigt. Es wurde im Rahmen dieser Arbeit dargelegt, dass „Process Templates“ bereits jetzt zur Erleichterung und Qualitätssicherung von strukturiert ablaufenden Patientenbehandlungen beitragen können. Trotzdem können Prozessschablonen durch schrittweise Umsetzung der offenen Punkte sowie zusätzlicher Ideen die Behandlungsabläufe in Zukunft noch weiter verbessern.

Literaturverzeichnis

[BC89] K. Beck and W. Cunningham. A laboratory for teaching object oriented thinking. In *Conference proceedings on Object-oriented programming systems, languages and applications*, OOPSLA '89, pages 1–6, New York, NY, USA, 1989.

[BKL05] Mario Beyer, Klaus Kuhn, and Richard Lenz. Potential der CDA in verteilten Gesundheitsinformationssystemen. In *EAI*, 2005.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, New York, NY, USA, 1996.

[Che76] Peter Pin-Shan Chen. The entity-relationship model — toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, March 1976.

[DRK97] Peter Dadam, Manfred Reichert, and Klaus Kuhn. Clinical Workflows - The Killer Application for Process-oriented Information Systems? In *4th International Conference on Business Information Systems (BIS 2000*, pages 36–59, 1997.

[FL90] M. J. Field and K. H. Lohr. *Clinical practice guidelines: Directions for a new program*. National Academy Press, 1990.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[GHS95] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. *Distrib. Parallel Databases*, 3:119–153, April 1995.

[Haa05] Peter Haas. Grundlagen zur Elektronischen Krankenakte. In *Medizinische Informationssysteme und Elektronische Krankenakten*, pages 185–274. Springer Berlin Heidelberg, 2005.

[Han10] S. Hanisch. Konzeption und Implementierung einer Infrastruktur für aktive Dokumente. Diplomarbeit, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, 2010.

[Hol95] D. Hollingsworth. Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition, January 1995.

[IBM] IBM. IBM Patterns for E-Business. `http://www.ibm.com/developerworks/patterns/`. *zuletzt abgerufen am 22.10.2011.*

[IBM09] IBM Deutschland GmbH. IBM Rational Team Concert 2.0.0.2 Information Center. `http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp`, 2008, 2009. *zuletzt abgerufen am 22.10.2011.*

[Kol93] Janet Kolodner. *Case-based reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Lan10] Frank Langanke. Einführung/Evaluation eines kooperativen Entwicklungswerkzeuges zur Unterstützung von Projektteams am Beispiel von IBM Rational Team Concert. Diplomarbeit, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Department Informatik, 2010.

[LBC+95] Lucian L. Leape, David W. Bates, David J. Cullen, Jeffrey Cooper, Harold J. Demonaco, Theresa Gallivan, Robert Hallisey, Jeanette Ives, Nan Laird, and Glenn Laffel. Systems Analysis of Adverse Drug Events. *JAMA: The Journal of the American Medical Association*, 274(1):35–43, 1995.

[LED+99] A. LaMarca, W. K. Edwards, P. Dourish, J. Lamping, I. Smith, and J. Thornton. Taking the work out of workflow: mechanisms for document-centered collaboration. In *6th ECSCW*, pages 1–20. Kluwer Academic Publishers Norwell, USA, 1999.

[LO06] M. Lelgemann and G. Ollenschläger. Evidenzbasierte Leitlinien und Behandlungspfade. *Der Internist*, 47:690–698, 2006.

[LS05] Y. Lin and D. Strasunskas. Ontology-based Semantic Annotation of Process Templates for Reuse. In *Proceedings of 10th International workshop EMMSAD'05*, Porto, Portugal, June 2005.

[MCH03] Thomas W. Malone, Kevin Crowston, and George A. Herman. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, Cambridge, MA, USA, 2003.

[MDFK97] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '97, pages 181–194, New York, NY, USA, 1997.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society.

[MLLL10] Joanna McGrenere, Jin Li, Jimmy Lo, and Elena Litani. Designing effective notifications for collaborative development environments. In Mark Chignell, James Cordy, Joanna Ng, and Yelena Yesha, editors, *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 65–87. Springer Berlin / Heidelberg, 2010.

