Fakultät Informatik
Institut für Systemarchitektur
Lehrstuhl Rechnernetze

Diplomarbeit

Entwicklung eines Konzeptes zur Kopplung von heterogenen, funktionalen Diensten und UI Services

Verfasser:

Ronny Mennerich

eingereicht bei:

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Betreuer:

Dipl. Medieninf. Christian Liebing

Ronny Mennerich
Hauptstraße 24
01097 Dresden

Medieninformatik
Matrikelnummer: 2852162
31.Oktober 2009

Selbstverständniserklärung

Hiermit erkläre ich, Ronny Mennerich, die vorliegende Diplomarbeit zum Thema

Entwicklung eines Konzeptes zur Kopplung von heterogenen, funktionalen Diensten und UI Services

selbständig und ausschließlich unter Verwendung der im Literaturverzeichnis aufgeführten Literatur- und sonstigen Informationsquellen verfasst zu haben.

Ort, Datum Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Web Services haben sich zu einer der zentralen Technologien im Umfeld serviceorientierter Architekturen (SOA) herausgebildet. Die wesentlichen Konzepte einer einheitlichen Beschreibung von Service-Schnittstellen, der standardisierte Austausch von Nachrichten sowie die lose Kopplung führen vor allem im Bereich der *Business-to-Business* (*B2B*) Kommunikation zwischen Unternehmen zu einer hohen Popularität von *funktionalen Diensten*. Durch die verwendeten Konzepte lassen sich gekapselte Funktionalitäten eines bestehenden Dienstes dabei zu immer neuen weitaus komplexeren funktionalen Diensten und auch Anwendungen kombinieren.
Neben der enormen Verbreitung funktionaler Dienste im Bereich der Unternehmenskommunikation finden diese auch im Bereich der *Business-to-Consumer* (*B2C*) Kommunikation zunehmend Anwendung. Viele namhafte Unternehmen, wie beispielsweise Microsoft, Google oder Yahoo, bieten Informationen und Daten über heterogene Schnittstellen an und ermöglichen so deren Verbreitung. Hinzu kommt die steigende Popularität sogenannter *sozialer Netzwerke* und Community-Plattformen, die in der Regel ebenfalls über Service-Schnittstellen verfügen, um dem Anwendungsnutzer den Zugriff auf öffentliche, aber auch private Daten zu gewähren, ohne auf diese Plattformen direkt zugreifen zu müssen. Jedoch liegt ein großes Problem funktionaler Dienste innerhalb der B2C Kommunikation in der Visualisierung der über Service-Schnittstellen angebotenen Daten. Schnittstellen-Beschreibungsformate, wie beispielsweise die standardisierte *Web Service Description Language* (*WSDL*), wurden vor allem im Hinblick auf eine Maschine-zu-Maschine Kommunikation entwickelt. Möglichkeiten zur Abbildung der so beschriebenen Schnittstellen auf die Präsentationsebe-

ne spielten bei der Entwicklung und Etablierung von Beschreibungsformaten bisher nur eine unwesentliche Rolle.

In diesem Zusammenhang haben sich auf Präsentationsebene (Nutzeroberfläche, UI) gerade im Umfeld von Webanwendungen bereits diverse Frameworks und Toolkits (z. B.: Google Web Toolkit[1], Yahoo! User Interface Library[2]) herausgebildet und etabliert, die es ermöglichen, Komponenten für Nutzeroberflächen (User Interfaces, UI) von Webanwendungen, sog. *Rich Internet Application* (*RIA*), zu entwickeln. Diese Komponenten lassen sich dabei, unterstützt durch Frameworks und Toolkits, dermaßen realisieren, so dass sie sich mit traditionellen Desktopanwendungen in Funktionalität und optischer Aufmachung durchaus messen können. Gleichwohl verfügen diese Frameworks und Toolkits aber auch über eine Vielzahl von Hilfsmitteln und Werkzeugen für Entwickler, die u. a. eine einfache Kommunikation mit funktionalen Diensten ermöglichen. In der Regel handelt es sich bei den so entwickelten UI Komponenten jedoch um sehr proprietäre, auf einen spezifischen Dienst bzw. eine Dienst-Funktionalität angepasste Komponenten, die eine zeit- und damit kostenintensive *manuelle Entwicklung* erfordern und zudem eine Wiederverwendung in anderen Anwendungsszenarien ohne zusätzlichen Aufwand kaum ermöglichen.

Diese Nachteile des manuellen Entwicklungsprozesses von UI Komponenten für einen Zugriff auf Schnittstellen funktionaler Dienste greift ein neuer wissenschaftlicher Ansatz auf. Dieser versucht, vorhandene Schnittstellen-Beschreibungen funktionaler Dienste, teils durch zusätzliche Informationen (Annotationen) angereichert, derart auf die Präsentationsebene abzubilden, dass eine Mensch-zu-Maschine Interaktion realisiert werden kann. Die wichtigsten Forschungsprojekte in diesem Bereich sind das Projekt *Dynvoker*[3] sowie das *ServFace*[4] Projekt.

Das Forschungsprojekt Dynvoker befasst sich mit dem zentralen Problem der Realisierung von Zugriffmechanismen durch Abbildung funktioneller Service-Beschreibungen auf verständliche, grafische Nutzeroberflächen. Das Forschungsprojekt ServFace verfolgt wiederum einen Modell-basierten Ansatz zur Entwicklung Service-basierter Anwendungen, wobei die Entwicklung von Diensten einerseits sowie von korrespondierenden Beschreibungen für Nutzeroberflächen andererseits, aber auch die Entwicklung von Nutzeroberflächen kompositer Dienste die Schwerpunkte der Untersuchungen bilden. Dem in diesen Projekten verfolgten Ansatz der *Generierung grafischer Be-*

[1] Google Web Toolkit; GWT: http://code.google.com/webtoolkit/
[2] Yahoo! User Interface Library; YUI: http://developer.yahoo.com/yui/
[3] Dynvoker: http://dynvoker.org
[4] ServFace: http://www.servface.eu

nutzerschnittstellen steht der Ansatz der *gezielten (Wieder-)Verwendung bereits bestehender UI Komponenten* gegenüber. Das Forschungsprojekt *CRUISe* (*Composition of Rich User Interface Services*) versucht, verschiedene Problemfelder bei der Visualisierung und Kommunikation von Daten zwischen UI-Bestandteilen und funktionalen Diensten zu untersuchen. Ein wesentlicher Schwerpunkt innerhalb des Projektes liegt darin, das Paradigma der Serviceorientierung im Bereich von RIA auf die Präsentationsebene zu übertragen.
Unter diesem Gesichtspunkt sollen sogenannte *User Interface Services* (*UI Service, UIS*) etabliert werden, die den Aspekt der Visualisierung der durch funktionale Dienste angebotenen Daten übernehmen. Anders als bei den bereits angeführten proprietären UI Komponenten sollen dabei wiederverwendbare, konfigurierbare UI-Bestandteile geschaffen werden, die als *UI Komponenten* (*UIC*) innerhalb eines UI Services gekapselt sind. Neben der Erfassung, Modellierung und Verarbeitung von Kontextinformationen (z. B. Nutzer, Endgeräte, Umgebung) steht u. a. die Beschreibung von UI Komponenten bezüglich angebotener Funktionalitäten und (Daten-)Schnittstellen im Fokus des Projektes.
Da bisherige Untersuchungen zur UI Integration und Interaktion mit funktionalen Diensten zumeist von statischen, zur Designzeit festgelegten Kompositionsdokumenten ausgehen, liegt das wesentliche Ziel der vorliegenden Arbeit darin, eine Systemarchitektur zu entwickeln, um damit u. a. eine Kompositionsschnittstelle zu schaffen, die es ermöglicht, UI Komponenten zur Laufzeit zu suchen, auszuwählen und in eine Nutzeroberfläche zu integrieren. Die Grundlagen bilden damit Ansätze des Suchens, Findens und Bindens (allgemein des Discovery) und darüber hinaus auch das Matching mit Schnittstellen funktionaler Dienste sowie die Abbildung der Serviceorientierung auf die Präsentationsebene, die ein Beschreibungsformat für UI Komponenten notwendig machen, um sowohl den Prozess des Discovery und der Integration von UI Komponenten als auch die Kommunikation mit funktionalen Diensten zu gewährleisten.

1.2 Überblick über die Arbeit

Das zweite Kapitel befasst sich mit wichtigen Faktoren zu den Grundlagen sowie dem State-of-the-Art die das Thema der Arbeit betreffen. In Kapitel 3 wird zunächst auf die Spezifizierung eines Beschreibungsformates für User Interface Services bzw. den darin gekapselten User Interface Kompo-

nenten eingegangen, bevor es um die Konzeption einer Systemarchitektur für die dynamische Kopplung von heterogenen, funktionalen Diensten sowie User Interface Services geht. Kapitel 4 befasst sich mit der prototypischen Umsetzung der vorgeschlagenen Systemarchitektur und beschreibt detailliert beispielhafte Umsetzungsszenarien. Mit Kapitel 5 wird das definierte Beschreibungsformat, vor allem aber die konzipierte Systemarchitektur an einem selbst gewählten Beispiel evaluiert, bevor mit Kapitel 6 die gesamte Arbeit abschließend zusammengefasst wird.

Kapitel 2

Grundlagen und State-of-the-Art

Mit dem vorliegenden Kapitel wird zunächst eine Einordnung der Arbeit in das Forschungsprojekt CRUISe vorgenommen. Anschließend werden mit dem Abschnitt 2.2 die grundlegenden Begriffe des funktionalen Dienstes, des UI Services sowie der UI Komponenten geklärt, die innerhalb der Arbeit von Bedeutung sind und häufig Verwendung finden. Aus dem Thema der vorliegenden Arbeit sowie der Begriffsbestimmung des funktionalen Dienstes ergibt sich die Notwendigkeit, auf wichtige Schnittstellen-Beschreibungsformate in diesem Bereich sowie auf Erweiterungen dieser Formate, die sich mit der Komposition von Schnittstellen befassen, genauer einzugehen, womit sich der Abschnitt 2.3 genauer befasst.
Um den Prozess der Komposition von der Designzeit in die Laufzeit zu verschieben, sind Untersuchungen im Rahmen von Matching-Verfahren notwendig. Mit dem Abschnitt 2.4 wird auf grundlegende Verfahren verwiesen, die diesen Prozess unterstützen können. Anschließend werden in Abschnitt 2.5 dieses Kapitels einige wissenschaftliche wie auch proprietäre Ansätze im Rahmen der Integration von UI-Bestandteilen in Nutzeroberflächen und der Erstellung kompositer Anwendungen auf Basis von funktionalen Diensten und UI-Bestandteilen vorgestellt, die Anregungen für die zu entwickelnde Systemarchitektur liefern können.

2.1 Einordnung in das Projekt CRUISe

Das zentrale Ziel des Forschungsprojekts *Composition of Rich User Interface Services* (*CRUISe*) [1, 2] ist es, wie bereits eingangs erwähnt, die Vor-

teile serviceorientierter Architekturen (SOA) auf die Präsentationsebene zu überführen. Die in Abbildung 2.1 vorgestellte CRUISe Infrastruktur ermöglicht damit eine kontextabhängige Auswahl wiederverwendbarer, anpassbarer sowie technologieunabhängiger Komponenten, die als sogenannte User Interface Services (UIS) bezeichnet werden. Diese lassen sich dynamisch zur Laufzeit aus UI-Bestandteilen zu Nutzeroberflächen (User Interface, UI) zusammensetzen.

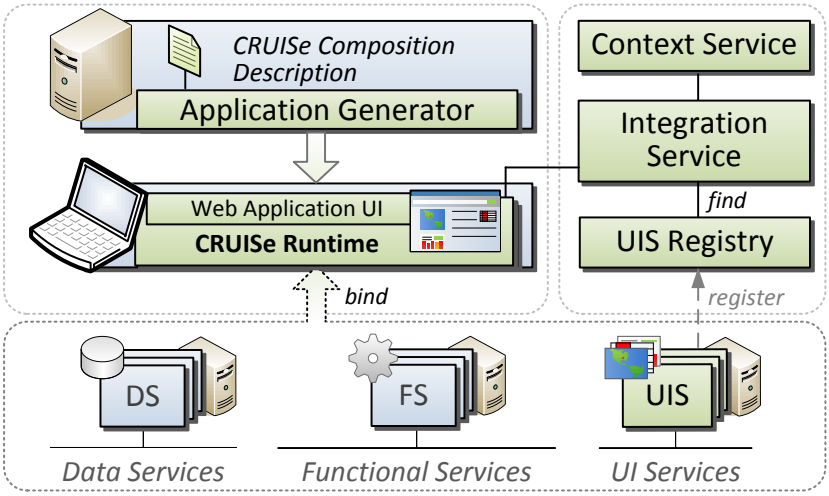


Abbildung 2.1 Überblick über die CRUISe Infrastruktur (aus [1])

Um eine Nutzeroberfläche zu erstellen, werden Konfiguration, Komposition sowie Kontrollfluss deklarativ zur Designzeit festgelegt. Diese sogenannte Kompositionsbeschreibung wird zur Laufzeit durch einen Application Generator ausgewertet und in eine ausführbare Webanwendung transformiert. Die Kompositionsbeschreibung enthält sogenannte Platzhalter, die über Bedingungen und Parameter definiert sind. Zur Laufzeit wird der CRUISe Integration Service (CIS) ausgeführt, der anhand von Platzhalter-Informationen sowie Kontext-Informationen eine Suche innerhalb einer UI Service Registry ausführt, womit die Platzhalter mit dem Integrationscode passender, in UI Services gekapselter UI Komponenten angereichert werden. Unter Integrationscode wird dabei typischerweise Scriptcode verstanden, der in die Anwendung eingebettet und innerhalb des Clients dynamisch ausgeführt werden kann. Neben der Integration, die sowohl serverseitig aber auch clientseitig innerhalb der Runtime ausgeführt werden kann, ist die Runtime zudem für die Steuerung von Ereignissen sowie des Kontrollflusses zwischen den UI Komponenten verantwortlich.
Innerhalb des Forschungsprojektes CRUISe wird eine deutliche Abgrenzung von Technologien zur Erstellung von UI Komponenten vorgenommen, die

kein eigenes Komponentenmodell besitzen. So wird u. a. in [2], aber auch in [3] auf *Web Widgets* und *Google Gadgets* eingegangen, die einem eigenen Komponentenmodell folgen und zudem über eine Spezifikation zur Beschreibung von UI Komponenten verfügen.
Mit den wissenschaftlichen Untersuchungen in [3] wird außerdem versucht, ein Beschreibungsformat für UI Services sowie den darin gekapselten UI Komponenten zu etablieren. Dabei wird deutlich herausgestellt, dass UI Komponenten grundlegend aus zwei Facetten betrachtet werden müssen: zum einen im Hinblick auf die grafische Benutzerschnittstelle (Nutzeroberfläche, UI) welche dem Endanwender präsentiert wird und diesem die Interaktion mit der Oberfläche gewährt; zum anderen im Hinblick auf eine, u. U. programmierbare *Schnittstelle* (*API*), die eine Komposition mit anderen UI Komponenten innerhalb der Anwendungsumgebung ermöglicht und darüber hinaus eine direkte oder indirekte Kommunikation zwischen verschiedenen UI Komponenten sowie zur darunter liegenden Anwendungsumgebung (Ebene der Anwendungslogik und der Daten) ermöglicht.
Das als *User Interface Description Language* (*UISDL*) eingeführte Beschreibungsformat geht neben den syntaktisch spezifizierten Eigenschaften eines UI Service zudem auf notwendige und optionale Beschreibungselemente einer UI Komponente ein. Diese Elemente beziehen sich sowohl auf die allgemeine Beschreibung einer UI Komponente als auch auf Beschreibungselemente, die für eine Integration einer Komponente in den Anwendungskontext bzw. die Benutzerschnittstelle notwendig sind. In diesem Zusammenhang werden Elemente eingeführt, die auf die verwendete Technologie zur Erstellung einer Komponente eingehen. Des Weiteren definieren diese Elemente Eingabe- und Ausgabe-Parameter einer UI Komponente, beschreiben die durch eine UI Komponente ausgelösten Ereignisse und liefern Informationen zu Ressourcen, aus denen eine UI Komponente besteht.

2.1.1 Fazit

Das Forschungsprojekt CRUISe verfolgt mit der Adaption des serviceorientierten Paradigmas auf die Präsentationsebene einen sehr interessanten Gedanken innerhalb der aktuellen Diskussion um Generierung von UI Komponenten auf der einen Seite und Wiederverwendung bereits bestehender UI Komponenten auf der anderen Seite. Das eingeführte Komponentenmodell für UI Services und den darin gekapselten UI Komponenten, aber auch Teile des spezifizierten Beschreibungsformates UISDL können dabei für die vorliegende Arbeit als Basis dienen. Mit Kapitel 3.1 wird das Beschreibungsformat

UISDL erneut aufgegriffen, indem auf die wesentlichen Kriterien und Merkmale von Elementen eines Beschreibungsformates für UI Services zur Umsetzung einer dynamischen Kopplung mit heterogenen, funktionalen Diensten eingegangen wird.

2.2 Begriffbestimmung

Dieser Abschnitt beschäftigt sich zunächst eingehend mit den zentralen Begriffen des *funktionalen Dienstes*, des *User Interface Services* sowie der *User Interface Komponente*, um innerhalb der Arbeit ein allgemeines Verständnis bei der Verwendung der Begriffe zu erzielen.

2.2.1 Funktionaler Dienst

Im Folgenden wird zunächst auf den Begriff des funktionalen Dienstes eingegangen, wobei bei der Bestimmung des Begriffes insbesondere auch die Kategorisierung von service- und ressourcenorientierten Architekturen Berücksichtigung findet.

2.2.1.1 Serviceorientierte funktionale Dienste

Bei der Beschreibung serviceorientierter Architekturen (SOA) wird immer häufiger der Begriff der *Web Services* verwendet, der heute neben Entwicklungen wie CORBA, RMI oder DCOM zu einer der bedeutendsten Technologien für den Aufbau einer SOA gehört.
Für den Begriff des Web Services als solchen existieren bereits eine Reihe von Definitionen, wobei durch das W3C[1] in [4] eine sehr verbreitete Definition geliefert wird:

> *„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“*

Nach dieser Definition werden vor allem standardisierte und plattformunabhängige Technologien wie *SOAP* oder die *Web Service Description Language*

[1] World Wide Web Consortium

(*WSDL*) verwendet, um eine SOA auf Basis von Web Services grundlegend beschreiben und umsetzen zu können.
WSDL dient in seiner Funktion als Beschreibungsformat der Definition einer Schnittstellenbeschreibung für Web Services, die zum einen sowohl maschinenlesbar als auch -verarbeitbar ist, und zum anderen durch einen *Uniform Resource Identifier* (*URI*) eindeutig referenziert werden kann. Ganz allgemein wird eine Schnittstellenbeschreibung dazu verwendet, die durch einen Web Service angebotenen Funktionalitäten näher zu spezifizieren. Dieser Ansatz ermöglicht es, dass ein Web Service in jeder beliebigen Entwicklersprache implementiert sein kann, da die Implementierung dem Entwickler hinter der so beschriebenen Schnittstelle grundlegend verborgen bleibt.

2.2.1.2 Ressourcenorientierte funktionale Dienste

Im Gegensatz zum Ansatz serviceorientierter Dienste, XML-beschriebene Nachrichten als zentrales Mittel zwischen Schnittstellen auszutauschen, besteht die grundlegende Idee ressourcenorientierter Dienste darin, Repräsentationen von Ressourcen über eine generische Schnittstelle zu übertragen. Die Basis liefert der durch R. T. Fielding in seiner Dissertation [5] als *Representational State Transfer* (*REST*) bezeichnete Architekturstil, der im Wesentlichen die Funktionsweise und das Verhalten verteilter Anwendungssysteme wie beispielsweise dem World Wide Web (WWW) beschreibt. Fielding definiert den Begriff REST innerhalb seiner Arbeit wie folgt:

> *„Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.“*

Aufgrund von REST sowie durch Verwendung standardisierter Technologien wie *Hypertext Transfer Protocols* (*HTTP*), URI sowie *Extensible Markup Language* (*XML*) lassen sich *RESTful Web Services* realisieren. Bei diesem Ansatz werden Ressourcen dazu verwendet, Informationen zu repräsentieren, die über eindeutige URIs identifiziert werden. Der Aufruf einer Ressource erfolgt über die generischen Methoden des HTTP, wobei vorrangig die Methoden GET, POST, PUT und DELETE verwendet werden, die damit eine einheitliche Schnittstelle bilden.
Anders als bei den bereits angeführten serviceorientierten Diensten, die über

das Format WSDL beschrieben werden können, werden RESTful Web Services in der Regel lediglich durch Kombinationen aus semi-strukturellen und strukturellen Formaten (z. B.: HTML, XML) dokumentiert, was zu dem Nachteil führt, dass diese nur teilweise maschinenlesbar sind.
Aus diesem Grunde wurde durch Marc J. Hadley die *Web Application Description Language* (*WADL*), in Anlehnung an WSDL, als maschinenlesbares und -verarbeitbares Format zur Beschreibung von Ressourcen im Internet entwickelt.

2.2.1.3 Weitere Verwendung des Begriffes

Unter dem Begriff des funktionalen Dienstes lassen sich auch weitere Ansätze anführen, die Daten bzw. Informationen über Schnittstellen zugänglich machen. Dabei handelt es sich im Wesentlichen um Techniken, die über ein XML-basiertes Format zur Beschreibung der angebotenen Informationen verfügen. In vielen Fällen kommen hierbei die in REST beschriebenen HTTP-Methoden zum Einsatz, die eine Interaktion mit den Schnittstellen ermöglichen. Auf einige Beispiele dieser Art funktionaler Dienste soll im Folgenden genauer eingegangen werden, um die Verwendung des Begriffes innerhalb dieser Arbeit weiter zu konkretisieren.

RSS

Unter der Bezeichnung *RSS* [10, 11] werden eine Reihe unterschiedlicher, sich teilweise ergänzender bzw. aufeinander aufbauender Spezifikationen zusammengefasst, die sich mit der standardisierten Veröffentlichung von Webinhalten befassen. Durchgesetzt hat sich RSS vor allem, um Artikel von Webseiten in Form kurzer Zusammenfassungen innerhalb eines einzelnen Dokumentes, welches als RSS-Feed bezeichnet wird, zu speichern und mittels Verwendung von XML in maschinenlesbarer Form zu veröffentlichen. Anders als beispielsweise bei Mailinglisten, die dazu dienen, Informationen per E-Mail einem größeren Leserkreis zukommen zu lassen, können Interessierte einen RSS-Feed abonnieren, indem durch eine Client-Anwendung (z. B. RSS-Parser, RSS-Reader) lesend auf einen Feed zugegriffen wird und die Informationen präsentiert werden.
Allgemein dient RSS jedoch nur als Format zur Beschreibung von Informationen im Internet. Ein RSS-Feed wird dabei über eine eindeutige URI identifiziert, wobei die eigentliche Dienst-Funktionalität durch den lesenden Zugriff auf einen RSS-Feed über die HTTP-Methode GET ermöglicht wird. RSS beschreibt darüber hinaus keine Schnittstellenfunktionalitäten, welche

eine Bearbeitung eines Feeds ermöglichen; diese Funktionalität wird in der Regel durch die Webanwendung selbst realisiert, über die ein RSS-Feed angeboten wird.

Atom

Unter dem Begriff *Atom* werden zwei Standards zusammengefasst. Zum einen wird mit dem XML-basierten *Atom Syndication Format* (ASF) [12] der plattformunabhängige Austausch von Webinhalten beschrieben, zum anderen wurde ein Anwendungsprotokoll, das *Atom Publishing Protocol* (APP) [13], definiert, um die Erstellung sowie die Bearbeitung angebotener Webinhalte zu ermöglichen. Anders als bei RSS ermöglicht Atom damit sowohl die Bearbeitung als auch die Veröffentlichung bzw. Verbreitung von Inhalten im Internet.

Das Beschreibungsformat ASF versteht sich als Nachfolger von RSS, wobei hier versucht wurde, die Vorteile der unterschiedlichen Ausprägungen von RSS, angereichert durch neue Elemente, in einem neuen Format zu etablieren. Eine wichtige Neuerung gegenüber RSS ist es, die sogenannten inhaltstragenden Elemente um das Format zu erweitern, in dem diese kodiert sind, womit die Maschinenlesbarkeit wesentlich erhöht wird. Die Gruppierung von Informationen in einem durch ASF beschriebenen Dokument wird allgemein auch als Atom-Feed bezeichnet.

Gegenüber ASF, welches der Beschreibung der angebotenen Informationen eines Atom-Feed dient, kann das APP als Programmierschnittstelle gesehen werden, die es Anwendungen ermöglicht, die Informationen eines Atom-Feeds zu bearbeiten. Die Schnittstelle ermöglicht dabei eine Interaktion über die durch den REST-Architekturstil (vgl. 2.2.1.2) bekannten HTTP-Methoden GET, PUT, POST und DELETE, um lesend auf Elemente eines Feeds zuzugreifen, Elemente eines bestehenden Feeds zu bearbeiten, neue Elemente hinzuzufügen bzw. um Elemente eines Atom-Feeds zu löschen.

OpenSearch

OpenSearch [14] ist ein durch Amazon (A9.com) entwickelter Ansatz, um Ergebnisse von Suchmaschinen in einem maschinenlesbaren Format anbieten zu können, wobei sich die zugrunde liegende Technologie in zwei wesentliche Bereiche einteilen lässt. Zum einen wird das XML-basierte OpenSearch-Beschreibungsformat spezifiziert, mittels dessen sich der Zugriff auf eine Suchmaschine syntaktisch beschreiben lässt. Zum anderen lassen sich Formate wie beispielsweise RSS oder Atom zur Präsentation dieser Ergebnisse verwenden.

Allgemein kann festgehalten werden, dass sich mit OpenSearch die Schnittstelle für die Interaktion mit einer Suchmaschine sowie das Rückgabeformat eines Aufrufes der Schnittstelle näher spezifizieren lässt.

2.2.1.4 Fazit

Mit diesen Vorbemerkungen kann die allgemeine Aussage getroffen werden, dass es sich bei einem funktionalen Dienst innerhalb der Arbeit um eine Technologie handelt, welche Informationen über eine Schnittstelle anbietet, deren Funktionalität im Wesentlichen durch ein maschinenlesbares, XML-basiertes Beschreibungsformat ausgedrückt werden kann.

2.2.2 User Interface Service, User Interface Komponente

An dieser Stelle soll der für das Thema wichtige und innerhalb der Arbeit oft verwendete Begriff des *User Interface Service* genauer spezifiziert werden. Zudem soll geklärt werden, welche Rolle *User Interface Komponenten* innerhalb der Arbeit spielen.
User Interface Services wurden bereits in [2] sowie in [3] für das Forschungsprojekt CRUISe (vgl. Kapitel 2.2) genauer definiert. Im Hinblick auf den Kontext der vorliegenden Arbeit zum Projekt CRUISe soll auf diese Definitionen nochmals genauer eingegangen werden, um den Begriff des User Interface Services herauszustellen.
In [2] wird der Begriff dabei wie folgt definiert:

> *„Ein User Interface Service ist ein entfernt liegender, durch ein Netzwerk erreichbarer Dienst. Bei dem Aufruf des Dienstes können Parameter zur Konfiguration übergeben werden. Als Ergebnis des Aufrufs wird eine User Interface Komponente zurück gegeben, die auf einem fest definierten Komponentenmodell beruht.“*

In einer weiteren Arbeit ([3]) wird der Begriff des User Interface Service auf ähnliche Weise definiert:

> *„Ein User Interface Service ist ähnlich dem Web Service ein Softwaresystem, welches durch einen URI identifizierbar ist, und verkörpert einen entfernt liegenden Dienst. Er kapselt eine reichhaltige, generische UI-Komponente, die unabhängig von der verwendeten UI-Technologie über eine Service-Schnittstelle eingebunden werden kann. UI-Services können in Abhängigkeit vom*

> *Kontext, d. h. Nutzer-, Nutzungs- und Systemkontext, zur Laufzeit ausgewählt werden und sind über die Service-Schnittstelle kombinier- und integrierbar.“*

Innerhalb der vorliegenden Arbeit wird der Begriff des User Interface Service etwas differenzierter als in den oben genannten Definitionen betrachtet.
Eine UI Komponente wird als ein funktionales Objekt einer Nutzeroberfläche verstanden. Dieses Objekt verfügt neben visuellen auch über funktionelle Merkmale bzw. Eigenschaften, die über öffentlich zugängliche Operationen angeboten oder unzugänglich umgesetzt werden. Eine Operation kann dabei sowohl über einen Dateneingang als auch einen Datenausgang verfügen, um Informationen aufzunehmen bzw. anzubieten. Zur vollständigen Umsetzung der visuellen aber auch funktionellen Merkmale greift eine UI Komponente u. U. auf externe, durch Dritte angebotene Style-Informationen und Script-Bibliotheken zurück.
Ein UI Service innerhalb der vorliegenden Arbeit ist demzufolge ein entfernt liegendes, über einen URI eindeutig identifizierbares Dokument, welches in einem maschinenlesbaren und -verarbeitbaren Format definiert wird. Dieses Dokument beschreibt eine oder mehrere UI Komponenten syntaktisch über ihre visuellen, funktionellen aber auch kontextuellen Merkmale. Über ein derart spezifiziertes Beschreibungsdokument ermöglicht ein UI Service den Zugriff auf eine UI Komponente, um diese innerhalb eines Anwendungsszenarios in eine Nutzeroberfläche zu integrieren und auf angebotene Funktionalitäten einer Komponente zugreifen zu können.

2.2.2.1 Fazit

Zusammenfassend lässt sich festhalten, dass der Begriff des UI Service innerhalb der Arbeit für ein u. U. extern referenziertes Beschreibungsdokument steht, welches unterschiedliche UI Komponenten sowie deren Funktionalitäten, Eigenschaften und Merkmale in syntaktischer Weise kapselt und weiterhin mit Meta-Informationen anreichert.

2.3 Beschreibung funktionaler Dienste

Wie die Vorbemerkungen der Begriffsbestimmung aufgezeigt haben, lässt sich der Begriff des funktionalen Dienstes in unterschiedlichen Ausprägungen verwenden. Sehr deutlich wurde, dass funktionale Dienste i.d.R. durch abstrakte, maschinenlesbare, XML-basierte Beschreibungsformate näher spezi-

fiziert werden, wobei die zugrunde liegenden Implementierungsdetails hinter der so beschriebenen Schnittstelle eines Dienstes verborgen bleiben.
Da sich das Thema der vorliegenden Arbeit mit der Kopplung funktionaler Dienste sowie User Interface Services bzw. den darin gekapselten User Interface Komponenten befasst, soll im Folgenden näher auf wesentliche Merkmale einiger wichtiger Formate zur Beschreibung funktionaler Dienste eingegangen werden. Des Weiteren wird dabei kurz auf Ansätze zur semantischen Erweiterung von Beschreibungsformaten verwiesen, welche eine dynamische Kopplung funktionaler Dienste unterstützen und folglich für die Arbeit von Interesse sein könnten.

