TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Rechnernetze

Diplomarbeit

VISUALISIERUNG VON SERVICE-FRONTENDS IN EINEM WERKZEUG ZUR PRÄSENTATIONSORIENTIERTEN KOMPOSITION ANNOTIERTER DIENSTE

Lars Dannecker
Matrikelnummer: 3027320

Verantwortlicher Hochschullehrer:
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Betreuer (TU Dresden):
Dipl.-Inf Marius Feldmann

Betreuer (SAP Research):
Dipl.-Medieninf Tobias Nestler

Eingereicht am 29. Oktober 2009

SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Diplomarbeit zum Thema:

"Visualisierung von Service-Frontends in einem Werkzeug zur präsentationsorientierten Komposition annotierter Dienste"

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 29. Oktober 2009

Lars Dannecker

ZUSAMMENFASSUNG

Das Ziel des ServFace Projektes ist es Endnutzern ohne umfangreiche IT-Kenntnisse das einfache Erstellen von servicebasierten Anwendungen zu ermöglichen. Dazu wird ein Kompositionswerkzeug mit dem Namen "ServFace Builder" verwendet. Das Werkzeug erlaubt es Endnutzern, interaktive Anwendungen durch die Komposition von Service-Operationen zu erstellen.

Dafür ist eine grafische Repräsentation der Service-Operationen durch UI-Fragmente notwendig. Die UI-Fragmente werden im ServFace-Umfeld als Frontends bezeichnet. In der vorliegenden Arbeit wird ein Konzept zur automatischen Visualisierung dieser Frontends vorgestellt. Um das Ergebnis der Visualisierung zu verbessern, nutzt der Visualisierungsprozess neben der Service-Beschreibung weitere Informationen in Form von Annotationen und Gestaltungsempfehlungen.

Konkret werden die folgenden Aspekte in dieser Arbeit beschrieben:

- Visualisierung von Frontends zur Repräsentation von Service-Operationen auf Basis von Service-Beschreibungen, Annotationen, Plattformspezifikationen und Gestaltungsempfehlungen unter der Maßgabe der Gebrauchstauglichkeit und Nutzbarkeit.
- Integration der Frontends in die jeweils aktuelle Instanzen des gegebenen Anwendungsmodells.
- Technische Umsetzung und Evaluation der entwickelten Konzepte

ABSTRACT

The aim of the ServFace Project is to enable users with limited IT skills to easily create service-based applications. In order to do so, a tool called ServFace Builder has been developed. This tool allows users to build a composite application by combining several service operations.

An important part of the ServFace Builder is the graphical representation of those service operations through user interfaces. This thesis describes an approach to automatically generate user interfaces for service operations. To enhance the graphical representation, the user interface generation process of the ServFace Builder comprises annotations and design recommendations next to the common service descriptions to enhance the result of the generation process.

This thesis discusses:

- Graphical representation of service operations on the basis of service description, annotations, plattform specifications and design recommendations.
- Integration of the graphical representation into the actual instanz of the given application model.
- Implementation and evaluation of the presented concepts.

DANKSAGUNG

An dieser Stelle bedanke ich mich bei allen Personen, die mich während der Erarbeitung meiner Diplomarbeit unterstützt haben.

Den folgenden Personen spreche ich meinen besonderen Dank aus:

Meinen Betreuern Marius Feldmann und Tobias Nestler für die vielen, nützlichen Hinweise und anregenden Diskussionen zur Verbesserung meiner Arbeit. Außerdem vielen Dank für das Korrekturlesen meiner Diplomarbeit.

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill für die Möglichkeit meine Diplomarbeit an seinem Lehrstuhl zu schreiben.

Meinen Eltern Falk Dannecker und Jutta Dannecker für die dauerhafte moralische Unterstützung während der gesamten Diplomphase und die grammatikalische Korrektur der Diplomarbeit.

Gregor Herrmann und Jan Scheperski für die moralische Unterstützung, das Lesen meiner Diplomarbeit und viele nützliche Hinweise.

Andre Preußner für das Lesen und Korrigieren meiner Arbeit.

Meinen Kollegen von SAP Research Dresden für die gute Zusammenarbeit.

Dem ServFace Consortium für das interessante Projekt in dessen Rahmen diese Arbeit entstanden ist.

Vielen Dank!

INHALTSVERZEICHNIS

Teil I

Einleitung

1 EINFÜHRUNG IN DIE DIPLOMARBEIT

Webservices und serviceorientierte Architekturen (SOA) sind ein wichtiger Bestandteil der heutigen IT-basierten Geschäftswelt. Services bieten den entfernten, plattformunabhängigen Zugriff auf Funktionen und Geschäftsdaten über Lokal- und Weitverkehrs-Netze. Durch die Integration von Funktionalitäten verschiedener Webservices lassen sich komplexe Geschäftsprozesse und Workflows realisieren. Diese Technologien sind vorrangig zur Verarbeitung durch Computer entwickelt worden. Eine direkte Nutzung von Webservices durch Nicht-IT-Experten ist dabei nicht vorgesehen.

Mit der Entwicklung von Web 2.0 haben auch Endnutzer die Möglichkeit, an der Entwicklung von Webinhalten teilzunehmen. Blogs, soziale Netzwerke und interaktive Applikationen bieten einen ersten Weg der aktiven Beteiligung. Die Erstellung umfassender Webanwendungen und Business-Prozesse bleibt dem Endnutzer meist vorenthalten. Ein erster Schritt in Richtung der Webanwendungserstellung durch Endnutzer ist die Definition von Mashups. Diese erlauben eine einfache Kombination mehrerer Services zu einer interaktiven Anwendung. Bisherige Ansätze sehen allerdings meist nur die Kombination mehrerer Datenquellen vor und setzen in der Regel informationstechnische Kenntnisse voraus. Wünschenswert wäre eine grafische Möglichkeit der Komposition, ohne die Notwendigkeit der manuellen Dateneingabe und Logikdefinition.

Mit der Visualisierung und Komposition von Services beschäftigt sich das durch die EU geförderte Forschungsprojekt ServFace. Ziel des Projektes ist es Endnutzern ohne umfassende IT-Kenntnisse, die einfache Erstellung servicebasierter Anwendungen zu ermöglichen. Dafür wurde ein Kompositionswerkzeug entwickelt, dass die Erstellung solcher Anwendungen durch die Kombination von Service-Operationen erlaubt. Die Komposition wird nach dem "What you see is what you get"-Prinzip (WYSIWYG) durchgeführt. Dieses Prinzip suggeriert schon während der Anwendungserstellung das Erscheinungsbild der resultierenden Anwendung. Dazu ist es notwendig die Service-Operationen im Kompositionswerkzeug grafisch zu repräsentieren. Das geschieht durch automatisch visualisierte User-Interface(UI)-Fragmente die als Frontends bezeichnet werden. Die Visualisierung der Frontends erfolgt auf Grundlage der Service-Beschreibung, die im Rahmen dieser Arbeit als WSDL-Beschreibung vorliegt. Die Service-Beschreibung bietet allerdings nicht ausreichend Informationen um die Visualisierung nutzbarer User-Interfaces sicherzustellen. Deshalb werden die Services mit zusätzlichen UI-Informationen versehen. Diese zusätzlichen Informationen werden Annotationen genannt. In den Visualisierungsprozess werden neben der Service-Beschreibung die Annotationen einbezogen um das Visualisierungsergebnis zu verbessern. Zusätzlich werden Gestaltungsempfehlungen beachtet, durch die die Gebrauchstauglichkeit der Frontends sichergestellt werden soll. Die vorliegende Diplomarbeit beschäftigt sich mit der Konzeption des Visualisierungsprozesses für die grafische Repräsentation der Service-Operationen im Kompositionswerkzeug.

1.1 MOTIVATION

Die Generierung von Frontends für Webservices auf Basis von Webservice-Beschreibungen unter Einbeziehung zusätzlicher Informationen wurde bereits in wissenschaftlichen Untersuchungen betrachtet. Diese Untersuchungen konzentrieren sich aber meist auf die Generierung von User-Interfaces zum Ausführungszeitpunkt einer Anwendung. Der in dieser Arbeit vorgestellte Ansatz beschreibt allerdings die Visualisierung von Frontends für Service-Operationen zum Zeitpunkt der Anwendungserstellung. Besonders die Fokussierung auf Endnutzer ohne umfassende informationstechnische Kenntnisse hebt diese Arbeit von anderen Ansätzen ab. Das Konzept bezieht deshalb in den Visualisierungsprozess neben den technischen Beschreibungen auch umfangreiche funktionale und visuelle Zusatzinformationen mit ein. Durch diese Annotationen lässt sich das Visualisierungsergebnis verbessern und die Nutzbarkeit der visualisierten Frontends erhöhen. Auf Grund der Endnutzerzentrierung ist wichtig eine annehmbare Gebrauchstauglichkeit der Endanwendung sicherzustellen. Aus diesem Grund werden schon bei der Visualisierung der Frontends Design-Empfehlungen beachtet, die sich aus der Konsolidierung verschiedener Gestaltungsrichtlinien ergeben. Die meisten anderen Ansätze beachten die Gebrauchstauglichkeit nicht ausreichend. Zusätzlich soll die Ausführung der Anwendungen auf verschiedenen Plattformen möglich sein. Dazu müssen Plattformeigenschaften identifiziert werden, die Auswirkungen auf die Visualisierung der Frontends haben. Zusätzlich muss die Visualisierung entsprechend der Vorgaben der Zielplattform erfolgen.

1.2 ZIELE DER ARBEIT

Hauptziel der Arbeit ist die Entwicklung und prototypische Implementierung eines Konzepts zur Visualisierung von Frontends für Service-Operationen. Mit Hilfe dieser Frontends soll dem Nutzer eine grafische Komposition aus mehreren Service-Operationen ermöglicht werden. Die Visualisierung der Frontends soll dabei nicht nur auf die eigentliche Struktur, Definition und Beschreibung eines Services zurückgreifen, sondern auch zusätzliche Informationen wie Annotationen und Design-Empfehlungen beachten. Der aktuelle Zustand der modellierten Anwendung wird in einer Instanz des gegebenen Anwendungsmodells festgeschrieben. Durch Manipulation dieser Instanz werden die Frontends, zusätzlich zur eigentlichen Darstellung, in diese Datenstruktur integriert.

Grundlage für die Konzeption der automatischen Generierung ist eine Analyse von existierenden Konzepten für die automatische Generierung von Benutzerschnittstellen und die Betrachtung bestehender Gestaltungsrichtlinien. Weiterhin wird eine Kategorisierung der zusätzlich zur Dienst-Beschreibung zur Verfügung stehenden Annotationen nach ihren Auswirkungen auf die grafische Repräsentation und die zugrundeliegende Datenstruktur durchgeführt. Die innerhalb der Konzeption zu entwickelnde Annotationsauswertung soll eine Möglichkeit zur einfachen Erweiterung der unterstützten Annotationen aufweisen. Nach der Erstellung der Konzeption erfolgt prototypisch die Implementierung des Bindeglieds zwischen der grafischen Oberfläche und der zugrundeliegenden Datenstruktur, so dass der Import von annotierten Diensten in die Anwendung möglich ist. Alle erstellten Konzepte und Implementierungen werden im Laufe der Arbeit evaluiert.
Insgesamt lassen sich die Ziele der Arbeit wie folgt zusammenfassen:

1. Visualisierung von Frontends zur Repräsentation von Service-Operationen auf Basis von Service-Beschreibungen, Annotationen, Plattformspezifikationen und Gestaltungsempfehlungen unter der Maßgabe der Gebrauchstauglichkeit und Nutzbarkeit.
2. Integration der Frontends in die jeweils aktuelle Instanzen des gegebenen Anwendungsmodells.
3. Technische Umsetzung und Evaluation der entwickelten Konzepte

1.3 AUFBAU DER ARBEIT

Die Abbildung 1.1 illustriert den Ablauf und die Argumentationsfolge der Arbeit.

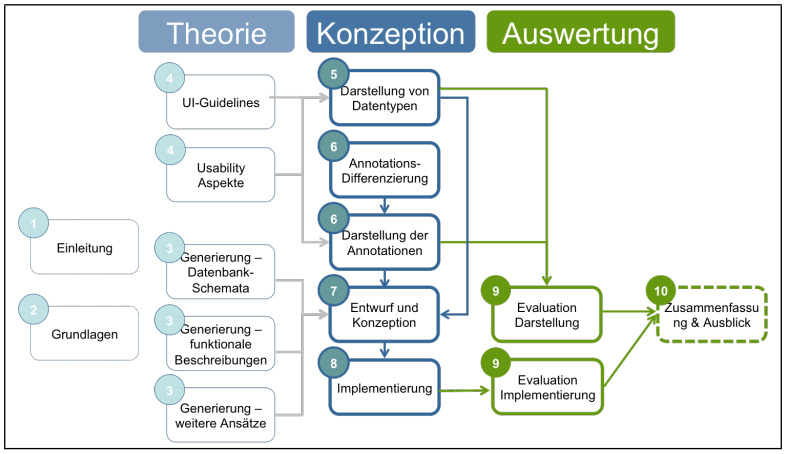


Abbildung 1.1: Ablauf und Argumentationsfolge der Diplomarbeit

Zur Erfassung aller Aspekte dieser Arbeit hat sich die Unterteilung in 10 Kapitel als günstig erwiesen. Zu Beginn wird in Kapitel 1 eine kurze Einführung in die Arbeit inkl. einiger Begriffsdefinitionen gegeben. Im zweiten Kapitel erfolgt die Einführung in wichtige themenspezifische Grundlagen. Anschließend werden existierende Konzepte zur Generierung von User-Interfaces aus Datenbanken-Schemata und funktionalen Beschreibungen betrachtet. Die Konzepte werden gegen aufgestellte Kriterien an die Visualisierung von Service-Frontends evaluiert und das zu entwickelnde Konzept von den vorhandenen Konzepten abgegrenzt. Die Betrachtung von Gestaltungsrichtlinien ist Inhalt von Kapitel 4. Auf Basis der Vorbetrachtung wird im folgenden Kapitel 5 die Art der Darstellung von Parametern in Abhängigkeit von ihren Datentypen festgelegt. Kapitel 6 widmet sich der Betrachtung, Kategorisierung und Abstraktion der Annotationen, sowie der Beschreibung der Auswirkungen auf das User-Interface und die Datenstruktur. Die Konzeption der Visualisierungskomponente findet in Kapitel 7 statt. Die Implementierung der Schnittstelle wird in Kapitel 8 beschrieben. In Kapitel 9 erfolgt die Evaluation der Gestaltungs- und Implementierungskonzepte. Zum Schluss wird die Arbeit im 10ten Kapitel zusammengefasst und entsprechende Schlüsse aus der Konzeption gezogen.

Teil II

Stand der Technik und Theoretische Betrachtung

2 RAHMENBEDINGUNGEN

Die vorliegende Arbeit wurde im Rahmen des ServFace Projektes erstellt. Im folgenden Kapitel sollen die Rahmenbedingungen erläutert werden, die sich aus dem ServFace Projekt ergeben. Die Rahmenbedingungen beeinflussen sowohl die Konzeption als auch die Implementierung der vorliegenden Arbeit. Zunächst wird das ServFace Projekt und die verwendete Methodologie für die Anwendungserstellung erläutert. Anschließend erfolgt die Vorstellung des Kompositionswerkzeuges. Dadurch wird der Visualisierungsprozess in den Gesamtrahmen des Projekts eingeordnet. Zum Abschluss werden die in diesem Werkzeug verwendeten Meta-Modelle erläutert. Diese Modelle dienen als Ein- und Ausgabe für die Visualisierung und stellen damit eine wichtige Vorbedingung dar.

2.1 DAS SERVFACE-PROJEKT

Ziel des ServFace Projektes ist es Endnutzer ohne umfassende IT-Kenntnisse die Erstellung servicebasierter Anwendungen zu ermöglichen. Der Fokus der Erstellung liegt dabei auf der visuellen Komposition verschiedener Service-Operationen. Für den Anwendungserstellungsprozess ist eine dreistufige Methodologie vorgesehen. Die Grundlage für die Erstellung interaktiver servicebasierter Anwendungen sind dabei Services, die um zusätzliche funktionale und visuelle Informationen erweitert wurden. In einem ersten Schritt der Methodologie werden deshalb diese als Annotationen bezeichneten Informationen zur Service-Beschreibung hinzugefügt. Dieser Schritt wird in einem speziellen Werkzeug zur Service-Annotation durchgeführt. Das Ergebnis sind annotierte Services, die als Eingabe für den zweiten Schritt der Methodologie verwendet werden.

Der zweite Schritt sieht die Komposition der interaktiven Anwendung vor. Dazu ist die Visualisierung der verwendeten Service-Operationen notwendig, um eine präsentationsorientierte Anwendungserstellung zu ermöglichen. Die Visualisierung, der als Frontends bezeichneten UI-Fragmente, wird auf Basis der WSDL-Beschreibungen der annotierten Services durchgeführt. Die Annotationen werden dabei verwendet, um das Visualisierungsergebnis zu verbessern. Zusätzliche Funktionen, wie beispielsweise eine automatische Vervollständigung für Formularfelder, können ebenfalls durch Annotationen spezifiziert werden. Eine direkte Ausführung der Funktionen zum Zeitpunkt der Anwendungserstellung ist allerdings nicht möglich. Der Endnutzer kann die visualisierten Frontends konfigurieren und durch die Festlegung von Datenflüssen miteinander kombinieren. Dabei interagiert er immer direkt mit den einzelnen Elementen der Frontends, was die präsentationsorientierte Anwendungserstellung des ServFace Projektes von anderen Ansätzen abhebt. Der aktuelle Zustand der modellierten Anwendung inkl. aller Frontends und Konfigurationen wird in einer Instanz des gegebenen Anwendungsmodells reflektiert.

Die Generierung einer lauffähigen Anwendung ist im letzten Schritt der Methodologie vorgesehen. Die Instanz des Anwendungsmodells der erstellten Anwendung dient als Eingabe für den weiteren Anwendungsgenerierungsprozess. Weiterführende Informationen zum ServFace Projekt und der Methodologie der Anwendungserstellung sind in [FJN+09] zu finden.

Die Abbildung 2.1 illustriert die Methodologie des Servface-Projekts.

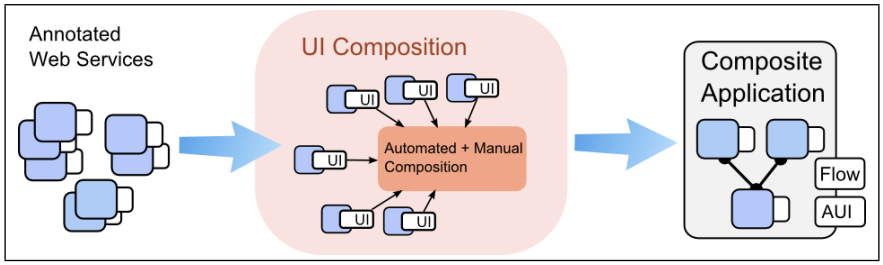


Abbildung 2.1: Illustration der ServFace Methodologie [Jug09]

2.2 DER SERVFACE BUILDER

Der ServFace Builder ist ein Kompositionswerkzeug zur Erstellung servicebasierter Anwendungen. Das Werkzeug ist für die Benutzung durch den unter A.5.2 definierten Endnutzer ausgelegt. Die Komposition der Webservices erfolgt präsentationsorientiert und ohne die Notwendigkeit Programmcode schreiben zu müssen. Der ServFace Builder ist an das Programm- und Interaktionsdesign von Microsoft PowerPoint angelehnt. Durch die Verwendung dieser PowerPoint-Metapher soll es neuen Anwender des Kompositionswerkzeugs ermöglicht werden, sich schnell in der Werkzeugumgebung zu Recht zu finden. Auf mehreren Seiten (im Werkzeug: Pages) kann der Nutzer die durch Frontends visualisierten Operationen verschiedener Services platzieren. Die Position auf der Seite ist innerhalb bestimmter Grenzen frei wählbar. Die Service-Operationen können durch die Festlegung von Datenflüssen miteinander verschaltet werden. Zusätzlich lassen sich Navigationselemente auf der Seite platzieren, um einen Übergang zwischen verschiedenen Seiten zu generieren. Das Ergebnis des Kompositionsprozesses ist eine Instanz des gegebenen Anwendungsmodells, die für den Anwendungsgenerierungsprozess verwendet wird.

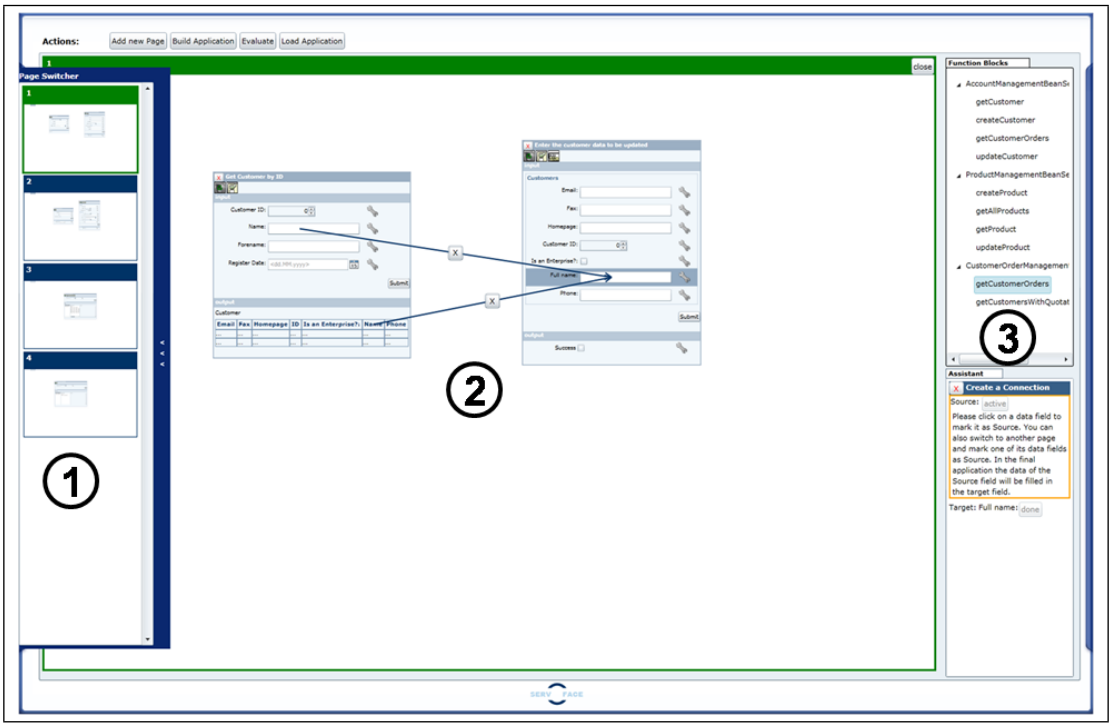


Abbildung 2.2: Screenshot des ServFace Builders

Die Grundansicht des ServFace Builder teilt den Bildschirm in drei Bereiche:

Die Seiten-Übersicht (in Abbildung - 1):
Ähnlich wie die Folien-Übersicht in Microsoft PowerPoint, werden in der Seiten-Übersicht alle Seiten (Pages) der Applikation inkl. einer abstrahierten Vorschau des Inhalts angezeigt.

Der Designbereich (in Abbildung - 2):
In der Bildschirmmitte befindet sich der Designbereich, auf welchem der Endnutzer die Kompositions durchführt. Dieser Bereich wird im Kompositionswerkzeug als "Modeling area" bezeichnet.

Der Service-Browser (in Abbildung - 3):
Im Service-Browser werden alle verfügbaren Services inkl. ihrer Service-Operationen angezeigt. Der Endanwender kann eine Operation aus dem Service-Browser in den Designbereich ziehen, um ein entsprechendes Frontend zu erzeugen. Dieses Frontend kann anschließend modifiziert und für die Komposition verwendet werden.

Das Kompositionswerkzeug ist für die Umsetzung in Microsofts Programmierumgebung für Webapplikationen "Silverlight 3.0" vorgesehen. Ähnlich wie Adobe Flash steht das Silverlight-Framework als Plug-In für viele Webbrowser verfügbar. Ein ausführliches Klassendiagramm des Kompositionswerkzeugs ist im Anhang A.2 als Abbildung A.1 zu finden. Ein erheblicher Teil der vorliegenden Arbeit befasst sich mit der Implementierung der Frontend-Visualisierung und -Darstellung für Service-Operationen innerhalb des Kompositionswerkzeugs. Deshalb erfolgt eine ausführliche Beschreibung dieser Aspekte und der Implementierung des Editors in den folgenden Kapiteln.

2.3 META-MODELLE IM KOMPOSITIONSWERKZEUG

Innerhalb des Kompositonswerkzeugs gibt es mehrere Meta-Modelle, die zur Beschreibung von Services, Annotationen und der erstellten Anwendung genutzt werden. Das Service-Modell wird für die Repräsentation der Services und das Annotationsmodell für die Beschreibung der Annotationen verwendet. Zur Beschreibung der erstellten Anwendung existiert ein Anwendungsmodell (Composite Application Model). Die genannten Modelle werden in den folgenden Abschnitten nur kurz vorgestellt. Detaillierte Beschreibungen sind in [FNH+09] und [JPS+09] zu finden.

2.3.1 Service-Modell und Service-Repository

Zur abstrakten Beschreibung von Services dient das Service-Modell. Dieses Modell wird zur Repräsentation und Zwischenspeicherung der Servicestruktur auf dem Client benötigt, da die Services auf einem Remote-Server in einem Service-Repository verwaltet werden. Im Gegensatz dazu ist das Kompositionswerkzeug als Client-Anwendung konzipiert. Sowohl die Struktur als auch die einzelnen Komponenten von Webservices lassen sich durch das Service-Modell beschreiben. Es enthält allerdings nur noch für die Komposition und spätere Anwendungsgenerierung relevante Service-Elemente. Das Service-Modell ist die Eingabe für die Visualisierung der Service-Operationen im Kompositionswerkzeug. Die notwendigen Informationen zur Erstellung einer Instanz des Service-Modells werden über die Service-Repository-API zur Verfügung gestellt, welche vom Kompositionswerkzeug genutzt wird. Die Informationen werden durch die Analyse der WSDL-Beschreibung eines Services gewonnen. Durch die Auswertung der in einer Instanz enthaltenen Komponenten eines Webservices lassen sich User-Interfaces zur Nutzung des Service generieren. Neben der Verwendung für die Generierung von User-Interfaces ist das Service-Modell auch Teil des Composite Application Model, um für die spätere Anwendungsgenerierung eine Beschreibung der Services zur Verfügung zu stellen. In der Abbildung 2.3 ist das Service-Modell dargestellt:

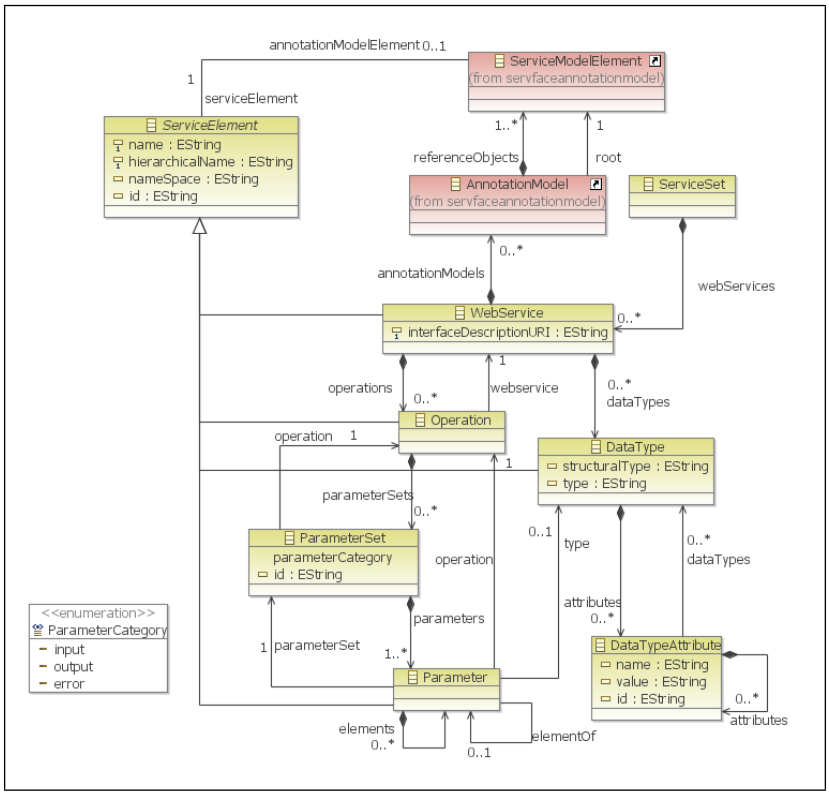


Abbildung 2.3: Das Service-Meta-Modell des Kompositionswerkzeugs

2.3.2 Composite Application Model (CAM)

Zur Beschreibung und Persistierung der mit dem Kompositionswerkzeug erstellten Anwendung wird das Composite Application Model verwendet. Das Anwendungsmodell dient auch als Eingabeformat für den weiteren Anwendungsgenerierungsprozess. Es beschreibt alle Aspekte der interaktiven Applikation: Aussehen, Art, Strukturierung und Verschachtelung der Elemente, sowie Funktions- und Seitenübergänge, Element- und Service-Operations-Beziehungen, Parameter-Verknüpfungen, Operationsaufrufe und Layoutdefinitionen. Jegliche Änderungen an der erstellten Anwendung werden im Anwendungsmodell umgesetzt. Zur Persistierung der erstellten Anwendung wird die jeweilige Instanz des Anwendungsmodells in eine XML-Datei serialisiert, welche bei einem Ladevorgang wieder in die Klassenstruktur der Modellinstanz überführt und im Editor dargestellt wird. Das Anwendungsmodell ist in mehrere Bereiche zerlegt, die verschiede Aspekte der Anwendung beschreiben. Eine komplette Übersicht über das Anwendungsmodell ist im Anhang A.3.1 als Abbildungen A.2, A.3 und A.4 zu finden.

Der Aufbau und die Struktur der Service-Frontends werden im View-Bereich des Modells beschrieben. Dieser Bereich ist in der Abbildung 2.4 dargestellt. Die Datenstruktur eines Service-Frontends besteht aus einer Menge von UI-Elementen, die in verschiedenen Ausprägungen im Modell vorkommen können. Jedes UI-Element wird dabei durch eine Menge von Parametern und Eigenschaften näher bestimmt und kann entsprechend der Definitionen verschiedene Unterelemente enthalten.

Durch seinen speziellen Aufbau ist das Anwendungsmodell für die Abbildung von strukturierten, verschachtelten, mehrseitigen Kompositionsanwendungen ausgelegt. Durch die Möglichkeit der Serialisierung des Modells in eine XMI-Struktur ist nicht nur eine lokale Speicherung des Modells möglich, sondern auch eine Rückübertragung zu einem Remote-Server denkbar. Auf diesem kann, auf Grund der meist höheren Rechenleistung eines Server-Systems, der Anwendungsgenerierungsprozess erfolgen.

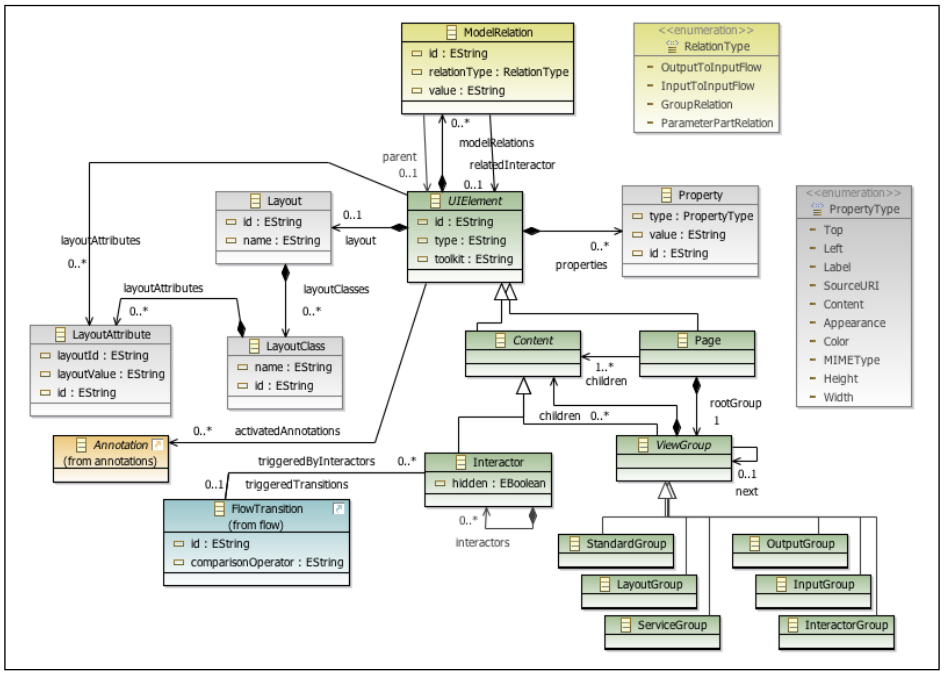


Abbildung 2.4: Composite Application Model: View-Bereich

2.3.3 Das Annotations-Modell

Die Qualität von User-Interfaces, die ausschließlich anhand einer Service-Beschreibungen erzeugt werden, ist nicht optimal. Insbesondere die Benennung der Elemente und die Darstellung der Service-Struktur können nicht ausreichend umgesetzt werden. Deshalb werden die Webservice-Beschreibungen mit zusätzlichen Informationen angereichert. Diese Annotationen werden in einem Modell beschrieben, welches ebenfalls Teil des Anwendungsmodells ist. Das Annotations-Modell umfasst eine spezifische Beschreibung für jede Annotation. Zu jedem Webservice gehört dabei eine eigene Instanz des Annotations-Modells. Die 2.5 stellt einen Auszug des Annotations-Modells dar. Sie illustriert dessen Grundstruktur.

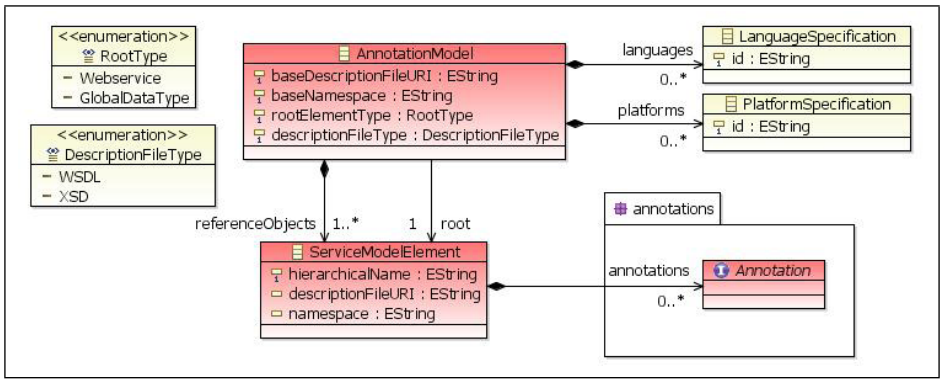


Abbildung 2.5: Darstellung des Annotation-Meta-Modell [JPS+09]

Das Service-Modell-Element (in Abbildung: ServiceModelElement) stellt die Schnittstelle zwischen dem Annotations- und Servicemodell dar. Jedes Service-Modell-Element repräsentiert jeweils ein Element aus dem Servicemodell und enthält alle Annotationen für dieses Element.

Eine Annotation wird durch die folgenden Parameter bestimmt:

- einen Namen
- Attribute der Annotation
- Referenzen auf Service-Modell-Elemente
- Referenzen auf Unterelemente
- Beziehungen und Abhängigkeiten zu anderen Annotationen

Die Abbildung 2.6 stellt als Beispiel die Struktur der Feedback-Annotation dar. Zwar implementieren alle Annotationen die abstrakte Klasse "Annotation", dennoch definiert jede Annotation ihre eigene Struktur. Ein generischer Zugriff auf alle Annotationen ist deshalb nicht möglich. Die Feedback-Annotation beispielsweise ist selbst als abstrakte Klasse vorgesehen, von der die beiden Feedback-Implementierungen "TextFeedback" und "MultimediaFeedback" erben. "TextFeedback" wird direkt als Text dargestellt. "MultimediaFeedback" enthält die Referenzadresse auf eine Multimedia-Datei. Der Feedback-Typ definiert die Art der Feedback-Annotation näher. Zusätzlich referenziert die Annotation Feedback auch Sprach- und Plattformspezifikationen, um mehrsprachige Texte mit Formatierung für unterschiedliche Plattformen zu gewährleisten.

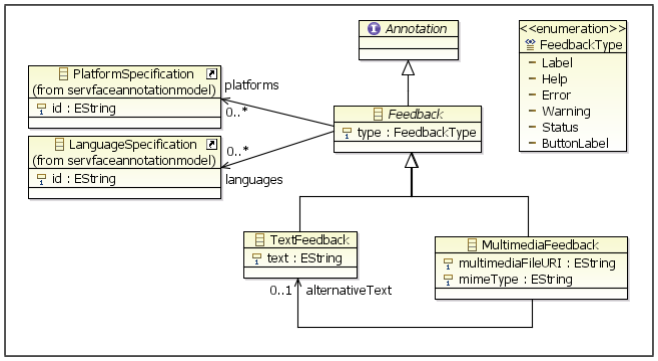


Abbildung 2.6: Klassenstruktur der Feedback-Annotation [JPS+09]

Als zweites Beispiel ist in Abbildung 2.7 die Suggestion-Annotation dargestellt. Anders als die Feedback-Annotation definiert die Suggestion keine zusätzlichen Parameter, sondern lediglich eine Menge von Referenzen. Entweder sind die Werte der Suggestion in einer weiteren Annotation festgelegt oder die Vorschläge entstammen einem speziellen Webservice. Es ist also möglich, sowohl eigene Parameter und Strukturen für eine Annotation zu definieren, als auch andere Annotationen oder Webservice-Elemente zu referenzieren[JPS+09].

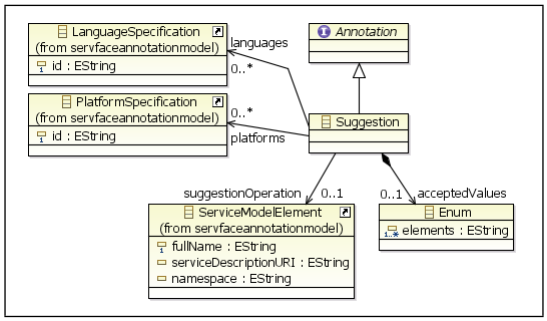


Abbildung 2.7: Klassenstruktur der Suggestion-Annotation [JPS+09]

Eine Übersicht über alle im ServFace-Projekt verwendeten Annotationen ist im Anhang A.3.2 zu finden. Nähere Informationen zu den Annotationen können [JPS+09] entnommen werden.

3 ÜBERBLICK ÜBER VERFAHREN ZUR AUTOMATISIERTEN GENERIERUNG GRAFISCHER USER-INTERFACES

Die automatische Generierung von User-Interfaces wird häufig verwendet, um einen schnellen Zugriff auf Datenstrukturen und Funktionen zu ermöglichen. Durch die automatische Generierung ist keine manuelle Programmierung notwendig. Das spart Zeit und verhindert Fehler. Als Basis dienen meist Meta-Modelle, Schemata oder funktionale Beschreibungen, die durch geeignete Techniken ausgewertet werden müssen. Im Laufe dieses Kapitels wird der aktuelle Stand der Technik in Hinblick auf die Generierung von grafischen User-Interfaces auf Basis von ER-Schemata und funktionalen Beschreibungen analysiert und bewertet. Dabei werden sowohl die grundlegenden Ansätze und Mechanismen kurz vorgestellt, als auch industrielle Produktentwicklungen präsentiert. Außerdem erfolgt eine Untersuchung, wie sich diese Konzepte auf die vorliegende Arbeit auswirken. Zunächst werden einige Anforderungen an die Darstellung der erzeugten User-Interfaces formuliert, um die in diesem Kapitel vorgestellten Ansätze gegen diese Anforderungen zu überprüfen.

3.1 ANFORDERUNGEN AN DEN VISUALISIERUNGSPROZESS UND DIE ERZEUGTE REPRÄSENTATION DER ZU GENERIERENDEN ANWENDUNG

In diesem Abschnitt werden Anforderungen an User-Interfaces in Hinblick auf die Gebrauchstauglichkeit formuliert. Die Anforderungen in diesem Abschnitt basieren auf bestehenden Gestaltungsrichtlinien und werden in Kapitel 4 um weitere Aspekte ergänzt. Anforderungen an den Funktionsumfang der erzeugten Anwendung werden in diesem Abschnitt nicht aufgestellt, da dieser maßgeblich von dem Funktionsumfang der beteiligten Webservices abhängig ist und auf die Visualisierung nur im Sinne der Komplexität Auswirkungen hat. Die Anforderungsanalyse in diesem Abschnitt dient auch als Grundlage für die weiterführende Analyse im Kapitel 7.

Das Kompositionswerkzeug ist ein im Rahmen des ServFace Projekts erstelltes Werkzeug zur Komposition von Webservice-Operation. Mit Hilfe dieses Werkzeugs soll die später generierte Endanwendung über eine grafische Oberfläche erstellt werden. Dazu ist es notwendig, schon zu Beginn des Kompositionsprozesses Frontends zur Repräsentation der Services zu visualisieren. An diese Visualisierung werden folgende Anforderungen gestellt:

Vollständigkeit:
Alle Daten zur Ausführung einer Funktion müssen eingegeben werden. Die Darstellung der Rückgabewerte muss ebenfalls gewährleistet sein.

Funktionen:
Funktionale Aspekte und daran beteiligte Elemente müssen erkennbar sein.

Konsistenz:
Die Darstellung der Ein- und Ausgabeelemente soll möglichst spezifisch und konsistent erfolgen.

Verständlichkeit:
Ein User-Interface muss möglichst klar strukturiert, verständlich und lesbar.

Zugehörigkeit:
Die Zugehörigkeit eines Elements zu einer Funktion muss eindeutig sein.

Beziehungen:
Beziehungen zwischen Funktionen müssen für den Nutzer des Kompositionstools erkennbar sein.

Sonderelemente:
Navigationselemente und Elemente, die eine Funktion auslösen, müssen gesondert gekennzeichnet sein.

Benutzbarkeit
Alle Elemente bieten eine gute Benutzbarkeit und verhindern Falscheingaben

Layout:
Die Anwendung sollte ein dem Einsatzgebiet angepasstes Layout aufweisen.

Anhand dieser formulierten Anforderungen werden die folgenden Konzepte betrachtet.

3.2 GENERIERUNG ANHAND VON ER-SCHEMATA

Für das Arbeiten mit einer Datenbank lassen sich aus den Informationen des zugrundeliegenden Entity-Relationship-Schemas Anwendungen mit einfachen grafischen Benutzeroberflächen erzeugen. Die einfachste Möglichkeit stellen dabei die so genannten CRUD-Anwendungen dar. CRUD ist ein Akronym und steht für Create, Read, Update, Delete und repräsentiert den Funktionsumfang der Anwendungen. Weitergeführt wird die Idee der UI-Erzeugung im Rahmen einer CRUD-Anwendung durch das Object Relation Mapping (ORM). Dabei werden die Relationen auf Objekte abgebildet. Aus dem resultierenden Modell kann dann ähnlich dem modell-basierten Ansatz eine grafische Benutzerschnitttelle erzeugt werden. Die in dieser Arbeit betrachteten User-Interfaces sind zur Ausführung von Services gedacht und basieren auf funktionalen Beschreibungen. Die automatische UI-Generierung für Datenbanken-Schemata ist deshalb nicht direkt relevant. Da aber auch bei diesen Ansätzen eine Schemainferenz im Mittelpunkt der UI-Generierung steht, können auch diese Konzepte wertvolle Informationen für diese Arbeit liefern. Innerhalb der nächsten Abschnitte wird die Erstellung von CRUD Anwendungen unter Verwendung von Inferenzmechanismen und das Object Relation Mapping beschrieben.

3.2.1 CRUD-Anwendungen

Entsprechend der Bedeutung des Akronyms haben CRUD-Anwendungen die Aufgabe, einfache Manipulationen an einer Datenbank auszuführen. Die Manipulationen umfassen das Auslesen und Darstellen von Daten (Read), das Aktualisieren und Löschen von Datensätzen (Update, Delete) und das Erstellen neuer Einträge in einer Datenbank (Create). CRUD-Anwendungen können dabei aus dem Schema einer Datenbank generiert werden. Sie bestehen meistens aus einer Reihe von miteinander verschalteten Formularen [BK07][JJ08]. Generell können CRUD-Anwendungen sowohl dem gesamten Datenbank-Schema genügen [BK07], als auch nur eine Menge von Anfragen repräsentieren [JJ08]. Dabei ist die Komplexität des zugrundeliegenden Schemas ausschlaggebend, welcher Ansatz eingesetzt werden sollte.

Im Allgemeinen wird die Erzeugung von Formularen als relativ einfach angesehen [SS07]. Zieht man allerdings die Gebrauchstauglichkeit mit in Betracht, ist die Erzeugung weniger trivial. Nicht alle Formulare sind in der automatisch erzeugten Form uneingeschränkt nutzbar. Gerade in Hinblick auf die Verständlichkeit und Übersichtlichkeit der erzeugten Formulare können Probleme auftreten [Sze96][Lie08]. Die Nutzbarkeit von Formularen wird unter anderem durch folgende drei Aspekte bestimmt: Eine hohe Prägnanz, eine gute Übersichtlichkeit und eine ausreichende funktionale Mächtigkeit [JJ08]. Deshalb mag zwar die generelle Formularerzeugung einfach sein, die Generierung von tatsächlich nutzbaren Formularen aus den zugrundeliegenden Daten ist allerdings deutlich schwieriger. Einige Meinungen gehen sogar soweit, dass die Gestaltung von User-Interfaces generell nicht komplett automatisiert durchgeführt werden sollte. Vielmehr wird empfohlen, die automatische UI-Erzeugung mit der Arbeit eines Designers zu kombinieren. Ziel ist dabei, hochqualitative Benutzeroberflächen zu erstellen, die unter anderem auch den in Abschnitt

3.1 formulierten Anforderungen genügen. Der Generierungsprozess sollte den eigentlichen Entwickler mit Vorschlägen und Alternativen versorgen, wobei dieser die endgültigen Entscheidungen trifft und die gegebenen Vorschläge und Alternativen annehmen oder ablehnen kann. Ein häufiger Ansatz für den beschriebenen Dialog ist das sogenannte Postprocessing. Die automatisch erzeugten User-Interfaces dienen dabei als erster Entwurf, welcher dann durch den Entwickler an sich verfeinert und an die Bedürfnisse der Endnutzer angepasst wird [Sze96]. Dieses Konzept verwendet man auch häufig bei der Erzeugung von CRUD-Anwendungen aus Datenbank-Schemata [BK07].

Der Ansatz von Akilesh Bajaj und Jason Knight (University of Tulsa, USA) sieht zur Erstellung einer CRUD-Anwendung einen Satz von Regeln vor, um die Objekte des Entity Relationship Schemas auf Objekte in dem erzeugten User-Interface abzubilden. Der Algorithmus zur Erstellung einer Nutzeroberfläche durchläuft die folgenden Schritte, um aus den Elementen des ER-Modells ein User-Interface zu erstellen [Sze96][BK07]:

1. Automatische Bestimmung der Präsentationseinheiten und damit der Fenster und der Informationen, die in jedem Fenster gezeigt werden.
2. Bestimmung der Verlinkung und der Navigation zwischen den Präsentationseinheiten.
3. Bestimmung von abstrakten Interaktionsobjekten der Präsentationseinheiten. Diese Objekte stellen die Elemente zur Repräsentation von Datenfeldern dar (bspw. Combobox).
4. Abbilden der abstrakten Interaktionsobjekten auf konkrete Interaktionsobjekte. Die konkreten Interaktionsobjekte beschreiben die Umsetzung der abstrakten Interaktionsobjekte (bspw. die Werteliste einer Combobox).
5. Bestimmen des Layouts der einzelnen Fenster. Dieses Layout definiert nur die Basiseigenschaften wie Schriftart, Positionierung und Größe. Weiterführende Layouts können an dieser Stelle nicht definiert werden. Allerdings lassen sich auch hier schon Regeln für die Lesbarkeit und Konsistenz des Layouts angeben.

Einen für diese Arbeit wichtigen Aspekt stellen die Gestaltungsrichtlinien dar. Diese können während des UI-Generierungsprozesses angewendet werden, um die Nutzbarkeit der erzeugten Benutzeroberflächen zu erhöhen. Unter Umständen fließen sie in die späteren Layoutentscheidungen der in dieser Arbeit beschrieben Service-Frontends ein. Dazu gehören [BK07]:

- Gruppierung ähnlicher Objekte (Ähnliche Funktion, gleicher Datentyp, etc.)
- Nutzung einer verständlichen Element-Benennung
- Nutzung von Farben
- Sicherstellung eines konsistenten Erscheinungsbild der erzeugten Formulare
- Klar erkennbare Aktionselemente und Programmausgänge
- Anbieten einer Navigation zwischen verschiedenen Programmelementen
- Undo-Funktionalität für Nutzeraktionen ermöglichen

Die Nutzung einer verständlichen Element-Benennung ist ein extrem wichtiger Punkt bei der Erstellung von nutzbaren User-Interfaces. Ohne eine lesbare Benennung kann ein Endanwender die Formulare nicht verstehen. Er müsste dazu das zugrundeliegende Daten-Schema kennen. Die Gruppierung und das Sicherstellen eines konsistenten Erscheinungsbildes sorgen dafür, dass sich der Nutzer innerhalb der jeweiligen Fenster der erzeugten CRUD-Anwendung zurecht findet

und Zusammenhänge erkennen kann. Beispielsweise sollten Primärschlüssel, die editierbare Parameter identifizieren, gruppiert werden. Aktionselemente müssen klar erkennbar sein und ebenfalls gruppiert angezeigt werden. Dazu zählt auch eine klar erkennbare Navigation zwischen den Fenstern. Das Ziel eines Navigationselements muss für den Endnutzer eindeutig sein. Für eine konsistente Darstellung sorgt vor allem die Festlegung, welche UI-Elemente für welche Modellelementtypen verwendet werden. Für Buttons mit gleicher Funktion ist auch das gleiche Design zu verwenden [BK07]. Die von Bajaj eingesetzten Gestaltungsrichtlinien sorgen dafür, dass einige der in Abschnitt 3.1 formulierten Anforderungen erfüllt werden. Allerdings wird gerade der wichtige Punkt der spezifischen Darstellung von Datentypen nicht betrachtet. Diese ist aber notwendig, um in einer erzeugten Anwendung die Eindeutigkeit von Eingabeelementen sowie deren Resistenz gegen Fehleingaben sicherzustellen.

Das folgende Beispiel soll den oben beschriebenen Ablauf verdeutlichen. Dazu wird zuerst ein einfaches Datenbank-Schema gezeigt (Abbildung 3.1) und anschließend die mit Hilfe des Algorithmus erzeugte grafische Benutzerschnittstelle(Abbildung 3.2) dargestellt.

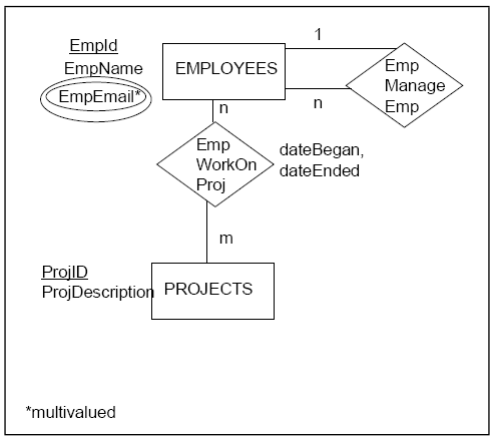


Abbildung 3.1: Einfaches Datenbank-Schema zur Erzeugung einer CRUD-Anwendung [BK07]

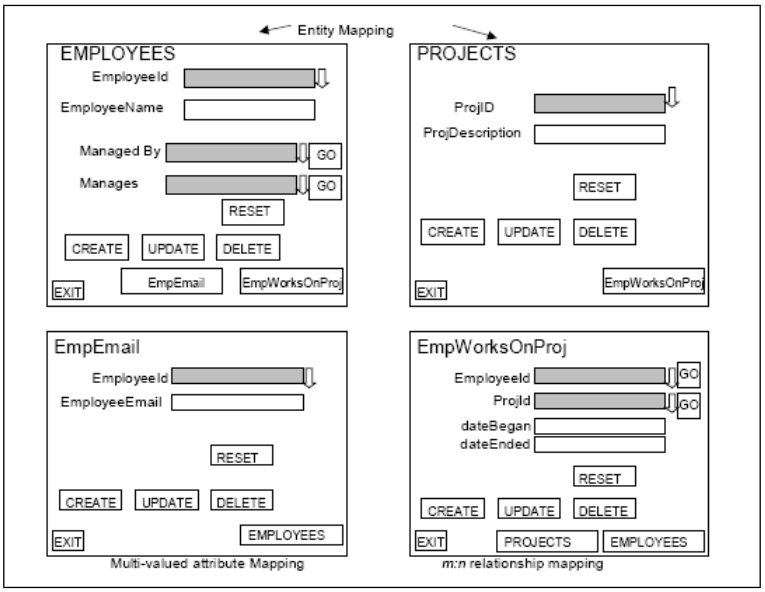


Abbildung 3.2: Ergebnis der automatischen CRUD-GUI aus dem Datenbankschema (Abbildung 3.1) [BK07]

Die hier (Abbildung 3.2) schematisch angedeuteten Formulare bieten die für CRUD-Anwendungen typischen Funktionen. Sie genügen dem Anspruch, diese Funktionen für das zugrundeliegende

Schema zur Verfügung zu stellen. Die Nutzbarkeit der Formulare ist sehr stark von der Lesbarkeit des eigentlichen Schemas abhängig, denn alle Benennungen des Schemas werden übernommen und in Label umgesetzt. In dem für das Beispiel gegebenen Schema sind alle Bezeichnungen so gewählt, dass ein einigermaßen lesbarer Satz an Formularen erzeugt wird. Sind die Namen im Schema allerdings nicht vorteilhaft gewählt, dann können die erzeugten Formulare komplett unlesbar und damit unbenutzbar werden. Weiterhin ist festzustellen, dass selbst für dieses einfache Schema, bestehend aus zwei Entitäten, zwei Relationen, einer überschaubaren Anzahl an Attributen etc., schon vier umfangreiche Formulare erzeugt wurden. Gerade bei großen und umfassenden Schemata würde eine unüberschaubar große Anzahl an komplexen Formularen entstehen. Das steht im Widerspruch zu den Anforderungen nach Struktur und Verständlichkeit aus dem Abschnitt 3.1.

Ein etwas anders angelegter Ansatz bei der Erzeugung von grafischen Benutzeroberflächen sieht deshalb neben der Analyse des Schemas auch die Analyse einer Menge von Anfragen vor. Dabei wird angenommen, dass die Unterstützung aller möglichen Anfragen auf ein Schema zu komplex ist. Ergebnis wäre eine zu große Menge an komplexen Formularen. Das würde wiederum die Nutzbarkeit der resultierenden Anwendung stark einschränken. Deshalb wird in dem Ansatz von Magesh Jayapandian und Haripriya Jagadish (University of Michigan, USA) die Anzahl der unterstützten Datenbankanfragen auf ein annehmbares Maß reduziert. Durch eine gezielte Analyse des Anfragesatzes und des Daten-Schemas wird sichergestellt, dass die potentiell wichtigsten und am häufigsten verwendeten Anfragen abgedeckt werden. Dabei wird die Komplexität der Formulare, also die Anzahl der Felder eines Formulars, durch die Festlegung von Schwellwerten beschränkt. Dadurch wird verhindert, dass zu komplexe, also nicht mehr lesbare Formulare entstehen. Die Evaluation des Verfahrens hat gezeigt, dass es möglich ist, einen vernünftigen Kompromiss zwischen der Anzahl unterstützter Anfragen und der Anzahl notwendiger Formulare sowie der Komplexität der Formulare zu finden [JJ08].Da dieses Verfahren vor allem darauf abzielt, mit möglichst wenig Formularen von möglichst geringer Komplexität möglichst viele Datenbankabfragen abzudecken, fehlen diesem Verfahren Richtlinien für das Layout. Die Verfeinerung des User-Interface in diesem Bereich wird komplett auf die manuelle Nachbearbeitung verschoben.

Relevant für den Verlauf der vorliegenden Arbeit ist besonders, dass die Verwendung von sehr komplexen Schemata in die Generierung nicht verwendbarer Anwendungen mündet. Grund dafür ist, dass die Anzahl der Formulare auf eine unüberschaubare Menge ansteigt und die Formulare so komplex werden, dass ein Verständnis deutlich erschwert wird. Weiterhin ist wichtig, dass man durch das Hinzuziehen und Analysieren von zusätzlichen Informationen die Komplexität und die Anzahl der Formulare reduzieren kann. Damit würden auch Verständlichkeit und Verwendbarkeit der Anwendung erhöht. Dies geschieht allerdings auf Kosten der Anzahl unterstützter Datenbankabfragen. Das ist bei sorgfältiger Analyse der Abfragewahrscheinlichkeit akzeptierbar [JJ08].
Für beide vorgestellten Verfahren gilt, dass die erzeugten Formulare im Regelfall einer Überarbeitung bedürfen, um tatsächlich nutzbaren Anwendungen zu erstellen. Zu den Schritten der Überarbeitung zählen unter anderem [BK07]:

- Erstellen von Menüs und Navigationsbereichen.
- Zusammenfassen von mehreren Fenstern, wenn diese durch einen Prozess zusammengehören.
- Entfernen unerwünschter oder nicht benötigter Attribute.
- Hinzufügen zusätzlicher Informationen oder Design-Elemente, um die Anwendung verständlicher für den Endanwender zu gestalten.
- Hinzufügen von grafisch aufbereiteten Informationen.
- Prüfung und Änderung von Benennungen, um Eindeutigkeit zu erreichen.

Die meisten der oben genannten Aspekte sind auch für die Präsentation der Service-Frontends wichtig.

Zusammenfassend lässt sich festhalten, dass die Erzeugung von einfachen Anwendungen inkl. eines nutzbaren User-Interfaces auf Basis von Relationen möglich ist. Gerade die nach dem Ansatz von Bajaj generierten CRUD-Anwendungen genügen durch die Nutzung von Layout-Richtlinien vielen Anforderungen der Visualisierung im Kompositionswerkzeug (Abschnitt 3.1). Eine Nachbearbeitung ist dennoch zu empfehlen.

