Assistierte Ad-hoc-Entwicklung von kompositen Webanwendungen durch Nicht-Programmierer

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht im April 2019 von
Dipl.-Medieninf. Carsten Radeck
geboren am 02.07.1985 in Meißen

Gutachter:
Prof. Dr.-Ing. Klaus Meißner (Technische Universität Dresden)
Prof. Dr.-Ing. Jürgen Ziegler (Universität Duisburg-Essen)

Tag der Verteidigung: 04.11.2019

Dresden im Februar 2020

Danksagung

Diese Arbeit ist im Verlauf von vielen Jahren Forschung entstanden und sie wäre nicht möglich gewesen ohne die Unterstützung einer Vielzahl an Personen, denen ich an dieser Stelle meinen herzlichen Dank ausdrücken möchte.

Zuallererst richtet sich mein besonderer Dank an Prof. Dr.-Ing. Klaus Meißner für die Betreuung der vorliegenden Dissertation und für die Chance in einem spannenden Forschungsgebiet zu arbeiten. Der gewährte Gestaltungsspielraum sowie die konstruktiven inhaltlichen Diskussionen, Kommentare und Anregungen haben entscheidend zur Entwicklung und Vollendung der Ergebnisse beigetragen. Mein herzlicher Dank gilt weiterhin Prof. Dr.-Ing. Jürgen Ziegler für die Bereitschaft, diese Arbeit zu begutachten.

Allen Kollegen möchte ich für die zurückliegenden Jahre am Lehrstuhl danken. Das angenehme, kollegiale Arbeitsklima weiß ich zu schätzen. Mein Dank gebührt Stefan Pietschmann, der mich als studentische Hilfskraft, als Beleg- und Diplomstudent an das Themengebiet und an das wissenschaftliche Arbeiten herangeführt hat. Besonderer Dank gilt natürlich denjenigen Kollegen, die mit ihren wertvollen Anmerkungen und inhaltlichen Diskussionen aktiv zur Reifung der Arbeit beigetragen haben. Hervorheben möchte ich an dieser Stelle Gregor Blichmann. Ebenso wichtig wie die fachliche Unterstützung ist Hilfe auf organisatorischer Ebene, für die ich mich bei Ramona Behling und Jana Bohl bedanken möchte.

Zahlreiche Studenten haben im Rahmen ihrer Beleg-, Diplom-, Bachelor- sowie Masterarbeiten, in Praktika und als studentische Hilfskräfte unverzichtbare Beiträge zur vorliegenden Arbeit geleistet. Stellvertretend genannt seien Ronny Kursawe, Clemens Große, André Lorenz und Johannes Pflugmacher.

Ganz herzlich bedanke ich mich bei meiner Familie und meinen Freunden für den bedingungslosen Rückhalt und die Unterstützung in jeglicher Beziehung während des Studiums und der Promotion. Auch das Mittragen so mancher Last und der moralische Beistand sind von unschätzbarem Wert für den Entstehungsprozess der Dissertation. Insbesondere sei Marc Mosch für das Korrekturlesen sowie die zahlreichen inhaltlichen Anregungen gedankt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis von Quellcodebeispielen

Abkürzungsverzeichnis

API	Application Programming Interface
CEP	Complex Event Processing
CRUISE	Composition of Rich User Interface Services for Everybody
CSR	Client-Server Runtime
CSS	Cascading Style Sheet
CWA	Composite Web Application
DOM	Document Object Model
EUD	End-User Development
IWC	Inter-Widget Communication
JAXB	Java Architecture for XML Binding
JSON	JavaScript Object Notation
MCM	Mashup Composition Model
MRE	Mashup Runtime Environment
OWL	Web Ontology Language
POI	Point of Interest
QUDT	Quantities, Units, Dimensions and Data Types Ontologies
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
SAWSDL	Semantic Annotations for WSDL
SMCDL	Semantic Mashup Component Description Language
SOA	Service-oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SUS	System Usability Scale
SWS	Semantic Web Services
TLX	Task Load Index
TSR	Thin-Server Runtime

UI	User Interface
URI	Uniform Resource Identifier
WSDL	Web Service Description Language
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Einleitung

Die Zahl der über das Web verfügbaren Ressourcen und Dienstleistungen steigt ständig an. Im Zuge zunehmender Dienstorientierung der IT werden diese Ressourcen verstärkt entsprechend dem Paradigma des *Programmable Web* [MRT07] komponenten- oder serviceorientiert bereitgestellt, zum Beispiel in Form von Webservices auf Basis von SOAP oder gemäß REST, als JavaScript-API und als RSS-Feeds. Folglich stellt das Web für vielfältige Anwendungen eine moderne, geeignete Plattform dar, in deren Zentrum solche Web-APIs stehen, und es wird als derzeit größte und am weitesten akzeptierte Plattform für verteilte Systeme angesehen [LDB15]. Dieser Wandel der klassischen serviceorientierten Architektur (SOA) zu einem Ökosystem für Web-APIs zeigt sich auch an den Interessen im Forschungsgebiet Service-Computing: Während im Jahr 2005 noch Begriffe wie »Grid«, »Semantik« und »BPEL4WS« prägend waren, spielten 2015 beispielsweise »Empfehlung« und »Mashup« eine wichtige Rolle [Tan+16].

Die Komponenten- und Dienstorientierung aufgreifend stellt das Modell der *Mashups* eine leichtgewichtige Umsetzung einer SOA dar. Vorhandene Bausteine, das heißt Services und Komponenten, werden hierbei miteinander komponiert, um einen funktionalen Mehrwert zu erzielen [@Mer; FBN08]. Verglichen mit traditioneller Softwareentwicklung steht eine möglichst zeit- und kostengünstige Erstellung im Mittelpunkt, sodass Mashups tendenziell kurzlebige Anwendungen sind und vor allem situative Probleme lösen sollen. Neben der ursprünglichen Integration von Bausteinen der Daten- und Anwendungslogikebene wie in Yahoo! Pipes [Pru07], existieren neuere Ansätze, welche das Prinzip auch auf die Benutzerschnittstelle übertragen, wie CRUISe [Pie12] und OMELETTE [Chu+12]. Dies verspricht den, aufgrund verschiedenster Nutzer- und Gerätekontexte, hohen Aufwand der Entwicklung von Benutzerschnittstellen zu verringern. Um Komponenten aller Anwendungsebenen einheitlich modellieren, integrieren und behandeln zu können, haben sich Ansätze zur *universellen Komposition* [Dan+09] herausgebildet, zum Beispiel CRUISe [Pie12] und PEUDOM [Mat+13]. Die Basis hierfür sind ein klar definiertes Komponentenmodell, welches einzelne Bausteine beschreibt, sowie ein Kompositionsmodell, das anwendungsspezifische Aspekte wie die Kommunikation zwischen Komponenten und das Layout eines resultierenden Mashups umfasst. Im Rahmen dieser Arbeit werden einzig solche komponentenbasierten Mashups gemäß universeller Komposition betrachtet und nachfolgend als komposite Webanwendungen, englisch *Composite Web Applications (CWAs)*, bezeichnet.

Als weiterer Trend ist die im Sinne des Web 2.0 zunehmende Einbeziehung von Endnutzern und Domänenexperten in den Entwicklungsprozess von Anwendungen zu beobachten, was als End-User Development (EUD) bezeichnet wird. EUD umfasst dabei eine Menge von Methoden, Techniken und Werkzeugen, die es nicht-professionellen Softwareentwicklern erlauben, Softwareartefakte zu erstellen, zu verändern oder zu erweitern [Lie+06]. Endnutzer werden also in die Lage versetzt, in einem bestimmten Umfang Software selbst zu entwickeln. Prinzipiell ist EUD ökonomisch äußerst lukrativ, da die Entwicklung effizienterer und innovativer Arbeitspraktiken stimuliert wird [WJ04] und sehr spezifische Nutzeranforderungen, der sogenannte *Long Tail of User Needs*, erfüllt werden können [Jan+09]. Diese Aspekte entfalten insbesondere im Unternehmensumfeld ihre vorteilhafte Wirkung. Die klassische Entwicklung von Applikationen durch die IT-Abteilung ist kosten- und zeitintensiv, sodass sie spezifischen, situationsbezogenen Bedürfnissen kaum gerecht werden kann, da diese mit hohen Investitionskosten verbunden sind. Dies wird verschärft durch die *Impossible Equation* [Bez09], welche aussagt, dass eine langsam steigende Zahl von Entwicklern einem stark anwachsenden Bedarf an Individuallösungen gegenübersteht. An diesen Stellen verspricht EUD Potential, birgt aber zahlreiche grundlegende Herausforderungen, nicht zuletzt deshalb, weil es sich um ein hochgradig interdisziplinäres Forschungsgebiet handelt.

Die Prinzipien von CWA fördern EUD und stellen eine gut geeignete konzeptionelle Basis für entsprechende Entwicklungswerkzeuge dar. Dies liegt zunächst an dem Kompositionsansatz, der vorsieht, vorhandene Bausteine wiederzuverwenden. Die Wiederverwendbarkeit und Verknüpfung von Komponenten, auch über Anwendungsebenen hinweg, erlaubt eine Reduzierung des Entwicklungsaufwandes durch Senkung von benötigter Zeit und Kosten. Eine dadurch zugängliche Ad-hoc-Erstellung passender Softwarelösungen für Nischenanforderungen, vergleiche *Long Tail of User Needs*, versetzt Endnutzer potentiell in die Lage, entsprechend ihrer situationsbezogenen Anforderungen, ein Mashup zur Lösung eines konkreten Problems zu komponieren – und dies idealerweise ohne Programmierkenntnisse vorauszusetzen.

Wird als Endnutzer ein technischer Laie – im Folgenden *Nicht-Programmierer* – adressiert, ergeben sich vielfältige Herausforderungen für eine Kompositionsplattform. Im Kontext dieser Arbeit sind Nicht-Programmierer Personen, die regelmäßig Computer und Webanwendungen nutzen und somit mit Grundfunktionen vertraut sind. Sie besitzen jedoch keine Kenntnis von Programmiersprachen und -konzepten, kennen sich allerdings gut in der fachlichen Domäne des zu lösenden Problems aus. Zunächst existiert ein substantieller Unterschied im Vorgehen von Nicht-Programmierern und professionellen Softwareentwicklern. Beim EUD findet typischerweise keine strikte Trennung von Entwurf und Implementierung statt [Cao+10]. Demnach durchziehen implizit winzige Entwurfsentscheidungen die »Programmierung«. Nicht-Programmierer durchlaufen iterativ folgende drei Phasen in unterschiedlichen Abfolgemustern [Cao+10]: *Framing* (Identifikation und Verstehen des Problems), *Acting* (Handeln mit dem Ziel die aktuelle Situation in eine »bessere« zu überführen) und *Reflecting* (Rückblick auf die Handlung, um deren Auswirkungen zu verstehen). Aus diesen und weiteren Eigenheiten der Zielgruppe und deren Vorgehen ergeben sich unter anderem folgende Herausforderungen für eine Kompositionsplattform von Webservices, vergleiche [Meh+10b; KPW06; NWM10; AP13; RO14; Cao13], die grundsätzlich auf CWA übertragbar sind:

- Es gilt, den Prozess der Anwendungsentwicklung in handhabbare Teilschritte zu zerlegen und Nutzer durch adäquate Hilfsmittel bei deren Abarbeitung zu unterstützen.

Hierzu gehören Herausforderungen wie geeignete Interaktions- und Visualisierungsmetaphern, Aufgabenzentrierung und Abstraktion technischer Details.

- Qualitätssicherung spielt aufgrund des Wegfalls einer Testabteilung und der fehlenden Versiertheit der Nutzer eine wichtige Rolle.
- Es wurde gezeigt, dass Endnutzer die Trennung von Entwicklung und Nutzung einer Anwendung schwer nachvollziehen können [NND10a].
- Unterstützung durch Assistenzsysteme und Empfehlungen ist nötig unter anderem bei der Formulierung von Anforderungen, beim Komponieren sowie beim Inspizieren einer CWA, zum Beispiel damit der Nutzer neue Ideen generiert.
- Es bedarf der Automatisierung von Teilschritten des Kompositionsprozesses, zum Beispiel zur Herstellung von Interoperabilität zu verknüpfender Komponenten.

Insbesondere die beiden letztgenannten Punkte sind von zentraler Bedeutung, um Entwurfs- und Problemlösungsbarrieren [Cao13] bei der Entwicklung von CWA zu überwinden, etwa durch Empfehlung von Komponenten und deren automatisierte korrekte Verknüpfung mit der bestehenden Anwendung während der Phasen *Framing* oder *Acting*. Dies gewinnt an Bedeutung angesichts einer zunehmenden Anzahl von Komponenten und Anwendungen, die zudem bestmöglich zur aktuellen Aufgabe und zum Kontext des Nutzers passen sollen und von unterschiedlichen Anbietern stammen können.

Insgesamt lässt sich festhalten, dass CWA prinzipiell das Potential besitzen, Nicht-Programmierer in die Rolle des Anwendungsentwicklers zu versetzen. Die Bereitschaft von Endnutzern, in Entwicklungsarbeiten Zeit und Aufwand zu investieren, steigt mit den wahrgenommenen Nutzen und Vorteilen, wie einer spürbaren Effizienzverbesserung [Sut05]. Sie sinkt durch die Komplexität eines Werkzeugs und durch hohen initialen Lernaufwand [Sut05]. Daher spielt eine adäquate Führung und Unterstützung beim Kompositionsprozess eine zentrale Rolle für eine Plattform zur Komposition von CWA. Bisherige Ansätze in der Mashup- und Webservice-Domäne liefern jedoch nicht für alle der genannten Herausforderungen zufriedenstellende Lösungen, was im nachfolgenden Kapitel vertiefend behandelt wird. Grundlegend stellen sich daher unter anderem folgende **Forschungsfragen**: Welche Verfahren, Modelle und Architekturen werden benötigt, um Nicht-Programmierer bei der Abbildung eines fachlichen Problems auf eine technische Lösung in Form eines Mashups zu unterstützen? Sind Empfehlungen und automatisierte Kompositionsschritte während der gesamten Ad-hoc-Anwendungsentwicklung dafür geeignet? Wie sehen geeignete Algorithmen aus, um diese Hilfestellungen kontextsensitiv und unter Ausnutzung des Wissens aus der Community und der semantischen Komponentenbeschreibungen abzuleiten? Können Empfehlungen in Bezug zur aktuellen Aufgabe des Nutzers ermittelt und dem Endnutzer auf diesem Abstraktionsniveau präsentiert werden? Für diese und weitere Fragestellungen existieren bisher nur unzureichende Antworten. Daraus ergeben sich wesentliche Probleme, Forschungsziele und Thesen, die Gegenstand der nachfolgenden Betrachtungen sind.

1.1 Analyse von Herausforderungen und Problemen

Wie in der kurzen Motivation dargelegt wurde, ergänzen sich Mashup-Paradigmen und EUD prinzipiell sehr gut. Zwar wurde von Beginn an postuliert, dass Mashup-Kompositions-

plattformen auch für Nicht-Programmierer geeignet seien. Für aktuelle Ansätze wird dies jedoch in diesem Abschnitt widerlegt. Es sind daher noch wesentliche Forschungsfragen zu beantworten und adäquate Lösungen zu konzipieren.

Um die fundamentalen Herausforderungen und Probleme, die sich beim EUD von CWA durch Nicht-Programmierer ergeben, zu identifizieren, ist es unabdingbar, zunächst zu klären, von welchen »Endnutzern« im Rahmen dieser Arbeit ausgegangen wird. Dazu wird nachfolgend der bereits mehrfach genannte Personenkreis der *Nicht-Programmierer* genauer charakterisiert.

1.1.1 Zielgruppendefinition

Zunächst ist es erforderlich, den Begriff des Endnutzers genauer einzugrenzen, damit anhand von dessen Charakteristiken zu lösende Probleme und schließlich Anforderungen an eine Konzeption zielgerichtet hergeleitet werden können. Dazu werden mehrere Dimensionen definiert, die zur Charakterisierung und Klassifizierung von Nutzergruppen dienen, siehe Abbildung 1.1. Aufbauend auf der Einteilung von Silva et al. [SPS10], die Laien, Domänenexperten, technische Experten und Fortgeschrittene anhand der Dimensionen *Domänenwissen* und *Technikwissen* unterscheiden, werden für diese Arbeit folgende Nutzereigenschaften zugrunde gelegt.