2.3.1 Web Service Description Language

Mit der Web Service Description Language (WSDL) [6, 7] wurde bereits der bekannteste Vertreter von Formaten zur Beschreibung serviceorientierter funktionaler Dienste genannt (vgl. Kapitel 2.2.1.1). Die Beschreibungssprache ermöglicht die Definition einer Dienst-Schnittstelle durch die Angabe unterschiedlicher Elemente. Eine solche Schnittstellenbeschreibung teilt sich in den abstrakten Bereich der Schnittstellen-Details, in dem die Elemente <*types*>, <*message*>, <*portType*> definiert werden sowie in einen konkreten Teil, der zur Beschreibung der Elemente <*binding*>, <*port*> und <*service*> dient und darüber die Interaktionsmöglichkeiten mit einem Dienst ausdrückt.
Das Element <*definitions*> ist das einleitende Wurzelelement einer WSDL-Beschreibung. Es dient der Angabe eines spezifischen Names für den Dienst sowie der verwendeten Namensräume des Dienstes. Außerdem kapselt das Wurzelelement alle im Folgenden vorgestellten Elemente, die einen WSDL-beschriebenen funktionalen Dienst näher spezifizieren.
Innerhalb des Elementes <portType> wird die gesamte Funktionalität eines WSDL-beschriebenen Dienstes durch die Angabe einer Sammlung von <operation>-Elementen gekapselt. Jedes <operation>-Element lässt sich dabei durch Dateneingabe (<*input*>-Element), Datenausgabe (<*output*>-Element) oder Fehlerausgabe (<*fault*>-element) näher beschreiben, welche auf ein bestimmtes <message>-Element verweisen. Das <message>-Element spezifiziert über eines oder mehrere <*part*>-Elemente die Datentypen für Dateneingabe, Datenausgabe und Fehlerausgabe näher. Datentypen, die nicht dem XML-Schema des W3C entsprechen und für eine Interaktion mit dem Dienst benötigt werden, werden innerhalb des <types>-Elementes definiert. Damit lassen sich neben simplen Datentypen des XML-Schemas

auch eigene, möglicherweise sehr komplexe Objekte beschreiben.
Um auf eine bestimmte, innerhalb des <portType>-Elementes definierte Operation zugreifen zu können, müssen ein konkretes Nachrichtenformat sowie das Netzwerkprotokoll festgelegt werden. Hierzu dient das <binding>-Element der WSDL-Beschreibung, wobei das Attribut *transport* das zu verwendende Transportprotokoll näher spezifiziert. Durch die Angabe von <port>-Elementen, die innerhalb des <service>-Elementes gekapselt werden, lassen sich mehrere Kommunikationsendpunkte für die Interaktion mit der zugrunde liegenden Implementierung angeben.
Die angeführten Beschreibungselemente basieren auf der 2001 durch das W3C als Standard festgelegten Version 1.1 von WSDL. Mit der seit 2007 vorliegenden Version 2.0 des Beschreibungsformates haben sich Änderungen hinsichtlich einiger Element-Bezeichner ergeben. Innerhalb der vorliegenden Arbeit wird auf diese Änderungen jedoch nicht näher eingegangen, da WSDL 2.0 bisher noch lediglich als Empfehlung des W3C gilt. Listing A.1 im Anhang dieser Arbeit zeigt ein Beispiel eines einfachen WSDL-Dokumentes, Abbildung 2.2 verdeutlicht dagegen den schematischen Aufbau von WSDL grafisch.

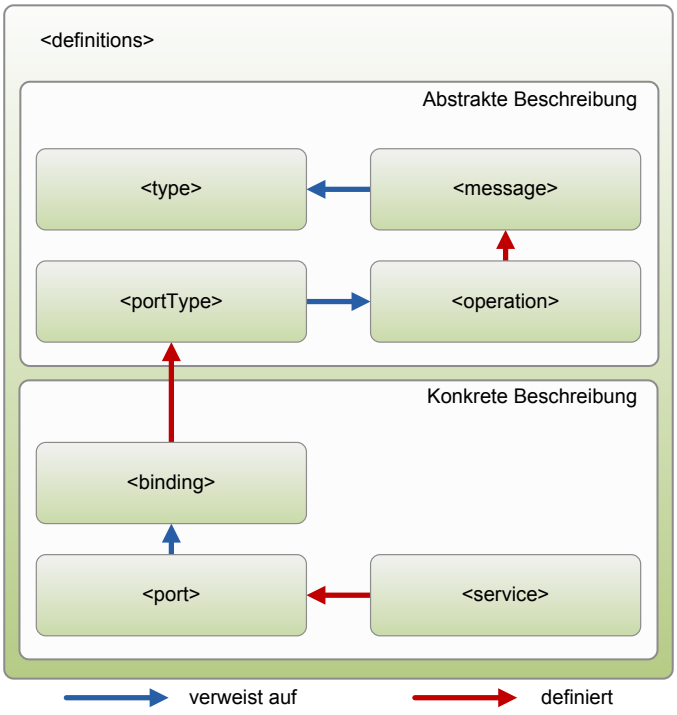


Abbildung 2.2 Schematischer Aufbau von WSDL

Mit diesen Ausführungen zeigt sich, dass WSDL im Wesentlichen der syntak-

tisch funktionellen Beschreibung von Diensten dient, d. h. mit WSDL lässt sich beispielsweise der Aufbau von Nachrichten beschreiben, die zwischen einem Client und einem WSDL-beschriebenen Dienst ausgetauscht werden können. WSDL bietet ohne Weiteres jedoch keine Möglichkeit zur Spezifizierung der Bedeutung, die einer so kommunizierten Nachricht beizumessen ist. Dieses Beispiel macht deutlich, dass die Interpretation der Bedeutung einer Servicebeschreibung bislang dem Entwickler überlassen bleibt und sich Beziehungen zwischen Operationen und Nachrichten nicht direkt mit WSDL beschreiben lassen. Es fehlt dem WSDL Standard hier an Ausdrucksmächtigkeit in Bezug auf die semantische Beschreibung von Merkmalen und Eigenschaften eines Dienstes.

An dieser Stelle setzt *Semantic Annotations for WSDL and XML Schema* (SAWSDL) [16, 17, 18, 19, 20, 21] ein, das sich mit der semantischen Annotation der abstrakten WSDL-Beschreibung eines funktionalen Dienstes befasst, um Schnittstellenbeschreibungen nicht nur maschinenlesbar, sondern auch für Maschinen verständlich zu machen, um damit die Möglichkeit zu schaffen, Dienste zu neuen funktionalen Diensten mit einem minimal notwendigen menschlichen Aufwand zusammenstellen zu können.

Aktuell liegt SAWSDL als Empfehlung des W3C vor und adaptiert die grundlegenden Ideen und Konzepte, die bereits mit *Web Service Semantics* (WSDL-S) [22] präsentiert wurden.

SAWSDL befasst sich dabei vor allem mit der Anreicherung definierter Operationen sowie Eingabe- und Ausgabeparametern von Nachrichten einer WSDL-Beschreibung um zusätzliche Informationen durch die Einführung der Attribute *Model Reference* sowie *Schema Mapping*.

Das *Model Reference* Attribut wird innerhalb eines WSDL Dokumentes im Allgemeinen über den Bezeichner *sawsdl:modelReference* deklariert. Es dient der Angabe von Beziehungen zwischen WSDL-Elementen <portTypes>, <operation> bzw. den innerhalb des <types>-Elementes definierten Datentypen sowie einem oder mehreren (externen) semantischen Konzepten, die über entsprechende URIs referenziert werden. Damit lassen sich die so erweiterten Elemente durch Schemata eines entsprechenden Wissensgebietes bzw. eines Kompetenzbereiches (Domäne) näher beschreiben.

Das Attribut *Schema Mapping* definiert zwei Bezeichner mit unterschiedlichen Anwendungsfeldern. Der Bezeichner *sawsdl:liftingSchemaMapping* dient der Transformation von XML Strukturdaten einer Nachricht in die Struktur eines semantischen Modell. Der zweite Bezeichner *sawsdl:loweringSchemaMapping* dient hingegen der Transformationen von Strukturdaten eines semantischen Modells in die entsprechende Repräsenta-

tion von XML Strukturdaten einer Nachricht. In beiden Fällen wird dabei das Schema für das jeweilige Mapping über die Angabe einer URI referenziert.
Da SAWSDL auf WSDL 2.0 aufsetzt, wird mit *sawsdl:attrExtension* ein weiteres Attribute definiert, durch welches die Kompatibilität zu WSDL 1.1 gewahrt werden soll.
Durch SAWSDL lassen sich WSDL-beschriebene funktionale Dienste dergestalt erweitern, dass neben dem Suchen und Finden von Diensten auch der Austausch von Nachrichten erleichtert wird. Sofern zwei Schnittstellen unter einer bestimmten Domäne übereinstimmen (Model Reference), lassen sich über definierte Mappings (Schema Mapping) die auszutauschenden Nachrichten zwischen den Schnittstellen transformieren.

2.3.2 Web Application Description Language

Auch die bereits in Abschnitt 2.2.1.2 angeführte Web Application Description Language (WADL) [8, 9] sollte durch die ansteigende Verbreitung ressourcenorientierter funktionaler Dienste (RESTful Web Services) in ihrer syntaktischen Spezifikation näher betrachtet werden.
Das <*application*>-Element ist das einleitende Wurzelelement einer WADL-Beschreibung. Es kapselt alle nachfolgend vorgestellten Hauptelemente und definiert als Attribute alle verwendeten Namensräume eines RESTful Web Service, wie beispielweise den durch alle WADL-Elemente verwendeten XML Namensraum[2].
Das darauf folgende <*grammars*>-Element dient der direkten Definition verwendeter Datentypen. Auch die Einbindung externer Definitionen innerhalb eines <*include*>-Elementes sowie durch Angabe einer URI über das zugehörige *href*-Attribut wird an dieser Stelle unterstützt.
Das <*resources*>-Element kapselt die Ressourcen eines RESTful Web Service und definiert über das Attribut *base* eine Basis-URI für alle Ressourcen. Jede gekapselte Ressource wird mit einem <resource>-Element näher beschrieben. Dabei dient das *path*-Attribut zur relativen Addressierung und erweitert die Basis-URI, womit die Referenzierung aller Ressourcen einem gemeinsamen URI-Template unterliegt.
Mit dem <*method*>-Element werden die auf einer Ressource ausführbaren Methoden durch ihre Eingabe und Ausgabe beschrieben, wobei eine Methode direkt definiert oder aber über eine Referenz auf eine andere Methode verweisen kann, welche wiederum an anderer Stelle definiert wurde.

[2] http://wadl.dev.java.net/2009/02

Für die Eingabe wird das <*request*>-Element definiert, welches eine Menge von Eingabe-Parametern beschreibt. Diese Parameter werden durch das Element <*params*> angegeben und bei der Ausführung einer Methode auf einer Ressource der Basis-URI sowie dem relativen Pfad der Ressource hinzugefügt. Mit dem Attribut *required* lässt sich zudem festlegen, ob ein Eingabe-Parameter notwendig für den Aufruf einer Methode ist. Eine zu erwartende Antwort wird mit dem <*response*>-Element näher spezifiziert, welches die verwendeten HTTP-Header, die zur Verfügung stehenden Repräsentationen sowie möglicherweise aufgetretene Fehler als Elemente kapselt. Das im Anhang dieser Arbeit angefügte Listing A.2 zeigt ein Beispiel eines einfachen WADL-Dokumentes zur Beschreibung einer Anwendung, Abbildung 2.3 verdeutlicht den schematischen Aufbau von WADL.

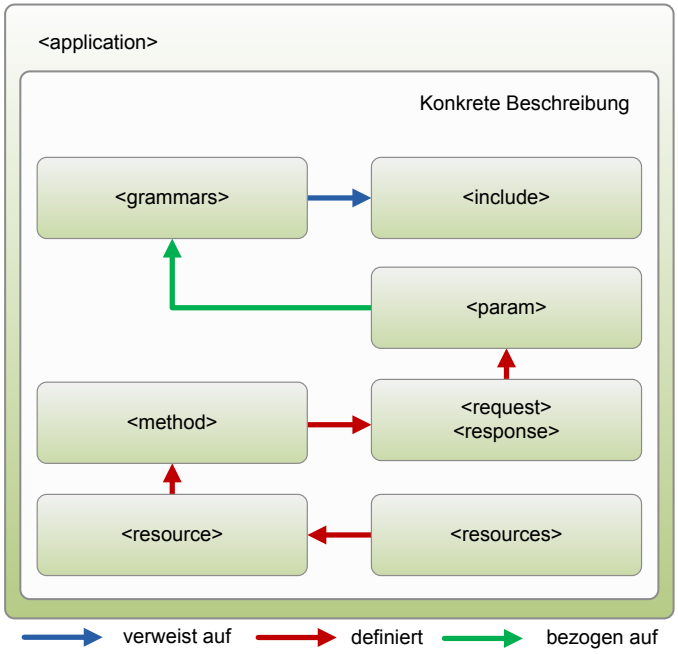


Abbildung 2.3 Schematischer Aufbau von WADL

Auch REST-basierte Dienste können durch Mechanismen zur semantischen Annotation erweitert beschrieben werden. Mit *Semantic Annotation of RESTful Web Services* (*SA-REST*) [23, 24, 25, 26] wurde ein erster Ansatz geschaffen, wobei die grundlegenden Ideen zur Annotation von SAWSDL übernommen wurden. Im Gegensatz zu SAWSDL, das sich auf die semantische Erweiterung von WSDL-Beschreibungsdokumenten bezieht, geht SA-REST einen anderen Weg und setzt nicht auf das Beschreibungsformat WADL auf. Vielmehr begründet sich SA-REST auf dem weit verbreitete

Ansatz, REST-basierte Dienste durch Webseiten, die im Wesentlichen im (X)HTML-Format vorliegen, näher zu beschreiben.
Da SA-REST in erster Linie auf den Ideen von SAWSDL beruht und insofern keinen echten Mehrwert für die vorliegende Arbeit generiert, wird auf diese Möglichkeit der semantische Annotation REST-basierter Dienste an dieser Stelle nicht genauer eingegangen.

2.3.3 Fazit

Die vorgestellten Formate WSDL und WADL machen deutlich, dass XML-basierte Beschreibungsformate die Maschinenlesbarkeit wesentlich unterstützen. Die semantischen Annotation der Schnittstellenbeschreibungen funktionaler Dienste durch Referenzierung gemeinsamer semantischer Modelle sowie die Erstellung notwendiger Schema-Mappings, um den korrekten Austausch von Daten zwischen unterschiedlichen Schnittstellen zu ermöglichen, können zudem eine Komposition sowie die Interaktion zwischen funktionalen Diensten deutlich fördern. In der Regel werden die verwendeten Mappings jedoch manuell erstellt, indem Schnittstellen zu komponierender funktionaler Dienste zur Designzeit untersucht und entsprechende Kompositionsdokumente erstellt werden.
Um diesen manuellen Prozess auf die Laufzeit verlagern zu können, befasst sich der folgende Abschnitt mit aktuellen Ansätzen und Verfahren im Bereich des Schema-Matching, welche als Basis zur Automatisierung von Schnittstellenvergleichen im Rahmen der Konzeption einer Systemarchitektur dienen können.

2.4 Schema-Matching

Mit dem *Matching*[3] verschiedener Schemata[4] [27, 28, 29, 30] wird das Ziel verfolgt, Übereinstimmungen zwischen semantisch verbundenen Elementen dieser Schemata innerhalb einer bestimmten Domäne, einem Wissensgebiet bzw. Kompetenzbereich, herauszustellen. Zu diesen Elementen von Schemata werden u. a. Klassen (class), Eigenschaften (property) oder Individuen (individual) gezählt, wobei aber auch komplexere Ausdrücke wie Formeln, konzeptionelle Definitionen, Anfragen (query) oder begriffsbildende Ausdrücke

[3] „match“ zu deutch: finden von Übereinstimmung, Entsprechung, Gegenstück
[4] In der angeführten Literatur wird in diesem Zusammenhang i. d. R. der Begriff Ontologie verwendet. Für die vorliegende Arbeit soll jedoch der Begriff des Schemas bzw. der Schemata verwendet werden, um die Allgemeingültigkeit der vorgestellten Inhalte herauszustellen.

zu den semantisch verbundenen Elementen zählen. Entsprechnungen, d. h. Resultate eines *Schema-Matching* zwischen Elementen oder auch deren Beschreibungen werden in der Literatur als *Alignments* [27] oder *Mappings* [29] bezeichnet und tragen dazu bei, dass Schemata paarweise verwendet oder zusammengeführt werden können.

2.4.1 Matching Problem

Das *Problem des Matching* von Schemata besteht, wenn sich zwei oder mehr innerhalb einer bestimmten Domäne modellierten Schemata auf verschiedene Weise unterscheiden. Diese Unterschiede lassen sich als *semantische Heterogenität* bezeichnen. Dabei führt nicht nur die Verwendung unterschiedlicher Sprachen zur Beschreibung von Schemata zu Heterogenitäten, auch Unterschiede in den gewählten Begriffen, verschiedene Detaillierungsgrade oder unterschiedliche Sichtweisen können dazu führen, dass Interoperabilität zwischen Schemata nicht gewährleistet werden kann [29].

2.4.2 Matching-Verfahren

Um Schemata aufeinander abstimmen (matchen) zu können, lassen sich, wie oben angesprochen, Mappings oder Alignment verwenden. So kann beispielweise die Aussage getroffen werden, dass im einfachsten Fall ein Element eines Schemas S_1 einem Element eines Schemas S_2 entspricht (vgl. Abbildung 2.4). Diese als Mapping bezeichnete Aussage kann einerseits manuell erstellt werden, andererseits lässt sich jedoch auch ein automatisiertes Auffinden von Mappings mit Hilfe von *Matching-Verfahren* realisieren.

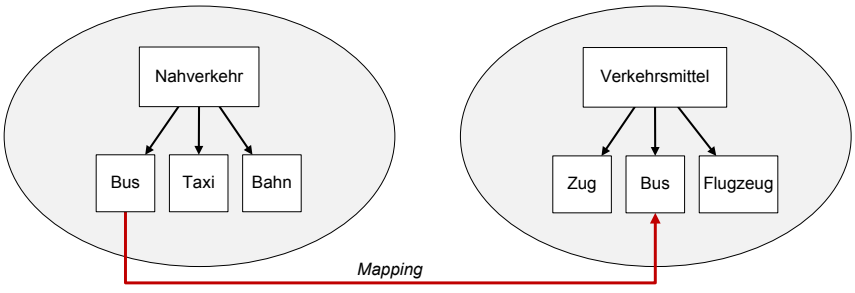


Abbildung 2.4 Übereinstimmung zw. Elementen zweier Ontologien (nach [29])

Matching-Verfahren lassen sich nach [28] zum einen in *Element-basierte* Verfahren, zum anderen in *Struktur-basierte* Verfahren einteilen. Beide Kategorien lassen sich zudem in *interne* als auch *externe Verfahren* gliedern. Bei den interne Verfahren werden nur die zu vergleichenden Schemata selbst als

Eingabedaten verwendet, wogegen externe Verfahren für einen Vergleich zusätzlich auf externe Quellen bzw. Ressourcen zurückgreifen. Im Folgenden wird sowohl auf Element-basierte als auch auf Struktur-basierte Verfahren genauer eingegangen sowie die Einordnung in interne bzw. externe Verfahren verdeutlicht.

2.4.2.1 Element-basierte Verfahren

Bei den *Element-basierten* Matching Verfahren handelt es sich um paarweise Vergleiche einzelner Elemente von Schemata, wobei hier in der Regel der Name eines Elementes für ein Matching herangezogen wird. Neben einer häufig eingesetzten feingranularen Unterscheidung der Elemente einzelner Entitäten von Schemata wird hier auf eine grobgranulare Unterscheidung von Elementen größerer Strukturen von Schemata angesprochen, was nach [30] auch als schwache und starke semantische Unterscheidung bezeichnet wird. Bei diesen Verfahren werden Elemente und deren Beziehungen isoliert zu anderen Elementen betrachtet, so dass Relationen wie Subtyp-Supertyp-Beziehungen hier keinerlei Berücksichtigung finden.

String-basierte Verfahren

Bei den *String-basierten* Matching Verfahren werden, wie einleitend erwähnt, Vergleiche zwischen den Bezeichnern (Namen) und auch den Beschreibungen von Elementen durchgeführt, wobei Worte (Strings) als Sequenz von Buchstaben eines zugrunde liegenden Alphabetes angesehen werden. Die am häufigsten verwendeten String-basierten Matching Verfahren lassen sich wie folgt zusammenfassen:

- *Prefix*: zwei Worte dienen als Eingabe; es wird untersucht, ob das längere Wort mit dem kürzeren Wort beginnt (z.B. „Netz“ und „Netzwerk“, „Schnee“ und „Schneematsch“)
- *Suffix*: zwei Worte dienen als Eingabe; es wird untersucht, ob das längere Wort mit dem kürzeren Wort endet (z.B. „PersonalID“ und „ID“, „PID“ und „ID“)
- *Distanz* bzw. *Abstand*: zwei Worte dienen als Eingabe; es wird, ausgehend vom längeren Wort, über die Operationen EINFÜGEN, LÖSCHEN, SUBSTITUIEREN berechnet, wieviele Schritte notwendig sind, um ein Wort in das andere zu transformieren (z.B. distance(NKN,Nikon) = 0.4)[5]

In der Literatur werden *Sprach-basierte* Verfahren als Erweiterung der String-basierten Verfahren angegeben, die zusätzliche Verarbeitungsschritte bei der Untersuchung von Ähnlichkeiten zwischen Worten durchführen.

[5] bekannteste Distanz-Verfahren: Hamming-Distanz, Levenshtein-Distanz

Dabei werden die zu vergleichenden Bezeichner in einzelne sprachliche Abschnitte (Token) zerlegt. Es wird nun versucht, eine Rückführung der häufig in Flexionsform (Steigerung) vorliegenden Token in Ihre Grundform vorzunehmen, irrelevante Token[6] zu entfernen, diakritische Zeichen[7] zu ersetzen. Auf die so vorverarbeiteten Strings lassen sich im weiteren Verlauf der Untersuchung nun u. a. die bereits angeführten String-basierten Verfahren anwenden.

Constraint-basierte Verfahren

Bei der Untersuchung von Eigenschaften (Constraints) der Elemente unterschiedlicher Schemata werden Untersuchungen vorgenommen, die über den Bezeichner oder die Beschreibung von Elementen (vgl. String-basierte Verfahren) hinausgehen.

Zu den Eigenschaften von Elementen zählen u. a. die Angabe von Datentypen sowie deren Wertebereiche. Hier lässt sich beispielsweise festhalten, dass zwei Elemente mit dem gleichen Wertebereich mit einer gewissen Wahrscheinlichkeit übereinstimmen und als gleich angesehen werden können. Nach [28] zählen zu den zu untersuchenden Eigenschaften auch Elemente, die über mengenwertige Angaben verfügen (vgl. Abbildung 2.5); damit gelten gleiche oder ähnliche Kardinalitätsbeschränkungen.

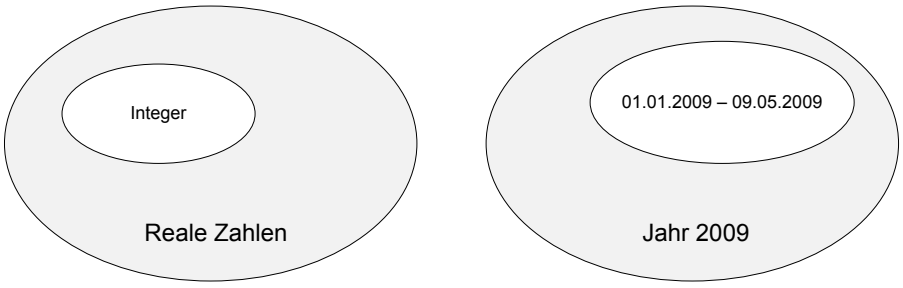


Abbildung 2.5 Betrachtung von Wertebereichen und Mengen (aus [29])

Jedoch werden Constraint-basierte Verfahren in der Regel mit anderen Verfahren kombiniert angewendet, da mit der Angabe einer Eigenschaft zu wenig über Ähnlichkeit zwischen Elementen ausgesagt werden kann. So lassen sich beispielweise Telefonnummer und Hausnummer als numerischer Typ angeben, jedoch ist ein Mapping zwischen diesen Elementen wenig sinnvoll.

[6] sogenannte „stop words“; Artikel bzw. Konjugationen
[7] Umlaute, Zeichen mit Akzenten; z.B. „München“ wird zu „Muenchen“

Linguistische Verfahren

Bei Linguistischen Verfahren handelt es sich im Wesentlichen um Verfahren, die auf externe Quellen zurückgreifen und damit als externe Verfahren bezeichnet werden. Hierbei werden allgemeine wie auch domänenspezifische Thesauri oder Wörterbücher eingesetzt, um auf Ähnlichkeiten zwischen Elementen der zu untersuchenden Schemata schließen zu können. Zu den bekanntesten Thesauri gehören *WordNet*[8] für die englische Sprache, *GermaNet*[9] im Deutschen sowie das mehrsprachige Thesaurus *EuroWordNet*[10].

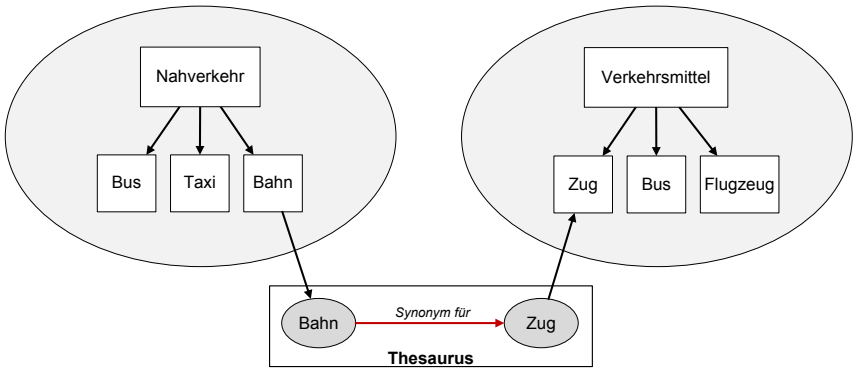


Abbildung 2.6 Externes Matching mit Hilfe eines Thesaurus (nach [29])

Bei der Betrachtung werden zu untersuchende Worte mit Einträgen aus Thesauri abgeglichen, die u. a. Einträge mit Synonym-[11] bzw. Hypernym-[12]Relationen speichern. Dabei werden Äquivalenzen, also Gleichwertigkeiten, zwischen zu untersuchenden Worten und den gespeicherten Einträgen einer externen Quelle identifiziert, die im Wesentlichen einem Abgleich mit sinn- bzw. sachverwandten Worten entsprechen. Die folgenden Beispiele sollen die Verwendung von bzw. den Abgleich mit Thesauri oder Wörterbüchern verdeutlichen:

- $A \subseteq B$ wenn A Hyponym (A Unterbegriff von B) oder Meronym (A Teil von B) von B ist (z.B. $Markenname \subseteq Name$)
- $A \supseteq B$ wenn A ein Hypernym (A Oberbegriff von B) oder Holonym (B ist Teil von A) von B ist (z.B. $Europa \supseteq Griechenland$)
- $A = B$ wenn sie Synonyme sind (z.B. $Anzahl = Menge$)

[8] WordNet: Universität von Princeton, http://wordnet.princeton.edu/
[9] GermaNet: Universität Tübingen, http://www.sfs.uni-tuebingen.de/GermaNet/
[10] EuroWordNet: Universität von Amsterdam, http://www.illc.uva.nl/EuroWordNet/
[11] Synonyme: Bedeutungsgleiche oder sinnverwandte Worte, z. B. „electronic-mail“ als Synonym für „email“
[12] Hypernym: Verallgemeinerung eines Ursprungswortes, z. B. Tier ist Hypernym von Hund

2.4.2.2 Struktur-basierte Verfahren

Struktur-basierte Verfahren arbeiten im Gegensatz zu den vorgestellten Element-basierten Verfahren nicht mit einzelnen Schemata-Elementen, sondern untersuchen ganze Gruppen von Elementen, die zueinander in Beziehung stehen.
Auch die Struktur-basierten Verfahren lassen sich in interne und externe Verfahren unterteilen, wobei interne Verfahren hier nicht eigenständig verwendet werden können, da sie auf bereits vorliegenden Schema-Mappings aufbauen. Diese lassen sich zum einen manuell angeben oder durch bereits beschriebene Element-basierte Verfahren gewinnen. Im Folgenden wird nun auf eine Auswahl von Verfahren in diesem Bereich näher eingegangen.

Graph-basierte Verfahren
Bei dieser Art von Verfahren werden Schemata in Form von *Bäumen* oder *Graphen* betrachtet. In [28] wird davon ausgegangen, dass aus einer Ähnlichkeit zwischen zwei Elementen unterschiedlicher Schemata auch auf eine Ähnlichkeit zwischen den Nachbarn dieser Elemente geschlossen werden kann. Dabei wird zwischen Verfahren unterschieden, die nur unmittelbare Nachbarn bzw. Nachbarknoten (Local Context) untersuchen, und Verfahren, die überdies weiter entfernte Knoten (Non-Local Context) in die Betrachtung einbeziehen.

Taxonomie-basierte Verfahren
Mit Verfahren zur Analyse der *Taxonomie* wird ein Spezialfall der Graph-basierten Verfahren beschrieben. Bei diesem Verfahren wird davon ausgegangen, dass Nachbarknoten (Kindknoten) eines identifizierten Konzeptes (Elternknoten) in Relation („is-a“ Relation) zueinander stehen, d.h. wenn diese in ihrem übergeordneten Konzept übereinstimmen, seien auch Gemeinsamkeiten bzw. Entsprechungen zwischen den Knoten vorhanden.

Modell-basierte Verfahren
Bei *Modell-basierten* Verfahren (auch: semantisch gestützte Verfahren) werden Schemata und die hierfür bereits identifizierten Mappings mit Hilfe einer logischen Beschreibungssprache formuliert und anschließend verarbeitet, um weitere Ähnlichkeiten bzw. Entsprechungen herauszustellen. Es handelt sich bei diesem Struktur-basierten Verfahren folglich um ein externes Verfahren. Die Idee [27] hinter diesem Verfahren ist, dass, wenn zwei Elemente übereinstimmen, sie auch die selbe Beschreibung haben.

2.4.3 Fazit

Mit den Ausführungen zum Schema-Matching wurden eine Reihe von Verfahren vorgestellt, die gerade im Rahmen des Matching von Schnittstellen funktionaler Dienste immer größere Bedeutung erfahren, um eine zur Laufzeit ausgeführte Auswahl und Komposition von Schnittstellen zum einen zu ermöglichen, zum anderen vor allem aber auch zu optimieren.
In Bezug auf die Visualisierung von Schnittstellen funktionaler Dienste durch Elemente einer Nutzerschnittstelle können vor allem die Element-basierten Verfahren ein wesentliches Potential zur Kopplung funktionaler Dienste und UI Services bieten. Die angeführten String-basierten Verfahren als eine Teilmenge der Element-basierten Verfahren lassen sich in dieser Hinsicht aus konzeptioneller, aber auch implementierungstechnischer Sicht am ehesten dafür in Betracht ziehen. Jedoch kann damit die Möglichkeit einer Kopplung nicht eindeutig prognostiziert werden. Infolgedessen kann möglicherweise eine Unterstützung durch Linguistische Verfahren mittels Zugriff auf externe Quellen hilfreich sein, um eine Kopplung funktionaler Dienste und UI Services zu ermöglichen.
Abschließend muss deutlich hervorgehoben werden, dass mit den an dieser Stelle vorgestellten Verfahren kein Anspruch auf Vollständigkeit erhoben werden kann. Vielmehr dient der zurückliegende Abschnitt dazu, grundlegende Verfahren im Bereich des Schema-Matching herauszustellen, wobei sich innerhalb einer Folgearbeit tiefgründiger mit dem aktuellen Stand der Forschung zu diesem Thema befasst werden sollte.

2.5 UI Integration, Komposition mit funktionalen Diensten

Nachdem in den vorangegangenen Abschnitten im Wesentlichen auf den State-of-the-Art funktionaler Dienste eingegangen wurde, soll mit dem vorliegenden Abschnitt auf einige wissenschaftlich relevante Konzepte im Bereich der Integration von UI Komponenten sowie der Komposition dieser Komponenten mit funktionalen Diensten verwiesen werden.