3.2.2 Object-Relation-Mapping (ORM)

Grundgedanke des Object-Relation-Mapping(ORM) ist das Zusammenbringen objektorientierter Programmierung mit relationalen Datenbanken, um eine Persistierung der Objekte zu ermöglichen [BS98][Amb03]. Es ist allerdings nicht nur möglich, die Objekte aus einem objektorientierten Modell auf relationale Objekte eines Datenbankschemas abzubilden, sondern diese Abbildung auch umzukehren und damit aus einem relationalen Schema ein objektorientiertes Modell zu erhalten. Aus einem solchen Modell lässt sich dann neben dem funktionalen Code einer Applikation auch ein Benutzerinterface generieren. Als Beispiel zur Verdeutlichung dieses Verfahrens, soll das QCodo Projekt dienen. QCodo ist ein "Rapid Application Development (RAD)" Tool, das auf Basis von ORM aus einem relationalen Datenmodell ein Objekt-Modell und eine Basis CRUD-Anwendung erstellen kann. Die Entwickler des QCodo Frameworks weisen allerdings selbst darauf hin, dass die erzeugten Applikationen zwar alle Basisfunktionalitäten zur Verfügung stellen, aber eher für das Prototyping als für das Erstellen von wirklich nutzbaren Anwendungen gedacht sind [Ho09]. Dennoch ist die Betrachtung der erzeugten CRUD-Anwendungen inkl. der generierten Benutzerschnittstellen interessant. Sie bietet die Möglichkeit, die Mächtigkeit dieser UI-Generierung einzuschätzen und mit den Anforderungen an die Visualisierung im Kompositionswerkzeug (Abschnitt 3.1) zu vergleichen. Das QCodo Framework geht nach dem in Abbildung 3.3 dargestellten Ablauf vor:

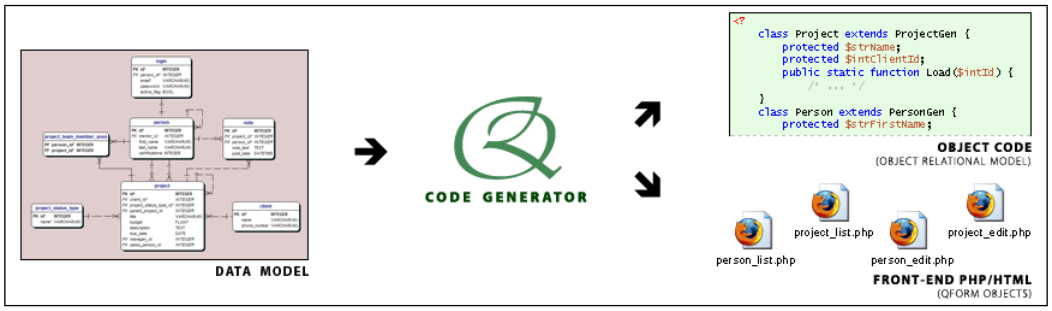


Abbildung 3.3: Schematische Darstellung QCodo Framework; Codegenerierung nach ORM Prinzip [Ho09]

Das übergebene Schema wird analysiert, das Datenmodell und auch erste Formulardateien werden erzeugt. Die Funktionalität und das Design der Formulare wird in der Abbildung 3.4 illustriert:

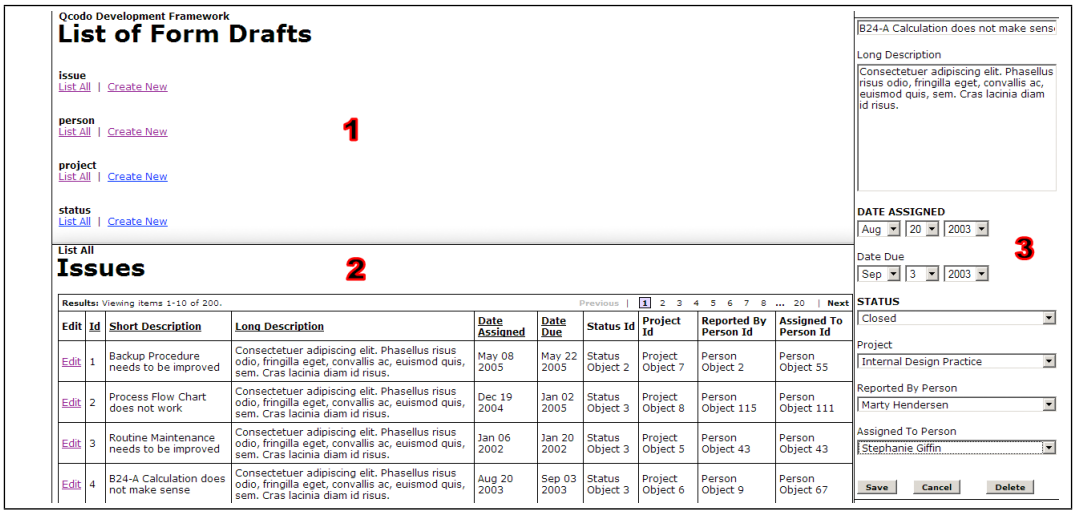


Abbildung 3.4: Durch den QCodo Code Generator erzeugte Formularkomponenten [Ho09]

Der mit 1 markierte Teil der Abbildung stellt das Übersichtsmenü der erzeugten Formulare dar. Von hieraus kann man jedes Formular inkl. der mit 2 gekennzeichneten Tabelle erreichen. Diese stellt die Basisübersicht der vorhandenen Daten zur Verfügung. Durch das Navigieren über den "Edit"- Link erreicht man den mit 3 nummerierten Ausschnitt der Abbildung. Der dortige Teil des Formulars stellt die CRUD-Funktionalitäten zur Verfügung.

Positiv ist zu bewerten, dass das gesamte Layout strukturiert und verständlich wirkt. Das liegt unter anderem an dem Vorhandensein eines Basismenüs, in welchem alle Formulare verfügbar sind. Weiterhin ist die Datentabelle klar strukturiert, so dass nicht alle vorhandenen Daten auf einer Seite angezeigt werden müssen. Vielmehr wird nach einer übersichtlichen Anzahl an Einträgen die Tabelle auf einer weiteren Seite fortgesetzt. Die Forderungen nach Struktur und Verständlichkeit der Anwendung (Abschnitt 3.1) ist damit weitgehend abgedeckt. Die Seiten sind über ein Menü am Kopf und am Fuß der Tabelle zu erreichen. Die Sortierfunktion der Tabelle ist in der Abbildung 3.4 nicht sichtbar. Mit ihr lässt sich die Tabelle ab- oder aufsteigend nach einer Spalte sortieren. Eine Sortierung ist nach allen Spalten möglich, solange die Spalte nicht ein Fremdschlüsselattribut repräsentiert. Ebenfalls nicht sichtbar sind die "Create New Issues"- und "Back to Form Draft Menu"- Links, da diese unter der Tabelle sitzen. Durch diese Menüstruktur wird ansatzweise der Anspruch einer klaren Struktur erfüllt. Da die Menüelemente allerdings nur einfache Links sind und die Sortierfunktion optisch überhaupt nicht zu erkennen ist, wird die Erkennbarkeit von Funktions- und Navigationselementen noch nicht in ausreichendem Maße umgesetzt. Das über den "Edit"-Link zu erreichende Formular ist dem aus den erzeugten CRUD-Anwendungen sehr ähnlich. Allerdings fließt bereits eine einfache Datentypauswertung mit ein. Datumseingaben werden über spezielle Eingabefelder (siehe "Date Assigned" im Bereich 3 der Abbildung 3.4) erfasst und längere Texte über mehrzeiliges Textfeld. Die Fremdschlüsselbeziehungen werden durch Drop-Down-Listen repräsentiert. Die Visualisierung der Datentypen ist damit differenzierter und spezifischer als bei den in Abschnitt 3.2.1 erzeugten CRUD-Anwendungen.

Eine manuelle Nachbearbeitung der generierten User-Interfaces wird auch bei dem QCodo-Ansatz empfohlen. Denn bei diesem gibt es ebenfalls das Problem der Element-Benennung. Ohne eine sinnvolle Benennung im Schema ist auch die Anwendung letztendlich nicht nutzbar. Deutlich wird das durch die Anzeige der Fremdschlüsselspalten im Bereich 2 der Abbildung 3.4. Dort werden Referenzen zu Objekten repräsentiert, aber keine verständlichen Namen angezeigt. Dies rührt daher, dass der Codegenerator ohne das Eingreifen des Entwicklers nicht festlegen kann, was von einem Objekt angezeigt werden soll. Die Änderung der Problematik ist relativ einfach und kann durch das Ändern von einer Zeile Code pro Formular behoben werden [Ho09].

Zusammenfassend lässt sich beim QCodo ORM-Ansatz sagen, dass die erzeugten Formulare gut strukturiert sind und es eine Art Basisdesign gibt. Dadurch weist die Anwendung eine gute Verständlichkeit auf. Viele der in Abschnitt 3.1 formulierten Anforderungen werden durch die

automatisch erzeugten User-Interfaces bereits realisiert. Die Erfüllung vieler Kriterien des automatischen Visualisierungsprozesses ist für die Erzeugung der Nutzeroberflächen im Rahmen dieser Arbeit sehr hilfreich.

Viele Entwicklertools, die Rapid Application Development (RAD) unterstützen, versuchen durch die automatische Erzeugung von Code und Basis-User-Interfaces den Entwicklern die Arbeit zu erleichtern. Der vorgestellte ORM-Ansatz wird dabei unter anderem von "Ruby on Rails" [TH06], "Subsonic" [Sha07][Tes08] und der "Netbeans" IDE [Mic05] unterstützt.

3.2.3 Zusammenfassung

Die Erzeugung von Nutzerinterfaces aus ER-Schemata wird vor allem eingesetzt, um die Programmierung durch die Schaffung von Rahmenapplikationen zu erleichtern und Fehler bei der Interpretation von Datenbank-Schemata zu minimieren. Aus diesem Grund ist der Funktionsumfang sehr gering und entspricht bei jedem betrachteten Generierungsansatz der üblichen "Create - Read - Update - Delete" - Funktionalität. Dennoch erfüllen die Algorithmen bei der Generierung der Benutzerschnittstellen einen großen Teil der Anforderungen, die an den Visualisierungsprozess gestellt worden (Abschnitt 3.1).

Ähnlich wie bei den vorgestellten Ansätzen, sieht das Visualisierungskonzept dieser Arbeit ebenfalls die Auswertung von Schemata vor. Dabei handelt es sich allerdings um Service-Beschreibungen und nicht um Datenbank-Schemata. Dennoch sind die Auswertungsansätze vergleichbar. Die UI-Generierung, der vorgestellten Konzepte, basiert ausschließlich auf der Auswertung der Datenbank-Schemata. Zusätzliche Informationen etwa zur Verbesserung des Visualisierungsergebnisses oder zur Erweiterung des Funktionsumfangs werden nicht verwendet. Die Gebrauchstauglichkeit der generierten User-Interfaces ist daher beschränkt und hängt im hohen Maße von der Qualität des zugrundeliegenden Schemas ab. Im Gegensatz dazu sieht der Visualisierungsansatz dieser Arbeit die Verwendung von Annotationen vor. Diese enthalten zusätzliche Informationen, die bei der Auswertung der Service-Beschreibung zur Verbesserung des Visualisierungsergebnisses verwendet werden. Dadurch lässt sich eine bessere Gebrauchstauglichkeit der Frontends im Kompositionswerkzeug und ein höherer Funktionsumfang in der resultierenden Anwendung erreichen.

3.3 GENERIERUNG ANHAND VON SERVICE-BESCHREIBUNGEN

Neben der Möglichkeit der Auswertung von Datenbank-Schemata können auch funktionale Beschreibungen dazu verwendet werden, um automatisch Benutzerschnittstellen zu erzeugen. Als Beispiel soll in diesem Abschnitt die automatische Erzeugung von User-Interfaces für Webservices dienen. Jeder Webservice wird durch ein auf der Webservice Description Language (WSDL) basierendes Modell beschrieben. Dieses Modell beschreibt alle Bestandteile eines Webservice inkl. seiner Operationen, Parameter und Datentypen und eignet sich damit als Basis für die Erzeugung von Benutzeroberflächen.

3.3.1 Einfache Inferenzmechanismen

Um die notwendigen Elemente für die Generierung einer Nutzeroberfläche zu ermitteln, werden im Bereich von Webservices Inferenzmechanismen eingesetzt. Unter einem Inferenzmechanismus versteht man ein Verfahren aus dem Bereich der Logik, mit dessen Hilfe man aus formalen Beschreibungen oder formalen Modellen Schlussfolgerungen ziehen kann [BHS98]. Mit

einem solchen Verfahren lassen sich beispielsweise aus WSDL-Beschreibungen die Webservice-Operationen inklusive deren Komponenten und zugehörigen Informationen gewinnen, die für die Generierung einer Benutzerschnittstelle notwendig sind. Die dafür besonders wichtigen Bestandteile einer WSDL-Beschreibung sind:

- Operationen des Webservice
- Eingabe- und Ausgabeparameter der Operation
- Parameterdatentypen

Ähnlich wie bei der automatischen Erzeugung von User-Interfaces aus Datenbanken-Schemata kann man an Hand der gewonnenen Informationen recht einfach formularbasierte User-Interfaces generieren [SBS07]. Der Grad der Nutzbarkeit der anfänglich generierten Benutzerschnittstellen ist auch bei diesem Verfahren stark von der Qualität der Quellinformationen und der Art der Parameterdarstellung abhängig. Generell lassen sich alle Parameter durch stringbasierte Formularelemente darstellen: Eingabeparameter als Textboxen und Ausgabeparameter als Textfelder. Das führt zu einer schlechten Lesbarkeit durch mangelnde Differenzierung. Zudem erhält der Nutzer unbegrenzte Freiheiten bei der Eingabe von Parametern, so dass Fehleingaben vorprogrammiert sind. Stattdessen ist es sinnvoll, die zusätzlich vorhandenen Informationen einzubeziehen und die Wahl der UI-Elemente abhängig vom jeweiligen Datentyp der Parameter zu machen. Parameter vom Datentyp "Boolean" könnten beispielsweise als Checkbox oder Parameter mit einem eingeschränkten "Decimal"-Datentyp als Stepper dargestellt werden. Dies würde erstens für eine differenziertere Darstellung des Formulars sorgen und damit die Lesbarkeit erhöhen und zweitens Fehleingaben durch Beschränkung der Eingabefreiheit vermeiden. Das erhöht die Benutzbarkeit deutlich [SBS07][Lie08].

Bei der Einbeziehung semantischer Aspekte stoßen die Inferenzmechanismen jedoch an ihre Grenzen. Besonders bei den Problemstellungen der Benennung der Elemente oder der Unterscheidung unterschiedlicher Eingabemöglichkeiten in Textboxen sind die Informationen der WSDL nicht ausreichend. Das verhindert, dass durch den Inferenzmechanismus eine sinnvolle Entscheidung getroffen werden kann. Als Namen für die Elemente des generierten User-Interfaces, werden häufig die Bezeichnungen der Komponenten aus der WSDL-Beschreibung verwendet. Da WSDL eher zur Maschinen-Maschinen-Kommunikation und nicht zur Maschinen-Mensch-Kommunikation entwickelt wurde, gibt es für diese Bezeichnungen keine Benennungs-Richtlinen. Die Benennung innerhalb einer WSDL-Beschreibung dient eher zur Identifikation der Webservice-Komponenten und ist nicht für die Darstellung beim Endnutzer ausgelegt. Die Sinnhaftigkeit der Bezeichnungen ist deshalb vom Entwickler des Webservice abhängig und kann zu unverständlichen Bezeichnungen der Formularelemente führen [Lie08].

Textboxen gibt es in verschiedenen Ausprägungen und mit vielen Anzeigemöglichkeiten. Beispielsweise ist es sinnvoll, für längere Texte ein mehrzeiliges Textfeld oder für die Eingabe eines Passworts die verdeckte Darstellung zu verwenden. Allerdings ist eine Differenzierung nicht durch die Betrachtung des Datentypen möglich, da für alle Darstellungs- und Anzeigemöglichkeiten der generische String-Datentyp verwendet wird. Um diese Sachverhalte zu umgehen, ist das Hinzuziehen von weiteren Informationen notwendig [SBS07]. Ein einfacher Inferenz-Mechanismus scheitert an vielen der in Abschnitt 3.1 aufgestellten Anforderungen. Es können lediglich folgende Anforderungen sichergestellt werden:

- *Vollständigkeit*
- *Sonderelemente*

Bei der Verwendung eines starken Inferenzmechanismus ist die Erfüllung der folgenden Anforderungen möglich:

- *Konsistenz*
- *Benutzbarkeit* (in beschränktem Maß)

3.3.2 GUI Deployment Descriptor

Eine Möglichkeit, weitere Informationen zusätzlich zu einer WSDL-Beschreibung zur Verfügung zu stellen, ist der "Graphical User-Interface Deployment Descriptor (kurz: GUIDD)". Der GUIDD ist ein XML-Dokument, das neben der WSDL-Beschreibung zusätzliche abstrakte Beschreibungen zur Verfügung stellt. Mit Hilfe dieser zusätzlichen Informationen können viele der Nachteile, der durch einen Inferenzmechanismus erstellten Nutzerinterfaces, behandelt werden. Die abstrakten Beschreibungen eines GUIDD-Dokuments sind dabei nicht an plattformspezifische Formate wie beispielsweise HTML gebunden. Sie können durch eine entsprechende Transformation in verschiedene Formate umgewandelt werden.

In der aktuellen Version 0.99.5 der GUIDD Spezifikation [KS06] besteht das XML-Dokument aus einem Header und vier weiteren Bereichen. Der Header ist ein standard XML-Header. Die 4 Bereiche des XML-Bodys sind im Wurzelelement Deployment enthalten. Schematisch ist der GUIDD-Aufbau in Abbildung 3.5 auf Seite 27 dargestellt. Jeder der 4 Bereiche ist in Kindelemente vom gleichen Namen in Einzahl unterteilt und beschreibt die folgenden Aspekte:

- **operations:** Allgemeine Beschreibung der Seitenerstellung
- **formComponents:** Beschreibung der Eingabeelemente
- **outputTypes:** Beschreibung der Ausgabeelemente
- **stylesheets:** Definition von CSS Stylesheets, die bei der Generierung einer Website verwendet werden können.

Der GUIDD wird als Zusatzinformationsquelle im WSGUI Ansatz verwendet, welcher in Abschnitt 3.3.3 auf Seite 27 beschrieben wird.

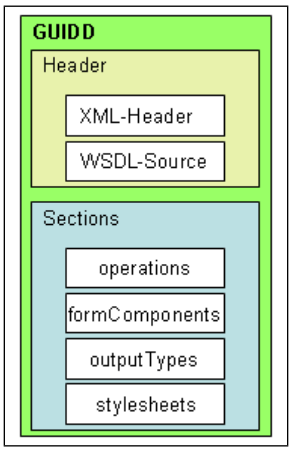


Abbildung 3.5: Aufbau des GUIDD nach Version 0.99.5 der GUIDD Spezifikation [KS06]

3.3.3 Webservice Graphical User-Interface - WSGUI

Um die in Abschnitt 3.3.1 genannten Probleme zu lösen, wurde von Michael Kassoff (Stanford University, UK) der "Webservice Graphical User-Interface" (kurz WSGUI) - Ansatz entwickelt [KKM03]. Der Ansatz nutzt zusätzlich zur WSDL-Beschreibung eines Webservice einen GUI-Deployment-Descriptor 3.5 sowie ein "eXtensible Stylesheet Language" (kurz: XSL) - Stylesheet, welches das "Look and Feel" beschreibt. Die abstrakten Beschreibungen des GUIDD und der WSDL

werden zusammengeführt und mit Hilfe eines "Look and Feel"-Stylesheets per XSL-Transformation (kurz: XSLT) in konkrete Ausgabeformate transformiert. Anschließend kann eine Darstellung auf dem Bildschirm erfolgen. Zu beachten ist, dass an dieser Stelle die manuelle Nachbearbeitung und Konfiguration der UI-Generierung beginnt. Dadurch grenzt sich dieses Konzept von der komplett automatischen Benutzerschnittstellen-Erzeugung ab.

Neben der Möglichkeit der Vergabe von verständlichen Namen für die UI-Elemente werden unter Zuhilfenahme dieser eben genannten Zusatzinformationen im WSGUI-Ansatz folgende Konzepte beschrieben[KKM03]:

- **Formular Komponenten:** Definition abstrakter Formularkomponenten, welche die Eigenschaften und Darstellung der WSDL-Elemente beschreiben.
- **Formular Navigation:** Entwicklern wird die Möglichkeit gegeben Elemente einzubinden, mit denen der Nutzer Einfluss auf die eigentliche Darstellung des Formulars hat. Kassoff führt als Beispiel die Möglichkeit an, den Endnutzer wählen zu lassen, wie er Daten eingeben möchte. Möglichkeiten sind beispielsweise die direkte Eingabe, die Auswahl aus einer Liste oder die Auswahl anhand von einschränkenden Kriterien.
- **Listen variabler Größe:** Der Nutzer kann die Anzahl der Elemente einer Liste selbst bestimmen. WSGUI übernimmt dabei automatisch die Behandlung von Minimum- (minOccurs) und Maximum-Schranken (maxOccurs) und bietet Möglichkeiten, Elemente zu einer Liste hinzuzufügen oder zu entfernen, nur entsprechend dieser Einschränkungen an.
- **Dynamische Enumeration:** Dieses Konzept sieht vor, dass ein Nutzer immer aus einer Liste gültiger Werte eine Auswahl trifft und keine eigenständige Eingabe mehr tätigen muss. Dadurch sollen insbesondere Fehleingaben vermieden und die Merkfähigkeit des Endnutzers entlastet werden. Für die Generierung dynamischer Enumerationen muss der Webservice eine entsprechende Funktion oder der Entwickler ein XML-Stylesheet, welches die Enumeration definiert, zur Verfügung stellen.
- **Virtuelle Operationen:** Ermöglicht eine oder mehrere Operationen aus einer WSDL-Beschreibung zu kombinieren und so eine neue virtuelle Operation zu erstellen.

Diese Ansätze dienen letztendlich der Erzeugung nutzerfreundlicher User-Interfaces.

Innerhalb der WSGUI-Engine wird nun die WSDL-Beschreibung mit dem GUIDD kombiniert und ein von Kassoff als "Screen Data" bezeichnetes Zwischenformat erzeugt. Anschließend wird auf Basis des XSL-Stylesheets eine XSL-Transformation durchgeführt, um beispielsweise ein HTML Formular zu erzeugen. Innerhalb dieses Generierungsprozesses wird versucht, die WSGUI-Konzepte anzuwenden und somit ein nutzerfreundliches Interface für einen Webservice zu erstellen [KKM03]. Das Listing 3.1 stellt einen Ausschnitt aus dem ScreenData Format dar.

```
<screen xmlns="http://.../2002/10/wsgui">
 <operation name="orderFood"
  <prettyName>Order Food</prettyName>
   <description>This is for ordering food</description>
   <input name="id1001">
    <label>Adress</label>
    <help>Please enter an adress.</help>
   </input>
   <list>
   <item>
    <select1 appearance="minimal" dynamicEnumeration="foodid" name="id1003">
     <label>Food Item</label>
      <item>
       <label>Choose a food</label>
        <value />
      </item>
      <item>
       <label>Hamburger (3.50)</label>
       <value>food001</value>
      </item>
      <item>
      ...
      </item>
      <help>Please select a Food.</help>
     </select1>
    </item>
    <trigger type="add" name="id1004"/>
   </list>
  <submit name="id1005">
  <label>Invoke</label>
  </submit>
 </operation>
</screen>
```

Listing 3.1: Ausschnitt aus dem ScreenData Zwischenformat [KKM03]

Gut zu erkennen ist die Vergabe von lesbaren Namen über den Tag <prettyName>, die Erstellung der dynamischen Enumeration innerhalb des <select>-Tags und das Hinzufügen von zusätzlichen Hilfsinformationen über den <help>-Tag. Die erzeugten User-Interfaces genügen einigen der in Abschnitt 3.1 formulierten Anforderungen. Besonders die Sicherstellung der Benennung (Anforderung: *Benutzbarkeit*), die Identifizierung aller Parameter (Anforderung: *Vollständigkeit*) und die Darstellung des Ausführungselements (Anforderung: *Sonderelemente*) sind wichtige Anforderungen, die erfüllt werden.
Das von Kassoff vorgeschlagene Konzept von WSGUI wurde von Josef Spillner in [SBS07] aufgegriffen und überarbeitet. Dabei wurden einige der Konzepte aus [KKM03] übernommen, einige entfernt und weitere Konzepte hinzugefügt. Mehrere Konzepte wurden auf Grund des hohen Aufwands für den Ersteller des GUIDD und des XSL-Stylesheets entfernt. Gründe dafür sind, dass die Beschreibung eher innerhalb der Webservice-Beschreibung vorgenommen werden sollte oder weil neuere Ansätze die alten Konzepte besser oder gleichwertig bei geringerem Aufwand erfüllen konnten. Hinzugefügt wurden die folgenden Konzepte [SBS07]:

- **Starke Inferenz:** Der Einsatz eines erweiterten Inferenzmechanismus soll die automatisch erzeugte Generierung direkt verbessern.
- **Beachtung von Mime-Typen:** Zur direkten Darstellung von Daten anstelle des Quellcodes. Beispielsweise kann durch die Angabe eines Mime-Typen direkt das Bild anstelle seines Quellcodes präsentiert werden.
- **Nutzerauswahl-Konzept:** Wie im Abschnitt 3.3.1 beschrieben, ist die Bestimmung der Darstellung bei generischen Datentypen wie Strings schwierig. Das Nutzerauswahl-Konzept sieht vor, dass, wenn keine Beschreibung des Elements im GUIDD erfolgt, der Nutzer die Wahl der Darstellungsform übernimmt.

- **Vorlagen (Templates):** Um eine konkrete Darstellung und Positionierung der Formularkomponenten passend zum umgebenden Layout zu ermöglichen, können Vorlagen definiert werden, die die Spezifikation der Darstellung übernehmen.

Neben der Verbesserung der Darstellung der visuellen Elemente und damit auch der besseren Erfüllung der Anforderungen (bspw. die Anforderung *Layout* durch Verwendung von Templates), wird durch den starken Inferenzmechanismus eine spezifischere Typisierung ermöglicht (Anforderung: *Konsistenz*) und damit Fehleingaben verhindert (Anforderung: *Benutzbarkeit*).
Die vorgestellten Ansätze sind zum Großteil in das Projekt Dynvocation [Spi06] eingeflossen, das die theoretischen Grundlagen praktisch implementiert.

3.3.4 Implementierungen der vorgestellten Ansätze

Sowohl für Kassoffs WSGUI als auch für den erweiterten Ansatz von Josef Spillner existieren Implementierungen. Der ursprüngliche WSGUI-Ansatz wurde in der WSGUI-Engine [KKM03] und der Ansatz von Josef Spillner [Spi06] im Dynvoker umgesetzt. In diesem Abschnitt sollen beide Implementierungen kurz erläutert werden.

3.3.4.1 WSGUI-Engine:

Die WSGUI-Engine arbeitet als eine Art universeller, generischer Webservice-Client. Dieser fungiert als Proxy zwischen dem eigentlichen Client des Endnutzers und dem Webservice. Auf der einen Seite generiert die Engine HTML-User-Interfaces für den Endnutzer und auf der anderen Seite wird die Kommunikation mit dem Webservice übernommen. WSGUI nimmt einen GUIDD sowie optional ein XSL-Stylesheet als Input. Um mit dem Webservice zu interagieren, stellt der Endnutzer unter Angabe des GUIDD eine entsprechende Anfrage an die WSGUI-Engine. Über die in dem GUIDD enthaltene Webservice-Adresse bezieht die WSGUI-Engine die WSDL-Beschreibung. Anschließend werden auf Basis der vorliegenden Informationen HTML-Formulare generiert. In diese werden die für den Service-Aufruf notwendigen Parameter vom Endnutzer eingegeben und an die WSGUI-Engine übermittelt. Die WSGUI-Engine leitet die Daten als SOAP-Nachrichten an den Webservice weiter und empfängt anschließend auch die Antwort vom Webservice. In der Engine erfolgen auch die Aufbereitung der Daten und die Erzeugung von HTML-Code zur Darstellung auf dem Client. Dieser Ablauf ist schematisch in Abbildung 3.6 dargestellt. Wie in der theoretischen Betrachtung in Abschnitt 3.3.3 erläutert, erfolgt die Generierung der Benutzerschnittstellen durch einen Inferenzmechanismus, der die WSDL-Beschreibung und den GUI Deployment Descriptor als Eingabe nimmt. Das Ergebnis im Zwischenformat Screendata wird durch eine XSL-Transformation auf Basis eines XSL-Stylesheets in den konkreten Code für das finale, generierte User-Interface umgewandelt [KKM03].

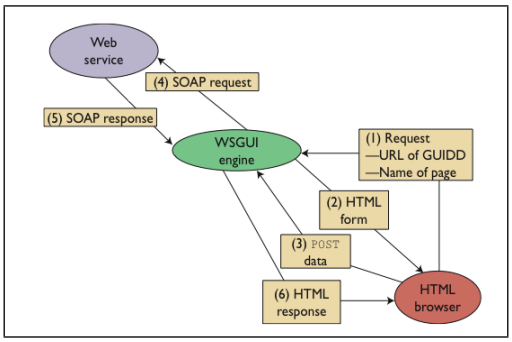


Abbildung 3.6: Ablauf der Webservice Interaktion mit der WSGUI-Engine [KS06]

3.3.4.2 Dynvoker:

[SBS07] griff die Ideen des WSGUI-Konzept auf und verfeinerte diese. Umgesetzt wurden die Konzepte im Rahmen des Projekts Dynvokation [Spi06] mit der Implementierung des Dynvokers. Der Dynvoker ist als ein Servlet implementiert, welches dem Endnutzer den dynamischen Zugriff auf beliebige Webservices erlaubt. Unter Nutzung von zusätzlichen Informationen können die resultierenden grafischen Benutzerschnittstellen optimiert werden. Ziel ist ähnlich wie bei der WSGUI-Engine die Nutzung von Webservices durch eine verständliche, grafische Benutzeroberfläche zu vereinfachen. Als Eingabe akzeptiert der Dynvoker Beschreibungen in verschiedenen Formaten, unter anderem:

- XSD
- WSDL
- OpenSearch
- WADL

Die Webservice-Beschreibungen können mit zusätzlichen Informationen über die User-Interfaces und die Semantik kombiniert werden. Als Ergebnis werden grafische Benutzerschnittstellen für die funktionellen Services erstellt. Der Dynvoker unterstützt als Ausgabevarianten:

- XForms
- XHTML
- Eclipse XML Web Standard Tools (geplant, Aufruf von Webservices aus Eclipse, siehe: [Jäg06])
- GUI4CWS (Behandlung der Kopplung mehrerer Webserviceoperationen, siehe: [Ble06])
- Command Line Tool (direkte Tool Anbindung oder Dynvoker Runner)

Genutzt wird der Dynvoker ähnlich wie die WSGUI-Engine. Der Einstiegspunkt ist eine vom Dynvoker automatisch generierte XHTML-Seite, die ein XForms-Formular integriert. Als Grundlage für die Generierung der Startseite dient ein XSD-Schema, das um GUIDD-Beschreibungen erweitert wurde. In das Formular muss die Adresse zur WSDL-Beschreibung und optional der passende GUIDD für den gewünschten Webservice eingegeben werden. Nach Auswahl der entsprechenden Operation erzeugt der Dynvoker das Eingabeformular, in welches der Endnutzer die notwendigen Parameter zum Aufruf der Webservice eingeben kann. Nach Abschluss der Eingaben wird das Formular an den Dynvoker übergeben. Dieser generiert daraus einen SOAP-Webservice Aufruf und übermittelt diesen an den Webservice. Die Antwort wird ebenfalls als SOAP-Message an den Dynvoker gesendet. Dort wird sie aufbereitet und eine für den Endnutzer lesbare Ausgabe generiert. Die Beschreibung bezieht sich auf das Ausgabeformat XForms bzw. XHTML [Spi06].
Abbildung 3.7 stellt schematisch den Aufbau des Dynvokers dar. Neben dem Dynvoker Core gibt es 4 Abstraktionsebenen:

- Webservice: Eingabe in den Dynvoker. Abstraktion von den verschiedenen Beschreibungsformaten für Webservices inkl. weitere Informationen zur Semantik und dem zu erzielenden grafischen User-Interface.
- Aufruf: Abstraktion des Aufrufs von verschiedenen Webservice-Typen (RESTful, SOAP)
- Laufzeit: Kopplung des Dynvokers. Abstraktion der Anbindungsmöglichkeit des Dynvokers an verschiedene andere Implementierungen.

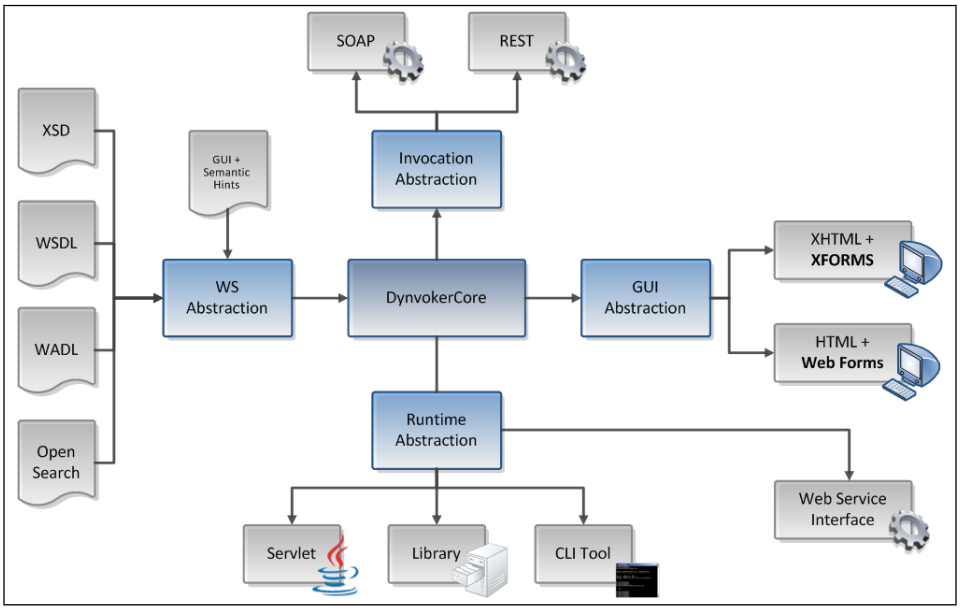


Abbildung 3.7: Schematischer Aufbau des Dynvokers [Spi06]

- Benutzerschnittstelle: Ausgabe des Dynvokers. Abstraktion über die verschiedenen Darstellungsformate der generierten Formulare.

Die Abstraktionsebenen sollen den Kern des Dynvokers unabhängig machen von konkreten Implementierungen für Ein- und Ausgabeformate. Sie stellen eine Sammlung von Adaptern dar. Diese nehmen die notwendigen Transformationen für die Ein- und Ausgabeformate des DynvokerCores vor. Beispielsweise wird das Eingabeformat OpenSearch nicht direkt interpretiert, sondern vorher im entsprechenden Adapter zu einer WSDL transformiert [SFB+08][SBS07][Spi06].

3.3.5 Weitere Ansätze

In diesem Abschnitt werden kurz weitere Ansätze zur automatischen Generierung von Benutzerschnittstellen angesprochen.

Webservice Experience Language (WSXL): WSXL ist ein von IBM entwickeltes Komponentenmodell für interaktive Webapplikationen basierend auf Webservices. Als Hauptziele nennt IBM die Verbreitung interaktiver Webapplikationen über unterschiedliche Verteilungswege und die Erstellung neuer Applikationen durch das wirksame Einsetzen bestehender Webapplikationen. Dafür implementieren die Komponentenservices [ACG+02]:

- Basisoperationen zum Lifecycle-Management
- Möglichkeiten zur Dateneingabe
- Generierung von Präsentationselementen

Da WSXL sowohl plattform- als auch darstellungsunabhängig ist, lassen sich die durch WSXL erstellten Applikationen auf verschiedenen Wegen präsentieren. IBM führt in der 2002 erschienen Spezifikation die direkte Darstellung im Browser, die Integration in ein Portal oder die Einbettung in andere Webapplikation als Beispiel an [ACG+02]. Die Erstellung neuer Applikationen soll

durch die nahtlose Kombination und Adaption bestehender Webapplikation erfolgen, die durch diesen Verbindungsprozess einen neuen Nutzen erhalten. Um dies zu erreichen, sieht die WSXL Spezifikation den Export von einer oder mehreren Operationen einer Webapplikation vor, die genügend Informationen zur Integration, Aggregation und Adaption bereitstellt. Dadurch werden Gestaltungsaufgaben separiert, die in Kombination die Erstellung einer Webapplikation schwierig machen würden. WSXL basiert auf etablierten Standards wie XForms und XLink [ACG+02].

Webservice User-Interface (WSUI): WSUI ist ein Komponentenmodell, dass mehrstufige und präsentationsorientierte Interaktionen zu SOAP- und XML basierten Netzwerk-Services hinzufügt [Cov02]. Das Ziel von WSUI ist es, Webservices dem Endnutzer zugänglich zu machen. Dabei setzt WSUI auf bekannte Standards wie XSLT, XPath und XHTML. Ähnlich wie bei WSGUI (Abschnitt 3.3.3) werden die Webservices mit Präsentations- und Interaktionsinformationen kombiniert. Das Komponentenmodell von WSUI soll diese Kombination spezifizieren und standardisieren. Die Komponenten lassen sich dann zur Laufzeit dynamisch in Container-Applikationen auf einer Webseite integrieren, welche dann nur als Webservice in Erscheinung treten. Die Kernkonzepte von WSUI sind:

- Komponenten und Container
- Pages und Views
- Events und Interaktionsabläufe
- Komponenten-Views unter Berücksichtigung von XSLT
- Variablen und Ausdrücke

Eine Container-Applikation ist die auf der Webseite zu integrierende Basis-Applikation, die sich aus mehreren Komponenten-Applikationen zusammensetzt. Diese sind dann für die Endnutzer-Interaktion, das Ausführen der Service, das Erzeugen von User-Interfaces usw. verantwortlich [Cov02]. Die Möglichkeit, Webservices und deren Komposition dem Endnutzer zugänglich zu machen, ist auch ein Ziel des Kompositionswerkzeugs. Deshalb ist die Betrachtung der WSUI-Konzepte in Verbindung mit den WSGUI-Konzepten auch für diese Arbeit interessant sind.

Webservice for Remote Portlets (WSRP): WSRP sind plattformunabhängige, von der OASIS standardisierte Interfaces zum Zugriff auf und zur Interaktion mit interaktiven, präsentationsorientierten Webservices. Besonderes Augenmerk wird dabei auf Portal Server gelegt, die Funktionalitäten als Portlets in ihre Portale integrieren können. Der Standard geht davon aus, dass der Webservice-Ansatz ungünstig für die dynamische Integration von Businessanwendungen und Inhalten innerhalb einer Plug&Play-Lösung ist. Ziel ist deshalb, Webservice-Funktionalitäten als eine Sammlung von Portlets zur Verfügung zu stellen, die ohne großen Aufwand in ein Portal integriert werden können. Webservices stellen im Regelfall nur reine Daten zur Verfügung, ohne präsentationsorientierte Aspekte zu betrachten. Die Auswertung und anschließende Präsentation der Daten wird vom Portal übernommen, welches die Daten verschiedener Webservices den Portal-Nutzern zur Verfügung stellt. Für jeden zu integrierenden Webservice muss ein spezifischer Adapter zur Verfügung gestellt werden, der die Anfrage beim Webservice tätigt und an das entsprechende Portlet des Portals weitergibt. Das Portlet übernimmt nun die Auswertung der Daten und die Erstellung der visuellen Repräsentation. Abbildung 3.8 stellt diesen Ablauf schematisch dar. Das Erstellen eines Adapters für jeden Webservice und die Implementierung der Logik zur Auswertung der gelieferten Daten ist sehr aufwendig. Deshalb sieht der WSRP-Standard eine Veränderung dieses Ablaufs vor. Der WSRP-Server stellt sowohl die Daten als auch die Präsentation der Daten als Portlet zur Verfügung. Auf Clientseite ist nur noch ein Proxy-Objekt eingerichtet, das die Integration der entfernten WSRP-Portlets in das Portal vereinfacht. Dieser Ablauf ist in Abbildung 3.9 dargestellt [ST03]. Im Sinne dieser Arbeit ist WSRP vor allem durch die Möglichkeit interessant, verschiedene Services zu kombinieren.

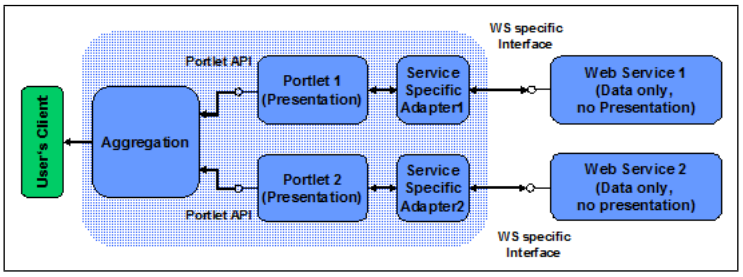


Abbildung 3.8: Darstellung der klassischen Portal <–> Webservice Integration [ST03]

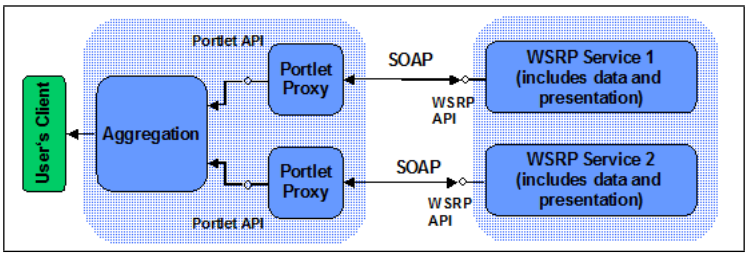


Abbildung 3.9: Darstellung der WSRP Portal <–> Webservice Integration [ST03]

Einfluss auf die vorliegende Arbeit: Die Auseinandersetzung mit den genannten Ansätzen liefert interessante Aspekte, die eine Betrachtung im Rahmen dieser Arbeit sinnvoll machen. Alle Ansätze stellen eine Möglichkeit dar, die Nutzung und Interaktion von Webservices durch den Endnutzer zu verbessern. Der WSXL-Ansatz beschreibt Möglichkeiten zur Integration verschiedener Service-Operationen, um dadurch eine neue Anwendung zu erstellen. Auch innerhalb des ServFace Builder werden interaktiven Anwendungen durch die Kombination von Frontends verschiedener Services modelliert. WSXL weist allerdings keine direkte Endnutzer-Zentrierung auf und bietet nur beschränkte grafische Bearbeitungsmöglichkeiten. Eine Ausrichtung auf den Endnutzer bietet der WSUI-Ansatz, die direkte Nutzung von Webservices durch den Endnutzer zum Ziel hat. WSUI sieht eine Kombination von Webservice-Beschreibungen mit zusätzlichen Präsentations- und Interaktionsinformationen vor. Die daraus erzeugten User-Interfaces lassen sich auf eine Webseite integrieren und durch den Endnutzer verwenden. Auch eine Kombination mehrerer Webservices innerhalb einer Container-Anwendung ist im WSUI-Ansatz vorgesehen. Im Gegensatz zu dem in dieser Arbeit beschriebenen Ansatz, sieht WSUI allerdings eine Generierung der User-Interfaces und die Integration der Webservices zur Laufzeit vor. In dieser Arbeit wird ein Ansatz beschrieben, der die UI-Generierung und die Komposition verschiedener Webservices in die Design-Time der Anwendung verlagert und dem Nutzer die Erstellung der Komposition überlässt.

WSRP fügt sich ebenfalls in dieses Gebilde, da der Ansatz ein Interface für die Integration und Interaktion verschiedener präsentationsorientierter Webservices durch wenige Mausklicks zum Ziel hat. Ein präsentationsorientierter Ansatz wird auch im Rahmen dieser Arbeit betrachtet. Allerdings wird innerhalb des Kompositionswerkzeugs nicht mit Portlets gearbeitet. Auch wenn die vorliegende Arbeit in einigen Bereichen andere Wege als die vorgestellten Konzepte geht, liefern die Konzepte dennoch wertvolle Informationen für die weitere Betrachtung.

3.4 ZUSAMMENFASSUNG UND ABGRENZUNG

Zusammenfassend lässt sich feststellen, dass bei den beiden betrachteten Generierungsquellen:

- Datenbank-Schemata
- Service-Beschreibungen (in diesem Fall Webservice-Beschreibungen)

eine Generierung von nutzbaren User-Interfaces möglich ist.
CRUD-Anwendungen aus Datenbank-Schemata haben nur einen sehr geringen Funktionsumfang und beschränken sich auf die Bereitstellung der Datenbank-Interaktionen. Die generierten Benutzerschnittstellen erfüllten dabei einige Anforderungen aus Punkt 3.1. Das Konzept von Bajaj [BK07] sieht sogar die Beachtung von Gestaltungsrichtlinien zur Verbesserung der Gebrauchstauglichkeit vor. Die meisten Anforderungen erfüllt der QCodo-Ansatz [Ho09]. Zu beachten ist, dass alle vorgestellten Konzepte lediglich das Datenbank-Schema als Quellinformation verwendet haben und im Gegensatz zu den Webservice-Ansätzen keine zusätzlichen UI- oder Funktions-Informationen zur Verfügung hatten. Demnach ist die besondere Abhängigkeit der erzeugten grafischen Benutzerschnittstellen von der Qualität der zugrundeliegenden Datenbankschemata zu erwähnen. Eine manuelle Nachbearbeitung ist gerade bei der Benennung der UI-Elemente bei allen Ansätzen notwendig gewesen. Das stellt allerdings kein Problem dar, denn im Bereich der UI-Generierung aus Datenbank-Schemata steht die Erstellung von Rahmenanwendungen zur Vereinfachung der Entwicklung und Vermeidung von Fehlern im Vordergrund. Die manuelle Umgestaltung und Verfeinerung der Rahmenanwendung ist explizit vorgesehen.

Im Gegensatz dazu liegt bei Webservices der Fokus auf dem Schaffen einer Nutzungsmöglichkeit von Webservices für Endnutzer ohne umfangreiches IT-Wissen. Dazu sind grafische User-Interfaces notwendig, welche die funktionellen Befehle der Webservices in visuelle Repräsentationen und damit in für Menschen lesbare Abbildungen umwandeln. Der einfachste Ansatz basiert auf der Auswertung der WSDL-Beschreibung mit einem Inferenzmechanismus. Das Verfahren identifiziert visualisierbare Elemente aus dem formalen Modell eines Webservices. Es hängt, ähnlich wie die Ansätze bei der Nutzung von Datenbankschemata, stark von der Qualität der zugrundeliegenden Beschreibung ab. Es existieren beispielsweise keine Benennungsrichtlinien. Somit kann die Lesbarkeit der Elementbezeichnungen im generierten Layout nicht sichergestellt werden. Um dieses Manko auszugleichen, verwenden Ansätze wie WSGUI und Dynvoker zusätzliche Informationen, die die Les- und Nutzbarkeit der erzeugten Layouts gewährleisten. Die mit diesen Ansätzen erzeugten User-Interfaces stellen im Regelfall Formulare zur Ausführung der Webservices dar. Die entsprechenden Implementierungen der Ansätze fungieren dabei als eine Art Proxy, der die Verwendbarkeit der Informationen zwischen Mensch und Webservice sicherstellt. Die erzeugten User-Interfaces erfüllen mehrere der in 3.1 aufgestellten Anforderungen, allerdings nur unter Nutzung von zusätzlichen Informationen. Der Visualisierungsansatz in dieser Arbeit greift das Konzept der User-Interface Generierung unter Nutzung von zusätzlichen Informationen auf, um die Nutzbarkeit der Frontends zur Repräsentation von Service-Operationen zu erhöhen. Da innerhalb des Kompositionswerkzeugs die präsentationsorientierte Erstellung interaktiver Anwendungen durch Endnutzer vorgesehen ist, wurden die verwendeten zusätzlichen Informationen deutlich erweitert [FJN+09]. Neben der Beeinflussung der Darstellung werden auch zusätzliche Funktionalitäten und Informationen zur Zusammenarbeit von Service-Operationen beschrieben.

Die Tabelle 3.1 illustriert die Einschätzung, in welchem Maß die wichtigsten der vorgestellten Konzepte die in Abschnitt 3.1 aufgestellten Anforderungen erfüllen. Damit wird die Zusammenfassung des Kapitel 3 abgeschlossen. In der Tabelle sind die Spalten mit den Kurznamen der Konzepte und die Zeilen mit den Überschriften der Anforderungen bezeichnet.

Der Grad der Erfüllung ist durch die folgenden drei Symbole:

- "+" (Voraussetzung wird erfüllt)
- "◦" (Anforderung wird teilweise oder nur unter bestimmten Voraussetzungen erfüllt)
- "-" (Erfüllung der Voraussetzung kann nicht sichergestellt werden)

gekennzeichnet.

Konzepte:	CRUD[BK07]	CRUD[JJ08]	Qcodo	Inferenz	WSGUI	Dynvoker
Anforderungen:						
Vollständigkeit	+	+	+	+	+	+
Funktionen	-	-	-	-	-	-
Konsistenz	-	-	◦	◦	-	◦
Verständlichkeit	+	-	+	-	◦	◦
Zugehörigkeit	+	-	+	-	+	+
Beziehungen	-	-	-	-	-	-
Sonderelemente	+	◦	◦	+	+	+
Benutzbarkeit	◦	-	◦	-	◦	+
Layout	-	-	-	-	◦	◦

Tabelle 3.1: Evaluation der vorgestellten Konzepte hinsichtlich der aufgestellten Anforderungen

4 DESIGN-REGELN FÜR SERVICE-FRONTENDS

"Don't make me think (frei: Lass mich nicht überlegen...)" ist der Leitspruch von Steve Krug bei der Gestaltung von einfach zu nutzenden Web-Publikationen und Anwendungen [Kru05]. Gemäß diesem Leitspruch propagieren er und viele andere Design-Entwickler die Gestaltung von einfachen und intuitiv verständlichen Benutzeroberflächen. Dieses Motto soll auch für die Service-Frontends gelten, die im Rahmen dieser Arbeit visualisiert werden. Deshalb ist die Aufstellung entsprechender Design-Regeln notwendig. UI-Gestaltung ist allerdings eine Thematik, bei der die Meinungen weit auseinander gehen und ein Konsenz nur in Ansätzen existiert. Im Rahmen dieses Kapitels werden daher die folgenden Design-Guidelines studiert:

- Steve Krug - Don't Make Me Think!: A Common Sense Approach to Web Usability [Kru05]
- James Hobart - Principals of Good GUI Design [Hob95]
- Apple Developer Connection - Apple Human Interface Guidelines [ADC09]
- Microsoft Developer Network - Windows User Experience Interaction Guidelines [MSD09]

Bei der Auswahl der UI-Guidelines wurde darauf geachtet, einen möglichst breiten Rahmen abzudecken. Mit [Kru05] und [Hob95] sind zwei Guidelines von Usability-Experten vertreten. Wohingegen [ADC09] und [MSD09] aus der Feder großer Software- und Betriebssystem-Konzerne stammen. Zusammengenommen bieten die Guidelines ein breites Spektrum an Meinungen einmal aus der Sicht von Experten und auf der anderen Seite aus der Sicht von Unternehmen, die seit vielen Jahren Betriebssysteme und die dazugehörigen User-Interfaces entwerfen. Ergänzt wurden die Guidelines durch eine Sammlung von UI-Design-Pattern [Tox09]. Die Idee der UI-Design-Pattern ist von den Software-Design-Pattern [GHJV94] abgeleitet und soll die Formalisierung von Gestaltungsaspekten sicherstellen. Die UI-Design-Patterns stellen Konzepte zur Verfügung, mit deren Hilfe bestimmte Probleme gelöst werden können. Die Empfehlungen der Guidelines und der UI-Design-Pattern wurden konsolidiert, um daraus Gestaltungsregeln für die Frontends abzuleiten. Im Rahmen dieses Kapitels werden kurz einige grundlegende Aspekte der UI-Gestaltung beleuchtet und anschließend die konsolidierten Design-Regeln für häufig verwendete UI-Elemente vorgestellt.

4.1 ASPEKTE DER GEBRAUCHSTAUGLICHKEIT UND BENUTZERFREUNDLICHKEIT

In einem zentralen Aspekt stimmen viele Guidelines und Usability-Experten überein. Ein klar strukturiertes und übersichtliches Layout ist der Schlüssel zu einem UI-Design, das die gewünschte Zielgruppe (Vergleiche Abschnitt A.5.2) intuitiv und einfach bedienen kann [Kru05][Hob95] [ADC09][MSD09]. Die Gestaltung des Interface-Designs sollte dabei nicht losgelöst, sondern gemeinsam mit den Endanwendern statt finden. Regelmäßige Evaluationen und UI-Studien sind daher ein wichtiger Bestandteil des UI-Entwicklungs-Prozesses [Kru05]. Im Rahmen dieser Arbeit wird die durch das ServFace-Projekt definierte Zielgruppe (siehe Abschnitt A.5.2 und A.1) angenommen. Zusätzlich sind im Anhang A.1 ausführliche Beschreibungen von Beispielpersonen aus der Zielgruppe zu finden. Entsprechend der Fähigkeiten und Erfahrungen der Zielgruppe soll die Gestaltung des UIs erfolgen und an die Bedürfnisse der Nutzer angepasst werden.

Auf die Visualisierung der Service-Frontends ist nur eine Untermenge aller UI-Design-Empfehlungen anwendbar. Die Frontends an sich stellen keine Gesamtapplikation dar, sondern sind selbst nur Interaktionselemente. Aus diesem Grund wurden die Guidelines nach Aspekten untersucht, die für die Darstellung der Frontends wichtige Kriterien beschreiben. Dabei stellen die folgenden Aspekte eine Auswahl der wichtigsten identifizierten Designempfehlungen dar:

1. Das User-Interface soll sich an den kognitiven Vorstellungen und Fähigkeiten der Zielgruppe orientieren. [Kru05](Kapitel: 8; Seite: 125)[Hob95](Seite: 2)[ADC09](Seiten: 19, 25f,32f, 39-41, 47ff) [MSD09](Seite: 8)
2. Die Leserichtung des Nutzers beginnt im mitteleuropäischen Kulturraum links oben und endet rechts unten. Der natürliche Anwendungsfluss soll sich daran orientieren und wichtige Elemente an der prominenteste Position platzieren. [Kru05](Seite: 21, 51ff, 60, 85ff)-[Hob95](Seite: 8)[ADC09](Seite: 47ff) [MSD09](Seiten: 14, 22, 272)
3. Eingabemöglichkeiten in ein UI-Element sollen so stark wie möglich eingeschränkt sein, um falsche Eingaben in das Element zu vermeiden [ADC09](Kapitel: Controlls; Seiten: 19, 32f, 39ff, 45)[MSD09](Seiten: 7, 11).
4. Die Art der Benutzung eines UI-Elements soll sofort erkennbar sein [Kru05](Seiten: 14ff, 37f) [Hob95](Seite: 6f)[ADC09](Seiten: 19, 39ff).
5. Elemente und UI-Teilbereiche sollen klar strukturiert und deren Bedeutung sofort erkennbar sein[Kru05](Seiten: 14ff, 31ff, 71ff)[ADC09](Kapitel: Layout Guidelines; Seiten: 343ff).
6. Zusammengehörende Design-Elemente sollen gruppiert angezeigt werden [ADC09](Kapitel: Layout Guidelines; Seiten: 343ff)[MSD09](Seiten: 14, 22f, 35).
7. Fehler bei der Eingabe in ein UI-Element sollen sofort angezeigt werden [MSD09](Seiten: 8, 27, 347f) [ADC09](Seiten: 42f).
8. Es sollen nur so viele Funktionen wie unbedingt notwendig angezeigt werden [ADC09] (Seiten: 28ff, 32) [MSD09](Seiten: 6f, 19ff).
9. Regelmäßige Nutzerstudien (auch mit kleinen Gruppen) helfen bei der Gestaltung brauchbarer UIs [Kru05](Seiten: 131ff, 138ff)[ADC09](Seiten: 25ff)[MSD09].
10. Wichtige Aspekte müssen hervorgehoben werden.[Kru05](Seiten: 21ff, 31ff, 71ff) [MSD09] (Seite: 27)
11. Wichtige oder häufig verwendete Elemente sollten immer zuerst genannt werden [Kru05] (Seiten: 24f)[ADC09](Seite: 46)[MSD09](Seite: 14, 48, 86).
12. Die Bezeichnung von UI-Elementen muss für die Zielgruppe verständlich sein [Kru05](Seiten: 14f, 67)[ADC09](Seiten: 40, 48, 133[MSD09](Seiten: 10f, 23).

Nicht alle der Aspekte lassen sich schon zur Design-Time im Kompositionswerkzeug umsetzen. Beispielsweise findet eine Auswertung der Eingaben in die UI-Elemente während des Kompositionsprozesses noch nicht statt. Die Anzeige einer Fehleingabe kann daher erst zur Laufzeit der fertig erstellten Anwendung erfolgen. Außerdem ist bei einer automatisierten Visualisierung nur eine begrenzte Semantik möglich. Bestimmte Aspekte lassen sich nicht ohne das direkte Eingreifen eines menschlichen Nutzers visualisieren. Dennoch ist die Identifizierung und Beachtung der allgemeinen Design-Aspekte wichtig. Aus den genannten Design-Empfehlungen lassen sich die folgenden Schlussfolgerungen für die Gestaltung der Frontends ableiten. Die Ableitungen sind über ihre Nummerierung den Design-Aspekten zugeordnet:

- 1 & 9: Planung und Durchführung mehrerer Evaluationen der Verständlichkeit, Nutzbarkeit und Übersichtlichkeit der Frontends.
- 2 & 11: Elemente zur Interaktion mit dem Frontend (Beispielsweise: Löschen, Verschieben) werden am oberen Teil der Frontends, im Idealfall oben links, platziert. Die eigentliche Reihenfolge der Elemente ist durch den Service vorgegeben.
- 3 & 4: Entsprechend des vorhandenen Datentypen und der angehängten Annotationen wird ein möglichst restriktives Datenelement gewählt. Interaktionselemente werden als solche kenntlich gemacht. Das gilt insbesondere für Aktions-Buttons.