- Motivation selbst als »Entwickler« in Erscheinung zu treten
- Expertise in der Entwicklung von Softwareanwendungen, speziell Mashups
- Expertise in der Domäne, in der das zu lösende Problem angesiedelt ist
- Vertrautheit im Umgang mit üblichen Webanwendungen, beispielsweise mit Suchmaschinen, Karten- und Kalenderanwendungen

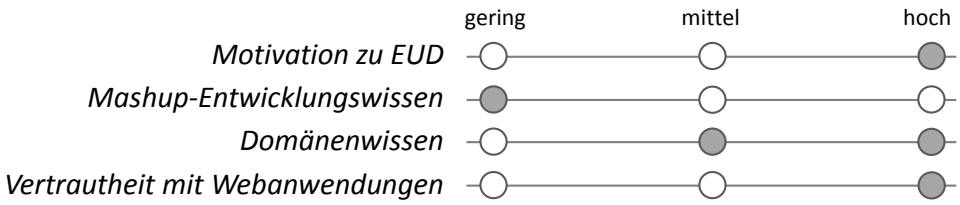


Abbildung 1.1: Dimensionen zur Einteilung von Nutzern und relevante Ausprägungen zur Charakterisierung der Zielgruppe Nicht-Programmierer

Der relevante Ausschnitt aus dem aufgespannten mehrdimensionalen Raum ist in Abbildung 1.1 durch ausgefüllte Punkte pro Dimension angedeutet. Die markierten Bereiche veranschaulichen die tendenzielle Ausprägung der Zielgruppe für das jeweilige Merkmal. Dies führt zu folgender Definition von Endnutzern, die für diese Dissertation im Mittelpunkt stehen: Die Zielgruppe von Endnutzern sind Domänenexperten ohne tiefgreifende Programmierkenntnisse, die im Umgang mit Webanwendungen geübt und daran interessiert sind, sich selbstständig maßgeschneiderte Anwendungen zu erstellen. Solche Nutzer werden im weiteren Verlauf der Arbeit als *Nicht-Programmierer* bezeichnet und lassen sich unter anderem wie folgt charakterisieren:

- Ihnen fehlt es an Verständnis für informationstechnische Konzepte und Begriffe sowie an Erfahrung und Wissen hinsichtlich Softwareentwurfspraktiken.
- Sie denken in zu lösenden Aufgaben ihrer Domäne und kennen typischerweise auch eine Lösung in den fachlichen Begriffen und Prozessen. Sie scheitern jedoch bei deren Übertragung auf ein Mashup oder trauen sich diese nicht zu [Liz+16].

Diese Charakteristiken führen zu einer Reihe von Problemen, denen ein Nicht-Programmierer beim Entwerfen, Anpassen und Nutzen einer CWA gegenüberstehen kann. Sie stellen daher Mashup-Plattformen vor große Herausforderungen, welche in den nächsten Abschnitten dargestellt werden.

1.1.2 Problemanalyse

In den vorherigen Abschnitten wurde motiviert, dass EUD für Nutzer prinzipiell eine Reihe von Vorteilen verspricht und dass CWA aufgrund des Komponentenansatzes eine geeignete konzeptionelle Grundlage dafür bieten. Die Bereitschaft von Nutzern, aktiv als Entwickler in Erscheinung zu treten, wird vom wahrgenommenen Nutzen positiv und vom Einarbeitungsaufwand sowie von der Komplexität des Werkzeugs negativ beeinflusst [Sut05]. Daher und vor dem Hintergrund von Nicht-Programmierern als Zielgruppe ist EUD ein Kompromiss zwischen Ausdrucksstärke und Freiheitsgrad auf der einen sowie Nutzbarkeit, Erlernbarkeit und Kontrolle auf der anderen Seite. Das Ziel Nicht-Programmierer in die Lage zu versetzen, eigenständig CWA zu entwickeln, anzupassen und zu verstehen, wirft verschiedenste Forschungsfragen auf, die substantiellen Forschungsbedarf illustrieren. Bisherige Ansätze im Bereich CWA und EUD bieten jedoch nicht für alle Fragen adäquate Antworten. In der Essenz dieses Abschnitts werden auftretende, im Fokus der Arbeit stehende Problemfelder diskutiert.

Als **Kernproblem** ist zu nennen, dass die bedarfsgerechte Ad-hoc-Entwicklung und Änderung von Webanwendungen komplex und zeitintensiv, und daher kaum durch Nicht-Programmierer zu bewältigen ist. Im Gegensatz zur klassischen Softwareentwicklung, obliegt es beim EUD von Mashups den Nutzern selbst, sämtliche Entwicklungsphasen vom Entwurf bis zum Test selbstständig zu bestreiten. Angesichts der Zielgruppendefinition stellt dies eine substantielle Herausforderung dar und bedarf passender Werkzeuge. Zwar bieten modellgetriebene Ansätze die Möglichkeit, ausgehend von fachlich abstrahierten Modellen softwaretechnologische Lösungen zu erstellen. Jedoch sind derartige Ansätze für das EUD durch Nicht-Programmierer ungeeignet. Denn sie sehen typischerweise eine separate Entwicklung (Modellierung, Transformationen) und Nutzung der Anwendung vor. Wie zuvor erörtert, ist eine derartige Trennung für Nicht-Programmierer schwierig zu verstehen. Bei der Abbildung des fachlichen Problems auf eine technische Lösung in Form eines Mashups gehen Nicht-Programmierer typischerweise iterativ vor. Bisher bieten Mashup-Plattformen keine durchgehende automatisierte Unterstützung von Nicht-Programmierern beim Kompositionsprozess, sodass der Abbildungsvorgang erschwert wird. Es treten bei der Entwicklung und der Nutzung von CWA durch Nicht-Programmierer Barrieren auf, die nachfolgend detailliert werden.

Für Nicht-Programmierer gestaltet sich die Suche nach passenden *Kompositionsfragmenten*[1] schwierig, unterstellt, dass deren Menge ständig steigt und funktionale

[1] Ein Kompositionsfragment bezeichne im Verlauf der Arbeit einen funktionstüchtigen Teilausschnitt einer CWA, d. h. eine einzelne Komponente bis hin zu einem kompletten Mashup.

Substitute enthält. Umfangreiche Forschung im Bereich der (explorativen) Informationssuche führte zu theoretischen Modellen, welche den Prozess der Informationssuche aus verschiedenen Blickwinkeln beschreiben, siehe [Hea09]. Diese wird dabei als eine Art der Problemlösung aufgefasst [Mar95] und umfasst mehrere zyklische Aktivitäten [SE98]. Wie [Kuh91] belegt, kennzeichnen oft Unsicherheit und Zweifel den Beginn der Suche. [Bat90; BMC93] beleuchten Suchprozesse aus strategischer Sicht und zeigen, dass gerade bei komplexem und unscharfem Informationsbedürfnis Strategiewechsel notwendig sind. Jedoch fehlt es an Werkzeugunterstützung und Nicht-Programmierern mangelt es typischerweise an Problemlösestrategien [Cao13]. Weiterhin fällt es Nicht-Programmierern schwer, ihre Anforderungen klar zu formulieren und auf die Suchparameter von Komponentenbeschreibungen oder genutzte Anfragesprachen abzubilden.

- Häufig kennen Nicht-Programmierer zwar eine fachliche Lösung für ihr Problem, können sie jedoch nicht in die »Systemsprache« übersetzen. Dies kann bspw. darin begründet sein, dass dem Nutzer nicht die passenden oder nur synonyme Begriffe bekannt sind und dass die strukturellen Eigenheiten einer »korrekten« Anfrage technisch geprägt oder ihm unbekannt sind. Auch können die resultierenden Anforderungen an ein Mashup oft nur vage spezifiziert werden oder sind unvollständig.

- Nutzer dekomponieren ihre zu lösende Aufgabe während der Entwicklung der Mashup-Anwendung oder gewinnen erst dabei ein detaillierte Vorstellung ihrer Aufgabe. Fehlt dem Nutzer eine umfassende Kenntnis über die Domäne, etwa zu alternativen oder von seinem üblichen Arbeitsfluss abweichenden Begriffen und Möglichkeiten, wird die Dekomposition und somit die Zielschärfung erschwert.

- Es mangelt aktuellen Mashup-Ansätzen häufig an der Möglichkeit, durch Feedback-Strategien die Kriterien zu ändern, und somit iterativ zu genaueren Ergebnissen zu gelangen. Zudem ist oftmals der Bezug von Suchergebnissen zur Anfrage, zum Beispiel der Erfüllungsgrad von Kriterien, nicht erkennbar.

Die semantisch korrekte Verknüpfung von Komponenten ist nicht trivial. Da Komponenten typischerweise von verschiedensten Anbietern stammen, können Inkompatibilitäten von Schnittstellen auftreten und infolgedessen die Interoperabilität einschränken. Beispiele hierfür sind verschiedene Maßeinheiten oder Granularitätsunterschiede der verwendeten Typisierung. In Mashup-Plattformen basiert die Verknüpfbarkeit von Komponenten zumeist auf Identität von syntaktisch definierten Datentypen, was das grundlegende Problem der Mehrdeutigkeit mit sich bringt. Im Bereich Semantic Web Services (SWS) entwickelte Konzepte erlauben es, die Komponierbarkeit auf die Ebene der Semantik (Bedeutung) der Datentypen zu abstrahieren. Diese Ansätze wurden auf die Komposition von Mashups übertragen [PRM11b], jedoch sind sie in ihrer Mächtigkeit limitiert und die Eignung zur Einbindung in das EUD durch Nicht-Programmierer ist nicht untersucht. Existierende Mashup-Plattformen geben somit nur unbefriedigende Hilfestellung zur automatisierten Überbrückung auftretender Heterogenität von Komponentenschnittstellen.

Empfehlungssysteme bieten zwar die Möglichkeit, unterstützende Vorschläge zu Kompositionsschritten anzubieten, besitzen im Hinblick auf die Zielgruppe der Nicht-Programmierer jedoch Defizite:

- Rein community-basierte Empfehlungssysteme zeigen Kaltstartprobleme und tendieren dazu, ohnehin beliebte Komponenten zu bevorzugen. Dies ist mehrfach

nachteilig, unter anderem hinsichtlich der Erfüllung von Nischenanforderungen und der Passgenauigkeit zum Nutzerkontext, da im *kollektiven Wissen* nicht zwangsläufig eine passende Lösung für sehr spezielle Probleme vorliegt. Semantikbasierte Empfehlungssysteme bedingen andererseits entsprechend ausdrucksstarke Komponentenbeschreibungen, was in erster Linie den Komponentenentwickler fordert. Diese Ansätze liefern zwar schnittstellenkompatible Komponenten, können jedoch in der Regel keine Garantien für ein reibungsloses funktionales Zusammenspiel der Komponenten geben. Dies ist insofern problematisch als im stark verkürzten Entwicklungsprozess von CWA keine funktionalen Tests im Sinne klassischer Softwareentwicklung vorgesehen sind. Hybride Lösungen kombinieren community- und semantikbasierte Konzepte, sind im Bereich der Ad-hoc-Entwicklung von CWA jedoch noch nicht hinreichend erforscht.

- Mashup-Plattformen werden mit einer zunehmenden Heterogenität in Nutzer-, Nutzungs- und Gerätekontexten konfrontiert. Kontextsensitivität von Vorschlägen spielt daher eine zentrale Rolle. Dem wird derzeit jedoch wenig Rechnung getragen. Insbesondere das Wissen über den aktuellen Aufgabenkontext kann nicht ausgeschöpft werden. Die aktuelle Aufgabe des Nicht-Programmierers und der Beitrag von Empfehlungen zur Erbringung dieser Aufgabe werden in existierenden Ansätzen nicht ermittelt. Folglich berücksichtigt die Visualisierung von Empfehlungen deren Beitrag zur Aufgabe und zur Funktionalität nicht, sondern ist technisch geprägt und erfolgt mit Mitteln des Kompositionsmodells. Eine solche Darstellung von Vorschlägen ist für Nicht-Programmierer nur schwer interpretierbar und die Nützlichkeit der Vorschläge durch sie kaum einschätzbar. Da die fokussierte Zielgruppe vorrangig in fachlichen Begriffen und Aufgaben denkt und idealerweise die Anforderungen und Ziele derart in das System eingegeben werden, muss das Empfehlungssystem in der Lage sein, die Funktionalität von Kompositionsfragmenten abzuschätzen und gegen die Nutzeranforderungen abzugleichen. Während die Funktionalitäten von Komponenten statisch durch Komponentenentwickler definiert werden können, mangelt es bisher an Ansätzen zur Bestimmung der Funktionalität beliebiger Kompositionsfragmente.

- Die Spezifik einer Verschmelzung von Entwicklung und Nutzung von Mashups wird durch bisherige Empfehlungsansätze in Kompositionsplattformen nicht adäquat berücksichtigt, da Empfehlungen nur in dedizierten Phasen, zum Beispiel bei der Suche nach Komponenten, oder nur unter bestimmten Umständen angeboten werden. In aktuellen Empfehlungsansätzen für Webservices und Mashups, wie [BDM10; Roy+10; CGT11], sind diese Auslöser jedoch starr vorgegeben, nicht erweiterbar und zumeist an das explizite Anfordern durch den Endnutzer gebunden. Zudem wird selten der gesamte Kompositionsprozess abgedeckt, sodass es an einer durchgängigen Unterstützung für Nicht-Programmierer mangelt. Das Empfehlungswesen ist typischerweise fest in die Mashup-Plattform integriert, sodass eine bedarfsgerechte Konfiguration, etwa in Abhängigkeit von der Nutzergruppe oder der Anwendungsdomäne, kaum möglich ist.

Das Verstehen und Testen der Funktionsweise dem Nutzer bisher unbekannter CWA kann Nicht-Programmierer vor zusätzliche Hürden stellen, was sich in erhöhtem Einarbeitungsaufwand und häufig erfolglosem Ausprobieren äußert. Der Kern der Funktionalitäten

einer CWA ergibt sich typischerweise aus dem Zusammenspiel der einzelnen Komponenten. Das nachzuvollziehen ist nicht trivial und fehlende Unterstützung durch die Mashup-Plattform kann in falschen Erwartungen und Fehlinterpretationen münden.

- Es ist nicht unmittelbar erkennbar, welche Fähigkeiten und Funktionalitäten einzelne Komponenten besitzen beziehungsweise erbringen.
- Nicht-Programmierern fehlt typischerweise das Bewusstsein des kompositen Charakters von CWA. Das zieht nach sich, dass insbesondere Aspekte der Kommunikation zwischen den Komponenten nicht verinnerlicht werden [Chu+13]. Es konnte gezeigt werden, dass in die Live-View integrierte optische Indikatoren den Nutzer bei dem Nachvollziehen von Kommunikationsbeziehungen unterstützen können. Geeignete Visualisierungstechniken seien jedoch noch zu erforschen [Tsc+14]. Zudem fehlt es an Konzepten zur funktionalen Beschreibung von Kommunikationskanälen. Komplexere Verknüpfungen verschärfen das Problem, zum Beispiel wenn eine Komponente aus mehreren anderen Komponenten mit Eingaben versorgt wird, wenn ein Ergebnis an verschiedenen Stellen in der CWA dargestellt wird oder wenn gerichtete Verbindungen auftreten.
- Komposite Webanwendungen können Komponenten ohne dediziertes User Interface (UI) beinhalten. Wie gezeigt werden konnte, ist dieses Konzept für Nicht-Programmierer nur schwer nachvollziehbar [RBM13; Pfl15].
- Nutzer lassen sich stark von ihren Annahmen und Erfahrungen im Umgang mit bekannten Webanwendungen leiten, beispielsweise hinsichtlich des Funktionsumfangs von Komponenten. Auch die Anordnung von UI-Komponenten kann die von Nutzern angenommenen funktionalen Zusammenhänge der Komponenten beeinflussen [RBM13; Pfl15]. Zudem kann das Auftreten mehrerer Instanzen einer Komponente in einer CWA zu Verwirrung führen.

Wie deutlich wurde, stehen Nicht-Programmierer vor einer Vielzahl von Herausforderungen bei der Benutzung und Entwicklung von CWA. Nachfolgend wird daher diskutiert, welche Thesen und Forschungsziele im Rahmen dieser Arbeit verfolgt werden, um diesen Herausforderungen zu begegnen.

1.2 Thesen, Ziele, Abgrenzung

Basierend auf den vorgestellten Herausforderungen und Problemen steht in den nachfolgenden Abschnitten im Mittelpunkt, welche Thesen der Arbeit zugrunde liegen und welche Forschungsziele verfolgt werden. Weiterhin wird geklärt, auf welchen Annahmen die Arbeit beruht und welche Aspekte hier nicht Forschungsgegenstand sind.

1.2.1 Forschungsthesen

Ausgehend von den zuvor identifizierten Problemen und Defiziten werden die nachfolgenden Thesen für diese Arbeit aufgestellt.

These 1: Für beliebige Kompositionsfragmente ist es – basierend auf semantischen Annotationen der Fähigkeiten und Schnittstellen von Komponenten sowie einem

Kompositionsmodell des Fragments – immer möglich, ein erwartungskonformes, semantisches Modell von dessen domänenspezifischen Fähigkeiten abzuleiten.

These 2: Die in These 1 genannten, semantischen Modelle der fachlichen Funktionalität von Kompositionsfragmenten bieten die Grundlage zur Vereinfachung des gesamten Entwicklungs- und Nutzungsprozesses von CWA, was nur durch durchgängige, intelligente Assistenzmechanismen erreichbar ist. Somit kann die Lücke zwischen der fachlichen Sicht von Nicht-Programmierern und der technischen Komposition überbrückt werden.