[MZM04] Therani Madhusudan, J.Leon Zhao, and Byron Marshall. A case-based reasoning framework for workflow model management. *Data and Knowledge Engineering*, 50(1):87 – 115, 2004.

[NL09] Christoph P. Neumann and Richard Lenz. alpha-Flow: A Document-based Approach to Inter-Institutional Process Support in Healthcare. In *Proc of the 3rd Int'l Workshop on Process-oriented Information Systems in Healthcare (ProHealth'09) in conjunction with the 7th Int'l Conf on Business Process Management (BPM'09)*, Ulm, Germany, September 2009.

[NL10] Christoph P. Neumann and Richard Lenz. The alpha-Flow Use-Case of Breast Cancer Treatment – Modeling Inter-Institutional Healthcare Workflows by Active Documents. In *Proc of the 8th International Workshop*

on Agent-based Computing for Enterprise Collaboration (ACEC) at the 19th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2010), Larissa, GR, June 2010.

[NL12] Christoph P. Neumann and Richard Lenz. The alpha-Flow Approach to Inter-Institutional Process Support in Healthcare. *International Journal of Knowledge-Based Organizations*, 2, 2012. Accepted for publication.

[NSWL11] Christoph P. Neumann, Peter K. Schwab, Andreas M. Wahl, and Richard Lenz. alpha-Adaptive: Evolutionary Workflow Metadata in Distributed Document-Oriented Process Management. In *Proceedings of the 4th International Workshop on Process-oriented Information Systems in Healthcare (ProHealth'11) in conjunction with the 9th International Conference on Business Process Management (BPM'11)*, Clermont-Ferrand, France, August 2011.

[RBB+06] K. Reinhart, F. Brunkhorst, H. Bone, H. Gerlach, M. Grundling, G. Kreymann, P. Kujath, G. Marggraf, K. Mayer, A. Meier-Hellmann, and et al. Diagnose und Therapie der Sepsis. S-2 Leitlinien der Deutschen Sepsis-Gesellschaft e. V. (DSG) und der Deutschen Interdisziplinären Vereinigung für Intensiv- und Notfallmedizin (DIVI). *Anaesthesist*, 55 Suppl 1:43–56, 2006.

[RHH+03] N. Roeder, P. Hensen, D. Hindle, N. Loskamp, and H.-J. Lakomek. Instrumente zur Behandlungsoptimierung. *Der Chirurg*, 74:1149–1155, 2003.

[SAP] SAP. SAP Business Maps. `http://www.sap.com/solutions/businessmaps/c-businessmaps/index.epx`. *zuletzt abgerufen am 22.10.2011.*

[Sch04] Ken Schwaber. *Agile Project Management with Scrum.* Prentice Hall, 2004.

[Sch11] Peter Schwab. alpha-Adaptive: Ein adaptives Attributmodell als Baustein einer Prozessunterstützung auf Basis von aktiven Dokumenten. Diplomarbeit, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, 2011.

[Swe10] Keith D. Swenson. *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done.* Meghan-Kiffer Press, Tampa, FL, 1st edition, April 2010.

[TN11] Aneliya Todorova and Christoph P. Neumann. alpha-Props: A Rule-Based Approach to 'Active Properties' for Document-Oriented Process Support in Inter-Institutional Environments. In Ludger Porada, editor, *Lecture Notes in Informatics (LNI) Seminars 10 / Informatiktage 2011*. Gesellschaft für Informatik, March 2011.

[Tod10] A. Todorova. Konzeption und Implementierung eines leichtgewichtigen und autonomen Regel-basierten Systems als eine Realisierung von „Active Properties“ im Kontext von aktiven Dokumenten. Diplomarbeit, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, 2010.

[vdAvH04] Wil van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2004.

[WZJ] P Wohed, J Zdravkovic, and P Johannesson. A Universal Repository for Process Models. `http://people.dsv.su.se/~petia/Applications/VR07/VR07-main+Abstract.pdf`. *zuletzt abgerufen am 22.10.2011.*