- 5 & 6: Durch farbliche Abgrenzungen heben sich die Frontends vom Untergrund ab. Zusammengehörende Elemente wie beispielsweise die Gruppe der Eingabeelemente werden gruppiert dargestellt. Die unterschiedlichen Elementgruppierungen innerhalb der Frontends werden durch Rahmen kenntlich gemacht.
- 7: Felder mit entsprechender Funktion werden farblich markiert. Eine direkte Umsetzung der Funktion zur Design-Time ist nicht möglich.
- 8: Das Design der Frontends orientiert sich so stark wie möglich an der späteren Umsetzung in der Endanwendung.
- 10: Ausgewählte Elemente innerhalb der Frontends werden markiert. Die Interaktions-Schaltflächen werden durch ein ansprechendes Symbol in passender Größe visualisiert.
- 12: Die Benennung der UI-Elemente erfolgt möglichst nicht über die IDs der Elemente, sondern über einen lesbaren Namen.

Im Rahmen der Frontend-Visualisierung wird versucht, möglichst viele der genannten Aspekte umzusetzen. Im folgenden Abschnitt werden Regeln für die Wahl von UI-Elementen aufgestellt. Die Wahl des richtigen UI-Elements ist für die Darstellung gebrauchstauglicher Frontends besonders wichtig, denn die Interaktion des Nutzers erfolgt zum Großteil mit diesen Elementen. Der Nutzer soll möglichst intuitiv erkennen, welche Daten als Eingabe gefordert werden und wie die UI-Elemente zu benutzen sind.

4.2 REGELN FÜR DIE DARSTELLUNG VON UI-ELEMENTEN

Das Kompositionswerkzeug stellt die Funktionalität zur Verfügung, mehrere Service-Operationen miteinander zu verknüpfen und so eine interaktive Anwendung zu gestalten. In [Pur09] werden die allgemeinen Gestaltungsapsekte des Kompositionsprozesses beschrieben. Die Erstellung der Frontends zur Repräentation der Service-Operationen ist Kernbestandteil dieser Arbeit. Besonders die Darstellung der Parameter durch Interaktions- und Eingabeelemente ist ein wichtiger Aspekt. Im Folgenden sollen Design-Regeln für die wichtigsten UI-Elemente dargelegt werden. Die Wichtigkeit der Elemente ergibt sich aus der Häufigkeit des Einsatzes und der Gebrauchstauglichkeit für bestimmte Einsatzgebiete. Die Regeln und Werte resultieren aus der Konsolidierung der genannten Guidelines [Kru05][Hob95] [ADC09][MSD09].

4.2.1 Textboxen

Das am häufigsten eingesetze UI-Element ist die Textbox. In eine Textbox können beinah beliebige Daten eingegeben werden. Das ist Vor- und Nachteil der Textboxen. Ist bei der Erstellung des UI nicht bekannt, welche Daten in das UI-Element eingegeben werden sollen, ist die Darstellung der Textbox eine sichere Wahl. Die Textbox wird in den meisten Fällen in der Lage sein, die gewünschten Daten entgegenzunehmen. Deshalb werden Textboxen auch besonders bei der Darstellung generierter User-Interfaces eingesetzt (Vergleiche dazu Kapitel 3). Weiterhin ist die Darstellung von mehrzeiligen Textboxen möglich, um die Eingabe von längeren Texten zu ermöglichen. Die umfassende Freiheit bei der Eingabe provoziert allerdings auch Fehleingaben durch den Nutzer, da dieser beispielsweise das geforderte Format oder vorgegebene Werte nicht kennt. Die Textbox sollte deshalb nur eingesetzt werden, wenn kein anderes UI-Element genutzt werden und die Eingabe des Nutzers auch wirklich beliebig sein kann. Zusätzlich sollte, wenn möglich, innerhalb der Textbox das gewünschte Format eingetragen sein.
Zusammengefasst lassen sich die folgenden Empfehlungen für den Einsatz von Textboxen definieren:

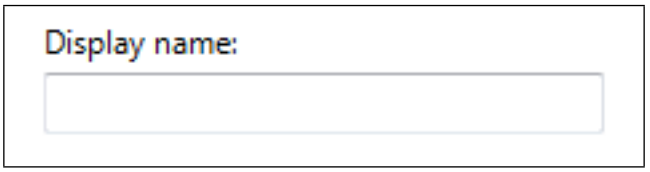


Abbildung 4.1: Beispiel einer Textbox [MSD09]

- Nutzung nur, wenn kein anderes, passenderes UI-Element möglich.
- Möglichst Nutzung nur, wenn die Eingabe beliebig sein kann.
- Bei vorgegebenem Format sollte dieses innerhalb des Textfeldes sichtbar sein.
- Scrollbalken bei Textboxen müssen bei einzeiligen Textboxen unter allen Umständen vermieden werden.
- Für lange Eingaben sollten mehrzeilige Textboxen verwendet werden.

4.2.2 Combobox & Listbox

Comboboxen und Listboxen werden verwendet, um dem Nutzer eine Auswahl aus einer Menge vorgegebener Werte zu ermöglichen. Die Combobox wird in der Regel eingesetzt, wenn ein Wert ausgewählt werden soll. Die Listbox verwendet man für Mehrfachselektionen. Grundsätzlich gibt es 2 Ausprägungen der Combobox. Die Drop-Down-Liste bietet eine einfache Möglichkeit, ein Element aus einer begrenzten Menge an Werten auszuwählen. Die Reihenfolge der Elemente ist dabei beliebig und eine Kategorisierung ist nicht zwingend notwendig. Für einer größeren Menge an verfügbaren Werten wird die eigentliche Combobox eingesetzt. In dieser sind die Werte in der Regel nach Buchstaben sortiert und können auch nach weiteren Kritierien kategorisiert sein. Weiterhin kann der Nutzer in das Auswahlfeld der Combobox eine Eingabe tätigen. In der Combobox werden dann nur die Elemente angezeigt, die der Eingabe entsprechen. Soll beispielsweise eine Stadt aus der Liste aller verfügbaren Städte mit den Buchstaben "S-A-N" ausgewählt werden, dann sind alle Städte, die mit diesen Buchstaben beginnen, anzuzeigen. In diesem Beispiel sollte die Combobox "San Franzisco" anzeigen. Die Combobox wird leider nicht von allen Toolkits unterstützt.

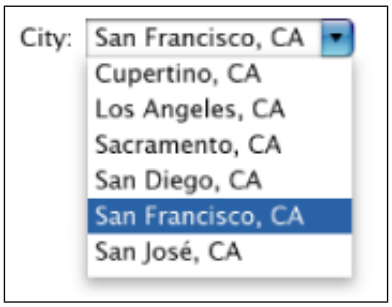


Abbildung 4.2: Beispiel einer Drop-Down-Liste [ADC09]

Zusammengefasst ergeben sich die folgenden Einsatzempfehlungen für Combo- und Listboxen:

- Combo- und Listboxen für eine Auswahl aus einer Menge von unterschiedlichen Werten
- Comboboxen für Einfachauswahl, Listboxen für Mehrfachauswahl
- Einsatz Drop-Down-Liste bei einer Wertemenge von 3-20

- Einsatz Combobox bei einer Wertemenge größer 20
- 8 Auswahlmöglichkeiten werden gleichzeitig angezeigt.
- Eine Sortierung der Werte einer Combobox ist generell empfohlen. Bei mehr als 20 Werten ist eine Sortierung aus Sicht der Gebrauchstauglichkeit Pflicht.

4.2.3 Spinner & Slider

Spinner und Slider werden hauptsächlich zur Repräsentation von Zahlenwerten eingesetzt. Der Spinner ermöglicht dabei die Erhöhung eines Zahlenwertes in diskreten Schritten mit einer definierten Genauigkeit. Spinboxen sollten nur für fortlaufende Zahlenreihen eingesetzt werden. Wichtig ist dabei, zusätzlich eine Eingabemöglichkeit für Zahlen zur Verfügung zu stellen, um übermäßiges Klicken des Nutzers zu vermeiden. Durch die Möglichkeit der Eingabe kann der Nutzer beispielsweise einen ungefähren Wert direkt eingeben und diesen dann mit dem Spinner näher definieren. Der Slider wird häufig eingesetzt, um nicht fortlaufende Zahlenwerte, Verhältnisse oder diskrete Werte einzugeben. Dabei werden die Veränderungen des Slider meist direkt optisch sichtbar. Beispielsweise sollte die Größenveränderung eines Bildes über einen Slider erfolgen. Der Nutzer sieht bei Veränderung des Sliders direkt die Vergrößerung bzw. Verkleinerung des Bildes. Slider werden deshalb häufig eingesetzt, wenn die Angabe eines ungefähren Wertes ausreichend aber die Darstellung des Verhältnisses wichtig ist.

Abbildung 4.3: Beispiel eines Spinners [MSD09]

Die Nutzung von Spinnern und Slidern unterliegt den folgenden Empfehlungen:

- Nutzung von Spinnern für fortlaufende Zahlenreihen mit notwendiger genauer Eingabe.
- Nutzung von Slidern für optisch darstellbare Veränderungen, Verhältnisse und nicht fortlaufende Zahlenreihen.
- Einschränkung der Klickhäufigkeit durch die Möglichkeit der direkten Eingabe.
- Spinner sollten nur für numerische Werte eingesetzt werden.
- Für Slider ist ein passendes Verhältnis der notwendigen Bewegung zum Zielwert festzulegen.

4.2.4 Radiobuttons & Checkboxen

Radiobuttons und Checkboxen werden für die Darstellung von Entscheidungen eingesetzt. Checkboxen verwendet man, um komplett gegensätzliche Optionen anzubieten, beispielsweise das An- oder Abschalten einer Option. Außerdem können Checkboxen als Markierung dazu dienen, ob bestimmte Elemente zu einer Auswahl gehören oder nicht. Radiobuttons hingegen werden ähnlich wie Comboboxen eingesetzt und bieten die Möglichkeit, Werte aus einer vorgegebenen Wertemenge auszuwählen. Radiobuttons werden häufig bei einer kleinen Wertemenge eingesetzt, so dass der Nutzer alle Werte gleichzeitig sieht. Generell ist bei einer größeren Wertemenge die Combobox oder Listbox vorzuziehen, um die Übersichtlichkeit zu erhöhen und den Platzbedarf zu begrenzen.
Die Empfehlungen für die Darstellung von Checkboxen und Radiobuttons sind:

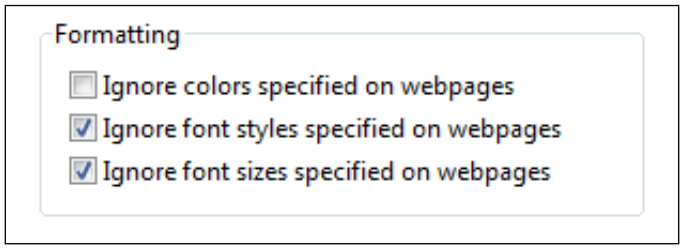


Abbildung 4.4: Beispiel einer Checkbox-Gruppe [MSD09]

- Einsatz nur bei einer geringen Menge an Optionen. Max. 10 Checkboxen o. 6 Radiobuttons
- Gruppierte Darstellung zur Kennzeichnung der Zusammengehörigkeit
- Checkboxen zur Auswahl von gegensätzlichen Optionen
- Radiobuttons zur Auswahl von einem Wert aus einer Wertemenge
- Bei Radiobuttons mit vielen Optionen auf Combo- und Listboxen zurückgreifen.

4.2.5 Tooltips

Tooltips werden eingesetzt, um zusätzliche Informationen zu einem Element zur Verfügung zu-
stellen. Dabei ist immer zu beachten, dass der Tooltip zu Beginn nicht sichtbar ist und erst nach
dem Auslösen eines Events (bspw. Maus über Element) dargestellt wird. Wichtige Informationen
sollten immer direkt sichtbar sein. Für die Darstellung von Tooltips gibt es die folgenden Empfeh-

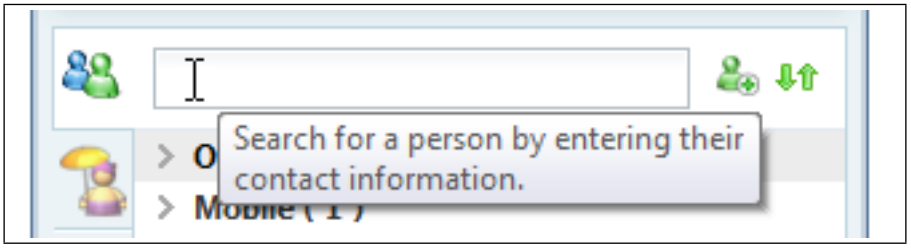


Abbildung 4.5: Beispiel eines Tooltips [MSD09]

lungen:

- Tooltips nur für die Anzeige zusätzlicher und nicht existenzieller Informationen.
- Die Anzeigedauer sollte der Länge des angezeigten Textes angepasst werden.
- Tooltips sollten direkt an dem erläuterten UI-Element angebracht sein.
- Verdeckungen von Eingaben durch den Tooltip sollten vermieden werden.

4.2.6 Kalender

Als Beispiel für ein relevantes Spezial-UI-Element soll der Kalender dienen. Ein Kalender-Widget
ist eine vereinfachte Repräsentation von realen Kalendern und soll ähnlich verwendet werden.
Die Darstellung der Daten erfolgt entsprechend in einer Matrix-Anzahl und ist nach Tageszahl,

Wochentagen, Wochen und Monaten kategorisiert. Die Eingabe eines Datums im korrekten, geforderten Format ist deutlich schwieriger als die Auswahl des richtigen Tages über ein Kalender-Widget. Dennoch sollte dem Nutzer die Möglichkeit gegeben werden, das Datumsformat per Hand einzugeben. Zu beachten ist, dass die Verfügbarkeit, das Verhalten und das Aussehen von Spezialwidgets vom Toolkit der Zielplattform abhängig ist. Dementsprechend muss immer eine alternative Eingabemöglichkeit vorhanden sein. Für die Darstellung eines Kalender-Widgets erge-

Abbildung 4.6: Beispiel für ein Kalender-Widget [MSD09]

ben sich die folgenden Empfehlungen und Hinweise:

- Spezialwidgets sind von der Darstellung im Zieltoolkit abhängig.
- Für die Datumseingabe soll möglichst ein Kalender gewählt werden.
- Der Kalender ist auf die verfügbare Zeitspanne zu begrenzen.
- Das aktuelle Datum sollte vorausgewählt sein.
- Zusätzlich zum Kalender sollte die manuelle Datumseingabe möglich sein.

4.3 ZUSAMMENFASSUNG

Die Erstellung von intuitiven und einfach nutzbaren User Interfaces ist eine umfassende Aufgabe, mit der sich viele Unternehmen und Experten beschäftigen. Eine eindeutige Meinung ist dabei meist nicht vorhanden, sondern ergibt sich aus den Ansichten einzelner Personen oder Personengruppen. Die Konsolidierung mehrerer Design-Guidelines ist daher die günstigste Möglichkeit, Empfehlungen für die Darstellung der Service-Frontends zu formulieren. Dabei wurden im Laufe des Kapitels 4 Guidelines von Usability-Experten und großen Softwarekonzernen betrachtet. Aus den Guidelines wurden zunächst allgemeine Empfehlungen für den Aufbau von User-Interfaces entnommen. Da nicht alle Aspekte auf die Darstellung automatisch visualisierter Frontends anwendbar sind, wurden die Guidelines nach besonders zutreffenden Empfehlungen ausgewertet. Neben allgemeinen Aspekten wurde besonders die Darstellung der unterschiedlichen UI-Elemente betrachtet. Die Interaktionselemente stellen den wichtigsten Aspekt der visualisierten Frontends dar und sollten daher möglichst einfach verwendbar sein. Die Vermeidung von möglichen Fehleingaben ist dabei ein Hauptaspekt. Die formulierten Empfehlungen werden soweit wie möglich in den weiteren Entwurf und die Umsetzung der Frontends einbezogen.

5 KONZEPTION VON SERVICE-FRONTENDS

Um dem Nutzer des Kompositionswerkzeugs eine Möglichkeit der präsentationsorientierten Servicekomposition [FJN+09] zu bieten, werden Services und im Speziellen deren Operationen durch Service-Frontends visualisiert. Diese werden zur Design-Time automatisiert erzeugt. In diesem Kapitel soll der allgemeine Aufbau dieser Frontends beschrieben werden. Besonders wird dabei auf die Darstellung der Parameter, die als verschiedene, einfache und komplexe Datentypen definiert sein können, eingegangen. Für umfangreiche Datentyp-Strukturen werden spezielle Darstellungen konzipiert. Anschließend werden mögliche Schwierigkeiten bei der Visualisierung der Frontends und der Betrachtung der Datentypen aufgezeigt und Lösungsmöglichkeiten vorgeschlagen.

5.1 AUFBAU DER SERVICE-FRONTENDS

Der Modellierungsprozess im Kompositionswerkzeug sieht eine Verknüpfung verschiedener Webservices zu einer servicebasierten Anwendung vor. Der Endnutzer kombiniert dazu Operationen von Webservices, die durch automatisch erzeugte Frontends visualisiert werden. Um eine korrekte Ausführung des Services in der resultierenden Anwendung zu gewährleisten, müssen die Frontends die Komponenten und die Struktur einer Service-Operation widerspiegeln. Folgende Komponenten einer Operation müssen für deren Visualisierung betrachtet werden:

- Parameter
- Parameter-Gruppen
- Datentypen der Parameter
- Parameter-Strukturen durch komplexe, verschachtelte Datentypen

Hinzu kommt die Betrachtung der Annotationen, durch deren Einfluss die Frontends verändert werden können. Um die Funktion der Parameter einer Operation näher zu erläutern, erfolgt die Unterteilung der Parameter in verschiedene Gruppen. Besonders wichtig für die Visualisierung ist die Unterteilung der Parameter in Input- und Output-Parameter. In der resultierenden Anwendung repräsentieren die Input-Parameter die vom Service zur Ausführung benötigten Eingabe-Werte und die Output-Parameter die Rückgabe-Werte des Services. Hinzu kommt eine Parameter-Gruppe zur Darstellung von Fehlermeldungen des Webservices. Über die Umsetzung der Fehlermeldungen ist zum aktuellen Zeitpunkt noch nicht entschieden worden. Die Umsetzung ist nicht Teil dieser Arbeit. Das Service-Frontend einer Service-Operation wird entsprechend der vorhandenen Parameter-Gruppen unterteilt und die Parameter je nach Gruppenzugehörigkeit in dem passenden Bereich platziert. Die Abbildung 5.1 stellt eine Umsetzungsmöglichkeit für die Visualisierung einer Service-Operation dar.

Abbildung 5.1: Umsetzungsmöglichkeit eines Service-Frontends

Zu erkennen ist der Wurzel-Container der Service-Operation "Flugsuche" mit den beiden Parameter-Containern "Eingabe" und "Ausgabe". Innerhalb der Parameter-Container werden die Visualisierungen der Eingabe- bzw. Ausgabeparameter dargestellt. Abhängig von den Datentypen der Parameter sind auch weitere Schachtelungen möglich. Die detaillierte Betrachtung der Parameter-Darstellung erfolgt im folgenden Abschnitt 5.2

5.2 DARSTELLUNG VON SERVICE-PARAMETERN UNTER BEACHTUNG SIMPLER DATENTYPEN

Die zentralen Elemente der Service-Frontends sind die Visualisierungen der Service-Parameter. Die Parameter eines Services werden in der resultierenden Anwendung zur Übermittlung der Eingabe-Daten an den Webservice und zur Darstellung der Service-Rückgaben verwendet. Eine einfache Möglichkeit der Parameter-Darstellung ist die Visualisierung durch Textboxen (Input-Parameter) und einfachen Text (Output-Parameter). Wie allerdings bereits in Abschnitt 3.3 beschrieben, ist die Gebrauchstauglichkeit von Visualisierungen, die nur aus diesen beiden generischen Elementen bestehen, relativ gering. Bspw. ist eine Eingabemaske aus einer großen Anzahl Textboxen für einen Nutzer relativ schwer zu erfassen und Fehleingaben werden auf Grund fehlender Eingabe-Beschränkungen nicht verhindert. Deshalb ist zu empfehlen, jeden Parameter durch ein Element zu visualisieren, das den Nutzer des Service-Frontends bei der Eingabe der korrekten Daten unterstützt und die Gefahr von Fehleingaben minimiert. In einem ersten Schritt kann dies durch die Auswertung der vordefinierten Datentypen der XML-Schema-Definition (XSD) erfolgen. Allerdings sind auch nach dieser Auswertung bei einigen Parametern noch relativ viele Eingabemöglichkeiten vorhanden. Die Betrachtung der Datentyp-Beschränkungen simpler Datentypen und der Definitionen komplexer Datentypen kann weitere hilfreiche Informationen für die Visualisierung der Parameter liefern. In den folgenden Abschnitten werden die Darstellungsmöglichkeiten der einzelnen Parameter-Datentypen inkl. der möglichen Beschränkungen und Erweiterungen von simplen und komplexen Datentyp-Definitionen betrachtet. Besonderes Augenmerk wird auf die Betrachtung der komplexen Datentypen gelegt, da deren Darstellung sehr umfangreich sein kann.

5.2.1 Darstellung einfacher Datentypen

In diesem Abschnitt sollen die Darstellungsmöglichkeiten der vordefinierten Datentypen beschrieben werden. Die allgemeinen Definitionen und Eigenschaften sowie Beispiele der XSD-Datentypen sind in der XML-Schema-Definition [TBMMb] beschrieben. In der folgenden Liste werden den einzelnen Datentyp-Gruppen UI-Elemente zur Darstellung in einem Service-Frontend zugeordnet:

Logische Datentypen:
Darstellung: Es sind lediglich die beiden sich gegenseitig ausschließenden Werte true oder false möglich. Die Darstellung sollte deshalb über ein UI-Element erfolgen, dass eine exklusive Auswahl zwischen 2 Werten ermöglicht:

- Eingabeparameter: Darstellung als Checkbox
- Ausgabeparameter: Darstellung durch deaktivierte Checkbox mit Repräsentation des Wertes

Textuelle Datentypen
Darstellung: Textuelle Datentypen können beliebige Zeichenfolgen aufnehmen. Deshalb kann ein einschränkendes UI-Element nur bei einer Beschränkung des textuellen Datentypen verwendet werden.

- Eingabeparameter: Darstellung als Textbox
- Ausgabeparameter: Darstellung als Text

Numerische Datentypen
Darstellung: Die numerischen Datentypen können je nach Definition eine sehr hohe Anzahl an Zahlenwerten annehmen. Aus diesem Grund ist ohne eine weitere Einschränkung des Datentyps die Darstellung nur durch unbeschränkte UI-Elemente möglich. Die UI-Elemente sollten allerdings nur Zahlenwerte annehmen:

- Eingabeparameter:
  - decimal/integer: Textbox inkl. Stepper
  - double/float: Textbox mit Stepper und Unterstützung für Nachkommastellen
- Ausgabeparameter: Darstellung als deaktivierter Stepper.

Datum- und Zeit-Datentypen
Darstellung: Die Datums- und Zeittypen zeichnen sich durch ihre spezielle Formatierung aus. Die Unterteilung unterscheidet sich je nach Datumstyp. Das UI-Element sollte eine Vorformatierung entsprechend des Datumsformats aufweisen:

- Eingabeparameter:
  - Datumsangaben: Datumswahl über Kalender-Widget, Uhrzeitwahl über eingeschränkten Stepper
  - Tag/Monat/Jahr separat: Angabenwahl über Combobox
  - Zeitspanne: Auswahl je Komponente der Zeitspanne über Combobox

- Ausgabeparameter:
  - Datumsangaben: Markierung im Kalender-Widget inkl. zusätzlichem Textbereich mit textueller Datumsangabe, Vorgabe im Stepper
  - Tag/Monat/Jahr separat: Text
  - Zeitspanne: Text möglichst mit Auflösung der Komponenten-Bezeichnungen

Trotz der Merkmalsunterscheidung der Datentypen sind die gewählten UI-Elemente noch relativ generisch und akzeptieren eine große Anzahl an Eingabemöglichkeiten. Besonders textuelle und numerische Typen akzeptieren durch die Verwendung einer Textbox eine große Menge unterschiedlicher Werte. Der Nutzer erhält nach der einfachen Auswertung der Datentypen keinen Hinweis auf die geforderte Eingabe in das angezeigte UI-Element. Auch die Namen der Parameter werden vom Service-Anbieter meist nach funktionalen Aspekten vergeben. Sie enthalten deshalb oftmals keinen Hinweis auf die erforderliche Eingabe. Sowohl der Aspekt der Einschränkung der Eingabemöglichkeiten als auch die gezielte Nutzerführung müssen verbessert werden, um wirklich nutzbare Service-Frontends zu visualisieren. Die in Abschnitt 2.3.3 vorgestellten Annotationen können für diese Verbesserung herangezogen werden. Beispielsweise lässt sich durch die Feedback-Annotation ein verständlicher Name und ein Tooltip definieren. Der Nutzer erhält somit wichtige Hinweise, welche Eingaben für ein UI-Element erwartet werden. Weiterhin können durch Annotationen wie die Format-Annotation oder die Enumerations-Annotation die Eingabemöglichkeiten durch Festlegung des Eingabeformats oder das zur Verfügung stellen einer Auswahl gültiger Werte die Eingabemöglichkeiten und damit die Nutzerführung verbessert werden. Weitere Möglichkeiten und Auswirkungen der Annotationen werden im Kapitel 6 erläutert.

5.2.2 Weitere Datentypen

Die XSD-Spezifikation beschreibt auch binäre Datentypen (base64Binary, hexBinary), XML Datentypen und den generischen Datentyp "anyType". Während die Darstellung der XML-Datentypen nicht notwendig ist, erfordert die Behandlung der binären Datentypen und des generischen Spezialtyps "anyType" eine umfangreichere Betrachtung. Der Datentyp "anyType" wird verwendet, um auszudrücken, dass ein Element einen beliebigen Inhalt haben kann. Die Darstellung eines Parameters mit diesem Datentyp muss also durch ein UI-Element erfolgen, das ebenfalls beliebige Eingaben akzeptieren kann. Das generischste UI-Element für die Eingabe ist die Textbox und für die Ausgabe einfacher Text. Diese beiden UI-Elemente sind damit am besten für die Darstellung des "anyType" Datentypen geeignet. Da allerdings keinerlei Einschränkungen für die zu übermittelnden bzw. die übermittelten Daten vorliegen, kann eine korrekte Anzeige auch mit den beiden generischen UI-Elementen nicht garantiert werden. Weiterhin gibt es keine direkte Möglichkeit, dem Nutzer einen Hinweis zu geben, welche Daten er für den Service-Aufruf zur Verfügung stellen soll. Sinnvoll ist daher die Erweiterung von Parametern, die dem Typen "anyType" gehorchen. Das geschieht durch zusätzliche Annotationen, die den Parameter näher spezifizieren. Parameter, die durch einen binären Datentyp definiert sind, repräsentieren in der Regel Binärdaten, wie beispielsweise Bild- oder Multimedia-Dateien. Meist werden binäre Daten per URL referenziert und nicht direkt an den Service übergeben. Dennoch bieten die binären Datentypen diese Möglichkeit an. Die Zuweisung eines UI-Elements ist nicht direkt möglich, da nicht bekannt ist, um welche Daten es sich handelt. Auch in diesem Fall kann die Nutzung von Annotationen weiterhelfen. Anhand der MIME-Type Annotation kann festgelegt werden, um welche Binärdaten es sich handelt. Dadurch ist anschließend auch die Zuordnung eines UI-Elements und damit die Darstellung der Binärdaten möglich.

5.2.3 Beschränkte einfache Datentypen

XML-Schema bietet die Möglichkeit, neue Datentypen auf Basis der zuvor genannten eingebetteten Datentypen zu definieren. Eine erste Variante stellen die simplen Datentyp-Definitionen dar, welche allerdings auf Umbenennungen und die Definition von Einschränkungen limitiert sind. Folgende Einschränkungen sieht die XML-Schema-Definition für simple Datentypen vor:

- length: Legt die genaue Anzahl der Zeichen einer Zeichenfolge fest.
- min/maxLength: Legt die minimale/maximale Anzahl der Zeichen einer Zeichenfolge fest.
- pattern: Definiert ein Format-Pattern, in welchem eine gültige Eingabe vorliegen muss.
- enumeration: Definiert eine Menge gültiger Werte.
- whiteSpace: Legt fest, wie mit Leerzeichen umgegangen wird.
- min/maxInclusive: Legt einen Wertebereich gültiger Zahlenwerte fest. Die Grenzen werden zum Wertebereich gezählt.
- min/maxExclusive: Legt einen Wertebereich gültiger Zahlenwerte fest. Die Grenzen zählen nicht zum Wertebereich.
- totalDigits: Legt die maximale Anzahl der Stellen eines Zahlenwertes fest.
- fractionDigits: Legt die maximale Anzahl der Nachkommastellen eines Zahlenwertes fest.

Ähnliche Einschränkungen lassen sich auch durch Annotationen realiseren. Sind die Datentypen also nicht durch Datentyp-Definitionen beschränkt, so lassen sich die Beschränkungen in bestimmtem Maße durch die Annotationen nachträglich hinzufügen. Neben den Vorteilen und Ergänzungsmöglichkeiten können durch solche Annotations-Beschränkungen auch Konflikte auftreten. Diese werden in Abschnitt 5.4.3 diskutiert. Nicht alle der zuvor genannten Beschränkungen lassen sich auf jeden Basis-Datentypen anwenden. Die Tabelle 5.1 stellt die Kompatibilität der Einschränkungen mit den Datentyp-Gruppen dar.

Datentyp-Gruppe:	Logische	Textuelle	Numerische	Datumstypen
Einschränkungen:				
length	-	+	-	-
min/maxLength	-	+	-	-
pattern	+	+	+	+
enumeration	-	+	+	+
whiteSpace	+	+	+	+
min/maxInclusive	-	-	+	+
min/maxExclusive	-	-	+	+
totalDigits	-	-	+(float/double: -)	-
fractionDigits	-	-	+(float/double: -)	-

Tabelle 5.1: Kompatibilitätsmatrix: Mögliche Einschränkungen je Datentyp-Gruppe [Vin03]

Beschränkungen im Rahmen simpler Datentypdefinitionen sind bindend. Das heißt, dass ein Webservice für einen eingeschränkten Parameter nur Werte innerhalb der Beschränkung als Eingabe akzeptiert. Andere Werte werden nicht angenommen und verursachen ein Fehlschlagen des Service-Aufrufs.
Durch die Einschränkung der Ausprägung, der Anzahl und des Formats gültiger Werte können die Datentyp-Beschränkungen auch die Wahl der UI-Elemente beeinflussen. Für die Darstellung eines eingeschränkten Eingabe-Parameters sollte ein UI-Element gewählt werden, dass nur Eingaben innerhalb der definierten Datentyp-Begrenzung bzw. in dem vorgegebenen Format akzeptiert. Dadurch lassen sich Fehleingaben in der Ergebnisanwendung vermeiden und somit die Nutzbarkeit der Frontends erhöhen. Bei Ausgabe-Parametern ist die Betrachtung der Datentyp-Beschränkungen weniger für die Wahl des repräsentierenden UI-Elements interessant als vielmehr im Hinblick auf die Definition von Datenflüssen. Durch die Analyse der Beschränkungen lassen sich Kollisionen der Datentyp-Definitionen mit dem zu befüllenden Parameter erkennen. Diese Thematik wird näher im Kapitel 7 beschrieben.

Die Einschränkungen der simplen Datentypen beeinflusst vor allem die Anzahl der möglichen Eingabewerte. In Kapitel 4 wurden Design-Empfehlungen formuliert, die unter anderem die Eignung von UI-Element in Abhängigkeit von der darzustellenden Anzahl möglicher Werte beschreiben. Beispielsweise wird für eine maximale Anzahl von 20 Eingabemöglichkeiten die Wahl der Drop-Down-Liste nahegelegt. Liegt die Anzahl darüber, wird aus Sicht der Benutzerfreundlichkeit und Übersichtlichkeit die ComboBox empfohlen. Die Drop-Down-Liste eignet sich also besonders für die Anzeige von durch Enumerationen beschränkten Parametern, die nicht mehr als 20 Eingabemöglichkeiten definieren. Die folgende Übersicht stellt eine Darstellungsempfehlung für Eingabeparameter im Hinblick auf ihre Datentypen mit und ohne Beschränkungen dar. Weiterhin werden Grenzen aufgezeigt, bei denen der Wechsel zu einem alternativen UI-Element erfolgen kann.

Logische Datentypen:

Darstellung unbeschränkt: Checkbox

Darstellung beschränkt: Checkbox

Textuelle Datentypen:

Darstellung unbeschränkt: Textbox

Darstellung Einschränkung Buchstabenanzahl:
Textbox mit Größenanpassung und ggf. Markierung für Buchstaben. Darstellung als mehrzeilige Textarea, bei entsprechend definierter Textlänge.

Darstellung Pattern: Aufteilung der Textboxen in feste und variable Bestandteile, Begrenzung der Eingabemöglichkeiten in den jeweiligen Textboxen auf Vorgaben des Pattern.

Darstellung Enumeration: Verwendung von Radiobuttons (1 o. 2 gültige Werte), Drop-Down-Listen (<=20 gültige Werte) und ComboBoxen (>20 gültige Werte).

Numerische Datentypen

Darstellung unbeschränkt: auf Zahlen beschränkte Textbox

Darstellung Einschränkung Wertebereich:
Je nach Anzahl der möglichen Werte kommen Stepper (<= 20 gültige Werte) in Frage. Für größere Wertebereiche ist eine Textbox in Kombination mit einem Stepper (zur Feinjustierung) zu empfehlen. Zusätzlich soll der Wertebereich als Hinweistext oder Tooltip genannt werden.

Darstellung Pattern: Aufteilung der Textboxen in feste und variable Bestandteile, Begrenzung der Eingabemöglichkeiten in den jeweiligen Textboxen auf Vorgaben des Pattern.

Darstellung Enumeration: Verwenden von Radiobuttons (1 o. 2 gültige Werte), Drop-Down-Listen (<=20 gültige Werte) und ComboBoxen (>20 gültige Werte).

Darstellung Einschränkung Stellenzahl:
Textbox mit angepasster Größe und Markierung für die geforderte Stellenanzahl der numerischen Eingabe

Datums Datentypen

Darstellung unbeschränkt: Kalender-Widget

Darstellung Einschränkung Datumsbereich:
Beschränkung des Kalender-Widgets auf den vorgegeben Datumsbereich. Bei Zeitdauerangaben sollte weiterhin eine ComboBox je Komponente verwendet werden.

Darstellung Pattern: Bei Wahl über Kalender-Widget muss das Widget die Speicherung und Übermittlung im richtigen Format gewährleisten. Wird das Pattern der Zeitdauer-Angabe verändert, so sind die ComboBoxen zur Teilauswahl an das neue Format anzupassen.

Darstellung Enumeration: Verwendung von Radiobuttons (1 o. 2 gültige Werte), Drop-Down-Listen (<=20 gültige Werte) und ComboBoxen (>20 gültige Werte).

Die Behandlung der Whitespace-Beschränkung hat keinen Einfluss auf die Wahl der UI-Elemente, da diese nur den Umgang mit Leerzeichen beschreibt. Das ist für die Speicherung der Daten wichtig und wird durch das Tool selbst sichergestellt.

5.3 DARSTELLUNG VON PARAMETERN MIT KOMPLEXEN DATENTYPEN

Die Betrachtung von komplexen Datentypen ist ein wichtiger Aspekt bei der Visualisierung von Frontends für Services. Komplexe Datentypen definieren umfassende Datenstrukturen, die beliebig tief geschachtelt sein können. Auch Rekursionen innerhalb der Datenstruktur sind möglich. Die Abbildung 5.2 dient als Beispiel und visualisiert den fiktiven, komplexen Datentyp "Kunde". Für die Darstellung komplexer Typen wird deren Struktur auf die Blattelemente heruntergebrochen. Je nachdem, ob es sich um einen Eingabe- oder Ausgabeparameter handelt, wird dann die Darstellungsform der Struktur bestimmt. Für die simplen Blattelemente werden UI-Elemente entsprechend der Beschreibungen in den Abschnitten 5.2.1 - 5.2.3 ausgewählt. Bei komplexen Datentypen kann für jedes Kindelement definiert werden, wie häufig das Kind innerhalb des Datentypen vorkommt. Dies geschieht über die Occurence-Angaben minOccurs und maxOccurs. Auch diese Angaben haben Einfluss auf die Darstellung komplexer Typen.

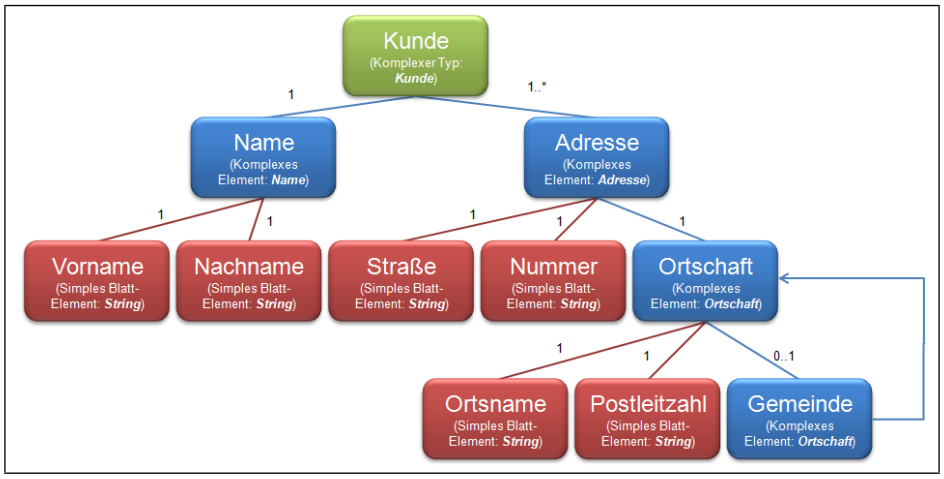


Abbildung 5.2: Beispielvisualisierung des komplexen Datentyps Kunde

5.3.1 Komplexe Eingabeparameter

Generell ist der Umfang komplexer Datentyp-Strukturen nicht begrenzt. Bei sehr umfangreichen Datentyp-Stukturen kann die Darstellung durch tiefe Verschachtelungen und große Anzahl von Parametern sehr komplex werden. Das äußert sich vor allem durch einen gesteigerten Platzbedarf und die Reduzierung der Lesbarkeit, Übersichtlichkeit und Benutzbarkeit der visualisierten Frontends. Wird zusätzlich horizontales Scrollen notwendig, hat das besonders negative Auswirkungen auf die Gebrauchstauglichkeit. Allerdings steht die Visualisierung gebrauchstauglicher Service-Frontends im Vordergrund des Kompositions- und Anwendungsgenerierungsprozesses. Zusätzlich ist die Handhabung von sehr großen Frontends innerhalb des Kompositionswerkzeugs schwierig, denn die Größe der Kompositionsfläche ist begrenzt. Im Folgenden werden Darstellungsmöglichkeiten für einfache und umfangreiche Strukturen komplexer Typen beschrieben.

5.3.1.1 Darstellung komplexer Eingabeparameter

Für die Darstellung von einfachen Eingabeparametern mit komplexen Typ kann eine verschachtelte und gruppierte Darstellung gewählt werden. Das bedeutet, dass die komplexe Wurzel und alle komplexen Kindelemente durch Gruppierungselemente dargestellt werden. Die Gruppierungselemente einer Ebene enthalten dann die UI-Elemente aller simplen Kindelemente und erneut die Gruppierungselemente der komplexen Kindelemente. Die Abbildung 5.3 stellt eine mögliche Visualisierung des komplexen Datentypen “Flight Class“ mit den beiden ebenfalls komplexen Kindelementen “Economy Class“ und “Business Class“ dar.

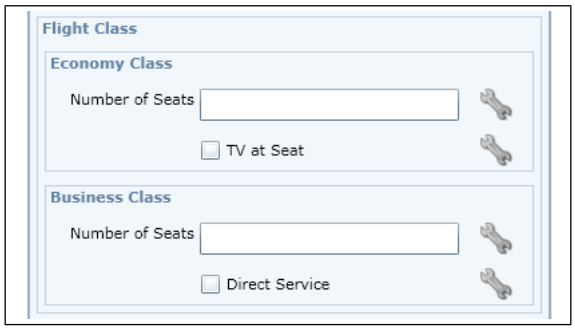


Abbildung 5.3: Beispielvisualisierung einer komplexen Schachtelung

Ist auf Grund der Häufigkeitsangaben eine eine mehrfach Darstellung des komplexen Datentypen oder eines komplexen Kindelements notwendig, dann kann neben der duplizierten Darstellung auch die Darstellung als Liste in Frage kommen. Für diese Darstellung müssen allerdings weitere Voraussetzungen erfüllt werden, die im Folgenden aufgeführt sind:

Darstellung als Liste:

- Der komplexe Datentyp darf nur ein Kindelement mit einem simplen Datentypen aufweisen
- Sind mehrere Kindelemente vorhanden, muss für jedes Kindelement eine eigene Liste dargestellt werden.
- Sind komplexe Kindelemente vorhanden, sind die Bedingungen für deren Kindelement erneut zu betrachten.
- Das simple Kindelement muss durch eine Enumeration beschränkt sein. Diese kann als Datentyp-Beschränkung oder Annotation vorliegen.
- Hinzufügen von Listeneinträgen ist innerhalb der Grenzen der Häufigkeitsangaben möglich (maxOccurs > 1) bzw. notwendig (minOccurs > 1)

Die Darstellung von komplexen Parameteren als Tabelle ist nur in Ausnahmefällen praktikabel.

5.3.1.2 Darstellung umfangreicher komplexer Eingabeparameter

Ab einem bestimmten Komplexitätsgrad ist eine vereinfachte Darstellung der komplexen Eingabeparameter zu empfehlen. Zu beachten ist allerdings, dass die Visualisierung im Kompositionswerkzeug möglichst nicht von der Darstellung in der resultierenden Anwendung abweichen darf. Das Visualisierungskonzept muss also eine Darstellungsmöglichkeit bieten, die gleichermaßen im Kompositionswerkzeug und der resultierenden Anwendung verwendbar ist.

Die Bestimmung, ab wann die alternative Darstellung bei Eingabeparametern gewählt werden muss, ist von verschiedenen Kriterien abhängig, zum Beispiel:

1. Die maximale Anzahl der Verschachtelungsebenen
2. Die maximale Anzahl der zu erzeugenden UI-Elemente

Wird bei einem der genannten Kriterien ein festgelegter Grenzwert überschritten, dann ist das alternative Darstellungselement für die komplexen Datentypen zu wählen. Wichtig ist die korrekte Festlegung der Grenzwerte, die unter anderem von der gewählten Zielplattform abhängig sind. Die Anzahl der Schachtelungsebenen stellt dabei einen entscheidenden Faktor dar, denn die mehrfache Schachtelung von Eingabeparametern führt schnell zu unlesbaren Frontends. Wichtig ist, dass jedes Kindelement für die Definition von Datenflüssen einzeln anwählbar bleibt.

Zusammengefasst lassen sich die folgenden Anforderungen an die Visualisierung für umfangreiche komplexe Eingabeparameter formulieren:

- Alle Komponenten des komplexen Datentypen werden dargestellt und sind möglichst einfach zu erreichen.
- Die Zugehörigkeit der Kindelemente zu den jeweiligen Eltern-Elementen ist erkennbar.

- Die Häufigkeitsangaben des komplexen Parameters und seiner Kindelemente müssen beachtet werden.

- Alle Datentyp-Elemente lassen sich als Ziel bzw. Quelle einer Datenverbindung zwischen Frontends definieren.

Das Frontend-Mockup 5.4 ist die Konzeption der Visualisierung von umfangreichen Eingabeparametern:

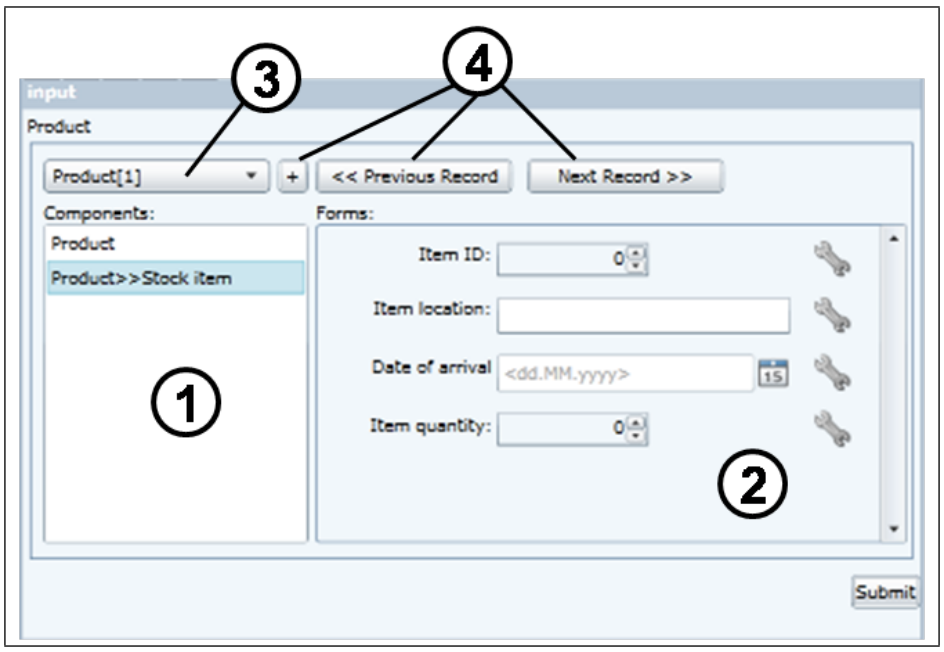


Abbildung 5.4: Visualisierungsentwurf für eine komplexe Eingabeparameterstruktur

Das gezeigte Frontend visualisiert das Beispiel für einen komplexen Datentypen aus Abbildung 5.2. Auf der linken Seite ist eine Liste der Datentyp-Komponenten platziert (1). Die unterschiedlichen Pfade des komplexen Datentyps werden sequentiell durchlaufen, und alle komplexen Kinder werden in die Liste aufgenommen. Da es sich um eine flache Liste handelt, gibt man die Pfade innerhalb der Liste über den Elementnamen an. Als Beispiel dient hier das komplexe Element "Stock Item". Dieses Element ist ein komplexes Kind-Element des komplexen Typs "Product". Dementsprechend ist der Listeneintrag dafür "Product » Stock Item". Bei Auswahl eines Elements aus der Liste, werden alle einfachen Kinder des ausgewählten Elements dargestellt (2). Kinder mit simplen Typen werden durch ein entsprechendes Eingabe-Element visualisiert. Das Gesamt-Frontend steht für jeweils einen Parameter-Datensatz. Entsprechend der Häufigkeitsangaben können mehrere Datensätze existieren, zwischen denen mit einer Menü-Liste (3) oder sequentiell mit den Menübuttons (4) navigiert werden kann. In der resultierenden Anwendung werden automatisch leere Datensätze zur Eingabe entsprechend der minOccurs-Angabe angelegt. Außerdem kann der Nutzer in der Ergebnis-Anwendung weitere Datensätze hinzufügen ("+" - 4). Die maximale Anzahl zusätzlicher Datensätze entspricht der maximalen Häufigkeitsangabe maxOccurs. Zusätzlich müssen die Häufigkeitsangaben der Kindelemente des komplexen Typen beachtet werden. Dazu werden automatisch die Kindelemente entsprechend der minimalen Häufigkeit angelegt. In der Endanwendung kann der Nutzer dann auch komplexe Kinder bis zur maximalen Anzahl hinzufügen.

5.3.2 Komplexe Ausgabeparameter

Bei der Anzeige eines Ausgabeparameters mit einem komplexen Datentyp geht es hauptsächlich um das übersichtliche Anzeigen der zurückgelieferten Daten. Da innerhalb des Kompositionsprozesses die Nutzung der Service-Rückgaben als Input für andere Services vorgesehen ist, müssen die gewählten UI-Elemente eine einfache Möglichkeit der Selektion der Datentyp-Komponenten und der einzelnen Datensätze anbieten. Beispielsweise ist die Darstellung als einfacher Text zur Selektion der einzelnen Unterelemente der komplexen Struktur nicht geeignet. Ähnlich wie bei Eingabeparametern können umfangreiche Strukturen schnell zu einem ausufernden, unübersichtlichen Frontend führen. Für die Ausgabeparameter ist deshalb ebenfalls eine alternative Visualisierung für umfangreiche Strukturen notwendig.

5.3.2.1 Darstellung komplexer Ausgabeparameter

Eine empfehlenswerte Möglichkeit der Darstellung komplexer Ausgabeparameter ist eine Tabelle oder Liste, wie in der Abbildung 5.5 beispielhaft visualisiert.

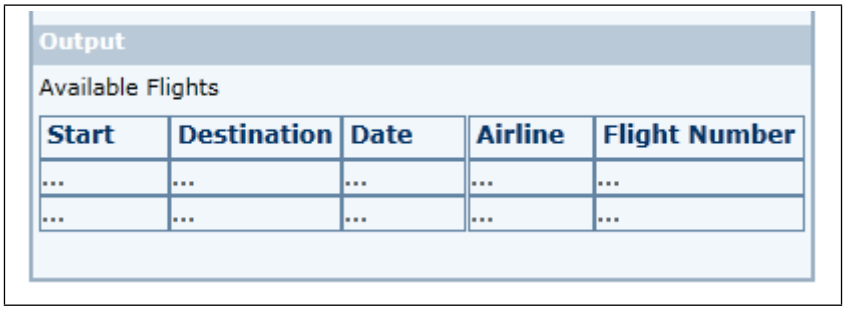


Abbildung 5.5: Beispielvisualisierung eines Ausgabeparameters mit komplexem Datentyp als Tabelle

Damit komplexe Ausgabeparameter als Tabelle dargestellt werden können, müssen die folgenden Bedingungen erfüllt sein:

Darstellung als Tabelle:

- Der komplexe Datentyp darf nur Kindelemente enthalten, die selbst durch einen einem simplen Datentyp definiert sind.
- Jedes Kindelement wird als Spalte der Tabelle dargestellt. Ist nur ein simples Kindelement vorhanden, dann sollte die Darstellung als Liste erfolgen.
- Das simple Kindelement muss in textueller Form darstellbar sein.
- Die Mehrfachauswahl aus einer Ausgabetabelle ist in der Endanwendung innerhalb der Grenzen der Häufigkeitsangaben möglich (maxOccurs > 1) bzw. notwendig (minOccurs > 1)

Erfüllt ein komplexer Typ diese Bedingungen nicht, dann ist eine Darstellung als Tabelle nicht sinnvoll. Besonders das Auftreten komplexer Kindelemente und damit tieferer Datentyp-Strukturen macht die Darstellung komplizierter. Eine Verschachtelung innerhalb einer Tabelle oder Liste ist schwierig umzusetzen, wenn gleichzeitig die Übersicht erhalten bleiben soll. Ähnliche Probleme bereitet die Darstellung von zu vielen einfachen Kindelementen, da in diesem Fall die Tabelle sehr

breit werden kann. Möglich wäre die Darstellung über einen Baum, welcher die bereits baumartige Struktur eines komplexen Datentypen reflektieren kann. Allerdings ist die Größe eines Baumes nicht fix. Der effektive Platzbedarf ist abhängig von der Komplexität des Datentypen und von der Tiefe der aktuell geöffneten Ebene. Eine Größenabschätzung zur Design-Time ohne das Vorhandensein konkreter Daten ist schwierig. Der Platzbedarf kann sich zur Laufzeit der resultierenden Applikation stark verändern. Die Folge sind ungewollte und schwer kalkulierbare Effekte, wie beispielsweise Verschiebungen oder Überlagerungen. Bei der Begrenzung der maximalen Fläche eines Baum-Elements kann bei umfangreichen komplexen Datentypen die Darstellung eines horizontalen und vertikalen Scrollbalkens notwendig werden, um trotz der begrenzten Fläche den gesamten Baum zugänglich zu machen. Das führt in vielen Fällen zu einem Verlust der Übersicht und des Gesamtkontextes. Außerdem muss der Endnutzer gegebenenfalls vertikal scrollen, was als sehr umständlich gilt und vermieden werden sollte [Nie05][Nie02].

Eine Alternative stellt eine dynamische Schachtelung komplexer Elemente durch Überblendung mehrerer Tabellen dar. Dabei wird in der Grundansicht das komplexe Kind-Element nicht direkt in der Tabelle dargestellt, sondern nur namentlich erwähnt. Fährt der Nutzer nun mit der Maus über das Element, öffnet sich ein Overlay-Bereich, der die Informationen zu den Kindelementen enthält. Diese Variante lässt sich beliebig tief schachteln und ist deshalb auch für umfangreiche komplexe Typen geeignet. Durch die Möglichkeit der Überlagerung ist der resultierende Platzbedarf deutlich besser kalkulierbar, als bei einem Baum-Element. Bei zu umfangreichen Typen gleichen sich aber die Probleme, denn auch bei dieser Lösung verliert der Nutzer schnell den Gesamtkontext und bei vielen geöffneten Overlays vor allem die Übersicht. Weiterhin ist die Selektion der einzelnen Datentyp-Komponenten recht schwierig. Allerdings kann man durch das dynamische Schließen der Overlayfenster schnell und ohne scrollen auf die Ebenen der Eltern-Elemente zurückspringen. Die Abbildung 5.6 soll das Konzept der dynamischen Schachtelung an Hand einer zweispaltigen Tabelle verdeutlichen.

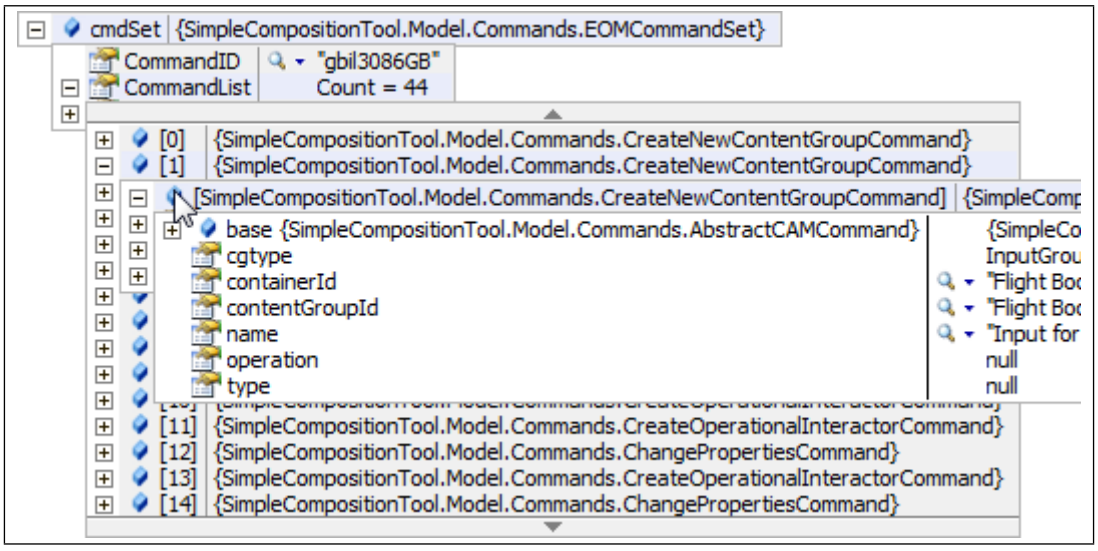


Abbildung 5.6: Dynamische Schachtelungsdarstellung in Microsoft Visual Studio 2008 [Mic07]

Die Häufigkeitsangaben minOccurs und maxOccurs spielen bei der Wahl der UI-Elemente für Output-Parameter eine untergeordnete Rolle, da die Ausgabe wie vom Service geliefert angezeigt werden muss. Genutzt werden können die Häufigkeitsangaben, um vorab zu erkennen, wie viele Ergebnisse maximal zurückgegeben werden können. Dadurch lässt sich schon zur Design-Time der Platzbedarf der Service-Antwort approximieren.

Da die Visualisierung von Ausgabeparametern mit einer umfangreichen Datentyp-Struktur ein umfassendes Problem darstellt und beide vorgeschlagenen Lösungen nicht ideal sind, wird im folgenden eine alternative Möglichkeit der Darstellung umfangreicher Ausgabeparameter erläutert.

5.3.2.2 Darstellung umfangreicher komplexer Ausgabeparameter

Ähnlich wie bei umfangreichen Eingabeparametern ist eine alternative Darstellung der komplexen Ausgabeparameter ab einem bestimmten Komplexitätsgrad notwendig. Die folgenden Kriterien müssen betrachtet werden, um zu entscheiden, ab wann die alternative Darstellung erfolgen muss:

1. Die Anzahl der Verschachtelungsebenen ist größer als 1
2. Die maximale Anzahl der direkten Kindelemente

Die Anforderungen an die Darstellung komplexer Ausgabeparameter sind ähnlich denen komplexer Eingabeparameter. Hinzu kommt aber die Notwendigkeit, dass in der Endanwendung einer oder mehrere Datensätze der Service-Rückgabe ausgewählt werden können, um den direkten Input für einen anderen Service festzulegen. Die Anzahl der Auswahlmöglichkeiten legt dabei der min/maxOccurs Wert des zu befüllenden Elements fest. Dementsprechend muss auch die Anzeige für Ausgabeparameter angepasst werden. Zusammenfassend lassen sich folgende Anforderungen identifizieren:

- Alle Komponenten des komplexen Datentypen werden dargestellt und sind möglichst einfach zu erreichen.
- Die Zugehörigkeit der Kindelemente zu den jeweiligen Elternelementen ist erkennbar.
- Datensätze müssen in der Endanwendung entsprechend der Häufigkeitsangaben des Zielelements auswählbar sein
- Alle Datentyp-Elemente lassen sich als Quelle einer Datenverbindung zwischen Frontends definieren.