- Die Kommunikation der Plattform mit den Nutzern kann vollständig auf Basis fachlicher Terminologie erfolgen, indem anhand der Modelle verständliche Beschriftungen generiert werden.
- Es können geeignete Assistenten konzipiert werden, die es Nicht-Programmierern leichter als in bisherigen Ansätzen erlauben, Anforderungen vollständig auf fachlicher Ebene der Problemdomäne zu formulieren.
- Es ist möglich, die Komposition von Komponenten durch Abstraktion vollständig auf fachlicher Ebene durchzuführen und somit durch Nicht-Programmierer beherrschbar zu gestalten.
- Eine generische, automatische Erzeugung verständlicher natürlichsprachlicher und interaktiver Erklärungen der Funktionsweise von Kompositionsfragmenten im Sinne einer Bedienungsanleitung wird ermöglicht. Die Anreicherung des UI involvierter Komponenten mit diesen Erklärungen ist trotz Blackbox-Eigenschaft möglich und vereinfacht das Verstehen und Entwickeln von CWA für Nicht-Programmierer.

These 3: Die belang-orientierte Modularisierung eines Empfehlungssystems ist möglich und verspricht einen Grad an Konfigurierbarkeit und Adaptierbarkeit, der über bisherige Ansätze weit hinausgeht. Somit kann ein Empfehlungssystem an den Einsatz in heterogenen Umgebungen maßgeschneidert werden und Nutzer stets adäquat und durchgängig unterstützen.

1.2.2 Forschungsziele

Aufbauend auf den identifizierten Problemen und den aufgestellten Thesen ist die Vision dieser Arbeit die Konzeption einer Kompositionsplattform zur intuitiven Entwicklung und Nutzung von CWA für Nicht-Programmierer. Diese soll ausgehend von den Ergebnissen des CRUISe-Projekts notwendige Aspekte des EUD bereitstellen. Modelle zur Beschreibung von Komponenten, Fähigkeiten und Kompositionswissen, Empfehlungsstrategien und weitere Automatismen bilden hierbei die Grundlage für innovative Werkzeuge zur Entwicklung, zur Anpassung und zum Verstehen von CWA durch die zuvor definierte Zielgruppe von Nutzern. Dazu adressiert diese Arbeit folgende Forschungsziele, welche sich in vier Schwerpunkte einteilen lassen:

1. Auf Basis einer Plattform für universelle Komposition gilt es zu untersuchen, welche für ein assistiertes EUD notwendigen Aspekte zu modellieren sind und wie dies möglichst plattformunabhängig und generisch erfolgen kann. Das umfasst die folgenden Teilziele:

- Identifizierung von Konzepten zur automatischen Erkennung, modellhaften Repräsentation und Verwaltung von Kompositionswissen
- Untersuchung und Entwicklung eines generischen formalen Metamodells für Fähigkeiten und Funktionalitäten von Kompositionsfragmenten
- Schaffung eines generischen Metamodells für funktionale Anforderungen von Nutzern basierend auf existierenden Lösungen, zum Beispiel aus dem Bereich der Aufgabenmodellierung

2. Erforschung der folgenden anwendungsunabhängigen, domänenübergreifenden Mechanismen und Algorithmen als Grundlage für EUD-Werkzeuge
   - Mechanismen zur Bestimmung der domänenspezifischen Funktionalität beliebiger Kompositionsfragmente
   - Konzeption eines Empfehlungssystems für das EUD von CWA. Dieses umfasst intelligente Algorithmen zum kontextsensitiven Matching und Ranking von Kompositionsfragmenten und weist eine hochgradige Anpassbarkeit an den Einsatzkontext auf.
   - Untersuchung und Identifikation von Mechanismen zur Erfassung von Nutzerfeedback
   - Auf Basis existierender Lösungen sollen Techniken zur Überbrückung semantisch auflösbarer Inkompatibilitäten der Signaturen von Schnittstellenbestandteilen untersucht und konzipiert werden. Es gilt insbesondere diese Techniken im Hinblick auf das EUD zu optimieren, beispielsweise hinsichtlich der Komplexität bei der Ableitung und der Ausführung.
3. Konzeption generischer Architekturkomponenten einer Kompositionsplattform zum assistierten EUD von CWA. Die zu erforschenden Funktionsblöcke sollen die zuvor genannten Modelle sowie Mechanismen bereitstellen. Die Übertragung der Architekturbestandteile in existierende Kompositionsumgebungen soll gezeigt werden.
4. Entwicklung eines generischen Vorgehensmodells für das EUD von CWA für Nicht-Programmierer. Systematisierung und Definition von EUD-Werkzeugen inklusive geeigneter Interaktions- und Visualisierungsmetaphern zur ganzheitlichen Unterstützung von Nicht-Programmierern während des semi-automatischen Kompositionsprozesses unter Beachtung des zuvor beschriebenen Vorgehensmodells.

Die Forschungsziele werden im Rahmen der Anforderungsanalyse in Kapitel 2.3 weiter konkretisiert. Die entwickelten Konzepte, und somit die aufgestellten Forschungsthesen, werden zudem umfassend validiert und die Ergebnisse in Kapitel 8 dokumentiert. Dies wird durch die Bewertung hinsichtlich der Anforderungen sowie durch die prototypische Umsetzung im Rahmen einer Referenzimplementierung geschehen. Der realisierte Prototyp wird weiterhin dazu dienen, Konzepte in mehreren Beispielszenarien praktisch zu erproben und Nutzerstudien zur Erhebung von Evaluationsergebnissen, etwa hinsichtlich Gebrauchstauglichkeit und Verständlichkeit für Nicht-Programmierer, durchzuführen.

1.2.3 Annahmen und Abgrenzungen

Die vorliegende Dissertation ist in einen konzeptionellen Rahmen eingeordnet, der Annahmen und Gegebenheiten nach sich zieht, welche wiederum direkten Einfluss auf Entwurfsentscheidungen und Abgrenzungen haben.

- Als Anwendungstyp stehen komponentenbasierte Webanwendungen, CWA, im Fokus. Konzeptionelle und architektonische Basis sind die Prinzipien der universellen Komposition. Hierzu zählen deklarativ beschriebene Blackbox-Komponenten und Anwendungen sowie grobe architektonische Funktionsblöcke, wie plattformspezifische Laufzeitumgebungen, Repositorien und ein Kontextdienst.

- Die Ad-hoc-Entwicklung und Nutzung von CWA stellt eine grundlegende Rahmenbedingung dieser Arbeit dar. Nicht-Programmierer gemäß der Zielgruppendefinition entwickeln CWA, um ein situatives Problem zu lösen, und gehen dabei iterativ vor. Entwicklung und Nutzung der Anwendung sind eng verzahnt.

Folgende Themenkomplexe werden in dieser Arbeit nicht vertiefend behandelt und es wird auf existierende Lösungen aufgesetzt oder auf parallele Arbeiten verwiesen.

Modellieren, Erfassen und Auswerten von Qualitätseigenschaften: In dieser Arbeit wird auf existierende Modelle und Techniken zur Erfassung der Qualität von Komponenten und Anwendungen, wie sie beispielsweise in [Rüm+13] konzipiert wurden, zurückgegriffen. Es wird angenommen, dass Qualitätseigenschaften gegeben sind und Algorithmen existieren, mit denen diese erfasst oder aggregiert werden können. Jedoch sind Methoden Gegenstand dieser Dissertation, die sicherstellen, dass die Ergebnisse des semi-automatischen Kompositionsprozesses funktional korrekt sind und dem Interesse des jeweiligen Nicht-Programmierers entsprechen. Auch spielen nicht-funktionale Eigenschaften im Ranking eine zentrale Rolle.

Mustererkennung: Die vertiefte Spezifikation neuartiger Mechanismen zur Erkennung von Mustern in Kompositionsmodellen steht nicht im Fokus der Arbeit. Existierende Ansätze werden daher in die Konzeption einbezogen.

Entwicklung von Komponenten: In dieser Arbeit wird das Erstellen von Komponenten durch Endnutzer nicht behandelt. Vielmehr wird davon ausgegangen, dass sich Nicht-Programmierer beim Entwickeln von CWA auf ein Repertoire von Komponenten stützen, die von Drittanbietern stammen.

Multi-Device- und Multi-User-Szenarien: Im Rahmen dieser Arbeit werden solche Szenarien ausgeklammert, da bereits Single-User-Szenarien unter Berücksichtigung der Zielgruppe substantiellen Forschungsbedarf erkennen lassen. Solche Aspekte werden in parallelen Forschungsarbeiten und -projekten behandelt, zum Beispiel in den Projekten EDYRA [@EDYRA] und DoCUMA [@DoC].

Datenschutz und Privatsphäre: Insbesondere bei der Erhebung, Verwaltung und Nutzung von Feedback sowie bei der Nutzerprofilierung spielen in einem System im produktiven Einsatz Aspekte des Datenschutzes eine wichtige Rolle. Im Kontext der vorliegenden Forschungsarbeit werden sie jedoch ausgeklammert.

Zur Implementation und Evaluation baut die vorliegende Arbeit auf die CRUISE-Mashup-Plattform auf, die in Kapitel 2.1 vorgestellt wird, und erweitert diese substantiell an den für die neuen Konzepte relevanten Stellen.

1.3 Aufbau der Arbeit

Aus den zuvor vorgestellten Thesen und Forschungszielen geht hervor, dass im Rahmen dieser Arbeit wissenschaftliche Fragestellungen aus mehreren Forschungsgebieten adressiert werden. Der Aufbau der vorliegenden Arbeit orientiert sich an den Schwerpunkten der Zielsetzung. Sie umfasst neun Kapitel, deren inhaltliche Ausgestaltung nachfolgend erläutert wird.

Kapitel 2 thematisiert das konzeptionelle Rahmenwerk, in welches die Dissertation eingegliedert ist. Dazu wird die CRUISE-Mashup-Plattform vorgestellt und es werden deren Defizite aufgezeigt. Basierend darauf dienen praxisnahe Referenzszenarien zur Herleitung eines Katalogs von Anforderungen, die eine geeignete Lösung erfüllen muss.

Kapitel 3 analysiert daraufhin den aktuellen Stand von Forschung und Technik im interdisziplinären Spannungsfeld von EUD, Semantischem Web und CWA. Den Schwerpunkt bildet dabei die Analyse existierender Mashup-Plattformen in Bezug auf die aufgestellten Anforderungen. Des Weiteren finden Ansätze zu Empfehlungssystemen, zur Eingabe von Anforderungen und zur Datenmediation vertiefte Betrachtung. Die Diskussion von Lösungsansätzen und Defiziten bildet die Grundlage eines eigenen Konzepts.

Kapitel 4 beinhaltet einen Gesamtüberblick der neuartigen Konzepte des assistierten EUD von CWA durch Nicht-Programmierer. Dabei wird das Vorgehensmodell der Live-Sophistication eingeführt und die Architektur des Gesamtansatzes erklärt.

Kapitel 5 stellt die grundlegenden Konzeptbestandteile der Modellebene und die Basismechanismen im Detail vor. Capabilities, semantische Komponentenbeschreibungen, Kontextmodelle und ein neues Feedbackmodell bilden eine reichhaltige Datengrundlage. Zudem spielen neue Mechanismen wie die semantische Datenmediation sowie ein Algorithmus zur Abschätzung der Capabilities beliebiger Kompositionsfragmente eine tragende Rolle im Gesamtkonzept.

Kapitel 6 geht im Detail auf das vorgeschlagene Empfehlungssystem ein. Dieses zeichnet sich dadurch aus, dass es Nicht-Programmierer durchgängig unterstützt und anhand von Empfehlungsstrategien hochgradig an Nutzer- und Nutzungskontexte anpassbar ist.

Kapitel 7 präsentiert die konzipierten Werkzeuge, welche es Nicht-Programmierern erstmals erlauben, selbstständig sämtliche Aktivitäten der Live-Sophistication zu bewerkstelligen. Dazu gehören Assistenten zur Komposition, zur Recherche nach Kompositionsfragmenten und zum Benutzen sowie Verstehen von Anwendungen.

Kapitel 8 beschreibt die Evaluation der vorgeschlagenen wissenschaftlichen Konzepte. Hierzu werden prototypische Implementierungen im Rahmen der CRUISE-Plattform erörtert, welche zunächst die praktische Umsetzbarkeit der Konzepte untermauern. Zur weiteren Bewertung der Resultate dienen Performanzmessungen und Nutzerstudien, deren Methodik, Aufbau und Ergebnisse jeweils diskutiert werden. Daneben reflektiert das Kapitel die Konzepte hinsichtlich ihrer Anforderungserfüllung.

Kapitel 9 fasst schließlich sämtliche Teile dieser Dissertation zusammen und reflektiert die Ergebnisse sowie die wissenschaftlichen Beiträge bezüglich der Forschungsziele und -thesen. Zudem werden die Grenzen der entstandenen Lösung aufgezeigt und letztlich ein Ausblick auf Ansatzpunkte für weiterführende Arbeiten gegeben.

2 Grundlagen und Anforderungsanalyse

2.1 CRUISE – Architektur und Modelle

In diesem Kapitel werden Konzepte der universellen Komposition stellvertretend anhand der Kompositionsplattform Composition of Rich User Interface Services for Everybody (CRUISE) erläutert. Diese dient im weiteren Verlauf der Arbeit als Referenzplattform zur Konzeption und Evaluation, ohne die Übertragbarkeit und die Allgemeingültigkeit der Ansätze einzuschränken. Den Grundpfeilern der universellen Komposition folgend, basiert CRUISE auf modellhaft beschriebenen Komponenten, die mithilfe der Konstrukte eines deklarativen Kompositionsmodells zu CWA zusammengefügt werden können. Diese Metamodelle sowie das Zusammenspiel von Architekturkomponenten zur Entwicklung und Nutzung von CWA werden in den nachfolgenden Abschnitten behandelt.

2.1.1 Komponentenmetamodell

Das grundlegende Komponentenmetamodell beschreibt Komponenten aller Anwendungsebenen, das heißt der Benutzerschnittstelle, der Geschäftslogik sowie der Datenschicht, einheitlich als Blackboxen mit öffentlicher Schnittstelle [Pie12]. Komponenteninterna und Implementierungsdetails werden somit gekapselt und sind exklusiv über diese Schnittstelle zugänglich. Dahingehend, ob eine Komponente eine grafische Benutzerschnittstelle anbietet, wird zwischen UI- und Nicht-UI-Komponenten unterschieden. Komponenten können beliebige Web-Ressourcen und APIs repräsentieren, zum Beispiel Webservices auf Basis von Representational State Transfer (REST) und SOAP, Feeds, Application Programming Interfaces (APIs) für JavaScript und Widgets. Das Komponentenmetamodell ist somit Voraussetzung für die universelle, einheitliche Komposition solch verschiedener Web-Ressourcen zu kompositen Webanwendungen. Folgende Abstraktionen sind zentraler Bestandteil des Komponentenmodells:

- *Operations* stellen die bereitgestellten ausführbaren Methoden einer Komponente mit einer Signatur aus typisierten Parametern dar.
- *Events* propagieren Zustandsänderungen von Komponenten. Sie weisen ebenfalls eine Signatur aus typisierten Parametern auf.

- *Properties* sind die typisierten Eigenschaften von Komponenten, welche sowohl zur Konfiguration dienen als auch den aktuellen Zustand zur Laufzeit repräsentieren.

Als konkrete Syntax zur Beschreibung von Komponenten dient die Semantic Mashup Component Description Language (SMCDL) [PRM11b], eine deklarative Sprache basierend auf XML Schema Definition (XSD). SMCDL setzt das Komponentenmodell um und fügt weitere Metadaten und Details, welche für die Umsetzung des Lebenszyklus einer Komponente relevant sind, hinzu.

Ergänzend zum Kern des Komponentenmodells sieht SMCDL semantische Annotationen vor. Diese bieten die Möglichkeit, durch Referenzen auf Domänenontologien die folgenden Aspekte der Komponentensemantik zu beschreiben.

- *Datensemantik*: Typisierung der Properties und der Parameter von Operations beziehungsweise Events einer Komponente
- *Funktionale Semantik*: Funktionale Kategorien der gesamten Komponente oder einzelner Operations
- *Nicht-funktionale Semantik*: Metadaten und Qualitätseigenschaften, zum Beispiel Autor, Preis, visuelle Eigenschaften und technische Aspekte wie benötigte Bandbreite und unterstützte Security-Mechanismen
- *Verhaltenssemantik*: eingeschränkt ausgedrückt über Abhängigkeiten von Events zu Operationen und zu deren Funktionalität

2.1.2 Kompositionsmodell

Zur deklarativen Beschreibung aller Aspekte eines Mashups wird auf das Kompositionsmodell, Mashup Composition Model (MCM), aufgesetzt. Dieses Metamodell spezifiziert die Struktur und die Semantik gültiger Modellinstanzen. Im Wesentlichen werden die folgenden anwendungsspezifischen Aspekte vom Kompositionsmodell abgedeckt:

- zu nutzende Komponenten und deren Konfiguration (*Conceptual Model*)
- einheitliche Styling-Definitionen für UI-Komponenten (*Conceptual Model*)
- Kommunikationsbeziehungen von Komponenten (*Communication Model*)
- Anordnung der Komponenten in Ansichten mit verschiedenen Layout-Typen sowie Übergänge zwischen Ansichten (*Layout Model* und *Screenflow Model*)
- Adaption der Anwendung bei Kontextänderungen (*Adaptivity Model*)

Kommunikation zwischen Komponenten erfolgt auf verschiedene Weisen. In erster Linie kann ein lose gekoppelter Austausch von Daten über Kommunikationskanäle (*Channels*) zwischen Publisher (*Event*, *Property*) und Subscriber (*Operation*, *Property*) stattfinden. Weitere Modellkonstrukte erlauben klassische Request-Response-Kommunikation sowie die Kopplung von Properties zum Zwecke der Synchronisation. Die Datenübertragung erfolgt über den Austausch von Nachrichten, deren Nutzdatenbereich pro Parameter des jeweiligen Publisher-Schnittstellenelementes ein Instanzdatum gemäß dem jeweiligen semantischen Typ aufweist.