2.5.1 Mixup

Die Entwicklungs- und Laufzeitumgebung Mixup [31, 32] beschreibt die Erstellung von kompositen Webanwendungen durch Wiederverwendung bereits

existierender Komponenten auf Präsentationsebene. Mit dem Ziel der Wiederverwendung von Komponenten verfolgt Mixup den Ansatz zur Integration auf Präsentationsebene, bei dem die Integration von Komponenten durch Kombination der Präsentationslayouts, aber auch der Anwendungslogik sowie der Daten stattfindet.
Das Konzeptionelle Framework, das mit Mixup etabliert wird, beinhaltet ein abstraktes Komponentenmodell. Komponenten lassen sich dabei sowohl durch ihren Zustand als auch durch Zustandsänderungen sowie erweiterte Eigenschaften wie folgt beschreiben. Eine Komponente...:

- ...veröffentlicht Ereignisse über Zustandsänderungen,
- ...ermöglicht anderen Komponenten den Zugriff auf interne Operationen,
- ...lässt sich über Eigenschaften zur Entwicklungszeit, aber auch zur Laufzeit anpassen bzw. konfigurieren.

Weiterhin werden in Mixup sogenannte Darstellungsmodi angeführt, die es ermöglichen, das visuelle Erscheinungsbild einer Komponente (sichtbar, versteckt, minimiert, maximiert) und auch Informationen über den Lebenszyklus einer Komponente (instanziiert, bereit, beschäftigt, zerstört) zu verwalten.
Das in Mixup eingeführte Kompositionmodell wird durch die Spezifikation von Ereignis-*Listenern* umgesetzt. Dabei werden Ausgaben von Ereignissen einer Komponente auf Eingaben von Operationen anderen Komponenten abgebildet, d. h. ein Ereignis einer Komponente kann auf eine Operation anderer Komponenten verweisen.
Um Komponenten sowie die Komposition dieser zu modellieren, verwendet Mixup die *Extensible Presentation Integration Language* (*XPIL*), welche, in Anlehnung an WSDL für Web Services, über eine Reihe XML-basierter Elemente zur Beschreibung des Komponentenmodells verfügt und zudem XML-basierte Elemente anbietet, um das Modell der Komposition auszudrücken.

Die *Runtime Middleware* (*Presentation Integration Middleware*) dient letztlich der Ausführung der kompositen Anwendung durch die Interpretation der zugrunde liegenden, in XPIL beschriebenen Spezifikation der Anwendung. Durch die Middleware wird das abstrakte Komponentenmodell auf eine konkrete Komponentenimplementierung abgeleitet, wobei ein Komponenten-Adapter die Kommunikation zwischen Runtime Middleware und den funktionellen Komponenten über eine spezielle Technologie wie beispielsweise

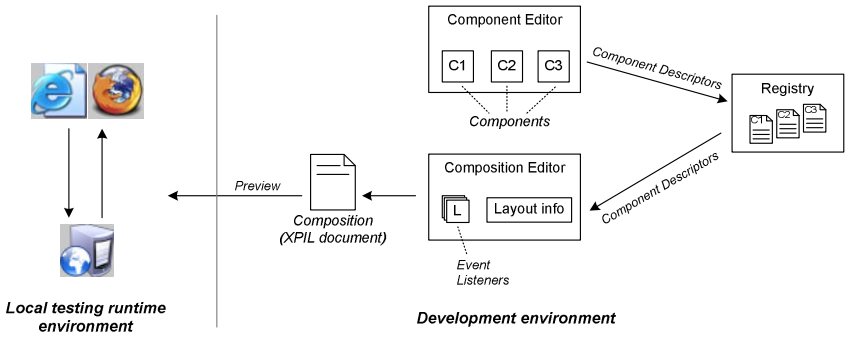


Abbildung 2.7 Überblick über die Mixup Entwicklungsumgebung (aus [31])

ActiveX oder Java Applet ermöglicht.
Mit Mixup wurde zudem ein Authorentool entworfen, der Mixup Editor, welcher als Hilfsmittel dient, um einen schnellen Entwicklungsprozess kompositer Anwendungen zu gewährleisten.

Bewertung

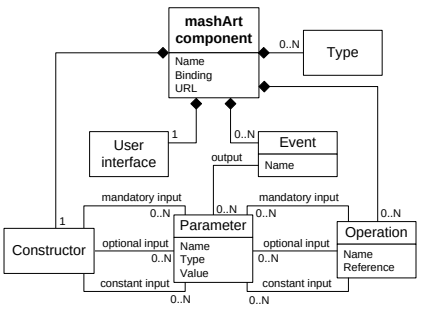
Die Entwicklungs- und Laufzeitumgebung Mixup befasst sich ausschließlich mit der Komposition wiederverwendbarer UI Komponenten. Dabei wird neben der clientseitigen Integration der Darstellungslogik von Komponenten außerdem auf die clientseitige Integration von Anwendungs- sowie Datenlogik fokussiert. Die Spezifikation von Ereignis-Listener bringt einen interessanten Ansatz mit sich, der für die Kommunikation von Daten zwischen UI Komponenten einer komponierten Anwendung weiter zu untersuchen wäre. Da sich das Thema der vorliegenden Arbeit jedoch auf die Kopplung heterogener, funktionaler Dienste und UI Komponenten bezieht, ist dieser Ansatz innerhalb dieser Arbeit lediglich von hintergründigem Interesse und sollte aufgegriffen werden, um das Ergebnis der Arbeit durch die Kommunikation zwischen UI Komponenten zu erweitern. Die mit XPIL eingeführte Spezifikation zur Beschreibung des Komponentenmodells sowie der Komposition ähnelt Ansätzen, die auch in anderen wissenschaftlichen Untersuchungen wie beispielsweise bei CRUISe (vgl. Kap. 2.1), verfolgt werden, wobei im Zusammenhang mit der vorliegenden Arbeit die Möglichkeit einer Komposition zur Laufzeit zu untersuchen ist und einer manuellen Beschreibung der Komposition an dieser Stelle keine große Bedeutung beizumessen ist.

2.5.2 mashArt

Unter der Bezeichnung *mashArt* [33, 34] wird die Entwicklung der Mixup-Konzepte vorangetrieben, wobei *mashArt* eine Erweiterung hin zu einem

universellen Konzept zur Komposition von UI Komponenten als auch von funktionalen Diensten darstellt.

Für mashArt wurde die Beschreibungssprache *mashArt Description Language* (*MDL*) eingeführt, die auf der für Mixup-Kompositionen definierten Beschreibung XPIL aufsetzt und im Wesentlichen WSDL zur Beschreibung von Web Services ähnlich ist. Die MDL wurde dahingehend erweitert, dass mit der Beschreibungssprache das erweiterte Komponentenmodell von mashArt abgedeckt wird und neben Ereignissen und Operationen, die eine Komponente charakterisieren, zudem Datentypen sowie ein Konstruktor definiert werden können. Abbildung 2.8 zeigt das Komponentenmodell von mashArt sowie ein Beispiel einer in MDL beschriebenen Komponente für die mashArt Plattform.



(a) UML class diagram of the UI component model.

```
<?xml version="1.0" encoding="utf-8" ?>
<mdl version="0.1">

 <component name="Policy" binding="component/UI"
            stateful="yes"  url="http://mashart.org/
            registry/234/PolicyBrowser/">

  <event name="PolicySelected">
   <output name="policy" type="xsd:string" />
  </event>

  <operation name="ShowPolicy" ref="showPolicy">
   <input name="policy" type="xsd:string"></input>
  </operation>
  <operation name="ShowProcess" ref="showProcess">
   <input name="proc" type="xsd:string"></input>
  </operation>

  <constructor>
   <input name="NumVisible" type="xsd:integer">5</input>
   <input name="StartPolicy" type="xsd:string" optional="yes"></input>
  </constructor>

 </component>
</mdl>
```

(b) MDL descriptor of the Policy component.

Abbildung 2.8 Überblick über das Klassendiagramm des mashArt Komponentenmodells (a) sowie ein Beispiel einer Komponente beschrieben in der mashArt Description Language (b) (aus [33])

Neben der Erweiterung der Beschreibungssprache wurde zudem das *mashArt Event Annotation* (*MEA*) Mikroformat eingeführt, um HTML-Elemente durch zusätzliche Attribute zu erweitern. Dabei lassen sich die durch Nutzerinteraktion ausgelösten Ereignisse mittels des MEA Mikroformates mit den Ereignissen von Komponenten verbinden.

Nachdem eine mashArt-Anwendung in MDL beschrieben und durch MEA annotiert wurde, lässt sich diese entsprechend zusammensetzen, wobei ein Objekt bzw. eine Funktionalität in JavaScript oder auch in einer anderen Technologie, wie Flash bzw. JavaFX, zur Ausführung der Komposition in der Laufzeitumgebung sowie der Zugriff auf die Anwendung in Form einer Schnittstelle (API) erzeugt wird. Neben Ereignissen, die durch die API behandelt werden, werden auch Operationen der UI Komponenten bereitgestellt. Eine Wrapper-Komponente interpretiert auftretende Operationsaufrufe als URIs für den Zugriff auf die Anwendung. Hierzu werden notwendige

Eingabe-Parameter für den Aufruf an die URI einer Operation angefügt und eine HTTP-Anfrage durchgeführt. Das Ergebnis der Anfrage und damit der Ausführung der aufgerufenen Operation wird anschließend genutzt, um eine UI Komponente mit den Ergebnis-Information zu initialisieren, woraufhin diese innerhalb eines HTML-Container Elementes (z. B. DIV, SPAN, IFRAME) angezeigt wird.

Bewertung

Da mashArt im Wesentlichen eine Erweiterung des Mixup Konzeptes darstellt, kann hier ähnlich wie in Abschnitt 2.5.1 argumentiert werden. Die wesentlichen Änderungen bringt das erweiterte Beschreibungsformat MDL mit sich, das in mashArt neben Operationen und Ereignissen auch die Beschreibung von Datentypen sowie von einem Konstruktor ermöglicht. Diese Überlegungen können neue Impulse für das in Kapitel 3.1 vorgeschlagene Beschreibungsformat im Hinblick auf das Thema der Arbeit bieten. Jedoch hat das von mashArt verfolgte Konzept keinen weiteren Einfluss auf den zentralen Punkt der Arbeit, die Systemarchitektur, da UI Komponenten und funktionale Dienste in mashArt bereits als eine Einheit verstanden werden und damit die Suche und Kopplung zur Laufzeit nicht mehr notwendig sind.

2.5.3 SOAUI Framework

Innerhalb der wissenschaftlichen Arbeit *Service-Oriented User Interface Modeling and Composition* [35] wird vorgeschlagen, die Entwicklung von Nutzeroberflächen in den Prozess der Entwicklung serviceorientierter Anwendungen einzubinden, indem der Nutzer als Anbieter einer Dienstleistung (Service-Provider) und seine Nutzerinteraktionen als eine Menge von Diensten modelliert werden.

Mit dem *SOAUI Framework* wird eine UI Service Registry eingeführt, um UI Services zu speichern und anzubieten sowie Discovery- und Matching-Prozesse ausführen zu können. Um UI Discovery und Matching zu unterstützen, müssen in Anlehnung an eine SOA die wesentlichen Merkmale einer UI in einer definierten Schnittstelle gekapselt und ausgedrückt werden können. Hierfür eignen sich formale oder semi-formale Spezifikationen, so dass vor allem die Datenkommunikationsmöglichkeiten einer UI, visuelle Komponenten sowie Layout-Eigenschaften, aber auch die Interoperabilität mit dem Anwendungsworkflow in dieser Schnittstelle beschrieben werden können.
Das vorgeschlagene SOAUI Framework fokussiert auf zwei wesentliche Punkte. Zum einen wird eine Application Template Registry definiert, welche

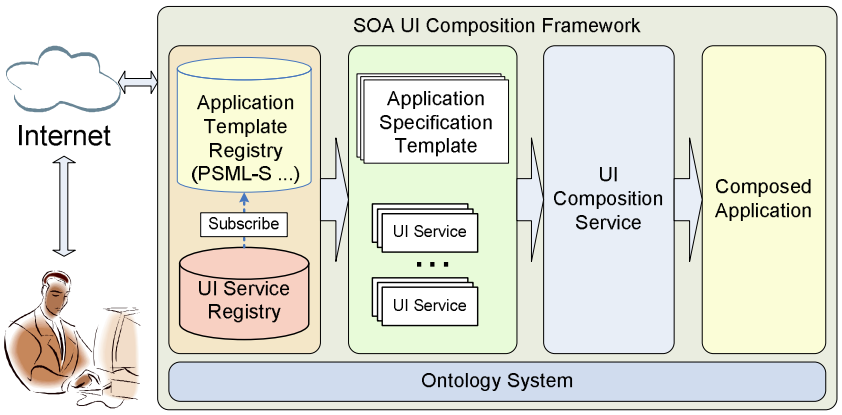


Abbildung 2.9 Überblick über das SOAUI Framework (aus [35])

Anwendungsspezifikationen in Form von sogenannten Templates speichert. Diese Templates enthalten zusätzlich Informationen zu den durch eine UI zu erbringenden Anforderungen (UI Requirements) sowie zu den Schnittstellen für den Zugriff bzw. die Komposition von Anwendung und UI Komponenten.

Zum anderen wird für das Framework eine UI Service Registry beschrieben, die einerseits die Speicherung von Beschreibungen serviceorientierter UIs ermöglicht, andererseits eine Schnittstelle für Entwickler bietet, um Discovery und Matching ausgewählter Templates von Anwendungsspezifikationen mit der UI Service Registry durchzuführen. Diese in der UI Service Registry abgelegten Beschreibungen enthalten sogenannte UI Profile, die durch einen UI Service Provider bei der Veröffentlichung von UIs mittels Referenzierung der Templates von Anwendungsspezifikationen erweitert werden können. Dadurch lässt sich der Prozess des Discovery und Matching dermaßen optimieren, dass bei der Auswahl eines Anwendungstemplates zunächst alle UIs innerhalb der UI Service Registry gefiltert und als kompatible UI Services angeboten werden, die dieses Template direkt referenzieren. Ein wiederholtes Durchsuchen der gesamten UI Service Registry ist in diesem Fall nicht notwendig und es wird eine möglichst schnelle Service-Komposition ermöglicht.

Um das UI Discovery und vor allem den Matching Prozess von Anwendungstemplates sowie UI Profile Beschreibungen weiter zu optimieren, wird außerdem die Verwendung eines Ontology-Systems vorgeschlagen. Dabei sind Beschreibungen zu Daten, visuellem Layout aber auch der Anwendungsworkflows durch semantische Informationen zu erweitern.

Bewertung

Das SOAUI Framework geht neben der Application Template Registry sowie der UI Service Registry mit dem Prozess des Discovery und Matching sowie der semantischen Erweiterung syntaktischer Beschreibung unter Einbeziehung eines Ontology-Systems auf sehr wichtige und zudem spezielle Punkte ein, auf die aktuelle Ansätze in diesem Bereich bisher nicht zu sprechen kommen. Auch die angeführte Erstellung von Anwendungsquellcodes durch einen entsprechenden Generator zur Laufzeit bietet einen Ansatz, der innerhalb der Konzeption für das Thema der vorliegenden Arbeit u. U. einen interessanten Betrachtungspunkt bietet. Leider wird mit der Vorstellung des Frameworks auf wesentliche Implementierungsdetails hinsichtlich des Discovery und Matching Prozesses, sowie dem Einfluss einer semantischen Erweiterung auf das vorgeschlagene Framework nicht näher eingegangen, was an dieser Stelle sehr wünschenswert wäre.

2.5.4 Weitere

Neben den genannten wissenschaftlichen Ansätzen sollen im folgenden Abschnitt weitere Lösungen auch aus dem Unternehmenssektor kurz vorgestellt werden.

2.5.4.1 IBM Mashup Center

Das IBM Mashup Center [37] ist eine kommerzielle Komplettlösung, die sich in ihrer Gesamtheit aus einer Reihe einzelner Anwendungen zusammensetzt, um unternehmensrelevante Informationen in UI Bestandteilen zu visualisieren und zu aggregieren.
IBM selbst beschreibt die Lösung kurz mit den folgenden Worten:

> *„IBM Mashup Center ist eine komfortable Mashup-Lösung für Unternehmen, die einzelnen Geschäftsbereichen die Zusammenstellung dynamischer, situationsbezogener Anwendungen ermöglicht - mit den für die IT erforderlichen Verwaltungs-, Sicherheits- und Governance-Funktionen.“*

Die wesentlichen Funktionen und Anwendungsbestandteile des IBM Mashup Center lassen sich wie folgt zusammenfassen:

- Zugriff auf eine Vielzahl relevanter Informationsquellen,
- Unterstützung der Entwicklung dynamischer Widgets über eine Desktopanwendung,

- Kombination und Transformation von Informationen zu sogenannten Feeds,
- Möglichkeit, so erstellte Feeds zu suchen sowie gemeinsam zu nutzten,
- Browsergestütztes Assembliertool zur Erstellung kompositer Nutzeroberflächen.

Vor allem die Möglichkeit, die zu Feeds aggregierten unternehmensrelevanten Informationen suchen zu können, sowie der browsergestützte Ansatz, diese Feeds in UI Bestandteile zu integrieren und zu kompositen Nutzeroberflächen zusammenzuführen, sind aufschlussreiche Ansätze für das Thema der vorliegenden Arbeit. Da es sich bei der Lösung von IBM jedoch um eine kommerzielle Lösung handelt, sind wissenschaftlich relevante Informationen zu den realisierten Funktionalitäten nicht zugänglich.

2.5.4.2 Open Mashups Studio

Bei Open Mashups Studio[13] handelt es sich um eine grafische Nutzeroberfläche, welche in Form eines Open Source Plugins die Funktionalitäten des Browsers Mozilla Firefox erweitert. Die als grafische Nutzeroberfläche realisierte Anwendungsumgebung ermöglicht eine Drag-und-Drop-basierte, Modell-getriebene Komposition von UI Bestandteilen sowie den über externe Dienstschnittstellen angebotene Ressourcen (z. B.: RSS-Feeds) zu sogenannten Mashup-Anwendungen, ohne zusätzliche Entwicklungserfahrung voraus zu setzen.
Ein wesentlicher Nachteil von Open Mashups Studio ist die bisher noch sehr überschaubare Menge an unterstützten Dienstschnittstellen und UI Bestandteilen; jedoch besteht über ein online verfügbares Repository die Möglichkeit, neu erstellte bzw. bereits umgesetzte Anwendungen anderen Nutzern zur Verfügung zu stellen.
Die Anwendungsumgebung selbst bietet verschiedene Funktionalitäten, die in unterschiedlichen Views arrangiert sind. Die interessantesten Views sind zum einen der Interface View, der die eigentliche Entwicklung der Nutzeroberflächen aus verfügbaren UI Bestandteilen ermöglicht, sowie dem Behavior View, der für die Spezifizierung der eigentlichen Komposition zwischen UI Bestandteilen und externen Schnittstellen aber auch der Definition des Datenflusses zwischen den so komponierten Komponenten zuständig ist.
Die so zu einer Komposition zusammengeführten UI Bestandteile und externen Ressourcen werden über integrierte Komponentenbibliothek innerhalb der Laufzeitumgebung in eine lauffähige Anwendung überführt.

[13] http://www.open-mashups.org

2.5.4.3 Corizon Plattform

Die von der Corizon Ltd. vertriebene kommerzielle Corizon Plattform [37] bietet eine Reihe von Werkzeugen, um komponierte, serviceorientierte Unternehmensanwendungen (Service-Oriented Business Applications, SOBAs) zu erstellen und diese zu verwalten. Die Grundlage dabei bilden UI Services als abstrakte, wiederverwendbare Elemente einer Benutzerschnittstelle, wobei UI Services die in einer SOA verwendeten, sonst nicht über eine Nutzeroberfläche verfügenden Dienste um den Aspekt der Visualisierung angebotener Dienstfunktionalitäten erweitern.
Die für das Thema der vorliegenden Arbeit aufschlussreichsten Bestandteile der Corizon Plattform sind die UI Service Library, der UI Service Provider sowie der Corizon Composer.
Die UI Service Library bietet Designzeitunterstützung für den Zugang zu allen bekannten UI Services. Ergänzend zu den angebotenen visuellen Repräsentationen von UI Services werden dabei die wesentlichen Informationen zu funktionellen Eigenschaften u. U. inkompatibler Dienste in einer maschinenlesbaren Schnittstelle gekapselt, um so eine einfache Integration der UI Services zu ermöglichen.
Der UI Service Provider dient der Ausführung eines UI Service zur Laufzeit über eine einheitliche Schnittstelle, denen die REST-Prinzipien zugrunde liegen.
Der Corizon Composer verfügt über eine Laufzeitumgebung, womit die erstellten Anwendungen für eine Nutzung veröffentlicht werden. Weiterhin bietet der Composer grundlegende Möglichkeiten zur Verwaltung und Überwachung von Anwendungen bzw. Anwendungsteilen zur Laufzeit sowie eine Nutzer- und Berechtigungsverwaltung.
Durch den proprietären und kommerziellen Ansatz lässt sich jedoch die Corizon Plattform im Allgemein sowie die zugrunde liegende Architektur im Speziellen aus wissenschaftlichen Gesichtspunkten nur sehr unzureichend tiefgründig betrachten.

2.5.5 Fazit

Die vorgestellten Konzepte gehen, teils mit sehr ähnlichen wissenschaftlichen, aber auch kommerziellen Ansätzen und Lösungen, einerseits auf die Integration und Komposition von UI Komponenten, andererseits auf die Komposition von funktionalen Diensten und UI Komponenten ein.
Neben der Einordnung in das Forschungsprojekt CRUISe zu Beginn dieses

Kapitels lassen sich mit den hier vorgestellten Konzeptideen grundlegende Anforderungen an eine zu entwickelnde Systemarchitektur formulieren, was Teil des folgenden Kapitels sein soll.
Zusammenfassend lässt sich dabei festhalten, dass die Notwendigkeit eines Discovery und Matching Prozesses besteht, um potentielle Kandidaten von UI Komponenten zu identifizieren, die an funktionale Dienste gekoppelt werden können. Das SOAUI Framework bietet zudem Ansätze, einen solchen Discovery Prozess durch zusätzliche Informationen zu erweitern und damit zu optimieren bzw. den Aufwand für das Discovery zu minimieren.
Auch die angeführten Beschreibungsformate sowohl für UI Komponenten als auch einer Komposition als solche können als wesentliche Anforderungen für ein Konzept zur Kopplung von funktionalen Diensten und UI Services gesehen werden. Sowohl die in mashArt verwendete MDL-Beschreibung als auch die eingangs zu CRUISe angesprochene UISDL-Beschreibung definieren wesentliche Elemente zur Beschreibung von UI Komponenten. Das SOAUI Framework fasst den Punkt für das Thema der vorliegenden Arbeit sehr gut zusammen, wonach die Notwendigkeit einer (semi-)formalen Spezifikation besteht, so dass funktionelle, visuelle als auch kontextuelle Eigenschaften in einer Schnittstellen beschrieben werden können.
Da für die vorliegende Arbeit bisher noch kein Beschreibungsformat für UI Services bzw. UI Komponenten vorliegt, das alle notwendigen Kriterien und Merkmale abdeckt, um die Kopplung funktionaler Dienste und UI Services zur Laufzeit einer Systemanwendung zu unterstützen, wird dieser Umstand zu Beginn der sich im Folgekapitel anschließenden Konzeption erneut aufgefasst.

2.6 Zusammenfassung

In diesem Kapitel wurden zunächst grundlegende Begriffe geklärt, die in den nachfolgenden Kapiteln verwendet werden. Weiterhin wurde auf wichtige Konzepte und Lösungsansätze eingegangen, die dem Thema der vorliegenden Arbeit zugeordnet werden können.
Die wesentlichen Erkenntnisse in den Bereichen der Beschreibung von funktionalen Diensten und UI Komponenten, des Matching unterschiedlicher Schemata sowie der UI Integration und Erstellung kompositer Anwendungen bilden dabei die Basis, um im folgenden Kapitel ein Konzept für das Matching von Schnittstellen und die Kopplung von funktionalen Diensten und UI Services zu entwerfen.

Kapitel 3

Konzeption

Vor dem Hintergrund der Einbettung der vorliegenden Arbeit in den Kontext des Forschungsprojektes CRUISe (vgl. Kapitel 2.1) wird innerhalb der vorliegenden Konzeption zunächst auf die Formulierung wesentlicher Beschreibungsmerkmale und -kriterien eingegangen, da das für CRUISe in [3] angeführte Beschreibungsformat UISDL den Fokus der Betrachtungen auf die visuelle Repräsentation von UI Komponenten legt und dabei nur marginal auf die Betrachtung von Kriterien eines Beschreibungsformates eingeht, welche für das Thema der vorliegenden Arbeit vom Autor als notwendig erachtet werden.
Das vorliegende Kapitel gliedert sich in zwei wesentliche Bereiche. Zum einen geht es um die konzeptionelle Formulierung eines Beschreibungsformates für UI Services sowie die darin gekapselten UI Komponenten. Zum anderen soll eine Systemarchitektur konzipiert werden, die auf die dynamische Kopplung von heterogenen, funktionalen Diensten und UI Service bzw. UI Komponenten ausgelegt ist.
Nach Vorstellung dieser Bereiche wird anhand eines beispielhaften Szenarios der Ablauf und die Arbeitsweise der innerhalb der Systemarchitektur vorgestellten Elemente kurz erläutert.

3.1 Beschreibungsformat für UI Komponenten

Technologien zur syntaktischen Beschreibung von UI Services bzw. UI Komponenten sind im Wesentlichen nicht vorhanden und, wie bereits angesprochen, Teil aktueller wissenschaftlicher Untersuchungen (vgl. Kapitel 2.1 sowie 2.5). In den folgenden Abschnitten sollen Elemente und Bestandteile eines Formates zur Beschreibung von UI Services und darin gekapselten UI Komponenten herausgestellt werden. Mit den Abschnitten 3.1.2 sowie

3.1.3 werden dabei neben allgemeinen Merkmalen auch mögliche erweiternde Merkmale herausgestellt, die UI Komponenten innerhalb eines UISDL-Beschreibungsdokumentes für UI Services näher spezifizieren.

3.1.1 Vorbemerkungen

Bei der Kopplung von funktionalen Diensten und UI Komponenten geht es in der Regel darum, Daten bzw. Informationen, die durch Objekte repräsentiert werden, zwischen angebotenen Eingabe- und Ausgabe-Schnittstellen auszutauschen.
Die Schnittstellen funktionaler Dienste, die eine zugrunde liegende Funktionalität repräsentieren, können allgemein auch als Operationen bezeichnet werden, die über einen Dateneingang, einen Datenausgang sowie einen Fehlerausgang genauer definiert sein können.
Auch UI Komponenten verfügen im Allgemeinen über öffentlich zugängliche Operationen, die sich in ihrer Gesamtheit als Schnittstelle einer Komponente definieren lassen. Anders als die bereits angeführten Operationen funktionaler Dienste wird innerhalb der Arbeit für Operationen von UI Komponenten festgelegt, dass diese über einen Dateneingang verfügen können, der in der Lage ist, Daten bzw. Fehler der Ausgänge eines funktionalen Dienstes aufzunehmen. Weiterhin wird festgelegt, dass UI Komponenten über einen Datenausgang verfügen können, an dem Daten bzw. Informationen bereitgestellt werden, die an den Dateneingang eines funktionalen Dienstes kommuniziert werden.
Abbildung 3.1 verdeutlicht diese Ausführungen zur Kopplung der Schnittstellen von funktionalen Diensten und UI Komponenten grafisch.

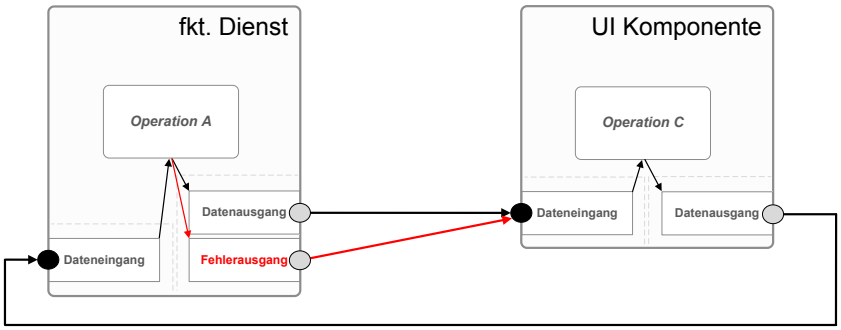


Abbildung 3.1 Gegenüberstellung von Schnittstellen funktionaler Dienste und UI Komponenten

Neben der Kommunikation von Daten und Informationen kann es zudem

notwendig sein, eine UI Komponente zu konfigurieren. Dazu sollte jede UI Komponente über einen Dateneingang zur Konfiguration verfügen. Dieses allgemeine Merkmal jeder UI Komponente hat jedoch keinen Mehrwert für die syntaktische Beschreibung einer Komponente und kann aus diesem Grund innerhalb der Beschreibung vernachlässigt werden. In Kapitel 3.1.3 wird dieser Aspekt einer UI Komponente jedoch nochmals genauer betrachtet.

An diesen allgemeinen Vorbemerkungen wird sehr gut deutlich, dass sich Beschreibungsmerkmale für eine Kopplung von Schnittstellen wesentlich mehr auf Merkmale zur Beschreibung von Operationen sowie Parametern beziehen müssen und damit nicht zu allgemein definiert sein sollten, worauf in den folgenden Abschnitten näher eingegangen wird.

3.1.2 Allgemeine Beschreibungsmerkmale

Über allgemeine Beschreibungsmerkmale ist der Entwickler einer UI Komponente zum einen in der Lage, notwendige Elemente und deren Attribute zu definieren, zum anderen ist er in der Lage, eine Abgrenzung zu anderen Komponenten zu formulieren. Das einleitende Element einer UI Komponente wird mit *<uic>* definiert, worin alle in den folgenden Abschnitten beschriebenen Elemente zur allgemeinen Beschreibung einer UI Komponente gekapselt werden.

```
1 <uic>
2   // kapselt Elemente einer UI Komponente
3 </uic>
```

Listing 3.1 Element <uic>

3.1.2.1 Abgrenzungskriterien

Um eine UI Komponente von anderen UI Komponenten abzugrenzen sowie im Anwendungsverlauf identifizieren zu können, ist die Deklaration entsprechender *Abgrenzungskriterien* notwendig, die innerhalb der vorliegenden Arbeit auch als Meta-Informationen zusammengefasst werden können.

```
1 <meta>
2   // kapselt Meta-Informationen einer UI Komponente
3 </meta>
```

Listing 3.2 Element <meta>

Die im Folgenden definierten Abgrenzungskriterien, die Informationen über eine UI Komponente zur Verfügung stellen und diese somit näher spezifizieren, werden innerhalb des Elementes <*meta*> (vgl. Listing 3.2) gekapselt.

- <*name*> gibt den Namen einer UI Komponente an, der über das Attribut *id* definiert wird.
- <*version*> gibt die Version einer UI Komponente an, die über das Attribut *id* definiert wird.
- <*author*> liefert Informationen über den Entwickler (Person, Organisation) einer UI Komponente. Über das Attribut *name* wird der Name des Entwicklers definiert. Das Attribut *email* dient der Angabe einer E-Mail Adresse für eine eventuelle Kontaktaufnahme mit dem Entwickler. Das Attribut *url* dient der Angabe einer eindeutigen URI, die eine direkte Verbindung mit dem Autor spezifiziert (vgl. [3]).
- <*constructor*> spezifiziert über das Attribut *name* den Namen des Konstruktors der Komponente, um diese zu initialisieren.
- <*component*> kann über das Attribut *href* bspw. auf eine HTML-Seite referenzieren, die eine Beispiel-Implementierung der Komponente zur Verfügung stellt.
- <*description*> enthält eine Beschreibung in menschenlesbarer Form zu einer mit dem Attribut *operation* innerhalb der UI Komponente definierten Operation. Durch das Attribut *lang* wird zudem die Möglichkeit eingeräumt, eine Beschreibung in unterschiedlichen Sprachen angeben zu können.
- <*screenshot*> referenziert über das Attribut *href* ein Vorschaubild zu einer mit dem Attribut *operation* innerhalb der UI Komponente definierten Operation. Das Attribut *title* dient dabei der Angabe einer Bildbeschreibung.