Das Mockup 5.7 stellt einen Entwurf für die vereinfachte Darstellung komplexer Ausgabeparameter dar:

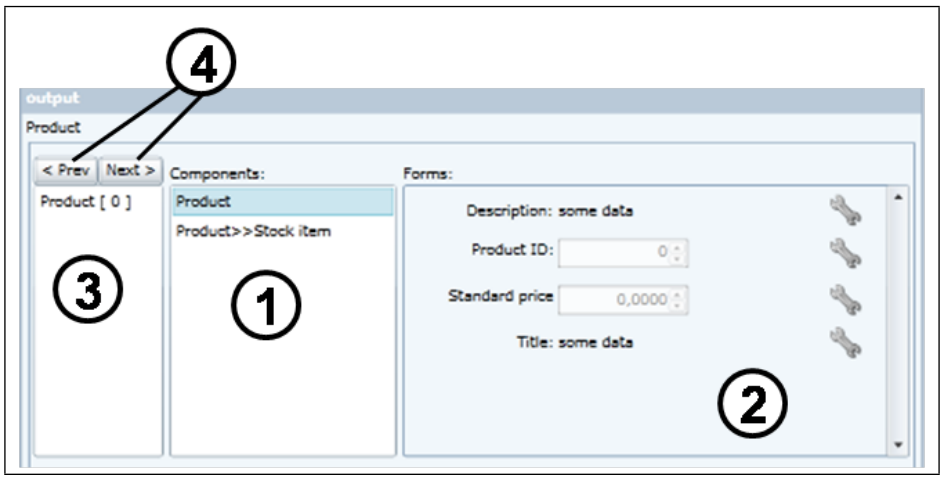


Abbildung 5.7: Visualisierungsentwurf für eine komplexe Ausgabeparameterstruktur

Ähnlich der Darstellung der Eingabeparameter gibt es eine Übersicht über die Komponenten des komplexen Datentypen (1) und eine detaillierte Übersicht über die direkten Kind-Elemente der ausgewählten Komponente (2). Hinzugekommen ist eine Übersicht über alle verfügbaren Datensätze (3). Diese Liste erlaubt die Auswahl von Datensätzen für die Weitergabe an eine weitere

Service-Operation in der resultierenden Anwendung. Das selbstständige Hinzufügen neuer Datensätze oder das Verändern zurückgegebener Datensätze ist bei Ausgabeparametern nicht möglich. Neben der Auswahlliste (3) kann auch über die Navigation (4) zwischen den Datensätzen gewechselt werden. Die Selektion der gewählten Komponente bleibt dabei erhalten. Im Kompositionswerkzeug wird ein nicht ausgefülltes Frontend entsprechend der Datentyp-Definition zur Verfügung gestellt. Dadurch ist es möglich, die einzelnen simplen Komponenten eines komplexen Typen als Quelle für einen Datenfluss zu definieren.

5.3.3 Darstellungsverbesserung durch Annotationen

Die Darstellung von Parametern mit simplem Datentypen lässt sich durch die Nutzung von Annotationen verbessern. Auch für komplexe Parameter können Annotationen dazu verwendet werden, um die Darstellung zu verbessern. Neben der obligatorischen Feedback Annotation zur Darstellung eines Labels ist die Group-Annotation eine gute Möglichkeit zur strukturierten Darstellung komplexer Parameter. Beispielsweise können durch die Group-Annotation Zusammengehörigkeiten und Gruppierungen definiert und gleichzeitig mit einem Namen versehen werden. Dadurch lassen sich logische Einheiten aus mehreren komplexen Unterelementen bilden, die zusammen dargestellt werden können. In der alternativen Darstellung umfangreicher komplexer Parameter können dadurch die Elemente mehrerer komplexer Untertypen in einem Formular unter einem gemeinsamen Namen zusammengefasst werden. Das reduziert die Anzahl der Einträge in der Komponentenliste. Wird beispielsweise die Gruppierung "Customer Data" über die Komponenten "Name" und "Address" definiert, dann wäre durch die Gruppierung die Zusammengehörigkeit der beiden Komponenten bekannt. Statt nun weiterhin beide Komponenten getrennt aufzuführen, würde nur noch der Eintrag "Customer Data" angezeigt werden. Die dazugehörige Detailansicht enthält alle Bestandteile beider, gruppierter Komponenten.

Weiterhin kann die Mandatory-Annotation dafür genutzt werden, um die Anzeige komplexer Typen weiter zu optimieren. Beispielsweise könnten in der Ausgangsansicht nur Elemente dargestellt werden, für die eine Eingabe vorgesehen ist. Optionale Elemente wären zu Beginn ausgeblendet. Möchte der Nutzer dennoch Daten in die optionalen Felder eingeben, dann kann er diese Felder durch einen "Ausklappmechanismus" sichtbar machen.

5.4 PROBLEMIDENTIFIKATION UND LÖSUNGSVORSCHLÄGE BEI DER DARSTELLUNG UND KOMPOSITION VON PARAMETERN

Bei der Darstellung der Parameter müssen viele unterschiedliche Aspekte betrachtet werden. Deshalb treten während des Visualisierungs- und Kompositionsprozesses Probleme auf, die gesondert betrachtet werden müssen. In den folgenden Abschnitten werden die Probleme und deren Auswirkungen diskutiert. Gleichzeitig werden verschiedene Lösungsmöglichkeiten vorgeschlagen und deren Tauglichkeit für den produktiven Einsatz im Kompositionswerkzeug eingeschätzt.

5.4.1 Rekursive Strukturen in komplexen Datentypen

Problemdarstellung Wie in Abschnitt 5.3 bereits dargestellt, können komplexe Datentypen rekursive Strukturen enthalten. Das heißt, dass ein komplexes Kindelement, das sich auf einer beliebigen Ebene der komplexen Datentyp-Struktur befindet, ein Element auf einer höheren Ebene als Datentyp referenziert. Ohne gesonderte Behandlung würde das Analysieren eines Datentypen mit einer Rekursion in eine Endlosschleife laufen.

Lösungsvorschläge

1. Abbruch der Rekursion:
Eine Lösungsmöglichkeit besteht darin, eine Rekursion nicht direkt weiter zu verfolgen, sondern den Rücksprung auf die höhere Ebene zu verhindern. Das Vorhandensein der Rekursion wird vermerkt und die Analyse weiterer Pfade des komplexen Datentypen fortgesetzt. Die konkrete Anzeige wird erst in der modellierten Endanwendung notwendig, da erst zu diesem Zeitpunkt konkrete Daten zur Verfügung stehen. Für die Service-Frontends ist deshalb der Hinweis auf eine Rekursion notwendig, aber mangels Daten ist die direkte Verfolgung der Rekursion nicht notwendig. Die Erkennung einer Rekursion wird durch einen einfachen Katalog-Mechanismus sichergestellt. Beim Durchlaufen eines komplexen Datentypen wird eine Liste mitgeführt, in der alle bereits betrachteten Datentypen katalogisiert werden. Wird ein Parameter gefunden, der durch einen Datentypen definiert ist, der sich bereits in der Liste befindet, wurde eine Rekursion gefunden. Diese wird vermerkt und später in der Visualisierung entsprechend angezeigt.

2. Rekursions-Entfernung beim Speichern im Service-Modell:
Eine weitere Möglichkeit stellt dar, die Datentyp-Strukturierung direkt im Service-Modell des Kompositionswerkzeugs anzupassen und in dieser Struktur Rekursionen auszuschließen bzw. bei der Erstellung des Modells zu entfernen. Erreicht werden kann das, indem die Schachtelung komplexer Datentypen als Schachtelung von Parametern umgesetzt wird. Die Datentypen an sich sind nicht mehr geschachtelt, sondern entsprechen einer flachen Liste. Jeder Parameter referenziert dabei seinen zugehörigen Datentypen aus der Menge der vorhanden Datentypen. Referenziert in der ursprünglichen Datentyp-Struktur ein Kind-Element ein Eltern-Element als Datentyp, dann wird diese Datentyp-Beziehung im neuen Modell als Referenz auf ein gemeinsames Datentyp-Objekt umgesetzt. Die Parameter selbst beziehen sich nicht rekursiv aufeinander. Durch diese Trennung der Struktur von der Datentypzugehörigkeit werden Endlosschleifen beim Parsen des Service-Modells verhindert. Der Parser muss nun lediglich die Parameter-Struktur unter Beachtung der referenzierten Datentypen umsetzen. Um dennoch ein korrektes Verhalten und eine akkurate Darstellung zur Laufzeit zur garantieren, muss die Rekursion im Modell vermerkt werden. Das kann beispielsweise mit einem entsprechenden Datentyp-Attribut realisiert werden.
Die Aufgabe, die Rekursionen aufzulösen, wird bei diesem Vorschlag in den WSDL-Parser verschoben, welcher dafür sorgen muss, dass:

- 1. Datentyp-Verschachtelungen als Parameter-Verschachtelungen umgesetzt werden.
- 2. Jeder Parameter genau einen Datentypen referenziert.
- 3. Beim Finden einer Rekursion die Referenz auf den bereits erstellten Datentypen der referenzierten Eltern-Klasse gesetzt wird.
- 4. Existenz der Rekursion im Modell vermerken.

Einschätzung Die Definition von rekursiven Datentyp-Referenzen ist bei umfangreichen Services keine Seltenheit und stellt für die Präsentation komplexer Daten ein wichtiges Konzept dar. Deshalb ist eine korrekte Darstellung der Rekursionen zur Laufzeit und ein Hinweis auf entsprechende Strukturen zur Design-Time der interaktiven Applikation notwendig. Eine entsprechende Betrachtung rekursiver Strukturen ist demnach erforderlich. Die Möglichkeit der direkten Behandlung während der Auswertung des Service-Modells stellt eine Möglichkeit dar. Die Behandlung müsste aber generell bei jeder Verwendung des Services erfolgen. Deshalb wäre es von Vorteil, wenn das Service-Modell des Kompositionswerkzeugs bereits keine Rekursionen mehr enthält. Bei diesem Ansatz findet die Auswertung rekursiver Strukturen beim Erstellen des Service-Modells und damit vor dem eigentlichen Kompositionsvorgang statt. Da die Modelle der Services lokal zwischengespeichert werden, ist eine erneute Auswertung der Rekursion bei mehrfacher Verwendung des Service nicht notwendig. Da die Zwischenspeicherung auf Ebene des Services durchgeführt wird, gilt das auch für die Arbeit mit unterschiedlichen Operationen eines Services.

5.4.2 Behandlung von Auswahldefinitionen bei komplexen Datentypen

Problemdarstellung Innerhalb der WSDL-Standard Spezifikation [CCMW01], basierend auf XML Schema [TBMMa], sind für die Reihenfolgen-Definition innerhalb komplexer Datentypen die folgenden Indikatoren vorgesehen:

- **<xsd:all>:** Die Kindelemente dürfen in beliebiger Reihenfolge auftreten. Jedes Kindelement darf allerdings nur genau einmal vorkommen.
- **<xsd:sequence>:** Die Kindelemente müssen in der definierten Reihenfolge auftreten. Jedes Kindelement darf entsprechend der Häufigkeitsangaben vorkommen
- **<xsd:choice>:** Nur eins der definierten Kindelemente darf vorkommen.

Während die <xsd:all> und die <xsd:sequence> Indikatoren bei der Verarbeitung der Services keine Probleme verursachen, ist die automatisierte Auswertung von <xsd:choice> problembehaftet. Der Choice-Indikator sieht vor, dass nur eins der definierten Kindelemente verwendet werden darf. Eine Entscheidung für eins der definierten Kindelemente muss getroffen werden.

Lösungsvorschläge

1. Wahl zur Design-Time:
Während der Design-Time kann erstmals direkt Einfluss auf die Auswertung des Services genommen werden. Sowohl innerhalb des Annotationstools als auch im Kompositionswerkzeug kann deshalb die Entscheidung getroffen werden, welches Kindelement einer XSD-Choice-Ordnung genutzt wird. Wird die Entscheidung zur Design-Time getroffen, muss an dieser Stelle bereits detailliert bekannt sein, welche Eingabedaten in der Endanwendung benötigt werden. Während der Annotation der Dienste ist das aber noch nicht unbedingt der Fall. Außerdem ist zu beachten, dass die Eingabemöglichkeiten vorab begrenzt und damit die Mächtigkeit der Anwendung gegenüber dem eigentlichen Service eingeschränkt wird. Beispielsweise beschäftigt jede Universität eine bestimmte Anzahl Mitarbeiter. Diese Mitarbeiter können entweder Professoren, wissenschaftliche Mitarbeiter oder Verwaltungsmitarbeiter sein. Das komplexe Element könnte also folgendermaßen definiert sein:

```
1 <xsd:element name="university_worker">
2   <xsd:complexType>
3     <xsd:choice>
4       <xsd:element name="professor" type="employee"/>
5       <xsd:element name="researcher" type="employee"/>
6       <xsd:element name="administration_staff" type="employee"/>
7     </xsd:choice>
8   </xsd:complexType>
9 </xsd:element>
```

Listing 5.1: XSD-Beispiel: Mitarbeiter einer Universität

Ist während der Annotation der Services oder während der Anwendungsgestaltung bereits bekannt, dass die Endanwendung nur die Eingabe von Professoren unterstützen soll, dann lässt sich bereits zum jeweiligen Zeitpunkt die Entscheidung treffen. Die Mächtigkeit der Anwendung wäre dann aber auch generell auf die Eingabe von Professoren beschränkt. Die Eingabe anderer Mitarbeiter wäre trotz entsprechender Unterstützung auf Service-Seite nicht mehr möglich.

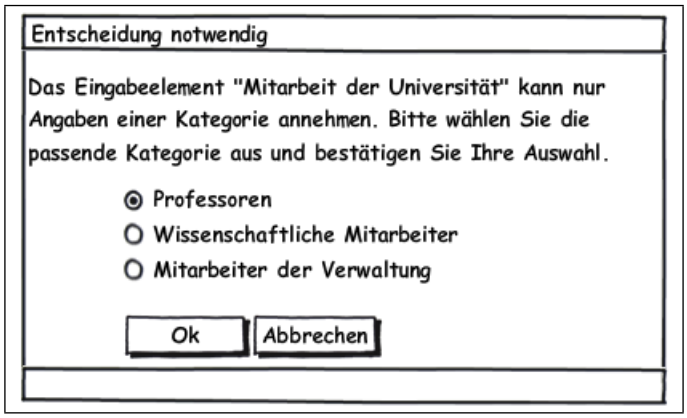


Abbildung 5.8: Beispiel für eine mögliche Entscheidungsfindung für ein XSD-Choice-Element

2. Implizite Entscheidung durch den Nutzer der Endanwendung:
Da der Service prinzipiell alle Möglichkeiten einer XSD-Choice-Ordnung annimmt, ist es möglich, die Entscheidung erst zur Programm-Laufzeit zu fällen. Dadurch würden der Umfang der Eingabemöglichkeiten und die Mächtigkeit der Anwendung nicht vorab eingeschränkt. Der Nutzer der Endanwendung kann dann direkt entscheiden, von welcher Art seine Eingabe ist. Prinzipiell werden alle Elemente einer XSD-Choice-Ordnung visualisiert. Eine Möglichkeit, vom Nutzer eine Entscheidung abzufordern, ist die Auswahl über eine ComboBox oder die Anzeige eines Pop-up. Abbildung 5.8 stellt eine mögliche Visualisierung für eine entsprechende Entscheidungsfindung dar:
Sind viele XSD-Choice-Elemente vorhanden, müsste auch eine entsprechend große Anzahl an Pop-ups angezeigt werden. Das würde den Nutzer der Anwendung stark belästigen. Möglich wäre die Zusammenfassung aller Entscheidungen in einem einzelnen Pop-up oder das Anbieten eines Wizards. Eine alternative Möglichkeit ist die Entscheidung implizit durch die vorgenommenen Eingaben des Anwendungsnutzers zu bestimmen. Diese Variante ist deutlich intuitiver, als die explizite Entscheidung per Pop-up oder Wizard. Nimmt der Nutzer eine Eingabe in das UI-Element von einem der angebotenen Parameter vor, werden die UI-Elemente der übrigen Parameter ausgeblendet oder deaktiviert. Der vorgestellte Lösungsansatz ist in beiden Varianten (explizite und implizite Entscheidung durch den Nutzer) nur mit der Hilfe von Annotationen sinnvoll umsetzbar. Eine Entscheidung zwischen verschiedenen Datentyp-Elementen ist für einen Endnutzer nur möglich, wenn die Namen der Elemente verständlich sind. Die Namen in einer XSD-Beschreibung werden aber meist unter technischen Aspekten vergeben und sind deshalb oftmals nicht direkt für den Endnutzer verständlich. Eine verständliche Benennung muss also sichergestellt werden, damit eine Entscheidungsfindung durch den Nutzer stattfinden kann. Durch die Feedback-Annotation ist das realisierbar. Mit Hilfe der Annotation lassen sich verständliche Namen und Tooltips realisieren, so dass dem Endnutzer nähere Informationen für eine Entscheidungsfindung zur Verfügung stehen. Bei der impliziten Variante der Entscheidungsfindung muss außerdem die dynamische Veränderung des Erscheinungsbilds in Abhängigkeit von der Eingabe in die UI-Elemente umgesetzt werden. Dazu kann die Appearance-Change-Rule-Annotation verwendet werden. Diese Annotation definiert Regeln und Aktionen, mit deren Hilfe das Erscheinungsbild von UI-Elementen in Abhängigkeit von der Eingabe in andere UI-Elemente verändert werden kann.

Einschätzung Durch beide vorgeschlagene Lösungen lassen sich Auswahldefinitionen mit Hilfe von XSD-Choice-Indikatoren behandeln. Die erste Lösungsmöglichkeit schränkt allerdings die Eingabemöglichkeiten im Vorfeld ein, was die Mächtigkeit der resultierenden Anwendung reduziert. Die Möglichkeiten des Service können somit nicht vollständig ausgenutzt werden. Außerdem wird eine genaue Kenntnis über die gewünschten Eingabemöglichkeiten vorausgesetzt, was zumindestens während der Annotation eines Service nicht gegeben sein muss. Um das zu verhindern, kann man die Entscheidung, welches Element einer XSD-Choice-Ordnung verwendet wird, auf den Nutzer der Endanwendung verlagern. Das kann sowohl manuell per Pop-up als auch automatisiert in Abhängigkeit von den Eingaben des Nutzer geschehen. Zu beachten ist, dass die zweite Lösungsmöglichkeit die sinnvolle Annotierung des Service im Vorfeld voraussetzt.

5.4.3 Kollisionen von Datentypendefinitionen und Annotationen

Problemdarstellung Innerhalb des ServFace-Projekts kann man die gültigen Eingabewerte eines Parameters nicht nur durch Datentyp-Definitionen einschränken sondern auch durch Annotationen. Dabei gibt es einige Annotationen, die sich mit den Beschränkungen der simplen Datentypen überschneiden. Die Tabelle 5.2 stellt die möglichen, direkten Überschneidungen dar:

Datentyp-Beschränkungen	Annotations-Beschränkungen
Pattern	Format
Pattern	Validation - Pattern Match
Min/Max In/Exclusive	Validation - Value Range
Enumeration	Validation - List of Accepted Values
Length	Format - ColumnCount
Total Digits	Format - ColumnCount

Tabelle 5.2: Direkte Überschneidungen zwischen Datentyp-Beschränkungen und Annotations-Beschränkungen.

Generell können sich die genannten Annotationen und die Datentyp-Beschränkungen gut ergänzen. Beispielsweise kann eine Validation-Annotation eingesetzt werden, um eine Eingabe-Überprüfung vor Aufruf des Services durchzuführen. Dadurch wird der Nutzer der Endanwendung bereits während er das Formular ausfüllt über Fehleingaben informiert und nicht erst nach dem Service-Aufruf. Da Datentyp-Beschränkungen und Annotations-Beschränkungen zu unterschiedlichen Zeitpunkten und durch unterschiedliche Personen erstellt werden, sind allerdings auch Konflikte möglich. Dabei kann man die folgenden drei Konfliktarten unterscheiden:

1. Eine Definition enthält die andere Definition vollständig.
2. Beide Definitionen überschneiden sich.
3. Beide Definitionen sind disjunkt.

Zu beachten ist, dass die Datentyp-Definition immer bindend ist. Ein Service akzeptiert nur Werte innerhalb der Definition. Andere Werte werden nicht angenommen und verursachen einen Fehler bei der Serviceausführung. Das bedeutet, dass in allen genannten Konfliktfällen der Datentyp-Definition der Vorrang gegeben werden muss. Definiert beispielsweise die Datentyp-Definition einen Wertebereich von 10 - 1000, dann ist auch nur dieser Wertebereich zulässig, selbst wenn eine Annotation einen Wertebereich von 5 - 2000 zulässt. Im Gegensatz dazu ist allerdings eine weitere Einschränkung durch die Annotation möglich. Legt die Annotation den Wertebereich auf 20 - 100 fest, dann ist dieser Wertebereich komplett durch die Datentyp-Definition abgedeckt und kann umgesetzt werden.

Lösungsvorschläge

1. Behandlung während der Service-Annotation
Die erste Möglichkeit, um Konflikte zwischen Datentyp-Einschränkungen und Annotations-Einschränkungen zu behandeln, besteht direkt während der Annotation der Services. Versucht der Annotator eine Annotation zu erstellen, die mit einer Datentyp-Definition kollidiert, dann kann das dafür verwendete Werkzeug direkt auf diesen Konflikt hinweisen und gegebenenfalls eine Lösung vorschlagen. Bei dieser Lösungsmöglichkeit spielt die Art des Konflikts eine untergeordnete Rolle, da diese direkt vermieden werden und Kollisionen beim eigentlichen Kompositionsprozess nicht auftreten können.

2. Automatische Konfliktbehandlung
Ist eine Konfliktvermeidung im Annotationstool nicht möglich, müssen Konflikte während des Kompositionsprozesses oder zur Laufzeit der Anwendung gelöst werden. Das sollte möglichst automatisiert passieren, um den Designer der interaktiven Anwendung nicht mit massenhaften Entscheidungen zu überfordern. Generell hat, wie bereits in der Problembeschreibung erwähnt, die Datentyp-Einschränkung immer Vorrang vor einer Annotations-Einschränkung, da diese Datentyp-Definitionen bindend für einen erfolgreichen Service-Aufruf sind. Folgende Konfliktlösungsmöglichkeiten bestehen:

1. **Annotations-Einschränkung vollständig in Datentyp-Einschränkung enthalten:**
   Schränkt eine Annotation eine Datentyp-Beschränkung weiter ein, dann kann die Annotations-Einschränkung vollständig angewendet werden. Sind beispielsweise durch den Datentyp Zahlenwerte zwischen 10000 und 60000 möglich, innerhalb der Anwendung soll der Nutzer aber nur Werte zwischen 15000 und 30000 wählen dürfen, dann kann die Einschränkung entsprechend der Annotations-Definition umgesetzt werden.
2. **Annotations-Einschränkung überschneidet Datentyp-Einschränkung:**
   In diesem Fall überschreitet die Annotations-Einschränkung die obere oder untere Grenze des durch den Datentypen definierten Wertebereichs. Beispielsweise definiert der Datentyp den Wertenbereich 10000 - 60000 und die Annotation erlaubt Werte zwischen 30000 und 80000. Eine mögliche Lösung ist nun die Kompromissfindung. Da die Datentyp-Beschränkungen bindend sind, muss die obere Grenze an die Datentyp-Beschränkung angeglichen werden. Da die untere Grenze innerhalb des Wertebereichs des Datentypen liegt, kann diese weiterhin verwendet werden. Die Kompromisslösung ist also ein gültiger Wertebereich zwischen 30000 und 60000.
3. **Annotations-Einschränkung liegt außerhalb der Grenzen der Datentyp-Definition:**
   Überschreitet die Annotations-Einschränkung sowohl die obere, als auch die untere Grenze der Datentyp-Definition, besteht die einzige Lösungsmöglichkeit darin, die Annotation zu ignorieren und den Wertebereich des Datentypen zu übernehmen.
4. **Annotations-Einschränkung und Datentyp-Definition beschreiben disjunkte Mengen:**
   In diesem Fall findet gar keine Überschneidung der Definitionen statt. Man kann daher von einem Fehler in der Anwendungsspezifikation ausgehen: Die Anwendung soll vollkommen andere Werte akzeptieren, als der eigentliche Service. In diesem Fall muss die Einschränkung der Datentyp-Definition verwendet oder der Anwendungsdesigner auf das Problem aufmerksam gemacht werden, damit dieser die Entscheidung treffen kann.

Für die oben genannten Konfliktlösungen werden nur Wertebereichs-Definitionen als Beispiele verwendet. Das liegt daran, dass bei diesen Konflikten am ehsten eine Lösung möglich ist. Stimmen Format- oder Musterdefinitionen nicht überein, dann ist im Regelfall eine Lösung nur möglich, wenn die Datentyp-Definition übernommen wird. Weiterhin ist immer zu hinterfragen, ob bei Konflikten der Kategorie 2, 3 und 4 kein Designfehler bei der Softwarespezifikation vorliegt. Generell sollten immer Services verwendet werden, die Daten im spezifizierten Bereich akzeptieren, so dass keine Kompromisslösungen notwendig sind.

Einschätzung Generell sollten Konflikte zwischen Datentyp-Definitionen und Annotationen vermieden werden. Der komfortabelste Weg ist dabei Konflikte direkt bei der Erstellung der Annotation zu verhindern. Wird eine Annotation angelegt, deren Definition mit einer Datentyp-Definition kollidiert, dann sollte bereits der Annotator darauf hingewiesen werden. Wenn möglich könnten gleichzeitig Lösungsvorschläge präsentiert werden. Ist eine Konfliktvermeidung innerhalb des Annotationstools nicht möglich, dann müssen die Konflikte werden der Anwendungserstellung oder zur Laufzeit der Anwendung behandelt werden. Die Möglichkeiten sind an dieser Stelle begrenzt und im Zweifelsfall ist immer der Datentyp-Definition der Vorrang zu gewähren.

5.4.4 Zusammenfassung

In diesem Kapitel wurden die Service-Frontends vorgestellt, die im Kompositionswerkzeug zur Visualisierung der Service-Operationen verwendet werden. Dabei wurde besonders auf die Darstellung der Parameter eingegangen, da diese die zentralen Interaktionselemente sind. Parameter mit komplexem Datentyp mussten aufgrund ihrer Strukturierung gesondert behandelt werden. Treten umfangreiche Datentyp-Strukturen auf, besteht die Gefahr, dass die Service-Frontends schnell unübersichtlich werden und der Platzbedarf enorm steig. Weiterhin ist die Darstellung von Verschachtelungen für Ausgabeparameter recht schwierig. Insbesondere da jedes einzelne Datentyp-Element direkt auswählbar sein muss, um Datenflüsse zu definieren. Für einfache komplexe Datentypen, die nur eine Schachtelungsebene und eine geringe Anzahl an Unterelementen definieren, ist eine einfache Darstellung per Tabelle oder Gruppierungsdarstellung wie in Abschnitt 5.3 beschrieben am einfachsten. Wichtig ist also die Definition von Grenzwerten, die Fixpunkte zum Wechsel zur alternativen Darstellung definieren. Für eine alternative Darstellung umfangreicher komplexer Parameter wurden spezielle Visualisierungsmöglichkeiten konzipiert. Zu beachten ist, dass sich die Visualisierung im Kompositionswerkzeug nur minimal von der Darstellung in der Endanwendung unterscheiden darf. Das UI der Endanwendung muss schon während der Anwendungsgestaltung erkennbar sein.

Außerdem fand in diesem Kapitel eine Diskussion über mögliche Probleme während der Visualisierung der Frontends statt. Für diese wurden mehrere Lösungsmöglichkeiten vorgeschlagen, die anschließend bewertet wurden.

6 UMSETZUNG DER SERVICE-ANNOTATIONEN

In Abschnitt 2.3.3 wurden Annotationen als zusätzliche Informationsquelle vorgestellt. Sie dienen zur Verbesserung des Visualisierungs- und Generierungsprozesses für interaktive servicebasierte Anwendungen. Für die Implementierung des Kompositionswerkzeuges ist es wichtig, dass die Auswirkungen der Annotationen auf einzelne Service-Elemente betrachtet werden. Dafür ist es sinnvoll, eine zur in [JPS+09] (siehe auch: Anhang A.3.2) genannte Grund-Unterteilung alternative Kategorisierung nach der Art der Auswirkungen zu definieren. Auf Basis dieser Kategorisierung werden dann die Auswirkungen dieser Gruppen betrachtet und an verschiedenen Beispielen erläutert. Weiterhin gibt es Annotationen, die sich nicht direkt im Kompositionswerkzeug umsetzen lassen. Für diese Annotationen muss eine passende, visuelle Repräsentationsmöglichkeit gefunden werden. Die genannten Punkte werden in den folgenden Abschnitten dieses Kapitels abgehandelt.

6.1 DIFFERENZIERUNG DER VERWENDETEN ANNOTATIONEN

Die innerhalb des ServFace-Projektes verwendeten Annotationen lassen sich neben der Unterteilung in die [JPS+09] (siehe auch: Anhang A.3.2) genannten Grund-Kategorien auch nach ihren Auswirkungen auf die modellierte Anwendung unterteilen. Die UI-Annotationen haben unmittelbaren Einfluss auf die Darstellung und werden direkt im zugrundeliegenden Modell umgesetzt. Im Gegensatz dazu werden funktionale Annotationen im Anwendungsmodell nur vermerkt und im generierten User-Interface abstrahiert dargestellt. Dazwischen platzieren sich die hybriden Annotationen, die begrenzte, direkte Auswirkungen auf das Model und das User-Interface haben. Allerdings können diese Annotationen nicht vollständig im Modell aufgelöst werden. In diesem Abschnitt werden die Annotations-Typen detailliert beschrieben und die Annotationen den entsprechenden Typen zugeordnet.

6.1.1 UI-Annotationen

Die in diesem Abschnitt betrachteten UI-Annotationen werden vor allem zur weiterführenden Beschreibung von Aussehen, Anordnung und Formatierung der Elemente eines Webservices verwendet. Außerdem können zusätzliche Informationen, wie beispielsweise ein Name oder eine weiterführende Beschreibung, zur Verfügung gestellt werden. Die Auswirkungen von UI-Annotationen beschränken sich dabei nicht nur auf die Darstellung im Kompositionswerkzeug, sondern manifestieren sich direkt im zugrundeliegenden Composite Application Model (CAM). Durch die Erzeugung neuer CAM-Elemente, die Veränderung vorhandener CAM-Elemente oder die Modifizierung der Struktur des CAMs, werden die durch die Annotationen zur Verfügung gestellten Beschreibungen fest im CAM integriert. Deshalb müssen UI-Annotationen nach ihrer Verarbeitung auch nicht mehr explizit weitergeführt werden, um sie im späteren Generierungsprozess oder beim Laden des CAMs erneut auszuwerten.
Folgende Annotationen lassen sich den UI-Annotationen zuordnen:

- Feedback
- Group&Order
- Format
- Example Data
- Visual Property
- Mime-Type
- Bundle
- Design Templates

6.1.2 Funktionale Annotationen

Funktionale Annotationen beschreiben in den meisten Fällen eine zusätzliche Funktion, die für ein Webservice-Element zur Verfügung gestellt wird. Da zur Design-Time diese Zusatzfunktionen nicht ausgeführt werden können, wird im Kompositionswerkzeug nur die Existenz dieser Annotationen dargestellt. Diese Darstellung erfolgt durch eine metaphorische Abstraktion der Funktion und die Markierung der durch eine Annotation beeinflussten Elemente. Im Composite Application Model wird nur die Existenz einer funktionalen Annotation vermerkt, um eine Verarbeitung im weiteren Generierungsprozess zu ermöglichen. Beim Laden des Modells wird der Vermerk über die Existenz der Annotation zur Wiederherstellung der abstrakten Darstellung benötigt.
Zu den funktionalen Annotationen gehören:

- Validation
- Suggestion
- Special Data Type
- Mandatory Field
- Semantic Data Type Relation
- Authentication
- Appearance Change Rule
- Form Completion
- Continous Update

6.1.3 Hybride Annotationen

Die letzte Gruppe der Annotationen vermischt die Aspekte der UI- und funktionalen Annotationen. Hybride Annotationen beeinflussen direkt das Anwendungsmodell und manifestieren sich auch in diesem. Allerdings ist auch nach der Verarbeitung der Annotationen ein Vermerk über deren Existenz notwendig, um die weitere Verarbeitung im Generierungsprozess der Endanwendung zu gewährleisten. Grund dafür ist, dass hybride Annotationen sowohl Auswirkungen auf die Darstellung der Webservice-Elemente haben, als auch zusätzliche Funktionen beschreiben, die später umgesetzt werden müssen.
Zu den hybriden Annotationen zählen:

- Units inkl. Conversion Rule
- Enumeration
- Defaul Value

6.2 AUSWIRKUNGEN DER ANNOTATIONEN

Nachdem die Annotationen als Hilfsmittel für die Service-Frontend-Generierung eingeführt wurden und eine Gruppierung der Annotationen nach Ihren Auswirkungen auf die interaktive Applikation erfolgt ist, wird im folgenden Abschnitt erläutert, wie die verschiedenen Annotationen umgesetzt werden. Neben einer abstrakten Beschreibung der Auswirkungen erfolgt anhand von 6 ausgewählten Annotationen eine detaillierte Erläuterung der jeweiligen Umsetzung. Die Beispielannotationen sind:

- Group (UI-Annotation)
- Text Feedback (UI-Annotation)
- Format (UI-Annotation)
- Form Completion (funktionale Annotation)
- Validation (funktionale Annotation)
- Enumeration (hybride Annotation)

Da die funktionalen Annotationen zusätzlich abstrakt visualisiert werden müssen, findet im Rahmen der Umsetzungsbeschreibung auch die Darstellung von möglichen Metaphern statt.

6.2.1 Umsetzung der UI-Annotationen

Mit UI-Annotationen soll das Visualisierungsergebnis bei der automatischen Generierung der User-Interfaces verbessert werden. Neben Aussehen, Anordnung und Erscheinungsbild werden auch Aspekte wie Benennung und Formatierung der UI-Elemente näher spezifiziert. Weiterhin ist es möglich Beispieldaten anzugeben um dem Nutzer des Kompositionswerkzeugs ein besseres Verständnis für die zu erwartenden Daten in den annotierten Elementen zu geben. Die UI-Annotationen verändern direkt das zugrundeliegende Datenmodell (CAM). Dabei wurden folgende mögliche Auswirkungen identifiziert:

- Erzeugung eines neuen CAM-Elements:
  - UI-Elemente (Contentgroup, Interactor,...)
  - Seite (Page)
  - Eigenschaften-Definition (Property)
  - Übergänge (FlowTransitions)
  - Element-Beziehungen (ModelRelation)
- Änderung von Eigenschaften:
  - Änderung des Darstellungs-Typen
  - Änderung von Eigenschaften (Property)
  - Änderung der Struktur
  - Änderung von Übergängen
  - Änderung von Element-Beziehungen

Nachdem das Anwendungsmodell durch die Umsetzung der Annotationen verändert wurde, wirken sich diese Änderungen direkt auf die Darstellung der Service-Elemente im Kompositionswerkzeug aus. Im Detail wird eine UI-Annotation wie folgt abgearbeitet. Beim Einlesen eines Service-Elements wird überprüft, ob diesem Element Annotationen zugeordnet sind. Ist eine Annotation vorhanden, wird die Annotation gelesen und die Auswirkungen ausgewertet. Anschließend werden diese Auswirkungen in das Composite Application Model eingebracht. Durch das Hinzufügen oder Ändern von Elementen wird die grafische Oberfläche des Kompositionswerkzeugs automatisch über die Änderung benachrichtigt. Nach Empfang und Auswertung der Benachrichtigung erfolgt die Änderung beim entsprechend zugehörigen UI-Element. An den folgenden Annotationen werden die Auswirkungen von UI-Annotationen näher erläutert:

Group:

- **Annotation für:** Services, Service-Operationen, komplexe Datentypen
- **Beschriebene Auswirkungen:** Gruppierung und Ordnung von Elementen
- **Visuelle Auswirkungen:**
  - Positionierung betroffener Elemente in relativer Nähe zueinander
  - Umrahmung der Elementgruppe
  - Anzeige der Elemente in der spezifizierten Reihenfolge und Ordnung
- **Auswirkungen auf das Modell:**
  - Erstellen einer neuen Contentgroup
  - Einfügen der betroffenen Elemente in die Contentgroup in der spezifzierten Reihenfolge

Der folgende Code-Ausschnitt stellt die Umsetzung der Group-Annotation als Pseudo-Code dar.

```
1  If (annotation = Group)
2  {
3          //CAM
4          create new Contentgroup
5          sort Elements via orderspecification
6          put Elements in Contentgroup
7
8          //UI
9          //realize CAM actions
10         Create new Stackpanel
11         put Elements in Stackpanel according to CAM
12         put border around stackpanel
13 }
```

Listing 6.1: Pseudo-Code: Umsetzung Group-Annotation

Feedback

- **Annotation für:** Services, Service-Operationen, Parameter, komplexe Datentypen, Lokale Datentypen, Globale Datentypen
- **Beschriebene Auswirkungen:** Darstellung einer textuellen oder multimedialen Erläuterung zur näheren Definition eines Elements oder zur Rückmeldung von Status und Fehlern an den Nutzer.

- **Visuelle Auswirkungen:**
  - Bei näherer Definition eines Elements:
    - Darstellung der Beschreibung in unmittelbarer Nähe zum annotierten Element
    - Bei multimedialem Feedback Darstellung unter Beachtung des MIME-Types
  - Bei Fehler-, Status- oder Warnungsmeldung:
    - Darstellung der Meldung in dem für eine Service-Operation vorgesehen Bereich für Fehlermeldungen
    - Bei multimedialem Feedback Darstellung unter Beachtung des MIME-Types
- **Auswirkungen auf das Modell:**
  - Erstellen eines Properties für das annotierte Element
  - Benennung des Properties entsprechend der Art des Feedbacks
  - Eintragen des String-Wertes oder der Multimedia URI und des zugehörigen MIME-Types

Der folgende Code-Ausschnitt stellt die Umsetzung der Feedback-Annotation als Pseudo-Code dar.

```
If (annotation = Feedback)
{
        //CAM
        create new Property
        name Property according to FeedbackType
        if (Feedback == TextFeedback)
        {
                value = Feedback_TextString
        }
        if (Feedback == MultimediaFeedback)
        {
                value = URI§MIMEType

        }
        Put Property in Propertylist of annotated Element
        //UI
        //realize CAM actions
        if (FeedbackType == (Label || Help || Button)
        {
                        if (Feedback == TextFeedback)
                {
                        create textblock near Element
                        content of textblock = value of Property
                }
                if (Feedback == MultimediaFeedback)
                {
                        create UI-Element near Element according to MIMEType
                        fill URI of new UI-Element with URI from Property value
                }
        }
        if FeedbackType == (Error || Warning || Status)
        {
                if (Feedback == TextFeedback)
                {
                        create textblock in service_operation_error_area
                        content of textblock = value of Property
                }
                if (Feedback == MultimediaFeedback)
                {
                        create UI-Element in service_operation_error_area according to
                            MIMEType
                        fill URI of new UI-Element with URI from Property value
                }
        }
}
```

Listing 6.2: Pseudo-Code: Umsetzung Feedback-Annotation

Format

- **Annotation für:** Parameter, Globale Datentypen, lokale Datentypen, Elemente komplexer Datentypen
- **Beschriebene Auswirkungen:** Formatierung der Darstellung von Ein- und Ausgabe-Elementen entsprechend des vorgegebenen Formats
- **Visuelle Auswirkungen:**
  - Anzeige mehrerer Ein- bzw. Ausgabeelemente
  - Trennung der Elemente durch Format-Delimiter
- **Auswirkungen auf das Modell:**
  - Erstellen eines Interactors für jeden Formatteil
  - Erstellen eines Interactors für jeden Delimiter zwischen den Formatteilen
  - Erstellen von ModelRelations zur Darstellung, dass die einzelnen Felder zu einem einzelnen Parameter gehören

Der folgende Code-Ausschnitt stellt die Umsetzung der Format-Annotation als Pseudo-Code dar.

```
1  If (annotation = Format)
2  {
3          //CAM
4          create new ContentGroup
5          name contentgroup like operation parameter
6          foreach(Formatpart in Formatdefinition)
7          {
8                  create new Interactor
9                  name Interactor like contentgroup + 'formatpart_' + counter(i, i=1, i++)
10                 set parameter to operation parameter
11                 set uiType to textbox
12                 create new property
13                 name property = content
14                 property value = definition of format part
15         }
16         foreach(delimiter in Formatdefinition)
17         {
18                 create new Interactor
19                 name Interactor like contentgroup + 'formatpart_' + counter(i, i=1, i++)
20                 set uiType to textblock
21                 create new property
22                 name property = content
23                 property value = delimiter
24         }
25         put Elements in Contentgroup according to format definition
26         create modelRelations to express dependency between each formatpart and the one
               parameter
27
28         //UI
29         //realize CAM actions
30         create UI-Element for each format part
31         create UI-Element for each delimiter part
32         put elements in relative closeness to each other
33 }
```

Listing 6.3: Pseudo-Code: Umsetzung Format-Annotation

6.2.2 Metaphern und Abstraktionen der funktionalen Annotationen

Funktionale Annotationen werden nicht direkt im zugrundeliegenden Modell umgesetzt. Dennoch sollen die Auswirkungen bzw. zumindestens die betroffenen Elemente erkennbar gemacht werden. Aus diesem Grund ist es nötig von der Funktion, die eine funktionale Annoation repräsentiert, zu abstrahieren. Um die Funktionen zu illustrieren müssen passende Metaphern gefunden werden. Im Nachgang können dann aus diesen Metaphern Icons gewonnen werden, die zum markieren betroffener Elemente eingesetzt werden können. Um die spätere Darstellung der Metapher als Icon zu ermöglichen, müssen möglichst einfache, gut erkennbare Gegenstände oder Handlungsabläufe gewählt werden. Diese sollten im Idealfall intuitiv die Begrifflichkeit und Funktion der Annotation widerspiegeln. Als Metaphergrundlage wurde die Darstellung der Annoationen durch repräsentative Gegenstände ausgewählt, welche die Funktion illustrieren oder üblicherweise mit der Funktion in Verbindung gebracht werden. Folgende Metaphern sind für die Annotationen definiert:

- Authentication -> Vorhängeschloss
- Validation -> Klemmbrett mit abgehaktem Kästchen
- Suggestion -> Bildliche Darstellung der Funktion: Kästchen mit angefangener Eingabe und gelbem Komplettierungsvorschlag
- Mandatory -> Umkehrung der Darstellung: Markiert werden alle Felder die nicht als notwendig (Mandatory) markiert sind und damit gelöscht werden können. Die dafür gefundene Metapher ist ein Papierkorb.
- Appearance Change -> Darstellung durch 2 Felder im Vorher-/Nachher-Vergleich
- Form Completion -> Darstellung durch ein leeres und ein ausgefülltes Blatt Papier für einen Vorher-/Nachher-Vergleich
- DataType Relation -> Bildliche Darstellung der Verbindung von 2 Feldern (2 Pfeile verbinden die Felder)
- Continous Update -> Nutzung der verbreiteten Doppelpfeil-Metapher für Datenübertragung/-Synchronisation

Entsprechend diesen Metaphern wurden die folgenden Icons erstellt. Die maximale Größe der Icons war auf 24x24 Pixel begrenzt:

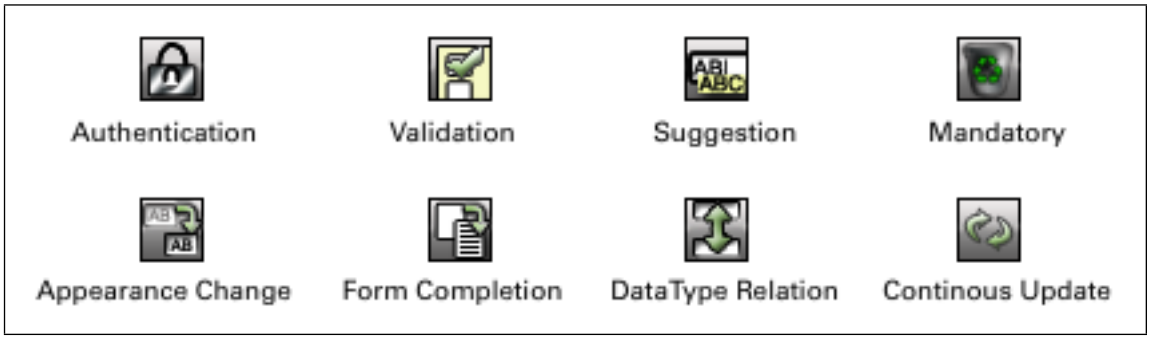


Abbildung 6.1: Icons als metaphorische Umsetzung funktionaler Annotationen

Die Metaphern inkl. der Icons werden durch eine Befragung von Endnutzer verschiedener Nutzergruppen evaluiert. Besonderes Augenmerk wird dabei auf die Intuitivität und Verständlichkeit der Icons gelegt. Die Testpersonen sollen möglichst ohne das Studieren der Beschriftung oder einer Erläuterung die dargestellte Funktion identifizieren können. Die Evaluierungsergebnisse der Nutzerstudie sind in Abschnitt 9.1.1 zu finden. Die Ergebnisse wurden direkt in die Verbesserung der Metaphern und der Icons einbezogen. Letztendlich wird erwartet, dass die intuitive, metaphorische Abstraktion der Funktionen gepaart mit einer Erläuterung per Tooltip zu einem guten Repräsentationsergebnis führt und die Nutzer des Kompositionswerkzeugs die einzelnen funktionalen Annotationen leicht identifizieren können.

6.2.3 Umsetzung funktionaler Annotationen

Der ServFace-Builder ist als Kompositionswerkzeug für Webservice-Operationen ausgelegt. Die angezeigten Frontends dienen demnach nur zur Illustration des User-Interfaces in der resultierenden interaktiven Anwendung, um eine präsentationsorientierte Komposition der Operationen zu ermöglichen. Weder die Funktionalitäten der Webservice-Operationen noch die zusätzlichen Funktionen durch die funktionalen Annotationen können direkt im Kompositionswerkzeug dargestellt werden. Aus diesem Grund werden funktionale Annotationen auch nicht explizit im Composite Application Model umgesetzt, sondern lediglich zur späteren Verarbeitung im Annotations-Modell referenziert. Dennoch ist, wie bereits in Abschnitt 6.2.2 erwähnt, eine abstrahierte Darstellung der Funktionen und eine Markierung der beteiligten UI-Elemente erwünscht. Somit wird der Nutzer des Kompositionswerkzeugs auf eine spätere, zusätzliche Funktion in der Endanwendung hingewiesen. Außerdem haben funktionale Annotationen auch tiefergreifendere Auswirkungen. Zum Beispiel dürfen mit der Mandatory-Annotation annotierte Elemente während der Komposition im Kompositionswerkzeug nicht gelöscht werden. Die Darstellung der funktionalen Annotationen erfolgt gebündelt innerhalb einer Annotationsleiste direkt unter der Titelzeile der Service-Operationen. Innerhalb dieser Leiste werden die Icons aller funktionaler Annotationen angezeigt, die auf einem der Elemente innerhalb der entsprechenden Service-Operation aktiv sind. Fährt der Nutzer mit der Maus über ein Icon, dann werden alle von der Annotation beeinflussten Elemente durch einen farblichen Overlay so lange markiert, wie sich die Maus über dem Icon befindet. Der Nutzer kann sich somit einen Überblick über die durch funktionale Annotationen bedingten Zusatzfunktionen in seiner resultierenden Endanwendung verschaffen und auch die jeweils beteiligten Elemente identifizieren. Die folgenden beiden Beispiele verdeutlichen die Umsetzung der funktionalen Annotationen:

Validation:

- **Annotation für:** Parameter, lokale Datentypen, Globale Datentypen, Komplexe Datentypen, Elemente komplexer Datentypen
- **Beschriebene Auswirkungen:** Eingabevalidierung für Webservice-Elemente
- **Visuelle Auswirkungen:**
  - Darstellung des Validation-Icons
  - Overlay der zu validierenden Elemente
  - Darstellung der Validierungsregeln als Tooltip des Validation Icons
- **Auswirkungen auf das Modell:**
  - Einfügen in die Liste der activatedAnnotations beim annotierten Element

Der folgende Code-Ausschnitt stellt die Umsetzung der Validation-Annotation als Pseudo-Code dar.

```
1  If (annotation = Validation)
2  {
3          //CAM
4          add Annotation to activatedAnnotations List hold by annotated element
5
6          //UI
7          //realize CAM actions
8          load and display icon for validation annotation
9          mark annotated elements
10         add validation rules to validation tooltip
11 }
```

Listing 6.4: Pseudo-Code: Umsetzung Validation-Annotation

Form Completion:

- **Annotation für:** Service-Operationen, komplexe Datentypen
- **Beschriebene Auswirkungen:** Formularvervollständigung nach Ausfüllen bestimmter Formularelemente (Trigger-Elemente)
- **Visuelle Auswirkungen:**
  - Darstellung des FormCompletion-Icons
  - Overlay der Trigger-Elementer
  - Overlay der zu vervollständigenden Elemente
  - Beschreibung des Ursprungsorts der Daten für das Ausführen der Form Completion
- **Auswirkungen auf das Modell:**
  - Einfügen in die Liste der activatedAnnotations beim annotierten Element

Der folgende Code-Ausschnitt stellt die Umsetzung der FormCompletion-Annotation als Pseudo-Code dar.

```
If (annotation = Validation)
{
        //CAM
        add Annotation to activatedAnnotations List hold by annotated element

        //UI
        //realize CAM actions
        load and display icon for formCompletion annotation
        mark trigger elements
        mark affected elements
        add hint to formCompletion source to tooltip
}
```

Listing 6.5: Pseudo-Code: Umsetzung FormCompletion-Annotation

Ein Beispiel für die Anzeige der funktionalen Annotationen ist in der Abbildung 6.2 zu sehen:

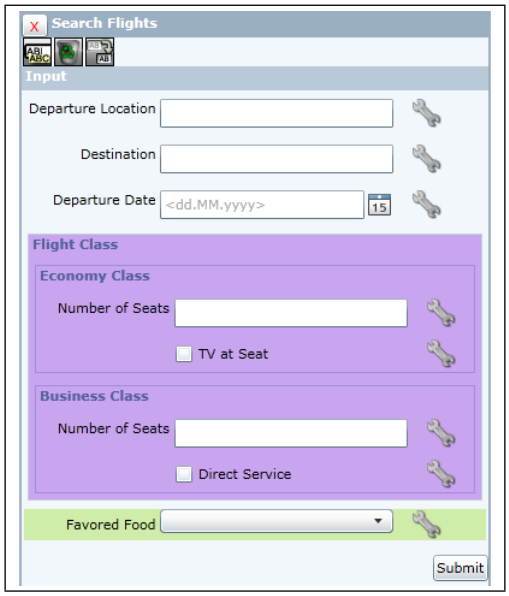


Abbildung 6.2: Anzeige der funktionalen Annotation “Appearance Change“ im ServFace Builder

6.2.4 Umsetzung der Hybrid-Annotationen

Hybride Annotationen wirken sich direkt auf die Darstellung von UI-Elementen aus und repräsentieren eine zusätzliche Funktion. Die Auswirkung auf die Darstellung wird vollständig im Anwendungsmodell umgesetzt. Der funktionale Aspekt wird bei der späteren Anwendungsgenerierung umgesetzt. Anders als bei den funktionalen Annotationen ist aber keine repräsentative, metaphorische Darstellung notwendig. Die hybriden Annotation werden auf Grund der optischen Auswirkungen direkt dargestellt. Als Beispiel zur näheren Erläuterung der Umsetzung hybrider Annotationen soll die Enumeration-Annotation dienen:

Enumeration:

- **Annotation für:** Parameter, lokale Datentypen, Globale Datentypen, Elemente komplexer Datentypen
- **Beschriebene Auswirkungen:** Vorgabe einer Menge gültiger Werte für ein Webservice-Element
- **Visuelle Auswirkungen:**
  - Bestimmung des optimalen Eingabeelementypen anhand der Enumerationsdefinition
  - Darstellung des Elements durch den ausgewählten Elementtypen
  - Hinzufügen der Enumerationswerte zur Anzeige
- **Auswirkungen auf das Modell:**
  - Änderung des UI-Element-Typen entsprechend der Enumerationsdefinition
  - Einfügen in die Liste der activatedAnnotations beim annotierten Element

Der folgende Code-Ausschnitt stellt die Umsetzung der Enumeration-Annotation als Pseudo-Code dar.

```
1  If (annotation = enum)
2  {
3          //CAM
4          //just examples
5          if (# of items >3 && <20)
6          {
7                  elementtype = combobox
8          }
9
10         if (# of items >20 && <60)
11         {
12                 elementtype = advancedCombobox
13         }
14
15         if (# of items >60 && <200)
16         {
17                 elementtype = Listbox
18         }
19
20         else
21         {
22                 elementtype = textbox
23         }
24
25         add Annotation to activatedAnnotations List hold by annotated element
26
27         //UI
28         //realize CAM actions
29         display ui-elements according to enumeration definition
30         }
```

Listing 6.6: Pseudo-Code: Umsetzung Enumeration-Annotation

6.3 ZUSAMMENFASSUNG

In dem vorliegenden Kapitel wurde die Umsetzung der verwendeten Annotationen besprochen. Zunächst erfolgte eine Kategorisierung der Annotation nach ihren Auswirkungen auf den Visualisierungsprozess. Dabei konnten UI-Annotationen, funktionale Annotationen und Hybrid-Annotationen identifiziert werden. UI-Annotationen werden direkt im Anwendungsmodell umgesetzt und spiegeln sich durch visuelle Veränderungen in den Frontends wieder. Funktionale Annotationen stellen zusätzliche Funktionalitäten zur Verfügung, die allerdings während der Anwendungserstellung im Kompositionswerkzeug noch nicht ausgeführt werden können. Deshalb erfolgt die Darstellung dieser Annotationen nur abstrahiert. Hybride-Annotationen sind eine Mischform aus beiden Annotationskategorien. Nach der Kategorisierung wurden die Auswirkungen der einzelnen Annotationskategorien an verschiedenen Beispielen erläutert. Für die funktionalen Annotationen wurden zusätzlich Metaphern und Icons für die visuelle Repräsentation entworfen.

Teil III

Praktische Konzeption und Umsetzung

7 KONZEPTION EINER VISUALISIERUNGS-ENGINE FÜR SERVICE-FRONTENDS

Die innerhalb des Tools verwendeten Service-Frontends stellen die Basis für den Kompositionsprozess dar. Die Frontends sollen automatisch zur Laufzeit des Kompositionstools visualisiert werden. Innerhalb des Tools platziert und kombiniert der Nutzer die visualisierten Frontends. Im Hintergrund wird die Komposition des Nutzers in einem Anwendungsmodell realisiert, dass als Input für den weiteren Anwendungsgenerierungsprozess verwendet wird. Für die Visualisierung und Integration der Frontends müssen verschiedene Komponenten für das Kompositionstool konzipiert und implementiert werden. In dem folgenden Abschnitt werden nach einer Analyse der Anforderungen an die Visualisierungskomponenten die einzelnen Bestandteile identifiziert und beschrieben sowie Konzepte für deren Realisierung vorgestellt.

7.1 ANFORDERUNGS-ANALYSE

Das Kompositionswerkzeug soll in einer drei Schichtenarchitektur umgesetzt werden, die an die Model-View-Controller(MVC) -Architektur angelehnt ist. Die Visualisierungskomponenten sind Kernbestandteil des Kompositionswerkzeugs und Teil der Controller-Schicht. Sie bilden die Schnittstelle zwischen dem Service-Repository, dem User-Interface des Tools und dem Anwendungsmodell im Hintergrund. Im folgenden Abschnitt werden zunächst die funktionalen Anforderungen und deren zugehörige Bestandteile definiert. Anschließend erfolgt die Formulierung von nichtfunktionalen Qualitätsanforderungen. Da es sich bei dem Kompositionswerkzeug um eine Browserapplikation handelt, sind vor allem spezielle nichtfunktionale Anforderungen zu beachten.

Den Einstiegspunkt in den Visualisierungsprozess stellt das Service-Modell des Kompositionswerkzeugs dar, dass in Abschnitt 2.3.1 kurz vorgestellt wurde. Erzeugt wird dieses Modell von der Service-Repository-API. Die eigentlichen WebService werden auf einem entfernten Server gelagert. Die Service-Repository-API des Kompositionswerkzeugs nutzt selbst einen Webservice um alle notwendigen Informationen zum Aufbau des Service-Modells abzurufen.

7.1.1 Serviceabbildung und -zwischenspeicherung

Die Serviceabbildung und -zwischenspeicherung ist der erste Schritt bei der Visualisierung der Service-Frontends. Diese Anforderung setzt sich aus den folgenden Teilanforderungen zusammen:

R1.1: Abruf und Speicherung des Service-Modells
Wird eine Service-Operation zur Komposition hinzugefügt, dann soll zur Verringerung des Kommunikationsaufwands immer das Modell des gesamten Service von der Service-Repository-API angefordert und auf dem Client zwischengespeichert werden.

R1.2: Bereitstellen der Service-Modelle
Auf Anfrage müssen die Service-Modelle anderen Programmkomponenten zur Verfügung gestellt werden.

R1.3: Service/Operationsliste
Listen aller verfügbaren Services und Operationen müssen anderen Programmkomponenten bereit gestellt werden. Die Listen beziehen sich nicht nur auf lokal zwischengespeicherte Services, sondern müssen alle im Service-Repository vorhanden Services und deren Operationen enthalten.

7.1.2 Serviceauswertung und -umsetzung

Der zweite Schritt im Visualisierungsprozess ist, die Komponenten der Services in das Anwendungsmodell (siehe Abschnitt 2.3.2) einzubringen und damit die Datenstruktur für die Frontend-Visualisierung zu erstellen. Dies geschieht sowohl unter Beachtung der Service-Komponenten selbst, als auch der an die Service-Komponenten angehängten Annotationen. Für diese Anforderung lassen sich die folgenden Bestandteile identifizieren:

R2.1: Analyse des Service-Modell
Die Visualisierungskomponente muss das gegebene Service-Modell analysieren und die einzelnen Bestandteile erkennen und auswerten können.

R2.2: Erstellung und Integration des Anwendungsmodell
Auf Basis der Servicekomponenten muss eine geeignete Datenstruktur zur Speicherung im Anwendungsmodell angelegt werden. Diese umfasst unter anderem Strukturierungs- und Interaktionselemente. Die Datenstruktur muss anschließend gemeinsam mit dem Service-Modell in das Anwendungsmodell eingebracht werden.

R2.3: Wahl passender UI-Elemente
Jedem Service-Element ist für die Darstellung im Service-Frontend ein passendes User-Interface-Element zuzuordnen. Bei der Wahl des UI-Elements sind neben den Gegebenheiten des Service-Elements auch Annotationen, Plattform-Spezifikationen und die in Kapitel 4 formulierten Design-Empfehlungen zu beachten.

R2.4: Auswertung und Einbringung der Annotationen
Die dem Service beigefügten Annotationen sind während des Visualisierungsprozesses zu beachten, auszuwerten und so weit wie möglich im Anwendungsmodell umzusetzen bzw. zumindest zu vermerken.