Im *Conceptual Model* ist neben der Referenzierung konkreter Komponenten die Möglichkeit vorgesehen, Schablonen (*Templates*) zu definieren. Dies erfolgt ähnlich zu Komponenten durch die Beschreibung einer Schnittstelle als funktionale Kriterien und optional weiterer nicht-funktionaler Anforderungen, die ein geeigneter Kandidat erfüllen muss.

2.1.3 Architekturüberblick

Eine Kompositionsplattform nach dem CRUISE-Ansatz beinhaltet eine Reihe von Architekturkomponenten, deren Zusammenwirken es erlauben, Mashup-Anwendungen zu modellieren und dem Nutzer zur Laufzeit zur Verfügung zu stellen.

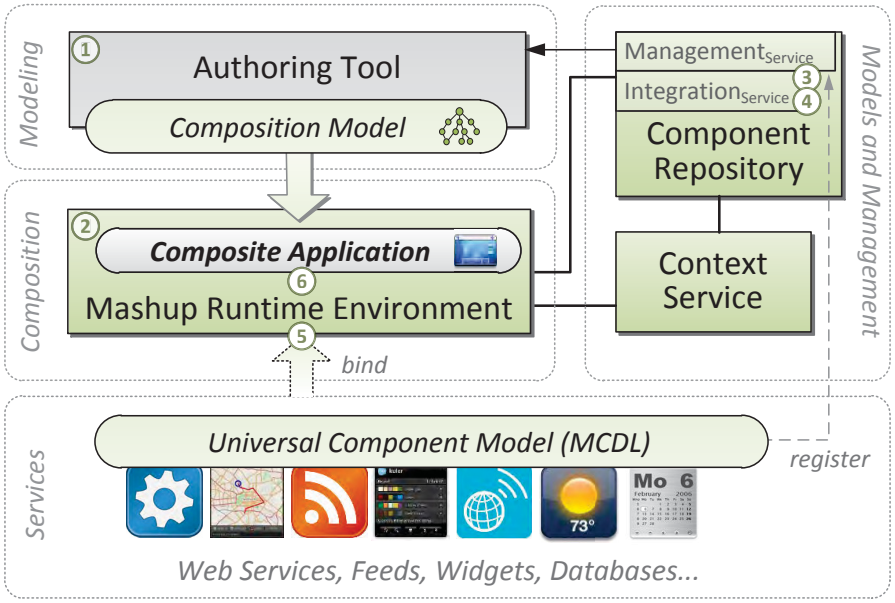


Abbildung 2.1: CRUISe-Architektur [Pie12]

Im Autorenwerkzeug erfolgt die deklarative Beschreibung eines Mashups gemäß dem Kompositionsmetamodell. Die genaue Ausgestaltung des Werkzeugs ist nicht näher vorgegeben. Wie in Abbildung 2.1 veranschaulicht, findet jedoch eine strikte Trennung zur Ausführungsumgebung, dem *Mashup Runtime Environment (MRE)*, statt.

Die vorrangige Zuständigkeit eines MRE stellt die Interpretation von Kompositionsmodellen dar, um sie in funktionstüchtige Mashup-Anwendungen zu übersetzen und diese auszuführen. Dazu gehört unter anderem das Integrieren von Komponenten sowie das Verwalten von deren Lebenszyklen, das Etablieren der Kommunikationsbeziehungen zwischen den Komponenten, das Umsetzen von Layouts sowie das Realisieren des Screenflows. Weiterhin umfasst ein MRE ein Adaptionssystem. Dieses System ermöglicht das Erfassen und Verwalten von Kontextinformationen, zum Beispiel bezüglich des Nutzer-, Geräte- und Nutzungskontextes. Dabei stützt es sich auf Kontextmonitore und einen *Kontextdienst*, welcher dem konsistenten Aktualisieren, Konsolidieren sowie Verwalten eines ontologiebasierten Kontextmodells dient und eine Registrierung auf Benachrichtigungen über Kontextänderungen erlaubt. Das Adaptionssystem ist dafür verantwortlich, Kontextänderungen aus dem Kontextdienst an das MRE und an Komponenten weiterzuleiten. Zudem bietet ein *Adaptationsmanager* eine Reihe von Techniken zur systematischen Anpassung einer Anwendung, wie den Austausch von Komponenten und das Hinzufügen

von Kommunikationskanälen. Somit kann die im Adaptivity-Model vorgesehene Adaption der Anwendung realisiert werden.

Bei der Integration von Komponenten im Rahmen der Modellinterpretation nutzt ein MRE ein *Component Repository*, das zur semantischen Verwaltung von SMCDL-Instanzdokumenten dient. Sämtliche registrierte Komponentenbeschreibungen können abgefragt und durchsucht werden. Zur Unterstützung des kontextsensitiven Integrationsprozesses besteht die Option, für ein Template passende Komponenten abzurufen. Dies ermöglicht die späte Bindung konkreter Komponenten für Templates in einem Kompositionsmodell. Dabei wird im Matching die Passgenauigkeit eines Kandidaten hinsichtlich der Anforderungen, welche durch die Template-Schnittstelle ausgedrückt sind, überprüft. Weiterhin findet ein Abgleich von nicht-funktionalen Anforderungen mit dem Kontextmodell statt, um die Reihenfolge der Kandidaten möglichst passend am aktuellen Kontext auszurichten. Falls Unterschiede zwischen Template und tatsächlicher Komponentenschnittstelle existieren, stellt das MRE Techniken zur Mediation von Nutzdaten in Kommunikationsnachrichten zur Laufzeit bereit.

2.1.4 Fazit

Die Konzepte und Modelle in CRUISE sind eine valide Basis für eine Kompositionsplattform für das EUD von CWA durch Nicht-Programmierer. An verschiedenen Stellen lassen sich jedoch Defizite identifizieren.

- Das Komponentenmetamodell und die SMCDL bieten durch semantische Annotationen eine gute Ausgangsbasis und hohes Potential für die automatische Unterstützung von Nicht-Programmierern. Es besteht jedoch Erweiterungsbedarf aufgrund fehlender oder limitierter Aspekte. So ist die Beschreibung der funktionalen und der Verhaltenssemantik in der vorliegenden Form nicht ausdrucksstark genug und Anforderungen von Komponenten an den Kontext werden nicht unterstützt. Zudem werden die semantischen Annotationen nicht vollumfänglich ausgenutzt, zum Beispiel weder zur Datenmediation noch zum Berechnen von Empfehlungen.

- Mediationstechniken existieren zwar, sind jedoch auf syntaktische Aspekte, wie die Namen und die Reihenfolge von Parametern, und lediglich Vererbungsbeziehungen von Konzepten als einzigen semantischen Aspekt begrenzt.

- Wie einleitend in Kapitel 1.1 erörtert ist eine Trennung von Entwicklung bzw. Modellierung und eigentlicher Nutzung einer Anwendung für die Zielgruppe Nicht-Programmierer aus vielerlei Gründen problematisch. Die Ad-hoc-Entwicklung von Mashups wird durch die Separation des Autorenwerkzeugs und des MRE, welche Medienbrüche verursacht, und durch mangelnde Assistenz, beispielsweise anhand von Empfehlungen, erschwert. Vielmehr bedarf es endnutzertauglicher Autorenwerkzeuge als integraler Bestandteil der Laufzeitumgebung. Die Ausgestaltung der Werkzeuge hinsichtlich Interaktions- und Visualisierungsmetaphern spielt eine ebenso zentrale Rolle wie die Anbindung an ein Empfehlungs- und ein Hilfesystem.

- Zur Unterstützung von Nicht-Programmierern durch Empfehlungen gibt es bisher keine Konzepte. Geeignete Ansatzpunkte stellen das *Component Repository* und die semantischen Annotationen von Komponenten dar. Ein Repositorium von

Kompositionsmodellen fehlt in der Architektur bisher, erscheint jedoch sinnvoll, um kollaboratives Filtern auf Basis der CWA von Nutzern einzubeziehen.

2.2 Referenzszenarien

Nachdem einleitend bereits Problembereiche diskutiert wurden und im vorherigen Abschnitt die Kompositionsplattform CRUISE eingeführt wurde, stehen in diesem Kapitel Szenarien im Fokus, welche die Herausforderungen einerseits und Anforderungen beziehungsweise Bewertungskriterien andererseits illustrieren. Ziel ist es, anhand praxisnaher Anwendungsfälle in verschiedenen Domänen einen Soll-Zustand zu skizzieren, an den sich im Verlauf dieser Dissertation durch entsprechende Konzepte angenähert wird.

2.2.1 Ad-hoc-Erstellung einer CWA zur Konferenzplanung

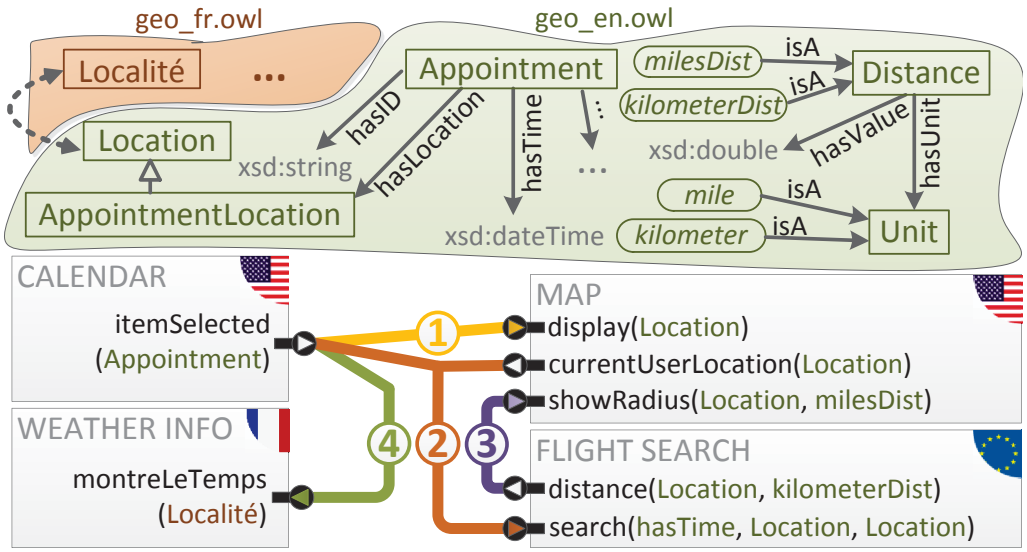


Abbildung 2.2: Beispielhafte CWA zur Reiseplanung: Im oberen Bereich ist ein Ausschnitt aus dem Domänenvokabular dargestellt, während unten die enthaltenen Komponenten (Calendar, Map, Weather Info, Flight Search) und deren Kommunikationsbeziehungen repräsentiert sind [Rad+14].

Bob ist ein aus den USA stammender Wissenschaftler und den Nicht-Programmierern zuzuordnen. Er möchte eine Konferenzteilnahme in Frankreich organisieren und entschließt sich für diesen Zweck eine Kompositionsplattform zu nutzen, mit derer Hilfe er die in Abbildung 2.2 dargestellte CWA schrittweise entwickelt.

Zuerst nutzt Bob einen angebotenen Facettenbrowser zur Suche nach Komponenten. Darin gelingt es ihm für seine gewünschte Funktionalität *Termin anlegen* ein Kalender-Widget zu finden. Nach dessen Integration in eine Anwendung durch die Plattform wählt Bob die Konferenz aus, die er zuvor als Termin in seinem Kalender hinterlegt hat. Diese Selektion propagiert das Kalender-Widget als Event, das jedoch noch von keiner anderen Komponente empfangen wird. Diesen Umstand erkennt die Kompositionsplattform und schlägt Bob die Funktionalität vor, den Ort des Termins auf einer Karte anzeigen zu lassen. Da das seinen Vorstellungen entspricht, nimmt Bob die Empfehlung an und das

Kalender-Widget sowie die gerade integrierte Karte werden verknüpft. Obgleich die dabei involvierten Schnittstellen syntaktisch nicht übereinstimmen, vergleiche Verknüpfung ① in Abbildung 2.2, gelingt es der Plattform unter Ausnutzung semantischer Annotationen und Techniken zur Datenmediation, die Kompatibilität semantisch herzustellen. Dabei wird das Domänenwissen genutzt, das ein `Appointment` über die Relation `hasLocation` mit einer `AppointmentLocation`, einer Spezialisierung von `Location`, assoziiert ist. Daher kann semantisch die `AppointmentLocation` aus dem `Appointment`, welches über das Event des Kalender-Widgets veröffentlicht wird, extrahiert und anschließend in eine `Location` umgewandelt werden.

Die Kartenkomponente zeigt Bobs aktuelle Position an und bietet diese über ihre öffentliche Schnittstelle als Property an. Bob vermittelt der Kompositionsplattform, dass er Routen von seinem aktuellen Standort zum Konferenzort suchen möchte. Das Empfehlungssystem schlägt eine funktional passgenaue, von Nutzern sehr gut bewertete Lösung vor. Diese entspricht Verknüpfung ② und bezieht ein Widget zur Flugsuche ein. Nach Bestätigung durch Bob werden automatisch die notwendigen Kommunikationskanäle und Komponenten integriert. In diesem Fall muss dafür Sorge getragen werden, dass die drei Zielparameter für die Operation `search` der Flugsuche korrekt typisiert sind und synchronisiert bereitgestellt werden. Während Bob mit der Anwendung arbeitet, ist der Backend-Service des Flugsuche-Widgets für einige Minuten nicht verfügbar. Dieser Umstand wird detektiert und alternative Komponenten werden dem Anwender unter Nennung des Grundes für diese Vorschläge empfohlen.

Die Kompositionsplattform analysiert zyklisch die vorliegende CWA von Bob. Basierend auf Kompositionswissen, das aus existierenden CWA von Nutzern der Plattform geschlussfolgert wurde, erkennt sie, dass Nutzer mit ähnlichen Komponenten und Verknüpfungen in ihren Mashups in vielen Fällen zusätzlich eine Wetteranzeige eingebaut haben. Dies wird Bob vorgeschlagen, indem ihm angezeigt wird, dass er seine Anwendung um die Funktionalität *Wetter am Ort des Termins aus dem Kalender-Widget* erweitern kann. Das interessiert Bob und er lässt sich Bewertungen sowie den Grund für den Vorschlag anzeigen. Er nimmt die Empfehlung schließlich an, wobei ihm mehrere Optionen für die Erfüllung der Funktion angeboten werden. Er wählt ein Wetterwidget aus Frankreich aus, um die Wetterprognose für den Ort und die Zeit der Konferenz einsehen zu können. Die Verknüpfung ④ ist möglich, obwohl das Wetter-Widget mit `Localité` ein Konzept aus einer anderen Ontologie referenziert als das Kalender-Widget. Semantisch ist jedoch bekannt, dass beide Konzepte äquivalent sind.

Während Bob die CWA nutzt, schätzt die Kompositionsplattform deren Funktionalität ab. Basierend darauf werden ihm nützliche Erweiterungen vorgeschlagen, etwa zur Hotelsuche. Zudem dient diese Abschätzung zur Klassifikation der Anwendung. Bob bekommt die Möglichkeit, beim Speichern der CWA darauf Einfluss zu nehmen, findet die vergebene Kategorie *Reise planen* jedoch sehr treffend und bestätigt sie deshalb.

Nachdem Bob die Plattform oft eingesetzt hat und somit seine Kenntnisse in der Mashup-Entwicklung substantiell ausbauen konnte, reduziert die Plattform automatisch das Maß an Empfehlungen.

Kernfunktionalitäten Das Szenario veranschaulicht die zentrale Bedeutung proaktiver und reaktiver Empfehlungsstrategien für eine Kompositionsplattform für das EUD von CWA. Neben einzelnen Komponenten gilt es insbesondere komplexe Kompositionsfragmente zu empfehlen und automatisch in eine CWA zu integrieren. Es wird mehrfach

angedeutet, dass Vorschläge sowohl auf strukturellen als auch auf funktionalen Vergleichen mit Kompositionswissen bestimmt werden können. Angebotene Empfehlungsstrategien sollten den jeweiligen Kenntnisstand von Nutzern beachten. Des Weiteren wird mit der Überbrückung von syntaktischen Inkompatibilitäten, die auf semantischer Ebene aufgelöst werden können, eine fundamentale Funktionalität erläutert.