Eine beispielhafte Beschreibung der zuvor angeführten Abgrenzungskriterien einer UI Komponente wird in Listing 3.3 vorgestellt.

```
1 <meta>
2   <name id="WeatherUIC" />
3   <version id="0.0.1" />
4   <author name="Firstname␣Lastname" email="author@email.de" url="http://www.
        authorpage.com/uis_components" />
5   <description operation="showResult" lang="de">
6     Diese Operation zeigt Wetterdaten zu einer Stadt an.
7   </description>
8   <screenshot operation="showResult" href="http://www.authorpage.com/
        uis_components/weatherUIService/0.0.1/showResultResponse.png" title="
        Weather␣Response" />
9 </meta>
```

Listing 3.3 Abgrenzungskriterien einer UI Komponente

An dieser Stelle könnte nun argumentiert werden, dass u. a. auch ein Element zur Klassifikation einer UI Komponente innerhalb der Meta-Informationen notwendig sein kann. Diesem Argument muss hier jedoch widersprochen werden. Der Autor vertritt innerhalb der vorliegenden Arbeit die Auffassung,

dass eine UI Komponente nicht nur auf ein einfaches Visualisierungsszenario ausgelegt sein muss. Vielmehr können über die öffentlich zugänglichen Operationen, d.h. den darin gekapselten Funktionalitäten, u. U. mehrere, auch sehr komplex gestaltete UI-Bestandteile angeboten werden, die einzeln, aber auch in ihrer Gesamtheit innerhalb eines Anwendungsszenarios zum Einsatz kommen können. Es kann also sinnvoll sein, nicht eine UI Komponente als solche direkt zu klassifizieren, sondern vielmehr jede angebotene Operation bezüglich ihrer Verwendung einzuordnen. Hinsichtlich einer möglichen Klassifikation wird hierbei auf den Abschnitt 3.1.3.3 verwiesen, in dem auf die Klasse Domain-spezifischer UI Komponenten eingegangen wird.

3.1.2.2 Inhaltskriterien

Zu den *inhaltlichen Kriterien* der Beschreibung einer UI Komponente zählen neben der Angabe von *Layout-Eigenschaften* auch Referenzen zu *Script-Bibliotheken*. Da UI Komponenten in der Regel auf unterschiedliche, heterogene, u. U. durch Drittanbieter bereitgestellte Ressourcen zugreifen, um ihren vollen visuellen als auch funktionellen Umfang zu erreichen, ist es notwendig, die Referenzierung unterschiedlicher Ressourcen zu gewährleisten.

```
1 <content>
2   // kapselt Inhalt/Ressourcen einer UI Komponente
3 </content>
```

Listing 3.4 Element <content>

Referenzen zu den Ressourcen werden dabei innerhalb des Elementes <*content*> gekapselt (vgl. Listing 3.4) und lassen sich durch die folgenden Elemente differenzieren:

- <*initial*> bezieht sich auf Layout-Eigentschaften oder Script-Bibliotheken der UI Komponente selbst. Über das Attribut *type* (z.b. javascript, css) lässt sich der Typ der Ressource festlegen. Das Attribut *href* ermöglicht die Angabe einer eindeutigen URI, über die eine Ressource direkt referenziert wird.
- <*source*> bezieht sich auf notwendige Layout-Eigentschaften oder Script-Bibliotheken, die u. a. auch durch Drittanbieter bereitgestellt und von der UI Komponente selbst verwendet werden. Über das Attribut *type* (z.b. javascript, css) lässt sich auch hier der Typ der Ressource festlegen. Mit dem Attribut *href* wird die Ressource über eine eindeutige URI direkt referenziert.

Eine beispielhafte Beschreibung möglicher Inhaltskriterien einer UI Komponente wird in Listing 3.5 vorgestellt.

```
<content>
  <initial type="javascript" href="http://www.authorpage.com/uis_components/
      weatherUIService/0.0.1/js/weatherUIS.js" />

  <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/build/
      yahoo-dom-event/yahoo-dom-event.js" />
  <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/build/
      container/container-min.js" />
  ...
  <source type="css" href="http://yui.yahooapis.com/2.7.0/build/container/
      assets/skins/sam/container.css" />
  <source type="css" href="http://yui.yahooapis.com/2.7.0/build/fonts/fonts-
      min.css" />
  ...
</content>
```

Listing 3.5 Inhaltskriterien einer UI Komponente

3.1.2.3 Funktionelle Kriterien

Unter den *funktionellen Kriterien* einer UI Komponente, die innerhalb der UISDL-Beschreibung zu spezifizieren sind, werden *öffentliche Operationen und Ereignisse* gesehen, auf die ein direkter Zugriff möglich ist. Diese Operationen und Ereignisse können, wie eingangs bereits angedeutet, der Bereitstellung aber auch Aufnahme von Daten und Informationen dienen und können als *öffentliche Schnittstelle* einer Komponente bezeichnet werden.

```
<operations>
  // kapselt Operationen einer UI Komponente
</operations>
```

Listing 3.6 Element <operations>

Öffentliche Operationen werden dabei durch das Element <*operations*> (vgl. Listing 3.6) gekapselt. Jede Operation wird zudem über das Element <*operation*> sowie unter Angabe von Parametern der Dateneingabe bzw. der Datenausgabe für eine Operation wie folgt definiert:

- <*request*> bezieht sich auf die Dateneingabe einer Operation. Mögliche Eingabe-Parameter werden durch das Attribut *name* näher spezifiziert. Das Attribut *type* dient der Angabe eines Datentyps für den jeweiligen Parameter.
- <*response*> bezieht sich auf die Datenausgabe einer Operation. Mögliche Ausgabe-Parameter werden durch das Attribut *name* näher spezifiziert. Das Attribut *type* dient der Angabe eines Datentyps für den jeweiligen Parameter.

Eine beispielhafte Beschreibung möglicher Operationen einer UI Komponente unter Angabe von Dateneingabe bzw. Datenausgabe wird in Listing

3.7 vorgestellt.

```
<operations>
  <operation name="getXYZ">
    <response name="var_name" type="xsd:int" />
  </operation>
  <operation name="setXYZ">
    <request name="var_name" type="xsd:string" />
  </operation>
</operations>
```

Listing 3.7 Operationen einer UI Komponente

Um die Interoperabilität und Plattformunabhängigkeit zu gewährleisten, wird zur Definition von Datentypen allgemein die Verwendung des XML-Schema vorgeschlagen. Neben der hier aufgezeigten *Definition einfacher Datentypen* werden Parameter für Datenein- und Datenausgabe von Operationen oftmals auch durch *komplex definierte Datentypen* erweitert. Der Abschnitt 3.1.3.1 geht auf diese Erweiterung von Parameter-Beschreibungen für Operationen von UI Komponenten detaillierter ein.

Öffentliche Ereignisse dienen gegenüber den angeführten öffentlichen Operationen in erster Linie dazu, Daten bzw. Informationen zwischen UI Komponenten eines Anwendungsszenarios zu vermitteln. Damit werden beschriebene Ereignisse vor allem dazu verwendet, UI Komponenten bzw. deren Operationen miteinander zu koppeln. Dieses Vorgehen liegt jedoch nicht im Fokus der vorliegenden Arbeit und ist damit in einer weiteren wissenschaftlichen Arbeit gesondert zu betrachten. Aus diesem Grund wird auf die weitere Beschreibung öffentlicher Ereignisse an dieser Stelle verzichtet.

3.1.3 Erweiterte Beschreibungsmerkmale

Neben den bis hierher angeführten allgemeinen Beschreibungsmerkmalen von UI Komponenten soll im Folgenden auf einige Beschreibungsmerkmale eingegangen werden, die zu einer erweiterten Beschreibung von UI Komponenten führen. Dabei geht es in diesem Abschnitt vor allem um die komplexe Definition von Datentypen für Parameter, die Unterscheidung zwischen starren und flexiblen Operationen von UI Komponenten sowie Möglichkeiten der Beschreibung Domain-spezifischer UI Komponenten.

3.1.3.1 Definition komplexer Datentypen

Wie bereits in Abschnitt 3.1.2.3 angeführt, kann es notwendig sein, die Parameter der Operationen von UI Komponenten komplexer zu beschreiben und damit die Möglichkeit zu schaffen, neben einfachen, auf XML-Schema basierenden Datentypen auch komplex strukturierte Datentypen zu definieren, die als Dateneingabe an eine Operation übergeben bzw. als Datenausgabe durch eine Operation angeboten werden können.
Für diese Betrachtung wird das Element <*types*> eingeführt (vgl. Listing 3.8), das zur erweiterten Beschreibung von Parametern verwendet werden kann. Innerhalb dieses Elementes lassen sich komplexe Datentyp-Definitionen anführen, die sowohl einfache als auch komplexe Datentypen beschreiben und somit zu neuen, komplex geschachtelten Datentypen führen können.

```
1 <types>
2   // kapselt Datentyp-Definitionen eines UI Services
3 </types>
```

Listing 3.8 Element <operation>

Das Element <*types*> wird dabei neben dem Element <*uis*> als weiteres Kind-Element innerhalb des mit dem Element <*uisdl*> eingeführten UI Service Beschreibungsdokumentes angeführt (vgl. in [3]). Das in Listing 3.9 angeführte Beispiel verdeutlicht die Definition eines komplexen Datentyps „RequestVar“, der als Ausgabe-Parameter der Operation getXYZ beschrieben ist.

```
1 <types>
2   <xsd:schema targetNamespace="http://www.example.org/Example/">
3     <xsd:complexType name="RequestVar">
4       <xsd:sequence>
5         <xsd:element name="var1" type="xsd:string" />
6         <xsd:element name="var2" type="xsd:ing" />
7         <xsd:element name="var3" type="xsd:float" />
8       </xsd:sequence>
9     </xsd:complexType>
10  </xsd:schema>
11 </types>
```

Listing 3.9 Beschreibung des komplexen Datentyps „RequestVar“

Für eine ausführliche Definition komplexer Datentypen sei aufgrund der Übersichtlichkeit an dieser Stelle auf die im Anhang angeführten Beispiele in Listing A.4 sowie Listing A.5 verwiesen.

3.1.3.2 Starre vs. flexible Operationen

Die Parameter der Dateneingabe bzw. Datenausgabe einer UI Komponente lassen sich, wie bereits angeführt, sowohl einfach als auch komplex definieren, womit die Struktur der Daten festgelegt ist, die einer Operation übergeben werden kann bzw. die durch eine Operation angeboten wird. Eine solche Art der Beschreibung einer Operation lässt sich allgemein auch als *starre Operation* bezeichnen.
Im Gegensatz dazu kann eine Operation einer UI Komponente funktionell derart implementiert sein, dass sie in der Lage ist, ihre visuelle Repräsentation generisch auf die Struktur der übergebenen Daten anzupassen.

```
1 <operations>
2   <operation name="getXYZ" flexible>
3     <response name="var_name" />
4   </operation>
5 </operations>
```

Listing 3.10 Verwendung des Attributes *flexible* zur Erweiterung von Operationen

Damit werden die bereits angeführten funktionellen Kriterien zur Beschreibung von Operationen von UI Komponenten dahingehend erweitert, dass eine syntaktische Unterscheidung zwischen starren und *flexiblen Operationen* kenntlich gemacht werden kann.
Sollte es sich bei einer Operation um eine flexible Operation handeln, so ist die Beschreibung des Elementes *<operation>* durch das leere Attribut *flexible* zu erweitern. Man könnte bei diesem optionalen Attribut auch von einem sogenannten *Flag* sprechen, das die funktionelle Flexibilität einer Operation kennzeichnet. Listing 3.10 verdeutlicht eine mögliche Verwendung des Attributes *flexible* zur Kennzeichnung einer flexiblen Operation.
Es kann festgehalten werden, dass im Falle der syntaktischen Erweiterung einer Operation hin zu einer flexiblen Operation das Attribut *type* zur direkten Datentyp-Angabe bzw. Referenzierung einer Datentyp-Definition an dieser Stelle entfallen kann, was sich aus der funktionellen Flexibilität der Operation gegenüber einer übergebenen Datenstruktur ergibt.
An einem kurzen Anwendungsszenario soll die Verwendung des Attributes und damit die Beschreibung flexibler Operationen nochmals näher verdeutlicht werden. Beispielweise könnte eine UI Komponente über eine Operation verfügen, die anhand einer bekannten Datenstruktur (Datentyp-Definition) ein Formular als UI Element zur Verfügung stellt. Die Datenausgabe, d. h. die eingegebenen und anschließend am Datenausgang der Komponente be-

reitgestellten Informationen dieses flexiblen Formulars könnten an den Dateneingang einer Operation eines funktionalen Dienstes übergeben werden, was im Wesentlichen einer Kopplung der Schnittstellen entspricht. Um das flexible Formular nun so darzustellen, dass es alle Informationen der Dateneingabe der Operation des funktionalen Dienstes abbildet, also entsprechend notwendige Eingabefelder visualisiert, muss die Datentyp-Definition dieser Dateneingabe des funktionalen Dienstes jedoch zuvor an die UI Komponente übermittelt werden. An dieser Stelle kommt wieder der bereits angeführte Dateneingang zur Konfiguration einer UI Komponente zum Tragen, der auch zur Konfiguration einer flexiblen Operation verwendet werden kann. Stehen die Datentyp-Definitionen über die Konfigurationsschnittstelle einer UI Komponente zur Verfügung, so kann das Formular visuell auf diese Definition hin realisiert werden.
Hier lassen sich auch weitere Beispiele für flexible Operationen anführen, jedoch liegt dieser Aspekt nicht im Fokus der vorliegenden Arbeit und sollte als Teilaspekt in weiteren wissenschaftlichen Arbeiten erneut aufgegriffen werden.

3.1.3.3 Domain-spezifische Erweiterung

Neben den bisher angeführten Merkmalen und Kriterien des Beschreibungsformates UISDL, mit denen sich *nicht Domain-spezifische* UI Komponenten syntaktisch beschreiben lassen, soll im Folgenden auf Erweiterungen zur Beschreibung *Domain-spezifischer* UI Komponenten eingegangen werden. In dieser Klasse lassen sich UI Komponenten zusammenfassen, die durch zusätzliche Beschreibungsmerkmale einerseits direkt auf die Operation eines speziellen funktionalen Dienstes, andererseits auf eine Klasse von funktionalen Diensten verweisen. Mit Abbildung 3.2 werden diese Erweiterungen grafisch verdeutlicht.

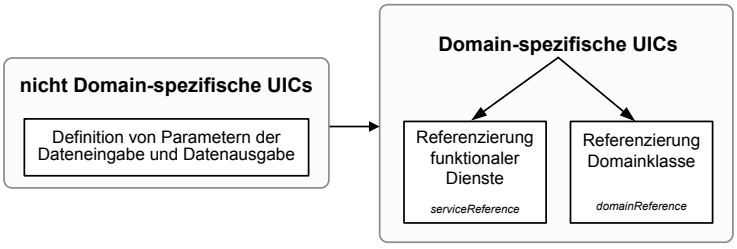


Abbildung 3.2 Erweiterung nicht Domain-spezifischer hin zu Domain-spezifischen UI Komponenten

Neben den jeweiligen syntaktischen Merkmalen sollen im Folgenden angeführte beispielhafte Quellcode-Listings dazu dienen, die Zweckmäßigkeit sowie die Verwendung dieser erweiterten Merkmale herauszustellen.

Direkte Referenzierung eines funktionalen Dienstes

Durch Angabe eines Verweises auf die Ein- bzw. Ausgabe-Parameter einer Operation eines funktionalen Dienstes innerhalb der Beschreibung von Ein- bzw. Ausgabe-Parametern einer Operation einer UI Komponente lassen sich funktionale Dienste direkt referenzieren. Damit kann auf die Interoperabilität zwischen den Schnittstellen der Operation einer UI Komponenten sowie der Operation eines funktionalen Dienstes verwiesen werden. Diese Art der direkten Referenzierung kann den Prozess der Suche nach kompatiblen zu koppelnden Ein- bzw. Ausgabe-Operationen wesentlich vereinfachen. Neben einer direkt referenzierten Schnittstelle eines funktionalen Dienstes besteht jedoch zusätzlich die Möglichkeit der Kopplung einer so erweitert beschriebenen Operation einer UI Komponente an nicht direkt referenzierte Operationen weiterer funktionaler Dienste.

```
1 <operation name="operationABC">
2   <request name="var_name" type="xsd:string" serviceReference="http://www.
        example1.org/example.wsdl#OperationName1#ResponseType"/>
3   <response name="var_name" type="xsd:string" serviceReference="http://www.
        example2.org/example.wsdl#OperationName5#RequestType"/>
4 </operation>
5 <operation name="operationXYZ">
6   <request name="var_name" type="xsd:string" serviceReference="http://www.
        example1.org/example.wsdl#OperationName2#FaultType"/>
7 </operation>
```

Listing 3.11 Verwendung des Attributes *serviceReference*

Das in Listing 3.11 angeführte Beispiel zeigt die Referenzierung von Ein- bzw. Ausgabe-Parametern verschiedener Operationen funktionaler Dienste durch das in diesem Zusammenhang eingeführte Attribut *serviceReference*. Das Attribut nimmt dabei einen Verweis auf eine Operation eines funktionalen Dienstes auf, die sich in drei Teilabschnitte, getrennt durch das Zeichen #, gliedert.

Der erste Teilabschnitt verweist dabei direkt auf das Beschreibungsdokument des funktionalen Dienstes. Mit dem zweiten Teilabschnitt wird der Name einer Operation innerhalb dieses referenzierten Beschreibungsdokumentes angegeben. Der dritte Teilabschnitt bezieht sich auf die Dateneingabe, Datenausgabe bzw. Fehlerausgabe, die durch eine Operation eines funktionalen Dienstes angeboten wird, und die entsprechend Abbildung 3.1 an die so erweiterte Dateneingabe bzw. Datenausgabe der Operation einer UI

Komponente gekoppelt werden soll.

Referenzierung einer gemeinsamen Domain-Klasse

Die im Folgenden angeführte Domain-spezifische Erweiterung bezieht sich, anders als die direkte Referenzierung durch das Attribut *serviceReference*, auf die Referenzierung von Domain-Klassen. Domain-Klassen entsprechen dabei speziellen Kategorien einer übergeordneten, *gemeinsamen Klassifikation*, die sowohl innerhalb der Dateneingabe, Datenausgabe bzw. Fehlerausgabe von Operationen funktionaler Dienste als auch innerhalb der Dateneingabe bzw. Datenausgabe von Operationen von UI Komponenten durch ein entsprechendes Attribut referenziert werden können.
Ein Beispiel soll die Verwendung des Attributes verdeutlichen. Eine Operation eines funktionalen Dienstes liefert über die Datenausgabe geografische Koordinaten in Form von der Parameter für Längengrad sowie Breitengrad zu einer bestimmten Stadt. Die Dateneingabe einer Operation einer UI Komponente nimmt Parameter für Längengrad und Breitengrad auf und stellt diese in einer Karte grafisch dar. Unter diesem Gesichtspunkt ließen sich beide Schnittstellen einer speziellen Kategorie für das Anbieten bzw. Verarbeiten geographischer Daten zuordnen, die mit der Abkürzung „MAP“ gekennzeichnet sein könnte.
Das folgenden Listing 3.12 zeigt ein mögliches Szenario für die Verwendung eines Attributes *domainReference* zur Erweiterung von Eingabe- bzw. Ausgabe-Parametern der Operation *operationABC* einer UI Komponente.

```
1 <operation name="operationABC">
2   <request name="var_name" type="xsd:string" domainReference="http://www.
        example1.org/domainReference.xml#Categorie#Subcategorie"/>
3   <response name="var_name" type="xsd:string" domainReference="http://www.
        example2.org/domainReference.xml#Categorie#Subcategorie"/>
4 </operation>
```

Listing 3.12 Verwendung des Attributes *domainReference*

Wie das Listing zeigt, wird getrennt durch das Zeichen „#“ neben einer Hauptkategorie („Categorie“) zudem eine Unterkategorie („Subcategorie“) angegeben. Durch dieses spezielle Vorgehen lassen sich sehr feingranulare Angaben zu bestehenden, u. U. sehr umfangreichen Klassifikationen in die syntaktische Beschreibung einbinden.
Neben der Einführung dieses Attributes zur Referenzierung einer Domain-Klasse innerhalb der Beschreibung von Operationen auf Seite der UI Komponenten sind Untersuchungen zur Einführung eines entsprechenden Attributes seitens der Beschreibungen funktionaler Dienste notwendig.

Sowohl Möglichkeiten zur Erweiterung der Beschreibung funktionaler Dienste als auch notwendige Domain-spezifische Klassifikationen stehen zum Zeitpunkt der Themenbearbeitung nicht zur Verfügung und sollten daher Teil weiterer wissenschaftlicher Arbeiten sein. Die Ausführungen zum Attribut *domainReference* sollten an dieser Stelle den möglichen Nutzen hervorheben, werden innerhalb der folgenden Kapitel aus den genannten Gründen jedoch vernachlässigt und nicht weiter betrachtet.

Allgemein spiegelt dieses Vorgehen der erweiterten Beschreibung nicht Domain-spezifischer UI Komponenten einige wesentliche Aspekte der semantischen Erweiterung funktionaler Dienste wieder, wie sie bereits im Kapitel 2.3 angeführt wurden.

3.1.4 Fazit

Diese Ausführungen zum Beschreibungsformat für UI Komponenten innerhalb der Thematik der vorliegenden Arbeit fokussieren deutlicher auf die Betrachtung funktioneller, aber auch informeller Eigenschaften. Der folgende Abschnitt befasst sich mit der Systemarchtiktur, in der das vorgeschlagene Beschreibungsformat eine wesentliche Rolle spielt.

3.2 Systemarchitektur

Nachdem im zurückliegenden Kapitel auf notwendige Merkmale und Kriterien eines Beschreibungsformates von UI Komponenten eingegangen wurde, soll in diesem Abschnitt eine Systemarchitektur entwickelt werden, die eine Kopplung von funktionalen Diensten und UI Services ermöglicht. Dabei werden zunächst grundlegende Anforderungen an die Systemarchitektur formuliert. Anschließend werden konkrete Elemente innerhalb dieser Systemarchitektur vorgestellt, bevor deren Einteilung in die Bereiche System-Anwendung sowie Client-Anwendung und die damit verbundenen Aufgaben und Abläufe beschrieben werden.

3.2.1 Anforderungen

Um einen Einstieg in das Konzept einer Systemarchitektur zu vermitteln, ist es zunächst notwendig, die Anforderungen an die zu konzipierende Architektur zu formulieren. Dabei wird einerseits auf die funktionalen Anforderungen eingegangen, mit denen die wesentlichen Ansprüche an die zu konzipierende

Architektur definiert werden. Weiterhin werden nicht-funktionale Anforderungen angeführt, die die wichtigsten Eigenschaften der Architektur näher verdeutlichen.

3.2.1.1 Funktionale Anforderungen

Zu den funktionalen Anforderungen zählen:

[R1] Allgemeingültigkeit
Die zu konzipierende Architektur sollte eine Vielzahl von Formaten zur Beschreibung funktionaler Dienste (vgl. Kapitel 2.2) aber auch zur Beschreibung von User Interface Services unterstützen und verarbeiten können.

[R2] Discovery
Die zu entwickelnde Architektur sollte über Algorithmen verfügen, die eine Suche und Auswahl (Discovery) von UI Komponenten zu ausgewählten funktionalen Diensten ermöglichen. Diese Algorithmen sollten zudem dermaßen konzipiert werden, dass neben UI Komponenten, deren Schnittstelle der eines funktionalen Dienstes entspricht, zudem angepasste Schnittstellen potentieller UI Komponenten als Ergebnis geliefert werden.

[R3] Generierung von Kopplungen
Die Systemarchitektur sollte in der Lage sein, Schnittstellen sowie notwendige Anpassungen zur Laufzeit in Form von Objekten für einen Austausch zwischen gekoppelten Schnittstellen zu modellieren.

[R4] Interaktion zwischen System-Komponenten
Darüber hinaus sollte die Systemarchitektur über grundlegende Möglichkeiten zum Austausch von Daten zwischen den ausgewählten System-Komponenten verfügen.

[R5] Leistung und Effizienz
Die Systemarchitektur sollte in der Art und Weise realisiert werden, dass der Prozess des Discovery, sofern möglich, in vertretbarer Zeit ein positives Ergebnis, d. h. potentielle UI Komponenten, liefert oder anhand einer entsprechenden Meldung ein negatives Ergebnis signalisiert und den Prozesse beendet.

[R6] Dynamik zur Laufzeit
Eine weitere funktionale Anforderung an die zu konzipierende Systemarchitektur ist die Integration sowie der Austausch von System-Komponenten zur Laufzeit.

3.2.1.2 Nicht-funktionale Anforderungen

Zu den nicht-funktionalen Anforderungen zählen:

[R7] Modularität und Erweiterbarkeit
Durch einen modularen Aufbau und den Einsatz weit verbreiteter sowie u. U. auch standardisierter Programmier- und Scriptsprachen bis hin zu Beschreibungsformaten kann die Erweiterbarkeit, aber auch die Wiederverwendbarkeit einzelner Komponenten des zu konzipierenden Systems garantiert werden.

[R8] Plattformunabhängigkeit und Portabilität
Durch den Einsatz als Webanwendung soll das zu konzipierende System neben einer Vielzahl von Betriebssystemen auch gängige Web-Browser unterstützen, wodurch solche Programmiersprachen favorisiert werden, deren Schwerpunkt auf die Entwicklung von Webanwendungen abzielt. Damit kann neben der vollständigen Plattformunabhängigkeit ebenfalls die Portabilität des Systems gewährleistet werden.

[R9] Benutzbarkeit
Um dem Nutzer den Umgang mit dem System zu erleichtern und ihn mit wenig Aufwand zu einem Ergebnis zu führen, muss die Bedienbarkeit des Systems so einfach und vor allem so intuitiv wie möglich umgesetzt werden und ihn hinreichend bei der Arbeit mit dem System unterstützen.

3.2.2 Systemarchitektur

Mit dem folgenden Abschnitt wird das Konzept einer Systemarchitektur zur (dynamischen) Kopplung von heterogenen, funktionalen Diensten sowie UI Services bzw. UI Komponenten beschrieben.
Mit diesem Abschnitt soll u. a. die Frage untersucht werden, ob und wie sich UI Komponenten anhand der Schnittstellenbeschreibung eines zugehörigen Beschreibungsdokumentes zur Laufzeit vor allem suchen und finden, aber auch in eine Anwendungsumgebung integrieren lassen. Wie aus den einführenden Bemerkungen im Kapitel zu den Grundlagen der Arbeit, vor allem in

Abschnitt 2.1, 2.2.2 sowie 2.5, deutlich wurde, liegen Beschreibungsformate im Bereich von UI Services und UI Dokumenten bisher nicht in standardisierter Form vor. Außerdem wird bei den aktuellen wissenschaftlichen Untersuchungen oftmals nur auf eine Beschreibung zur direkten Kopplung von UI Komponenten eingegangen; die Kopplung dieser an funktionale Dienste gerät dabei zu sehr in den Hintergrund. Infolgedessen wurde mit dem Abschnitt 3.1 dieses Kapitel auf syntaktische Merkmale eingegangen, die für die Konzeption einer Systemarchitektur innerhalb der vorliegenden Arbeitsthematik von grundlegender Bedeutung sind.

Zu den Hauptbestandteilen der Systemarchitektur gehören:

- Ein bzw. mehrere **Service-Parser**, die in der Lage sind, notwendige Informationen von Beschreibungsdokumenten funktionaler Dienste vor allem hinsichtlich der beschriebenen Operationen und Parameter einzulesen und dem System für eine Kommunikation mit den Diensten zur Verfügung zu stellen.

- Ein **UISDL-Parser**, der in der Lage ist, die Informationen eines UISDL-Beschreibungsdokumentes vor allem hinsichtlich der UI Komponenten, den darin beschriebenen Operationen, Ereignissen und Parametern sowie zusätzlichen Meta- und Content-Informationen einzulesen und diese dem System für den Prozess des Discovery zur Verfügung zu stellen.

- Eine **UI Service-Discovery Komponente**, die die Suche und Auswahl von UI Komponenten hinsichtlich der Anforderungen einer ausgewählten Operation eines funktionalen Dienstes realisiert. Im Rahmen des vorliegenden Konzeptes wird dabei auf *denkbare* Ansätze für die Suche und Auswahl von UI Komponenten eingegangen; ein Anspruch auf Vollständigkeit kann an dieser Stelle nicht gegeben werden. Die Discovery Komponente stellt dem System auf eine erfolgreiche Suche und Auswahl hin entsprechende Discovery-Objekte mit Informationen zu den potentiellen UI Komponenten sowie notwendigen Schnittstellen-Anpassungen zur Verfügung.

- Eine **Service-Access Komponente**, die einerseits für den Aufruf eines funktionalen Dienstes verantwortlich ist, wobei eine Vielzahl von Technologien[1] unterstützt werden sollten. Andererseits realisiert die Service-Access Komponente einen Informations- und Datenaustausch mit der

[1] In Anlehnung an Parser für Beschreibungsformate funktionaler Dienste sollte der Service-Adapter in der Lage sein, mit den entsprechenden Diensten kommunizieren zu können

Client-Anwendung um Ergebnisse des Discovery-Prozesses und Daten bezüglich der Kommunikation mit einem funktionalen Dienst zu kommunizieren.

- Ein **Binding-Creator Modul** als Erweiterung der Service-Access Komponente, welches die Funktionalität anbietet, zu kommunizierende Daten anhand der zur Verfügung stehenden Schnittstelleninformationen der Parser und unter Berücksichtigung notwendiger Schnittstellen-Anpassungen des Discovery-Prozesses, zur Laufzeit entsprechende Objekt-Repräsentationen zu modellieren.

- Eine **Client-Runtime**, die u. a. eine Kommunikationsschnittstelle zur System-Anwendung bietet. Über diese Schnittstelle werden Informationen und Daten mit der System-Anwendung ausgetauscht. Weiterhin verfügt die Client-Runtime über einen Mechanismus, um auf eingehende[2] und ausgehende[3] Ereignisse zu reagieren. Zusätzlich sollte die Client-Runtime über Mechanismen und Funktionalitäten verfügen, um die dynamische Integration, d.h. das (Nach-) Laden von UI Komponenten hinsichtlich ihrer Style-Informationen (z. B. CSS) und Script-Bibliotheken (z. B. Javascript) innerhalb der Client-Anwendung zu ermöglichen.

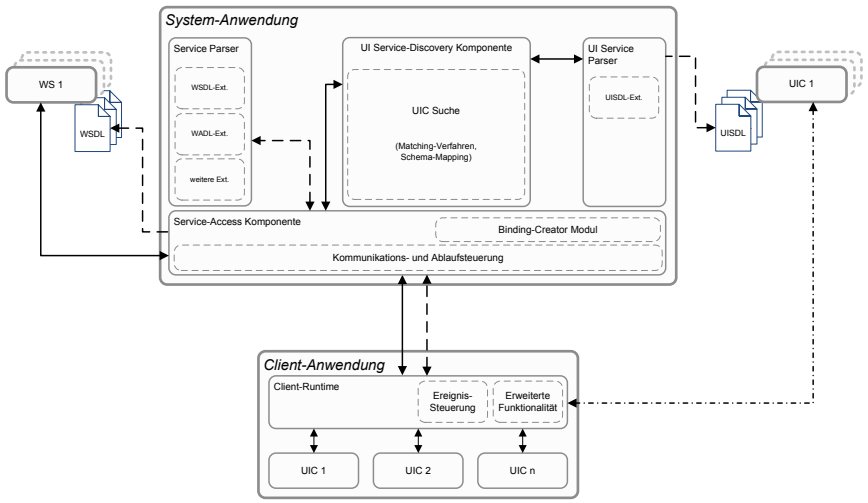


Abbildung 3.3 Gesamtüberblick der Systemarchitektur

Nachdem nun die Hauptbestandteile der zu konzipierenden Systemarchitektur kurz vorgestellt wurden, befasst sich der folgende Abschnitt 3.2.2.1

[2] eingehendes Ereignis: Ereignis, das aufgrund von anstehenden Informationen bzw. Daten der System-Anwendung auftritt

[3] ausgehendes Ereignis: Ereignis, das aufgrund von anstehenden Informationen bzw. Daten der Client-Anwendung auftritt

mit einer *detaillierten Beschreibung dieser Hauptbestandteile* sowie deren Einordnung in die Bereiche *System-Anwendung* und *Client-Anwendung*, wie Abbildung 3.3 auf der vorangegangenen Seite verdeutlicht.