R2.5: Speicherung von Konfigurations-Parametern
Alle Konfigurationsparameter und Relationen von Anwendungsmodell-Elementen müssen identifiziert und im Modell gespeichert werden.

R2.6: Synchronität View/Modell
Die Anzeige des Tools muss immer und ausschließlich den aktuellen Stand des Anwendungsmodells widerspiegeln. Änderungen am aktuellen Stand der Komposition müssen direkt im Modell reflektiert werden.

7.1.3 Integration der Visualisierungskomponenten in das Kompositionstool

Die Visualisierungskomponenten müssen nahtlos mit den anderen Programmschichten des Kompositionstools zusammen arbeiten. Entsprechende Kommunikationsschnittstellen und -möglichkeiten sind zu integrieren. Diese Anforderung lässt sich in folgende Details zerlegen:

R3.1: Änderung des Anwendungsmodells
Änderungen am Anwendungsmodell dürfen ausschließlich von der Controller-Schicht initiiert werden.

R3.2: Änderungen durch den Tool-Nutzer
Die Änderungen des Tool-Nutzers sind durch geeignete Möglichkeiten an den Controller zu übermitteln, welcher die Änderungen interpretieren und die Integration in geeigneter Weise initiieren muss.

R3.3: Zusätzliche Informationen
Der Controller muss dem User-Interface zusätzliche Informationen, wie beispielsweise die Darstellung funktionaler Annotationen, in geeigneter Weise bereitstellen.

R3.4: Kommunikation mit dem Modell
Alle Änderungen an der Komposition müssen in das Modell integriert werden. Die Übermittlung der Änderungen ist auf geeignete Weise zu realisieren.

7.1.4 Nichtfunktionale Anforderungen an die Visualisierungskomponenten

Nichtfunktionale Anforderungen formulieren neben den Funktionalitäten Qualitätsmerkmale einer Software. Diese Qualitätsmerkmale sind unter anderem in der ISO/IEC 9126 [ISO01] und der (ehemaligen) DIN 66272 niedergeschrieben. Im folgenden Abschnitt werden die wichtigsten Qualitätsmerkmale für die Visualisierungskomponenten formuliert:

NFR1: Funktionalität

- Die Visualisierungskomponente erfüllt die in den Anforderungen festgelegten Funktionalitäten.
- Das Anwendungsmodell der erstellten Applikation entspricht den definierten Vorgaben und ist als Input für die Transformation in ein weiterführendes Modell geeignet.
- Direkte Zugriffe auf das Dateisystem sind Browser-Applikationen nicht erlaubt. Eine Alternative Möglichkeit der Speicherung muss gefunden werden.

NFR2: Zuverlässigkeit

- Fehlerzustände werden von den Programmkomponenten verhindert. Treten dennoch Fehler auf, sind diese durch aussagekräftige Fehlermeldung anzuzeigen.
- Die Visualisierungskomponenten weisen eine hohe Fehlertoleranz und Robustheit auf.

NFR3: Benutzbarkeit

- Die Visualisierungskomponenten arbeitet im Hintergrund und benötigen möglichst wenig Eingriffe des Nutzers.

7.1.5 Nebenbedingungen

Zusätzlich zu den nichtfunktionalen Anforderungen werden die folgenden Nebenbedingungen formuliert. Diese Vorgaben sollen während der Implementierung der Konzepte zusätzlich beachtet werden:

NB1: Effizienz

- Als Teil einer Browser-Applikation, soll die Visualisierungskomponenten System-Ressourcen so wenig wie möglich belasten.
- Vorgenommene Veränderungen am aktuellen Stand der Komposition müssen möglichst ohne bemerkbare Verzögerung im Modell umgesetzt werden.
- Beim Verwenden eines neuen Service, soll die Verzögerung bei der Visualisierung der Frontends möglichst gering sein.
- Beim Verwenden eines bereits genutzten Service, soll die Visualiserung der Frontends möglichst ohne Verzögerung erfolgen

NB2: Änderbarkeit

- Die Kommunikation zwischen den einzelnen Anwendungsschichten soll möglichst generisch erfolgen, um eine Austauschbarkeit der einzelnen Schichten zu ermöglichen.
- Die Erweiterbarkeit der Annotationsbehandlung soll sichergestellt werden.
- Das Hinzufügen von Regeln für eigene Plattformdefinitionen soll möglich sein.
- Die Visualisierungskomponenten sollen so einfach wie möglich wartbar, veränderbar und erweiterbar sein.

NB3: Sicherheit

- Unberechtigte Zugriffe sowohl versehentlich als auch vorsätzlich und die Möglichkeit Schadcode auf dem Client-Computer auszuführen dürfen nicht möglich sein.

7.2 ABLAUF DER VISUALISIERUNG

Bevor die einzelnen Komponenten der Visualisierungsengine vorgestellt werden, wird das Konzept für den Ablauf des Visualisierungsprozesses betrachtet. Der Ausgangspunkt der Visualisierung ist das Modell eines Service. Sobald ein Nutzer des Tools eine Service-Operation auf den Designbereich des Kompositionswerkzeugs zieht, wird zur Erstellung des Frontends das Modell des gesamten Service vom Service-Repository des Kompositionswerkzeugs angefordert (Abbildung 7.1 - 1). Das Service-Repository bezieht die notwendigen Informationen zur Erstellung des Modells über einen entfernten Webservice. Um den Kommunikationsaufwand zu begrenzen, werden alle angeforderten Service-Modelle lokal zwischengespeichert. Um Performance- und Speicherprobleme zu vermeiden, müssen Caching-Strategien für die Zwischenspeicherung eingesetzt werden. DIe Betrachtung von Caching-Mechanismen ist nicht Bestandteil dieser Arbeit. Anschließend wird das Service-Modell geparst und Modell-Kommandos generiert, die im späteren Verlauf das Anwendungsmodell beeinflussen (Abbildung 7.1 - 2). Sind Annotationen an einem Service-Element vorhanden, dann werden diese Annotationen (Abbildung 7.1 - 3) ausgewertet und deren Effekt in die Visualisierung einbezogen. Dazu werden während der Annotations-Auswertung Effekt-Objekte erstellt, die die Auswirkungen der Annotationen beschreiben. Für die erstellten Kommandos werden die Basis-Parameter festgelegt, die das zu erstellende Anwendungsmodell-Element oder die vorzunehmende Modell-Änderung näher beschreiben (Abbildung 7.1 - 5). Handelt es sich bei dem aktuell betrachteten Element um einen Parameter wird zur Festlegung des UI-Elements, durch das der Parameter in der resultierenden Anwendung repräsentiert wird, eine spezielle Bestimmung in Abhängigkeit von der Zielplattform vorgenommen (Abbildung 7.1 - 4). Zum Schluss werden die Modell-Kommandos als Kommando-Satz an die Anwendungsmodell-API übergeben (Abbildung 7.1 - 5) und von dieser ausgeführt. Die Modell-Kommandos bringen nun die durch sie repräsentierten Änderungen in das Anwendungsmodell ein. Die folgende Abbildung stellt den Ablauf des Visualisierungskonzepts grafisch dar.

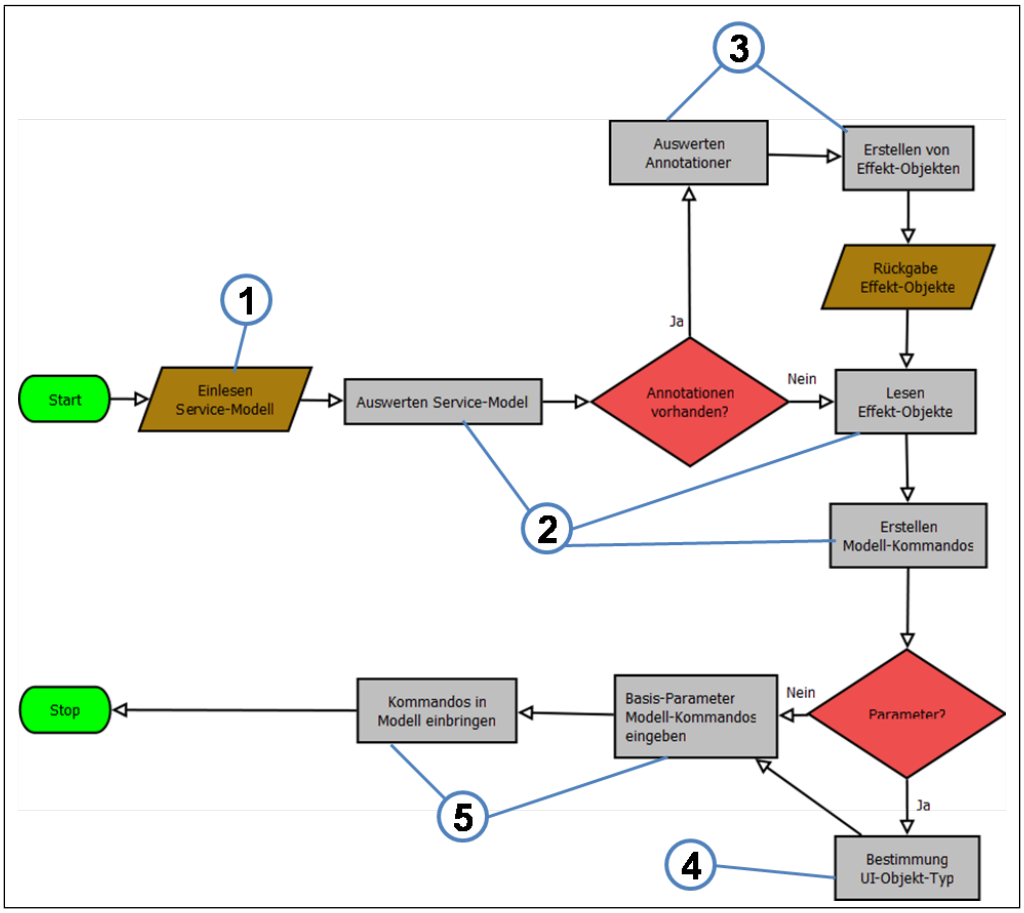


Abbildung 7.1: Ablauf des Visualisierungsprozesses

Im Folgenden werden die Bestandteile des Controllers vorgestellt, die für die Visualisierung der Frontends zuständig sind.

7.2.1 Komponenten der Visualisierungs-Engine

Die Visualisierungs-Engine ist innerhalb der Kompositionswerkzeugs für die Visualisierung der Service-Frontends verantwortlich. Sie unterteilt sich dabei selbst in 3 Komponenten, die jeweils einen Part des Visualisierungsprozesses umsetzen:

Visualisierungskernkomponente (in Abbildungen: VisualizationCoreComponent):
Die Visualisierungskernkomponente ist der Hauptbestandteil der Visualisierungsengine. Sie ist zuständig für die Basisanalyse des Service-Modells, die Erzeugung der repräsentativen Datenstruktur und das Einbringen der Datenstruktur in das Anwendungsmodell (Anforderungen: R2.1, R2.2, R2.5, R3.1). Außerdem übernimmt sie die Steuerung der weiteren Bestandteile der Visualisierungsengine. Als Input dient der Visualisierungskernkomponente das Service-Modell. Dessen Struktur und Bestandteile werden von der Komponente analysiert und eine entsprechende Datenstruktur zur Repräsentation der angeforderten Service-Operation erstellt. Diese Datenstruktur wird am Ende des Visualisierungsprozesses in das Anwendungsmodell integriert.

Annotations-Effekt-Bestimmung (AEB - in Abbildungen: AnnotationEffectDetermination):

Diese Komponente ist für die Auswertung der an Service-Elementen angehängten Annotationen zuständig (Anforderungen: R2.4). Die Annotationen können wie in Kapitel 6 beschrieben verschiedene Effekte auf die Visualisierung des Service haben. Die AEB-Komponente erstellt eine Beschreibung für diese Effekte. Die Beschreibung wird dem User-Interface zur Verfügung gestellt (Anforderung: R3.3) bzw. an die Kernkomponente übergeben. Die Kernkomponente nutzt diese Beschreibungen um die Annotationseffekte in die Datenstruktur der Service-Operation mit einfließen zu lassen.

Regelsatz (in Abbildungen: RuleSet):
Die Regelsatz-Komponente bestimmt passende UI-Elements zur Repräsentation der Service-Parameter (Anforderung: R2.5). Mit dem Kompositionswerkzeug sollen Anwendungen für verschiedene Plattformen modelliert werden. Aus diesem Grund müssen bei der UI-Element-Bestimmung neben den Datentypen der Parameter und den angehängten Annotationen auch Plattformspezifika beachtet werden. Die Regelsatz-Komponente wertet alle verfügbaren Faktoren aus und bestimmt das resultierende UI-Element und dessen Basis-Konfiguration.

Die folgende Abbildung stellt die drei Bestandteile der Visualisierungskomponente und deren Aufgabenverteilung grafisch dar.

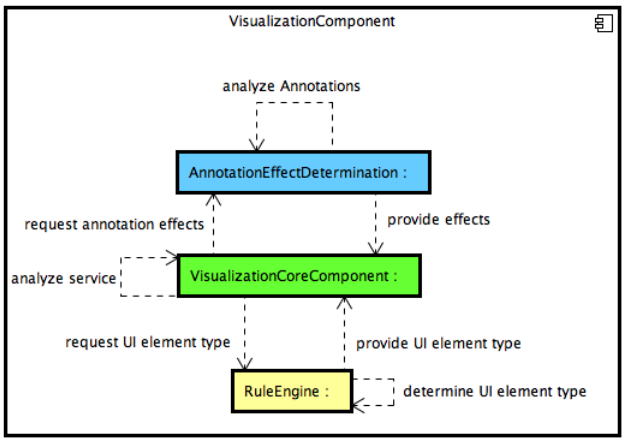


Abbildung 7.2: Bestandteile der Visualisierungskomponente

Ergänzt wird die Visualisierungs-Engine durch die folgenden weiteren Komponenten:

Update-Manager (in Abbildungen: CAMUpdateManager):
Der Update-Manager ist für die Verarbeitungen der Änderungen an der modellierten Anwendung zuständig (Anforderung: R3.1, R3.2). Jede Änderung, die durch den Nutzer des Kompositionstools ausgelöst wird, meldet das User-Interface über ein Änderungs-Kommando an den Update-Manager. Wenn es sich um das Einfügen einer neuen Service-Operation handelt, gibt der Update-Manager die Aufgabe an die Visualisierungskomponente weiter. Alle sonstigen Änderungen, wie verändern der Positin, umbenennen, löschen, etc. werden vom Update-Manager umgesetzt.

Service-Set (in Abbildungen: ServiceSet): Das Service-Set stellt eine Sammlung von lokal zwischengespeicherten Service-Modellen dar (Anforderung: R1.2). Sobald ein Nutzer die Operation eines Service auf die Arbeitsfläche zieht, wird von dem Service-Repository das Modell des kompletten Service der Operation angefordert (Anforderung: R1.1). Das Service-Modell wurde im Abschnitt 2.3 vorgestellt. Da die Wahrscheinlichkeit hoch ist, dass der Nutzer weitere Operationen des gleichen Service verwendet, wird das Modell in der Service Set -Komponente zwischengespeichert. Dadurch ist keine erneute Kommunikation mit dem entfernten Webservice notwendig,

wenn Operationen des zwischengespeicherten Service in die Anwendung integriert werden sollen. Neben der Zwischenspeicherung der Service-Modelle hat die Service-Set-Komponente noch die Aufgabe eine Liste aller im entfernten Repository verfügbaren Services inkl. deren Operationen vorzuhalten (Anforderung: R1.3).

Die Implementierung der Komponenten wird im Kapitel 8 beschrieben.

7.3 KOMMUNIKATION UND INFORMATIONSWEITERGABE DER PROGRAMMKOMPONENTEN

Das Kompositionswerkzeug ist nach dem Model-View-Controller Prinzip aufgebaut. Neben der Festlegung des Architekturmusters muss zusätzlich bestimmt werden, wie die Schichten untereinander im Detail miteinander kommunizieren und Änderungen weitergeben. Das wird auch durch die Anforderung R3.4 aus der Anforderungsanalyse gefordert. Eine Möglichkeit ist die direkten Kommunikation. Dabei erfolgt die Kommunikation über Methodenaufrufe innerhalb der anderen Komponente. Für die Kommunikation innerhalb des Kompositionswerkzeuges wurde sich für eine Kommunikation über Objekte entschieden. Die Objekte verstehen sich als Daten- und Befehls-Container, die alle Informationen zur Ausführung einer Operation enthalten. Diese Objekte werden zwischen den verschiedenen Komponenten eines Programms ausgetauscht. Der Sender übergibt die erstellten Objekte an die Zielkomponente, die die Objekte interpretiert und die gewüschten Aktionen ausführt. Zwischen den unterschiedlichen Schichten des Kompositonstools wird die Kommunikation wie folgt umgesetzt:

- View –> Controller: Änderungs-Objekte (In Abbildungen: ChangeCommand)
- Controller –> Modell: Modell-Kommando-Objekte (In Abbildungen: CAMCommand)
- Model –> View: Event-Objekte

Eins der Hauptargumente für die Nutzung von Objekten zur Kommunikation zwischen den einzelnen Programmkomponenten ist die Unabhängigkeit der Komponenten voneinander. Die einzelnen Komponenten müssen nicht aufeinander angepasst werden, sondern müssen lediglich die Kommunikations-Objekte interpretieren können. Wird eine Komponente ausgetauscht, ist es nicht notwendig die anderen Komponenten ebenfalls zu ändern. Es muss nur sichergestellt werden, dass die veränderte Komponente die Objekte interpretieren kann. Neben der leichteren Austauschbarkeit der Komponenten bietet die Kommunikation über Objekte aber auch noch einige weitere Vorteile:

Undo-Funktionalität:
Neben der Logik zur Änderungseinbringung in das Anwendungsmodell, ist es möglich in die Kommandos auch entsprechende Kompensationsoperationen zu implementieren. Diese Operationen erlauben es, die Änderungen eines Kommandos wieder rückgängig zu machen und somit eine Undo-Funktionalität zur Verfügung zu stellen. Alle bereits ausgeführten Kommandosets können auf einem Stack (in der Abbildung 7.3 "Executed Objekt Stack") abgelegt werden, um später auch die Änderungen von mehreren Kommandosets zu widerrufen. Eine Undo-Funktionalität über mehrere Änderungen hinweg ist bei einer direkten Kommunikation deutlich schwieriger umzusetzen. Die Undo-Funktionalität für die letzte Änderung kann relativ einfach umgesetzt werden, indem eine Backup-Version des Modells gespeichert wird. Bei Bedarf kann dann die aktuelle Version gegen die Backup Version getauscht werden. Generell kann man natürlich für jede Änderung eine eigene Modell-Version speichern. Allerdings nimmt der Speicherplatz bei sehr komplexen Modellen relativ stark zu. Eine weitere Möglichkeit ist das exakte Loggen der durchgeführten Änderungen. Mithilfe des Logs können dann die Änderungen am Modell schrittweise revidiert werden. Für einen Logging Mechanismus mit einer Möglichkeit Änderungen rückgängig zu machen, ist allerdings eine deutlich umfangreichere Implementierung notwendig, als dies bei der Undo-Funktionalität der Objekt-Kommunikation der Fall ist.

Atomarität:
Zur Ausführung einer Tool-Operation oder Änderung im Kompositionswerkzeug wird im Regelfall genau ein Kommandoset erstellt. Die Kommandosets können deshalb als eine Art atomare d.h. unteilbare Einheit angesehen werden. Atomare Einheiten werden nach dem „Alles oder Nichts" -Prinzip ausgeführt. Ist die Ausführung eines einzelnen Kommandos nicht erfolgreich, dann wird das gesamte Kommandoset nicht ausgeführt. Dieses Konzept ist ein Grundprinzip im Bereich der Datenbankentransaktionen und soll die Integrität des Datenbestandes sicherstellen. Dieses Konzept lässt sich auch innerhalb des Kompositionswerkzeugs verwenden. Kann ein Kommandoset und dementsprechend eine Änderung nicht vollständig ausgeführt werden, dann sollte der Stand des Anwendungsmodells dem Stand vor der Ausführung des Kommando-Sets entsprechen. Eine nicht vollständig ausgeführte Modelländerung kann zu einem fehlerhaften Modell und damit zu Problemen beim Kompositionsvorgang führen. Mit dem Vorhandensein einer Undo-Funktionalität für jedes einzelne Kommando ist eine Umsetzung der Atomarität eines Kommando-Sets möglich. Auch bei einer direkten Kommunikation kann eine Atomarität der Kommando-Sets umgesetzt werden. Allerdings sind hierfür komplexere Mechanismen und umfangreiches Logging notwendig. Das Vorhalten einer Backup-Kopie des Models ist eine Umsetzungsmöglichkeit. Der Nachteil ist aber auch in diesem Fall der gesteigerte Speicherbedarf bei komplexen Modellen.

Unabhängigkeit:
Die Verwendung von Objekten zur Kommunikation zwischen zwei Schichten erhöht die Unabhängigkeit der einzelnen Schichten voneinander. Werden beispielsweise innerhalb der Controller-Schicht direkt Methoden für die Veränderung des Modells implementiert, dann kann der Controller nur für dieses eine zugrundeliegende Modell verwendet werden. Denn der Controller ist in diesem Fall speziell auf dieses Modell angepasst. Ändert sich das Modell, dann muss auch der Controller verändert werden. Verwendet man im Gegensatz dazu Objekte zur Kommunikation zwischen den Schichten, dann ist die jeweilige Schicht nur auf die Erstellung der möglichst generischen Objekte angepasst. So lange die jeweilige Ziel-Schicht die Objekte lesen und umsetzen kann, kann diese auch beliebig ausgetauscht werden. Ändert sich beispielsweise das Anwendungsmodell, dann muss nur sichergestellt werden, dass die neue Modell-Schicht die Objekte des Controllers lesen und interpretieren kann.

Kappselung logischer Einheiten:
Ein weiterer Vorteil der Verwendung von Kommando-Objekten ist die Kappselung von logisch zusammengehörenden Einheiten. Innerhalb der Objekte können alle Informationen, die zur Umsetzung einer Operation notwendig sind, gespeichert werden. Dadurch werden logische Einheiten zur Ausführung von Updates und Änderungen gebildet. Im Allgemeinen kann dadurch eine leichtere Lesbarkeit und eine bessere Verständlichkeit der Implementierung erreicht werden. Bei der direkten Kommunikation erfolgt eine Kapselung meist durch die Unterteilung in mehrere für eine Aufgabe angepasste Methoden.

In der Abbildung 7.3 ist am Beispiel der Weitergabe von Änderungen die Kommunikation per direktem Methoden-Aufruf (in Abbildung: Direct Change Propagation) und Objekten (in Abbildung: Object Change Propagation) grafisch dargestellt.

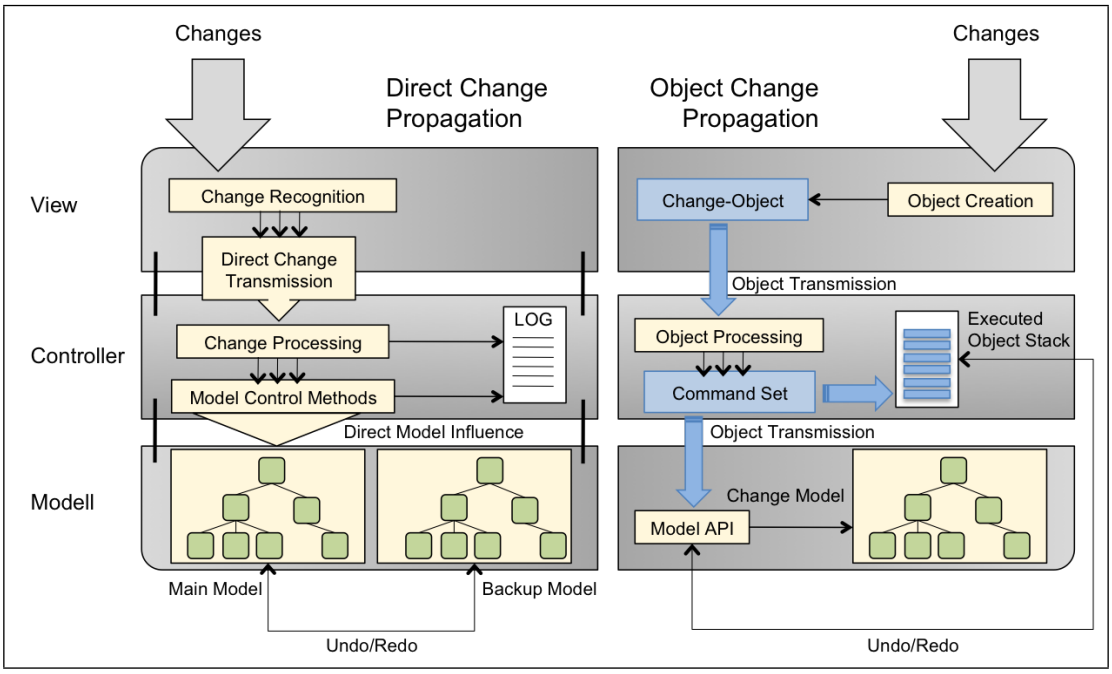


Abbildung 7.3: Zwei Möglichkeiten der Änderungsumsetzung

Beide Möglichkeiten bieten Vor- und Nachteile. Die Tabelle 7.1 stellt die beiden Möglichkeiten der Kommunikation gegenüber.

	Direkte Kommunikation	**Objekt-Kommunikation**
Kommunikation	Angepasste Methoden	Objekte
Implementierungs-aufwand	Mittel	Hoch
Anzahl der Klasse	Wenige, umfang-reiche Klassen	Viele, kleine Klassen
Undo-Funktionalität	Log/Modell-Kopie	Stack ausgeführ-ter Operationen
Atomarität	Umsetzbar	einfach umsetzbar
Unabhängigkeit	Speziell angepasst	hohe Unabhängigkeit
Kapselung	Kappselung in Methoden	Kappselung in Objekten

Tabelle 7.1: Evaluation der vorgestellten Konzepte hinsichtlich der aufgestellten Anforderungen

Für die Implementierung im Kompositionswerkzeug überwiegen die Vorteile der Kommunikation über Objekte, weshalb die Entscheidung zugunsten dieses Konzepts ausfiel. Die einfache Möglichkeit der Umsetzung einer Undo-Funktionalität und die Unabhängigkeit der einzelnen Komponenten waren ausschlaggebende Argumente. Im Rahmen eines Forschungsprojektes sind viele Änderungen an einzelnen Komponenten und dem zugrundeliegenden Modell während des Entwicklungsprozesses zu erwarten. Änderungen an einer Komponente sollten daher möglichst wenig Auswirkungen auf andere Programmbestandteile haben. Das kann durch die Nutzung der Objekt-Kommunikation ermöglicht werden.

7.4 KONZEPTION DER ANNOTATIONSAUSWERTUNG

Eine der wichtigsten Anforderungen an die Annotationsauswertung ist deren möglichst einfache Erweiterbarkeit. Da sich der Umfang der verfügbaren Annotationen jederzeit verändern kann, sollen neue Annotationen möglichst einfach in das Kompositionstool integriert werden können. Das

heißt, dass die Implementierung und Auswertung der Annotationen möglichst generisch erfolgen muss. Eine einfache Möglichkeit der generischen Implementierung wäre die Nutzung eines generischen Modells zur Beschreibung der Annotationen. Ein generisches, d.h. allgemeingültiges Modelle würde die Möglichkeiten bieten, alle vorhandenen Annotationen durch ein einziges Modell zu beschreiben. Die Zugriffe auf Definitionen und Parameter wären in dem Modell für alle Annotationen gleich definiert. Eine Auswertung der Annotationen auf Basis dieses Modells könnte alle Annotationen behandeln, die dem generischen Modell gehorchen. Ein weiterer Vorteil eines generischen Modells ist die Möglichkeit der Code-Generierung. Die Erstellung eines Editors zur Definition neuer Annotationen wäre automatisiert möglich.

Allerdings sieht das in Abschnitt 2.3.3 vorgestellte Annotations-Modell eine spezifische Definition für jede Annotation vor. Eine generische Auswertung über dieses Modell ist deshalb nicht möglich. Die folgende Ausgangssituation ist also gegeben:

1. Das Annotations-Modell sieht keine generische Definition der Annotationen vor.
2. Jede Annotation spezifiziert ihre eigene Struktur. Der Zugriff auf Parameter und Referenzen ist für jede Annotation unterschiedlich definiert.
3. Das Hinzufügen neuer Annotationen bedeutet die Erweiterung des Annotations-Modells um mindestens eine neue, spezifische Klasse
4. Das Hinzufügen neuer Annotationen soll möglichst einfach sein, d.h. ohne die Notwendigkeit Code in verschiedenen Komponenten zu implementieren.
5. Die Auswertung der Annotationen soll möglichst generisch und unabhängig von der restlichen Implementierung erfolgen.
6. Die Auswirkungen funktionaler Annotationen muss der UI zur Verfügung gestellt werden.
7. Die Auswirkungen von UI- und hybriden Annotationen müssen in das Anwendungsmodell integriert werden.

Durch diese Faktenlage ist eine separate Implementierung in einer eigenen Komponente notwendig. Dies soll durch die Annotations-Effekt-Bestimmungs (AEB) -Komponente realisiert werden.

7.4.1 Funktionsweise der Annotations-Effekt-Bestimmung

Die Annotations-Effekt-Bestimmungs (AEB) -Komponente ist Teil der Visualisierungsengine und wird von der Visualisierungskernkomponente verwaltet. Werden während der Analyse des Service-Modells Annotationen an einem Service-Element gefunden, dann ruft die Kernkomponente die AEB-Komponente auf und übergibt das annotierte Service-Element. Jede Annotation des Service-Elements muss nun entsprechend ihrer Spezifika analysiert und deren Effekt bestimmt werden.

Für die funktionalen Annotationen müssen dem User-Interface Informationen zur Verfügung gestellt werden, um die in Abschnitt 6.2.2 definierte metaphorische Anzeige der Annotationen zu ermöglichen. Die folgenden Informationen sind für eine korrekte Darstellung notwendig:

- Name der Annotation
- Enthaltende Service-Operation
- Betroffene Interaktionselemente
- Durch die Annotation spezifizierte Anzeigegruppen (bswp. Trigger-Elemente)

- Pfad zum Annotations-Icon
- Farbe der Markierung für die Anzeigegruppen

Bereitgestellt werden die Informationen als Datenstruktur, die vom User-Interface gelesen werden kann. Die eigentliche Auswertung der funktionalen Annotationen und die Bereitstellung der entsprechenden Funktionalität übernimmt die Runtime-Generierung.

Die Hybrid- und UI-Annotationen beeinflussen direkt das Anwendungsmodell und dessen Elemente. Deshalb müssen die Effekte der Annotationen in die Erstellung der repräsentativen Datenstruktur des Service-Frontends einfließen. Die angeforderten Effekt-Beschreibungen der Annotationen eines Service-Elements werden als Effekt-Objekte an die Visualisierungskernkomponente übergeben. In den Effekt-Objekten werden die Auswirkungen der Annotationen generisch beschrieben. Die Visualisierungskernkomponente muss nur auf die Auswertung der Objekte ausgelegt sein. Auf die verschiedenen Annotationen muss nicht direkt zugegriffen werden. In den Objekten sind die folgenden Informationen enthalten:

- Art der Änderung: Hinzufügen neuer Elemente, Verändern des Anwendungsmodells
- Beschreibung des neuen Elements: Art des Elements, Element-Konfiguration, Element-Container etc.
- Beschreibung der Modell-Änderungen: Darstellungstyp, Annotationsaktivierung, Elementsortierung, etc.
- IDs der betroffenen Elemente

Die Effekt-Objekte entkoppeln die spezifische Auswirkungsbestimmung der einzelnen Annotationen von der Erstellung der eigentlichen Datenstruktur. Die Effekt-Objekte werden von der Visualisierungskernkomponente ausgewertet und von der spezifischen Definition der Annotationen unabhängig in die Generierung der Datenstruktur einbezogen werden. Die Kernkomponente muss daher nicht an die Beschaffenheiten der einzelnen Annotationen angepasst werden.

7.4.2 Erweiterbarkeit der Annotationsauswertung

Die Effekt-Objekte sorgen dafür, dass die Realisierung der Annotations-Auswirkungen in der Kernkomponente unabhängig von den Spezifika einzelner Annotationen erfolgen kann. Eine einfache Erweiterbarkeit der Annotationsauswertung muss ebenfalls sichergestellt werden. Neue Annotationen sollen unabhängig von der Annotationsart problemlos zur Auswertung hinzugefügt werden können.

Das Konzept für die Erweiterbarkeit der Annotationsauswertung sieht eine Auswertung der Annotationen durch Beschreibungsklassen vor. Dabei wird für jede Annotation eine eigene Effekt-Beschreibung als Klasse implementiert. Diese Klasse ist speziell auf die jeweilige Annotation d.h. deren Struktur, Funtionsweise, Parameterzugriffe und Referenzen angepasst. Zusätzlich enthält die Beschreibungsklasse eine annotationsspezifische Logik zur Auswertung der Annotation und Erzeugung der Effekt-Objekte. Die Beschreibungsklassen aller Annotationen werden in einem zentralen Repository verwaltet. Wird ein Service-Element mit angehängten Annotationen übergeben, dann wird für jede Annotation deren zuständige Beschreibungsklasse aufgerufen. Die Beschreibungsklasse wertet die Annotation aus und erstellt für Hybrid-/UI-Annotationen ein entsprechendes Effekt-Objekt. In dem Effekt-Objekt werden die IDs aller betroffenen Elemente und die notwendigen Informationen hinterlegt. Anschließend wird das Effekt-Objekt an die Visualisierungskernkomponente zur weiteren Verarbeitung übergeben. Für funktionale Annotationen werden die zur metaphorischen Darstellung notwendigen Informationen der UI zur Verfügung gestellt. Der Ablauf der Annotationsauswertung ist in der folgenden Abbildung illustriert:

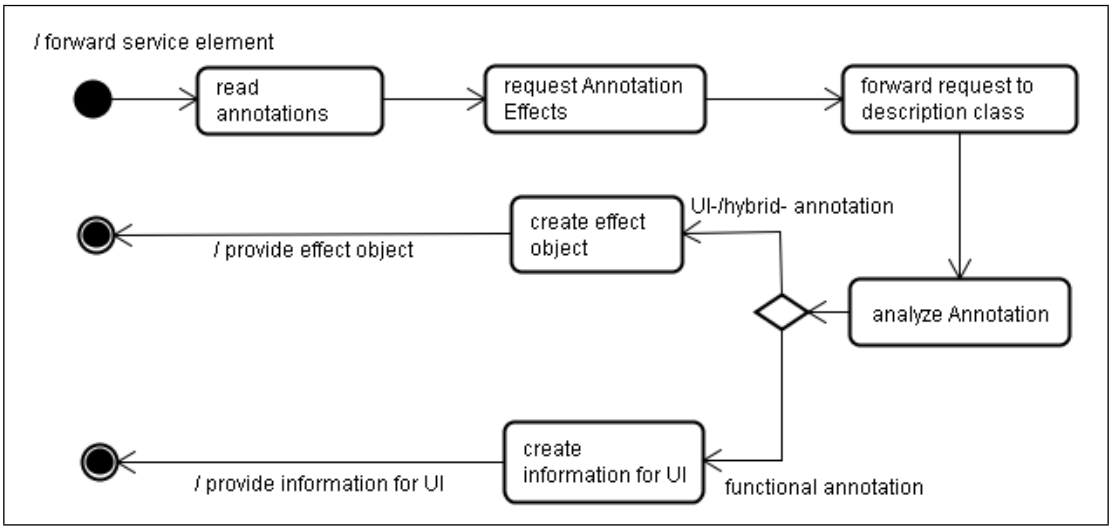


Abbildung 7.4: Ablauf der Annotationsauswertung

Da die einzelnen Beschreibungsklassen auf die zugehörige Annotation angepasst sind, kann direkt die Struktur der Annotation umgesetzt werden. Der Zugriff auf die spezifisch definierten Parameter und Referenzen ist unmittelbar möglich. Im Zusammenhang mit dem generischen Effekt-Objekt wird durch dieses Konzept versucht die spezifische Definition der einzelnen Annotationen auszugleichen. Die eigentliche Erstellung der Datenstruktur bleibt unabhängig von den verschiedenen Gegebenheiten der Annotationen. Die Notwendigkeit der Implementierung wird durch die Nutzung eines Repositories auf eine zentrale Stelle beschränkt. Ändern sich die Spezifika einer Annotation, dann muss nur die Beschreibungsklasse angepasst werden. Das Effekt-Objekt und die Visualisierungskernkomponente bleiben von der Änderung unberührt. Allerdings sind durch die Notwendigkeit der Klassen-Implementierung Programmierkenntnisse unumgänglich.

Ein alternatives Konzept zum zentralen Beschreibungsklassen-Repository sieht eine Beschreibung der Annotationen per XML vor. Für jede Annotation wird eine XML-Datei erstellt, die detaillierte Informationen über die Auswirkungen der Annotation enthält. Beschrieben werden:

- Ort der Auswirkung (UI, Anwendungsmodell)
- Die Art der Auswirkung (Hinzufügen eines Elements, Ändern eines Elements, ...)
- Bei neuen Objekten eine detaillierte Spezifikation des neuen Objekts
- Bei Elementänderungen eine detaillierte Spezifikation der Änderung
- Auswirkungen auf andere Elemente

Die XML-Beschreibungen bieten ein höheres Maß an Generik als die Beschreibungsklassen. Außerdem sind für die Beschreibung von Auswirkungen keine Programmierkenntnisse notwendig. Die Erstellung der XML-Dateien kann beispielsweise über einen automatisch generierten Editor funktionieren. Mit Hilfe dieses Editors könnten Annotationen sogar ohne XML-Kenntnisse hinzugefügt werden. Eine detaillierte Beschreibung dieses Alternativ-Konzepts ist im Anhang A.4.0.1 zu finden.

Eine notwendige Voraussetzung für die Nutzung von XML-Beschreibungen ist eine Programmiersprache mit Unterstützung für dynamisches casten. Damit erfolgt nach Einlesen der Beschreibungen das Casting der abstrakten Annotationen auf ihren konkreten Typen. Das Konzept der Beschreibungsklassen ist nicht von Casting-Spezifika der Programmiersprachen abhängig. In der Tabelle 7.2 werden beide zuvor vorgestellten Konzepte gegenüber gestellt.

	Auswirkungsbeschreibung per XML	Annotations-Beschreibungs-Repository
Beschreibungsformat	XML-Beschreibung	Beschreibungsklassen
Rückgabe	Effekt-Objekte	Effekt-Objekte
Verwaltung	XML-Description-Collection	Description-Repository
Einheitliches Schema	XSD-Schema	kein Schema
Code für Erweiterungen	nur XML	implementieren einer Klasse
Generierter Editor	Eclipse EMF	nicht direkt möglich
Parser	XML-Parser + Linq	nicht notwendig
C# Implementierung	Casting Problem	unproblematisch
Performance	Einbuße durch Reflection API	keine Einbuße
Erweiterbarkeit	einfach	bedingt einfach
Aufwand Umsetzung	mittel	hoch

Tabelle 7.2: Vergleich der Erweiterbarkeitskonzepte für die Annotationsauswertung

Das Konzept des Beschreibungsklassen-Repositorys versucht das Hinzufügen neuer Annotationen einfach zu halten, ohne die Casting-Abhängigkeiten des XML-Konzepts aufzuweisen. Die Erweiterung der Annotationen ist im Gegensatz zum XML-Konzept nicht ohne Programmiererfahrung und Kenntnis der Implementierung des Kompositionstools möglich. Allerdings ist durch die Kapselung der Annotationsauswertung an einer zentralen Stelle der Erweiterungsaufwand dennoch überschaubar. Implementierungstechnisch kommt das Beschreibungsklassen-Repository ohne die Nutzung der Reflection API aus. Dadurch sind keine Performanz- und Debug-Probleme zu befürchten. Mangels generischem Annotations-Modell und aus den zuvor genannten Gründen wurde sich für die Implementierung innerhalb des Kompositionswerkzeugs für das Beschreibungsklassen-Repository entschieden.

7.5 DYNAMISCHE PLATTFORMUNTERSTÜTZUNG

Ein Kernziel des ServFace-Projekts ist die Möglichkeit der Erstellung interaktiver Anwendung für unterschiedliche Plattformen. Sollen mehrere Plattformen unterstützt werden, müssen deren individuellen Merkmale betrachtet werden, um die Auswirkungen auf die Visualisierung der Frontends und den Kompositionsprozess zu identifizieren. Beispielsweise unterscheidet sich für jede Plattformen die Menge der unterstützten UI-Elemente. Dementsprechend unterschiedlich ist das Erscheinungsbild der visualisierten Service-Frontends. Innerhalb dieses Abschnitts werden die unterschiedlichen Merkmale diskutiert und deren Bedeutung für die Visualisierung der Frontends erläutert. Anschließend wird die Regelset-Komponente beschrieben, die für die plattformabhängige Bestimmung der UI-Elemente zur Präsentation von Service-Parametern zuständig ist. Zusätzlich wird ein Konzept vorgestellt, dass eine Anpassung der Anzeige des Kompositionstools entsprechend der Spezifika einer Plattform ermöglicht. Dem Nutzer soll damit schon zur Design-Time ein Eindruck vermittelt werden, wie die erstellte Anwendung auf der gewünschten Zielplattform dargestellt wird.

7.5.1 Merkmalidentifikation

Die folgende Auflistung präsentiert die für die Visualisierung wichtigsten Eigenschaften einer Zielplattform. Zusätzlich werden mögliche Auswirkungen auf die Darstellung aufgeführt.

Bildschirmauflösung:
Die Bildschirmauflösung definiert den zur Verfügung stehenden sichtbaren Platz einer Anwendung. Besonders mobile Plattformen bieten eine deutlich geringere Auflösung als Desktop-Plattformen. Für die zu erstellende Anwendung kann die Verkleinerung der Anzeigefläche die folgenden Auswirkungen haben:

- Verkleinerung der gesamten Anwendung, um das verringerte Platzangebot zu kompensieren: Besonders auf mobilen Endgeräten ist eine starke Verkleinerung notwendig, was zur Unlesbarkeit der Elemente führen kann.
- Virtuelle Vergrößerung der Anzeigefläche durch Scrolling und Panning: In diesem Fall behält die Anwendung ihre ursprüngliche Größe und erweitert die Anzeigefläche durch die Möglichkeit horizontal und vertikal zu scrollen. Dadurch kann sich aber die Nutzbarkeit der Anwendung verringern. Besonders vertikales Scrollen gilt als sehr umständlich[Nie05] [Nie02].
- Verteilung der Anwendung auf mehrere Seiten: Die Anwendung wird in mehrere Abschnitte unterteilt und auf mehrere Seiten platziert. Das ist allerdings nicht bei allen Anwendungen und nicht auf allen Plattformen möglich

Alle drei beschriebenen Möglichkeiten können auch kombiniert verwendet werden.

Plattform-Toolkit:
Für jede Plattform wird ein Toolkit spezifiziert, dass Schnittstellen zur Programmierung von Anwendungen auf der jeweiligen Plattform zur Verfügung stellt. Ein Toolkit legt dabei den Rahmen und die Möglichkeiten der ausführbaren Anwendungen fest. Unter anderem unterscheiden sich die Toolkits in der Menge und den Eigenschaften der unterstützten UI-Elemente. Um eine Anwendung auf eine andere Plattform zu portieren ist also die Umstellung auf die Toolkitspezifika der Zielplattform notwendig. Das bedeutet unter anderem auch die Umstellung auf die UI-Elemente des entsprechenden Toolkits.

UI-Elemente:
Unterschiedliche Plattformen unterstützen eine unterschiedliche Menge an UI-Elementen. Zusätzlich unterscheiden sich die UI-Elemente in Aussehen, Ausmaßen, Möglichkeiten und Konfiguration. Beispielsweise wird eine Tabelle im iPhone SDK als tableView und in Silverlight als DataGrid bezeichnet. Neben der unterschiedlichen Bezeichnung ist auch die Darstellung und die Konfiguration der Elemente verschieden. Soll eine Anwendung nun von einer Plattform auf eine andere Plattform portiert werden, dann ist die Anpassung der UI-Elemente an die Eigenschaften der neuen Plattform notwendig. Unter Umständen hat dies aber zur Folge, dass einige Elemente nicht angezeigt werden können, wenn diese auf der Zielplattform nicht zur Verfügung stehen. Außerdem können die Frontends auf verschiedenen Plattformen unterschiedlich aussehen, da sich die Eigenschaften der UI-Elemente unterscheiden.

Interaktionsart:
Einige Plattformen unterscheiden sich auch in den Interaktionsmöglichkeiten mit der Plattform. Während bei den meisten Desktop-Computern mit Maus und Tastatur gearbeitet wird, nutzt das Apple iPhone eine Interaktion per Multitouch. Die unterschiedlichen Interaktionsarten haben Einfluss auf den Nutzungskomfort der verschiedenen UI-Elemente. Beispielsweise ist die Eingabe von Daten auf einer Software-Tastatur deutlich schwieriger als auf einer Hardware-Tastatur. Wird der Nutzer aufgefordert sehr umfangreiche Daten einzugeben, dann sollte als Interaktionsmöglichkeit in diesem Fall eine Hardware-Tastatur zur Verfügung stehen.

Unterstützung mehrseitiger Anwendungen:
Anwendungen lassen sich oftmals in mehrere Abschnitte unterteilen, die auf mehrere Seiten verteilt werden können. Nicht alle Plattformen unterstützen allerdings mehrseitige Anwendungen.

Unterstützung von Scrolling:
Scrolling wird zur virtuellen Vergrößerung der Arbeitsfläche verwendet. Zu umfangreiches Scrolling führt aber zu einer Verschlechterung der Nutzerbarkeit einer Anwendung.

Kommunikationsmöglichkeiten:
Verschiedene Endgeräte bieten unterschiedliche Möglichkeiten des Internetzugangs an. Je höher die Datenrate eines Gerätes ist, desto schneller sind Daten auf dem Gerät verfügbar. Auf die Darstellung des User-Interfaces kann die Datenrate ebenfalls Auswirkungen haben. Bei Geräten mit geringerer Datenraten können ggf. UI-Elemente mit geringerer Qualität und Größe verwendet werden, um die Darstellungszeiten zu verringern und die Reaktionsfähigkeit zu erhöhen.

Unterstützung unterschiedlicher Medien-Typen
Die Unterstützung für verschiedene Multimedia-Formate wie Videos, Musik oder Flash unterscheiden sich je nach Zielplattform. Das entsprechende Toolkit definiert welche Medientypen unterstützt werden. Beispielsweise kann Adobe Flash aktuell nur auf wenigen mobilen Plattformen ausgeführt werden, ist aber auf den meisten Desktop-Computern vorhanden.

Die Anwendung muss also an die Gegebenheiten der Zielplattform angepasst werden. Zur Verdeutlichung der Merkmale vergleicht die Tabelle 7.3 zwei Beispielplattformen miteinander.

	Web-Anwendung	**iPhone**
Bildschirmauflösung	unterschiedlich	480x320 Pixel
Kommunikation	verschiedene	UMTS/HSDPA
Toolkit	Silverlight	iPhone SDK, Cocoa
UI-Elemente	Silverlight-UI-Elemente	iPhone-UI-Elemente
Interaktion	Maus, Tastatur	Multitouch
Mehrseitige Anwendung	ja	ja
Scrolling	ja	ja
Dateneingabe	Tastatur	Software-Tastatur
Multimediatypen	viele, entsprechend Silverlight	mittel, entsprechend iPhone SDK

Tabelle 7.3: Vergleich unterschiedlicher Plattformen nach den identifizierten Plattform-Merkmalen

7.5.2 Bestimmung der UI-Elemente für Parameter in Abhängigkeit von der Zielplattform

Die Wahl der Zielplattform beeinflusst einige darstellungsrelevante Faktoren. Besonders die Darstellung von Parametern ist von der gewählten Zielplattform abhängig, da jedes Toolkit einen eigenen Satz an UI-Elementen definiert. Die UI-Elemente der verschiedenen Plattformen unterscheiden sich in vielen Punkten. Für die Visualisierung relevant sind unter anderem:

- Benennung der UI-Elemente
- Optische Darstellung
- Ausmaße
- Interne Konfiguration

Für die Darstellung der Parameter ist deshalb besonders wichtig, dass in die Bestimmung des passenden UI-Elements neben dem Datentyp und Annotationen auch die Gegebenheiten der gewählten Plattform einfließen. Für die Bestimmung der passenden UI-Elemente unter Beachtung der genannten Faktoren ist die Regelset-Komponente verantwortlich. Für jede unterstützte Plattform wird ein Regelset definiert. Die Regelsets enthalten allgemeinen Plattformspezifikationen und Regeldefinitionen für jeden unterstützten Datentypen. Die Regel-Engine nutzt diese Regeldefinitionen um zu entscheiden welches UI-Element für einen Parameter gewählt werden soll. Die Datentyp-Regeln definieren dabei:

- Zuordnung von Datentypen zu passenden UI-Elementen.
- Konfigurationen der UI-Elemente.
- Bedingungen für die Wahl des UI-Elements.
- Regeln zur Behandlung und Speicherung von Datentyp-Attributen.

Nach erfolgter Festlegung des UI-Elements wird ein UI-Element-Objekt an die Visualisierungskernkomponente zurückgegeben. Das Objekt enthält die Bezeichnung und Konfigurationsparameter für das gewählte UI-Element. Die Konfigurationsparameter können verwendet werden um schon während der Design-Time die Anzeige der UI-Elemente an die Ziel-Plattform anzupassen. Den Ablauf der UI-Element-Bestimmgun stellt die Abbildung 7.5 dar.

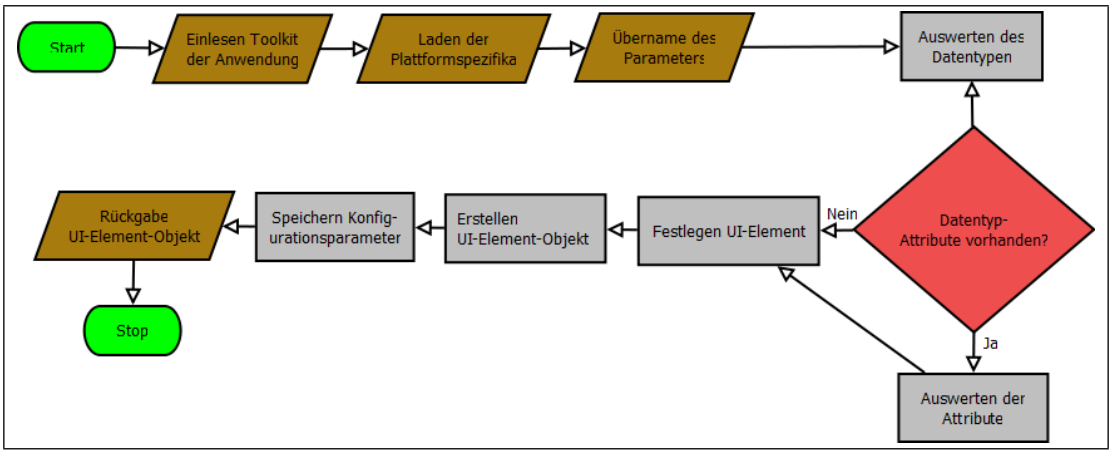


Abbildung 7.5: Flussdiagramm der Regelset-Komponente

Die Visualisierungskernkomponente nutzt das UI-Element-Objekt um einerseits das UI-Element für einen Parameter in das Anwendungsmodell einzubringen und andererseits die Konfigurationsparameter als Eigenschaften zu hinterlegen.

7.5.3 Plattformspezifische Veränderung der Toolumgebung

Um dem Nutzer des Kompositionstools schon während der Gestaltung der Anwendung einen korrekten Eindruck zu vermitteln, wie die resultierende Anwendung auf der Zielplattform aussehen wird, soll das User-Interface dynamisch an die Gegebenheiten der Zielplattform angepasst werden. Dazu werden in den Regelsets auch Plattformspezifikationen vorgehalten. Diese Spezifikationen enthalten Angaben über die unter Punkt 7.5.1 definierten Merkmale. Zusätzlich können Konfigurationen der auf einer Plattform verfügbaren UI-Elemente enthalten sein. Die Plattformspezifikationen liegen im XML-Format vor und werden durch ein XSD-Schema bestimmt, das in Anhang A.3 zu finden ist. Beim Start des Kompositionswerkzeugs kann der Nutzer die gewünschte Plattform für seine zu erstellende Anwendung auswählen. Entsprechend der Auswahl lädt die Regelsatz-Komponente die Daten der gewünschten Plattform aus der Plattformspezifikation aus. Die Daten werden dann in einer plattformspezifischen Klasse zwischengespeichert und dem User-Interface zur Verfügung gestellt. Das User-Interface kann nun diese Informationen nutzen, um die Anzeige der Elemente an die Plattformspezifikation anzupassen. Folgende Veränderungen am UI sind möglich:

- Größenanpassung der Arbeitsfläche um der Zielplattform zu entsprechen
- Nutzung der UI-Element-Palette der Zielplattform

- Darstellung der UI-Elemente entsprechend der Konfiguration auf der Zielplattform
- Automatische Aufteilung der Frontends auf mehrere Seiten um übermäßiges Scrollen zu vermeiden
- Emulation der Interaktionstechniken
- Auschließliches Anzeigen von unterstützten Multimedia-Typen

Der Ablauf innerhalb der Regelsatzkomponente ist in der Abbildung 7.6 grafisch dargestellt.

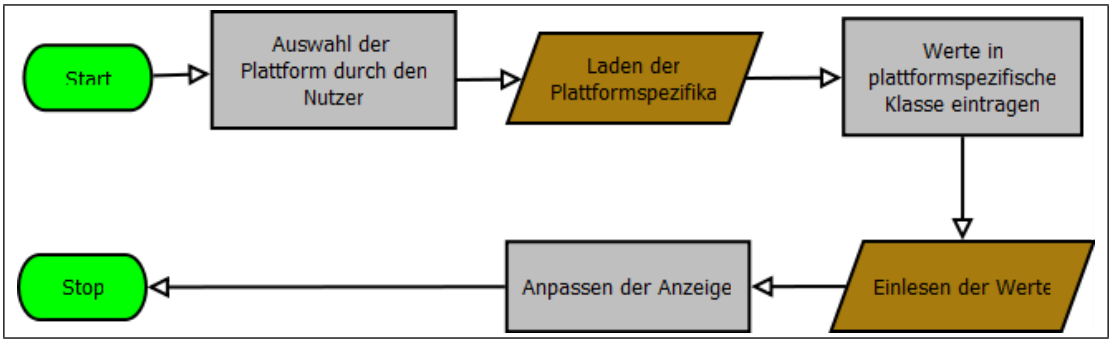


Abbildung 7.6: Flussdiagramm für die Anpassung des User-Interface

Die folgende Abbildung stellt 2 Frontends auf 2 verschiedenen Plattformen gegenüber. Die Darstellung erfolgt einmal für eine Webanwendung und einmal für das Apple iPhone.

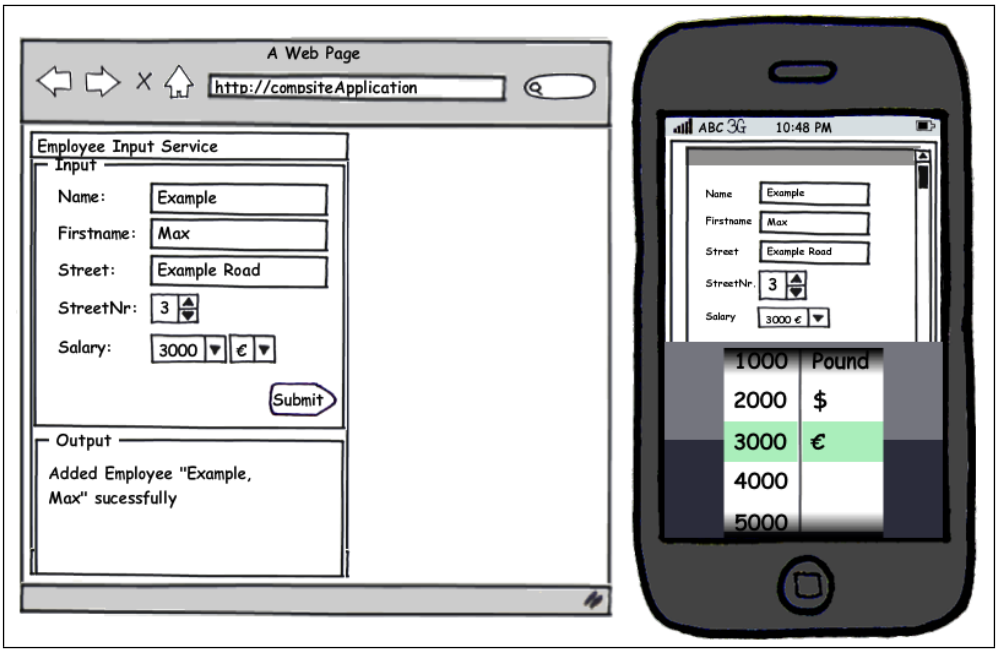


Abbildung 7.7: Mockups eines Service-Frontend für eine Webanwendung (links) und Apples iPhone (rechts)

7.6 ZUSAMMENFASSUNG

In diesem Kapitel wurde eine Engine konzipiert, die für Service-Operationen ein entsprechendes Service-Frontend visualisiert. Zu Beginn wurden funktionale und nicht-funktionale Anforderungen

an diese Engine fomuliert, die durch verschiedene Konzepte erfüllt werden sollen. Besonders wichtig war, dass eine einfache Erweiterbarkeit der Annotationsauswertung mit neuen Annotationen möglich ist und das Visualisierungsentscheidungen unter Beachtung von Plattformspezifika ausgeführt werden. Die Visualisierungsengine an sich besteht aus mehreren Komponenten, die von der Visualisierungskernkomponente verwaltet werden. Die Kernkomponente liest ein Service-Modell ein und erzeugt unter Zuhilfenahme der Annotations-Effekt-Bestimmungs (AEB) -Komponente und der Regelset-Komponente Modell-Kommandos. Diese Kommandos werden anschließend verwendet um am Anwendungsmodell alle notwendigen Änderungen vorzunehmen und um eine Service-Operation erfolgreich zu integrieren.
Für die AEB-Komponente wurden ein Konzept vorgestellt, dass die spezifische Definition der Annotationen durch generische Effek-Objekte und ein Beschreibungsklassen-Repository löst. Die Visualisierungskernkomponente bleibt dabei unabhängig von den Spezifika der einzelnen Annotationen.