2.2.2 Geführte Recherche nach einer CWA

Wissensarbeiterin Alice besitzt tiefgründige Kenntnisse in ihrer Fachdomäne, jedoch keinerlei Programmierfähigkeiten. Sie benötigt für ihr aktuelles Projekt Experten zum Thema Ontology-Engineering aus dem Großunternehmen, in dem sie angestellt ist. Da sie mit diesem Thema bisher keinerlei Erfahrung hat und daher keine in Frage kommenden Personen kennt, entscheidet sich Alice die firmeninterne Anwendungsplattform zu verwenden. Diese Plattform stellt für verschiedene Problemstellungen passende CWA bereit, unter anderem zur Unternehmenssuche. Da es neben generischen Anwendungen eine Vielzahl von Spezialsuchlösungen gibt und Alice die Plattform noch nie benutzt hat, hilft ihr das System dabei, die passende Anwendung zu identifizieren.

In einem geführten Dialog mit Alice versucht die Suchplattform durch geeignete Fragen und entsprechende Antwortmöglichkeiten die Anforderungen und Ziele von Alice systematisch einzugrenzen. Beispielsweise gibt Alice zunächst an, dass sie eine existierende Anwendung aus der Domäne *Unternehmenssuche* benutzen möchte. Anschließend erhält sie die Möglichkeit ihr Anliegen durch Eingabe von Suchbegriffen zu verfeinern. Das System hat abhängig von der gewählten Domäne und Alice' Kontext bereits begonnen, den Lösungsraum einzugrenzen, was sich etwa darin äußert, dass in der Ergebnisliste bereits nur noch CWA auftauchen, deren Funktionalitäten mit *Unternehmenssuche* semantisch in Beziehung stehen. Weiterhin erhält Alice – basierend auf Techniken des kollaborativen Filterns – Vorschläge zu Suchbegriffen, jedoch ist noch kein aus ihrer Sicht geeigneter dabei. Da Alice Experten sucht, gibt sie »Personen« ein. Das System vervollständigt den Suchbegriff für Alice zu *Suche Personen*, da sich die Fähigkeiten von Anwendungen durch ein Tupel aus Domänenobjekt und Aktivität bzw. Aufgabe auszeichnen. Im Rahmen dessen bekommt Alice Hinweise und Empfehlungen zu existierenden, ähnlichen und ergänzenden Begriffen und Aufgaben. Weiterhin steht Alice die Beantwortung optionaler Fragen zur Wahl, die sich auf nicht-funktionale Kriterien der Zielanwendung beziehen. Zum Beispiel kann sie ausdrücken, dass ihr die Anpassbarkeit des Layouts wichtig ist. Weitere Anforderungen werden implizit angenommen, beispielsweise dass nur zum Ausführungskontext passende Komponenten verwendet werden dürfen. Die Ergebnisliste umfasst stets CWA, welche die aktuellen Anforderungen zumindest partiell erfüllen. Welche Anforderungen zu welchem Grad erfüllt sind, wird Alice visualisiert. Da es mehrere Kandidaten gibt, die der groben funktionalen Anforderung *Suche Personen* entsprechen, werden Alice die optionalen oder alternativen Funktionalitäten zur Auswahl gestellt, wodurch sie implizit die Anforderungen schärft und die Kandidaten filtert. Beispielsweise gibt es zwei passende CWA, wobei eine sogar die Möglichkeit bietet, die Personen zu kontaktieren. Diese Funktion benötigt Alice nicht und wählt sie ab, sodass die entsprechende CWA im Rating bezüglich des aktuellen Kontextes von Alice sinkt. Die Plattform bietet für Ergebniseinträge die Möglichkeit, Details und eine Vorschau einzusehen. Alice nutzt dies und untersucht eine vielversprechende Anwendung. Es stellt sich jedoch heraus, dass die Eignung nicht gegeben ist, sodass Alice die Vorschau

zeitnah schließt. Dies registriert die Plattform und verfeinert unbemerkt für Alice die Anfrage. Alice wählt schließlich eine Anwendung aus und arbeitet mit dieser. Nach erledigter Rechercheaufgabe wird sie vom System gebeten, die CWA hinsichtlich ihrer Eignung zu bewerten. Die Plattform speichert das Feedback und wird es in späteren Empfehlungsprozessen berücksichtigen.

Kernfunktionalitäten In diesem Szenario steht die dialogbasierte Erfassung der Intention des Nutzers im Vordergrund. Es gilt somit durch geeignete Techniken zu ermitteln, welches Ziel und welche funktionalen Anforderungen ein Nicht-Programmierer besitzt. Die Kompositionsplattform muss darauf aufsetzend in der Lage sein, Kompositionsfragmente hinsichtlich ihrer Funktionalität zu annotieren und diese gegen die erfassten Anforderungen abzugleichen. Weitere wesentliche Funktionalitäten sind Methoden zum Erfassen, Modellieren und Verwerten von implizitem und explizitem Relevanz-Feedback.

2.2.3 Unterstützte Nutzung einer CWA

Nicht-Programmierer Bob plant seine nächste Reise und betritt hierzu eine Kompositionsplattform. Auf einem Startbildschirm, der ihm nach dem Betreten der Mashup-Plattform angezeigt wird, erscheinen verschiedene Arten von Empfehlungen für Komponenten und Anwendungen: Items, die von ähnlichen Benutzern gemocht werden; Items, die ähnlich denen sind, die Nicht-Programmierer häufig verwenden; Items, die hinsichtlich der Nutzerhistorie relevant erscheinen. Bob inspiziert diese Empfehlungen und entdeckt die Rubrik »Reisen«. Darin findet er eine Anwendung, welche ein Freund von ihm sehr gut bewertet hat und entschließt sich, diese zu verwenden. Die Anwendung besteht aus zwei Karten, einem Widget zur Suche nach Routen mit dem öffentlichen Nahverkehr, einer Wetteranzeige und zwei Komponenten zur Suche nach Hotels und nach Sehenswürdigkeiten. Da Bob weder die gesamte Anwendung noch die darin befindlichen Komponenten kennt, steht er vor einer Reihe von Verständnisschwierigkeiten hinsichtlich der Funktionalitäten, die die CWA anbietet. Beispielsweise ist sich Bob nicht sicher, weshalb zwei Karten zu sehen sind. Zudem ist ihm unklar, ob die Orte der Karten Effekte auf andere Komponenten haben und falls ja, welche das sind. Auch weiß Bob nicht, wie er Hotels in der Nähe des Zielorts suchen kann.

Während Bob normalerweise die Anwendung durch manuelles Ausprobieren verstehen müsste, wird er durch die Plattform darin unterstützt, solche funktionalen Zusammenhänge zu erschließen. Dazu steht ihm zunächst eine Übersicht der Fähigkeiten der CWA zur Verfügung, die sich typischerweise aus Kommunikationsbeziehungen zwischen Komponenten ergeben und hierarchisch aufgebaut sind. Diese Übersicht erlaubt Bob zu inspizieren, welche Aufgaben er mit der vorliegenden Anwendung lösen kann und welche Komponenten daran jeweils beteiligt sind. Dadurch wird Bob zum Beispiel bewusst, dass die eine Karte für die Auswahl des Startorts und die andere für die Selektion des Zielorts der Routensuche dient. Bei Bedarf kann sich Bob Animationen anzeigen lassen, welche ihm direkt im UI der Komponenten Interaktionsschritte aufzeigen, die etwa notwendig sind, um Ergebnisse in der Routenliste zu sehen. Bob entdeckt die Option einen Modus zu aktivieren, der ihm bei Auftreten von Datenfluss zwischen Komponenten visualisiert, was genau gerade geschehen ist. Somit wird ihm klar, dass die Karte zur Auswahl des Zielorts und die Wetteranzeige miteinander verknüpft sind. Dies ist ihm vorher nicht bewusst geworden, da die Komponenten weit auseinander positioniert sind. Die Plattform

stellt Bob Werkzeuge bereit, mit denen er die Fähigkeiten einzelner Komponenten und deren Manifestation auf dem UI erkunden kann. Dadurch erkennt er, dass er zur Auswahl eines Ortes in einer Karte sowohl ein Textfeld als auch einen Marker nutzen kann.

Kernfunktionalitäten In diesem Szenario wird deutlich, dass eine Kompositionsplattform für das EUD von CWA eine Reihe von Assistenzmechanismen bereitstellen muss, die Nicht-Programmierer beim Verstehen der Funktionsweise einzelner Komponenten und vor allem der resultierenden Gesamtanwendung behilflich sind. Dabei sollte ein direkter Bezug zum UI der Komponenten hergestellt werden und geeignete interaktiv animierte Darstellungen und textuelle Beschreibung konzipiert werden.

2.3 Anforderungen

Aus den in Kapitel 2.2 dargestellten Szenarien, den Annahmen und Rahmenbedingungen der Arbeit aus Kapitel 2.1, aus der Analyse verwandter Arbeiten und aus den Ergebnissen durchgeführter Nutzerstudien, siehe zum Beispiel [RBM13; Pfl15], ergeben sich vielfältige Herausforderungen an eine Kompositionsplattform für CWA. Für diese gilt es, Konzepte zu schaffen, die es erlauben, die Vision und Zielsetzung der Arbeit – die Unterstützung von Nicht-Programmierern bei der Entwicklung und Nutzung von CWA – zu erreichen. Zentrale Anforderungen werden in diesem Kapitel thematisiert und sowohl zur Analyse und Bewertung verwandter Ansätze als auch zur Spezifikation und Validierung eigener Konzepte aufgegriffen. Bei der Formulierung der Anforderungen wird darauf geachtet, die Unterscheidung in Muss- und Kannkriterien deutlich zu kennzeichnen. Weiterhin können Anforderungen hierarchisch strukturiert sein, wobei die jeweiligen Unteranforderungen zur Erfüllung der übergeordneten beitragen.

Eine EUD-Plattform zur Komposition von CWA muss zunächst einer geeigneten Methodik folgen. Aus dieser und unter Beachtung aktueller Forschungsergebnisse wie [RO14; KPW06; Meh+10b; NWM10; AP13; Cao13] sowie der Szenarien ergeben sich konkrete Anforderungen an eine Kompositionsplattform für das EUD von CWA.

1 Iterative Entwicklungsmethode: Einer Kompositionsplattform muss ein Vorgehensmodell zugrunde liegen, das den Eigenheiten von Nicht-Programmierern gerecht wird. Die in [Cao+10] dargestellte stark iterative Natur von EUD-Prozessen muss unterstützt werden. Demnach sollten die Phasen möglichst eng verzahnt sein und sinnvolle Übergänge untereinander erlauben. Werkzeuge müssen passend konzipiert werden, zum Beispiel indem sie weitgehend integrierte Ansichten bieten.

- **1.1 Durchgängige Werkzeuge und Assistenzmechanismen:** Sämtliche Phasen, die nicht rein kognitiver Natur sind, müssen durchgängig mit zielgruppengerechten Werkzeuggen sowie Assistenzmechanismen unterstützt werden. Beispielsweise müssen Werkzeuge zur Verfügung gestellt werden, mit denen Nutzer Probleme analysieren und passende Entwicklungsschritte vornehmen können sowie die Auswirkungen vorgenommener Schritte inspizieren und gegebenenfalls rückgängig machen können.
- **1.2 Sofortiges Feedback zu Kompositionsschritten:** Dazu muss die Entwicklung und Nutzung einer CWA so weit wie möglich fließend ineinander übergehen, sodass keine signifikanten Brüche im Arbeitsfluss entstehen.

1.3 Generizität: Die Methodik sollte soweit allgemeingültig sein, dass sie unabhängig von der Domäne der mit einer Plattform erstellbaren CWA ist.

1.4 Zielgruppe Nicht-Programmierer: Primäre Zielgruppe der Kompositionsplattform müssen Nicht-Programmierer sein. Optional wäre wünschenswert, verschiedene Vorgehensweisen und Expertisen zu unterstützen.

2 Sprache des Nutzers: Bei der Kommunikation mit Nicht-Programmierern über das UI gilt es, das ihnen bekannte Domänenvokabular zu nutzen und technische Terminologie weitestgehend zu vermeiden.

3 Abstraktion von technischen Details: Es ist erforderlich, Nutzer so weit wie möglich von der zugrundeliegenden Komplexität einer CWA, beispielsweise hinsichtlich Kommunikationskanälen und Datentypen, abzuschirmen.

4 WYSIWYG-Prinzipien: Um die Nutzung und Entwicklung von CWA stärker zu verschmelzen, muss eine Kompositionsplattform dem Prinzip *What You See Is What You Get (WYSIWYG)* folgen.

5 Unterstützung beim Einstieg in den Kompositionsprozess: Theoretische Modelle zur Informationssuche, vergleiche Abschnitt 1.1, legen nahe, dass mehrere, nutzergerechte Einstiegspunkte angeboten werden müssen, um Nicht-Programmierern den Start des Kompositions- oder Nutzungsprozesses zu erleichtern.

5.1 Direkteinstieg: Dem Nutzer müssen Optionen zum unmittelbaren Arbeitsbeginn gewährt werden, zum Beispiel durch Auflistung von Komponenten und CWA basierend auf der Historie und auf kollaborativem Filtern [Hea99]. Dadurch soll der Nutzer inspiriert werden, was insbesondere bei vagem Informationsbedürfnis wichtig ist [Kec+13].

5.2 Erfassung funktionaler Anforderungen: Dies muss vorgesehen sein. Nutzer müssen mit fachlichen Begriffen ihre Vorstellung von der Anwendungsfunktionalität ausdrücken können. Assistenzmechanismen wie Suchanfragevorschläge sind vorzusehen. Der Dialog mit dem Nutzer muss iterativ erfolgen und verschiedene Strategien unterstützen, wie die Dekomposition grober und die Komposition feingranularer Ziele. Jederzeit müssen zu den Anforderungen passende Ergebnisse präsentiert werden. Zudem sollte es Nutzern möglich sein, die Relevanz von Ergebnissen zu erkennen oder zu bewerten.

6 Unterstützung bei der Komposition: Um Barrieren und Frustration zu vermeiden, muss eine Kompositionsplattform die Entwicklungstätigkeiten des Nutzers unterstützen. Dabei sollten endnutzergerechte Einflussmöglichkeiten in die Komposition vorgesehen sein, wie das Hinzufügen und Verknüpfen von Komponenten.

6.1 Intuitive Kompositionsmetapher: Es müssen für Nicht-Programmierer geeignete Visualisierungs- und Interaktionsmetaphern vorhanden sein. In aktuellen Ansätzen findet vorrangig *Orchestrierung* statt, wobei Nutzer Kommunikationsbeziehungen explizit definieren, oft graphisch nach der Metapher des *Verdrahtens* [Wil+12]. Ob diese für alle Nicht-Programmierer intuitiv ist, nicht nur für Personen aus bestimmten Domänen wie Ingenieurswissenschaften, ist unklar [@Wil]. Einen Gegenentwurf stellt *Choreographie* dar, bei der Kommunikationskanäle ohne Eingreifen des Nutzers durch vorhandene Komponenten entstehen. Fehlende Einflussmöglichkeiten und mangelnde Nachvollziehbarkeit bilden hierbei gravierende Nachteile.

6.2 Datenmediation: Es müssen semantischen Techniken zur Datenmediation unter Ausnutzung der annotierten Ontologiekonzepte von Schnittstellen bereitgestellt und im Bedarfsfall automatisiert appliziert werden. Die korrekt typisierte und formatierte Bereitstellung aller benötigten Eingabeparameter für das Schnittstellenelement der Zielkomponente eines Kommunikationskanals ist dabei die übergeordnete Anforderung.

6.2.1 Semantikbasiert: Mediationstechniken müssen sich auf Domänenvokabular stützen. Wünschenswerte Kriterien sind die Durchführung von Transformationen auf semantischer Ebene und der Austausch semantischer Daten, um Heterogenität bei der Serialisierung zu umgehen.

6.2.2 Unterstützte Heterogenität: Folgende überbrückbare Inkompatibilitäten müssen behandelt werden können. Syntaktische Unterschiede sind zu nennen, wie die Benennung und Reihenfolge von Schnittstellenelementen und von deren Parametern. Auf semantischer Ebene müssen variierende Abstraktionsgrade, wie Subklassenbeziehungen, und Granularitäten angeglichen werden. Für primitive Datentypen muss Formatanpassung (Konvertierung von Maßeinheiten und Dimensionen sowie String-Manipulation) angeboten werden. Weiterhin sollten Konzepte zum Umgang mit Kollektionen im Vergleich zu Einzelitems vorhanden sein. Wünschenswert ist die Unterstützung mehrerer Ontologien für eine Domäne.

6.2.3 Unterstützte Kommunikationsbeziehungen: Es sollten Kommunikationsbeziehungen mit beliebig vielen Start- und Zielschnittstellenelementen unterstützt werden (N:m-Konstellationen).

6.2.4 Eignung für Kompositionsplattformen von CWA: Zur Integration eines Ansatzes in eine EUD-Plattform für CWA müssen notwendige Abbildungen zwischen Komponentenschnittstellen automatisch abgeleitet und ausgeführt werden können. Zudem muss der Mediationsaspekt architektonisch geeignet eingegliedert sein.