3.2.2.1 System-Anwendung

In diesem Abschnitt werden die Hauptbestandteile sowie deren Funktion und Arbeitsweise innerhalb der System-Anwendung vorgestellt.

Service-Parser

Um mit heterogenen, funktionalen Diensten zu interagieren, werden angebotene Operationen durch Beschreibungsdokumente hinreichend beschrieben, wobei dem Nutzer die zugrunde liegende, funktionelle Implementierung verborgen bleibt. Viele Entwicklungs- und Programmiersprachen bieten bereits Unterstützung für die Kommunikation mit einem funktionalen Dienst, indem Beschreibungsdokumente eingelesen und die darin beschriebenen Schnittstellen für einen Aufruf gekapselt werden. Zudem werden Schnittstellenbeschreibungen vermehrt durch zusätzliche Annotationen erweitert, um eine Kopplung zwischen Operationen funktionaler Dienste zu erleichtern.
Dabei wurde bisher die Möglichkeit einer Kopplung zwischen funktionalen Diensten und UI Komponenten weniger betrachtet, wie bereits im Kapitel 2 deutlich wurde. Für das Thema der vorliegenden Arbeit ist es daher notwendig, Beschreibungsdokumente unter diesem Aspekt genauer zu betrachten und entsprechend zu erweitern (vgl. 3.1.3). Vorhandene Implementierungen für eine Kopplung funktionaler Dienste sind für ein solches Szenario möglicherweise nicht mehr ausreichend und damit in ihrer Funktionalität zu erweitern bzw. durch neue Implementierungen zu ersetzen, um innerhalb der hier beschriebenen System-Anwendung eingesetzt werden zu können.
Ein Service-Parser innerhalb der zu konzipierenden Systemarchitektur sollte daher in der Lage sein, alle notwendigen Informationen der Beschreibung eines funktionalen Dienstes einzulesen und dabei offen und flexibel für zukünftige Erweiterungen von Beschreibungsdokumenten gerade im Hinblick auf das Thema der Arbeit sein.
Der Service-Parser wird innerhalb der Systemarchitektur als ein zentrales Element in die System-Anwendung integriert und stellt eingelesene Informationen der Beschreibungsdokumente in Form von Service-Objekten zur Verfügung.

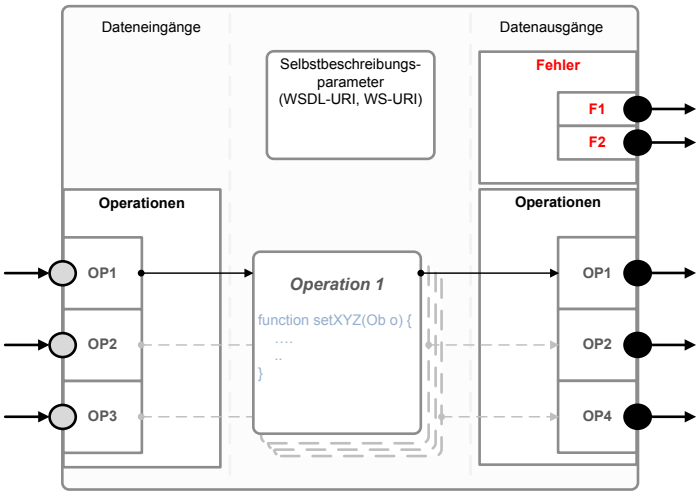


Abbildung 3.4 Objekt eines funktionalen Dienstes innerhalb der Systemarchitektur

Abbildung 3.4 verdeutlicht ein solches Objekt eines funktionalen Dienstes, wie es innerhalb der Systemarchitektur Verwendung findet. Das Objekt ist dabei vor allem auf die angebotenen Dateneingänge und Datenausgänge ausgerichtet, die für eine Kopplung mit Operationen von UI Komponenten notwendig sind. Datenausgänge beziehen sich, anders als die Dateneingänge, nicht nur auf Daten sondern auch auf Fehler, die durch die Ausführung einer Operation eines funktionalen Dienstes auftreten können.

UISDL-Parser

Für die Integration von UI Komponenten in eine Client-Anwendung bzw. die Kopplung einer Operation eines funktionalen Dienstes an eine bestimmte Operation einer UI Komponente ist es zuvor notwendig, alle Informationen aus dem XML-basierten UISDL-Beschreibungsdokument eines UI Services einzulesen.

Ein innerhalb der Systemarchitektur eingesetzter *UISDL-Parser* sollte dabei in der Lage sein, ein ihm per URI-Referenz übergebenes UISDL-Beschreibungsdokument einzulesen und alle notwendigen Informationen zu extrahieren. Abbildung 3.5 auf der folgenden Seite zeigt das Objekt einer UI Komponente (UIC-Objekt), das als Ergebnis dieser Extraktion gesehen werden kann. Dabei werden die in Kapitel 3.1 angeführten syntaktischen Beschreibungsmerkmale als funktionale Merkmale auf eine Objekt-Repräsentation abgebildet. Die wichtigsten Informationen, die ein UISDL-Parser für das Thema der vorliegenden Arbeit zu extrahieren hat, sind

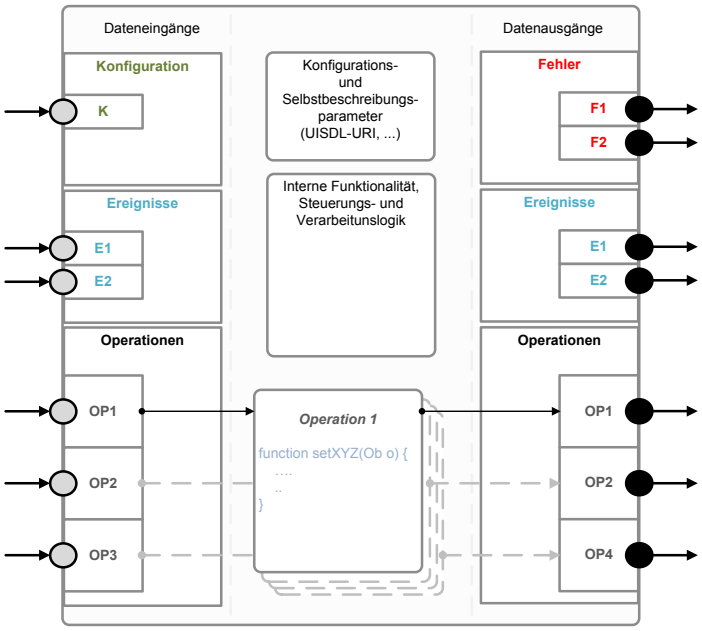


Abbildung 3.5 Objekt einer UI Komponente innerhalb der Systemarchitektur

Konfigurations- bzw. Selbstbeschreibungsparameter[4] sowie Dateneingänge und Datenausgänge. Letztere sollten sich im Rahmen der Arbeit vor allem auf die durch eine UI Komponente angebotenen Operationen beziehen. Zudem werden Ereignisse und Fehler als Dateneingänge bzw. Datenausgänge innerhalb der Objekt-Repräsentation einer UI Komponente angeführt, die für an die vorliegende Arbeit anknüpfende Untersuchungen von größerem Interesse sein könnten und an dieser Stelle lediglich der Vollständigkeit halber angeführt werden.
Aufgrund der Thematik der vorliegenden Arbeit, auf die das in Kapitel 3.1 vorgeschlagene UISDL-Beschreibungsformat ausgerichtet ist, sollte ein UISDL-Parser zudem modular aufgebaut und damit offen für mögliche, zukünftige Erweiterungen eines Beschreibungsdokumentes umgesetzt werden.

UI Service-Discovery Komponente

Mit der *UI Service-Discovery Komponente* ist die wichtigste Komponente innerhalb der Systemarchitektur zu realisieren. Die Komponente hat die Aufgabe, aufgrund einer zuvor ausgewählten Operation eines funktionalen Dienstes, potentielle Kandidaten für UI Komponenten aufzufinden[5], wobei es sich bei der Betrachtung potentieller Kandidaten um die Betrachtung der Dateneingänge bzw. Datenausgänge von Operationen von UI Komponenten

[4] Konfigurations- und Selbstbeschreibungsparameter einer UI Komponente: hierzu zählen sowohl Meta-Informationen als auch Content-Informationen
[5] auffinden: im Englischen „*to discover*“

handelt.
Aufgrund der Auswahl einer Operation eines funktionalen Dienstes lassen sich gewisse Anforderungen an UI Komponenten-Kandidaten ableiten. Eine UI Servce-Discovery Komponente wird mit diesen Anforderungen angesteuert und sucht gezielt in den durch den UISDL-Parser bereitgestellten UIC-Objekte nach möglichen Kandidaten, die diesen Anforderungen genügen (könnten) und damit für eine Kopplung an die ausgewählte Operation eines funktionalen Dienstes in Frage kommen.
Im Folgenden soll nun näher auf den eigentlichen *Prozess des Discovery* eingegangen werden, der sich grundlegend in das Discovery Domain-spezifischer UI Komponenten, vor allem aber der nicht Domain-spezifischen UI Komponenten gliedert.

a] Discovery Domain-spezifischer UI Komponenten

Wie bereits angedeutet (vgl. Kapitel 3.1.2) werden in der Kategorie der Domain-spezifischen UI Komponenten alle UI Komponenten eingeordnet, die über erweiterte Beschreibungsmerkmale verfügen. Dabei lassen sich in dieser Kategorie, wie bereits in Abschnitt 3.1.3.3 angeführt, zwei Arten von UI Komponenten unterscheiden:

Direkte Referenzierung funktionaler Dienste
Bei Operationen von UI Komponenten, dessen Dateneingang bzw. Datenausgang den Dateneingang, Datenausgang bzw. Fehlerausgang einer Operation eines funktionalen Dienstes über das syntaktische Attribut *serviceReference* referenzieren, ist davon auszugehen, dass sie als potentieller Kandidat für die Kopplung an die so referenzierte Schnittstelle des funktionalen Dienstes zu sehen sind. Wird nun in einem Anwendungsszenario eine Übereinstimmung zwischen ausgewählter Operation eines funktionalen Dienstes sowie den Angaben des Attribute *serviceReference* des Dateneinganges bzw. Datenausganges festgestellt, so wird diese Schnittstelle als potentieller Kandidat durch den Discovery Prozess zurück geliefert.
Es wird innerhalb der Arbeit folglich unterstellt, dass von einer direkten Referenzierung durch das Attribut *serviceReference* auch auf eine Übereinstimmung der zu koppelnden Schnittstellen (vgl. Abbildung 3.1) geschlossen werden kann und weitere Untersuchungen nicht notwendig sind.
Nicht auszuschließen ist an dieser Stelle jedoch, dass die derart erweiterte Dateneingabe bzw. Datenausgabe der Operationen von UI Komponenten auch mit Operationen anderer funktionaler Dienste gekoppelt werden kön-

nen, die über das Attribut *serviceReference* nicht direkt referenziert werden. In einem solchen Fall ist die Komponente der im Folgenden vorgestellten Art Domain-spezifischer UI Komponenten oder aber der Kategorie der nicht Domain-spezifischen UI Komponenten zuzuordnen und entsprechend zu verfahren.

Referenzierung einer Domain-Klasse
Neben der direkten Referenzierung lassen sich sowohl Operationen funktionaler Dienste, auf deren erweiterte Beschreibung durch semantische Annotation bereits im Kapitel 2.3 eingegangen wurde, als auch UI Komponenten dahingehend erweitert beschreiben, dass sie eine gemeinsame Domain-Klasse (z. B.: Darstellungsklasse) einer extern verfügbaren Klassifikation referenzieren. Durch das bereits angeführte Attribut *domainReference* lassen sich damit die Dateneingabe bzw. Datenausgabe[6] von Operationen funktionaler Dienste sowie UI Komponenten durch Angabe einer Domain-Klasse erweitern. Die Referenzierung einer *gemeinsamen Domain-Klasse* sagt jedoch nichts über eine mögliche Koppelbarkeit der so erweiterten Schnittstellen aus, sie kann aber in jedem Fall ein hilfreiches Mittel zur Einschränkung der Menge zu prüfender UI Komponenten sein, um den Prozess des Discovery in seiner Performance vor allem im Hinblick auf die Geschwindigkeit zu optimieren.
Um nun aus dieser eingeschränkten Menge an möglichen UI Komponenten potentielle Kandidaten auszuwählen, ist entsprechend dem Vorgehen zum Discovery nicht Domain-spezifischer UI Komponenten zu verfahren, das im Folgenden detailliert beschrieben wird.

b] Discovery nicht Domain-spezifischer UI Komponenten

In Kapitel 3.1.3.2 wurde bereits auf die syntaktische Erweiterung von Operationen von UI Komponenten eingegangen. Dabei wurde u. a. eine Unterteilung in die Bereiche der starren sowie der flexiblen Operationen vorgenommen. Da sich in der Kategorie der flexiblen Operationen zwar theoretische Beispiele anführen lassen, Implementierungen solcher Operationen jedoch im Bereich der hier verwendeten UI Komponenten bisher nicht verfügbar sind, lässt sich für die Konzeption einer Systemarchitektur festhalten, dass flexible Operationen derart realisiert sind, dass sie mit jeder ihnen übergebenen Datenstruktur sowohl direkt als auch über die Konfiguration umgehen können, wodurch flexibel gekennzeichnete Operationen immer als potentielle Kandi-

[6] auf Seite der Erweiterung funktionaler Dienste kann außerdem die Fehlerausgabe annotiert sein

daten des Discovery-Prozesses zu sehen sind. Aus diesem Grund soll es bei der weiteren Betrachtung nicht Domain-spezifischer UI Komponenten um starre Operationen gehen, die den wesentlichen Teil des Discovery-Prozesses einnehmen.

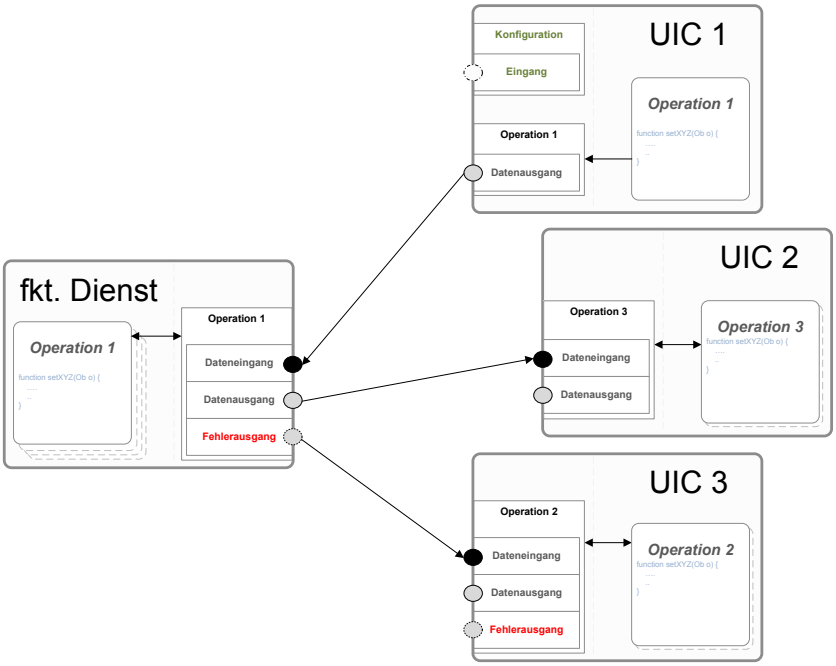


Abbildung 3.6 Gegenüberstellung der Kopplung von Operationen

In dieser Kategorie der starren Operationen von UI Komponenten sind Parameter der Dateneingabe bzw. Datenausgabe wohl definiert. Die UI Service Discovery Komponente sollte so umgesetzt werden, dass sie über Mechanismen bzw. Algorithmen verfügt, die einen Element-basierten Parameter-Vergleich der zu koppelnden Schnittstellen ermöglichen.
Abbildung 3.6 zeigt ein mögliches Szenario zur Kopplung der Operation eines funktionalen Dienstes an Operationen von UI Komponenten. Dabei ergeben sich folgende Konstellationen von den Dateneingabe-, Datenausgabe- und Fehlerausgabe-Parameter, die bei einen Element-basierten Parameter-Vergleich durch entsprechende Matching-Verfahren abgedeckt werden müssen:

- Dateneingabe Operation funkt. Dienst ⇐ Datenausgabe Operation UI Komponente
- Datenausgabe Operation funkt. Dienst ⇒ Dateneingabe Operation UI Komponente
- Fehlerausgabe Operation funkt. Dienst ⇒ Dateneingabe Operation UI Komponente

Für die Beschreibung von Übergabe-Parametern wurde in Kapitel 3.1.3.1 bereits auf die Definition einfacher als auch komplexer Datentypen bei UI Komponenten hingewiesen, aber auch funktionale Dienste verfügen über derartige Definitionen von Parametern. Abbildung 3.7 verdeutlicht die Gegenüberstellung von Parametern zwischen Datenausgabe einer Operation eines funktionalen Dienstes und der Dateneingabe einer Operation einer UI Komponente.

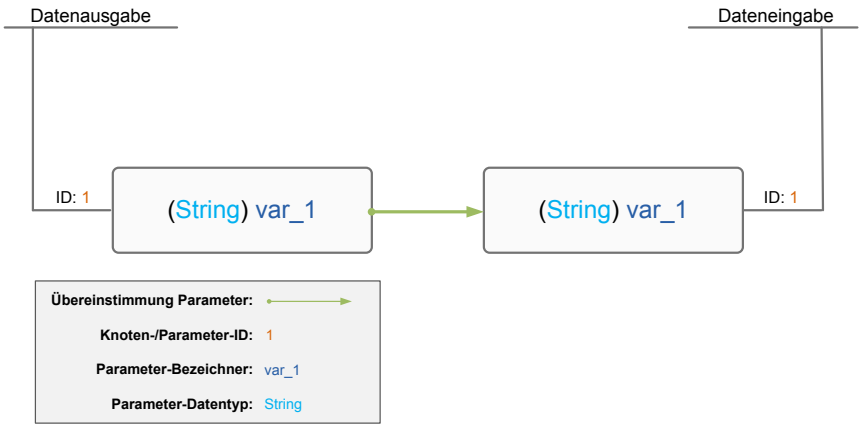


Abbildung 3.7 Gegenüberstellung identischer Parameter

Während bei einfach definierten Parametern ein Vergleich von Bezeichner und Datentyp ausreichend sein kann, sind bei komplex definierten Parametern u. U. weitere Attribute der Parameter einer Knoten-Ebene zu vergleichen. Sind die Parameter zudem (auch mehrfach) komplex geschachtelt, kann der Aufwand eines Parameter-Vergleiches schnell extrem zunehmen.
Bei komplexen Datentyp-Definitionen lassen sich die Parameter der Schnittstellen in einer Baumstruktur erfassen, um diese anschließend rekursiv zu durchlaufen. Bei einem Element-basierten Vergleich wird nun jeder Parameter einer Knoten-Ebene der einen Baumstruktur mit jedem Parameter der selben Knoten-Ebene der anderen Baumstruktur verglichen. Einige der Ansätze für einen Parameter-Vergleich wurden bereits mit den Element-basierten Matching-Verfahren in Kapitel 2.4.2.1 genannt. Innerhalb der Systemarchitektur sollten hier vor allem die String-basierten Verfahren unterstützt werden, mit denen sich Bezeichner von Parametern vergleichen lassen. Neben den bereits angeführten Prefix-, Suffix- und Distanz-Matching-Verfahren lassen sich auch die folgenden Verfahren für einen Element-weisen, String-basierten Vergleich einsetzen:

- *Metaphone*: zwei Worte dienen als Eingabe; es wird anhand der Metaphone-

Schlüssel der beiden Worte untersucht, ob der Schlüssel des kürzeren im längeren enthalten ist

- *Similarity*: zwei Worte dienen als Eingabe; es wird die Anzahl an Übereinstimmungen zwischen den beiden Worten berechnet

Erweitert wird der Element-basierte Vergleich von Parametern durch Constraint-basierte Verfahren (vgl. Kapitel 2.4.2.1), die aufzeigen sollen, inwieweit sich die jeweils untersuchten Parameter in ihrem Datentyp aber auch in anderen Attributen, ganz allgemein also in ihren Eigenschaften unterscheiden.

Für den eigentlichen syntaktischen Vergleich von Schnittstellen sollte dabei nach den drei, im Folgenden angeführten Phasen vorgegangen werden, um einen möglichst effizienten Vergleich zu realisieren:

Erste Phase
In der ersten Phase wird zunächst ein einfacher Vergleich durchgeführt, indem Schnittstellen direkt miteinander verglichen werden. Man könnte auch sagen, dass die Beschreibungen der Schnittstellen wie Schablonen übereinander gelegt werden. Stimmen die untersuchten Schnittstellen überein, so speichert der Discovery Prozess den gefundenen Kandidaten einer UI Komponente zwischen. Weitere Untersuchungen der folgenden Phasen sind bei der Schnittstelle des Kandidaten nicht mehr notwendig.

Zweite Phase
In der zweiten Phase werden Schnittstellen miteinander verglichen, sofern die Struktur von Parametern übereinstimmt. In dieser Phase werden neben den Parameter-Bezeichnungen auch Eigenschaften von Parametern sowie die Stellung innerhalb der Datentyp-Definition verglichen. Damit wird untersucht, ob zu jedem gegebenen Parameter einer Schnittstelle auch ein entsprechender Parameter innerhalb der zu koppelnden Schnittstelle beschrieben ist. Wird in dieser Phase des Discovery Prozesses ein Ergebnis zurückgegeben, so handelt es sich dabei um Schnittstellen, die aufgrund festgestellter Abweichungen u. U. angepasst werden müssen. Diese Anpassung ist für eine fehlerfreie Kommunikation von Daten und Informationen zwischen den zu koppelnden Operationen notwendig. Weiterhin ist in dieser Phase festzuhalten, ob für jeden Parameter einer Schnittstelle auch ein Parameter in der anderen Schnittstelle, unter Verwendung etwaiger Anpassungen, enthalten ist. Trifft dies nicht zu, so kann es notwendig sein, die Schnittstellen in Phase drei erneut zu untersuchen.

Dritte Phase
In der dritten Phase werden alle Schnittstellen miteinander verglichen, die nicht bereits in der ersten bzw. zweiten Phase erfolgreich verglichen und als Ergebnis gespeichert werden konnten. Zu den Ausprägungen gehören neben Schnittstellen der Phase zwei, bei denen nicht zu jedem Parameter einer Schnittstelle ein korrespondierender Parameter innerhalb der zu vergleichenden Schnittstelle festgestellt werden konnte, auch Schnittstellen, die über eine ungleiche Anzahl an Parametern verfügen bzw. bei denen sich die Struktur der Datentypen wesentlich unterscheidet.
Parameter, die in dieser Phase nicht zugeordnet werden können, müssen entsprechend markiert werden. Es ergeben sich nun zwei Möglichkeiten, diese unterschiedlichen Schnittstellen möglicherweise dennoch koppeln zu können.

- Möglichkeit 1: Zuordnung mehrerer Schnittstellen von UI Komponenten auf die Schnittstelle der ausgewählten Operation eines funktionalen Dienstes

- Möglichkeit 2: direkte Zuordnung offener Parameter durch einen Anwendungsnutzer

In Möglichkeit 2 wird davon ausgegangen, dass die untersuchten Schnittstellen sich in ihrer Komplexität (Struktur, Anzahl Parameter) gleichen, jedoch eine Anzahl von Parametern festgestellt wird, die über die verwendeten Matching-Verfahren nicht direkt zugeordnet werden konnten. An dieser Stelle könnte beispielsweise ein Entwickler in den Discovery-Prozess mit einbezogen werden, indem dieser eine Zuordnung der offenen Parameter der untersuchten Schnittstellen vornimmt, um eine mögliche Kopplung vorzunehmen.
Auch die in Phase 3 ermittelten Kandidaten sowie alle notwendigen Anpassung für eine Kopplung werden als Ergebnis des Discovery-Prozesses gespeichert, um im weiteren Anwendungsverlauf zur Verfügung zu stehen.

Binding-Creator Modul
Das *Binding-Creator Modul* ist eine Erweiterung der Service-Access Komponente und dient innerhalb der zu konzipierenden Systemarchitektur dazu, Objekte zu modellieren, welche die zwischen den gekoppelten Schnittstellen zu übertragenen Daten repräsentieren. Dabei sind auch während des Discovery-Prozesses festgestellte Anpassungen, gerade in Bezug auf Bezeich-

ner und Datentyp von Parametern, in die Modellierung der Objekte mit einzubeziehen, um eine fehlerfreie Kommunikation gewährleisten zu können.
Interessant ist das Modul dahingehend, da Objekte im Wesentlichen einer bestimmten Definition einer Objektklasse entsprechen und generell zur Designzeit festgelegt werden. Da jedoch die zu koppelnden Schnittstellen innerhalb der Systemarchitektur erst zur Laufzeit identifiziert werden, muss das System in der Lage sein, auch die entsprechenden Objektklassen zur Laufzeit zu modellieren.
Das Modul kann als Teil bereits vorgestellter Service-Access Komponenten umgesetzt werden und sollte dabei über erweiterbare Logiken verfügen, die beispielsweise die Transformation von Daten zwischen verschiedenen Datentypen realisieren, um der entsprechenden Objekt-Repräsentation zu genügen. Mit Abbildung 3.8 wird ein mögliches Szenario für die Modellierung einer Objektklasse aus der Schnittstellenbeschreibung sowie einer Instanz dieser Klasse unter Anreicherung von Daten aufgezeigt. Ein solches Objekt wird anschließend entweder für den Aufruf eines funktionalen Dienstes über die Service-Access Komponente eingesetzt oder aber an die im Folgenden noch vorgestellte Client-Anwendung zur Darstellung innerhalb einer UI Komponente übermittelt.

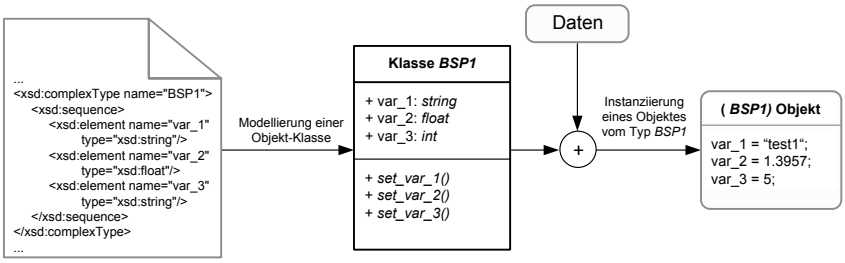


Abbildung 3.8 Instanziierung eines Objektes anhand einer Schnittstellenbeschreibung und spezifischen Daten

Service-Access Komponente

Während die bereits vorgestellte UI Service-Discovery Komponente die wichtigste Komponente der Systemanwendung für das Auffinden potentieller Kandidaten von Operationen von UI Komponenten für eine Kopplung an die ausgewählte Operation eines funktionalen Dienstes ist, gewährleistet die *Service-Access Komponente* die Koordination des Daten- und Informationsflusses zwischen allen Komponenten der System-Anwendung sowie zwischen den gekoppelten Schnittstellen.

Wie bereits angedeutet, greift die Service-Access Komponente auf das Binding-Creator Modul zurück, um Objekte zu modellieren, welche die zu kommunizierenden Daten repräsentieren.
Da die Systemarchitektur eine Vielzahl von funktionalen Diensten unterstützen sollte, sind innerhalb der Komponente entsprechende Ansätze für den Aufruf von funktionalen Diensten zu implementieren bzw. ist auf vorhandene Implementierung zurückzugreifen, um die Kommunikation von Daten zwischen der System-Anwendung und den Operationen funktionaler Dienste zu gewährleisten.
Neben dem Aufruf funktionaler Dienste wird innerhalb der Service-Access Komponente weiterhin der Austausch von Daten und Informationen mit der im Folgenden vorgestellten Client-Anwendung sowie die Auswertung von Informationen realisiert, die sich beispielsweise aus dem Discovery-Prozess ergeben. Damit spielt die Service-Access Komponente eine wesentliche Rolle bei der Verwaltung und Kommunikation von Daten und Informationen, die den gesamten Anwendungsablauf betreffen.

3.2.2.2 Client-Anwendung

Die *Client-Anwendung* verfügt über eine Client-Runtime, die zum einen den Datenfluss innerhalb der Client-Anwendung sowie mit der System-Anwendung realisiert, zum anderen über erweiterte Funktionalitäten verfügt, um UI Komponenten in die Client-Anwendung zu integrieren. Zudem sollte innerhalb der Client-Anwendung eine Ereignis-Steuerung umgesetzt werden.
Abbildung 3.9 auf der folgenden Seite zeigt einen Auszug der Systemarchitektur, der den wesentlichen Aufbau der Client-Anwendung sowie deren Einordnung innerhalb der Systemarchitektur nochmals verdeutlicht. Im Folgenden werden die wichtigsten Komponenten der Client-Anwendung kurz detaillierter vorgestellt.

Client-Runtime
Die *Client-Runtime* wertet Daten und Informationen der Interaktion mit der Client-Anwendung bzw. der Kommunikation zwischen Client-Anwendung und System-Anwendung aus.
Informationen beziehen sich dabei einerseits auf die Interaktion mit der Client-Anwendung, beispielsweise die Darstellung potentieller Kandidaten von UI Komponenten zur Kopplung an die ausgewählte Operation eines funktionalen Dienstes, andererseits auf Informationen, die sich auf die Inte-

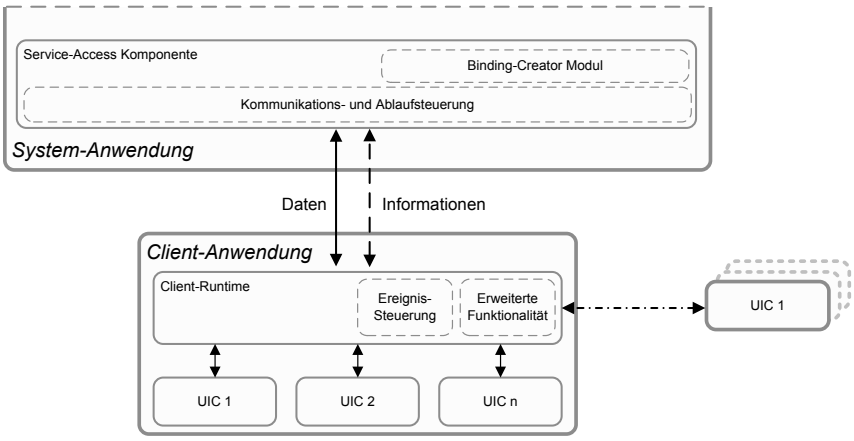


Abbildung 3.9 Aufbau und Einordnung der Client-Anwendung innerhalb der Systemarchitektur

gration von UI Komponenten beziehen.
Daten hingegen entstehen aus der Interaktion der gekoppelten Schnittstellen und sind dabei entsprechend über die Ereignis-Steuerung der Client-Anwendung an die jeweilige Operation einer UI Komponente oder aber zur weiteren Verarbeitung an die System-Anwendung weiterzuleiten. Eine Transformation von Daten findet innerhalb der Client-Runtime nicht statt; aufgrund der möglichen Komplexität ist hierfür, wie bereits angedeutet, ausschließlich die Service-Access Komponente bzw. das Binding-Creator Modul der System-Anwendung zuständig.
Neben diesen allgemeinen Funktionalitäten sollte die Client-Runtime vor allem über Funktionalitäten verfügen, die das *(Nach-) Laden* von UI Komponenten bzw. deren Style-Informationen (z.B. CSS-Datei(en)) und Script-Bibliotheken (z. B. JavaScript-Datei(en)) sowie das Initialisieren der jeweiligen UI Komponenten zur Laufzeit ermöglichen. Zusätzlich sollten Funktionalitäten zur Verfügung stehen, um UI Komponenten, in Anlehnung an die zu realisierende Kopplung, innerhalb der Ereignis-Steuerung zu registrieren. Damit wird realisiert, dass registrierte UI Komponenten über entsprechende Ereignisse informiert werden können.