Die gewünschte Unterstützung mehrerer Plattformen machte eine Betrachtung der spezifischen Merkmale einer Plattform und deren Auswirkung auf die Komposition notwendig. Diese Aufgabe übernimmt die Regelsatz-Komponente. Deshalb ist sie für die Bestimmung der UI-Elemente zur Repräsentation von Parametern zuständig. Für diese Bestimmung werden neben Datentypen und Annotationen auch Plattformspezifika beachtet, um die Unterstützung mehrerer Plattformen zu gewährleisten. Zusätzlich wurde ein Konzept vorgestellt mit dessen Hilfe auch das User-Interface die Plattform-Beschreibungen im XML-Format nutzen kann, um die Anzeige des Kompositionstools an die Gegebenheiten der ausgewählten Plattform anzupassen. Durch die Anpassung des User-Interface des Kompositionstools und der visualisierten Frontends an die Merkmale einer Plattform, bekommt der Nutzer des Werkzeugs schon zur Design-Time einen Eindruck über das Aussehen seiner Anwendung auf der gewählten Plattform. Insgesamt bietet die Visualiserungskomponente eine umfassende Lösung zur Visualisierung von Frontends. Im folgenden Kapitel wird die Implementierung der einzelnen Komponenten beschrieben.

8 IMPLEMENTIERUNG DER KOMPONENTEN DES KOMPOSITIONSWERKZEUGS

In dem vorliegenden Kapitel wird die Implementierung des Kompositionswerkzeugs erläutert. Zunächst wird die Gesamtarchitektur und die Kommunikation zwischen den einzelnen Schichten der Architektur beschrieben. Anschließend wird die Implementierung der wichtigsten Komponenten beleuchtet. Dabei liegt der Fokus besonders auf den im Konzeptionskapitel 7 vorgestellten Werkzeugbestandteilen der Visualisierungs-Engine.

8.1 GESAMTARCHITEKTUR DES KOMPOSITIONSWERKZEUGS

Das folgende Klassendiagramm visualisiert die vereinfachte Struktur des Kompositionswerkzeugs:

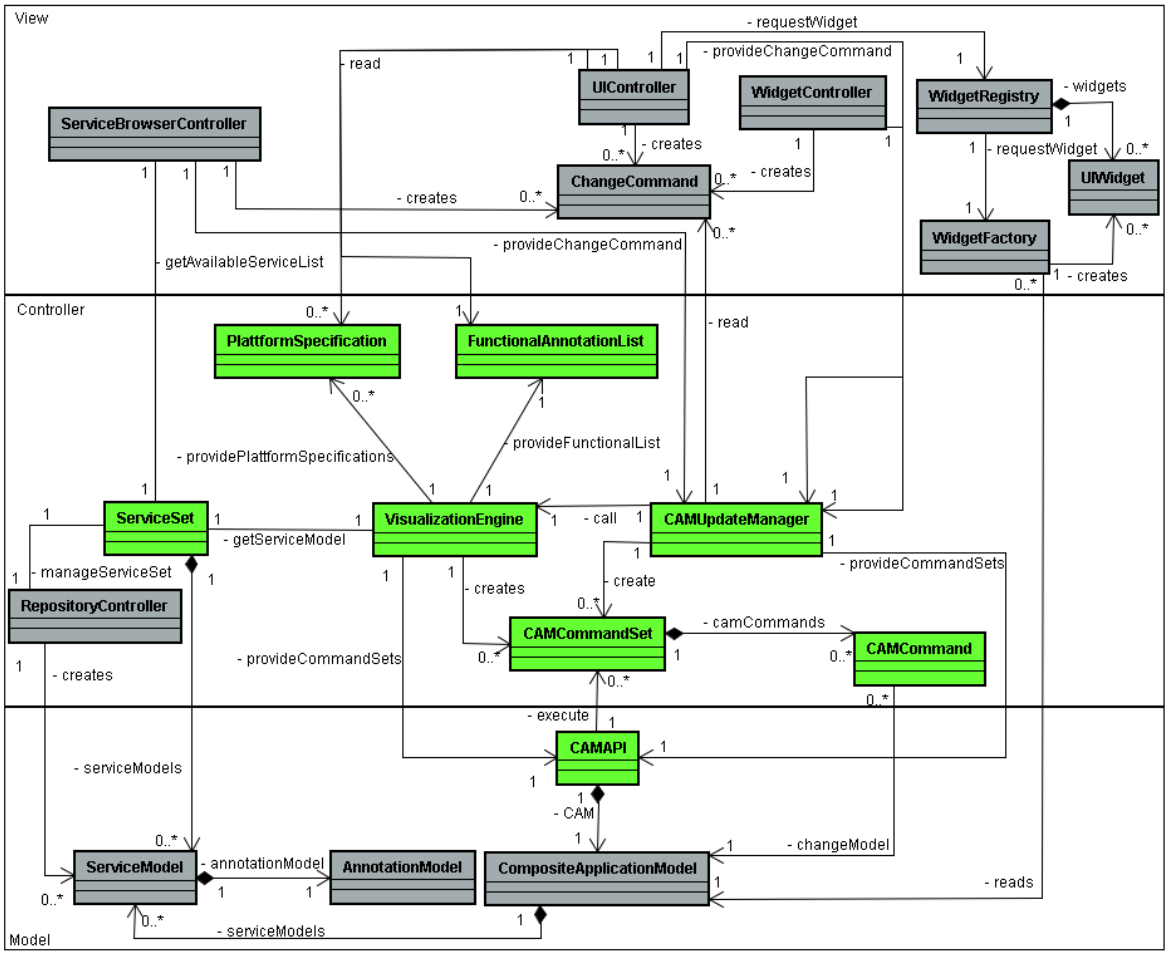


Abbildung 8.1: Klassendiagramm des Kompositionswerkzeugs

Die Gesamtarchitektur ist an das Model-View-Controller Architekturmuster angelehnt. Im Rahmen dieser Arbeit wurden die meisten Komponenten der Controller-Schicht implementiert, die in der Abbildung grün markiert sind. Die Controller-Schicht des Kompositionswerkzeugs ist für die Integration der Service-Frontends und die Umsetzung der Änderungen an der erstellten Anwendung zuständig. Außerdem stellt sie dem User-Interface wichtige Informationen für die Anzeige zur Verfügung. Kernbestandteil der Controller-Schicht ist die Visualisierungs-Engine (in Abbildung: VisualizationEngine), die durch den Update-Manager (in Abbildung: CAMUpdateManager) und die Service-Set-Komponente ergänzt wird. In den folgenden Abschnitten wird die Implementierung der wichtigsten Komponenten des Controllers erläutert.

8.2 KOMMUNIKATION UND INFORMATIONSWEITERGABE DER PROGRAMMKOMPONENTEN

Ziel der Model-View-Controller (MVC) -Architektur des Kompositionswerkzeugs ist die Separierung einzelner Programmkomponenten um eine flexiblere Softwaregestaltung zu erlauben. Dadurch soll die spätere Änderung, Erweiterung und Wiederverwendung der einzelnen Programmkomponenten ermöglicht werden. Für die Arbeit in Projektteams ist die Unterteilung der Software in mehrere Bereiche außerdem für die Arbeitsaufteilung wichtig. Die folgende Abbildung stellt die MVC-Architektur des Kompositionswerkzeugs dar.

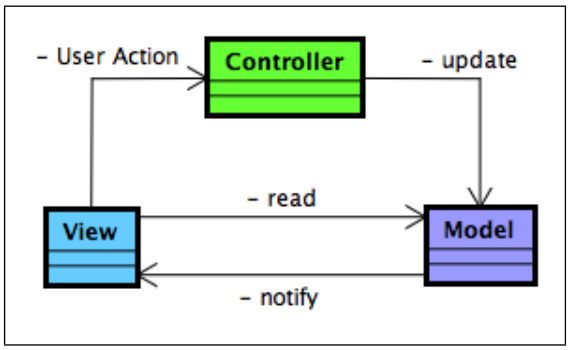


Abbildung 8.2: Umsetzung des Model-View-Controller Architekturmusters im Kompositionswerkzeug

Der Abbildung 8.2 entsprechend, liest das User-Interface als Teil der View-Schicht das zugrundeliegende Anwendungsmodell und zeigt eine entsprechende grafische Repräsentation der Anwendung an. Änderungen, die der Nutzer des Tools vollführt, werden an den Controller gemeldet, welcher das Modell aktualisiert(Anforderung: R2.6). Direkte Aktualisierungen durch die UI erfolgen gemäß dem MVC Muster nicht. Die Benachrichtigung des User-Interface über Änderungen am Modell wird im Kompositionswerkzeug in der Form des Observer Design Pattern vorgenommen [GHJV94]. Werden durch den Controller Änderungen am Modell durchgeführt, dann löst dies einen Event aus, der die Änderungen an die View-Schicht kommuniziert. Die View liest nun die beteiligten Komponenten des Modells neu aus und aktualisiert die Anzeige.

Die Kommunikation zwischen den verschiedenen Schichten des Kompositionswerkzeugs findet über Objekte statt. Dadurch soll eine erweiterte Unabhängigkeit ermöglicht werden. Zusätzlich begünstigt die Objektkommunikation die Implementierung einer Undo-Funktionalität. Änderungen des Toolnutzers werden wie in der Abbildung 7.3 und 8.1dargestellt von der View-Schicht als Change-Objekte(in Abbildungen: ChangeCommand) an den Controller übergeben. Die Change-Objekte enthalten die folgenden Informationen:

- Kommando ID
- Kennzeichnung der Interaktionsart (bswp. AddNew, Change, ...)
- Kennzeichnung der Art der zu ändernden Elemente (Page, UIElement, ...)
- Liste der betroffenen Elemente
- Liste der ausgewählten Elemente
- Liste der zu ändernden Werte

Der Controller wertet diese Objekte aus und entscheidet über die weitere Verarbeitung. Letztendlich müssen die Änderungen in das Anwendungsmodell integriert werden. Dazu werden Modell-Kommando-Objekte (in Abbildungen: CAMCommand) erstellt, die zu einem Set (in Abbildungen: CAMCommandSet) zusammengefügt und an die Modell-Ebene weitergegeben werden. Dabei wird für jede mögliche Interaktion mit dem Modell eine eigene Kommando-Klasse zur Verfügung gestellt. Bei Bedarf kann aus der Klasse eine passende Kommando-Objekt-Instanz erzeugt werden. Die Wahl des korrekten Kommando-Objekts erfolgt durch den Controller. Die Kommando-Objekte enthalten alle Informationen die zur Ausführung der repräsentierten Operation notwendig sind. Unter anderem:

- Beschreibung und Konfiguration neuer Modell-Elemente
- IDs von betroffenen Elementen
- Art und Konfigurationen der Änderungen
- Informationen über zu ändernde Objekte

Im Normalfall übernimmt eine Komponente der Modell-Schicht die Interpretation der Objekte und setzt die angeforderten Änderungen im Anwendungsmodell um. Für die Realisierung des Kompositionswerkzeugs fiel die Entscheidung für eine direkte Implementierung der Umsetzungslogik innerhalb der Kommando-Objekte. Dadurch soll erreicht werden, dass sowohl alle notwendigen Informationen als auch die Umsetzungslogik für eine Modell-Änderung in einem Objekt gekapselt sind. Als Konsequenz ergibt sich allerdings die Notwendigkeit die Kommando-Objekte an das Anwendungsmodell anzupassen. Bei einer Änderung am Anwendungsmodell muss also die Umsetzungslogik der Objekte ebenfalls angepasst werden. Prinzipiell ist die Umlagerung der Objektbehandlung von der Modell-API in die Objekte selbst nur eine Verlagerung der Verantwortlichkeit. Der notwendige Aufwand für eine Anpassung der Objekte ist letztendlich nicht deutlich höher als für die Anpassung der Modell-API an die Änderungen des Modells. Das führte zu der Entscheidung zu Gunsten der erweiterten Kapselung der Umsetzung-Logik in die Kommando-Objekte. Initiiert wird die Ausführung der Objekte dennoch von der API der Modell-Schicht durch den Aufruf der Ausführungsmethode.

8.3 UMSETZUNG DER VISUALISIERUNGS-BASIS-KOMPONENTEN

Die folgende Abbildung 8.3 stellt die Visualisierungskomponenten und deren Aufgaben innerhalb des Visualisierungsprozesses grafisch dar. Die nummerierten Markierungen entsprechen dabei den einzelnen Teilschritten der Visualisierung, die in Abschnitt 7.2 beschrieben sind. Die Teilschritte wurden dabei den verantwortlichen Komponenten zugeordnet.

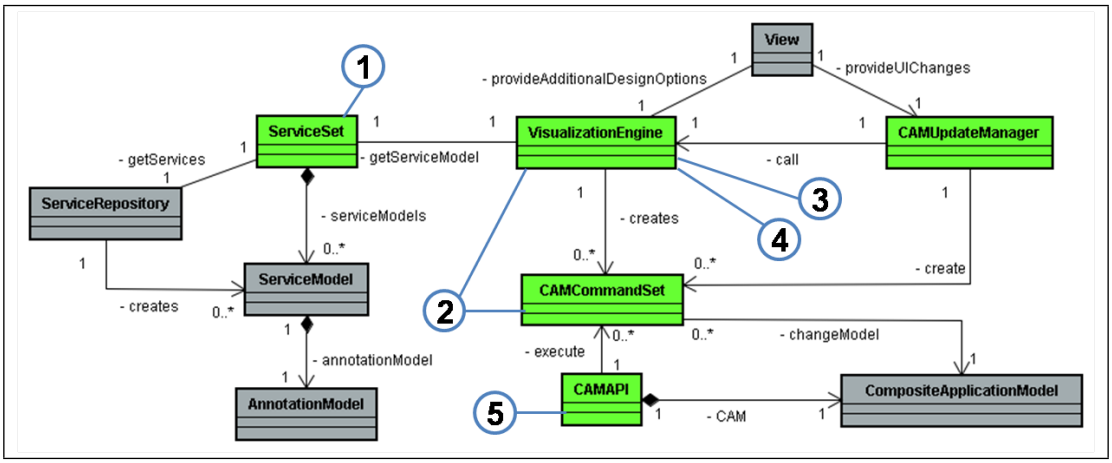


Abbildung 8.3: Darstellung der Visualisierungskomponenten

Die eigentliche Visualisierungsengine vereinigt dabei die Verantwortung von drei der fünf Schritte des Visualisierungsprozesses in sich. Hinzukommt die "CAMUpdateManager" -Komponente, die Änderungen des Nutzers an der Anwendung im Modell umsetzt. In den folgenden Abschnitten werden die die Funktionen der einzelnen Komponenten beschrieben.

8.3.1 Service Set

Die Service-Set-Komponente verwaltet die Modelle aller im Kompositionswerkzeug zwischengespeicherten Services. Auf Anforderung werden die Modelle an die anfordernde Komponente übergeben. Sind die Services noch nicht im Service-Set erhalten, werden diese vom Repository-Controller angefordert. Dieser bezieht die Informationen zur Erstellung der Service-Modelle selbst von einem Webservice. Die Service-Modelle werden nach deren Erstellung im Service-Set vorgehalten um unnötige Kommunikation zu vermeiden. Weiterhin stellt die Service-Set-Komponente mit Hilfe des Repository-Controllers eine Liste mit allen verfügbaren Services und Service-Operationen zur Verfügung.

8.3.2 Visualisierungsengine

Die Visualisierungsengine ist für die Erstellung und Integration der Service-Frontend-Datenstruktur verantwortlich. Dabei teilt sich die Visualisierungsengine selbst in mehrere Komponenten, die jeweils für einen der dargestellten Schritte verantwortlich sind. Die Hauptkomponente ist die Visualisierungskernkomponente (in Abbildung 8.4: VisualizationCoreComponent - grün), welche für das Parsen und Auswerten des Models, das Erstellen der Modellkommandos und das Setzen der Basisparameter zuständig ist. Unterstützt wird die Kernkomponente von der Annotations-Effekt-Bestimmungs-Komponente (in Abbildung 8.4: AnnotationEffektDetermination - blau) und dem plattformspezifischen Regelset (in Abbildung 8.4: RuleEngine - gelb). Die folgende Abbildung 8.4 stellt die Visualisierungsengine detailliert dar. Dabei sind die einzelnen Komponenten, deren Bestandteile und deren Beziehungen zueinander dargestellt.

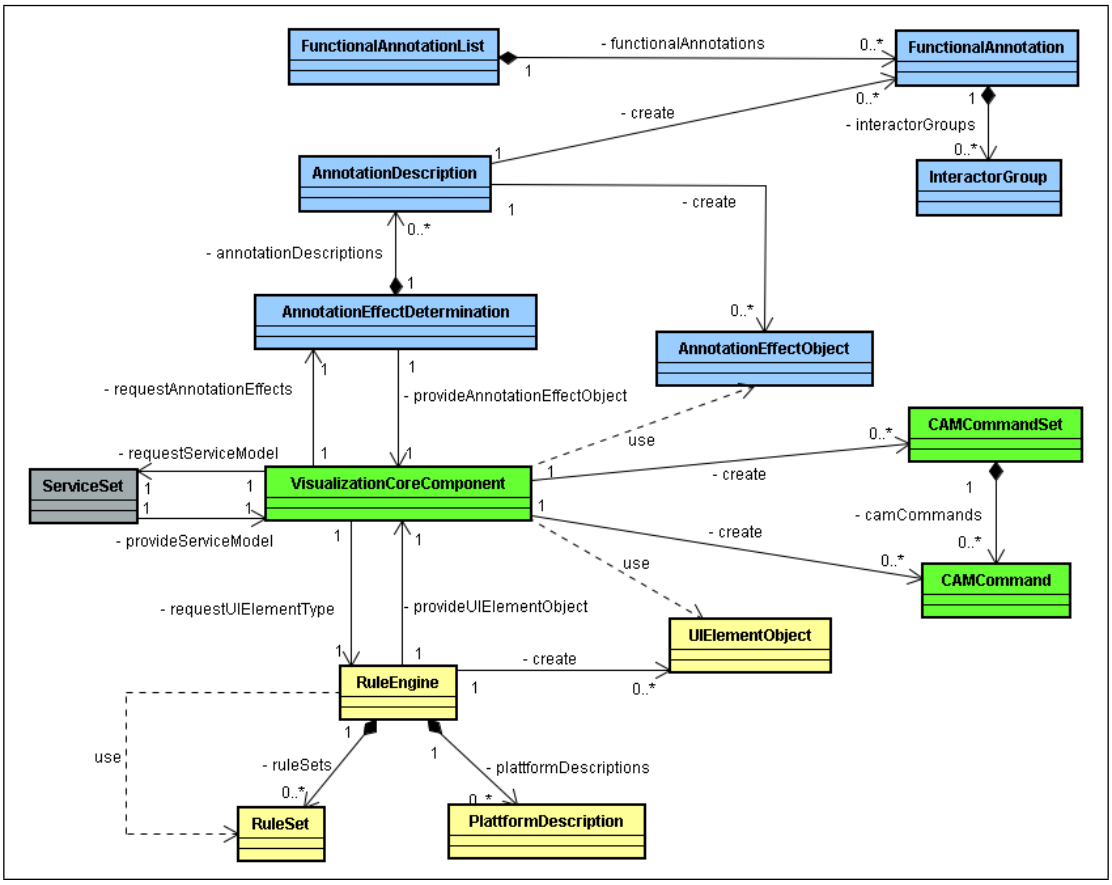


Abbildung 8.4: Komponenten der Visualisierungsengine

8.3.3 Visualisierungskernkomponente (in Abbildung: VisualizationCoreComponent)

Der gesamte Visualisierungsprozess wird dabei von der Visualisierungskernkomponente kontrolliert und gesteuert. Zu Beginn der Visualisierung werden das Modell der anzuzeigenden Service-Operation geparst und die einzelnen Service-Elemente identifiziert. Jedes Service-Element wird anschließend individuell behandelt und analysiert. Das Ergebnis der Behandlung sind Modell-Kommandos, die das Anwendungsmodell verändern und eine Repräsentation des behandelten Service-Elements einfügen. Innerhalb einer Operation können die folgenden Elemente des Service-Modells vorkommen:

- Die Operation selbst
- Parameter-Sets
- Parameter
- Parameter-Elemente

Wie bereits in Kapitel 5 dargelegt, ist besonders die Darstellung der Parameter signifikat. Operationen und Parameter-Sets werden hauptsächlich als strukturgebende Elemente dargestellt. Parameter erhalten eine repräsentative Darstellung, die einen möglichst hohen Eingabekomfort bieten. Das heißt, dass Fehleingaben des Nutzers durch die Einschränkung der Eingabemöglichkeiten verhindert werden. Ausnahme sind komplexe Parameter, deren komplexe Elemente ebenfalls als Strukturelemente umgesetzt werden. Die folgende Tabelle 8.1 zeigt die allgemeine Umsetzung der Service-Elemente:

Service-Element:	Darstellung	UI-Element	Element Anwendungmodell
Service-Operation	Struktur-Element	Horizontale Box	Service-Gruppe
Parameter-Set	Struktur-Unter-Element	Horizontale Box	Input-/Output-Gruppe
Einfache Parameter	Eingabe-/Ausgabeelement	Bestimmung durch Datentyp/Annotation	Interaktor
Komplexe Parameter	Struktur-Unter-Element	Horizontale/Vertikale Box	Interaktoren-Gruppe
Parameter-Elemente	Eingabe-/Ausgabeelement	Bestimmung durch Datentyp/Annotation	Interaktor

Tabelle 8.1: Evaluation der vorgestellten Konzepte hinsichtlich der aufgestellten Anforderungen

Während die strukturgebenden Elemente durch Boxen mit horizontaler bzw. vertikaler Anordnung repräsentiert werden, ist die Wahl des passenden UI-Elements für einfache Parameter bzw. einfache Parameter-Elemente differenzierter. Die Wahl ist abhängig von verschiedenen Faktoren wie dem Datentypen, angehängten Annotationen und der für die erstellte Anwendung gewählten Plattform. Für die Bestimmung des repräsentierenden UI-Elements eines Parameters ist die Regelset-Komponente zuständig.
An allen Elementen eines Service können Annotationen angehängt sein. Diese können, wie in Kapitel 6 beschrieben, in vielfältiger Weise das Visualisierungsergebnis beeinflussen. Die Auswertung der Annotationen übernimmt die Annotations-Effekt-Bestimmungs(AEB)-Komponente. Die beiden zusätzlichen Komponenten der Visualisierungsengine teilen jeweils ihr Ergebnis der Kernkomponente mit, die die Ergebnisse weiter verarbeitet. Das Ergebnis des Visualisierungsprozesses ist ein Modell-Kommando-Set. Die Kernkomponente ist dafür zuständig, dass innerhalb des Sets die Kommandos in der richtigen Reihenfolge vorliegen. Das ist notwendig, um eine korrekte Ausführung des Kommandosets zu ermöglichen. Beeinflusst beispielsweise ein Kommando ein Anwendungsmodell-Element, dass erst durch ein Folge-Kommando angelegt wird, dann ist eine Ausführung des Kommandos und damit des Kommando-Sets nicht möglich. Das Kommandoset wird an die API des Anwendungsmodells übergeben, welche die Kommandos sequentiell ausführt. Jedes Kommando bringt bei Ausführung seine Änderungen in das Anwendungsmodell ein.

8.3.4 Update-Manager

Der Update-Manager ist für die Verarbeitungen der Änderungen an der interaktiven Anwendung zuständig. Die meisten Änderungen werden vom Update-Manager direkt durch Modell-Kommandos umgesetzt. Das Einfügen von neuen Service-Operationen wird an die Visualisierungskomponente delegiert. Der Ablauf einer Änderung ist im Sequenzdiagramm der Abbildung 8.5 dargestellt.

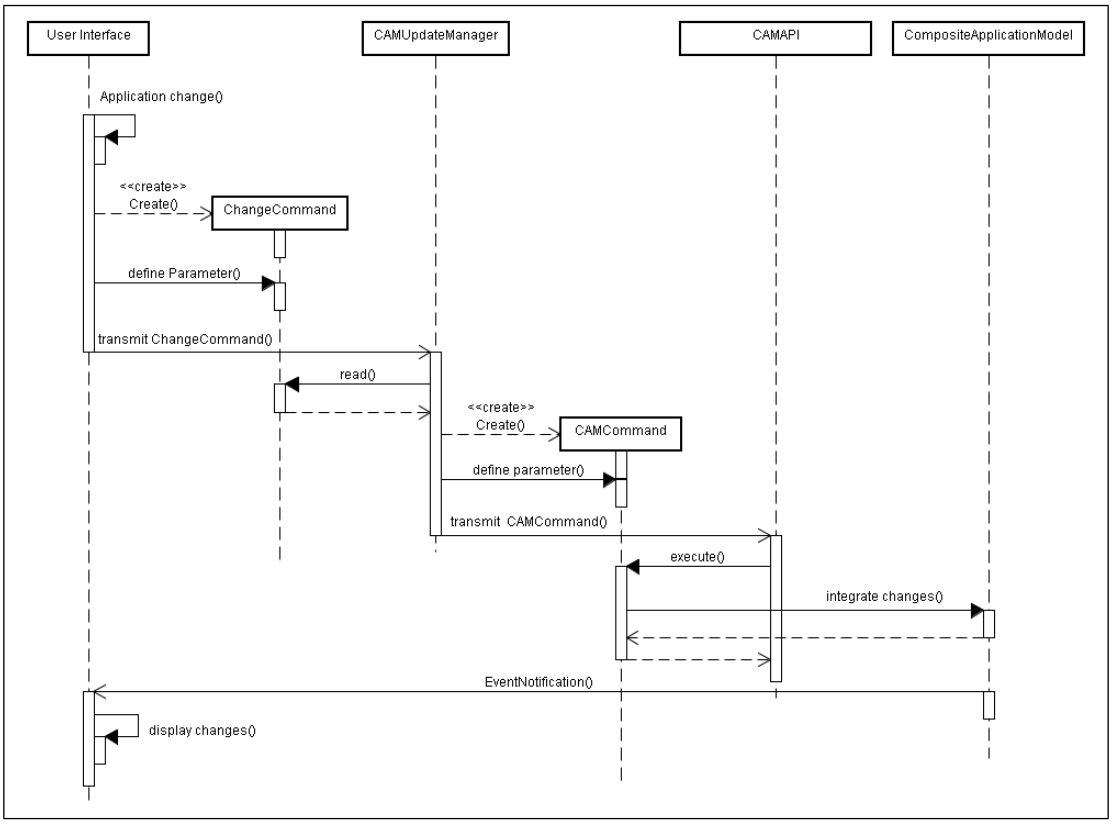


Abbildung 8.5: Sequenzdiagramm zum Ablauf einer Änderung an der Anwendung

Sobald der Nutzer eine Änderung an der modellierten Anwendung vornimmt, erzeugt die UI ein Änderungs-Kommando-Objekt. Dieses Objekt enthält alle relevanten Informationen über die Änderung. Nachdem die UI das Änderungs-Kommando an den Update-Manager übergeben hat, kann dieser die Informationen nutzen um Modell-Kommandos zu generieren. Auch diese Kommandos werden wieder in Kommando-Sets zusammengefasst, wobei ein Kommando-Set der Ausführung einer kompletten Änderung entspricht. Das Kommando-Set wird an die API des Anwendungsmodells übergeben und die Änderungen werden in das Modell eingebracht.

8.4 ANNOTATIONS-EFFEKT-BESTIMMUNG (AEB)

Die AEB-Komponente ist für die Auswertung der an einem Service-Element angehängten Annotationen verantwortlich. Dazu übergibt die Kernkomponente das Service-Element inkl. der Annotationen an die AEB-Komponente. Diese wertet die Annotationen in Abhängigkeit von den Gegebenheiten des Service-Elements aus. Je nachdem um welchen Annotationstyp es sich handelt (vergleiche Abschnitt 6), werden entweder Effekt-Objekte (UI-Annotationen/hybride Annotationen) erzeugt oder ein Eintrag in die Liste der funktionalen Annotationen (funktionale Annotationen) vorgenommen. Die Effekt-Objekte beschreiben die Auswirkungen der Annotation. Die Kernkomponente erhält diese Objekte als Rückgabe von der AEB-Komponente und bezieht die Auswirkungen in den weiteren Visualisierungsprozess mit ein. Für funktionale Annotationen stellt die AEB-Komponente der UI des Kompositionswerkzeugs eine Liste der aktiven Annotationen und der annotierten Elemente zur Verfügung. Anhand dieser Liste kann die UI die Icons für die funktionalen Annotationen (vergleiche Abschnitt 6.2.2) darstellen und die betroffenen Elemente markieren.

8.4.1 Repräsentation funktionaler Annotationen

Die funktionalen Annotationen werden wie in Abschnitt 6.2.3 beschrieben nicht direkt im Anwendungsmodell realisiert, sondern nur metaphorisch im User-Interface des Kompositionstools repräsentiert. Die Auswertung der funktionalen Annotationen ist deshalb relativ unproblematisch möglich. Um eine korrekte metaphorische Darstellung der funktionalen Annotationen sicherzustellen, müssen dem User-Interface die in Abschnitt 7.4.1 dargelegten Informationen in geeigneter Weise zur Verfügung gestellt werden. Das User-Interface kann mit Hilfe dieser Informationen die Darstellung der Annotations-Metaphern und die Markierung der betroffenen Elemente sicherstellten. Für das Kompositionswerkzeug wurde die in der folgende Abbildung 8.6 dargestellte Datenstruktur konzipiert:

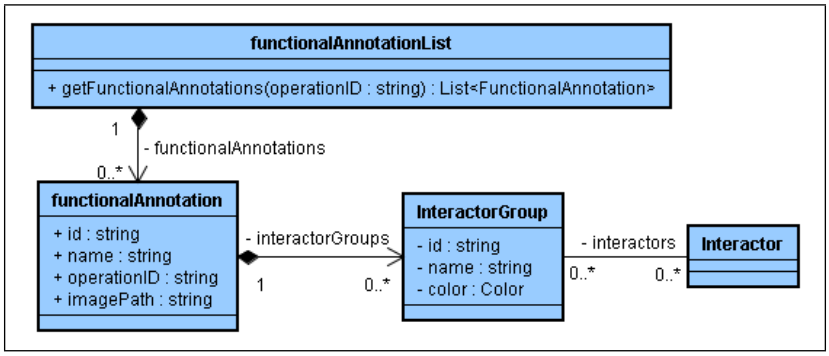


Abbildung 8.6: Datenstruktur für die Darstellung funktionaler Annotationen

Die Datenstruktur spezifiziert Objekte zur Repräsentation der funktionalen Annotationen (in Abbildung 8.6: functionalAnnotation) die für eine Service-Operation angezeigt werden sollen. Die Objekte enthalten alle notwendigen Informationen, die zur Darstellung der funktionalen Annotationen notwendig sind. Unter anderem werden Gruppen von Interaktoren spezifiziert, die von der Annotation betroffen sind und deshalb markiert werden sollen. Da einige funktionale Annotationen mehrere Interaktoren in unterschiedlicher Funktion referenzieren, lassen sich für jede Annotation mehrere Interaktorgruppen angeben. Für jede Interaktorengruppe lässt sich eine eigene Farbe für die Markierung definieren. Die Objekte sind in einer Liste gespeichert, die dem UI dauerhaft zur Verfügung steht. Die Liste implementiert eine Möglichkeit alle funktionalen Annotationen einer Service-Operation d.h. eines Frondends abzurufen. Mit Hilfe dieser Informationen, kann das UI die funktionalen Annotationen, wie in Abbildung 6.2 aus Abschnitt 6.2.3 darstellen.

8.4.2 Auswertung von UI- und Hybrid-Annotationen

Während funktionale Annotationen sich nur metaphorisch im UI widerspiegeln, müssen die Auswirkungen von UI- und Hybrid-Annotationen direkt im Anwendungsmodell umgesetzt werden. Der Umfang der möglichen Auswirkungen auf das Modell ist allerdings begrenzt. Die folgenden Änderungen sind möglich (Elemente aus Anwendungsmodell: Anhang Abschnitt A.3.1, Abbildungen: A.2 , A.3, A.4):

- Hinzufügen neuer Elemente: ViewGroup, Interactor, UI-Element-Property, Page, ModelRelation, FlowTransition
- Ändern von Element-Eigenschaften: Container-Element, Element-Eigenschaften
- Löschen von Elementen
- Änderungen von Referenzen

Für die generische Auswertung der Annotationen ist es also notwendig, die Auswirkungen der einzelnen Annotationen auf die möglichen Änderungen des Anwendungsmodells abzubilden. Beispiele für diese Abbildung sind in Abschnitt 6.2.1 dargestellt. Im Kompositionswerkzeug wird die Abbildung von der AEB-Komponente übernommen. Diese erzeugt generische Objekte ("Annotation-Effect-Objects"), die an die Visualisierungskernkomponente nach erfolgter Auswertung, zurück gegeben werden. Die Visualisierungskernkomponente muss nur eine Möglichkeit der Realisierung der Objekte implementieren und ist deshalb unabhängig von der spezifischen Umsetzung der einzelnen Annotationen. Die Erzeugung der Annotations-Effekt-Objekte ist in der folgenden Abbildung dargestellt:

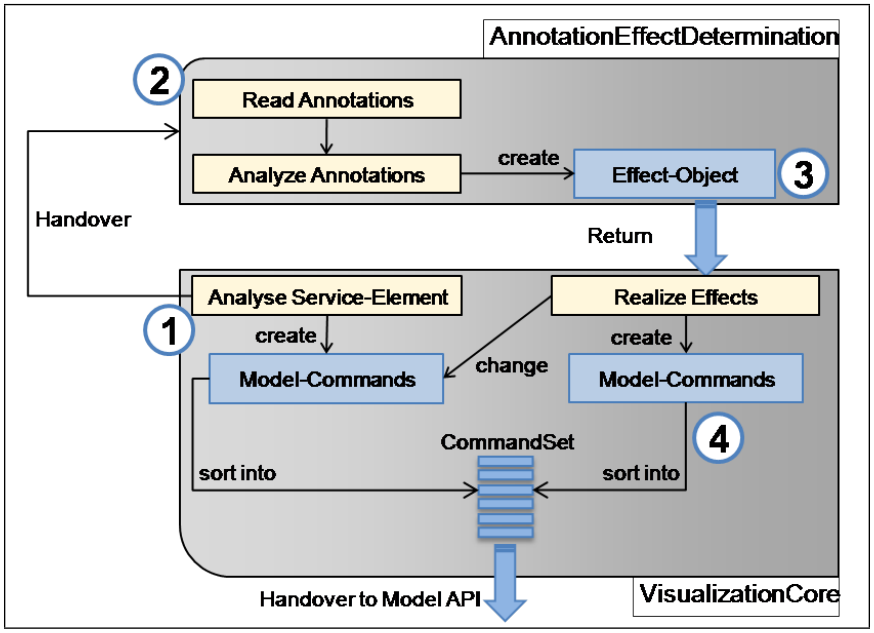


Abbildung 8.7: Erzeugung der Effekt-Objekte

Ausgangspunkt ist die Auswertung von annotierten Service-Elementen (Abbildung 8.7 - 1). Diese werden an die AEB-Komponente zur Auswertung übergeben. Die AEB-Komponente liest die Annotationen des Service-Elements und analysiert deren Auswirkungen (Abbildung 8.7 - 2). Anschließend werden für jeden einzelnen Effekt, den eine Annotation auf das Modell hat, ein generisches Effekt-Objekt erstellt (Abbildung 8.7 - 3). In das Objekt werden alle für die Ausführung der Änderung notwendigen Informationen integriert. Das Effekt-Objekt wird zur Realisierung der Änderungen an die Visualisierungskernkomponente zurückgegeben. Die Kernkomponente wertet die Effekt-Objekte aus, um neue Modell-Kommandos zu erstellen oder bestehende Kommandos zu verändern (Abbildung 8.7 - 4). Schlussendlich werden, wie bereits beschrieben, die Kommandos in das entsprechende Kommandoset einsortiert und an die Modell-API übergeben.

8.4.3 Erweiterbarkeit der Annotationsauswertung

Die Erweiterbarkeit der Annotationsauswertungen, wird entsprechend der Konzeption in Abschnitt 7.4.2 über Beschreibungsklassen in einem zentralen Repository realisiert. Der Zugriff auf die annotationsspezifischen Beschreibungen erfolgt nach dem "Chain of Responsibility" Prinzip [GHJV94]. Das Konzept sieht eine Weitergabe der Anfrage bis zur endgültig zuständigen Klasse vor, die die Anfrage letztendlich bearbeitet. Die Umsetzung geschieht unter Ausnutzung des Fakts, dass alle Annotationen von derselben Oberklasse "Annotation" erben. In dieser Oberklasse wird eine Ausführungsmethode definiert, die von den jeweiligen Kindklassen implementiert wird. Die Ausführungsmethode in der konkreten Annotationen ruft die Auswertungsmethode innerhalb der Beschreibungsklasse auf, die den gleichen Namen wie die zugehörige Annotation trägt. Die

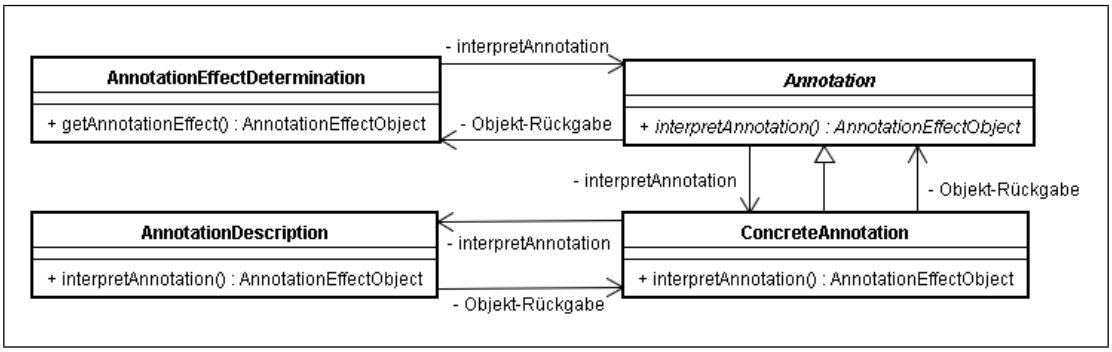


Abbildung 8.8: Anforderungsweitergabe nach dem Beschreibungsklasse-Konzept

Methode übernimmt die Auswertung der Annotation und die Erzeugung der Effekt-Objekte. Die folgende Abbildung stellt die Weitergabe der Anforderung im Rahmen des Kompositionswerkzeugs grafisch dar:
Alle notwendigen Informationen zur Erstellung der Effekt-Objekte werden direkt von den Annotationen entnommen. Das Hinzuziehen weiterer Informationen für die Auswertung ist nicht notwendig, da jede Beschreibungsklasse auf die jeweilige Annotation angepasst ist. Damit ist innerhalb der Klasse die Struktur der Annotation und die Zugriffsmöglichkeiten auf Parameter und Referenzen bekannt. Die folgende Abbildung 8.9 visualisiert Umsetzung des Konzepts der erweiterbaren Annotationsauswertung:

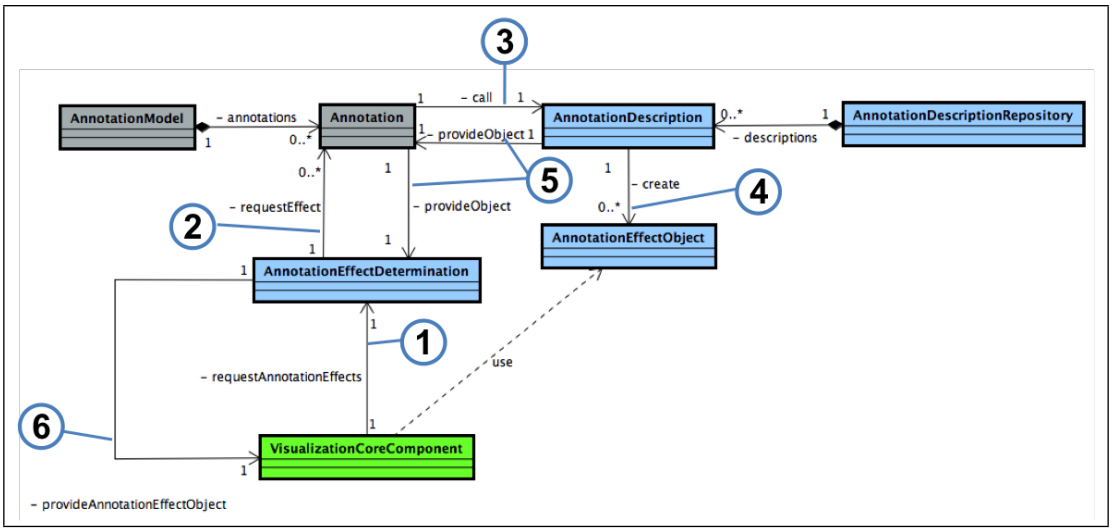


Abbildung 8.9: Auswertung der Annotationen mit Hilfe vom Annotations-Beschreibungsklassen

Die Auswertung der Annotationen umfasst die folgenden, in der Abbildung nummerierten Schritte:

1. Übergabe Service-Elemente an die AEB-Komponente
2. Aufruf der Ausführungsmethode innerhalb der Annotation
3. Weitergabe des Aufrufs an die Beschreibungsklasse der jeweiligen Annotation
4. Erzeugung des Effekt-Objekts
5. Rückgabe des Effekt-Objekts an die AEB-Komponente
6. Übergabe des Effekt-Objekts an die Kernkomponente zur weiteren Verarbeitung

8.5 UMSETZUNG DER REGELSATZ-KOMPONENTE

Die Bestimmung der UI-Elemente zur Repräsentation der Service-Parameter übernimmt die Regelset-Komponente. Dazu wird von der Kernkomponente das Parameter-Element inklusive seines Datentypen an die Regelset-Komponente übergeben. Diese nutzt die vordefinierten Plattformspezifikationen und Regeln um das passende UI-Element zu bestimmen. Die Entscheidung und einige Konfigurationsparameter werden in Form eines UI-Element-Objekts an die Kernkomponente zurückgegeben. Diese bringt das gewählte UI-Element in die definierten Modell-Kommandos ein. Um eine angepasste Darstellung schon während des Kompositionsprozesses zu ermöglichen, stellt die Regelset-Komponente die Plattformspezifikationen ebenfalls dem UI zur Verfügung. Entsprechend dieser Spezifikation kann durch die UI eine angepasste Darstellung gewählt werden, die der gewünschten Zielplattform nahe kommt. Die folgende Abbildung stellt die Bestandteile der Regelset-Komponente grafisch dar:

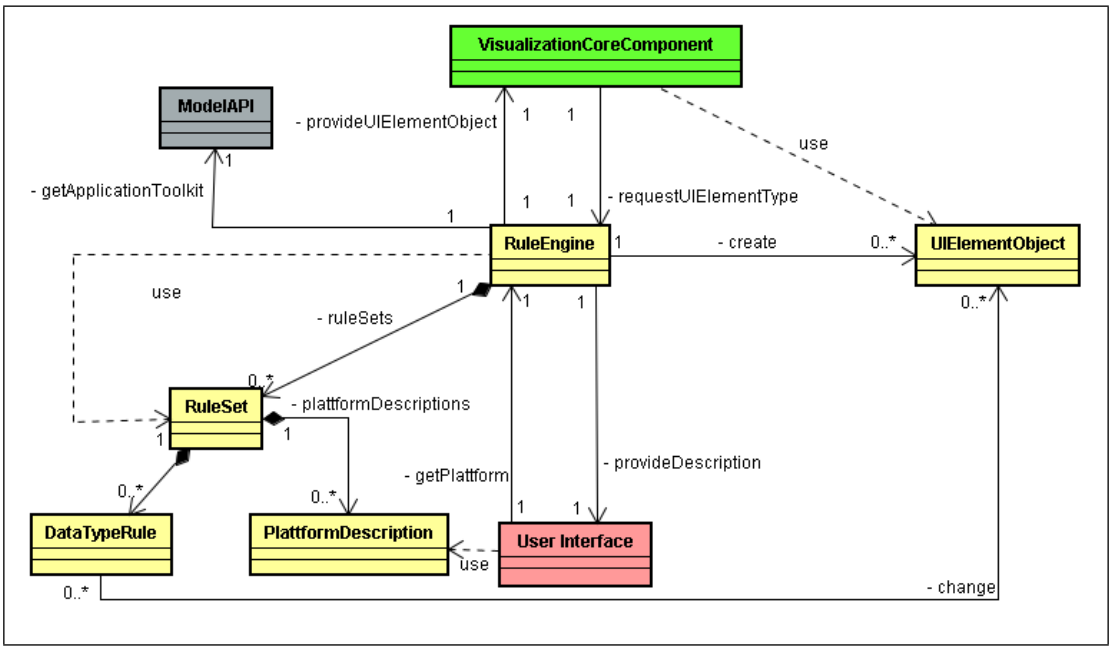


Abbildung 8.10: Bestandteile der Regelset-Komponente

Die Regel-Engine (in Abbildung: Rule-Engine) ist die Kontrolleinheit der Regelset-Komponente, die sowohl die Regelsets (in Abbildung: RuleSet) als auch die Plattformbeschreibungen (in Abbildung: PlattformDescription) verwaltet. Die Regel-Engine analysiert die übergebenen Service-Parameter in Hinblick auf den angehängten Datentyp und dessen Attribute. Dazu werden die plattformspezifischen Regeln für die einzelnen Datentypen herangezogen. Der Zugriff auf die Plattformspezifikationen erfolgt dabei nach einer Variante des "Abstract Factory" Design Pattern [GHJV94]. Die Regelengine stellt ein Verzeichnis zur Verfügung, bei dem alle verfügbare Toolkits registriert werden müssen. Die konkrete Plattform wird durch die Auswahl des Nutzers zum Start des Tools bestimmt. Der eigentliche Funktionsaufruf innerhalb der Regelengine erfolgt auf der abstrakten Oberklasse der Plattformbeschreibungen und wird an die gewählte konkrete Plattformbeschreibung weitergegeben. Auf Basis der Regeln werden der Datentyp und die zugehörigen Attribute ausgewertet. Letztendlich wird ein UI-Element festgelegt, dass den Parameter am komfortabelsten repräsentiert. Zusätzlich werden wichtige Konfigurationsparameter aus den Datentyp-Attributen entnommen. Alle Informationen werden in Form eines UI-Element-Objekts an die Visualisierungskernkomponente zurückgegeben, die das gewählte UI-Element und die Konfigurationsparameter in die Datenstruktur integriert.

8.5.1 Plattformspezifische Darstellung

Für die plattformspezifische Darstellung der Service-Frontends innerhalb des Kompositionswerkzeug sollen die Plattformspezifika herangezogen werden. Mit Hilfe der Spezifika kann die Darstellung der Frontends und im Speziellen der Interaktionselemente an die Gegebenheiten der Plattform angepasst werden. Dazu werden die Silverlight-Element entsprechend der Vorgaben der Zielplattform konfiguriert. Die Plattformspezifika selbst liegen in Form von XML-Dateien vor und werden nach einem XML-Schema definiert. Ein Beispiel für eine solche Plattformspezifikation stellt das folgende Listing dar:

```
<tns:PlattformConfiguration>
 <tns:minResolution>
  <tns:resolution_x>480</tns:resolution_x>
  <tns:resolution_y>320</tns:resolution_y>
 </tns:minResolution>
 <tns:interactionType>MultiTouch</tns:interactionType>
 <tns:dataInputDevice>OnScreenKeyboard</tns:dataInputDevice>
 <tns:supportedToolKit>Iphone SDK</tns:supportedToolKit>
 <tns:supportsMultiPage>true</tns:supportsMultiPage>
 <tns:supportsScrolling>true</tns:supportsScrolling>
 <tns:supportsDataInput>true</tns:supportsDataInput>
 <tns:numberElementsBeforePageBreak>20</tns:numberElementsBeforePageBreak>
 <tns:supportedMIMETyes>
  <tns:supportedMIMEType>png</tns:supportedMIMEType>
  <tns:supportedMIMEType>mp3</tns:supportedMIMEType>
  <tns:supportedMIMEType>mp4</tns:supportedMIMEType>
  <tns:supportedMIMEType>fla</tns:supportedMIMEType>
   ...
 </tns:supportedMIMETyes>
 <tns:supportedUIElements>
  <tns:UIElement>
   <tns:ElementName>ComboBox</tns:ElementName>
   <tns:StandardHeight>20</tns:StandardHeight>
   <tns:StandardWidth>100</tns:StandardWidth>
    ...
  </tns:UIElement>
   ...
 </tns:supportedUIElements>
</tns:PlattformConfiguration>
```

Listing 8.1: Ausschnitt: XML-Plattform-Beschreibung

Das zugehörige Schema ist im Anhang A.5.2 zu finden.

In der aktuellen Implementierung ist die Regelsatz-Komponente für den Einsatz der dynamischen UI-Veränderung vorbereitet. Dazu wurden die folgenden Aspekte in das Kompositionswerkzeug integriert:

- Möglichkeit der Hinterlegung von Plattform-Beschreibungen
- XML-Plattformbeschreibungen inkl. XML-Schema-Definition
- Auswertung der UI-Elemente unter Berücksichtigung der Plattform-Spezifika
- Berücksichtigung der festgelegten Konfigurationsparameter
- Möglichkeit der Auswertung der XML-Beschreibungen

Die Implementierung der View-Komponente ist momentan allerdings noch nicht in der Lage die Plattformspezifikationen zu verarbeiten und die Darstellung der UI-Elemente entsprechend anzupassen. Die Implementierung der View-Komponente ist nicht Teil dieser Arbeit. Die aktuelle Plattformdefinition basiert auf Silverlight, was als Ziel-Toolkit für browserbasierte Webanwendungen vorgesehen ist. Weitere Plattformdefinitionen können mit Hilfe des XML-Schemas definiert und in das Kompositionswerkzeug integriert werden.

8.6 ZUSAMMENFASSUNG

Innerhalb dieses Kapitels wurde die Implementierung der Controller-Schicht beschrieben. Dabei wurde neben der Gesamtarchitektur besonders auf die Visualisierungsengine, die Annotations-Effekt-Bestimmung und die Regelsatz-Komponente eingegangen. Die Visualisierungskernkomponente ist als Hauptbestandteil der Visualisierungs-Engine für die Steuerung des Visualisierungsprozesses zuständig. Sie wertet die Service-Modelle aus und erstellt die Datenstruktur für die Service-Frontends. Außerdem verwaltet sie die Annotations-Effekt-Bestimmungs- und die Regelsatzkomponente. Beide Komponenten werden für die Optimierung des Anzeigeergebnisses verwendet. Besonders die geforderte einfache Erweiterbarkeit der einzelnen Komponenten, machte die Nutzung von alternativen, generischeren Implementierungskonzepten notwendig. Die Annotationsauswertung setzt dabei auf ein zentrales Beschreibungsklassen-Repository und generische Kommunikationsobjekte um die Unabhängigkeit von anderen Kompontenen zu ermöglichen. Die Regelsatz-Komponente verwendet ebenfalls Objekte für die Kommunikation und XML für die Beschreibung der Plattformen.

9 EVALUATION UND TEST

Innerhalb der vorliegenden Arbeiten wurden mehrere Konzepte für die Visualisierung und Darstellung von Service-Frontends vorgestellt. Diese Konzepte gilt es zu evaluieren und nachzuweisen, dass diese innerhalb der Kompositionswerkzeugs angewendet werden können. Die Evaluation teilt sich in zwei Unterkapitel. Zunächst wurde eine Nutzerstudie inkl. einer Vorbetrachtung durchgeführt, bei der Testpersonen die Nutzbarkeit des Kompositionswerkzeugs und der Frontends abgeprüft haben. Die Testbeschreibungen und für diese Arbeit relevanten Ergebnisse sind im ersten Teil dieser Evaluation zu finden. Im Nachgang wurde eine technische Evaluation durchgeführt, bei der die erzeugten Datenmodelle für die Frontends überprüft wurden. Im Anschluss findet eine kurze Diskussion statt, deren Ergebnis einige Empfehlungen für die weitere Entwicklung darstellen.

9.1 NUTZERSTUDIE ZUM TEST DER NUTZBARKEIT DES KOMPOSITIONSWERKZEUGS UND DER FRONTENDS

Ziel der Evaluation war die Überprüfung des aktuellen ServFace Builder Prototypen hinsichtlich seiner Bedienbarkeit, Erlernbarkeit und Nachvollziehbarkeit. Dabei sollten wichtige Erkenntnisse über die Verständlichkeit der verwendeten Interaktions- und Gestaltungskonzepte sowie der Service-Frontends gewonnen werden.

Für die Evaluation im Rahmen dieser Arbeit waren besonders die folgenden Aspekte interessant:

- die Verständlichkeit der visualisierten Service-Frontends
- die Darstellung der Frontends in Hinblick auf die Umsetzung in der resultierenden Applikation
- die Darstellung der funktionalen Annotationen und die Verständlichkeit der zugehörigen Tooltips

Im Vorfeld fand eine kurze Vorbetrachtung über die Verständlichkeit der Metaphern und Icons der funktionalen Annotationen statt. Der Aufbau, die Durchführung und die Ergebnisse der Vorbetrachtung werden im Folgenden beschrieben.

9.1.1 Vorbetrachtung zur Darstellung funktionaler Annotationen

In Abschnitt 6.2 wurden die Auswirkungen der Annotationen auf die Darstellung der Service-Frontends beschrieben. Vorrangiges Ziel der Vorbetrachtung war das Abprüfen der Verständlichkeit der Metaphern und der zugehörigen Icons. Zusätzlich wurde die Akzeptanz der in Kapitel 4 vorgestellten Empfehlungen aus den Gestaltungsrichtlinien überprüft.

9.1.1.1 Beschreibung der Testpersonen

Die Testpersonen der Evaluations-Vorbetrachtung waren Computer-Nutzer mit unterschiedlicher Erfahrung im Bereich der Informations-Technologie. Dadurch sollte die Verständlichkeit der Metaphern und Icons möglichst über Erfahrungsgrenzen hinweg überprüft werden.

Insgesamt wurden 10 Personen befragt, die die folgenden Eigenschaften aufwiesen:

- 10 Teilnehmer
- 3 weiblich/ 7 männlich
- Alter: zwischen 21 und 55
- Bildung: Laufendes oder abgeschlossenes Studium
- Bildungsrichtungen:
  - Lehramt Bauingenieurwesen (1)
  - Mikroelektronik (1)
  - Germanistik (2)
  - Wirtschaftswissenschaften (2)
  - Informatik (3)
  - Informationsverarbeitungstechnik (1)

Auf Grund der unterschiedlichen Erfahrung wurde erwartet, dass die Testpersonen die metaphorische Darstellung der funktionalen Annotationen unterschiedlich einschätzen. Durch die Mischung der Erfahrungsgruppen sollte sowohl die umfassende Erfahrung der IT-nahen Personen als auch die unbedingte Vorstellungskraft IT-ferner Probanden einbezogen werden.

9.1.1.2 Inhalte, Ziele und Durchführung der Befragung

Das Hauptziel der Vorbetrachtung war die Überprüfung der Verständlichkeit der gefunden Metaphern und Icons, für die Repräsentation der funktionalen Annotationen. Zusätzlich sollte getestet werden, ob die gefundenen Design-Empfehlungen durch die Testpersonen angenommen werden. Die Testpersonen wurden von dem Tester durch die gesamte Evaluation geführt. Dieser stellte die Fragen, führte Protokoll über die Antworten und Reaktionen und stand für Rückfragen zur Verfügung. Die Inhalte der einzelnen Abschnitte sind nachfolgend aufgeführt:

Überprüfung der Verständlichkeit der Metaphern und Icons:
In diesem Abschnitt der Evaluation wurden zuerst die Begrifflichkeiten der 8 funktionalen Annotationen geklärt. Dabei waren die Testpersonen zunächst angehalten eine eigene Erklärung zu finden. Wenn die Erklärung der Testpersonen von der eigentlichen Bedeutung abwich oder keine Erklärung gefunden werden konnte, dann wurde die Begrifflichkeit durch den Tester dargelegt. Anschließend bekamen die Testpersonen die Aufgabe selbst eine bildliche Metapher zur Repräsentation der funktionalen Annotationen zu finden. Bedingung war dabei, dass die Beschreibung möglichst gegenständlich und in wenigen Worten erfolgt. Als drittes wurden die in dieser Arbeit entworfenen Icons den Testpersonen vorgestellt. Die Aufgabe der Testpersonen war es nun die funktionalen Annotationen den Icons zuzuordnen und so die Erkennbarkeit der Icons zu verifizieren. Zum Abschluss dieses Abschnitts der Evaluation hatten die Testpersonen die Möglichkeit, Feedback zu den vorgestellten Metaphern und Icons abzugeben und Verbesserungen vorzuschlagen.

Evaluation der Design-Empfehlungen:
Dieser Abschnitt der Evaluation soll überprüfen, ob die vorgeschlagenen Design-Empfehlungen tatsächlich von Endnutzern wahrgenommen und akzeptiert werden. Zu diesem Zweck wurde ein Webformular auf Silverlight-Basis implementiert, welches für jede Gestaltungsrichtlinie eine Auswahl von drei bis vier Design-Elementen bereitstellt. Die Design-Elemente waren dabei speziell konfiguriert. Eins der Elemente entsprach dabei der gefunden Design-Empfehlung. Die Testpersonen sollten nun jeweils aus der Auswahl der Design-Elemente die ihrer Meinung nach beste Konfiguration auswählen. Dadurch konnte überprüft werden, wie viele Testpersonen tatsächlich dem Profil der Gestaltungs-Experten entsprechen und welche Vorschläge der Gestaltungsrichtlinien sich verifizieren lassen.