6.2.5 Organisation von Abbildungsvorschriften: Abbildungsvorschriften sollten in wiederverwendbarer Form im System vorgesehen sein. Dazu gehört deren geeignete Repräsentation integriert im Domänenvokabular oder separiert davon.

6.3 Korrektheit der Komposition: Es müssen Ansätze vorgesehen sein, die es erlauben, sicherzustellen oder zu überprüfen, dass erstellte Anwendungen funktionstüchtig sind. Dazu müssen zumindest fehlerhafte Kompositionszustände vermieden werden. Wünschenswert sind Mechanismen, die gewährleisten, dass die Aufgabe oder die Ziele des Nutzers erfüllt werden.

7 Bereitstellung von Kompositionswissen: Als Basis zur Assistenz von Nutzern muss eine Plattform ein Repertoire an Kompositionswissen aufbauen und pflegen.

7.1 Modellierung von Kompositionswissen: Kompositionswissen sollte einem Metamodell entsprechend beschrieben und wiederverwendbar abgelegt werden. Als Basis sollten beliebig komplexe Kompositionsfragmente dienen. Daneben gilt es, den Nutzungskontext in geeigneter Form abzubilden.

7.2 Erfassungsmechanismen: Ansätze müssen vorhanden sein, die auch nach Initiierung der Plattform einen Grundstock an Kompositionswissen bereitstellen. Die kontinuierliche Verbesserung des Wissens sollte vorgesehen sein.

7.3 Metamodell für die Funktionalität von Kompositionsfragmenten: Zur Kommunikation mit Nicht-Programmierern und zur Bestimmung von Vorschlägen ist es notwendig, die funktionale Semantik von Kompositionsfragmenten zu modellieren. Dabei müssen domänenspezifische Begriffe und Aufgaben unterstützt werden, um die Verständlichkeit für Nicht-Programmierer zu erhöhen. Weiterhin sind hierarchische Strukturen zu ermöglichen, da durch das Zusammenspiel von Komponenten übergeordnete Fähigkeiten entstehen. Das Modell muss für beliebige Kompositionsfragmente anwendbar sein.

7.4 Ableiten der Funktionalität von Kompositionsfragmenten: Es muss ein Verfahren geben, mit dessen Hilfe zuvor genanntes Metamodell zur Beschreibung der funktionalen Semantik instantiiert werden kann. Das Ergebnis muss möglichst automatisch bestimmt werden können und passend sein. Es gilt, implizit vorhandene, übergeordnete Funktionalitäten zu erkennen.

8 Empfehlungssystem: Eine Kompositionsplattform für EUD von CWA muss Nicht-Programmierern relevante Empfehlungen zu Kompositionsschritte anbieten.

8.1 Durchgängige Unterstützung: Das Empfehlungssystem muss den Entwicklungsprozess vollständig abdecken. Die Empfehlungsberechnung muss proaktiv und reaktiv unter definierten Umständen ausgelöst werden können. Zudem muss es Strategien geben, die verschiedenen Nutzerintentionen zur Erlangung von Empfehlungen (explizit oder implizit angefordert) Rechnung tragen.

8.2 Hybridität: Das Empfehlungssystem sollte eine hybride Strategie zur Berechnung von Empfehlungen verwenden, um die komplementären Vor- und Nachteile von inhaltsbasierten Ansätzen und von Techniken des kollaborativen Filterns auszunutzen, siehe Kapitel 1.1.

8.3 Kontextsensitivität: Die Eignung von Kompositionswissen ist in der Regel vom jeweiligen Kontext abhängig. Daher muss das Empfehlungssystem bei der Berechnung von Vorschlägen unter anderem den aktuellen Kompositionskontext, den Nutzerkontext und die kontextabhängige Eignung von Komponenten berücksichtigen. Insbesondere die funktionale Relevanz muss ein Kriterium sein. Auch sollten Relevanz-Feedback und Bewertungen von Kompositionsfragmenten in das Ranking einfließen.

8.4 Nachvollziehbarkeit: Damit Nicht-Programmierer den Vorschlägen gegenüber Vertrauen entwickeln können, sollten zumindest Grund und Herkunft von Empfehlungen kommuniziert werden. Neben strukturellen Metadaten, wie enthaltene Komponenten, sollte die Beschreibung einer Empfehlung idealerweise verdeutlichen, welche Funktionalität sie zur CWA beiträgt.

8.5 Konfigurierbarkeit: Die Ansprüche an ein Empfehlungssystem hinsichtlich Grad und Art der Unterstützung hängt stark vom Kontext ab, beispielsweise von den Fähigkeiten des Nutzers sowie der Art und Domäne von Mashups, die mit der Plattform erzeugt werden können. Daher sollte ein Empfehlungssystem dahingehend konfigurierbar sein, wann welche Art von Empfehlungen berechnet wird und wie diese dem Nutzer präsentiert werden.

8.6 Automatische Integration: Wählt ein Nicht-Programmierer ein vorgeschlagenes Kompositionsfragment aus, muss eine Kompositionsplattform alle not-

wendigen Schritte zur Umsetzung des Vorschlags in der aktuellen CWA weitestmöglich automatisch vornehmen.

8.7 Mechanismen zur Erfassung von Nutzerfeedback: Zur Verbesserung von Empfehlungen über die Laufzeit des Empfehlungssystems ist es notwendig, sowohl implizites als auch explizites Feedback zur Relevanz von Vorschlägen und zur Güte von Kompositionsfragmenten zu erheben.

9 Unterstützung bei der Inspektion und beim Verstehen von CWA: Dies ist notwendig für Nicht-Programmierer, wie in Nutzerstudien gezeigt wurde. Deshalb muss eine Kompositionsplattform dahingehend geeignete Werkzeuge bereitstellen.

9.1 Erklärungstechniken: Es müssen Konzepte zum Erklären der Funktionsweise von Komponenten und insbesondere von deren Zusammenspiel im Mashup vorgesehen sein. Die Konzepte sollten einen Bezug zwischen modellhafter Beschreibung der Funktionalität und dem UI herstellen und Nicht-Programmierern auf interaktive, textuelle und animierte Weise Erklärungen anbieten. Die Techniken müssen dynamisch für beliebig komplexe Zusammenhänge angewendet werden können, beispielsweise mit sequentiellen und parallelen Abläufen sowie Synchronisationspunkten, mit transitiven Verbindungen und mit Verteilung von Komponenten über mehrere Anwendungsansichten.

9.2 Awareness für Kommunikation zwischen Komponenten: Konzepte sind erforderlich, die Nicht-Programmierer im Bedarfsfall auf stattfindende Kommunikation zwischen Komponenten aufmerksam machen.

9.3 Abgleich mit Anforderungen: Sofern Anforderungen vom Nutzer genannt wurden, wären Ansätze wünschenswert, die deren Erfüllungsgrad und Manifestation in einem Kompositionsfragment veranschaulichen.

Die überwiegende Zahl der genannten Anforderungen sind als Musskriterien zu verstehen. Lediglich die Teilanforderungen 6.2.3, 6.2.5 und 9.3 stellen Kannkriterien dar. Zusätzlich zu den bisher gelisteten funktionalen Anforderungen, gilt es eine Reihe nichtfunktionaler Anforderungen zu berücksichtigen. Hierzu zählen architektonische Aspekte wie **Erweiterbarkeit** und **Trennung von Belangen**. Gerade im EUD mit geringstmöglichen Brüchen zwischen Nutzung und Entwicklung spielt **Performanz** und **Stabilität** von Systemfunktionalitäten, etwa bei Datenmediation, eine wichtige Rolle. Weiterhin wäre es wünschenswert an geeigneten Stellen auf **Standards** oder weitgehend etablierte Lösungen aufzusetzen, da diese in der Regel einen gewissen Grad an Reife und Akzeptanz aufweisen. Vor allem in Bezug auf das UI der Plattform und der Werkzeuge gilt es, auf **Gebrauchstauglichkeit** zu achten. Der Frage, inwiefern die Ergebnisse der vorliegenden Disseration den gestellten Anforderungen gerecht werden, widmet sich Kapitel 8.

In diesem Abschnitt wurde zunächst der Ist-Stand typischer Ansätze für die universelle Komposition am Beispiel von CRUISE dargelegt und hinsichtlich der Zielstellung eingeordnet. Unter Zuhilfenahme von Szenarien im EUD-Kontext erfolgte anschließend die Anforderungsanalyse, welche schließlich in einen umfangreichen Anforderungskatalog resultierte. Basierend auf diesen Kriterien wird im nachfolgenden Kapitel der Stand von Forschung und Technik nach Lösungsansätzen untersucht.

3
Stand von Forschung und Technik

Für diese Dissertation relevante Ansätze aus Forschung und Technik befinden sich im Spannungsfeld von drei wesentlichen Forschungsgebieten, die in Abbildung 3.1 veranschaulicht werden. Unmittelbar bedeutsam ist der Bereich von Webservices und kompositen Webanwendungen. Weiterhin spielt das interdisziplinäre Gebiet des EUD eine zentrale Rolle. Letztlich liefern auch Erkenntnisse aus dem Bereich des Semantischen Web wichtige Beiträge für die Erfüllung von Anforderungen. Insbesondere zwei Schnittmengen dieser Forschungsgebiete haben direkten Einfluss auf diese Dissertation und beinhalten den Großteil der in diesem Kapitel untersuchten Vertreter und Ansätze. Hochgradig verwandt sind Kompositionsplattformen für das EUD kompositer Webanwendungen. Diesbezüglich wird untersucht, welche Unterstützung in Form von Werkzeugen und Automatismen Endnutzern angeboten werden und welcher Metaphern sich diese bedienen. Weitere Einflüsse haben Erkenntnisse aus dem Bereich EUD bezüglich Vorgehensweisen von Nutzern und deren Implikationen auf gestalterische Aspekte des UI. Der Bereich der Semantischen Webservices bietet Ansätze für grundlegende, automatisierbare Konzepte einer Kompositionsplattform, mit denen Nutzer typischerweise nicht in Berührung kommen, zum Beispiel semantische Modelle und Mediationstechniken.

Die vorliegende Dissertation bedient sich schließlich aus allen drei Gebieten an Erkenntnissen und ist somit in deren Schnittmenge zu verorten. Nachfolgend werden wesentliche verwandte Arbeiten näher analysiert. Den Schwerpunkt bildet zuerst die genaue Analyse von EUD-Kompositionsplattformen in Abschnitt 3.1. Mit der nachgelagerten Untersuchung von Empfehlungssystemen in Abschnitt 3.2, Eingabe funktionaler Anforderungen in Abschnitt 3.3 und Datenmediation in Unterkapitel 3.4 werden Detailaspekte betrachtet, wobei auch Ansätze aus anderen Bereichen einbezogen werden.

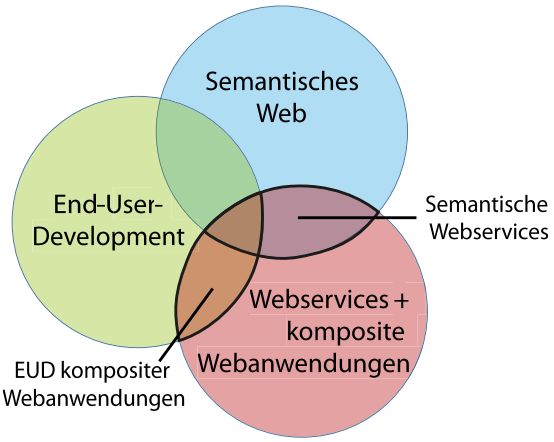


Abbildung 3.1: Verwandte Forschungsbereiche

3.1 Kompositionsplattformen für EUD

Die Analyse und Bewertung aktueller Plattformen zum EUD von kompositen Webanwendungen und zur Webservice-Komposition bilden den Schwerpunkt dieses Kapitels. Dabei kommen Bewertungskriterien zum Einsatz, welche die Anforderungen aus Kapitel 2.3 zur übersichtlicheren Darstellung wie folgt zusammenfassen.

★***Entwicklungsmethodik*** inkludiert Anforderung 1.
★***Abstraktionsgrad*** umfasst die Anforderungen 2 bis 4.
★***Unterstützung bei Einstieg und Suche*** entspricht Anforderung 5.
★***Unterstützung bei Komposition*** ist beschrieben in den Anforderungen 6 und 7.
★***Empfehlungssystem*** entspricht Anforderung 8.
★***Unterstützung bei Nutzen und Verstehen*** inkludiert Anforderung 9.

Eine Gesamtübersicht der Ergebnisse sämtlicher untersuchter Ansätze enthält Tabelle 3.13. Dabei wird eine fünfstufige Bewertungsskala eingesetzt, die den durchschnittlichen Erfüllungsgrad der Anforderungen des jeweiligen Bewertungskriteriums widerspiegelt.

3.1.1 Webservice-Komposition durch Endnutzer

Einen Überblick dieses Forschungsbereichs gewähren die Artikel von Lemos et al. [LDB15] sowie von Hang und Zhao [HZ15]. Demnach nutzen existierende Ansätze zur Webservice-Komposition zumeist workflow-basierte Editoren, in denen die Kompositionslogik vorrangig graphisch modelliert wird. Kennzeichnend ist weiterhin ein Mangel an WYSIWYG-Prinzipien und eine Trennung von Entwerfen und Nutzen. Insbesondere in kommerziellen Produkten werden weder Erkenntnisse aus den SWS zur Automatisierung noch Wiederverwendung von Kompositionswissen berücksichtigt. Oft ist lediglich Stichwortsuche vorhanden. Insgesamt zeigt sich daher eine begrenzte Eignung für Nicht-Programmierer.

Wenige akademische Vorschläge wurden unterbreitet, die explizit Nicht-Programmierer adressieren: BOP [Ro+08], SOA4All [Meh+14; Meh+10a; NND10b; NWM10] und ServFace [NNS11; Dan+10; NND10a; Fel+09]. Aufgrund ihrer Bekanntheit werden die zwei letztgenannten stellvertretend im Folgenden genauer vorgestellt.

Im Projekt **SOA4All** [Meh+10a; Meh+14] wurde ein Ansatz zur assistierten Komposition von SWS entwickelt. Die Zielgruppe ähnelt stark den Nicht-Programmierern gemäß der Definition in Kapitel 1.1. Der Ansatz basiert auf *Templates*, die prototypische Kompositionsgraphen beschreiben und an semantischen Aufgabenbeschreibungen orientiert sind. Templates stellen Kompositionswissen dar, das zum Beispiel aus vorhergehenden erfolgreichen Kompositionen oder von Experten bereitgestellt wird, und erlaubt dessen Wiederverwendung. Es existieren vordefinierte Templates für verschiedene Aufgabenbereiche. Die Komposition erfolgt in einem Werkzeug, das in Abbildung 3.2 dargestellt wird. Der Nutzer wählt dabei ein Template aus einer Taxonomie (siehe linke Seite), das daraufhin in der Canvas angezeigt wird, wobei pro Aufgabe eine Spalte erscheint (rechter Teil). Die Zuständigkeit des Nutzers besteht darin, pro Aufgabe einen Service auszuwählen. Das System hilft ihm hierbei dahingehend, dass es semantisch zur Aufgabe und zur bereits vorgenommenen Auswahl passende Services berechnet und hervorhebt. Dabei stützt es sich auf semantisches Schlussfolgern über annotierte Konzepte. Fertige Kompositionen können ausgeführt werden. **Fazit:** Die ★*Entwicklungsmethodik* wird der Anforderung

nach Generizität und der Zielgruppe gerecht. Allerdings ist sie nur eingeschränkt iterativ, es findet eine starke Trennung von Entwurf und Nutzung statt und die Assistenz deckt nur die Kompositionsphase ab. Der ★*Abstraktionsgrad* ist aufgrund der aufgabenbasierten Templates als gut zu bewerten, jedoch fehlt es an WYSIWYG-Prinzipien. Deutliche Defizite zeigen sich bei der ★*Unterstützung bei Einstieg und Suche*, da lediglich die Template-Taxonomie vorgesehen ist. Die simple Kompositionsmetapher der Template-Instanziierung und die semantikbasierte Fehlervermeidung tragen zur ★*Unterstützung bei Komposition* bei. Jedoch sind die Einflussmöglichkeiten des Nutzers begrenzt und es fehlt an Datenmediation. Ein ★*Empfehlungssystem* lässt der Ansatz missen. Die ★*Unterstützung bei Nutzen und Verstehen* erscheint limitiert, da außer der Zuordnung von Service zu Aufgabe keinerlei Mechanismen bereitstehen.