Ereignis-Steuerung

Mit der *Ereignis-Steuerung* wird eine weitere wichtige Komponente der Client-Anwendung vorgestellt. Diese sollte über Möglichkeiten verfügen, auf eingehende Daten einer Operation eines funktionalen Dienstes aber auch ausgehende Daten der Interaktion mit einer UI Komponente zu reagieren, d.h. diese an den eigentlichen Endpunkt der Kommunikation weiterzuleiten.

Da sich das Thema der vorliegenden Arbeit mit der Kopplung von funktionalen Diensten und UI Komponenten befasst, werden bei der Konzeption der Ereignis-Steuerung solche Ereignisse vernachlässigt, welche sich mit der direkten Kopplung von UI Komponenten befassen. Untersuchungen in diesem Bereich sollten als Teil weiterer wissenschaftlicher Arbeiten erneut analysiert werden.

3.3 Beispielhaftes Ablaufszenario

Das im Folgenden formulierte Interaktionsschema soll beispielhaft den Anwendungsablauf, ausgehend von der Initialisierung der Anwendung über die Möglichkeit der Auswahl einer Operation eines funktionalen Dienstes sowie der Auswahl potentieller Kandidaten zu koppelnder UI Komponenten an die Operation eines Dienstes, die Integration einer UI Komponente bis hin zur Interaktion mit der Operation der UI Komponente aufzeigen.

1. Mit dem ersten Aufruf der Anwendung werden dem Nutzer zunächst alle innerhalb der System-Anwendung bekannten Operationen funktionaler Dienste in der Nutzeroberfläche der Client-Anwendung präsentiert.

2. Aus der Nutzerauswahl einer Dienst-Operation auf Seite der Client-Anwendung wird innerhalb der System-Anwendung eine Suche nach potentiellen Kandidaten von Operationen dem System bekannter UI Komponenten vorgenommen. Alle aus dieser Suche resultierenden Operationen werden erneut dem Nutzer zur Auswahl präsentiert. Aufgrund der Beschreibung von UI Services bzw. den darin gekapselten UI Komponenten können dem Nutzer potentielle UI Komponenten über Meta-Informationen als auch in visueller Form durch Präsentation einer Vorschau-Grafik (Screenshot) angeboten werden.

3. Anhand der Nutzerauswahl möglicher Operationen von UI Komponenten sowohl für die Dateneingabe als auch für die Datenausgabe einer Operation eines funktionalen Dienstes wird innerhalb der System-Anwendung eine Kopplung zwischen den jeweiligen Dateneingaben und Datenausgaben der Operationen vorgenommen. Während des Discovery-Prozesses festgestellte Anpassungen werden bei den realisierten Kopplungen mit berücksichtigt.

4. Anschließend werden notwendige Content-Informationen (Style-Informationen, Script-Bibliotheken), die innerhalb einer UI Service

Beschreibung einer UI Komponente referenziert werden, zum Nachladen an die Client-Anwendung übermittelt. Weiterhin wird die UI Komponente innerhalb der Nutzeroberfläche instanziiert sowie notwendige Registrierungen der Instanz der UI Komponente, in Ahnlehnung an die realisierte Kopplung, über die Ereignis-Steuerung vorgenommen.

5. Im weiteren Verlauf kann eine Interaktion des Anwendungsnutzers mit den Operationen der integrierten UI Komponenten stattfinden. Die aus der Interaktion resultierende Kommunikation von Daten an eine gekoppelte Operation eines funktionalen Dienstes bleibt dem Anwendungsnutzer dabei verborgen.

3.4 Zusammenfassung

Mit dem vorliegenden Kapitel wurde zunächst das UISDL-Beschreibungsformat für UI Services in seiner syntaktischen Spezifikation erweitert, um für die des Themas der vorliegenden Arbeit eingesetzt werden zu können. Neben diesem Beschreibungsformat wurden Anforderungen an eine Systemarchitektur formuliert. Dabei wurden zunächst funktionale sowie nicht-funktionale Anforderungen an die Systemarchitektur aufgestellt, aus denen sich die wichtigsten Komponenten für die Entwicklung eines Konzeptes ergaben. Nach einem kurzen allgemeinen Überblick über die Systemarchitektur wurden einzelne Komponenten, die für die Umsetzung der Arbeitsthematik als notwendig erachtet wurden, sowie deren Positionierung und Aufgabe sowie Arbeitsweise innerhalb des Konzeptes detaillierter vorgestellt.
Das Konzept zeigt damit grundlegende Lösungsansätze für die folgenden Schwerpunkte auf:

- Einführung und Verarbeitung eines Beschreibungsformates für UI Services (UISDL),
- Auswahl heterogener, funktionaler Dienste und UI Services (Auswahl Operation UI Komponente),
- Dynamische Kopplung ausgewählter Operationen funktionaler Dienste und UI Komponenten zur Laufzeit,
- Interaktion zwischen den so gekoppelten Operationen.

Im folgenden Kapitel 4 wird auf die prototypische Umsetzung des vorgestellten Konzeptes eingegangen.

Kapitel 4

Prototypische Umsetzung

Mit dem vorangegangen Kapitel 3 wurde zum Einen auf das UISDL-Beschreibungsformat für User Interface Services, zum Anderen auf Anforderungen an eine Systemarchitektur sowie den Entwurf eines Konzeptes für die (dynamische) Kopplung von funktionalen Diensten und User Interface Services eingegangen. Das nun folgende Kapitel befasst sich mit dem praktischen Teil der Arbeit, in dem aufgezeigt werden soll, wie das entwickelte Konzept anhand einer prototypischen Umsetzung validiert werden kann.
Dabei geht der vorliegende Abschnitt 4.1 zunächst auf die Umsetzung der wichtigsten Hauptbestandteile des entwickelten Konzeptes ein und skizziert die wesentlichen Vorgänge und Abläufe der prototypischen Umsetzung innerhalb der Bereiche System-Anwendung sowie Client-Anwendung anhand eines beschriebenen Navigationspfades, der durch beispielhafte Abbildungen verdeutlicht wird.
Abschließend wird im Abschnitt 4.2 das Kapitel der prototypischen Umsetzung zusammengefasst, bevor im darauf folgenden Kapitel 5 auf die Evaluierung des entwickelten Konzeptes anhand der prototypischen Umsetzung eingegangen wird.

4.1 Systemarchitektur

Da die für das Thema der vorliegenden Arbeit wichtigsten Kriterien und Merkmale in einem Format zur Beschreibung von UI Services bzw. den darin gekapselten UI Komponenten in Kapitel 3.1 zunächst neu eingeführt wurden, war davon auszugehen, dass Implementierungen, die das entwickelte Beschreibungsformat unterstützen, nicht vorhanden sein würden. Auch in anderen Bereichen wie der Suche, Auswahl und Integration wurde deutlich, dass mögliche vorhandene Implementierungen nicht den notwendigen Anfor-

derungen entsprechen bzw. der Aufwand bei der Analyse von Programmcode oder aber dessen Erweiterung einen erheblichen zeitlichen Aufwand notwendig machen.
Aus diesen Gründen wurde für den umgesetzten Prototypen ein eigenständiges und in vielen Teilen erweiterbares Anwendungssystem entwickelt, welches neben weiteren Anforderungen vor allem die UI Service Discovery, aber auch die Integration sowie den Austausch von UI Komponenten zur Laufzeit ermöglicht. In den folgenden Teilabschnitten 4.1.1 und 4.1.2 wird daher vor allem auf die Umsetzung dieser komplexen Anforderungen eingegangen.

4.1.1 UI Service-Discovery

Mit der UI Service-Discovery Komponente wurde der wichtigste Bestandteil des entwickelten Konzeptes innerhalb des Prototypen umgesetzt. Die Komponente ermöglicht die Suche und das Auffinden von UI Komponenten zu einer ausgewählten Operation eines funktionalen Dienstes.

Abbildung 4.1 Operationen funktionaler Dienste (Ansicht Client-Anwendung)

Abbildung 4.1 zeigt eine Übersicht aller dem System bekannten Operationen funktionaler Dienste. Wird durch den Anwendungsnutzer aus dieser Übersicht eine Operation ausgewählt, so wird die Auswahl von der Client-Anwendung an die System-Anwendung übermittelt. Die System-Anwendung leitet die Auswahl sowie die URI-Referenz der Dienstbeschreibung des funktionalen Dienstes, der die ausgewählte Operation anbietet, an den Service-Parser weiter. Dieser liest Informationen des Beschreibungsdokumentes ein und stellt diese innerhalb der System-Anwendung zur Verfügung. Die notwendigen Informationen zu Dateneingabe und -ausgabe der ausgewählten Operation werden zur Ansteuerung an eine Instanz des Discovery Prozesses

übergeben, woraufhin eine Suche nach passenden UI Komponenten aus den der System-Anwendung bekannten UI Services durchgeführt wird. Da zum Zeitpunkt der Themenbearbeitung noch keine UI Service Registry verfügbar ist, werden UI Services innerhalb der System-Anwendung statisch über die Angabe der URI ihres UISDL-Beschreibungsdokumentes referenziert. Diese statische Definition soll im Folgenden, sofern notwendig, als *UIS Repository* bezeichnet werden.

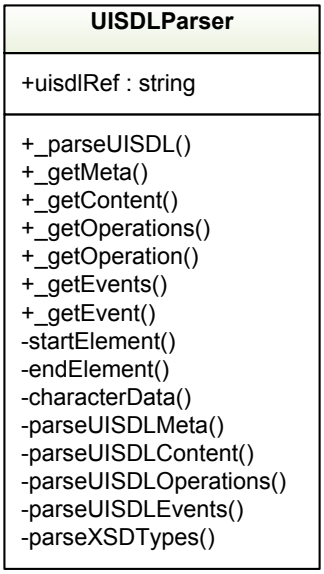


Abbildung 4.2 Klassendiagramm *UISDLParser*-Klasse

Um auf Informationen der UISDL-Beschreibungsdokumente zugreifen zu können, ruft der Discovery Prozess zur Laufzeit zunächst den ebenfalls prototypisch umgesetzten UISDL-Parser mit jeder statisch definierten URI-Referenz der UI Services des UIS Repository auf, liest die Informationen des jeweiligen Beschreibungsdokumentes ein und stellt diese dem Discovery Prozess zur Verfügung.
Abbildung 4.2 zeigt das Klassendiagramm der implementierten Klasse *UISDLParser*. Über die öffentlichen Methoden lassen sich, nach Instanziierung eines UISDL-Parser Objektes durch Übergabe einer URI-Referenz eines UISDL-Beschreibungsdokumentes, alle notwendigen Informationen (vgl. Kapitel 3.1.2 und 3.1.3) für den weiteren Anwendungsverlauf abrufen.
Das Ziel des Discovery Prozesses ist es, anhand der zur Verfügung stehenden Informationen des funktionalen Dienstes sowie der UI Services, möglichst viele Übereinstimmungen zwischen den Schnittstellen (vgl. Kapitel 3.1.1)

festzustellen, um der System-Anwendung daraus resultierende Informationen zu potentiellen Operationen von UI Komponenten für die Auslieferung an die Client-Anwendung zur Verfügung stellen zu können. Dabei wurde der bereits im Kapitel 3.2.2.1 angeführte mehrstufige, Element-basierte Vergleich von Dateneingabe- und Datenausgabe-Parametern umgesetzt, der mit der Instanziierung eines Objektes der Klasse *Discovery* und dem Aufruf der öffentlichen Methode *getUICs()* initialisiert wird.

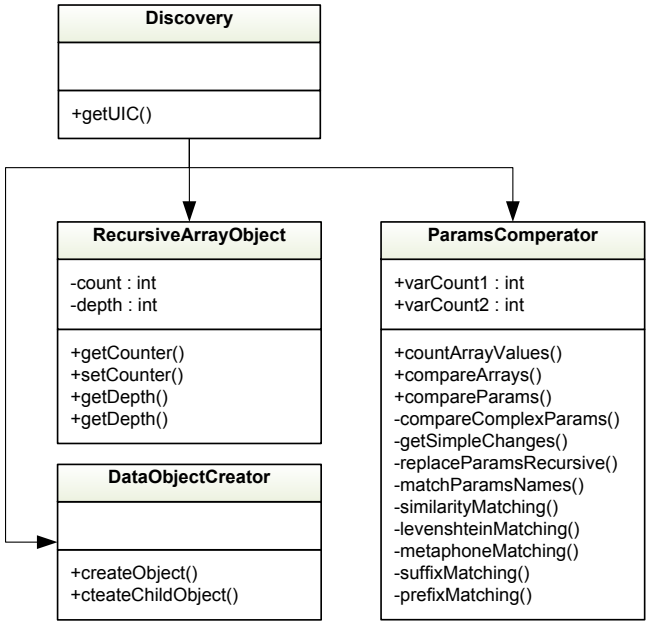


Abbildung 4.3 Klassendiagramm Discovery Prozess (Auszug aus A.1)

Die Klasse Discovery besteht, wie Abbildung 4.3 zeigt, lediglich aus dieser einen Methode. Die Methode wurde dermaßen umgesetzt, dass alle Phasen des mehrstufigen Vergleiches nacheinander ausgeführt werden könnten. Als notwendiges Hilfsmittel dient an dieser Stelle die Klasse *ParamsComparator*. In der *ersten Phase* liefert eine Instanz der Klasse *ParamsComparator* mit dem Aufruf der Funktion *compareArrays* lediglich einen boolschen Wert (true, false) über den direkten Vergleich der Dateneingabe- bzw. Datenausgabe-Parameter der zu untersuchenden Schnittstellen. Liefert ein Aufruf dieser Methode den Wert *false*, so wird die *zweite Phase* des Element-basierten Vergleiches mit dem Aufruf der Methode *compareParams* der *ParamsComparator*-Instanz ausgeführt. Diese Methode analysiert nun knotenweise jeden Parameter der übergebenen Dateneingabe- bzw. Datenausgabe-Parameter der Schnittstellen und reagiert auf Unterschiede zwischen Parametern. Die im zurückliegenden Kapitel 3.2.2.1 angeführte Phase drei des

Vergleiches konnte bisher nicht umgesetzt werden, da hier zum Element-basierten Vergleich zusätzlich ein aufwendiger Struktur-basierter Vergleich u. U. mit Entwickler- oder Anwenderunterstützung zu realisieren wäre.
Als Ergebnis aus den Phasen eins und zwei liefert der umgesetzte Discovery Prozess zu den potentiellen Kandidaten der Operationen von UI Komponenten neben Meta- und Content-Informationen außerdem Informationen über die jeweiligen Dateneingabe- bzw. Datenausgabe-Parameter, um im weiteren Anwendungsverlauf eine erfolgreiche Kommunikation von Daten zwischen UI Komponenten und funktionalen Diensten gewährleisten zu können.

4.1.2 Auswahl, Integration und Interaktion

Die durch den Discovery Prozess als Ergebnis ermittelten Schnittstellen-Informationen zu potentiellen Operationen von UI Komponenten werden über die System-Anwendung an die Client-Anwendung übergeben und hier innerhalb der Client-Runtime ausgewertet, um anschließend in einer nutzerfreundlich gestalteten Dialog-Ansicht präsentiert zu werden (vgl. Abbildung 4.4 sowie 4.5).

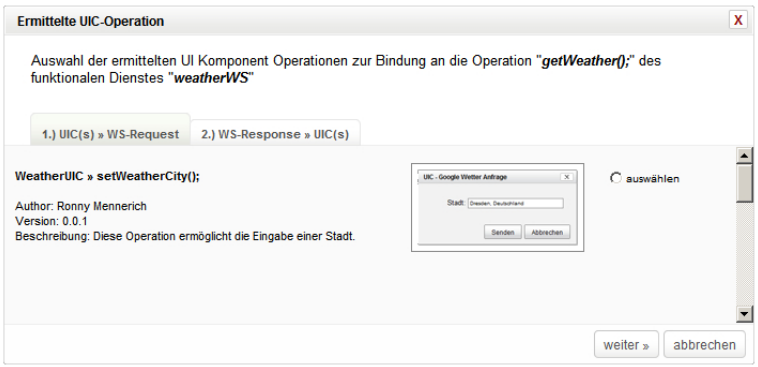


Abbildung 4.4 Dialog-Ansicht durch den Discovery Prozess ermittelter UIC Eingabe-Operation (Ansicht *UIC(s)* ⇒ *WS-Request*)

Sofern verfügbar, werden dem Nutzer in diesem Dialog alle notwendigen Informationen bezüglich der Kopplung an Dateneingabe (WS-Request), Datenausgabe (WS-Response) sowie Fehlerausgabe (WS-Fault) der ausgewählten Operation des funktionalen Dienstes aber auch der Abgrenzung zwischen den Operationen von UI Komponenten zur Auswahl präsentiert.
Der Nutzer hat somit die Möglichkeit, aus den ermittelten Operationen die für das Anwendungsszenario geeigneten Operationen auszuwählen und zu speichern, wodurch die Auswahl zur Generierung der entsprechenden Kopp-

lungen zwischen der ausgewählten Operation des funktionalen Dienstes und den ausgewählten Eingabe- bzw. Ausgabe-Operationen von UI Komponenten an die System-Anwendung übertragen wird.

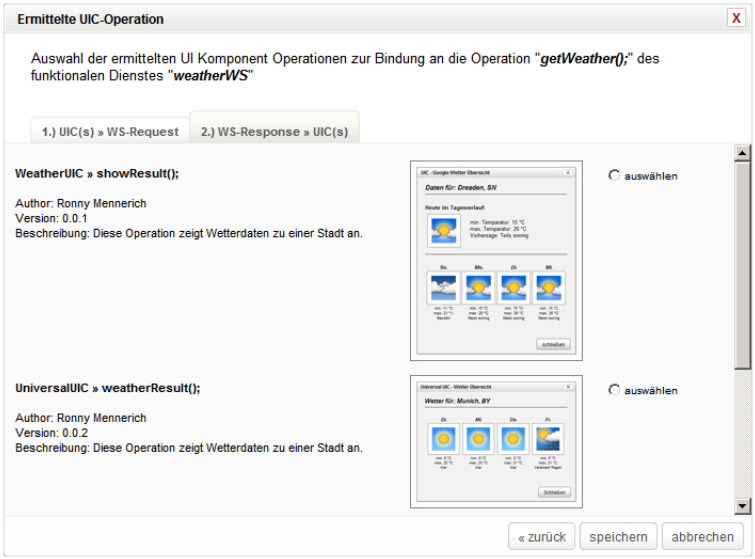


Abbildung 4.5 Dialog-Ansicht durch den Discovery Prozess ermittelter UIC Ausgabe-Operationen (Ansicht *WS-Response* ⇒ *UIC(s)*)

Neben den generierten Kopplungen werden anschließend alle notwendigen Informationen zur Integration der ausgewählten Operationen der UI Komponenten an die Client-Anwendung übertragen. Bei der Auswertung dieser Informationen durch die Client-Runtime werden neben Style-Informationen auch entsprechende Script-Bibliotheken identifiziert, die für die Integration und Instanziierung einer UI Komponente notwendig sind.
Da es sich bei der prototypisch umgesetzten Anwendung um eine Webanwendung handelt, werden diese Style-Informationen und Script-Bibliotheken zur Laufzeit nachgeladen. Der zugrunde liegende Ansatz beruht auf der Möglichkeit, eine Webanwendung mit Web-Browser-spezifischen und damit weniger umfangreichen Style-Informationen bzw. Script-Bibliotheken schneller laden zu können.
Mit diesem Ansatz wurde der Prototyp dahingehend erweitert, dass Style-Informationen und Script-Bibliotheken flexibel zur Laufzeit und nicht bereits zum Zeitpunkt der Initialisierung der Webanwendung integriert werden können. Das in Listing 4.1 vorgestellte Quellcode-Gerüst zeigt beispielhaft eine solche Funktion zur dynamischen Integration einer über den Parameter *url* referenzierten, externen Javascript-Bibliothek.

```
1 // load external JS Files stored on 3rd party server
2 function loadExtJS(url) {
3   var jScript = document.createElement('script');
4   jScript.type = 'text/javascript';
5   jScript.src = url;
6   document.getElementsByTagName('head')[0].appendChild(jScript);
7 }
```

Listing 4.1 Funktion zur dynamischen Integration von Javascript-Bibliotheken

Die mit dem Listing 4.1 vorgestellte Javascript-Funktion *loadExtJS* erzeugt zunächst ein *Script*-Element vom Typ *text/javascript*. Als Quelle wird diesem Element die mit dem Parameter *url* übergebene URI-Referenz der externen Javascript-Bibliothek zugeordnet. Anschließend wird das erzeugte Element in die HTML-Kopfdaten (*head*) der Webanwendung integriert, womit die externe Javascript-Bibliothek für eine interne Nutzung zur Verfügung steht.

Im Anschluss an die Integration der Style-Informationen bzw. Script-Bibliotheken werden notwendige Objekte der UI Komponenten instanziiert und innerhalb der umgesetzten Ereignis-Steuerung der Client-Runtime registriert. Mit diesem Schritt wird eine Abbildung der in der System-Anwendung generierten Kopplungen zwischen ausgewählten Operationen der funktionalen Dienste sowie der Eingabe- bzw. Ausgabe-Operationen der UI Komponenten erzeugt, um Daten aus Dienst-Aufrufen bzw. Interaktionen mit den Operationen von UI Komponenten zwischen Client-Anwendung und System-Anwendung entsprechend der Kopplungen austauschen zu können. Die innerhalb des Prototypen entwickelte Möglichkeit zum Aufruf der ausgewählten Operationen der UI Komponenten wird mit der Abbildung 4.6 verdeutlicht.

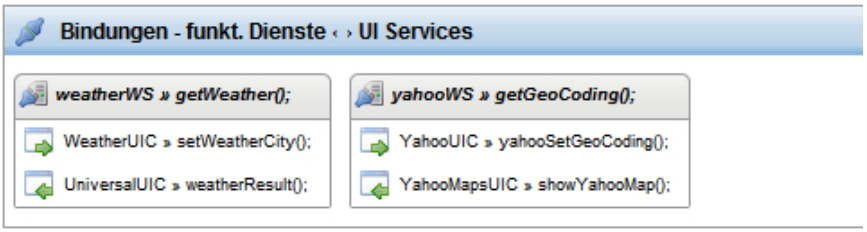


Abbildung 4.6 Übersicht gekoppelter Eingabe- bzw. Ausgabe-Operationen von UI Komponenten an Operationen funktionaler Dienste

Neben der Auswahl und Integration passender Eingabe- bzw. Ausgabe-Operationen von UI Komponenten wurde, wie bereits angedeutet, die Interaktion zwischen gekoppelten UI Komponenten und funktionalen Diensten realisiert. Die Interaktion wird dabei durch den Anwendungsnutzer ange-

steuert, indem eine Eingabe-Operation der gekoppelten UI Komponenten aufgerufen wird. Am Beispiel der prototypisch umgesetzten und in Abbildung 4.6 dargestellten Operation *setWeatherCity()* der UI Komponente *WeatherUIC* soll die Interaktion zwischen UI Komponenten und funktionalen Diensten im Folgenden beispielhaft erläutert werden.

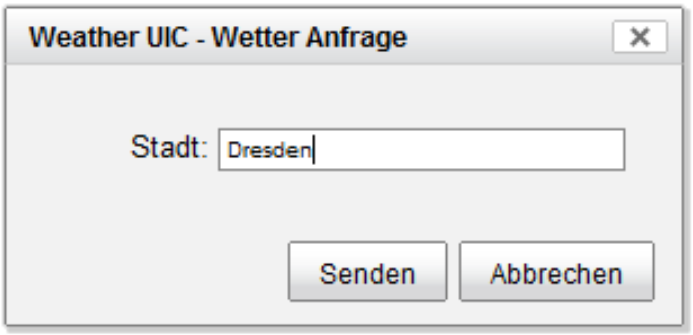


Abbildung 4.7 Beispiel einer aufgerufenen Eingabe-Operation einer UI Komponente

Nach Aufruf der Eingabe-Operation *setWeatherCity()* durch den Anwendungsnutzer wird die Operation innerhalb der Webanwendung, wie in Abbildung 4.7 dargestellt, geladen. Für eine erfolgreiche Interaktion mit der an diese Operation gekoppelten Operation *getWeather()* des funktionalen Dienstes *weatherWS* ist die Angabe einer beliebigen Stadt als Parameter notwendig. Nach Eingabe des Parameters durch den Nutzer in die erzeugte Formularkomponente und Anklicken des „Senden“-Buttons werden die Eingabe-Daten über die Ereignis-Steuerung der Client-Runtime mit Informationen über die an der Interaktion beteiligte Operation des funktionalen Dienstes an die System-Anwendung und hier gezielt an die prototypisch umgesetzte Service-Access Komponente für den Aufruf des funktionalen Dienstes weitergeleitet.
Das Ergebnis des Aufrufes des funktionalen Dienstes wird auf Basis einer gespeicherten Kopplung in eine entsprechende Objekt-Repräsentation überführt und an die Runtime der Client-Anwendung übermittelt.

Hier wird anhand der Informationen zur Operation *getWeather()* des funktionalen Dienstes *weatherWS*, die das Ergebnis lieferte, sowie das Ergebnis selbst über das Ereignis-System automatisch an die für die Datenausgabe des Dienstes registrierten Operation *weatherResult()* der UI Komponente *UniversalUIC* weitergeleitet und die Operation entsprechend ausgeführt. Abbildung 4.8 auf der folgenden Seite zeigt die Ausführung der

Abbildung 4.8 Beispiel einer aufgerufenen Ausgabe-Operation einer UI Komponente

Operation *weatherResult()* mit den übergebenen Daten aus dem Aufruf des funktionalen Dienstes.
Für die Verarbeitung sowie die Visualisierung der übergebenen Daten innerhalb der Webanwendung ist nach Aufruf der Operation ausschließlich die Operation bzw. die UI Komponente selbst verantwortlich. Für das Thema der Arbeit besteht an dieser Stelle keine Notwendigkeit weitere Möglichkeiten der Interaktion zwischen Webanwendung bzw. implementierter Client-Runtime und Ausgabe-Operation einer UI Komponente zu realisieren.

Funktionale Dienste beschreiben die Reaktion auf Fehler in der Ausführung einer Dienst-Operation in der Regel durch die Angabe eines *Fault*-Elementes innerhalb der Dienstbeschreibung. Sollte eine solche Angabe innerhalb der Beschreibung nicht definiert worden sein, so kann die Fehler-Ausgabe der Operation auch nicht an eine passende Operation einer UI Komponente gekoppelt werden. Dennoch kann es zu Fehlern während der Ausführung einer Dienst-Operation kommen, die in aller Regel eine Ausnahmebehandlung (Exception) innerhalb der System-Access Komponente zur Folge haben. In einem solchen Fall sollte der Anwendungsnutzer über mögliche Ausnahmen nicht nur bei der Kommunikation mit funktionalen Diensten sondern während des gesamten Anwendungsverlaufes informiert werden können.

Abbildung 4.9 Ausgabe-Dialog eines Fehlers aus der Kommunikation mit einem Dienst (Ansicht Client-Anwendung)

Hierzu wurde der Prototyp um eine Funktionalität zur Generierung von Dialog-Fenstern erweitert, die es ermöglicht, Daten entsprechend ihres Ausnahme-Typs zu präsentieren. Die Funktionalität ermöglicht dabei die Darstellung der Ausnahme-Typen *Error*, *Information*, *Warning* und *Question* in Anlehnung an Desktop-basierte Betriebssysteme. Abbildung 4.9 verdeutlicht beispielhaft den Ausnahme-Typ *Error* bei einem fehlerhaften Aufruf der Operation *getWeather()* des funktionalen Dienstes *weatherWS* durch die Service-Access Komponente. Die Operation des aufgerufenen funktionalen Dienstes antwortet auf die Anfrage zum Wetter in „dresde“ mit dem Fehler, dass dieser Ort nicht bekannt sei.

4.2 Fazit

In diesem Kapitel wurde die prototypische Implementierung der in Kapitel 3.2 vorgeschlagenen Systemarchitektur zur Kopplung funktionaler Dienste und UI Services vorgestellt. Dabei wurde vor allem auf die prototypische Umsetzung des wichtigsten Bereiches der UI Service-Discovery Komponente in Form des Discovery Prozesses eingegangen.
Um den Prozess des Discovery, aber auch der Integration von UI Komponenten zu ermöglichen, wurde neben der System-Anwendung mit den wohl wichtigsten Aufgaben des Discovery, der Kommunikation mit funktionalen Diensten sowie der Modellierung von Objekten aus Schnittstelleninformationen und zu kommunizierenden Daten zudem eine umfangreiche Client-

Anwendung realisiert, die sowohl die Integration als auch die Interaktion mit UI Komponenten über die angebotenen Funktionalitäten ermöglicht. Die umgesetzte Client-Anwendung dient damit der Visualisierung der Aufgabenstellung der vorliegenden Arbeit.
Das folgende Kapitel 5 befasst sich mit der Validierung der vorgeschlagenen Systemarchitektur mit Hilfe der prototypischen Implementierung an einem konkreten Beispiel und geht anschließend auf die Bewertung der an eine Systemarchitektur gestellten Anforderungen ein. Weiterhin werden Probleme bei der Umsetzung aufgezeigt und mögliche Lösungen skizziert.

Kapitel 5

Evaluierung

In den vorangegangenen Kapiteln wurde ein Konzept für die Kopplung funktionaler Dienste und UI Services bzw. den darin gekapselten UI Komponenten entwickelt und mit einer prototypischen Implementierung umgesetzt. Das vorliegende Kapitel befasst sich mit der Evaluierung des Konzeptes sowie der Validierung der prototypischen Umsetzung an einem konkreten Beispiel. Neben der Bewertung der für die zu entwickelnde Systemarchitektur aufgestellten Anforderungen wird abschließend auf einige Probleme verwiesen, die sich mit der Themenbearbeitung ergaben.
Anzumerken ist an dieser Stelle, dass eine umfangreiche Evaluierung nicht möglich ist, da einerseits funktionale Dienste fehlen, die innerhalb des Prototypen eingesetzt werden können, wie die im Folgenden vorgestellte Validierung des Prototypen zeigt, andererseits UI Servicebeschreibungen sowie funktionale UI Komponenten nicht bzw. nur ungenügend vorhanden waren.

5.1 Validierung des Prototypen

Der vorliegende Abschnitt befasst sich mit der Validierung des entwickelten Prototypen. Am Beispiel der *Amazon Web Services* (*AWS*) soll dabei die Verwendbarkeit der *Product Advertising API* demonstriert werden. Im folgenden Abschnitt 5.1.1 wird zunächst allgemein auf die *Product Advertising API* eingegangen, bevor im Abschnitt 5.1.2 die Interaktion des Nutzers mit der prototypisch umgesetzten Webanwendung sowie die Integration von UI Komponenten für eine Kommunikation mit der durch die *Product Advertising API* angebotenen Operation *ItemSearch* vorgestellt wird.