9.1.1.3 Ergebnisse der Vorbetrachtung

Bei der ersten Aufgabe sollten die Testpersonen ihre eigenen Metaphern zur Darstellung der funktionalen Annotationen aufstellen. Die Metaphern der Testpersonen wurden dann mit den Metaphern aus dieser Arbeit verglichen. Die selbstständige Findung von Metaphern für die doch recht komplizierten Begrifflichkeiten der funktionalen Annotationen erwies sich für die meisten Testpersonen als schwierig, was aus den Kommentaren und Reaktionen der Probanden erkannt werden konnte. Die Erwartungen nach Übereinstimmungen waren daher gering. Das folgende Säulendiagramm stellt die Anzahl der Übereinstimmungen nach funktionaler Annotation grafisch dar:

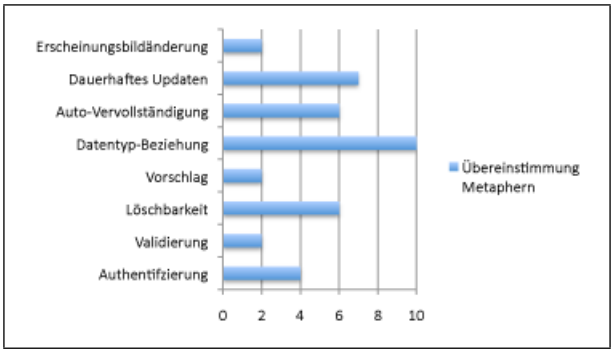


Abbildung 9.1: Übereinstimmungen Metapher Testpersonen zu Metaphern dieser Arbeit

Bei einigen Metaphern gab es recht hohe Übereinstimmungen. Dazu gehören die Metaphern für das "Dauerhafte Updaten", die "Auto-Vervollständigung", die "Datentyp-Beziehungen" und die Löschbarkeit. Bei der Annotation "Datentyp-Beziehungen" nannten sogar alle Testpersonen die gleiche Metapher, wie die in der Arbeit vorgeschlagene. Wenige Übereinstimmungen traten bei der Darstellung der "Validation", "Suggestion" und der "Appearance Change" - Metapher auf. Schon die Konzeption der Metaphern für diese beiden Annotationen im Rahmen dieser Arbeit erwies sich als schwierig. Auch die Testpersonen hatten Schwierigkeiten beim Finden von Metaphern für diese Annotationen. Insgesamt ist das Ergebnis aber zufriedenstellend, da es für jede Annotation mindestens 2 Personen gab, die eine ähnliche Metapher genannt haben. Zu erwarten ist nun bei den besonders oft genannten Annotations-Metaphern ein hoher Wiedererkennungswert bei der Icon-Zuordnung im zweiten Teil der Evaluation. In diesem Teil der Evaluation wurde die Erkennbarkeit der Icons evaluiert. Die Testpersonen mussten die im Rahmen dieser Arbeit entworfenen Icons den funktionalen Annotationen zuordnen. Das folgende Diagramm stellt das Ergebnis dieser Zuordnungsaufgabe dar:

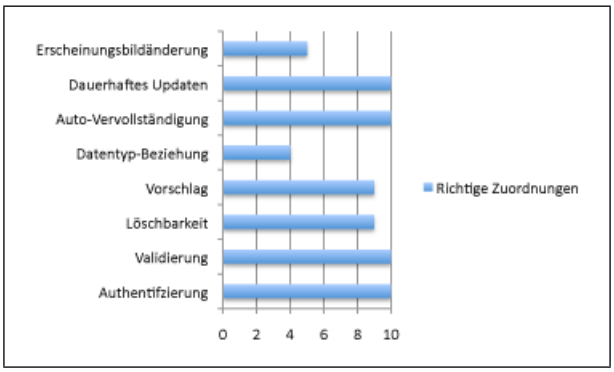


Abbildung 9.2: Richtige Zuordnung der Icons zu den funktionalen Annotationen

Bis auf zwei Icons war die Erkennungsrate mit 100% bzw. 90% sehr hoch, was für die gefundenen Metaphern und die erstellten Icons spricht. Lediglich bei dem Icon für die "DataType Relationship" und die "Appearance Change" - Annotation hatten die Testpersonen Problem. Diese beiden

Icons wurden sehr häufig verwechselt. Das heißt, dass sich beide Icons zu ähnlich sehen und die Darstellung der Metaphern nicht eindeutig voneinander abgegrenzt werden konnte. Das spricht für die Notwendigkeit die grafische Differenzierung für diese beiden Metaphern zu erhöhen. Ein wichtiger Fakt ist, dass im späteren Verlauf die Darstellung der funktionalen Annotationen durch Tooltips unterstützt wird. Die korrekte Zuordnung sollte in diesem Fall ohne Probleme funktionieren.

Der dritte Teil der Evaluation der Metaphern und zugehörigen Icons war die Möglichkeit der Testpersonen direkt Feedback und Verbesserungsvorschläge zu den einzelnen Icons einzugeben. Neben vielen Einzelnennungen wurden besonders häufig eine Verbesserung des Kontrastes der Icons und die Erhöhung der Helligkeit des Hintergrundes vorgeschlagen. Entsprechend der Vorschläge durch die Nutzer wurden die Icons überarbeitet und werden in der überarbeiteten Form im Kompositionswerkzeug verwendet.

Der letzte Teil der Vorbetrachtung war für Abprüfung der Empfehlungen aus Kapitel 4 vorgesehen. Die meisten Vorschläge wurden durch die Nutzer bestätigt. Eine detaillierte Aufschlüsselung der Ergebnisse ist auf dem Datenträger dieser Arbeit zu finden. Die durchschnittliche Übereinstimmung betrug für alle Nutzergruppen zusammengenommen 75,5%. Zwischen den Personen mit unterschiedlicher Computererfahrung konnten keine gravierenden Unterschiede festgestellt werden. Oftmals gingen die Antworten, die nicht den Erwartungen entsprachen, in eine restriktivere Richtung, weshalb diese Antworten den Guidelines nicht wirklich wiedersprechen, sondern diese nur weiter einschränken.

9.1.1.4 Zusammenfassung

Das Hauptziel der Evaluation der Designentscheidungen war die Verständlichkeit und Erkennbarkeit der Metaphern und Icons für die funktionalen Annotationen zu überprüfen. Die Erkennungsrate der Icons war in den meisten Fällen sehr hoch, was die Verständlichkeit der Metaphern und die Erkennbarkeit der Icons bestätigt. Bei einigen funktionalen Annotationen stimmten auch die durch die Testpersonen gefundenen Metaphern mit den Metaphern dieser Arbeit überein, was die Verständlichkeit der Metaphern dieser Annotationen zusätzlich konstatiert. Die Icons von 2 Annotationen wurden häufig verwechselt, was den Schluss auf eine unzureichende Abgrenzung dieser Icons zulässt. Diese Icons sollten demnach überarbeitet werden. Im Zuge der Evaluation wurden die Testpersonen mit den Begrifflichkeiten der Annotationen vertraut gemacht, was sowohl die Findung der Metaphern als auch die Zuordnung der Icons vereinfachte. Im fertigen Kompositionswerkzeug übernehmen diese Erläuterungsaufgabe ein Handbuch, eine Einführung und die Tooltips direkt an den funktionalen Annotationen. Für die Nutzer des Tools ist deshalb eine hohe Erkennungsrate zu erwarten. Die beiden häufig vorgeschlagenen Verbesserungen "Verbesserung des Kontrastes" und "Erhöhung der Hintergrundhelligkeit" sollten für alle Icons umgesetzt werden. Das Ergebnisse des Nebenaspekts der Vorbetrachtung entsprach weitestgehend den Erwartungen. Durchschnittlich 75,5% der Design-Empfehlungen wurden von den Testpersonen bestätigt.

9.1.2 Evaluation des Kompositionswerkzeugs

Im Hauptteil der Evaluation wurde ein funktionierender Prototyp des Kompositionswerkzeugs eingesetzt. Die Testteilnehmer konnten diesen Prototyp verwenden und alle Aspekte des Kompositionsprozesses ausprobieren. Der Test selbst war in 4 Abschnitte unterteilt:

1. Fragen zum Erfahrungshintergrund
2. Werkzeug Exploration und Feedback

3. Erstellung einer Beispielanwendung
4. Abschließende Fragen

Während der Evaluation wurden viele nützliche Daten gesammelt, die in den weiteren Entwicklungsprozess des Kompositionstools einfließen. Für die Evaluation dieser Arbeit ist nur eine Untermenge der Daten relevant. Auf diese Untermenge wird sich bei der Präsentation der Ergebnisse und der Schlüsse beschränkt.

9.1.2.1 Teilnehmer der Evaluation

- 12 Teilnehmer
- 5 weiblich/ 7 männlich
- Alter: zwischen 19 und 25
- Bildung: Laufendes oder abgeschlossenes Studium
- Bildungsrichtungen:
  - Kunstgeschichte (1)
  - Medizin (1)
  - Wirtschaftsingenieurwesen (2)
  - Internationales Management (2)
  - Wirtschaftsinformatik (1)
  - Chemieingenieurwesen (1)
  - Lehramt: Englisch/Geschichte (1)
  - Maschinenbau (1)
  - Psychologie (2)

Alle Teilnehmer absolvieren oder absolvierten ein Hochschulstudium. Da das Kompositionstool für die Verwendung durch Nutzer ohne umfassende Computerkenntnisse gedacht ist, wurde bei der Wahl der Probanden darauf geachtet, dass möglichst wenig Studenten der Informatik oder artverwandter Studiengänge, an der Studie teilnehmen. Die Probanden waren Studenten der folgenden Studiengänge:

9.1.2.2 Versuchsdurchführung

Den Probanden stand jeweils ein Computer zur Verfügung, auf welchem der Prototyp installiert war. Zusätzlich war Software zur Aufzeichnung der Programmbedienung und der Gespräche installiert. Die Probanden nahmen jeweils einzeln an dem Test Teil. Den Testkandidaten wurden kurze Informationen zu jedem Abschnitt und wenn notwendig über das Kompositionswerkzeug zur Verfügung gestellt. Es wurde darauf geachtet, dass die Informationsmenge minimal gehalten wurde. Die Nutzer sollten möglichst selbst die Lösung auf Probleme finden. 2 Tester betreuten die Probanden und standen für Rückfragen zur Verfügung. Rückfragen wurden nur beantwortet, wenn eine eigenständige Lösung durch die Probanden nicht mehr möglich war. Es folgt eine Kurzbeschreibung der Evaluationsteile:

Werkzeug Exploration und Feedback:
In diesem Teil konnten sich die Probanden selbstständig mit dem Tool vertraut machen und dessen Funktionen ohne umfangreiche Einführung oder Hilfestellung ausprobieren. Im Anschluss wurden einige Fragen zu dem ersten Eindruck gestellt.

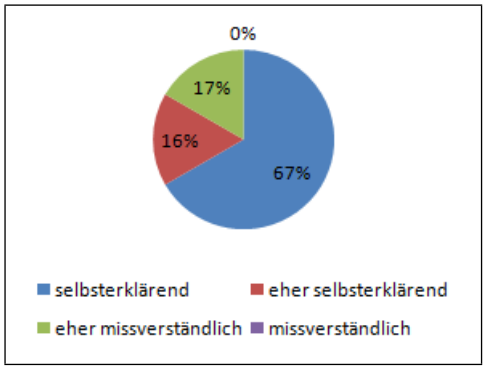
Erstellung einer Beispielanwendung
In diesem Teil der Evaluation sollten die Testteilnehmer eine kleine Anwendung erstellen. Dazu wurde Ihnen ein Szenario vorgegeben, in das sie sich hinein versetzen mussten. Im Rahmen dieses Szenarios konnten Sie dann die Beispielanwendung modellieren.

Abschließende Fragen
Im letzten Teil der Evaluation wurden Fragen zum Umgang mit dem Werkzeug im Rahmen der Erstellung der Beispielanwendung gestellt.

9.1.2.3 Relevante Ergebnisse der Evaluation

Beginnend mit dem Teil "Werkzeug Exploration und Feedback" werden im Folgenden die relevanten Ergebnisse der Evaluation präsentiert. Die vollständige Evaluation ist auf der beigelegten CD im PDF-Format zu finden.

1. (2.10. in Evaluation) Wie beurteilen Sie den Aufbau der Funktionsbausteine?:

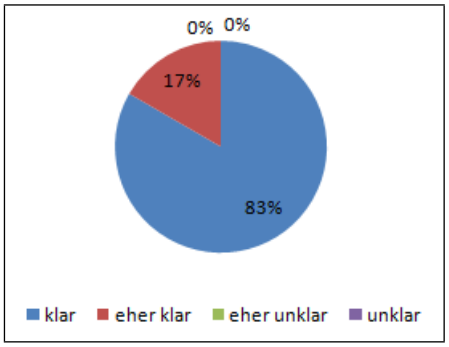


selbsterklärend	Eher selbsterklärend	Eher missverständlich	missverständlich
8	2	2	0

Tabelle 9.1: Wie beurteilen Sie den Aufbau der Funktionsbausteine?

Für 2/3 der Testteilnehmer war der Aufbau der Frontends sofort zu erfassen. Weitere 2 Personen fanden den Aufbau zwar nicht komplett selbsterklärend, tendierten aber eher in die positive Richtung. Lediglich 2 Personen empfanden die Gestaltung der Frontends als eher missverständlich. Daraus lässt sich schließen, dass der grundsätzliche Aufbau verständlich ist und beibehalten werden kann. Allerdings sind auch hier Verbesserungen möglich, die allerdings die Kandidaten nicht konkret benannt haben.

2. (2.12. in Evaluation) Wie bewerten Sie die Benennung der Elemente?:



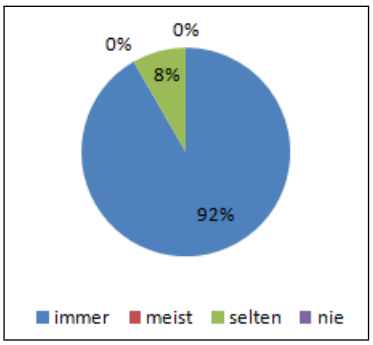
Klar	Eher klar	Eher unklar	Unklar
10	2	0	0

Tabelle 9.2: Wie bewerten Sie die Benennung der Elemente?

Die Ergebnisse bezüglich der Elementbenennung zeigen, dass die Begriffswahl im aktuellen Prototyp gut angenommen wurde. Zwar wurden für alle Elemente Annotationen für die Benennung verwendet. Das bestätigt allerdings den Nutzen der Annotationen.

Bei der Erstellung der Beispielanwendung war zu beobachten, dass die Anwender besser und schneller mit der Anwendung zurecht kamen. Die kurze Evaluationsphase reichte als Einarbeitung aus. Folgende konkrete Ergebnisse haben sich ergeben:

3. (4.7. in Evaluation) War Ihnen klar, welche Funktionsbausteine Sie auswählen sollten?:

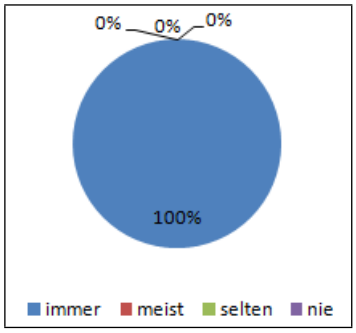


Immer	Meist	Selten	Nie
11	0	1	0

Tabelle 9.3: War Ihnen klar, welche Funktionsbausteine Sie auswählen sollten?

Die meisten Nutzer hatten zu keiner Zeit Probleme die richtigen Funktionsbausteine auszuwählen. Nur eine Person tat sich mit der Auswahl schwer. Die Auswahl der richtigen Funktionsbausteine ist allerdings hauptsächlich von der Benennung der Operation innerhalb der Service-Beschreibung oder einer entsprechenden Annotation abhängig. Die im Prototyp gewählten Namen waren als Annotationen gegeben und entsprechend auf den Wissensstand der Probanden angepasst.

4. (4.9. in Evaluation) Waren die einzelnen Funktionen der Bausteine transparent?:

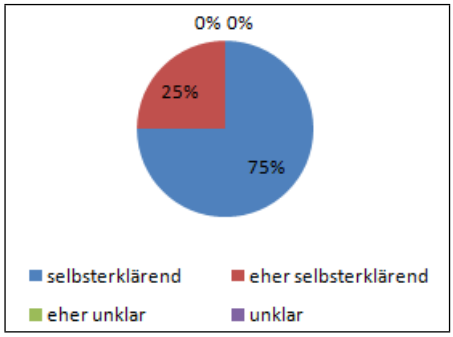


Immer	Meist	Selten	Nie
12	0	0	0

Tabelle 9.4: Waren die einzelnen Funktionen der Bausteine transparent?

Die Bedeutung der einzelnen Funktionen der Bausteine erkannten alle Benutzer ohne Probleme. Das deutet auf einen leicht erfassbaren Aufbau, eine gute Strukturierung und eine gut Verständlichkeit der Element-Benennung hin. Der aktuelle Weg der Frontend-Darstellung kann beibehalten werden.

4. (4.11. in Evaluation) Wie empfanden Sie die Gestaltung und den Aufbau der Funktionsbausteine?:

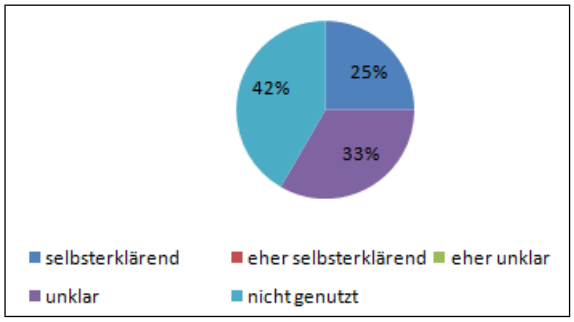


Selbsterklärend	Eher selbsterklärend	Eher unklar	Unklar
9	3	0	0

Tabelle 9.5: Wie empfanden Sie die Gestaltung und den Aufbau der Funktionsbausteine?

Schon in Frage 1 des explorativen Evaluationsabschnittes wurde der Aufbau der Funktionsbausteine gut bewertet. Allerdings gaben zu diesem Zeitpunkt immerhin noch 2 Personen an, dass der Aufbau eher missverständlich sei. Das Ergebnis hat sich insofern verbessert, dass nun alle Personen den Aufbau als selbsterklärend oder zumindest eher selbsterklärend bezeichneten. Das bestätigt das Designkonzept und besonders dessen leichte Erlernbarkeit.

5. (4.13. in Evaluation) Wie transparent waren die Icons der Funktionsbausteine?:

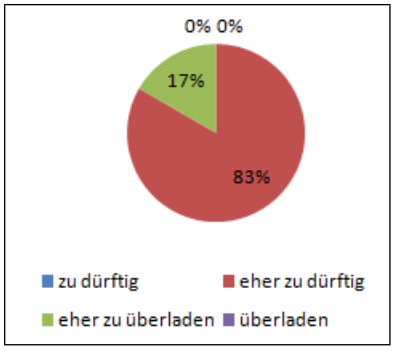


Selbsterklärend	Eher selbsterklärend	Eher unklar	Unklar	nicht genutzt
3	0	0	4	5

Tabelle 9.6: Wie transparent waren die Icons der Funktionsbausteine??

Anhand des Ergebnisses ist zu erkennen, dass einem Großteil der Probanden die Funktion der Icons unklar war oder gar nicht beachtet wurde. Bei den zusätzlichen Informationen zur Frage, gaben auch 4 Personen direkt an, dass die funktionalen Annotationen nicht benötigt waren, um die Aufgabe zu erfüllen. Tatsächlich war für die Erfüllung der Aufgabe die Nutzung von erweiterten Funktionen nicht notwendig. Weder die Umbenennung der Elemente, noch die direkte Verknüpfung von Seiten oder die benannten funktionalen Annotationen mussten verwendet werden um die Beispielanwendung zu erstellen. Demnach haben auch entsprechend wenig Probanden diese Funktionalitäten genutzt. Nicht geklärt werden konnte während der Evaluation ob die Probanden die Funktionen genutzt und verstanden hätten, wenn deren Nutzung erforderlich gewesen wäre. In gewisser Weise kann man diese Funktionen als eine erweiterte Funktion betrachten.

6. (4.15. in Evaluation) Wie bewerten Sie den Funktionsumfang der Bausteine?:



zu dürftig	Eher zu dürftig	Eher zu überladen	überladen
0	10	2	0

Tabelle 9.7: Wie bewerten Sie den Funktionsumfang der Bausteine??

Die Bewertungsskala der Frage 15 unterscheidet sich von den anderen Fragen. Die äußeren Grenzen standen für die negativen Extreme "zu dürftig" und "überladen" im Sinne des Funktionsumfangs. Das optimale Ergebnis lag demnach in der Mitte, wobei die Probanden dennoch eine Entscheidung treffen mussten, in welche Richtung sie tendierten. Die Antworten aller Probanden

lagen jeweils in der Mitte, was den Bausteinen einen guten Funktionsumfang attestiert. 10 Probanden tendierten dabei in die Richtung "eher zu dürftig". Die restlichen 2 Probanden tendierten in die entgegengesetzte Richtung. Das steht natürlich im Widerspruch zueinander. Eine der beiden Personen, die die Bausteine zu überladen fanden, gab sogar direkt an, dass der Funktionsumfang verringert werden sollte. Es handelt sich also nicht um eine Abweichung auf Grund der mehr oder weniger notwendigen Entscheidung, sondern tatsächlich um eine Empfindung der Probanden. Die Gründe warum 2 Probanden die Frontends als zu umfangreich empfanden, obwohl alle anderen diese eher als zu dürftig bezeichneten können vielfältig sein. Konkrete Angaben dazu konnten nicht gesammelt werden. Ein möglicher Grund ist, dass unter Umständen bei größerem technischem Verständnis, die Anzahl der Funktionen als übersichtlicher empfunden wird.

9.1.2.4 Zusammenfassung der Evaluation

Zusammenfassend lässt sich sagen, dass die erste Evaluation, mit einem funktionierenden Prototyp, sehr erfolgreich war. Die meisten Testpersonen kamen nach einer kurzen Einarbeitungszeit mit dem Kompositionswerkzeug und den Service-Frontends sehr gut zu recht und konnten ohne große Probleme die geforderte Beispielanwendung modellieren. Dabei konnte man zwischen dem explorativen Teil und der Erstellung der Beispielanwendung eine starke Lernkurve erkennen. Die meisten Konzepte für die Gestaltung der Frontends wurden von den Probanden erkannt und bereiteten keine Probleme. Allerdings wurde die Darstellung der funktionalen Annotationen von vielen Anwendern nicht benötigt oder deren Funktion war unklar. Für die Erstellung der Beispielanwendung war deren Einsatz nicht notwendig. Das wirft die Frage auf, ob eine Anzeige der funktionalen Annotationen in der Standard-Ansicht wirklich notwendig ist. Unter Umständen ist die Anzeige der erweiterten Funktionen in einen Expertenmodus empfehlenswert. Dies muss sich durch weitere Studien beweisen.

9.2 EVALUATION DER IMPLEMENTIERUNG

In diesem Teil der Arbeit wird die Implementierung des Kompositionswerkzeugs getestet. Dabei soll herausgefunden werden, ob sowohl die optische Darstellung als auch das zugrundeliegende Modell der Frontends korrekt erzeugt werden. Leider waren zum Zeitpunkt der Evaluation die Komponente zur Anlieferung der Webservices und die Komponente zur Serialisierung des Modells nicht fertiggestellt. Diese Komponenten gehören nicht zum Umfang dieser Arbeit oder zum Zuständigkeitsbereich des Autors. Das Fehlen dieser Komponenten hat folgende Auswirkungen auf die Evaluation:

- Die direkte Eingabe realer Services ist nicht möglich.
- Die Ausgabe des Anwendungsmodells in dem vorbestimmten Ausgabeformat ist nicht möglich.

Um die erschwerten Bedingungen zu lösen, wurden die Modelle von 10 Operationen aus 3 Webservices aus einem Webservice-Szenario direkt in das Tool implementiert. Der Umfang dieser Behelfslösung beträgt 4500 Zeilen Programmcode und ist daher schwer zu überblicken. Dennoch wurde dessen Korrektheit mehrfach überprüft, so dass von einem funktionsfähigen Szenario ausgegangen werden kann. Als Ausgleich für die fehlende Serialisierungs-Komponente wurde eine Evaluationsklasse geschrieben, die das Anwendungsmodell der Service-Frontends überprüft. Dabei testet die Klasse die folgenden Aspekte:

- Gesamtzahl der Elemente

- Anzahl der Seiten
- Anzahl der Seiten mit Kindelementen
- Schachtelungstiefe der Interaktionselemente
- Anzahl der Service-Gruppen
- Anzahl der Eingabe-Gruppen
- Anzahl der Ausgabe-Gruppen
- Anzahl der Transitionen
- Struktur der hinzugefügten Frontends
- Namen der komplexen und einfachen Interaktoren
- Anzahl und Typen der verwendeten Eigenschafts-Elemente
- Liste der verwendeten UI-Elemente
- Korrektheit der Seitenerstellung
- Korrektheit der Service-Gruppen-Erstellung
- Korrektheit der Interaktorerstellung

Die Korrektheit der Seitenerstellung ergibt sich dabei wie folgt:

- Es ist ein Anwendungsmodell für die modellierte Applikation vorhanden
- Die Applikation des Anwendungsmodells enthält genau einen Page-Flow
- Die Page hat eine ID
- Eine Page enthält immer ein Wurzelelement. Das Wurzelelement ist immer eine Standard-Group

Die Korrektheit der Service-Gruppen-Erstellung Bedarf der folgenden Bedingungen:

- Die Seitenerstellung ist korrekt
- Die enthält als Kindelemente nur Input- und Outputgruppen
- Die Servicegruppe hat eine ID
- Der Servicegruppe ist ein UI-Element-Typ zugeordnet

Für die Korrektheit der Interaktorerstellung gelten die folgenden Bedingungen

- Die Service-Gruppen-Erstellung ist korrekt
- Es darf nur einen Interaktor pro Servicegruppe geben, der kein zugewiesenes Service-Element hat
- Jeder Interaktor hat eine ID
- Jedem Interaktor ist ein UI-Element-Typ zugeordnet
- Eine Interaktorengruppe darf nur Interaktoren, Interaktorgruppen und Layoutgruppen als Kind-Elemente enthalten

9.2.1 Versuchsdurchführung

Für die Durchführung des Versuchs werden Services aus dem "Sales Szenario" verwendet. Dieses Szenario ist ein Referenz-Szenario des ServFace-Projekts und soll möglichst reale Webservices nachbilden. 10 Operationen aus 3 Services wurden in das Kompositionswerkzeug integriert. Aus dieser Auswahl von 10 Operationen werden einige Operationen für die folgenden Testszenarien verwendet:

1. Eine Operation mit max. Schachtelungstiefe von auf einer Seite
2. Eine Operation mit höherer Schachtelungstiefe bei Eingabeparameteren auf einer Seite
3. Eine Operation mit höherer Schachtelungstiefe bei Ausgabeparameteren auf einer Seiten
4. Zwei beliebige Operationen auf einer Seite und zwei andere Operationen auf einer anderen Seite

Mit den ersten drei Szenarien soll getestet werden ob die Visualisierung der Frontends mit verschiedenen Voraussetzungen korrekt ausgeführt wird. Der letzte Test dient als Überprüfung ob die Modellerstellung auch bei mehreren Seiten und mehreren verwendeten Operationen ordnungsgemäß funktioniert. Vor der eigentlichen Testdurchführung werden die Eckdaten der verwendeten Service-Operationen betrachtet und entsprechende Erwartungen an die Ergebnisse formuliert. Zusätzlich werden Mockups erstellt, die die erwartete visuelle Darstellung widerspiegeln. Die Tests dieser Testreihe überprüfen die Ausführbarkeit der geforderten Funktionalitäten und damit die nichtfunktionalen Anforderungen der Kategorie NFR1: Funktionalität.

Zur Abprüfung der nicht funktionalen Anforderungen der Kategorie NFR2: Zuverlässigkeit, werden alle 10 Operationen auf 7 Seiten im Kompositionswerkzeug verteilt. Anschließend werden die folgenden Aktionen durchgeführt:

1. Bewegung der Frontends
2. Löschen der Frontends
3. Verknüpfen der Frontends und löschen des Quellfrontends
4. Verknüpfen der Frontends und löschen des Zielfrontends
5. Umbenennung der Elemente

Bei diesem Test dürfen keine Fehler auftreten. Der Test wird fünf Mal wiederholt. Zusätzlich wurde diese Anforderung bereits in der in Abschnitt 9.1.2 beschriebenen Nutzerstudie mit überprüft.

Die Überprüfung der nichtfunktionalen Anforderung NFR3: Benutzbarkeit wurde ebenfalls zum großen Teil schon während der unter Punkt 9.1.2 durchgeführten Evaluation überprüft. Zusätzlich wird die Nebenbedingung NB1: Effizienz während der Tests für NFR2: Zuverlässigkeit mit verifiziert. Dabei wird beim Hinzufügen einer Service-Operation die Zeit bis zum Erscheinen dieser Operation gemessen. Außerdem wird im Rahmen des NFR2: Zuverlässigkeitstest die Performanz der Anwendung subjektiv bewertet.

Die Anforderung NFR5: Änderbarkeit wurde im Rahmen der Konzeption abgehandelt und ergibt sich aus der Beschreibung der Konzepte. Ein direkter Test wie gut eine Erweiterbarkeit des Kompositionswerkzeugs möglich ist, kann im Rahmen dieser Evaluation nicht durchgeführt werden.

Als Testsystem dient ein System mit den folgenden Eckdaten:

- Intel Core 2 Duo 2.4 GHz
- 4GB RAM
- Windows Vista SP1
- Mozilla Firefox 3.5.4
- Silverlight 3

9.2.2 Beschreibung der Erwartungen

In diesem Abschnitt werden die verwendeten Testoperationen beschrieben und die erwarteten Ergebnisse aufgeführt. Dazu wird zunächst eine Tabelle mit den Kennzahlen der Operationen aufgestellt. Anschließend werden aus diesen Kennzahlen die erwarteten Parameter der Frontend-Datenstruktur abgeleitet. Für den Funktionalitätstest werden für die einzelnen Aufgaben die folgenden Service-Operationen verwendet:

1. Service: Account Management - Operation: Update Customer
2. Service: Product Management - Operation: Update Product
3. Service: Customer Order Management - Operation: Get Customer Orders
4. Alle drei oben genannten Service + Service: Account Management - Operation: Create new Customer

Für die genannten Operationen wurden die folgenden Kennzahlen identifiziert:

	Update Customer	**Update Product**	**Get Customer Orders**	**Create Customer**
Anzahl Elemente	11	13	39	6
Schachtelungstiefe	2	3	6	0
Anzahl komplexer Parameter	1(Input)	1(Input)	1(Output)	0
Anzahl simpler Parameter	1(Output)	1(Output)	1(Input)	4
Anzahl simpler Unterelemente	7(Input)	8(Input)	30(Output)	0
Anzahl komplexer Unterelemente	0	1(Input)	6(Output)	0
Ein-/Ausgabe?	Ja/Ja	Ja/Ja	Ja/Ja	Ja/Ja
Datentypen	string, long, boolean,	string, long, double, boolean, dateTime, int	dateTime, string, long, boolean, double	string, boolean, long

Tabelle 9.8: Kennzahlen der Service-Operationen

Zusätzlich waren die Services mit den folgenden Annotationen versehen:

- TextFeedback.Label
- TextFeedback.Contextual Help
- Mandatory
- Validation
- Suggestion

Aus den genannten Kennzahlen ergeben sich die folgenden Erwartungen für die Ergebnisse.

	Update Customer	**Update Product**	**Get Customer Orders**	**Kombination**
Eingabeelemente Input	7+1	8+1	1+1	19+4
Ausgabelemente Output	1	1	30	33
Komplexe Gruppen Input	1	2	0	3
Komplexe Gruppen Output	0	0	7	8
Ein-/Ausgabegruppen	1/1	1/1	1/1	4/4
Servicegruppen	1	1	1	4
UI-Elemente	Textbox, Stepper, Checkbox, Button	Textbox, Stepper, Kalendar, Checkbox, Button	**AdvancedOutput**, Text, steppern, Kalendar, Checkbox, Button	siehe andere Spalten

Tabelle 9.9: Erwartungswerte Service-Operationen - Eingabeelemente +1 für Submit-Button

Für die erwartete Darstellung der Service-Frontends wurden die folgenden Mockups erstellt:
Update Customer & Update Product:

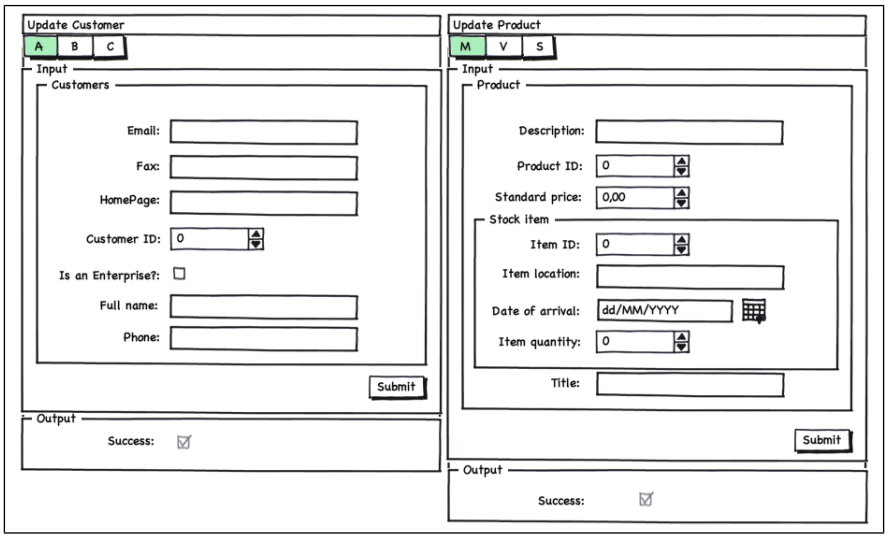


Abbildung 9.3: Mockup der Update Customer und Update Produkt Operationen

Get Customer Orders & Create Customer:

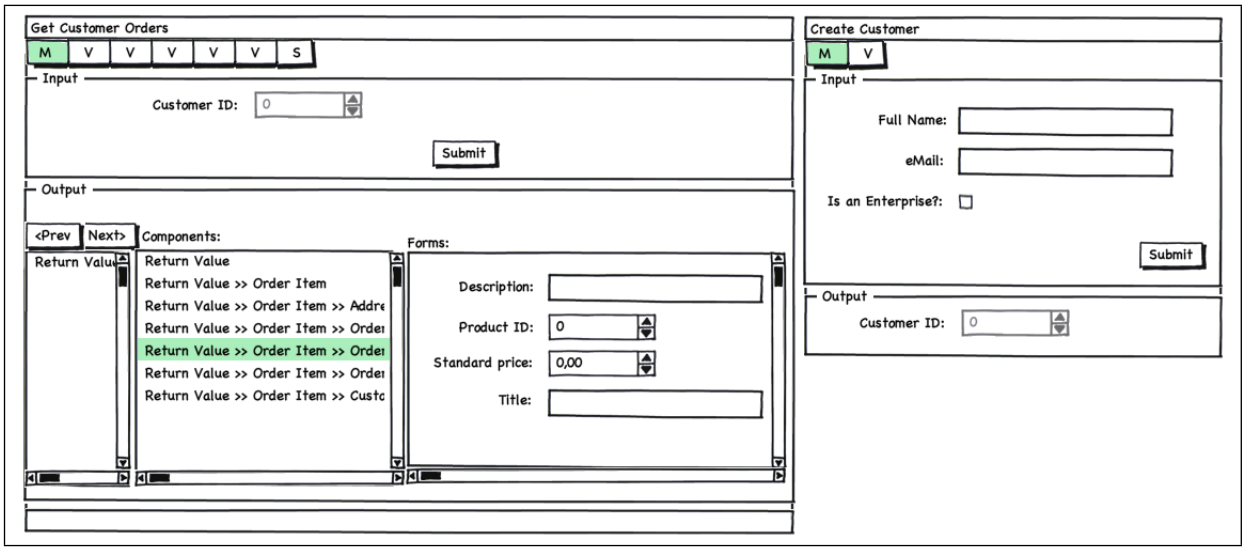


Abbildung 9.4: Mockup der Get Customer Orders und Create Customer Operationen

Das Mockup für den vierten Test setzt sich aus den Mockups der vorangegangenen Services zusammen.

9.2.3 Ergebnisse

Im nun folgenden Abschnitt werden die Ergebnisse der Tests dargestellt und interpretiert.

	Update Customer	**Update Product**	**Get Customer Orders**	**Kombination**
Seitenerstellung	korrekt	korrekt	korrekt	korrek
Service-Gruppen-Erstellung	korrekt	korrekt	korrekt	korrekt
Interaktorenerstellung	korrekt	korrekt	korrekt	korrekt
Komplexe Interaktoren	1	2	7	10
Simple Interaktoren	9	10	32	56
Interaktorenstruktur	korrekt	korrekt	korrekt	korrekt
Schachtelungstiefe	2	3	6	6
Ein-/Ausgabegruppen	1/1	1/1	1/1	4/4
Servicegruppen	1	1	1	4
UI-Elemente	vBox, Textbox, Stepper, CheckBox, Button	vBox, Textbox, Stepper, Kalendar, Button, Checkbox	**AdvancedOutput**, Stepper, Button, Kalendar, Text Checkbox	siehe andere Spalten
Anzahl Eigenschaften	28	33	97	175
Verwendete Eigenschaften	Label, Left, Top, Help	Label, Left, Top, Help	Label, Left, Top, Help, Appearance	Label, Left, Top, Help, Appearance
Dauer des Hinzufügens	sofort	sofort	3,5 sec	sofort

Tabelle 9.10: Ergebnisse Modell-Test

Die Tabelle 9.10 stellt eine Auswahl der Testergebnisse dar. Die vollständigen Ergebnisse sind auf dem Datenträger zu finden, der dieser Arbeit beiliegt.

Zum Vergleich mit den Mockups sind im folgenden die Screenshots der Frontends dargestellt:

Update Customer & Update Product:

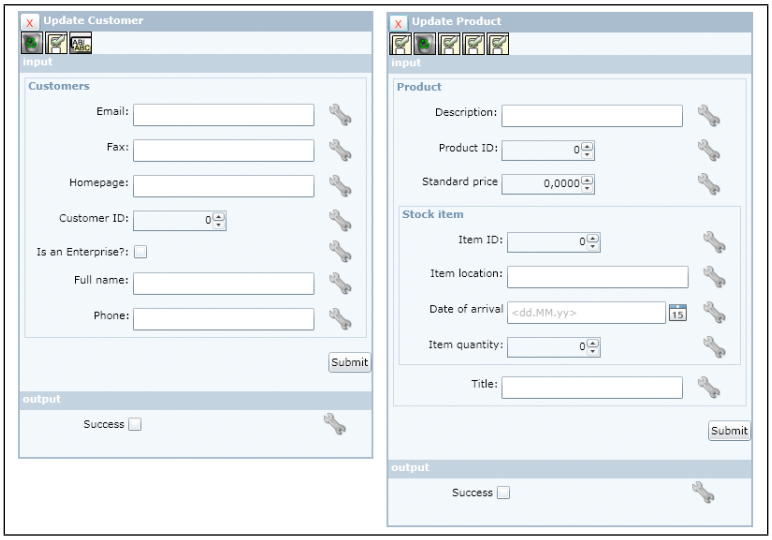


Abbildung 9.5: Screenshot der Update Customer und Update Product Operation

Get Customer Orders & Create Customer:

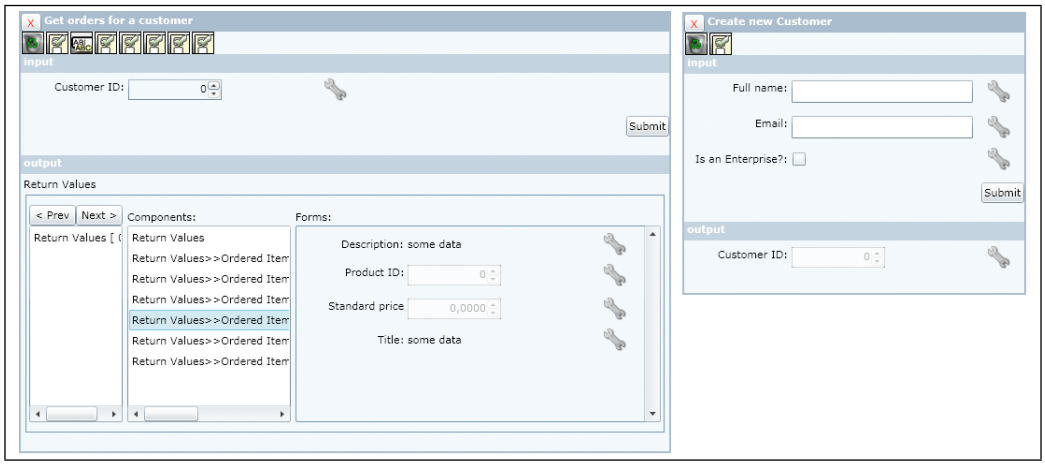


Abbildung 9.6: Screenshot der Get Customer Orders und Create Customer Operation

Sowohl die visuellen als auch die formellen Ergebnisse zeigen, dass die Erwartungen vollständig erfüllt wurden. Das erzeugte Modell war bei allen 4 Szenarien gültig. Außerdem wies sowohl die Gruppenstruktur als auch die Interaktorenstruktur einen korrekten Aufbau auf. Weiterhin ist zu erkennen, dass sich die vermuteten UI-Elemente in den erzeugten Frontends wieder finden. Auch die Kombination mehrerer Service-Frontends brachte keine Probleme bei der Visualisierung mit sich. Mit diesem Test konnte die Funktionstücktigkeit der Visualisierungskomponenten des Controllers nachgewiesen werden. Sowohl Modell-Instanz als auch Visualisierung sind als korrekt einzustufen.

Im zweiten Teil des Testes geht es um die Zuverlässigkeit der Anwendung. Zusätzlich wurden Noten für die Performanz nach dem Schulnotenprinzip vergeben. Die Bewertung der Performanz stellt einen subjektiven Eindruck dar. Die folgenden Ergebnisse wurden bei diesem Test erzielt:

	Test 1	Test 2	Test 3	Test 4	Test 5
Bewegung der Frontends	ok - 3	ok -3	ok - 3	ok - 3	ok -3
Löschen der Frontends	ok - 1	ok - 1	ok - 2	ok - 2	ok - 1
Verknüpfung 1	ok - 2	ok - 2	ok - 3	ok - 2	ok - 2
Verknüpfung 2	ok - 2	ok - 2	ok - 3	ok - 2	ok - 2
Umbenennung	ok - 1	ok - 1	ok - 1	ok - 1	ok - 1

Tabelle 9.11: Ergebnisse Zuverlässigkeitstest

Zusätzlich wurde noch die Zeit gemessen, die das Kompositionswerkzeug zum Anzeigen eines Frontends benötigt. Dazu wurde auf eine einzelne Seite eine Service-Operation gezogen und mit einer implementierten Zeitmessung die Dauer bis zum Erscheinen bestimmt. Nach jedem Testdurchlauf wurde die Anwendung neu gestartet. Der Test wurde mit einer einfachen (Update Customer) und einer umfangreichen (Get Customer Orders) Service-Operation durchgeführt. Die Ergebnisse sind in Tabelle 9.12 aufgeführt.

	Test 1	Test 2	Test 3	Test 4	Test 5	Schnitt
Update Customer	0.141	0.109	0.156	0.140	0.141	**0.137**
Get Customer Orders	1.482	1.404	1.545	1.435	1.467	**1.466**

Tabelle 9.12: Ergebnisse Zeitmessung in Sekunden

Alle Testläufe des zweiten Tests waren erfolgreich. Es traten keine Fehler bei diesen Tests auf. Das spricht für die Zuverlässigkeit der Anwendung. Allerdings ist für eine detaillierte Zuverlässigkeitsaussage eine Langzeitstudie mit mehreren Nutzern notwendig. Diese konnte im Rahmen der Arbeit nicht durchgeführt werden. Dennoch stellt dieser Test ein Indiz für die Zuverlässigkeit der Anwendung dar. Bei 10 Operationen, die auf 7 geöffnete Seiten verteilt sind, zeigen sich dennoch leichte Performanz-Schwächen. Das Bewegen der Elemente läuft in einigen Fällen leicht ruckend und nicht so flüssig ab wie bei 3 oder 4 Operationen. Zusätzlich steigt die Verarbeitungsdauer bei umfangreichen Operationen stark an. Verwendbar ist die Anwendung aber trotz dieser leichten Performanzprobleme ohne Einschränkungen.

9.3 DISKUSSION UND EMPFEHLUNGEN

Im Rahmen der Evaluation wurden eine Nutzerstudie und eine technische Evaluation durchgeführt. Die Nutzerstudie hat wertvolles Feedback zur Nutzbarkeit des Kompositionstools und der Frontends geliefert. Bis auf eine Ausnahme wurden die Konzepte der Service-Frontends angenommen und mit gutem Ergebnis durch die Probanden bewertet. Sowohl die Verständlichkeit, als auch die Übersichtlichkeit und der Funktionsumfang der Frontends wurden problemlos angenommen. Das spricht für eine Weiterverwendung und Weiterentwicklung der bisher eingesetzen Konzepte. Die Darstellung der funktionalen Annotationen und weiteren Zusatzfeatures wurden von den Testpersonen kaum beachtet. Das lässt die Empfehlung zu, diese Funktionen und Anzeigen in einen Expertenmodus umzulagern, den fortgeschrittene Nutzer verwenden können. Für den Einstieg und die Erstellung einfacher interaktiver Anwendungen sind diese Funktionen nicht notwendig. Insgesamt ergeben sich die folgenden Empfehlungen aus der Nutzerstudie die Frontends betreffend:

- Positionierung der Bausteine durch Raster vereinfachen. Erstes Bewegen auf die Arbeitsfläche verursacht einrasten an sinnvoller Position.
- Es ist zu überlegen, ob man Funktionale Annotationen, Umbenennungen und Seitenverknüpfungen in einem zusätzlichen Expertenmodus anbietet.
- Entwicklung einer Lösung zur besseren Vermittlung des Aussehens der Endanwendung.
- Die Design-Empfehlungen werden von den Nutzern grundsätzlich angenommen und können entsprechend weiter umgesetzt werden.

In diesem Teil der Evaluation konnten auch die nichtfunktionalen Anforderung nach Benutzbarkeit und Zuverlässigkeit mit abgeprüft werden. Während der Evaluation ergaben sich keine Probleme in den genannten Bereichen. Während der Evaluation traten nur marginale Fehler in der Software auf, die schnell beseitigt werden konnten. Große Probleme in den genannten Aspekten traten während der Nutzerstudie nicht auf.

Der zweite Teil der Evaluation war die technische Überprüfung der Visualisierungskomponenten. Dabei wurden mehrere Testszenarien mit unterschiedlich umfangreichen Service-Operationen überprüft. In allen Fällen waren die Modelle gültig und die Visualisierung entsprach den Erwartungen. Dementsprechend ist die korrekte Funktionstüchtigkeit der Visualisierungskomponente nachgewiesen. Sobald die fehlenden Komponenten der Service-Anlieferung und Serialisierung integriert wurden, sind weitere umfassende Tests mit diesen Komponenten und tatsächlich realen Szenarios geplant. Während der Effiziens und Zuverlässigkeitstest wurde festgestellt, dass das Hinzufügen von umfangreichen Services zeitlich einen etwas längeren Zeitraum in Anspruch nimmt. Der Durchschnitt lag bei rund 1,5 Sekunden. Die Dauer liegt noch im annehmbaren Bereich, dennoch sind Optimierungen möglich. Auch die Bewegung der Elemente bei großen Belastungen verursacht leichtes stocken, was allerdings keine Einschränkung der Funktionstücktigkeit bedeutet.

Insgesamt kann man die durchgeführte Evaluation sehr positiv sehen, da sowohl das Feedback der Nutzer als auch die Ergebnisse der technischen Evaluation auf gute Konzepte und eine funktionierende Umsetzung schließen lassen.

10 ZUSAMMENFASSUNG

In der vorliegenden Diplomarbeit wurde die "Visualisierung von Service-Frontends in einem Werkzeug zur präsentationsorientierten Komposition annotierter Dienste" untersucht. Im Folgenden soll die Arbeit kurz zusammengefasst und ein Ausblick für künftige Arbeiten gegeben werden.

10.1 ZUSAMMENFASSUNG DER DIPLOMARBEIT

Im Rahmen des Kompositionsprozesses von Webservices ist die Visualisierung von Service-Frontends ein wichtiger Schritt. Im Rahmen dieser Arbeit wurden Möglichkeiten untersucht präsentationsorientierte Frontends zu gestalten, diese zu visualisieren und eine passende Datenstruktur zu generieren. Zur Einarbeitung in das Thema wurden zunächst verschiedene Ansätze der UI-Generierung untersucht. Unter anderem wurden Techniken aus dem Bereich der CRUD-Anwendungen für Datenbanken und das Object-Relation-Mapping betrachtet. Anschließend wurden Vorgängerprojekte wie WSGUI und Dynvoker untersucht und eine Abgrenzung zu diesen Projekten vorgenommen.

Schon während der State-of-the-Art-Analyse hat sich gezeigt, dass die Erzeugung von benutzerfreundlichen Frontends eine umfassende Aufgabe ist. Um die Nutzbarkeit der im Rahmen dieser Arbeit visualisierten Frontends zu garantieren, wurden zunächst Gestaltungsrichtlinien von namenhaften Usability-Experten und Software-Konzernen betrachtet. Die Konsolidierung dieser Richtlinien führte zu Design-Empfehlungen, die die Visualisierung der Frontends stark beeinflusst haben. Für den anschließenden Entwurf der Frontends wurden die Design-Empfehlungen in den Gestaltungsprozess mit einbezogen.

Für die Entwicklung des Designs war eine umfassende Betrachtung der Darstellungsmöglichkeiten für Parameter notwendig. Dabei stellte sich heraus, dass die Darstellung nutzbarer Frontends stark von der Nutzung von Annotationen oder Datentyp-Beschränkung abhängt. Mit deren Hilfe lassen sich UI-Elemente verwenden, die dem Nutzer eine einfache Eingabe der geforderten Daten ermöglichen. Die Darstellung umfangreicher komplexer Parameter warf allerdings ein Problem auf. Die Darstellung im Rahmen der herkömmlichen Konzepte war gerade für die Datenausgabe nicht möglich. Besonders die Forderung jedes Interaktionselement für eine Datenverbindung auswählen zu können bereitete Probleme. Gelöst wurde das Problem durch den Entwurf eines alternativen Frontends zur Darstellung eben dieser umfangreichen Datenstrukturen.

Schon während der Konzeption der Frontend-Darstellung wurden immer wieder die Verbesserungsmöglichkeiten durch die Nutzung von Annotationen ersichtlich. Um die Umsetzung der Annotationen zu beschreiben, wurde zunächst eine Kategorisierung dieser Annotationen vorgenommen. Dabei ergaben sich die Gruppen der UI-Annotationen, funktionalen Annotationen und Hybrid-Annotationen. Während sich UI- und Hybrid-Annotationen direkt auf das Datenmodell auswirken, können die funktionalen Annotationen nur metaphorisch dargestellt werden. Die Ausführung der Funktionen dieser Annotationen ist zur Design-Time noch nicht möglich. Für die metaphorische Darstellung der funktionalen Annotationen wurden Icons entwickelt, die in die Service-Frontends integriert wurden.

Für die eigentliche Umsetzung der Visualisierung wurden verschiedene Komponenten konzipiert. Dabei galt die Anforderung die Auswertung der Annotationen möglichst einfach zu gestalten und die Integration weiterer Plattformen zu ermöglichen. Die Basisvisualisierung wird von der Kernkomponente übernommen, die durch die Annotations-Effekt-Bestimmung- und die Regelset-Komponente unterstützt wird. Die Annotations-Effekt-Bestimmung untersucht die angehängten Annotationen und wertet deren Auswirkungen auf die Frontends aus. Dabei greift die Komponente auf ein erweiterbares Repository von Beschreibungsklassen zurück. Die Regelset-Komponente nutzt XML-Plattformbeschreibungen um eine plattformabhängige Bestimmung des richtigen UI-Elements zur Repräsentation eines Parameters durchzuführen. Das in dieser Arbeit vorgestellte Visualisierungskonzept stellt eine erweiterbare Möglichkeit dar eine Datenstruktur für Service-Frontends zu erstellen, in das Anwendungsmodell zu integrieren und dem Nutzer zu präsentieren.

Die Visualisierung nutzt hierfür neben einer modellhaften Service-Repräsentation die angefügten Annotationen und die aufgestellten Design-Empfehlung zur Verbesserung des Visualisierungsergebnisses.

Die Visualsierungskomponenten wurden entsprechend implementiert und in das Kompositionswerkzeug eingefügt. Zum Nachweis der Richtigkeit der Konzepte und Implementierung wurde eine Nutzerstudie und eine technische Evaluierung durchgeführt. Die technische Evaluierung lieferte die erwarteten Ergebnisse, was die Funktionstüchtigkeit der Implementierung unterstreicht. Durch die Nutzerstudie wurden die Darstellungsentscheidungen abgeprüft. Die meisten Design-Konzepte wurden von den Testpersonen angenommen und sehr gut bewertet. Die metaphorische Darstellung der funktionalen Annotationen wurde allerdings nur von weniger Testern verwendet. Das spricht für eine Veränderung dieses Darstellungs-Konzepts zum Beispiel durch das Hinzufügen eines Expertenmodus.

Im Allgemeinen lässt sich festhalten, dass die Anforderungen im Rahmen dieser Arbeit erfüllt wurden und die Visualisierungskomponenten als Teil des Kompositionswerkzeuges in die weiteren Untersuchungen des ServFace-Projektes einfließen.

10.2 AUSBLICK UND ZUKÜNFTIGE ARBEITEN

Die Visualisierungskomponenten setzen die Möglichkeit um, eine Datenstruktur für die Darstellung von Service-Frontends zu erzeugen und in das Anwendungsmodell zu integrieren. Der aktuelle Stand des Tools unterliegt der ständigen Weiterentwicklung. Die nächsten Schritte in der Entwicklung des Kompositionswerkzeugs sind die Integration der Service-Repository-Komponente und der Serialisierungskomponente. Nach dem Hinzufügen dieser Komponenten ist das Durchlaufen der ersten Abschnitte der Werkzeug-Kette umfassend zu testen. Das umfasst:

- Einlesen von Service-Beschreibungen
- Annotation von Services
- Abrufen der Servicemodelle und Umwandeln in das Servicemodell des Kompositionswerkzeugs
- Visualisierung der Frontends und Durchführung des Kompositionsprozesses
- Ausgabe eines gültigen Modells und Speicherung in einem geeigneten Austauschformat
- Einlesen des Modells in der weiterführenden Komponente

Weitere Nutzerstudien sind ebenfalls Teil der aktuellen Planung um die Nutzbarkeit und Bedienerfreundlichkeit des Kompositionswerkzeuges sicherzustellen.

Neben den geplanten Aktivitäten gibt es auch Möglichkeiten die dargelegten Konzepte zu erweitern. Die Erzeugung der Datenstruktur wird durch die Auswertung von Annotationen verbessert. Eine mögliche Erweiterung der Annotationsauswertung wäre die Umsetzung der Annotationen in einem generischen Modell. Durch dieses wäre eine elegantere, einfachere und besser erweiterbare Auswertung der Annotationen möglich.

Auch die plattformspezifische Bestimmung der UI-Elemente und die geplante Darstellungsmöglichkeit für verschiedene Plattformen im Kompositionstool lassen sich erweitern. Eine mögliche Erweiterung des zuvor genannten Konzepts ist die dynamische Umschaltung der Anzeige zwischen verschiedenen Ansichten. Der Nutzer hat damit die Möglichkeit eine Anwendung simultan für mehrere Plattformen zu modellieren.

Die Schaffung eines Expertenmodus kann den Einstieg in das Tool verbessern. Dazu werden alle Funktionen ausgeblendet, die von einem Werkzeugeinsteiger zur Schaffung einfacher Anwendungen nicht benötigt werden. Nutzer die sich schon länger mit der Kompositionswerkzeug beschäftigen und umfassendere Anwendungen bauen möchten, können die erweiterten Funktionen jederzeit wieder aktivieren.

Zu guter Letzt lässt sich vor allem das Event-System der View-Komponente optimieren, um die Performanz des Werkzeugs weiter zu erhöhen.

Zusammengefasst ergeben sich die folgenen Empfehlungen für das ServFace Projekt:

- Erstellen eines generischen Annotations-Modells.
- Überarbeiten des Eventsystems des Kompositionswerkzeugs.
- Erweitern der View-Komponente des Kompositionswerkzeugs um eine plattformspezifische Ansicht.
- Einarbeiten der Empfehlungen aus der Nutzerevaluation.

Teil IV

Anhang

A ANHANG

A.1 PERSONAS ZUR GRUPPE DER ENDANWENDER

Die nachfolgenden fünf Personas wurden im Rahmen des ServFace-Projekts entwickelt. Sie sollen dabei helfen, ein möglichst genaues Bild der Zielgruppe zu entwickeln.

IT Manager

Nathalie ist eine 35 jährige IT-Managerin eines mittelständischen Betriebes. Nach ihrem Studium der Wirtschaftsinformatik an der Technischen Universität Dresden sammelte sie erste Berufserfahrung als IT Beraterin in einigen Projekten im Banken- und Automobilumfeld. Diese Erfahrungen festigten ihre IT Kenntnisse und ihr Verständnis für den Ablauf innerhalb der Softwareentwicklung. Ihr besonderes Interesse galt den Themen Prozess-Integration und Usability.

Nathalie lässt sich als „early adopter" bezeichnen. Sie experimentiert gern mit neuen Technologien und kann sich leicht in neue Sachen einarbeiten. Nach einigen Jahren beschloss sie aus dem Beratungsumfeld auszusteigen und übernahm einen Job als IT-Managerin bei ihrem heutigen Arbeitgeber. Die Firma umfasst 50 Mitarbeiter und hat einen Großteil der Kunden im Bankenumfeld. Nathalie ist dabei für das IT Team der Firma zuständig, welches aus 5 Personen besteht. Ihre Aufgaben erstrecken sich von der Verwaltung der internen Systeme (Homepage und Firmennetzwerk) über die Erstellung von speziellen Softwarelösungen für einzelne Kunden sowie Kollegen innerhalb der Firma. Ihre Arbeit empfindet Nathalie als spannend und abwechslungsreich, aber auch sehr zeitintensiv.

Viele der geforderten Softwarelösungen beinhalten immer wieder ähnliche Anforderungen, wie beispielsweise einfacher Zugriff und Verwaltung von Daten sowie die Erstellung von kleinen Webportalen. Sie müssen innerhalb kürzester Zeit den Projekten zur Verfügung stehen und werden meist für den internen Gebrauch benötigt. Die dafür nötigen Funktionalitäten sind bereits als Web Services verfügbar, müssen aber noch in einem aufwändigen Entwicklungsprozess eingebunden werden. Diese Aufgabe ist zeit- und kostenintensiv und führt oft erst nach mehreren Iterationen zu den gewünschten Zielen. Viele der Anwendungen sind meist auch speziell auf ein bestimmtes Projekt ausgerichtet und können nicht wiederverwendet werden.