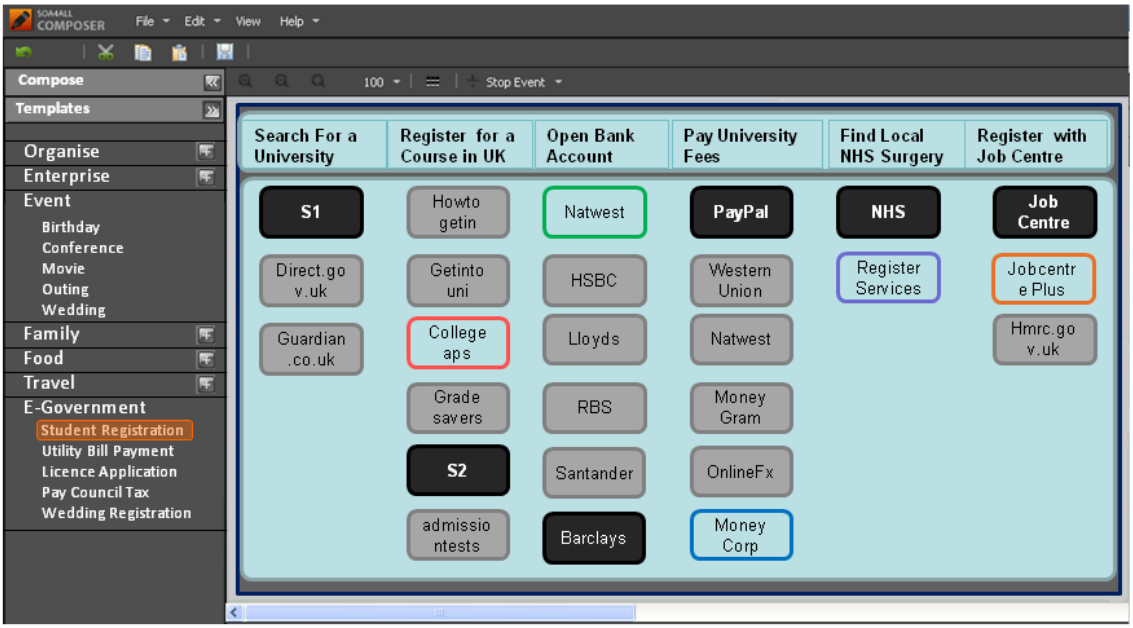


Abbildung 3.2: Werkzeuge zur assistierten Webservicekomposition in SOA4All [Meh+14]

Im Projekt **ServFace** entstand ein Ansatz zur Komposition von SOAP-Webservices für Nicht-Programmierer [NNS11; Dan+10; Fel+09]. Einheiten zur Komposition sind formularbasierte *Frontends*, welche für Webservice-Operationen auf Basis von Annotationen generiert werden. Den Datenfluss zwischen Frontends bzw. Webservices definieren *Frontend Relations*. Weiterhin ist es möglich, Frontends auf *Pages* zu positionieren und die Navigation durch Transitionen zwischen diesen, die *Page Relations*, festzulegen. Der Kompositionsprozess findet im *ServFace Builder* statt, den Abbildung 3.3 zeigt. Der Nutzer beginnt mit der Selektion einer Zielplattform und wählt anschließend aus einer Liste von Webservices und zugehörigen Operationen aus, woraufhin die entsprechenden Frontends generiert und angezeigt werden. Auf UI-Ebene können nun Relationen zwischen Frontends hergestellt werden, etwa zwischen Ausgabe- und Eingabeelementen verschiedener Webservice-Operationen. Weiterhin kann die Navigation in einer dedizierten, graphartigen Ansicht spezifiziert werden. Alternativ dazu werden vordefinierte Navigationsmuster angeboten, zum Beispiel ausschließlich sequentielle Abläufe von Pages wie in einem Wizard. **Fazit:** Hinsichtlich der ★*Entwicklungsmethodik* ergeben sich Defizite, da die Generizität durch Fokus auf SOAP-Webservices und den UI-Generierungsansatz

eingeschränkt wird. Zudem findet keine durchgängige Assistenz statt (begrenzt auf Komposition) und Entwurf und Nutzung sind stark getrennt. Der ★*Abstraktionsgrad* zeigt mit der UI-basierten Kompositionsmetapher einen guten Ansatz, der zudem WYSIWYG aufgreift. Allerdings wird der Nutzer mit technischer Terminologie konfrontiert. Abstriche bei der ★*Unterstützung bei Einstieg und Suche* ergeben sich daraus, dass einzig die Liste von Webservices vorgesehen ist. Die ★*Unterstützung bei Komposition* umfasst eine intuitive Kompositionsmetapher mit adäquaten Einflussmöglichkeiten. Teilautomatisierung wird nicht für beliebig komplexe Fragmente angeboten und es fehlt an Datenmediation. Der Ansatz sieht kein ★*Empfehlungssystem* vor. Und auch eine ★*Unterstützung bei Nutzen und Verstehen* ist nicht vorhanden. Einzig nennbar sind hier die Frontend-Relations, die jedoch nur bei der Komposition Verwendung finden.

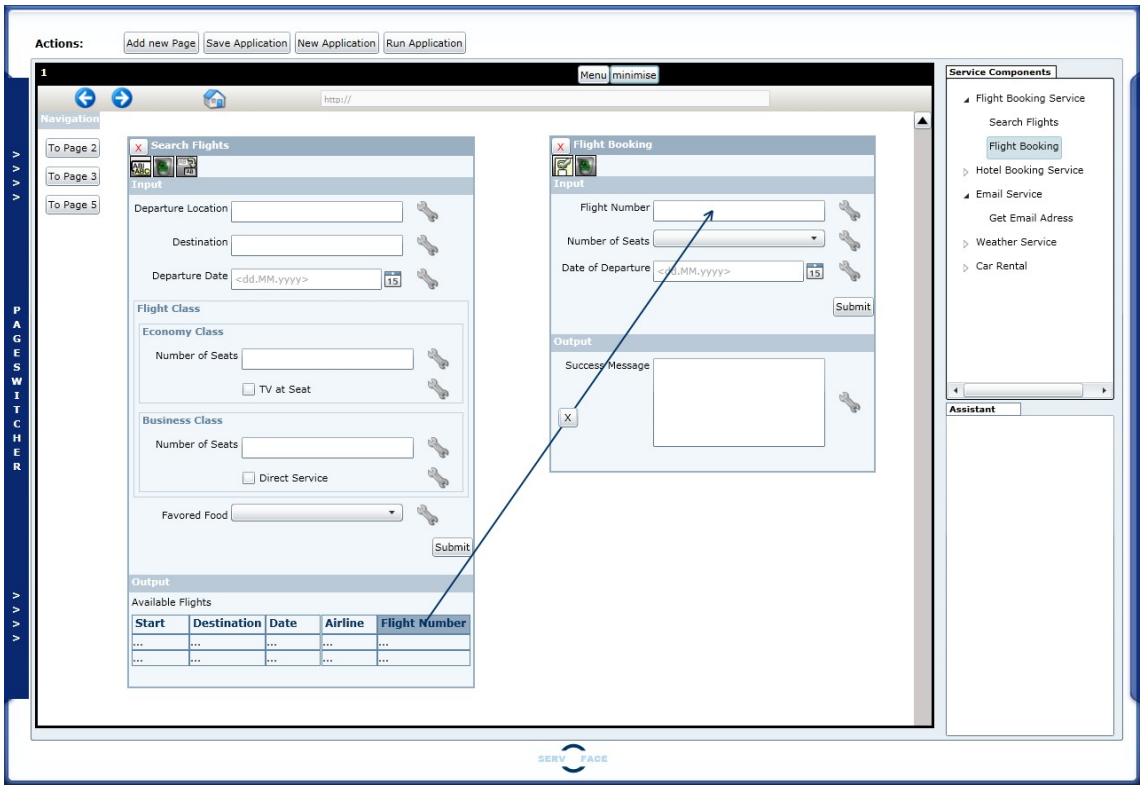


Abbildung 3.3: Der *ServFace Builder* [NNS11]

3.1.2 Mashup-Komposition durch Endnutzer

Zahlreiche Plattformen zur Erstellung von Mashups haben sich herausgebildet. Die Analyseergebnisse zu relevanten Vertretern werden nachfolgend im Detail vorgestellt. Daneben sei auf die Mashup-Plattformen **RUBIK** [Bia+14], **SMASHAKER** [BDM10; BDM13a; Mel11] und **ResEvalMash** [Imr13] hingewiesen, die aufgrund der vergleichsweise geringen Anforderungserfüllung, vergleiche Tabelle 3.13, nicht genauer erläutert werden.

Das Projekt **OMELETTE** erforschte die Komposition von W3C-Widgets. Ein wesentliches Ergebnis stellt eine Kompositionsplattform für CWA dar, die den Ansatz der *Live-Entwicklung* verfolgt. Dabei obliegt es einer Ausführungsumgebung, neben der

benutzbaren Bereitstellung auch die werkzeuggestützte Manipulation von CWA zu ermöglichen. Ähnlich zu CRUISE werden Widgets deklarativ beschrieben und in einer Registry hinterlegt [Con13a]. Als Alleinstellungsmerkmal basiert die Komposition auf dem Konzept der *Choreographie*. Komponenten werden demnach event-basiert standardmäßig in allen möglichen Konstellationen verknüpft und der Nutzer ist dafür zuständig, nachträgliche Einschränkungen vorzunehmen [Chu+13; Tsc+14]. Unterstützung erhält der Nutzer bei der Komposition in zweierlei Hinsicht [Roy+13], siehe Abbildung 3.4. Zunächst wird ein *Agent* angeboten, der in einem geführten Dialog die funktionalen Ziele des Nutzers abfragt. Für eine Detailbetrachtung sei auf Abschnitt 3.3 verwiesen. Anhand der Anforderungen kann ggf. eine passende Komposition durch die *Automatic Composition-Engine* hergeleitet werden und auf der Canvas angezeigt werden. Während der Bedienung erhalten Nutzer weiterhin Kompositionsvorschläge. Diese basieren auf einem Empfehlungssystem, das auf Kompositionswissen in Form von *Patterns* beruht, die z. B. häufige Konstellationen von Komponenten beschreiben [Roy+10; RDC14]. Nähere Ausführungen dazu sind Gegenstand von Kapitel 3.2. Wählt ein Nutzer ein *Pattern* aus, wird es automatisch in die Komposition eingewebt. Es wurden Konzepte entwickelt, die das Bewusstsein von Nutzern über Kommunikationsbeziehungen erhöhen sollen [Chu+13; Tsc+14]. Letztere werden hierzu auf der Kompositionsfläche angezeigt, indem zwischen Widgets Pfeile gezogen und isolierte Widgets gesondert hervorgehoben werden. Zusätzlich wird das Auftreten von Datenaustausch zur Laufzeit durch dezente Indikatoren an den involvierten Widgets verdeutlicht. OMELETTE bietet somit den einzigen Ansatz, der den zur Laufzeit auftretenden Datenaustausch indiziert. Die Autoren selbst weisen darauf hin, dass passendere Visualisierungstechniken für Inter-Widget Communication (IWC) noch zu erforschen sind. **Fazit:** Der Ansatz bietet eine ★*Entwicklungsmethodik*, die für Nicht-Programmierer geeignet, generisch sowie iterativ ist, und prozessbegleitend Assistenz vorsieht. Auch bezüglich des ★*Abstraktionsgrads* sind kaum Defizite zu erkennen. Anders verhält es sich hinsichtlich ★*Unterstützung bei Einstieg und Suche*. Zwar existiert neben einer Palette ein Agent zur Anforderungseingabe. Dieser wird jedoch nicht den Anforderungen gerecht und explorative Einstiegspunkte lässt der Ansatz missen. Die ★*Unterstützung bei Komposition* offenbart gute Ansätze durch das Bereitstellen und Einweben von Patterns sowie die Einflussmöglichkeiten des Nutzers. Die Kompositonsmetapher entspricht allerdings dem »Verdrahten« und ein Beleg, dass Choreographie ein intuitiver Ansatz ist, wird nicht erbracht. Es mangelt zudem an Datenmediation und Fehlerbehandlung. Brauchbare Konzepte, wie Patterns und zur Beachtung des Kompositionskontextes, liefert das ★*Empfehlungssystem*. Es weist jedoch zahlreiche Schwächen auf, wie Kapitel 3.2 aufzeigt. Durch die Anzeige von Relationen zwischen Komponenten und die Indikation von auftretendem Datenaustausch ist die ★*Unterstützung bei Nutzen und Verstehen* nur teilweise erfüllt. Hinsichtlich der Erklärungstechniken zeigen sich Defizite, da weder ein konkreter Bezug zum UI gegeben ist, noch in animierter und textueller Form die Funktionsweise beliebiger Kommunikationskanäle erklärt werden.

DEMISA [Tie15] verfolgt einen modellgetriebenen Ansatz zur Erstellung von Mashups auf Basis von Geschäftsprozessmodellen. Diese werden als Aufgabenmodelle durch Domänenexperten formal definiert und anschließend in einem semi-automatischen Prozess auf Komponenten und CWA abgebildet. Ein wesentlicher Bestandteile der Methodik ist ein semantisches Aufgaben-Metamodell, das die hierarchische Beschreibung von Aufgaben unter Zuhilfenahme von Referenzen auf Domänenkonzepte in Ontologien erlaubt. Für Endnutzer wurde ein Aufgabeneditor konzipiert, der zur Modellierung von zu lösenden Aufgaben

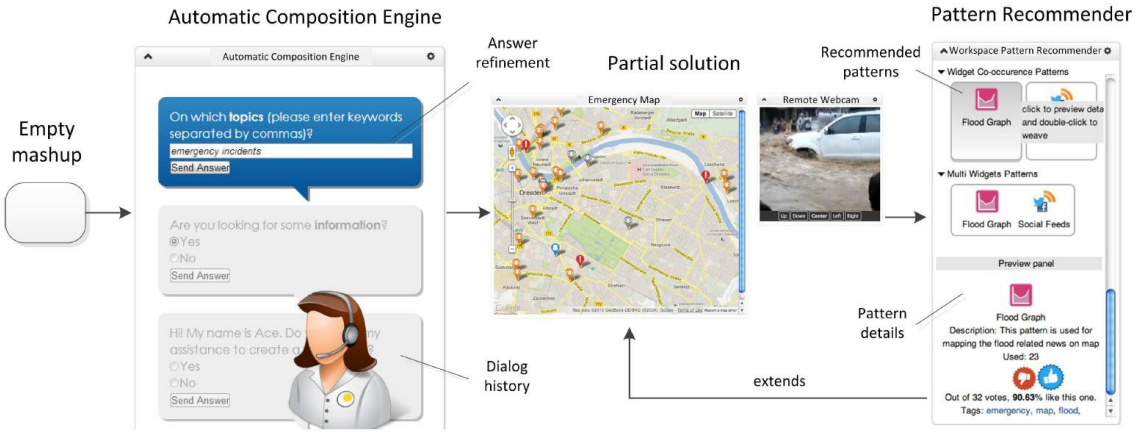


Abbildung 3.4: Unterstützung bei der Komposition in OMELETTE (nach [Roy+13])

verwendet wird und somit der Definition von Anforderungen an Mashup-Anwendungen dient. In Abschnitt 3.3 folgt eine genauere Untersuchung dieses Werkzeugs. Unterstützung erfährt der Nutzer im Kontext des Aufgabeneditors bei der Abbildung fachlicher Anforderungen auf technische Lösungen durch zwei Algorithmen. Zunächst dient ein Empfehlungssystem dazu, einzelne Komponenten für Aufgaben vorzuschlagen. Dabei findet ein Abgleich der annotierten Ontologiekonzepte an den Aufgaben mit denen der Komponenten statt. Darauf aufsetzend versucht ein Kompositionsalgorithmus automatisch für ein komplettes Aufgabenmodell die angesichts verfügbarer Komponenten bestmögliche CWA für die CRUISE-Plattform zu erzeugen. **Fazit:** Die ★*Entwicklungsmethodik* ist iterativ, generisch und für eine passende Zielgruppe. Jedoch tritt ein starker Bruch zwischen Entwerfen sowie Benutzen auf und Nutzern wird nicht im gesamten Prozess Assistenz geleistet. Der ★*Abstraktionsgrad* ist angesichts der Aufgabenorientierung sehr gut für Nicht-Programmierer, allerdings fehlen WYSIWYG-Ansätze. Mit dem Aufgabeneditor steht ein gutes Konzept für die ★*Unterstützung bei Einstieg und Suche* bereit, weitere Einstiegspunkte lässt DEMISA jedoch missen. Die ★*Unterstützung bei Komposition* zeigt Defizite, da kein Kompositionswerkzeug angedacht ist und somit keine Aussagen zu Fehlerbehandlung und Einflussmöglichkeiten getroffen werden können. Die Kompositionsmetapher des Verknüpfens von Aufgaben ist positiv zu werten, ebenso die hohe Automatisierung durch Anwendungsgenerierung, die allerdings nicht immer passgenaue Ergebnisse liefert. Ein inhaltsbasiertes ★*Empfehlungssystem* für Komponenten und Anwendungen wurde vorgestellt. Durch den Abgleich mit Aufgaben wird die funktionale Relevanz von Ergebnissen sichergestellt. Auch die automatische Integration von Komponenten in Anwendungen wird unterstützt. Das Empfehlungssystem lässt jedoch deutliche Defizite erkennen: Es ist nicht hybrid, bietet keine durchgängige Unterstützung, kaum Kontextsensitivität und nutzt kein Nutzerfeedback. Eine ★*Unterstützung bei Nutzen und Verstehen* ist nicht vorhanden. Lediglich die Zuordnung von Komponenten zu Aufgaben zur Kompositionszeit kann genannt werden.