5.1.1 AWS - Product Advertising API

Die Product Advertising API der Amazon Web Services, früher auch als Amazon Associated Web Service bezeichnet, ermöglicht Nutzern den Zugriff auf Daten der durch Amazon angebotenen Produkte. Weiterhin bietet die API Funktionalitäten um beispielsweise eine Produkt-Suche durchführen zu können, auf Kundenrezensionen zuzugreifen oder Produktvergleiche zu realisieren. Amazon fördert eine breite Unterstützung für Entwickler, indem die API in unterschiedlichen Sprachen angeboten und durch eine Reihe verschiedener Beispiele zur Implementierung ergänzt wird.
Der Zugriff auf die Funktionalitäten der API erfolgt generell über HTTP-Anfragen. Dabei bietet die Product Advertising API neben einer SOAP-Schnittstelle auch eine entsprechende Schnittstelle für die Kommunikation, die den REST-Prinzipien folgt. Für einen erfolgreichen Zugriff auf die API sind die Parameter *Service*, *AWSAccessKeyId* sowie die aufzurufende *Operation* des mit Service spezifizierten Dienstes als Name-Wert-Paare innerhalb jeder Anfrage zu definieren. Darüber hinaus kann jede einzelne Operation die Angabe zusätzlicher Parameter für einen erfolgreichen Zugriff erforderlich machen.
Im August 2009 wurde die aktuell vorliegende Version der Product Advertising API dahingehend geändert, dass jede Anfrage neben den notwendigen Parametern auch eine Authentifizierung erfordert, um die Echtheit des Aufrufers bzw. der aufrufenden Anwendung zu bestätigen. Sowohl für REST-basierte Aufrufe als auch für Zugriffe über die SOAP-Schnittstelle enthält der Developer Guide [38] der Product Advertising API weitere Informationen. Die notwendige Signatur wird sowohl für die Authentifizierung von Aufrufen über REST als auch über SOAP durch Kalkulation eines HMAC (keyed-Hash Message Authentication Code) mit anschließender SHA256- sowie Base64-Kodierung realisiert. Bei einer REST-basierten Kommunikation wird die erstellte Signatur direkt in die Anfrage-URI eingebettet. Für SOAP kann hier sowohl eine Authentifizierung über WS-Security erfolgen, sofern die eingesetzte Entwicklersprache dies unterstützt, oder ohne WS-Security, indem der ermittelte Wert als Signatur-Parameter des SOAP-Headers übertragen wird.
Mit dem folgenden Abschnitt wird auf einen Anwendungsfall eingegangen, der einen Zugriff auf die aktuelle Version der Amazon Product API über die angebotene WSDL-beschriebene SOAP-Schnittstelle realisiert. Die REST-Schnittstelle der API kam bei der Umsetzung des folgenden Szenarios nicht in Frage, da die angebotene REST-Schnittstelle bisher nicht über eine

WADL-Beschreibung verfügt und damit ein automatisierter Zugriff auf die API sowie das Discovery von potentiellen UI Komponenten nicht umgesetzt werden kann.

5.1.2 Verwendung des Prototypen am Beispiel

Durch die bereits angesprochene Notwendigkeit der Authentifizierung jedes Zugriffes auf die Amazon Product Advertising API ergab sich ein wesentliches Problem: die notwendige Authentifizierung wird innerhalb der WSDL-Beschreibung der API nicht definiert. Aus diesem Grund wurde ein sogenannter Proxy-Dienst entwickelt, der die SOAP-Schnittstelle bezogen auf die für das Beispiel-Szenario ausgewählte Operation *ItemSearch* der Product Advertising API nachbildet. Die einzige Funktionalität, die dieser Proxy-Dienst kapselt, ist die Umsetzung der Authentifizierung gegenüber der API, wobei Aufrufe des Proxy-Dienstes ansonsten unverändert an die API weitergereicht und Antworten eines API-Aufrufes wiederum direkt an die umgesetzte Service-Access Komponente der System-Anwendung zurückgeliefert werden.
Auch ein Aufruf der Operation ItemSearch erfordert neben den bereits angeführten Parametern, die jede Anfrage beinhalten muss, weitere notwendige Parameter, die bei einem Zugriff als Name-Wert-Paare zu übergeben sind. Dazu gehören neben der Angabe von Suchbegriffen über den Parameter *Keywords* auch der Parameter *SearchIndex*, mit dem die Kategorie (z. B.: Books, DVD, ALL) festgelegt wird, in der nach den betreffenden Begriffen gesucht werden soll. Durch Angabe des optionalen Parameters *ResponseGroup* kann das Ergebnis der Suche zudem eingeschränkt bzw. erweitert werden, so dass eine mehr oder weniger umfangreiche Auswahl an Informationen zu Produkten durch den Aufruf der Operation zurückgeliefert wird.
Die vollständige WSDL-Beschreibung des Proxy-Dienstes findet sich aufgrund der Übersichtlichkeit im Anhang (Listing A.3) der Arbeit angeführt. Mit Abbildung 5.1 auf der folgenden Seite wird der Einsatz des Proxy-Dienstes sowie der Datenfluss ausgehend von der implementierten Service-Access Komponente der System-Anwendung bis hin zum Zugriff auf die Amazon Product Advertising API verdeutlicht.

Da auf Seite der UI Services bis auf die für die prototypische Umsetzung realisierten UI Services bisher kaum UI Komponenten zur Verfügung stehen, wurde zum Einen ein bereits prototypisch umgesetzter UI Service um eine UI Komponente zur Dateneingabe erweitert, zum Anderen ein UI Service mit einer UI Komponente für die Datenausgabe neu erstellt sowie die

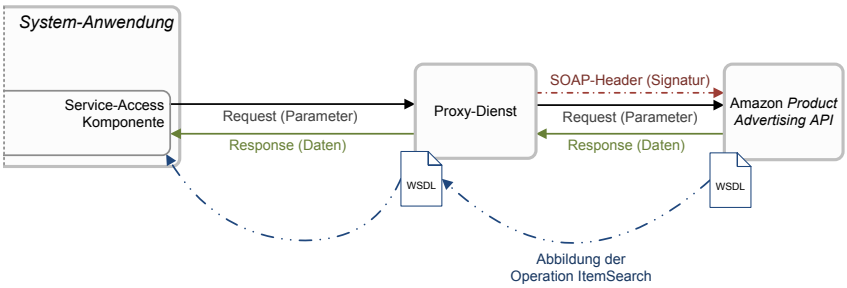


Abbildung 5.1 Proxy-Dienst als „Bindeglied“ für die Kommunikation

entsprechenden UISDL-Beschreibungen definiert bzw. angepasst. Bei den UI Komponenten handelt es sich im Wesentlichen um JavaScript-Klassen, welche die Realisierung von UI Bestandteilen zur Eingabe (Formular) sowie zur Ausgabe (Tabelle) von Daten über unterschiedliche öffentliche Operationen ermöglichen.

Die vollständigen UISDL-Beschreibungen der umgesetzten UI Services finden sich aufgrund der Übersichtlichkeit ebenfalls im Anhang (Listing A.4, A.5) der vorliegenden Arbeit angeführt.

Für den Prozess des Discovery sowie für die Integration wird, wie bereits bei der Vorstellung der prototypischen Umsetzung in Kapitel 4.1.1 beschrieben, die Operation ItemSearch zur Laufzeit zunächst durch den Anwendungsnutzer in der Nutzeroberfläche der Client-Anwendung ausgewählt (vgl. Abbildung 4.1). Anhand der aus der zugehörigen WSDL-Beschreibung des Proxy-Dienstes extrahierten Informationen der Operation ItemSearch wird daraufhin innerhalb der System-Anwendung der Prozess des Discovery aktiviert. Dieser vergleicht nun alle Schnittstelleninformationen dem System bekannter Operationen von UI Komponenten bezüglich der Dateneingabe und -ausgabe mit den extrahierten Informationen der WSDL-Beschreibung des Proxy-Dienstes und versucht so eine Übereinstimmung festzustellen.

Geht man dabei genauer auf den innerhalb der WSDL-Beschreibung in Listing A.3 definierten Parameter *ItemSearchResponse* der Datenausgabe sowie den innerhalb der UISDL-Beschreibung in Listing A.5 definierten Parameter *ItemSearchRequest* der Dateneingabe ein, so wird deutlich, dass sich diese Parameter zwar ähnlich sind, grundsätzlich aber nicht übereinstimmen. Der Discovery-Prozess durchläuft bei diesem Vergleich die in Kapitel 3.2.2.1 angeführte zweite Phase des Schnittstellen-Vergleiches und stellt darüber fest, dass die Schnittstellen gekoppelt werden können. Gleiches gilt auch für den Parameter *ItemSearchRequest* der WSDL-beschriebenen Dateneingabe des Proxy-Dienstes sowie den mit Listing A.4 definierten Parameter *respon-*

seItemSearchVars der UISDL-beschriebenen Datenausgabe des UI Services.

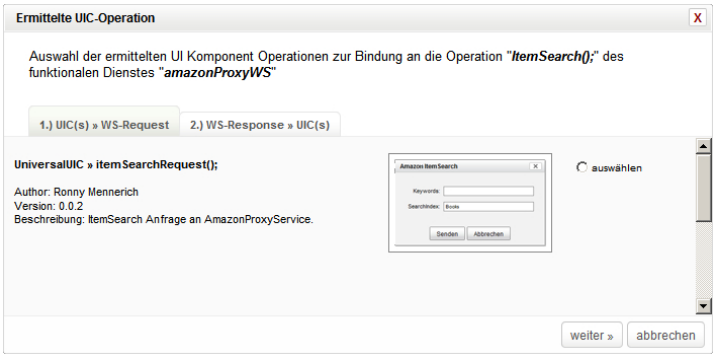


Abbildung 5.2 Dialog-Ansicht durch den Discovery Prozess ermittelter UIC Eingabe-Operation (Ansicht *UIC(s)* ⇒ *WS-Request*)

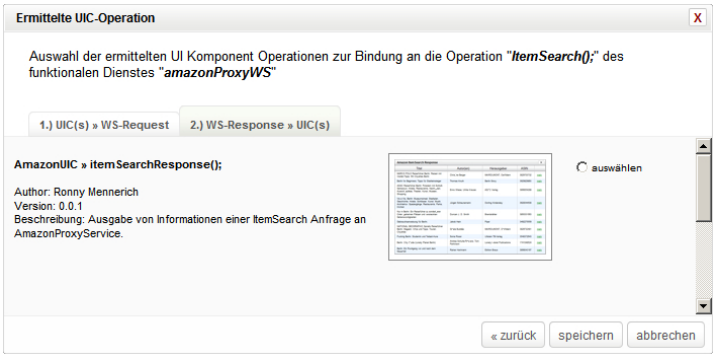


Abbildung 5.3 Dialog-Ansicht durch den Discovery Prozess ermittelter UIC Ausgabe-Operationen (Ansicht *WS-Response* ⇒ *UIC(s)*)

Damit können dem Nutzer sowohl für Dateneingabe als auch Datenausgabe des Proxy-Dienstes potentielle Kandidaten von UI Komponenten für eine Kopplung an den funktionalen Dienst vorgeschlagen werden. Die Auswahl der UI Komponenten durch den Nutzer entspricht dem bereits angesprochenen Vorgehen, das mit den Abbildungen 5.2 sowie 5.3 verdeutlicht wird.

Nach Auswahl der UI Komponenten für die Kopplung an die ausgewählte Operation ItemSearch werden innerhalb der System-Anwendung die Bindungen für den Datenaustausch zwischen Service-Access Komponente und Amazon Product Advertising API über den Proxy-Dienst umgesetzt. Zudem werden Informationen der ausgewählten UI Komponenten für eine Integration in die Nutzeroberfläche an die Client-Anwendung übermittelt. Abbildung 5.4 zeigt die visuelle Repräsentation der UI Komponenten, die zur Laufzeit in die Nutzeroberfläche der entwickelten Client-Anwendung integriert werden und dem Nutzer anschließend für eine Interaktion zur Verfügung stehen.

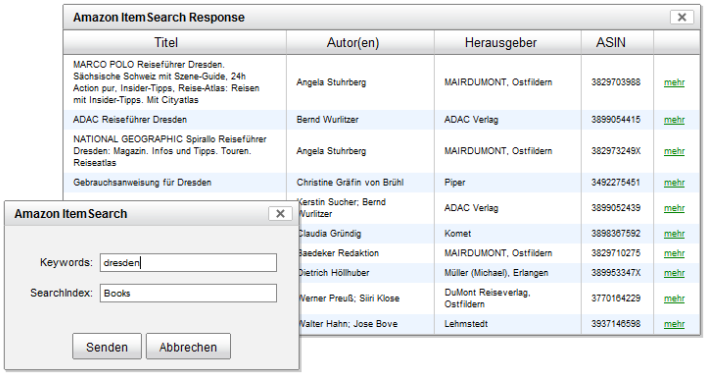


Abbildung 5.4 UI Komponenten für Dateneingabe (Vordergrund, Formular) und Datenausgabe (Hintergrund, Tabelle) der Operation *ItemSearch*

5.2 Bewertung der aufgestellten Anforderungen

Neben der zuvor vorgestellten Validierung der prototypischen Implementierung an einem konkreten Beispiel sollen die folgenden Abschnitte dazu dienen, die Umsetzung der im Vorfeld aufgestellten Anforderungen an eine Systemarchitektur zu untersuchen und konstruktiv zu bewerten.

[R1] Allgemeingültigkeit

Aufgrund der Mächtigkeit von Beschreibungsformaten wurde der Prototyp derart umgesetzt, dass er in der Lage ist, das WSDL-Format zur Beschreibung funktionaler Dienste in einem für das Thema der Arbeit notwendigen Umfang zu verarbeiten. Weitere Formate funktionaler Dienste, wie sie bereits in den Grundlagen angeführt wurde, lassen sich jedoch problemlos über eine entsprechende Verarbeitungskomponente integrieren. Daneben ist der Prototyp in der Lage, die in Kapitel 3.1 angeführten, für das Thema der Arbeit notwendigen Merkmale und Kriterien eines UISDL-Formates zur Beschreibung von UI Services verarbeiten zu können.

[R2] Discovery

Dem umgesetzten Prototyp ist es möglich, die in der Konzeption hinsichtlich der UI Service-Discovery Komponente angeführten Phasen eins und zwei bezüglich des Discovery nicht Domain-spezifischer UI Komponenten umzusetzen. Da eine Vielzahl verfügbarer Schnittstellenbeschreibungen funktionaler Dienste über eine komplexe Datentyp-Definition verfügen, wurde neben den String-basierten Matching-Verfahren für einen Parameter-Vergleich außerdem ein Ansatz für den Vergleich der Strukturen komplex definierter Parameter bis zur Knoten-Ebene 2 umgesetzt. Aus diesem Struktur-Vergleich

ergibt sich auch die Möglichkeit, angepasste Schnittstellen zu identifizieren und als Ergebnis zu liefern. Die Umsetzung der Phase drei konnte an dieser Stelle aufgrund eines sehr komplexen Struktur-Vergleiches nicht realisiert werden.

[R3] Generierung von Kopplungen
Das vorgestellte Konzept wurde mit dem Prototypen dermaßen umgesetzt, so dass dieser in der Lage ist, anhand der gekoppelten Schnittstellenbeschreibungen eine Anpassung der auszutauschenden Ergebnisse vorzunehmen und diese als Objekte für den Datenaustausch zu modellieren.

[R4] Interaktion zwischen System-Komponenten
Die prototypische Umsetzung des entwickelten Konzeptes verfügt in Bezug auf das unterstützte Beschreibungsformat funktionaler Dienste über eine erweiterbare Möglichkeit, die Schnittstellen funktionaler Dienste aufzurufen als auch Ergebnisse in Form der modellierten Objekte zwischen funktionalen Diensten und UI Komponenten auszutauschen.

[R5] Leistung und Effizienz
Die Leistung und Effizienz der prototypischen Umsetzung liefert wie gefordert Ergebnisse hinsichtlich der Suche und Auswahl von UI Komponenten in vertretbarer Zeit.

[R6] Dynamik zur Laufzeit
Die prototypisch umgesetzte Systemarchitektur wurde so realisiert, dass ein Matching von Schnittstellen zur Laufzeit stattfinden kann und UI Komponenten damit dynamisch gefunden werden können, womit eine der wichtigsten Anforderungen an die Systemarchitektur umgesetzt wurde. Weiterhin wurde ein Ansatz entwickelt, der es ermöglicht, ausgewählte UI Komponenten zur Laufzeit in eine Nutzeroberfläche zu integrieren (Laden, Nachladen) und einen Austausch von UI Komponenten vorzunehmen.

[R7] Modularität und Erweiterbarkeit
Die prototypische Implementierung unterstützt die Modularität insofern, dass die entwickelten Komponenten wie beispielweise der UISDL-Parser auch Anwendung in anderen Szenarien und System-Anwendungen finden können.

[R8] Plattformunabhängigkeit und Portabilität

Die implementierte Webanwendung setzt mit *PHP: Hypertext Preprocessor* (*PHP*) und *JavaScript* (*JS*) auf aktuell weit verbreitete Programmier- und Scriptsprachen im Rahmen der Umsetzung von Webanwendungen. Damit kann der umgesetzte Prototyp unabhängig von der Plattform und dem verwendeten Betriebssystem eingesetzt werden. Die Implementierung wurde zudem so plattformunabhängig realisiert, dass auch die aktuell gängigen Web-Browser Unterstützung finden und damit verwendet werden können.

[R9] Benutzbarkeit

Im Rahmen der Arbeit wurden keine umfangreichen Tests durch eine größere Nutzergruppe durchgeführt, wodurch an dieser Stelle keine eindeutigen Angaben zur Benutzbarkeit des entwickelten System gemacht werden können. Nach Meinung des Autors machen die in Kapitel 4.1 präsentierten Abbildungen deutlich, dass es sich bei der prototypischen Umsetzung um eine einfache, intuitive und den Nutzer bei der Arbeit mit dem System unterstützende Webanwendung handelt. Grundsätzlich wird dem Nutzer der Umgang mit dem entwickelten System dahingehend erleichtert, dass wesentliche Interaktionen über nutzerfreundliche Dialoge geführt werden, um dem Nutzer die Bedienung der Anwendung zu erleichtern.

Bewertung der aufgestellten Anforderungen	
R1 Allgemeingültigkeit	o
R2 Discovery	o
R3 Generierung von Kopplungen	+
R4 Interaktion zwischen System-Komponenten	o
R5 Leistung und Effizienz	o
R6 Dynamik zur Laufzeit	+
R7 Modularität und Erweiterbarkeit	+
R8 Plattformunabhängigkeit und Portabilität	+
R9 Benutzbarkeit	+

\+ Erfüllt, **o** Teilweise erfüllt, **-** Nicht erfüllt

Tabelle 5.1 Bewertung der aufgestellten Anforderungen

Diese allgemeine Bewertung zeigt deutlich, dass die in Kapitel 3.2.1 aufgestellten Anforderungen an eine Systemarchitektur mit der prototypischen Implementierung weitestgehend umgesetzt werden konnten. Die Tabelle 5.1

fasst die Bewertung der aufgestellten Anforderungen noch einmal zusammen, während der folgende Abschnitt 5.3 aufgetretene Probleme bei der Umsetzung einiger wesentlicher Anforderungen erneut auffasst und mögliche Lösungen diskutiert.

5.3 Aufgetretene Probleme

Mit diesem Abschnitt soll auf Probleme bei der Umsetzung und auch mögliche Lösungswege eingegangen werden, wobei einige wesentliche Anforderungen erneut aufgefasst werden.

[R1] Allgemeingültigkeit
Mit dem Kapitel 2.1 wurde die Verwendung des Begriffes des funktionalen Dienstes für die vorliegende Arbeit konkretisiert, wobei u. a. auf WADL-beschriebene REST-basierte Dienste verwiesen wurde. Da mittlerweile eine Vielzahl von Dienst-Anbietern neben WSDL-beschriebenen SOAP-basierten Schnittstellen auch auf REST-basierte Schnittstellen setzen oder teilweise nur diese anbieten, sollte das Beschreibungsformat WADL innerhalb der Systemarchitektur unterstützt werden. Aktuell sind in diesem Bereich jedoch nur bedingt WADL-Beschreibungen für Schnittstellen funktionaler Dienste verfügbar, womit diese zunächst zu erstellen und durch entsprechende Parser zu unterstützen sind.

[R2] Discovery
Neben der Umsetzung String-basierter Verfahren zeigte sich, dass ein Großteil von Schnittstellen sehr komplex beschrieben ist. Mit der Unterstützung des XML-Schema für die Definition von Datentypen innerhalb von Beschreibungsdokumenten ergeben sich nach Meinung des Autors teils unüberschaubare Datentyp-Definitionen (vgl. WSDL der Amazon Product Advertising API [38]), deren Verarbeitung einen hohen Aufwand gerade bei strukturellen Vergleichen notwendig machen, der mit der prototypischen Umsetzung der vorliegenden Arbeit jedoch in dem gewünschten Umfang nur teilweise realisiert werden konnte. Damit sollten zunächst eingesetzte Parser dermaßen entwickelt werden, dass sie das XML-Schema unterstützen; zudem sollten Matching-Verfahren im Bereich des strukturellen Vergleichens von Schnittstellen untersucht werden, die dann an gegebener Stelle durch die hier umgesetzten, aber auch durch weitere String-basierte Verfahren erweitert werden können.

Weiterhin bleibt innerhalb der Arbeit offen, wie die vorgeschlagenen und umgesetzten Matching-Verfahren zu gewichten sind und welche Aussagekraft den Ergebnissen dieser und auch weiterer Verfahren bei einem Vergleich von Schnittstellen beigemessen werden kann. Weitere wissenschaftliche Untersuchungen könnten daher im Bereich des Schnittstellen- bzw. Schema-Matching zu neuen und vor allem grundlegenden Erkenntnissen führen.

[R4] Interaktion zwischen System-Komponenten

Schon bei der Umsetzung der prototypischen Implementierung, aber vor allem das verwendete Beispiel zur Evaluierung haben gezeigt, dass sich im Bereich der funktionalen Dienste eine wesentliche Neuerung ergibt. Dabei lassen sich Schnittstellen zwar durch ein Format wie beispielsweise WSDL näher beschreiben, jedoch wird ohne Weiteres die Verwendung von Sicherheitsmechanismen, wie der durch Amazon kürzlich eingeführten Authentifizierung, nicht unterstützt und noch weniger innerhalb eines Beschreibungsdokumentes auf die Verwendung einer Authentifizierung verwiesen. Im Bereich WSDL-beschriebener funktionaler Dienste wurde für diesen Umstand WS-Security[1] als Erweiterung entwickelt, die Sicherheitsaspekte bei funktionalen Diensten berücksichtigt. Aufgrund fehlender Informationen bezüglich zu verwendender Sicherheitsmechanismen ist eine Laufzeit-basierte Verarbeitung von Schnittstellen jedoch quasi unmöglich.
Weiterhin stand der zugrunde liegende Kommunikations- bzw. Interaktionsmechanismus zwischen funktionalen Diensten und UI Komponenten nicht im Fokus der Betrachtungen innerhalb dieser Arbeit. Die Arbeit setzt grundlegend den als Polling bezeichneten Anfrage-Antwort-Mechanismus um. In diesem Bereich sollten weitere Untersuchungen hinsichtlich funktionaler Dienste sowie auf Seite der UI Komponenten zeigen, wie ein Streaming-Mechanismus, also der kontinuierliche Austausch von Daten, für die Kommunikation und Interaktion umgesetzt werden kann.

[R5] Leistung und Effizienz

Mit der Zahl an vorhandenen und zu analysierenden UI Services, die über die Zeit anwachsen kann, sinkt mit dem implementierten Ansatz der festen Definition von UI Services über deren URI sowie dem Parsen aller bekannten UI Services für einen einzigen Vergleich die Leistung des Systems, wiederum steigt so der Analyse-Aufwand, d. h. die Zeit für einen Vergleich wächst.
Hier sollte das System zu gegebener Zeit durch Mechanismen erweitert werden, die es ermöglichen, potentielle Kandidaten von UI Komponenten für

[1] http://www-128.ibm.com/developerworks/library/specification/ws-secure/

eine Kopplung an einen funktionalen Dienst zu speichern. Möglichkeiten dazu könnte eine UI Service Registry bieten, in der neben der URI-Referenz der Beschreibung eines UI Services auch die möglichen koppelfähigen funktionalen Dienste mit abdeckt werden, womit die Leistung des Systems erhöht werden kann. Denkbar wäre zudem eine Art Management-Komponente, die Beziehungen zwischen einem funktionalen Dienst und potentiellen Kandidaten von UI Komponenten speichert und zur Laufzeit zur Verfügung stellt.
Diese Erweiterungen zielen jedoch darauf ab, einen Teil des momentan sehr komplexen Discovery Prozesses schon bei der Veröffentlichung neuer UI Services auszuführen, um somit die Anzahl zu analysierender UI Services bei einer Suche zur Laufzeit weitestgehend zu minimieren.

Ein weiteres allgemeines Problem, welches sich mit der Bearbeitung der Thematik ergab, war der zusätzliche Aufwand zur Erstellung beispielhafter und aussagekräftiger UI Komponenten sowie der entsprechenden Beschreibungsdokumente in der für die Umsetzung der Arbeit notwendigen Form. Ohne diesen zusätzlichen praktischen Aufwand wäre eine konstruktive Bewertung sowie die Identifikation wesentlicher Probleme hinsichtlich der vorgeschlagenen Systemarchitektur sowie des eingeführten UISDL-Beschreibungsformates nur unzureichend möglich.

5.4 Fazit

Zusammenfassend lässt sich festhalten, dass mit der prototypischen Implementierung ein umfangreicher Teil der vorgeschlagenen Systemarchitektur umgesetzt werden konnte. Dabei ergaben sich neue Erkenntnisse und Problemstellungen, die Teil weiterer wissenschaftlicher Betrachtungen sein sollten. Neben der umfangreichen Umsetzung im Bereich der System-Anwendung setzt das realisierte Dialog-Konzept der Client-Anwendung, in Anlehnung an die Arbeit mit Autorenwerkzeugen, die vorliegende Thematik der Arbeit ansprechend in visueller Weise um.
In Bezug auf den wohl wichtigsten Teil der Arbeit, den Prozess des Discovery von UI Komponenten, lässt sich festhalten, dass die Arbeit in ihrer vorliegenden Form als Grundlagenarbeit im Bereich der Kopplung von funktionalen Diensten und UI Services zu sehen ist. Die vorgeschlagenen Aspekte des UISDL-Formates zur Beschreibung von UI Services sowie von darin gekapselten UI Komponenten unterstützt durch die herausgestellten Kriterien und Merkmale gerade in funktioneller Hinsicht den Prozess der

Suche kompositer Schnittstellen. Mit Hilfe der vorgestellten und umgesetzten Matching-Verfahren konnte deutlich gemacht werden, dass es möglich ist, den Discovery Prozess auch dynamisch zur Laufzeit auszuführen. Jedoch zeigen die aufgetretenen Probleme bei der Umsetzung der Systemarchitektur auch auf, dass eine Vielzahl weiterer Untersuchungen im Rahmen konkreter Nachfolgearbeiten notwendig ist, auf die bereits während der Arbeit verwiesen wurde und im Ausblick des folgenden Kapitels nochmals eingegangen wird.

Abschließend ist anzumerken, dass die prototypische Implementierung durch fehlende UI Komponenten sowie grundlegende UISDL-Beschreibungen einen erheblichen Mehraufwand notwendig machte, um aussagekräftige und vor allem ergebnisorientierte Anwendungsszenarien unter dem Aspekt der Umsetzung der vorgeschlagenen Systemarchitektur zu entwickeln.

Kapitel 6

Zusammenfassung und Ausblick

Das letzte Kapitel dieser Arbeit fasst die erreichten Ergebnisse aus den zurückliegenden Kapitel zusammen und gibt Anregungen für weitere Betrachtungen, die sich aus der Thematik der vorliegenden Arbeit ergeben.
Da die Arbeit im Kontext des Forschungsprojektes CRUISe zu sehen ist, wurde zunächst eine Einordnung der Arbeit in das Projekt vorgenommen. Anschließend wurde auf die Begriffe des funktionalen Dienstes, des User Interface Service sowie der User Interface Komponente näher eingegangen, um ein allgemeines Verständnis dieser Begriffe für die Arbeit zu gewährleisten. Bezüglich der funktionalen Dienste wurde im Weiteren mit WSDL und WADL auf die wichtigsten Beschreibungsformate in diesem Bereich näher eingegangen sowie ein Überblick über semantische Erweiterungen durch eine Einführung von SAWSDL gegeben. Des Weiteren wurden einige wesentliche Verfahren des Schema-Matching grundlegend betrachtet, wobei diese im Gegensatz zu den Betrachtungen in SAWSDL weniger manuell zur Designzeit sondern im Rahmen der Arbeit vielmehr dynamisch zur Laufzeit ausgeführt werden müssen. Abschließend wurden innerhalb des theoretischen Teils einige Konzepte vorgestellt, die sich mit wesentlichen Ansätzen zur Integration von UI Komponenten sowie der Erstellung kompositer Anwendungen basierend auf funktionalen Diensten und UI Komponenten befassen.
Bereits aus den angeführten Informationen bezüglich des Projektes CRUISe, aber auch aufgrund der vorgestellten Konzepte ergab sich für die in Kapitel 3 zu entwickelnde Systemarchitektur zunächst die Notwendigkeit der erneuten Betrachtung wesentlicher Merkmale und Kriterien an ein Beschreibungsformat für UI Services und UI Komponenten. Der Fokus lag hier, im

Gegensatz zu den bisherigen Betrachtungen im Rahmen von CRUISe, auf der Beschreibung funktioneller Aspekte, um eine für das Thema der Arbeit notwendige Kopplung realisieren zu können. Ausgehend von den angeführten Grundlagen sowie der konzeptionellen Formulierung des Beschreibungsformates wurden Anforderungen an eine Systemarchitektur formuliert sowie die wesentlichen Elemente dieser Architektur im Folgenden umfassend beschrieben.
Anschließend wurde mit Kapitel 4 die entwickelte Systemarchitektur prototypisch umgesetzt. Im Mittelpunkt standen hierbei die Umsetzung einer Komponente für die Suche nach potentiellen UI Komponenten zu einer Operation eines funktionalen Dienstes sowie die Modellierung von Objektklassen auf Basis der Schnittstellenbeschreibungen, die angereichert durch Daten zu konkreten Objekten der Laufzeitumgebung abgeleitet und zwischen funktionalen Diensten und UI Komponenten kommuniziert werden sollten. Darüber hinaus wurde eine als Webanwendung realisierte Nutzeroberfläche zur Visualisierung des gesamten Anwendungsworkflows entworfen, welche zusätzlich eine Integration ausgewählter UI Komponenten zur Laufzeit ermöglicht.
Mit dem Kapitel 5 wurde der Prototyp der Anwendungsumgebung an einem konkreten Beispiel, der Amazon Product Advertising API, sowie entwickelten UI Komponenten und den entsprechenden UISDL-Beschreibungen hinsichtlich der umgesetzten Funktionalitäten validiert. Ausgehend von dieser Validierung wurde eine konstruktive Bewertung der aufgestellten Anforderungen an die entwickelte Systemarchitektur vorgenommen. Nicht zuletzt wurde auf Probleme bei der Umsetzung der Anforderungen verwiesen, wobei auch mögliche Lösungsansätze diskutiert wurden.
Das entwickelte Konzept macht vor allem durch den umgesetzten Prototypen sowie die angeführten Probleme und Lösungsansätze deutlich, dass eine Kopplung zwischen funktionalen Diensten und UI Services bzw. den gekapselten UI Komponenten nicht ohne Weiteres realisiert werden kann. Die Idee, bereits bestehende UI Komponenten als wiederverwendbare Bestandteile einer Nutzeroberfläche zu betrachten, um diese in einer Vielzahl von Szenarien einzusetzen, bedingt neue Untersuchungsschwerpunkte im Rahmen der vorliegenden Thematik. Grundsätzlich lässt sich festhalten, dass das entwickelte und in seiner vorliegenden Form umgesetzte Konzept nicht grundsätzlich zu einer Laufzeitanwendung in Form eines Authorentools führt, welches letztendlich durch einen Endanwender bedient wird. Es kann jedoch als hilfreiches Mittel die Designzeit und damit den Entwickler beim Entwurf kompositer Anwendungen unterstützen.