Mittels des Kompositionswerkzeuges kann Nathalie nun die meisten der gewünschten Anwendungen auf einfache Weise zusammenbauen. Ein erstes Mashup ist schnell erstellt und kann durch einfache Konfigurationen angepasst werden. Das Tool ermöglicht es Nathalie schnell auf Wünsche zu reagieren und erspart ihr viel Zeit aufgrund der hohen Wiederverwendbarkeit der einzelnen Services. Bereits nach kurzer Zeit konnte Nathalie erste lauffähige Anwendungen erstellen und diese auch zur Zufriedenheit der jeweiligen Auftraggeber fertigstellen. Für komplexe und flexible Softwarelösungen nutzt sie weiterhin die Entwickler ihres Teams, da die Stärken der Plattform in Bezug auf die einfache, visuelle Entwicklung von Standardanwendungen hier nicht zum Tragen kommen würden. Insgesamt ist sie mit den ersten Eindrücken sehr zufrieden und empfiehlt es sogar ihren Kollegen von anderen Abteilungen.

Vertriebsmitarbeiter

Bernd ist 25 Jahre alt und hat vor seiner Anstellung als Vertriebs- und Marketingfachmann fünf Jahre BWL studiert. Eines seiner liebsten Arbeitswerkzeuge ist MS Excel, da es ihm hilft, seine Arbeit zu automatisieren und Rechenarbeit einzusparen. Er hat deshalb, während seines Studiums, ein Semester lang einen Kurs „VBA-Grundlagen mit MS Excel" besucht und mit der Note 3,0 abgeschlossen. Seitdem weiß er, dass er nicht gerne Quelltext programmiert. Ansonsten ist

Bernd aber sehr an neuer Technik und an IT-Unterstützung interessiert. Er nutzt Online-Banking und Online-Shopping und besitzt ein iPhone. Er recherchiert zu verschiedenen Themen in Foren und Blogs sowie via Google-Suche. Außerdem ist er in den Online-Communities StudiVZ und Facebook aktiv und pflegt dort auch regelmäßig seine persönlichen Präsentationsseiten und Kontaktlisten. Als Startseite hat er iGoogle eingestellt, auf der er verschiedene News-Feeds und anderer Gadgets abonniert und eingerichtet hat.

Bernd ist in der Bedienung von Office-Anwendungen insbesondere MS Excel sehr geübt und kann auch komplexe Funktionen dieser Werkzeuge nutzen. Erste Berührungen mit der Programmierung haben ihn nicht bestärkt sich weiter in diese Richtung zu orientieren. Sein Verständnis von Sachverhalten der Programmierung ist also sehr rudimentär. Er weiß allerdings, dass Programme sich aus einzelnen Funktionen zusammensetzen oder dass Daten meistens in Datenbanken abgespeichert werden. Auch die Verwendung von Variablen und Datentypen ist im geläufig.

Bernd arbeitet in einem mittelständischen Betrieb, der Gartenbewässerungsanlagen herstellt und vertreibt. Der Betrieb hat rund 200 Mitarbeiter. Die IT-Abteilung besteht aus 6 Personen, die mit ihren Serviceaufgaben weitestgehend ausgelastet sind. Die Firma möchte ihre Vertriebsmöglichkeiten erweitern und hat sich entschieden einen eigenen Online-Shop einzurichten. Um die IT-Abteilung zu entlasten wurde beschlossen, dass Bernd die wesentliche Entwicklungsarbeit übernimmt und von der IT-Abteilung, wenn nötig, entsprechende Zuarbeit erhält.

Bernd nutzt zur Erstellung des Online-Shops erstmals das Kompositionswerkzeug. Es stellt Bernd über Wizards grundlegende Funktionalität zur Verfügung. So werden User- und Account-Management, was Login-Funktionalität und Rechteverwaltung beinhaltet, sowie Navigation ohne großen Aufwand in die Anwendung integriert. Per Drag&Drop zieht er verschiedene UI-Elemente auf die Arbeitsfläche und erzeugt so mit Hilfe von etwas Konfiguration eine Registrierungsseite.

Kundenbetreuer

Martin ist 41 Jahre alt und ist als Kundenbetreuer tätig. Er war in verschiedenen Firmen in den Bereichen Vertrieb und Kundenbetreuung zuständig und kann über acht Jahre Berufserfahrung in seinem Tätigkeitsfeld vorweisen. Sein jetziger Arbeitgeber ist ein Londoner Unternehmen, das PC-Hardware vertreibt und Serviceleistungen für die Integration der Hardwarekomponenten und Support anbietet. Das Unternehmen umfasst rund 90 Mitarbeiter und verfügt über eine gut ausgebaute IT-Infrastruktur und branchenbedingt auch über entsprechendes Know-How in der IT-Abteilung.

Martins Haupttätigkeit liegt im Customer- und Sales-Management der Firma, d. h. er stellt Kontakt zu (potentiellen) Kunden her und steht diesen dann als Ansprechpartner zur Verfügung. Er verantwortet damit den Vertriebsprozess von der Bearbeitung von Kundenanfragen bis hin zur Erstellung von Kostenvoranschlägen und Bestellungen. Martins Aufgaben werden durch Customer Relationship Management (CRM) Systeme sowie Software der MS Office Produktreihe unterstützt. Durch die ständige Arbeit mit diesen Tools ist er in ihrer Bedienung sehr versiert.

Privat nutzt Martin das Internet, um sich über die verschiedensten Themen zu informieren. Außerdem pflegt er eine kleine einfache Internetseite, mit weitgehend statischen Inhalten und einer Fotogalerie. Da seine HTML-Kenntnisse nicht sehr ausgeprägt sind, kopiert er sich Quellcode aus anderen Seiten, passt diesen an und lässt er sich von Bekannten helfen, wenn größere Probleme auftreten. Auch mit MS Frontpage hat er bereits experimentiert.

Martins Arbeitgeber hat im letzten Jahr ein neues firmenweites Softwaresystem eingeführt, mit dem er nun arbeiten wird. Das Unternehmen sieht in der serviceorientieren IT-Architektur viele Vorteile und hat im Vertrauen auf diese Technologie die neue Firmensoftware auf der Basis von SOA aufgebaut. Sie ist damit leichter an die Geschäftsprozesse des Unternehmens anpassbar. Die IT Abteilung hat in diesem Zusammenhang eine Plattform für die Mitarbeiter bereitgestellt

mit der sich einzelne Zusatzapplikationen bauen bzw. existierende Anwendungen verändern lassen. Das Customer- und Sales-Management-Modul mit dem Martin arbeitet, ist sehr umfangreich gestaltet und bietet noch zu viele Funktionen die er nicht oder nur in Ausnahmefällen benötigt. Andere Funktionen wiederum fehlen. Auch die Menüstruktur ist nicht optimal auf Martins täglichen Arbeitsalltag abgestimmt. Mit Hilfe des entwickelten Kompositionswerkzeugs hat Martin die Möglichkeit, sich die Software seinen konkreten Arbeitsbedingungen anzupassen. Nicht benötigte Funktionen können leicht entfernt werden und neue Anforderungen lassen sich sehr leicht implementieren. Die Software lässt sich so viel intuitiver bedienen und ist sehr benutzerfreundlich.

Integrationsexperte

Frank ist 28 Jahre alt und hat nach seinem Maschinenbaustudium und verschiedenen Praktika eine Stelle in der metallverarbeitenden Industrie übernommen.

Er arbeitet für einen größeren Mittelständler Deutschlands und wird an verschiedenen Standorten des Unternehmens eingesetzt. Das Unternehmen stellt mechanische Ersatzteile für verschiedene Maschinen her. Dabei handelt es sich überwiegend um Einzelanfertigungen, die nach Vorgaben des Kunden gefertigt werden. Franks Hauptaufgabe ist der Aufbau neuer Produktionsstraßen oder die Anpassung der Produktionsprozesse an neue Produktionsanforderungen. Sein Arbeitgeber hat bereits an mehreren der Standorte neue hochmoderne Produktionsstraßen errichten lassen. Mit Hilfe von RFID-Chips können einzelnen Produkte und Materialien identifiziert und lokalisiert werden, sodass sich der Produktionsprozess an vielen Stellen weiter automatisieren lässt. An Stellen, wo Werker in den Produktionsprozess eingreifen, werden mittlerweile Touchscreens eingesetzt um die Interaktion mit dem System zu verbessern und den Werkern eine bessere Anleitung für ihre Arbeitsschritte zu geben. Einzelne Geräte (Messstationen, RFID Lesegeräte, Ampeln) sowie die involvierte Unternehmenssoftware (Enterprise Resource Planing (ERP) oder Manufacturing Execution System (MES)) sind über Web Services ansteuerbar. Für die individuellen Produktionsprozesse müssen die Geräte angepasst und für die Terminals entsprechende Nutzerschnittstellen erstellt werden. Für die Erstellung der Benutzeroberflächen ist die Zusammenarbeit von IT-Experten und Ingenieuren, die sich mit dem Produktionsprozess bestens auskennen, so wie Frank, erforderlich. Frank berät die IT-Experten und erhebt mit ihnen die Anforderungen der Software. Nach mehreren Iterationen ist die Software erstellt und die Produktionsstraße ist einsatzbereit.

Frank kann als aktiver Internetnutzer bezeichnet werden, wobei bei ihm vor allem der kommunikative Aspekt im Vordergrund steht. So nutzt er verschiedene Messenger und Community-Portale, um mit Freunden und Interessensgenossen in Kontakt zu treten. Auch erledigt eine viele Einkäufe sowie seine Bankgeschäfte über das Internet. Er hat ein iPhone und führt gerne seinen Blog über unternommene Reisen. Bei seiner Arbeit verwendet er häufig MS Word, MS Excel, ein ERP-System sowie das interne Firmenportal. Für die Arbeit mit dem MES System ist er auch mit der Nutzung von Datenbanken und der Anbindung von Produktionsplanung und Steuerungs- (PPS) Modulen vertraut. Er ist in der Nutzung dieser Programme sehr geübt und wenn er einmal Hilfe bei der Bedienung der Software braucht, recherchiert er im Internet danach. So nutzt Frank auch die Makrofunktionen der MS Office-Produkte, um Makros zu erstellen und sie dann mit der integrierten VBA-Entwicklungsumgebung anzupassen, bis sie die entsprechende Aufgabe erfüllen.

Mittels des Kompositionswerkzeuges kann Frank die Erstellung der Nutzerschnittstelle (Worker UI) und die Ansteuerung der Maschinen nun selbst vornehmen, was sehr viel Zeit und Kosten spart. Die IT-Abteilung übernimmt weiterhin die Bereitstellung der direkten Schnittstellen zu den Geräten, den Rest erledigen Frank und seine Ingenieure. Die Software lässt sich sehr einfach bedienen und bietet viele praktische Hilfestellungen, so sind die Nutzerschnittstellen schnell erstellt. Auch sind die Benutzeroberflächen durch das direkte Mitwirken der Ingenieure nun sehr verständlich und können bei veränderten Anforderungen ganz einfach direkt durch die Produktion angepasst werden.

Team-Assistenz

Annette ist eine Team-Assistenz für eine Forschungseinrichtung mit 80 Personen. Sie ist aufgrund ihrer Tätigkeit eine sehr geübte MS Office Nutzerin. Sie ist es gewohnt sich die Tools, die sie benutzt, so gut es geht an ihre Bedürfnisse anzupassen. Eine ihrer Aufgaben umfasst die Reiseplanung für die Mitarbeiter der Forschungseinrichtung. Dies ist eine sehr zeitintensive Aufgabe, zumal die einzelnen Mitarbeiter häufig auf Konferenzen oder Projekttreffen in der ganzen Welt unterwegs sind. Die Forschungseinrichtung hat Verträge mit zwei großen Airlines und verfügt über ein eigenes Hotelbuchungssystem. Der eigentliche Buchungsprozess ist also immer wieder derselbe. Auch sind mit den einzelnen Mitarbeitern immer wieder Absprachen bezüglich des ausgewählten Fluges bzw. Hotels nötig. Diese doch sehr zeitintensive und auch eintönige Arbeit möchte Annette nun gern den einzelnen Mitarbeiter überlassen. Sie nutzt das Kompositionswerkzeug und erstellt ein Mashup zur einfachen Reisebuchung. Dieses nutzt die Services des Hotelbuchungssystems und der beiden Airlines und erlaubt es, sich die Hotels auch auf einer Karte anzeigen zu lassen. Nach kurzer Einarbeitungszeit hat Annette das Mashup erstellt und verbreitet es unter den Mitarbeitern der Forschungseinrichtung. Diese einmalige Erstellung verschafft ihr in Zukunft mehr Zeit für andere Aufgaben.

A.2 IMPLEMENTIERUNG SERVFACE BUILDER

A.2.1 Klassendiagramm

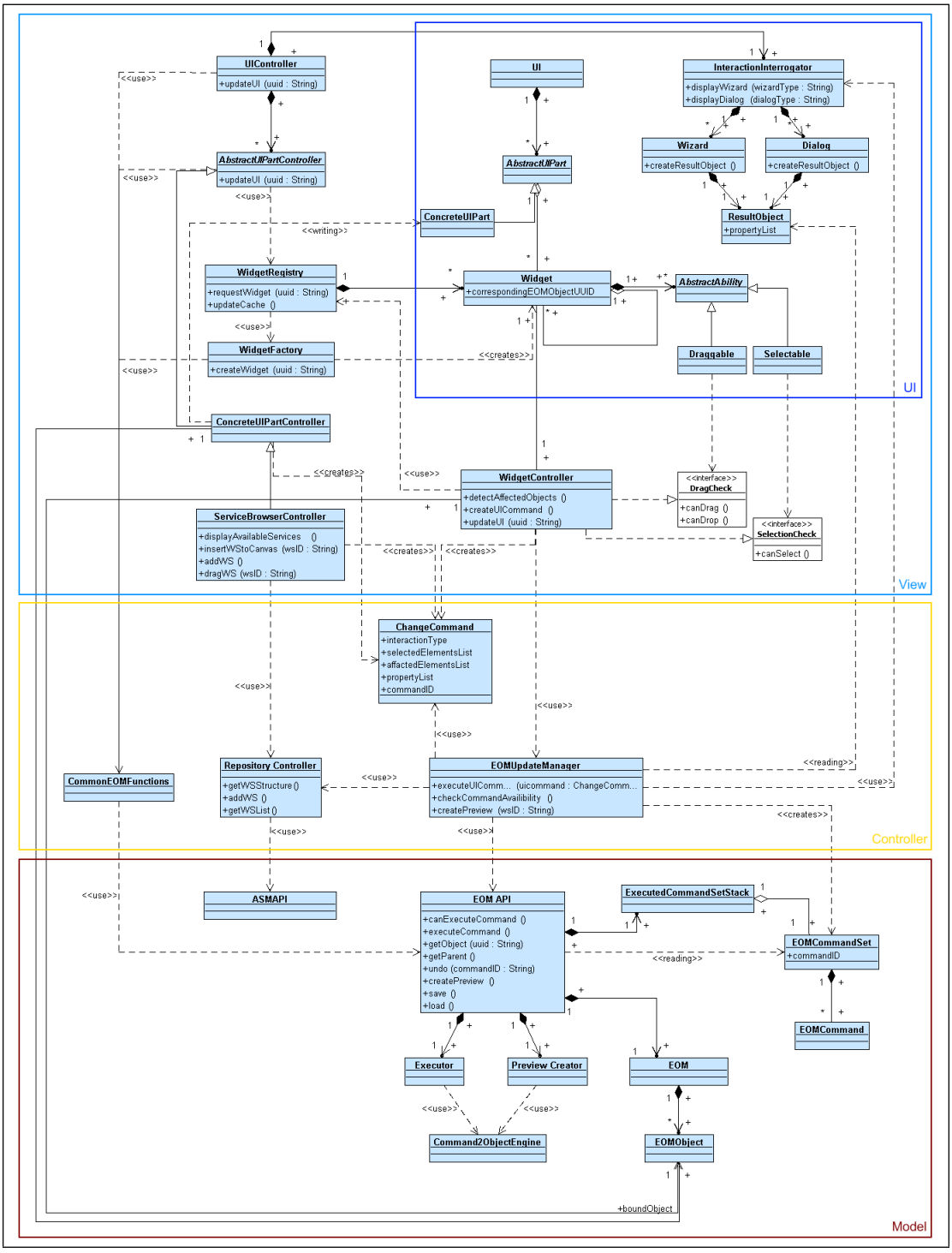


Abbildung A.1: Klassendiagramm Simple Composition Tool

A.3 MODELLE IM SIMPLE COMPOSITION TOOL

A.3.1 Composite Application Model

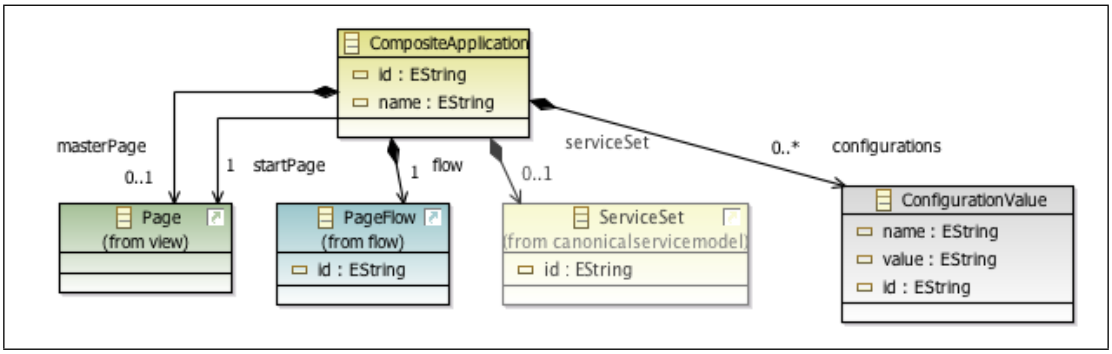


Abbildung A.2: eCore Modell Composite Application Modell: Application-Bereich

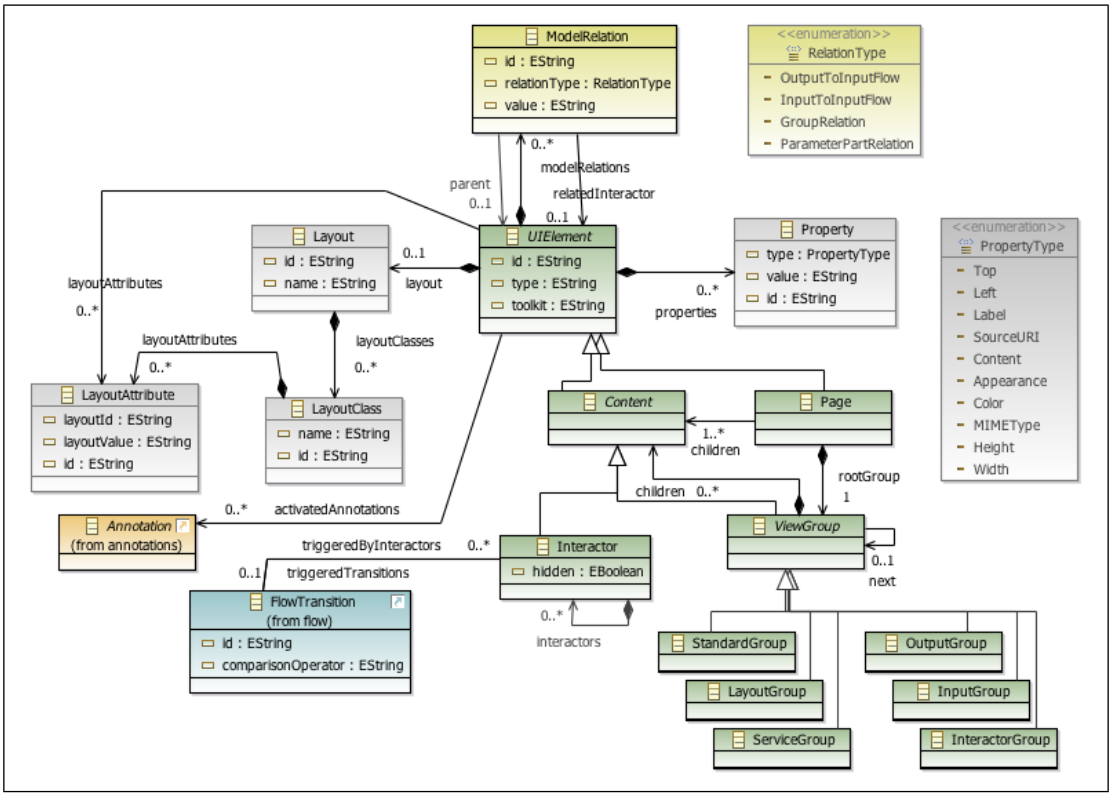


Abbildung A.3: eCore Modell Composite Application Modell: View-Bereich

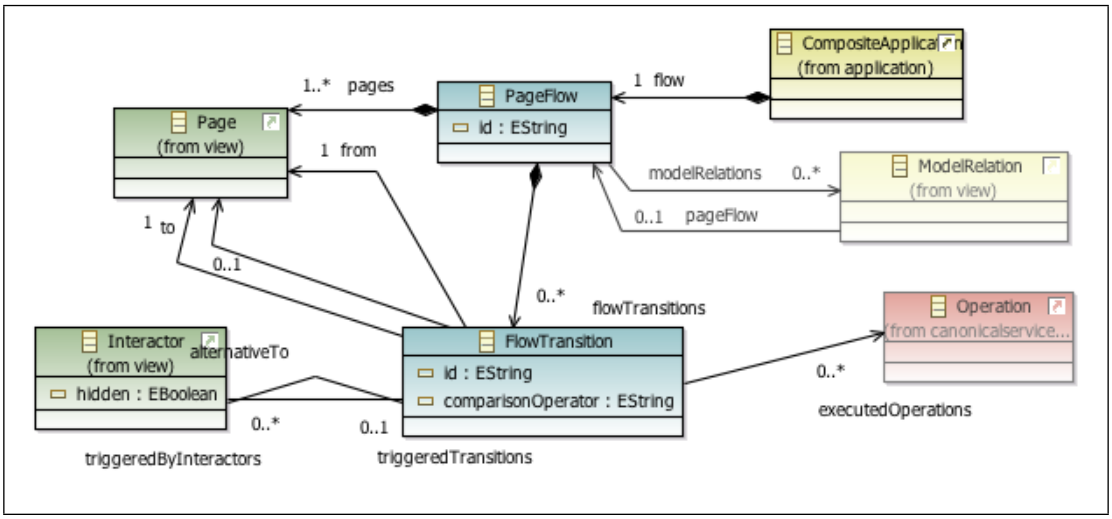


Abbildung A.4: eCore Modell Composite Application Modell: Flow-Bereich

A.3.2 Verfügbare Annotationen im ServFace Builder

- **Visuelle Annotationen:**
  - **Feedback:** Enthält zusätzliche Informationen, wie beispielsweise ein Benennungslabel oder ein Tooltipp, welche dem User des Kompositionswerkzeugs angezeigt werden.
  - **Format:** Definiert eine Formatierung für Ein- und Ausgabeelemente, um einerseits die korrekte Anzeige bei der Ausgabe und die korrekte Eingabe durch den Nutzer der Endapplikation sicherzustellen.
  - **Group:** Definiert eine Gruppierung und eine Ordnung von Webservice-Elementen.
  - **Units:** Ordnet eine Maßeinheit einem Service-Element zu. In Verbindung mit einer Umrechnungsdefinition, kann auch eine Konvertierungsmöglichkeit von Maßeinheiten definiert werden.
  - **Enum:** Definiert eine Gruppe gültiger Werte für ein Webservice-Element.
  - **MIME Type:** Definiert den Dokument-Typen und damit die Art der Anzeige eines Webservice-Elements.
  - **Special DataType:** Ordnet speziellen Datentypen die Darstellung durch Specialwidgets zu.
  - **Visual Property:** Definiert die Art der Darstellung für das zu einem Ein- oder Ausgabeparameter gehörende User-Interface-Element und dessen Inhalt.
  - **Design Templates:** Definiert Design-Templates, die für die Darstellung bestimmter User-Interface-Bereiche genutzt werden sollten.
- **Verhaltensannotationen:**
  - **Validation:** Definition von Überprüfungsregeln zur Feststellung ob eine Eingabe in ein User-Interface-Element den Vorgaben entspricht.
  - **Suggestion:** Hinweis auf die Möglichkeit eines User-Interface-Elements Vorschläge zur Vervollständigung von User-Eingaben anzuzeigen.
  - **Form Completion:** Anzeige, dass ein Formular automatisch ergänzt wird, sobald die Eingabe in die definierten Trigger-Elemente erfolgt ist.
  - **Synchronous Update:** Legt fest, dass ein User-Interface-Element ständig aktualisiert wird.

  - **Appearance Change Rule:** Definiert Regeln zur Beschreibung einer Darstellungsänderung eines User-Interface-Elements in Abhängigkeit von der Eingabe in anderen Elementen.
  - **Default Value:** Festlegung eines Standardwerts, welcher angezeigt bzw. übermittelt wird, wenn keine Eingabe in das entsprechende User-Interface-Element erfolgt ist.
  - **Example Data:** Beispieldaten, die verwendet werden, um dem User des Kompositionswerkzeugs einen besseren Eindruck über die Verwendung der erzeugten Service-Frontend-Elemente zu gewähren.
  - **Mandatory Field:** Spezifizierung eines Service-Elements als notwendig, was bedeutet, dass eine Eingabe in das zugehörige User-Interface-Element erfolgen muss und der Nutzer des Kompositionswerkzeugs dieses Element nicht ohne weiteres aus der Visualisierung des Service-Frontends entfernen kann.

- **Relationale Annotationen:**
  - **Authentication:** Spezifiziert die notwendigen Authentifizierungs-Angaben um eine Service-Operation verwenden zu können.
  - **Semantic Data Type Relation:** Definiert die Beziehung zwischen verschiedenen Datentyp-Elementen.
  - **Bundle:** Definiert eine Gruppe von Services und Service-Operationen, die im Idealfall gemeinsam verwendet werden sollten.
  - **Operation Properties:** Enthält Eigenschaften einer Service-Operation um mit dieser innerhalb des Kompositionswerkzeugs besser arbeiten zu können.

- **Erweiterungen:**
  - **Benutzerdefinierte Annotationen:** Wird zur schnellen und generischen Erstellung weiterer Annotationen verwendet.

A.4 ALTERNATIVE KONZEPTE

A.4.0.1 Auswirkungs-Beschreibungen per XML

Ein alternatives Konzept zum zentralen Beschreibungsklassen-Repository sieht eine Beschreibung der Annotationen per XML vor. Für jede Annotation wird eine XML-Datei erstellt, die detailierte Informationen über die Auswirkungen der Annotation enthält. Beschrieben werden:

- Ort der Auswirkung (UI, Anwendungsmodell)
- Die Art der Auswirkung (Hinzufügen eines Elements, Ändern eines Elements, ...)
- Bei neuen Objekten eine detailierte Spezifikation des neuen Objekts
- Bei Elementänderungen eine detailierte Spezifikation der Änderung
- Auswirkungen auf andere Elemente

Das folgende Listing stellt einen Auszug aus der XML-Beschreibung der Group-Annotation dar:

```
<tns:NewElement  amount="one">
        <tns:camElementID>/annotatedElementID/Group/Counter</tns:camElementID>
        <tns:camElementName>/annotatedElementID/Group</tns:camElementName>
        <tns:camElementType>contentGroup</tns:camElementType>
        <tns:camContainer>self</tns:camContainer>
        <tns:uiType>hBox</tns:uiType>
        <tns:newElementValue>
                <tns:FixValues>
                        <tns:valuePath>/labels/text</tns:valuePath>
                        <tns:valueType>Label</tns:valueType>
                </tns:FixValues>
                <tns:FixValues>
                        <tns:valuePath>/groupElements/element/_</tns:valuePath>
                        <tns:valueType>ServiceModelElements</tns:valueType>
                </tns:FixValues>
        </tns:newElementValue>
</tns:NewElement>
```

Listing A.1: Ausschnitt: XML-Beschreibung Group-Annotation

Verwaltet werden die XML-Beschreibungen von eine XML-Beschreibungs-Komponente. Innerhalb der Komponente ist für jede Annotation die URL der passenden Beschreibungsdatei hinterlegt. Benötigt die AEB-Komponente eine XML-Beschreibung zur Auswertung einer Annotation, dann fragt sie die URL bei der XML-Beschreibungs-Komponente an. Nach Erhalt der URL liest die AEB-Komponente die XML-Beschreibung direkt ein. Auf Basis der Informationen aus der Beschreibung werden dann die Effekt-Objekte für die Annotation erstellt. Der genaue Ablauf ist in der folgenden Grafik dargestellt:

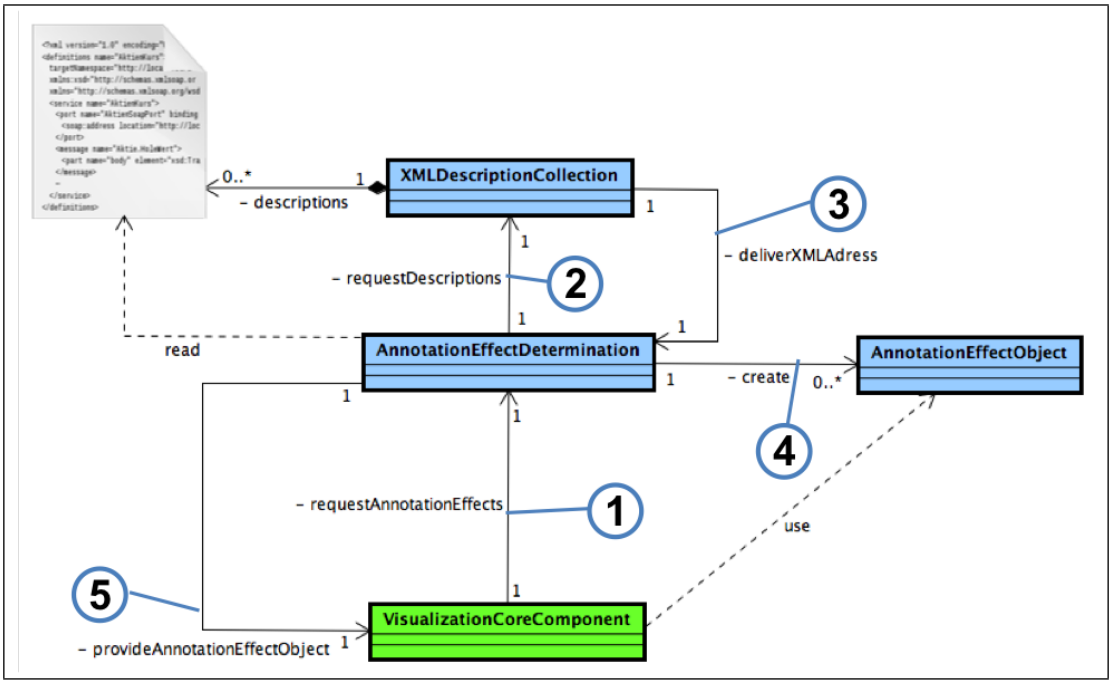


Abbildung A.5: Funktionsweise der Annotations-Auswertung mit XML-Beschreibungen

Während der Annotationsauswertung werden die folgenden, in der Grafik A.5 nummerierten Schritte durchlaufen:

1. Übergabe Service-Elemente an die AEB-Komponente
2. Anforderung der URL zur passenden Annotations-Beschreibung bei der XML-Beschreibungs-Komponente

3. Rückgabe der XML-URL und einlesen der XML-Beschreibung
4. Erstellung des Effekt-Objekts
5. Rückgabe des Effekt-Objekts an die Kernkomponente

Ein großer Vorteil der Auswertungs-Beschreibung per XML ist die Möglichkeit ein allgemeingültiges Schema für alle Annotations-Beschreibungen zu definieren. Dadurch wird ein feste Struktur vorgegeben, wie die Annotationsbeschreibungen aufgebaut sein müssen und welche Daten sie enthalten. Neu erzeugte Annotations-Beschreibungen können gegen das XML-Schema evaluiert und somit auf deren Richtigkeit überprüft werden. Ein Auszug aus dem für dieses Konzept vorgesehene XSD-Schema ist im Anhang A.5.1 zufinden.
Ein weitere Vorteil dieses Konzeptes ist, dass für die Beschreibung der Annotationsauswirkungen kein Code in das Tool implementiert werden muss. Soll eine neue Annotation hinzugefügt werden, dann müssen die Auswirkungen lediglich in XML formuliert werden. Zusätzlich kann mit Werkzeugen wie beispielsweise Eclipse EMF [SBPM08] und OpenArchitectureWare [Fou09] ein Editor automatisch generiert werden, der das Hinzufügen von Annotationen erlaubt. In diesem Fall muss weder Code implementiert noch XML geschrieben werden, um die Auswirkungen der Annotationen zu definieren. Der folgende Screenshot zeigt einen Editor der aus dem XML-Schema automatisch generiert wurde:

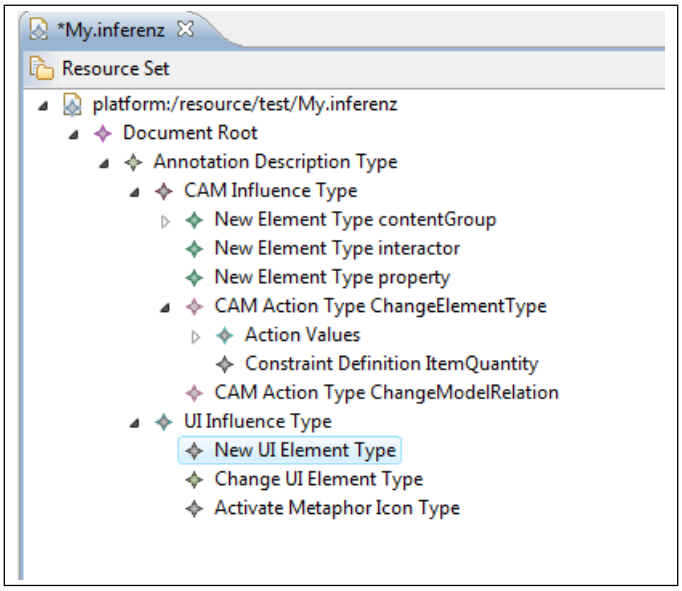


Abbildung A.6: Screenshot: XML Instanz Editor für XML-Beschreibungen nach dem XSD-Schema

Die Nachteile dieser Lösung ergeben sich aus Beschränkungen von C# und Silverlight. Die annotierten Service-Elemente beinhalten eine Liste von Annotationen. Um alle Annotationen gesammelt in einer Liste speichern zu können, müssen die einzelnen Annotationen in der Liste jeweils ihrem abstrakten Obertypen "Annotation" gehorchen. Der eigentliche Typ der Annotation wird versteckt. Liest die AEB-Komponente nun die Annotationen ein, dann ist nur ein Zugriff auf Parameter möglich, die auch in der abstrakten Oberklasse "Annotation" definiert sind. Auf Annotations spezifische Parameter kann nicht direkt zugegriffen werden. Das Problem entsteht, da die Annotationen spezifisch definiert sind und nicht alle einem generischen Modell gehorchen. Mit den folgenden drei Möglichkeiten lässt sich das Problem lösen:

- Nutzung von dynamischen Castings. Der eigentliche Datentyp einer Annotation wird ausgelesen, zwischengespeichert und die Annotation wird auf diesen Typen gecastet.
- Eine Menge von IF-Abfragen, um herauszufinden von welchem konkreten Typ die Annotation ist. Anschließend wird die Annotation auf ihren konkreten Typen gecastet.

- Nutzung der Reflection API zum Zugriff auf die spezifischen Parameter. Ein Casting auf die konkreten Typen muss nicht erfolgen.

Die erste Lösung kann prinzipiell ausgeschlossen werden, da C# 2008 und Silverlight 3 dynamisches Casten nicht direkt unterstützen. Die zweite Lösung egalisiert einige der Vorteile des eigentlichen Konzepts. Für jede Annotation muss eine if-Abfrage und damit Code implementiert werden. Die leichte Erweiterbarkeit wäre damit nicht mehr gegeben. Übrig bleibt die dritte Möglichkeit. Mit der Reflection API kann man unter Angabe des Parameternamens ohne Casting auf die Parameter und Referenzen des konkreten Typen zugreifen. Da jede Annotation andere Parameter und Referenzen definiert, müssen in der XML-Beschreibung jeder Annotation die Pfade zu den entsprechenden Werten angegeben werden. Nicht nur das die Nutzung der Reflection API Performance-Probleme hervorrufen kann [Pob05], auch die Nutzung von Strings für Parameternamen und -pfade ist unsicher. Das Debugging des Codes wird damit erschwert und auftretende Fehler sind schwierig zu finden. Zusätzlich kann auf Grund der vielen unterschiedlichen Annotations-Definitionen nicht sichergestellt werde, dass alle Annotationen ohne Fehler unterstützt werden können.

Zusammengefasst ergibt sich eine sehr gute Lösung, die das Problem der spezifisch definierten Annotationen durch ein generisches XML-Schema zur Beschreibung der Annotationsauswirkungen löst. Durch die Möglichkeit einen Editor zur Erstellung von Annotations-Beschreibungen automatisch zu generieren, wird die Erweiterbarkeit der Annotationsauswertung innerhalb der Kompositionstools deutlich vereinfacht. Leider ist die Implementierung auf Grund der Limitierungen von C# 2008 und Silverlight 3 recht schwierig und weißt einige Nachteile auf.

Das Konzept die Annotationsauswirkungen als XML zu speichern und dynamisch in das Kompositionstool zu laden, ist aus der Sicht der Erweiterbarkeit der Annotationsauswertung die bessere Lösung. Die automatische Generierung eines Editors, mit dem das Hinzufügen neuer Annotationen einfach und ohne Codeimplementierung möglich wäre, ist eine große Stärke dieses Konzepts. Allerdings machen die Integrationsprobleme dieses Ansatzes eine ordnungsgemäße Implementierung schwierig. Deshalb wurde sich entschieden das zweite Konzept der Beschreibungsklassen innerhalb des ServFace Builders zu verwenden. Die einfachere und performantere Integration in die C# und Silverlight-Umgebung ist dabei das entscheidende Argument für die Festlegung. Sollte sich zu einem späteren Zeitpunkt die Möglichkeit ergeben die Annotationen in einem generischen Meta-Modell zu beschreiben, würden die Integrationsprobleme der ersten Lösung wegfallen. Die Entscheidung würde in diesem Fall zu Gunsten des besser erweiterbaren ersten Konzepts getroffen werden.

A.5 CODE-BEISPIELE

A.5.1 XML-Schema für die Annotationsbeschreibung via XML

```
1  <!-- definition of document structure -->
2
3  <xs:element name="AnnotationDescription">
4   <xs:complexType>
5    <xs:all>
6     <xs:element name="Annotation" type="tns:annotationDescription" minOccurs="1"
         maxOccurs="1" />
7     <xs:element name="CAMInfluence" minOccurs="0" maxOccurs="1">
8      <xs:complexType>
9       <xs:sequence>
10       <xs:element name="NewElement" minOccurs="0" maxOccurs="unbounded">
11        <xs:complexType>
12         <xs:sequence>
13          <xs:element name="camElementID" type="tns:Parts" minOccurs="1" maxOccurs="1" />
```

```
14          <xs:element name="camElementName" type="tns:Parts" minOccurs="1" maxOccurs="1"
                />
15          <xs:element name="camElementType" type="tns:camElementTypes" minOccurs="1"
                maxOccurs="1" />
16          <xs:element name="camContainer" type="tns:affectableElements" minOccurs="1"
                maxOccurs="1" />
17          <xs:element name="uiType" type="tns:uiElements" minOccurs="0" maxOccurs="1" />
18          <xs:element name="newElementValue" type="tns:newElementValues" minOccurs="0" />
19          <xs:element name="constraint" type="tns:constraintDefinition" minOccurs="0"
                maxOccurs="unbounded" />
20         </xs:sequence>
21         <xs:attribute name="amount" type="tns:amounts" use="optional" default="one" />
22         </xs:complexType>
23        </xs:element>
24        <xs:element name="CAMAction" minOccurs="0" maxOccurs="unbounded">
25         <xs:complexType>
26          <xs:sequence>
27           <xs:element name="actionType" type="tns:actionTypes" minOccurs="1" />
28           <xs:element name="affectedElements" type="tns:affectableElements" minOccurs="1
                " />
29           <xs:element name="actionValue" type="tns:actionValues" minOccurs="0" />
30           <xs:element name="constraint" type="tns:constraintDefinition" minOccurs="0"
                maxOccurs="unbounded" />
31          </xs:sequence>
32         </xs:complexType>
33        </xs:element>
34       </xs:sequence>
35      </xs:complexType>
36     </xs:element>
37     <xs:element name="UIInfluence" minOccurs="0" maxOccurs="1">
38      <xs:complexType>
39       <xs:sequence>
40        <xs:element name="newUIElement" minOccurs="0" maxOccurs="unbounded">
41         <xs:complexType>
42          <xs:sequence>
43           <xs:element name="UIElementName" type="xs:string" minOccurs="1" maxOccurs="1"
                />
44           <xs:element name="UIElementType" type="tns:UiElementTypes" minOccurs="1"
                maxOccurs="1" />
45           <xs:element name="UIElementValue" type="tns:newUiElementValues" minOccurs="0"
                maxOccurs="unbounded" />
46          </xs:sequence>
47         </xs:complexType>
48        </xs:element>
49        <xs:element name="changeUIElement" minOccurs="0" maxOccurs="unbounded">
50         <xs:complexType>
51          <xs:sequence>
52           <xs:element name="ChangingUIElementName" type="xs:string" minOccurs="1"
                maxOccurs="1" />
53           <xs:element name="ChangingUIElementType" type="tns:UiElementTypes" minOccurs="
                0" maxOccurs="1" />
54           <xs:element name="ChangingUIElementValue" type="tns:newUiElementValues"
                minOccurs="0" maxOccurs="unbounded" />
55          </xs:sequence>
56         </xs:complexType>
57        </xs:element>
58        <xs:element name="activateMetaphorIcon" minOccurs="0" maxOccurs="unbounded">
59         <xs:complexType>
60          <xs:sequence>
61           <xs:element name="metaphorname" type="xs:string" minOccurs="1" maxOccurs="1" /
                >
62           <xs:element name="iconURI" type="tns:URL" minOccurs="1" maxOccurs="1" />
63           <xs:element name="metaphoredElementName" type="xs:string" minOccurs="1" />
64           <xs:element name="tooltipAddition" type="tns:tooltipAdditions" minOccurs="0"
                maxOccurs="1" />
65          </xs:sequence>
66         </xs:complexType>
67        </xs:element>
68       </xs:sequence>
69      </xs:complexType>
70     </xs:element>
```

```
    </xs:all>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Listing A.2: Ausschnitt: XSD-Schema für Annotationsbeschreibungen per XML

A.5.2 XML-Schema der Plattform-Beschreibung

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
 <xs:simpleType name="interActionTypes">
  <xs:restriction base="xs:string">
   <xs:enumeration value="SingleTouch" />
   <xs:enumeration value="MultiTouch" />
   <xs:enumeration value="Mouse" />
   <xs:enumeration value="ScrollWheel" />
   <xs:enumeration value="Stick" />
   <xs:enumeration value="NavigationButton" />
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="dataInputDevices">
  <xs:restriction base="xs:string">
   <xs:enumeration value="ComputerKeyboard" />
   <xs:enumeration value="OnScreenKeyboard" />
   <xs:enumeration value="MobilePhoneKeyboard" />
   <xs:enumeration value="ScrollWheel" />
   <xs:enumeration value="Stick" />
   <xs:enumeration value="NavigationButton" />
  </xs:restriction>
 </xs:simpleType>

 <xs:element name="PlattformConfiguration">
  <xs:complexType>
  <xs:sequence>
   <xs:element name="minResolution">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="resolution_x" type="xs:int" minOccurs="1" maxOccurs="1" />
      <xs:element name="resolution_y" type="xs:int" minOccurs="1" maxOccurs="1" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="interactionType" type="tns:interActionTypes" minOccurs="1"
       maxOccurs="1" />
   <xs:element name="dataInputDevice" type="tns:dataInputDevices" minOccurs="1"
       maxOccurs="1" />
   <xs:element name="supportedToolKit" type="xs:string" minOccurs="1" maxOccurs="1"/>
   <xs:element name="supportsMultiPage" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
   <xs:element name="supportsScrolling" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
   <xs:element name="supportsDataInput" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
   <xs:element name="numberElementsBeforePageBreak" type="xs:int" minOccurs="1"
       maxOccurs="1" />
   <xs:element name="supportedMIMETyes">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="supportedMIMEType" type="xs:string" minOccurs="1" maxOccurs="
          unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="supportedUIElements">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="UIElement" minOccurs="1" maxOccurs="unbounded">
       <xs:complexType>
```

```
55          <xs:sequence>
56           <xs:element name="ElementName" type="xs:string" minOccurs="1" maxOccurs="1"/>
57           <xs:element name="standardHeight" type="xs:int" minOccurs="0" maxOccurs="1"/>
58           <xs:element name="standardWidth" type="xs:int" minOccurs="0" maxOccurs="1"/>
59           <xs:element name="standardColor" type="xs:string" minOccurs="0" maxOccurs="1"/>
60           <xs:element name="standardNumberOfSigns" type="xs:int" minOccurs="0" maxOccurs=
                 "1"/>
61          </xs:sequence>
62         </xs:complexType>
63        </xs:element>
64       </xs:sequence>
65      </xs:complexType>
66     </xs:element>
67    </xs:sequence>
68   </xs:complexType>
69 </xs:element>
70 </xs:schema>
```

Listing A.3: Ausschnitt: XSD-Schema für Plattformbeschreibungen per XML

GLOSSAR

Dieses Glossar enthält Begriffserklärungen und Definitionen für die in dieser Arbeit verwendeten Fachtermini

Endnutzer

Der im Rahmen dieser Arbeit angenommene Endnutzer ist ein "Office-Poweruser". Das bedeutet, dass er in seiner täglichen Arbeit den Umgang mit dem Microsoft Office Paket gewohnt ist und mit diesem in hohem Maße umgehen kann. Weiterhin kennt er sich mit den Basis-Programmen und -Funktionen von Computern aus. Technische Details oder die Funktionsweise weiterführender IT-technischer Konzepte kennt er allerdings nicht. Programmiererfahrung ist nicht vorhanden. Eine detaillierte Beschreibung von Personen aus der definierten Endnutzergruppe sind im Anhang A, Abschnitt A.1 zu finden.

Services

Im Rahmen dieser Arbeit wird ein Service als Webservice angenommen, der über eine WSDL definiert ist. Ein Service selbst enthält Operationen, die durch Ein- und Ausgabeparameter spezifiziert wird.

Design-Time

Der Begriff Design-Time steht für den Zeitraum der Anwendungserstellung.

Run-Time

Die Run-Time stellt den Zeitraum der tatsächlichen Anwendungsausführung dar.

Gebrauchstauglichkeit

Eine weltweit akzeptierte Definition für den Begriff Gebrauchstauglichkeit liefert die ISO Norm 9241 [ISO08]: *"Gebrauchstauglichkeit bezeichnet die Eignung eines Produktes bei der Nutzung durch bestimmte Benutzer in einem bestimmten Benutzungskontext die vorgegebenen Ziele effektiv, effizient und zufriedenstellend zu erreichen. Der Nutzungskontext besteht aus den Benutzern, Arbeitsaufgaben, Arbeitsmitteln (Hardware, Software und Materialien) sowie der physischen und sozialen Umgebung, in der das Produkt eingesetzt wird."*. Gebrauchstauglichkeit wird auch als Usability oder Benutzbarkeit bezeichnet. Die Gebrauchstauglichkeit ist also kein eineindeutiger Wert, sondern ist für jeden Benutzer bzw. jede Benutzergruppe individuell. Im Bereich Computersoftware bezieht sich die Gebrauchstauglichkeit auch häufig auf die Klarheit und die Eleganz des Interaktionsdesigns. Die Bewertung der Gebrauchstauglichkeit setzt sich nach Jakob Nielsen folgendermaßen zusammen [Nie94]:

- Erlernbarkeit
- Effiziens
- Einprägsamkeit
- Fehlertoleranz
- Nutzerzufriedenheit

Frontend

Frontends sind im Rahmen dieser Arbeit UI-Blöcke zur Repräsentation von Service-Operationen. Frontends müssen alle Elemente enthalten, die zu einer Interaktion mit der Service-Operation notwendig sind.

Annotation

Annotation sind im Rahmen dieser Arbeit zusätzliche Informationen, die für einen Service zur Verfügung gestellt werden. Diese Informationen werden durch den Visualisierungsprozess verwendet, um das Ergebnis optisch und funktional zu verbessern.

Annotierte Services

Sind Services die mit zusätzlichen Informationen in Form von Annotationen versehen wurden.

Rahmenanwendung

Eine Rahmenanwendung ist ein früher Protoyp, der als Grundlage für die weitere Anwendungsimplementierung verwendet wird. Die Funktionsumfang von Rahmenanwendungen ist dabei meist sehr gering und variiert je nach Anwendungsfall stark.

Scaffolding

Scaffolding beschreibt die Möglichkeit der automatischen Generierung einer Rahmenanwendung aus zugrundeliegenden Modellen, Datenstrukturen oder funktionalen Beschreibungen.

Meta-Modell

Ein Meta-Modell ist ein Modell, dass beschreibt wie konkrete Modelle aufgebaut werden müssen.

ABKÜRZUNGSVERZEICHNIS

BPEL	Business Process Execution Language
EU	Europäische Union
SOA	Service-oriented Architecture
AEB	Annotations-Effekt-Bestimmung
API	Application Programming Interface
CAM	Composite Application Model
CRUD	Create-Read-Update-Delete
DIN	Deutsches Institut für Normung
EMF	Eclipse Modeling Framework
ER	Entity Relationship
ERD	Entity Relationship Diagram
GUI	Graphical User-Interface
GUIDD	GUI Deployment Descriptor
HSDPA	Highspeed Download Packet Access
HTML	Hypertext Markup Language
IT	Information Technology
MVC	Model-View-Controller
NFR	Non-Functional Requirement
ORM	Object Relation Mapping
RAD	Rapid Application Development
SDK	Software Development Kit
UI	User Interface
UMTS	Universal Mobile Telecommunication System
WSDL	Web Service Description Language
WSGUI	Webservice Graphical User-Interface
WSUI	Web Service User-Interface
WSXL	Webservice Experience Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation

ABBILDUNGSVERZEICHNIS

LISTINGS

LITERATURVERZEICHNIS

[ACG+02] ARSANJANI, Ali ; CHAMBERLAIN, David ; GISOLFI, Dan ; KONURU, Ravi ; MACNAUGHT, Julie ; MAES, Stephane ; MERRICK, Roland ; MUNDEL, David ; RAMAN, TV ; RAMASWAMY, Shankar ; SCHAECK, Thomas ; THOMPSON, Rich ; DIAZ, Angel ; LUCASSEN, John ; WIECHA, Charles F.: Web Service Experience Language. In: *Developer Works* IBM, 2002

[ADC09] ADC, Apple Developer C.: *Apple Human Interface Guidelines*. Apple Inc., 2009

[Amb03] AMBLER, Scott: *Agile Database Techniques*. 1. John Wiley & Sons, 2003

[BHS98] BIBEL, Wolfgang ; HÖLLDOBLER, Steffen ; SCHAUB, Thorsten: *Wissensrepräsentation und Inferenz*. 1. Vieweg Verlagsgesellschaft, 1998

[BK07] BAJAJ, Akhilesh ; KNIGHT, Jason: User Interface Generation From The Data Schema. In: *5th AIS SIGSAND Symposium*, 2007

[Ble06] BLEYH, Nicolás: *Analyse und Vergleich von Ansätzen zur Einbindung von menschlichen Interaktionen in komplexe Web Services*, Technische Universität Dresden, Diplomarbeit, 2006

[BS98] BARRY, Douglas ; STANIENDA, Torsten: Solving the Java Object Storage Problem. In: *IEEE Computing Practices* (1998)

[CCMW01] CHRISTENSEN, Erik ; CURBERA, Francisco ; MEREDITH, Greg ; WEERAWARANA, Sanjiva. *Web Services Description Language (WSDL) 1.1*. 2001

[Cov02] COVER, Robin: Web Services User Interface (WSUI) Initiative / Organization for the Advancement of Structured Information Standards (OASIS). 2002. – Forschungsbericht. none

[FJN+09] FELDMANN, Marius ; JANEIRO, Jordan ; NESTLER, Tobias ; HÜBSCH, Gerald ; JUGEL, Uwe ; PREUSSNER, Andre ; SCHILL, Alexander: An Integrated Approach for Creating Service-Based Interactive Applications. In: *Conference in Human-Computer Interaction (Interact)*, 2009

[FNH+09] FELDMANN, Marius ; NESTLER, Tobias ; HÜBSCH, Gerald ; MUTHMANN, Klemens ; JUGEL, Uwe ; SCHILL, Alexander: Overview of an End user enabled Model-driven Development Approach for Interactive Applications based on Annotated Services. In: *EEE ECOWS Workshop on Emerging Web Services Technology (WEWST'09)*, 2009

[Fou09] FOUNDATION, The E. *Eclipse Modeling Project*. Website. 2009

[GHJV94] GAMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns. Elements of Reusable Object-Oriented Software*. Bd. 1. Addison Wesley, 1994

[Ho09] HO, Mike: *QCodo - PHP Development Framework*. 1. USA: QuasIdea Development LLC, 2009

[Hob95] HOBART, James: Principals of Good GUI Design / Classic System Solutions Inc. 1995. – Forschungsbericht

[ISO01] ISO. *ISO/IEC 9126-1:2001*. 2001

[ISO08] ISO. *DIN EN ISO 9241*. DIN-Norm. 2008

[Jäg06] JÄGER, Robert: *Dynamische Generierung von Client-Anwendungen für die Integration von Web Services in Rich-Client-Plattformen*, Technische Universität Dresden, Diplomarbeit, 2006

[JJ08] JAYAPANDIAN, Magesh ; JAGADISH, H.V.: Automated Creation of a Forms-based Database Query Interface. In: *PVLDB 2008*, 2008

[JPS+09] JANEIRO, Jordan ; PREUSSNER, André ; SPRINGER, Thomas ; SCHILL, Alexander ; WAUER, Matthias: Improving the Development of Service-Based Applications Through Service Annotations. In: *Proceedings of IADIS WWW/Internet 2009*, 2009

[Jug09] JUGEL, Uwe: ServFace Methodology Overview. In: *ServFace Deliverable 1.2*, 2009

[KKM03] KASSOFF, Michael ; KATO, Daishi ; MOHSIN, Waqar: Creating GUIs for Web-Services. In: *IEEE Internet Computing* 7 (2003), Nr. 4, S. 66–73

[Kru05] KRUG, Steve ; KRUG, Steve (Hrsg.): *Don't Make Me Think!: A Common Sense Approach to Web Usability*. New Riders, 2005

[KS06] KASSOFF, Michael ; SPILLNER, Josef: Standard Specification and Documentation. In: *GUI Deployment Descriptor (GUIDD)*, 2006

[Lie08] LIEBING, Christian: *Dynamische GUI-Generierung für REST-basierte Web Services*, Technische Universität Dresden Lehrstuhl Rechnernetze Institut für Systemarchitektur Fakultät Informatik, Diplomarbeit, 2008

[Mic05] MICROSYSTEMS, Sun. *NetBeans IDE: Persistence Demo*. Tutorial. 2005

[Mic07] MICROSOFT. *Visual Studio 2008 Professional*. 2007

[MSD09] MSDN, Microsoft Developer N.: *Windows User Experience Interaction Guidelines*. Microsoft Corp., 2009

[Nie94] NIELSEN, Jakob: *Usability Engineering*. Morgan Kaufmann Publishers, 1994

[Nie02] NIELSEN, Jakob: *Top Ten Web-Design Mistakes of 2002*. 1. useit.com, 2002

[Nie05] NIELSEN, Jakob: *Scrolling and Scrollbars*. 1. useit.com, 2005

[Pob05] POBAR, Joel: Dodge Common Performance Pitfalls to Craft Speedy Applications. In: *msdn Magazine* 8 (2005)

[Pur09] PURSCHE, Andreas: *Präsentationsorientierte Service-Komposition von Enterprise Mashups*, Technische Universität Dresden, Diplomarbeit, 2009

[SBPM08] STEINBERG, Dave ; BUDINSKY, Frank ; PATERNOSTRO, Marcelo ; MERKS, Ed ; GAMMA, Erich (Hrsg.) ; NACKMAN, Lee (Hrsg.) ; WIEGAND, John (Hrsg.): *EMF: Eclipse Modeling Framework*. 2. Addison-Wesley, 2008

[SBS07] SPILLNER, Josef ; BRAUN, Iris ; SCHILL, Alexander: Flexible Human Service Interfaces. In: *Proceedings of ICEIS 2007*, IEEE, 2007

[SFB+08] SPILLNER, Josef ; FELDMANN, Marius ; BRAUN, Iris ; SPRINGER, Thomas ; SCHILL, Alexander: Ad-Hoc Usage of Web Services with Dynvoker. In: *ServiceWave 2008*, 2008

[Sha07] SHARKEY, Kent: Introduction to Subsonic. In: *dotnetslackers.com* (2007)

[Spi06] SPILLNER, Josef. *Project Dynvocation*. 2006

[SS07] SPILLNER, Josef ; SCHILL, Alexander: Analysis on Inference Mechanisms for Schema-driven Forms Generation. In: *XMLT-Tagungsbuch*, 2007

[ST03] SCHAECK, Thomas ; THOMPSON, Richard: Web Services for Remote Portlets White Paper / Organization for the Advancement of Structured Information Standards. 2003. – Forschungsbericht. none

[Sze96] SZEKELY, Pedro: Retrospective and Challenges for Model-Based Interface Development. In: *Third International Eurographics Workshop*, 1996

[TBMMa] THOMPSON, Henry S. ; BEECH, David ; MALONEY, Murray ; MENDELSOHN, Noah. *XML Schema Part 1: Structures Second Edition*

[TBMMb] THOMPSON, Henry S. ; BEECH, David ; MALONEY, Murray ; MENDELSOHN, Noah. *XML Schema Part 2: Datatypes Second Edition*

[Tes08] TESTA, Tony: HowTo: Subsonic Introduction. In: *Tony Testas World* (2008)

[TH06] TATE, Bruce ; HIBBS, Curt: *Durchstarten mit Ruby on Rails*. 1. O'Reilly Verlag GmbH & Co.KG, 2006

[Tox09] TOXBOE, Anders. *User Interface Design Pattern Library*. Website. 2009

[Vin03] VINT, Dan. *XML Schema - Data Types Quick Reference*. 2003