6.1 Ausblick

Bereits in Kapitel 5.3 wurden Probleme identifiziert, die innerhalb der vorliegenden Grundlagenarbeit nicht in vollem Umfang bearbeitet werden konnten, wobei auch mögliche Lösungsstrategien formuliert wurden. Abschließend können diese als Ausgangpunkte für zukünftige Arbeiten dienen, wobei auf einige innerhalb dieses Abschnittes nochmals eingegangen werden soll.
Für die Umsetzung des vorgeschlagenen Konzeptes wurde das UISDL-Format in einigen wesentlichen Punkten, insbesondere im Bezug auf das Thema der vorliegenden Arbeit aus funktioneller, aber auch informeller Sicht erneut untersucht. An dieser Stelle wurde außerdem auf mögliche Erweiterungen verwiesen, die auf eine Einteilung von UI Komponenten in Domain-spezifische und nicht Domain-spezifische UI Komponenten eingehen. Der Fokus wissenschaftlicher Nachfolgearbeiten sollte in diesem Bereich neben Untersuchungen zur Klassifikation von UI Komponenten auch die Erweiterung bestehender Beschreibungsformate auf Seiten funktionaler Dienste wie auch auf Seiten der UI Services zur Referenzierung gemeinsamer Klassifikationen sein.
Weiterhin wurde mit dem vorgeschlagenen Beschreibungsformat auf die Definition von Datentypen eingegangen, welche einen wesentlichen Teil der Betrachtungen bei der Umsetzung von Schema-Matching Verfahren ausmacht. Weitere wissenschaftliche Untersuchungen sollten das Thema des Schema-Matching mit Fokus auf dem aktuellen Stand der Forschung erneut aufgreifen, um Lösungsmöglichkeiten zur Optimierung der bereits angeführten Matching-Verfahren, mögliche Erweiterungen hinsichtlich Struktur-basierter Verfahren aber auch der Bewertung von Matching-Verfahren mit Bezug auf die Thematik zu identifizieren.
Bezogen auf den Discovery Prozess stellt sich die Frage, wie sich Beschreibungsformate zudem semantisch erweitern lassen, um den eigentlichen Prozess des Discovery in Hinblick auf die Ergebnismenge an potentiellen Kandidaten von UI Komponenten zu verbessern. Auf einige semantische Erweiterungen im Rahmen der Beschreibung funktionaler Dienste wurde innerhalb der Arbeit bereits verwiesen. Hier bedarf es weiterer Möglichkeiten und Ansätze im Bereich der Beschreibung von UI Services bzw. UI Komponenten, um so zentrale Lösungen für das bestehende Optimierungsproblem zu entwickeln.
Da der zentrale Fokus der Arbeit auf der Komposition von heterogenen, funktionalen Diensten und UI Services lag, wurde die direkte Komposition

von UI Komponenten an dieser Stelle vernachlässigt. Jedoch wurde bereits mit dem vorgeschlagenen Beschreibungsformat UISDL auf die erweiterte Beschreibung von Ereignissen verwiesen, die das zentrale Mittel zur Kommunikation zwischen UI Komponenten darstellen. An dieser Stelle kann die vorliegende Arbeit gerade bei der Betrachtung der Komposition als Grundlage dienen, um innerhalb weiterer wissenschaftlicher Arbeiten Möglichkeiten zur Komposition auf Ebene der Nutzeroberfläche zu (unter)suchen.

Anhang A

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
                xmlns="http://www.w3.org/2000/10/XMLSchema">
            <element name="TradePriceRequest">
                <complexType>
                    <all>
                        <element name="tickerSymbol" type="string"/>
                    </all>
                </complexType>
            </element>
            <element name="TradePrice">
                <complexType>
                    <all>
                        <element name="price" type="float"/>
                    </all>
                </complexType>
            </element>
        </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/
            soap/http"/>
        <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://example.com/GetLastTradePrice"/
                >
```

```
49            <input>
50                <soap:body use="literal"/>
51            </input>
52            <output>
53                <soap:body use="literal"/>
54            </output>
55        </operation>
56    </binding>
57
58    <service name="StockQuoteService">
59        <documentation>My first service</documentation>
60        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
61            <soap:address location="http://example.com/stockquote"/>
62        </port>
63    </service>
64
65 </definitions>
```

Listing A.1 Einfaches Beispiel einer WSDL-Beschreibung (aus [6])

```
1  <?xml version="1.0"?>
2  <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
4    xmlns:tns="urn:yahoo:yn"
5    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6    xmlns:yn="urn:yahoo:yn"
7    xmlns:ya="urn:yahoo:api"
8    xmlns="http://wadl.dev.java.net/2009/02">
9    <grammars>
10     <include href="NewsSearchResponse.xsd"/>
11     <include href="Error.xsd"/>
12   </grammars>
13   <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
14     <resource path="newsSearch">
15       <method name="GET" id="search">
16         <request>
17           <param name="appid" type="xsd:string" style="query" required="true"
                 />
18           <param name="query" type="xsd:string" style="query" required="true"
                 />
19           <param name="type" style="query" default="all">
20             <option value="all"/>
21             <option value="any"/>
22             <option value="phrase"/>
23           </param>
24           <param name="results" style="query" type="xsd:int" default="10"/>
25           <param name="start" style="query" type="xsd:int" default="1"/>
26           <param name="sort" style="query" default="rank">
27             <option value="rank"/>
28             <option value="date"/>
29           </param>
30           <param name="language" style="query" type="xsd:string"/>
31         </request>
32         <response status="200">
33           <representation mediaType="application/xml" element="yn:ResultSet"/
                 >
34         </response>
35         <response status="400">
36           <representation mediaType="application/xml" element="ya:Error"/>
```

```
37          </response>
38        </method>
39      </resource>
40    </resources>
41 </application>
```

Listing A.2 Einfaches Beispiel einer WADL-Beschreibung für die Yahoo! News Search Anwendung (aus [8])

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions
3    name="AWSECommerceProxyService"
4    xmlns="http://schemas.xmlsoap.org/wsdl/"
5    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
6    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7    xmlns:tns="http://www.example.org/AWSECommerceProxy/"
8    targetNamespace="http://www.example.org/AWSECommerceProxy/" >
9      <wsdl:types>
10     <xsd:schema targetNamespace="http://www.example.org/AWSECommerceProxy/"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.
           example.org/AWSECommerceProxy/" elementFormDefault="qualified" />
11     <xsd:complexType name="ItemSearchRequest">
12       <xsd:sequence>
13         <xsd:element name="Service" type="xsd:string" minOccurs="1" maxOccurs
               ="1"/>
14         <xsd:element name="Operation" type="xsd:string" minOccurs="1"
               maxOccurs="1"/>
15         <xsd:element name="Locale" type="xsd:string" minOccurs="1" maxOccurs=
               "1"/>
16         <xsd:element name="SearchIndex" type="xsd:string" minOccurs="1"
               maxOccurs="unbounded"/>
17         <xsd:element name="Keywords" type="xs:string" minOccurs="1" maxOccurs
               ="unbounded"/>
18       </xsd:sequence>
19     </xsd:complexType>
20
21     <xsd:complexType name="ItemSearchResponse">
22       <xsd:sequence>
23         <xsd:element name="Items" type="tns:Item" minOccurs="0" maxOccurs="
               unbounded"/>
24       </xsd:sequence>
25     </xsd:complexType>
26     <xsd:complexType name="Item">
27       <xsd:sequence>
28           <xsd:element name="ASIN" type="xsd:string" minOccurs="0" maxOccurs=
                 "1"/>
29           <xsd:element name="DetailPageURL" type="xsd:string" minOccurs="0"
                 maxOccurs="1"/>
30           <xsd:element name="Author" type="xsd:string" minOccurs="0"
                 maxOccurs="1"/>
31           <xsd:element name="Manufacturer" type="xsd:string" minOccurs="0"
                 maxOccurs="1"/>
32           <xsd:element name="Title" type="xsd:string" minOccurs="1" maxOccurs
                 ="1"/>
33         </xsd:sequence>
34     </xsd:complexType>
35     </wsdl:types>
36
37     <wsdl:message name="ItemSearchRequestMsg">
```

```
38      <wsdl:part name="body" element="tns:ItemSearchRequest"/>
39      </wsdl:message>
40      <wsdl:message name="ItemSearchResponseMsg">
41      <wsdl:part name="body" element="tns:ItemSearchResponse"/>
42      </wsdl:message>
43
44      <wsdl:portType name="AWSECommerceProxyServicePortType">
45      <wsdl:operation name="ItemSearch">
46        <wsdl:input message="tns:ItemSearchRequestMsg"/>
47        <wsdl:output message="tns:ItemSearchResponseMsg"/>
48      </wsdl:operation>
49      </wsdl:portType>
50
51      <wsdl:binding name="AWSECommerceProxyServiceBinding" type="
            tns:AWSECommerceProxyServicePortType">
52      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap
            /http"/>
53      <wsdl:operation name="ItemSearch">
54        <soap:operation soapAction="http://soap.amazon.com/ItemSearch"/>
55        <wsdl:input>
56          <soap:body use="literal"/>
57        </wsdl:input>
58        <wsdl:output>
59          <soap:body use="literal"/>
60        </wsdl:output>
61      </wsdl:operation>
62      </wsdl:binding>
63      <wsdl:service name="AWSECommerceProxyService">
64      <wsdl:port name="AWSECommerceProxyServicePort" binding="
            tns:AWSECommerceProxyServiceBinding">
65        <soap:address location="http://dev.mennerich.name/diplomApp/
              _WebServices/amazonWebService/amazonWS.php"/>
66      </wsdl:port>
67    </wsdl:service>
68  </wsdl:definitions>
```

Listing A.3 WSDL-Beschreibung des Proxy-Dienstes

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <uisdl
3    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4    xmlns="http://www.example.org/2009/UISDLSchema">
5
6    <types>
7        <xsd:schema targetNamespace="http://www.example.org/Universal/">
8        <xsd:complexType name="requestWeatherVars">
9          <xsd:sequence>
10           <xsd:element maxOccurs="1" minOccurs="1" name="city" type="
                 xsd:string" />
11           <xsd:element maxOccurs="1" minOccurs="1" name="postal_code" type="
                 xsd:string" />
12           <xsd:element maxOccurs="1" minOccurs="1" name="current_date" type="
                 xsd:string" />
13           <xsd:element maxOccurs="1" minOccurs="1" name="current_time" type="
                 xsd:string" />
14           <xsd:element maxOccurs="unbounded" minOccurs="0" name="weather_data
                 " type="Weather"/>
15         </xsd:sequence>
16       </xsd:complexType>
```

```
17      <xsd:complexType name="Weather">
18        <xsd:sequence>
19          <xsd:element maxOccurs="1" minOccurs="1" name="weather_day" type="
                xsd:string"/>
20          <xsd:element maxOccurs="1" minOccurs="1" name="weather_day_min"
                type="xsd:string"/>
21          <xsd:element maxOccurs="1" minOccurs="1" name="weather_day_max"
                type="xsd:string"/>
22          <xsd:element maxOccurs="1" minOccurs="1" name="weather_day_cond"
                type="xsd:string"/>
23          <xsd:element maxOccurs="1" minOccurs="1" name="weather_day_icon"
                type="xsd:string"/>
24        </xsd:sequence>
25      </xsd:complexType>
26
27      <xsd:complexType name="responseItemSearchVars">
28        <xsd:sequence>
29          <xsd:element name="Service" type="xsd:string" minOccurs="1"
                maxOccurs="1"/>
30          <xsd:element name="Operation" type="xsd:string" minOccurs="1"
                maxOccurs="1"/>
31          <xsd:element name="Locale" type="xsd:string" minOccurs="1"
                maxOccurs="1"/>
32          <xsd:element name="SearchIndex" type="xsd:string" minOccurs="1"
                maxOccurs="unbounded"/>
33          <xsd:element name="Keywords" type="xs:string" minOccurs="1"
                maxOccurs="unbounded"/>
34        </xsd:sequence>
35      </xsd:complexType>
36      </xsd:schema>
37  </types>
38
39  <uis>
40    <uic>
41      <meta>
42        <name id="UniversalUIC" />
43        <version id="0.0.2" />
44        <author name="Ronny␣Mennerich" email="ronny@mennerich.name" url="
              http://www.authorpage.com/uis_components" />
45        <description operation="weatherResult" lang="de">
46          Diese Operation zeigt Wetterdaten zu einer Stadt an.
47        </description>
48        <description operation="weatherResult" lang="en">
49          Component operation to present the weather data from a Web Service
                Request.
50        </description>
51        <description operation="itemSearchRequest" lang="de">
52          ItemSearch Anfrage an AmazonProxyService.
53        </description>
54        <description operation="itemSearchRequest" lang="en">
55          ItemSearch Request to AmazonProxyService.
56        </description>
57        <constructor name="UniversalUIC" />
58        <component href="http://dev.mennerich.name/diplomApp/_UIServices/
              universalUIService/0.0.2/component.html" />
59        <screenshot operation="weatherResult" href="http://dev.mennerich.name
              /diplomApp/_UIServices/universalUIService/0.0.2/
              weatherResultResponse.png" title="Weather␣Response" />
60        <screenshot operation="itemSearchRequest" href="http://dev.mennerich.
              name/diplomApp/_UIServices/universalUIService/0.0.2/
              itemSearchRequest.png" title="Amazon␣ItemSearch␣Request" />
61        <requirements>
62          <contentTechnologie name="xhtml" version="1.0" />
63          <scriptTechnologie name="javascript" version="" />
64          <layoutTechnologie name="css" version="" />
```

```
65          </requirements>
66        </meta>
67
68        <content>
69          <initial type="javascript" href="http://dev.mennerich.name/diplomApp/
                _UIServices/universalUIService/0.0.2/js/universalUIS.js" />
70          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/yahoo-dom-event/yahoo-dom-event.js" />
71          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/container/container-min.js" />
72          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/element/element-min.js" />
73          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/button/button-min.js" />
74          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/animation/animation-min.js" />
75          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/utilities/utilities.js" />
76          <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
                build/stylesheet/stylesheet-min.js" />
77          <source type="css" href="http://yui.yahooapis.com/2.7.0/build/
                container/assets/skins/sam/container.css" />
78          <source type="css" href="http://yui.yahooapis.com/2.7.0/build/fonts/
                fonts-min.css" />
79          <source type="css" href="http://yui.yahooapis.com/2.7.0/build/button/
                assets/skins/sam/button.css" />
80        </content>
81
82        <events>
83        </events>
84        <operations>
85          <operation name="weatherResult">
86            <request name="vars" type="tns:requestWeatherVars" />
87          </operation>
88          <operation name="itemSearchRequest">
89            <response name="vars" type="tns:responseItemSearchVars" />
90          </operation>
91        </operations>
92      </uic>
93    </uis>
94  </uisdl>
```

Listing A.4 UISDL-Beschreibung einer komplexen UI Komponente

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <uisdl
3    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4    xmlns="http://www.example.org/2009/UISDLSchema">
5
6    <types>
7        <xsd:schema targetNamespace="http://www.example.org/Amazon/">
8        <xsd:complexType name="ItemSearchRequest">
9          <xsd:sequence>
10           <xsd:element name="Items" type="tns:Item" minOccurs="0" maxOccurs="
                 unbounded"/>
11         </xsd:sequence>
12       </xsd:complexType>
13       <xsd:complexType name="Item">
14         <xsd:sequence>
```

```
15          <xsd:element name="DetailPageURL" type="xsd:string" minOccurs="0"
                maxOccurs="1"/>
16          <xsd:element name="ASIN" type="xsd:string" minOccurs="0" maxOccurs=
                "1"/>
17          <xsd:element name="Author" type="xsd:string" minOccurs="0"
                maxOccurs="1"/>
18          <xsd:element name="Title1" type="xsd:string" minOccurs="1"
                maxOccurs="1"/>
19          <xsd:element name="Manufacturer" type="xsd:string" minOccurs="0"
                maxOccurs="1"/>
20        </xsd:sequence>
21      </xsd:complexType>
22      </xsd:schema>
23  </types>
24
25  <uis>
26    <uic>
27      <meta>
28        <name id="AmazonUIC" />
29        <version id="0.0.1" />
30        <author name="Ronny␣Mennerich" email="ronny@mennerich.name" url="
              http://www.authorpage.com/uis_components" />
31        <description operation="itemSearchResponse" lang="de">
32          Ausgabe von Informationen einer ItemSearch Anfrage an
                AmazonProxyService.
33        </description>
34        <description operation="itemSearchResponse" lang="en">
35          Output of informations to an ItemSearch Request to
                AmazonProxyService.
36        </description>
37        <constructor name="AmazonUIC" />
38        <component href="http://dev.mennerich.name/diplomApp/_UIServices/
              amazonUIService/0.0.1/component.html" />
39        <screenshot operation="itemSearchResponse" href="http://dev.mennerich
              .name/diplomApp/_UIServices/amazonUIService/0.0.1/
              itemSearchResponse.png" title="Amazon␣ItemSearch␣Response" />
40        <requirements>
41          <contentTechnologie name="xhtml" version="1.0" />
42          <scriptTechnologie name="javascript" version="" />
43          <layoutTechnologie name="css" version="" />
44        </requirements>
45      </meta>
46
47      <content>
48        <initial type="javascript" href="http://dev.mennerich.name/diplomApp/
              _UIServices/amazonUIService/0.0.1/js/amazonUIS.js" />
49        <source type="javascript" href="http://yui.yahooapis.com/combo?2.7.0/
              build/yahoo-dom-event/yahoo-dom-event.js&2.7.0/build/dragdrop/
              dragdrop-min.js&2.7.0/build/element/element-min.js&2.7.0/build/
              datasource/datasource-min.js&2.7.0/build/datatable/datatable-min
              .js&2.7.0/build/container/container-min.js&2.7.0/build/
              stylesheet/stylesheet-min.js&2.7.0/build/animation/animation-min
              .js&2.7.0/build/button/button-min.js&2.7.0/build/calendar/
              calendar-min.js&2.7.0/build/utilities/utilities.js" />
50        <source type="css" href="http://yui.yahooapis.com/2.7.0/build/
              container/assets/skins/sam/container.css" />
51        <source type="css" href="http://yui.yahooapis.com/2.7.0/build/fonts/
              fonts-min.css" />
52        <source type="css" href="http://yui.yahooapis.com/2.7.0/build/button/
              assets/skins/sam/button.css" />
53        <source type="css" href="http://yui.yahooapis.com/2.7.0/build/
              datatable/assets/skins/sam/datatable.css" />
54      </content>
55
56      <events>
```

```
57        </events>
58        <operations>
59          <operation name="itemSearchResponse">
60            <request name="vars" type="xsd:ItemSearchRequest" />
61          </operation>
62        </operations>
63      </uic>
64    </uis>
65 </uisdl>
```

Listing A.5 UISDL-Beschreibung der Amazon UI Komponente

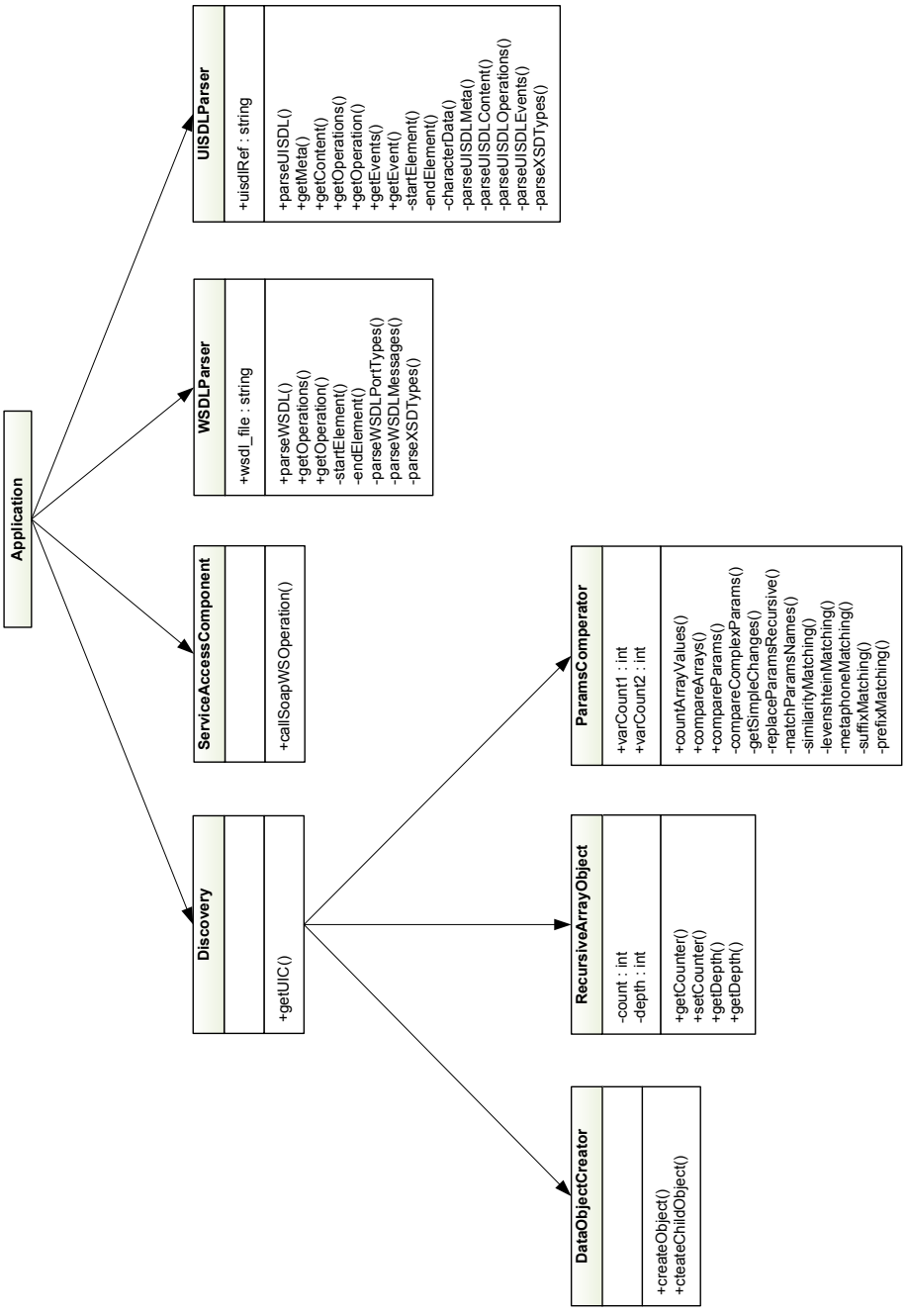


Abbildung A.1 Vollständiges Klassendiagramm der System-Anwendung

Anhang B

Glossar

API	Schnittstelle
APP	Atom Publishing Protocol
ASF	Atom Syndication Format
AWS	Amazon Web Services
B2B	Business-to-Business
B2C	Business-to-Consumer
CORBA	Common Object Request Broker Architecture
CRUISe	Composition of Rich User Interface Services
CSS	Cascading Style Sheets
DCOM	Distributed Component Object Model
HMAC	keyed-Hash Message Authentication Code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
MDL	mashArt Description Language
MEA	mashArt Event Annotation
PHP	PHP: Hypertext Preprocessor
REST	Representational State Transfer
RIA	Rich Internet Application
RMI	Remote Method Invocation
RSS	Rich Site Summary, RDF Site Summary, Really Simple Syndication
SAWSDL	Semantic Annotations for WSDL and XML Schema
SOA	Serviceorientierte Architektur
SOBA	Service-Oriented Business Application
SMTP	Simple Mail Transfer Protocol
UI	User Interface
UIC	User Interface Komponente, UI Komponente

UIS	User Interface Service, UI Service
UISDL	User Interface Description Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WS	Web Service
WSDL	Web Service Description Language
WWW	World Wide Web
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XPIL	Extensible Presentation Integration Language

bzw.	beziehungsweise
d. h.	das heißt
i.d.R.	in der Regel
u. a.	unter anderem
u. U.	unter Umständen
vgl.	vergleiche
z. B.	zum Beispiel

Quellcodeverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] Pietschmann, S.; Voigt, M.; Rümpel, A.; Meissner, K.: *CRUISe: Composition of Rich User Interface Services*, Springer, In: Proceedings of the 9th International Conference on Web Engineering (ICWE 2009), Edition 5648, S. 473-476, San Sebastian, Juni 2009

[2] Voigt, M.: *Entwicklung einer Service-Architektur für User Interfaces*, Belegarbeit, Technische Universität Dresden, 2008

[3] Deuse, S.: *Spezifikation einer Beschreibungssprache für komplexe User Interface-Bestandteile*, Belegarbeit, Technische Universität Dresden, 2008

[4] World Wide Web Consortium (W3C): *Web Services Architecture, W3C Working Group Note 11 February 2004*, W3C, http://www.w3.org/TR/ws-arch/, Abruf am 20.10.2009

[5] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation, 2000, Univertität von Californien (Irvine), http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm, Stand: 13.09.2000

[6] W3C: *Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001*, W3C, http://www.w3.org/TR/wsdl, Abruf am 20.03.2009

[7] W3C: *Web Services Description Language (WSDL) Version 2.0, W3C Recommendation 26 June 2007*, W3C, http://www.w3.org/TR/wsdl20/, Abruf am 22.03.2009

[8] World Wide Web Consortium (W3C): *Web Application Description Language, W3C Member Submission 31 August 2009*, W3C, http://www.w3.org/Submission/wadl/, Abruf am 20.10.2009

[9] Hadley, M. J.: *Web Application Description Language (WADL)*, https://wadl.dev.java.net/wadl20090202.pdf, Februar 2009

[10] NETSCAPE: *RSS 0.90 Specification*, http://www.rssboard.org/rss-0-9-0, Abruf am 16.10.2009

[11] RSS ADVISORY BOARD: *RSS 2.0 Specification*, http://www.rssboard.org/rss-specification, Abruf am 17.10.2009

[12] IETF: *The Atom Syndication Format*, http://tools.ietf.org/html/rfc4287, Abruf am 17.10.2009

[13] IETF: *The Atom Publishing Protocol*, http://tools.ietf.org/html/rfc5023, Abruf am 17.10.2009

[14] AMAZON, A9.COM: *The Atom Publishing Protocol*, http://www.opensearch.org/Specifications/OpenSearch/1.1, Abruf am 15.10.2009

[15] POLLERES, A.; LAUSEN, H.; LARA, R.: *Semantische Beschreibung von Web Services*, Springer, Buchkapitel aus: Semantic Web - Wege zur vernetzten Wissensgesellschaft, ISSN: 1439-3107, Juni 2006

[16] WORLD WIDE WEB CONSORTIUM (W3C): *Semantic Annotations for WSDL and XML Schema (SAWSDL), W3C Recommendation 28 August 2007*, W3C, http://www.w3.org/TR/sawsdl/, Abruf am 05.03.2009

[17] KOPECKY, J.; VITVAR, T.; BOURNEZ, C.; FARRELL, J.: *SAWSDL: Semantic Annotations for WSDL and XML Schema*, Internet Computing, IEEE, Volume 11, Issue 6, S. 60 - 67, Nov.-Dez. 2007

[18] VERMA, K.; SHETH, A.: *Semantically Annotating a Web Service*, Internet Computing, IEEE, Volume 11, Issue 2, March-April 2007, Page(s):83 - 85

[19] SONG, TING-XIN; TIAN, PEI-JUN; LIU, YAO-HE; HUANG, BI-QING: *Web Serviceś Semantic Annotation and Auto-matching Based on SAWSDL*, Internet Computing, IEEE, Volume: 2, S. 577-580, ISBN: 978-1-4244-2727-4, Dezember 2008

[20] LARVET, P.; CHRISTOPHE, B.; PASTOR, A.: *Semantization of Legacy Web Services: From WSDL to SAWSDL*, Internet and Web Applications and Services, 2008. ICIW apos; 08. Third International Conference on Volume , Issue , S. 130 - 135, Juni 2008

[21] W3C: *Semantic Annotations for WSDL and XML Schema - Usage Guide, W3C Working Group Note 28 August 2007*, W3C, http://www.w3.org/TR/sawsdl-guide/, Anruf am 27.03.2009

[22] World Wide Web Consortium (W3C): *Web Service Semantics - WSDL-S, W3C Member Submission 7 November 2005*, W3C, http://www.w3.org/Submission/WSDL-S/, Anruf am 11.03.2009

[23] J. D. Lathem: *SA-REST: Adding Semantics to REST-based Web Services*, University of Georgia, Master Thesis, 2007

[24] J. D. Lathem; K. Gomadam; A.P. Sheth: *SA-REST and (S)mashups : Adding Semantics to RESTful Services*, Semantic Computing, 2007. ICSC 2007. International Conference on Volume , Issue , 17-19 Sept. 2007 Page(s):469 - 476, Oktober 2007

[25] A.P. Sheth; K. Gomadam; J. D. Lathem: *SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups*, Internet Computing, IEEE, Volume 11, Issue 6, Nov.-Dec. 2007, Page(s): 91-94, November 2007

[26] R. Battle, E. Benson: *Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)*, ScienceDirect, November 2007

[27] J. Euzenat; P. Shvaiko: *Ontology Matching*, Springer, ISBN: 978-3-540-49611-3, Juni 2007

[28] J. Euzenat; P. Shvaiko: *A Survey of Schema-Based Matching Approaches*, In: Journal on Data Semantics IV, Nr. 3730, S. 146-171, 2005

[29] H. Paulheim: *Skalierbarkeit von Ontology-Matching-Verfahren*, Masterarbeit, Technische Universität Darmstadt, 2008

[30] F. Giunchiglia; P. Shvaiko: *Semantic matching*, In: The Knowledge Engineering Review, Bd. 18, S. 265-280, Cambridge University Press, 2003

[31] Yu, J.; Benatallah, B. , Casati, F.; Daniel, F.; Matera, M.; Saint-Paul, R.: *Mixup: A Development and Runtime Environment for Integration at the Presentation Layer*, In: Proceedings of the 7th International Conference on Web Engineering, S: 479-484, Juli 2007

[32] Yu, J.; Benatallah, B.; Saint-Paul, R.; Casati, F.; Daniel, F.; Matera, M.: *A Framework for Rapid Integration of Presentation Components*, In: Proceedings of the 16th International Conference on World Wide Web, S: 923-932, 2007

[33] Daniel, F.; Matera, M.: *Turning Web Applications into Mashup Components: Issues, Models, and Solutions*, In: Proceedings of the 9th International Conference on Web Engineering, S: 45-60, 2009

[34] Daniel, F.; Benatallah, B.; Shan, M.-C.: *Hosted Universal Composition: Models, Languages and Infrastructure in mashArt*, In: Proceedings of the 28th International Conference on Conceptual Modeling, 2009

[35] Tsai, W.-T.; Huang, Q.; Elston, J. Chen, Yinong: *Service-Oriented User Interface Modeling and Composition*, In: Proceedings of the 2008 IEEE International Conference on e-Business Engineering, S. 21-28, 2008

[36] IBM: *IBM Mashup Center*, http://www-01.ibm.com/software/info/mashup-center/, Anruf am 23.09.2009

[37] Corizon: *Corizon Plattform*, http://www.corizon.com/Products/corizon-enterprise-mashup-platform.html, Anruf am 20.08.2009

[38] Amazon: *Amazon - Product Advertising API, Developer Guide, Version 2009-10-01*, Amazon, http://docs.amazonwebservices.com/AWSECommerceService/2009-10-01/DG/ , Anruf am 21.10.2009