Das **VizBoard** [Voi14] stellt eine Plattform zur Informationsvisualisierung auf Basis von semantischen Technologien und kompositen Webanwendungen dar. Rückgrat des semantikgestützten Ansatzes ist eine Ontologie wesentlicher Konzepte im Bereich der Informationsvisualisierung. Weiterhin wird unterstellt, dass Komponenten deklarativ und semantisch beschrieben werden, wobei auf das Vokabular der Ontologie zur Annotation

zurückgegriffen wird. Gemäß dem Vorgehensmodell laden Nutzer zuerst Datensätze in das System. Danach erfolgt eine Datenvorauswahl unter Zuhilfenahme eines Ontologie-Browsers, welcher dem Nutzer Zugang zum Vokabular gewährt und ihm beispielsweise bei der Identifikation potentiell interessanter Konzepte assistiert. In einem Facettenbrowser, der im Detail in Abschnitt 3.3 behandelt wird, spezifizieren Nutzer ihre Anforderungen anhand verschiedener Facetten, die zum Beispiel auf darzustellenden Daten, durchführbaren Aktionen und visuellem Detailgrad beruhen. Auf Basis eines Empfehlungsansatzes (Details siehe Abschnitt 3.2) präsentiert das System anschließend passende Komponenten. Diese werden im Rahmen existierender Mashup-Plattformen ausgeführt, wobei Nutzer abhängig von der Plattform Einflussmöglichkeiten auf die Konfiguration und den Austausch von Komponenten erhalten. Als einer der wenigen Vertreter bietet VizBoard Unterstützung beim Nutzen und Verstehen eines Mashups. Dazu gehören domänenspezifische Ansätze, wie Kommentare zu Datenpunkten und das Erklären von Fachbegriffen. Generischer zeigt sich eine Auflistung der Fähigkeiten von Komponenten und deren Manifestation im UI. Zudem werden Comics statisch generiert, die für Komponentenfähigkeiten notwendige Interaktionsmöglichkeiten von Nutzern mit bestimmten UI-Elementen anzeigen, siehe Abbildung 3.5 links für ein Beispiel. Weiterhin wurde eine Ansicht vorgesehen, welche die Komponenten-UIs überlagert und Datenbeziehungen von Komponenten anhand von Pfeilen visualisiert (rechter Teil der Abbildung). **Fazit:** Die ★*Entwicklungsmethodik* ist iterativ und für eine ähnliche Zielgruppe. Es zeigen sich jedoch Defizite hinsichtlich Generizität, aufgrund des Fokus auf die InfoVis-Domäne, und dahingehend, dass die mangelnde Werkzeugintegration Brüche in der Arbeitsumgebung verursacht. Der ★*Abstraktionsgrad* wird den meisten Anforderungen gerecht. Jedoch werden WYSIWYG-Prinzipien nur limitiert eingesetzt. Die ★*Unterstützung bei Einstieg und Suche* beschränkt sich auf den Facettenbrowser, der gute Ansätze liefert, jedoch nicht allen Kriterien genügt. ★*Unterstützung bei der Komposition* erfahren Nutzer durch Fehlervermeidung, und ihnen stehen adäquate Einflussmöglichkeiten zur Verfügung. Es mangelt hingegen an Automatisierung für beliebig komplexe Kompositionsfragmente sowie an Datenmediation. Zudem liegt die Metapher des Verdrahtens zugrunde. Das ★*Empfehlungssystem* liefert brauchbare Ansätze bezüglich der Modellierung und der Nutzung von Feedback, weist allerdings deutliche Defizite auf, siehe Kapitel 3.2. Die ★*Unterstützung bei Nutzen und Verstehen* erfolgt durch das Erklären der Bedienung von Komponenten durch Comics und durch das Darstellen der Datenbeziehung von Komponenten. Dieser Ansatz erfüllt somit die Forderung nach animierten und textuell erläuterten Anleitungen. Die Comics stellen allerdings eine Entfernung vom Live-UI dar, sind weder interaktiv noch evaluiert und bieten nur Unterstützung für einzelne Verknüpfungen von Komponenten. Zudem fehlen Mittel für die Erzeugung von Aufmerksamkeit für den Datenaustausch zwischen Komponenten zur Laufzeit.

Die Mashup-Plattform **NaturalMash** [AP12; AP14] verfolgt einen ähnlichen Live-Entwicklungsansatz wie OMELETTE. Wie Abbildung 3.6 verdeutlicht, wird das aktuelle Mashup in einer Kompositionsfläche, dem *Visual Field*, angezeigt und kann durch den Nutzer verwendet werden. Um dieses herum sind verschiedene Werkzeuge zum Anpassen der Anwendung angesiedelt. Hierzu gehört eine durchsuchbare Auflistung verfügbarer Komponenten (*Ingredients Toolbar*, links) und ein *Ingredient Dock*, welches aktuell im Mashup vorhandene Komponenten repräsentiert. Als Alleinstellungsmerkmal gilt das *Text Field* unterhalb des *Visual Field*. In diesem kann der Nutzer in restriktierter natürlicher Sprache sowohl nach Komponenten suchen als auch die Kompositionslogik durch Wenn-

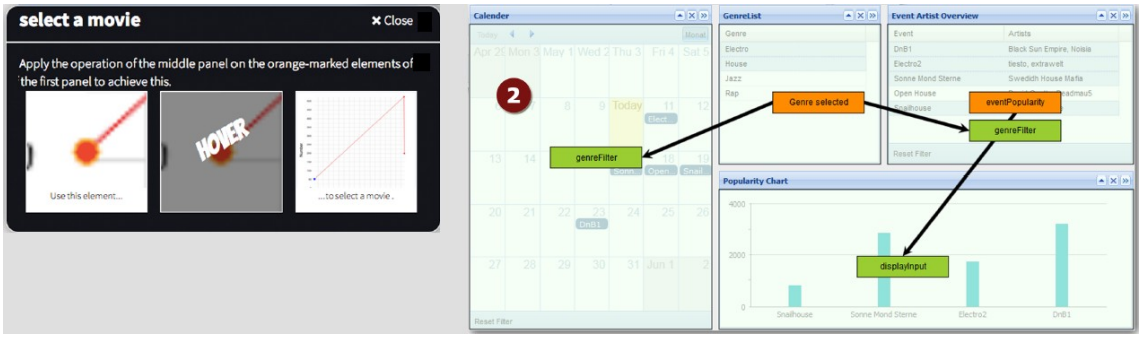


Abbildung 3.5: Unterstützung bei Nutzung und Verstehen in VizBoard [Voi14]

dann-Sätze definieren. Auto-Completion, Hervorhebung von Datenobjekten sowie Fehlern, z. B. keine verfügbaren Komponenten, helfen Nutzern dabei. Zum Beispiel definiert `when a slide is selected, find youtube videos about slide title` eine Verknüpfung zwischen den zwei in Abbildung 3.6 sichtbaren Komponenten. Die Hauptbestandteile des UI sind miteinander synchronisiert. Zum Beispiel beeinflusst eine Änderung im *Text Field* das *Visual Field*. Weiterhin führt etwa die Auswahl einer Komponente im *Visual Field* zum Hervorheben des zugehörigen Texts im *Text Field*. Interaktionen in Komponenten, die durch Events nach außen sichtbar propagiert werden, führen zur Anreicherung des *Text Field* mit einem zu komplettierenden Kausalsatz. Weiterhin ist vorgesehen, dass bei Auswahl von Textfragmenten in der Kompositionslogik die entsprechenden Komponenten und UI-Elemente hervorgehoben werden. Konzeptionelle Details hierzu bleiben jedoch unklar, z. B. auf welcher Modellbasis diese Relation hergestellt wird. **Fazit:** Die ★*Entwicklungsmethodik* wird als gut eingeschätzt, da sie iterativ, generisch und für eine ähnliche Zielgruppe ist. Lediglich hinsichtlich einer durchgängigen Assistenz sind Abstriche vorzunehmen. Ebenso kann der ★*Abstraktionsgrad* den Anforderungen gerecht werden. Die ★*Unterstützung bei Einstieg und Suche* umfasst eine Palette sowie die textuelle Anforderungseingabe, die allerdings kleinteilig ist und nah an Komponentenfunktionen operiert. Bezüglich ★*Unterstützung bei der Komposition* ist der natürlichsprachliche Ansatz als gut zu werten und bietet adäquate Einflussmöglichkeiten. Es mangelt hingegen an Automatisierung für beliebig komplexe Kompositionsfragmente, Bereitstellung von Kompositionswissen sowie an Datenmediation und Fehler werden nur hervorgehoben. Ein ★*Empfehlungssystem* sieht NaturalMash nicht vor. Zur ★*Unterstützung bei Nutzen und Verstehen* kann der Kompositionstext dienen, sodass die Anforderung nach textuellen Erklärungen erfüllt ist. Die Gesamtfunktion wird wegen der Kleinteiligkeit allerdings nur umständlich erfassbar. Die Synchronisation zwischen *Text Field* und Kompositionsfläche vermittelt dem Nutzer Zuständigkeiten von Komponenten, jedoch werden notwendige Interaktionsschritte weder animiert noch erklärt. Letztlich fehlt eine Visualisierung von Datenaustausch zwischen Komponenten zur Laufzeit. Zudem bleibt offen, wie der Ansatz mit komplizierten Kommunikationsbeziehungen umgeht.

Der **MashupEditor** [Ghi+16] entspricht einem Werkzeug für das graphische EUD von Mashups. Einheiten zur Komposition sind hierbei Fragmente von Webseiten, die der Nutzer ad-hoc markiert und somit zur Nutzung in den MashupEditor überträgt. Technologisch basiert der Ansatz auf einem Proxy-Server, der existierende Webseiten vorverarbeitet und dabei verwendbare Fragmente annotiert sowie notwendigen Quellcode injiziert. In derart präparierten Webseiten werden dem Nutzer verfügbare Fragmente

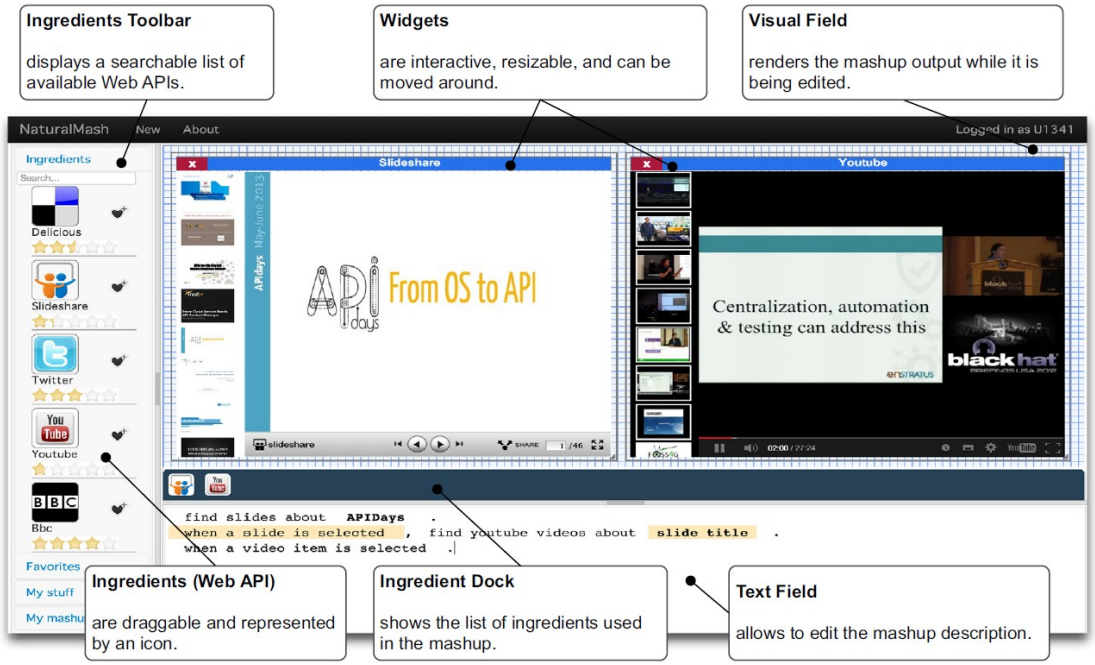


Abbildung 3.6: Grafische Benutzerschnittstelle von NaturalMash [AP14]

hervorgehoben, sodass er diese selektieren kann. Im MashupEditor werden diese Widgets im originalen Erscheinungsbild wiedergegeben und können vom Nutzer miteinander verknüpft werden. Die grundlegende Kompositionsmetapher ist hierbei *Kopieren-und-Einfügen*. Nutzer zeigen dem System dabei die gewünschten Korrespondenzen von Datenwerten in verschiedenen Komponenten. Dazu startet der Nutzer einen Aufzeichnungsmodus, kopiert einen Wert aus einer Komponente, fügt diesen andernorts ein und beendet den Modus. Daraufhin ist die Verbindung etabliert. Der Ansatz unterscheidet zwei Verknüpfungsarten: zwischen zwei Eingabedatenfeldern zum Zwecke der Synchronisation und zwischen einem Ausgabe- und einem Eingabefeld, was sequentielle Abläufe orchestriert. Vorhandene Verknüpfungen können vom Nutzer inspiziert werden. Dabei werden involvierte UI-Elemente hervorgehoben und durch Pfeile verbunden, vergleiche Abbildung 3.7. **Fazit:** Es kommt eine ★*Entwicklungsmethodik* für eine ähnliche Zielgruppe zum Einsatz. Unklar ist jedoch, inwiefern diese Iterationen unterstützt. Auch liegen Annahmen an die Webseiten zugrunde, sodass die Generizität eingeschränkt scheint. Durchgängige Assistenz fehlt. Der ★*Abstraktionsgrad* erfüllt die Anforderungen. Die ★*Unterstützung bei Einstieg und Suche* existiert nur in Form der Auswahl von Webseiten-Bestandteilen. Darüber hinausgehende Konzepte lässt der Ansatz missen. Im Rahmen der ★*Unterstützung bei der Komposition* ist vor allem die Kompositionsmetapher positiv hervorzuheben. Ansonsten zeigen sich Defizite bei den Einflussmöglichkeiten und der Automatisierung für beliebig komplexe Kompositionsfragmente. Es fehlt an der Bereitstellung von Kompositionswissen, an Datenmediation sowie an Ansätzen zur Fehlervermeidung. Ein ★*Empfehlungssystem* sieht der MashupEditor nicht vor. Bezugnehmend auf die ★*Unterstützung bei Nutzen und Verstehen* bietet lediglich die pfeilbasierte Visualisierung von Datenbeziehungen einen geeigneten Ansatz. Dabei wird ein direkter Bezug zum UI hergestellt. Es fehlen jedoch animierte und textuell untermauerte Erklärungen. Darüber hinaus sind kompliziertere Kommunikationsbeziehungen nicht konzeptionell durchdacht. Schließlich fehlt ein Ansatz, der Nutzer auf IWC aufmerksam macht.

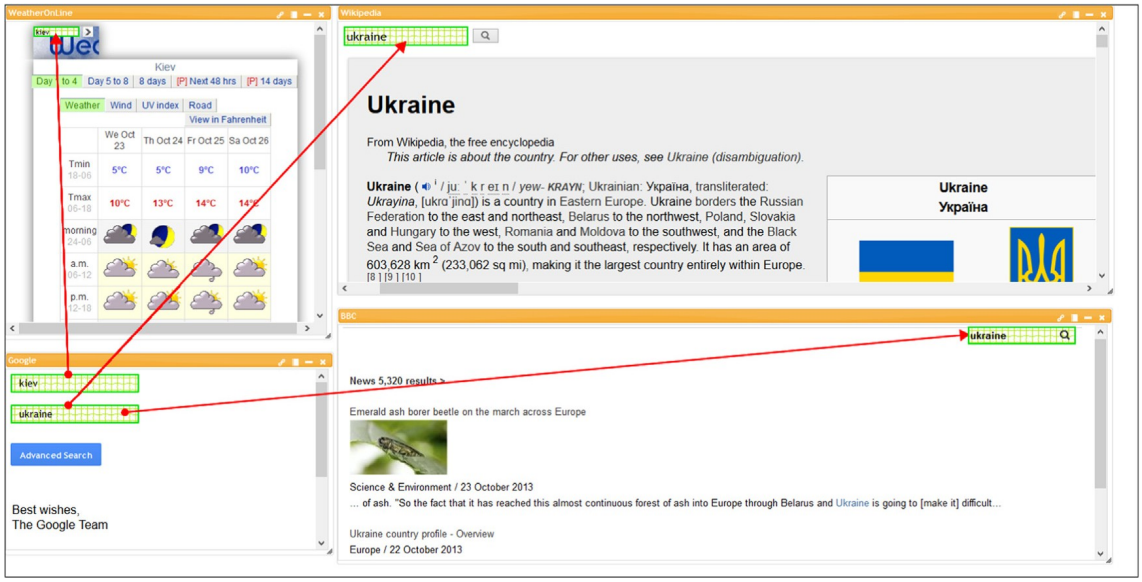


Abbildung 3.7: MashupEditor: Darstellen von Relationen zwischen Komponenten [Ghi+16]

Die Mashup-Plattform **PEUDOM** [Pic13] legt starken Fokus auf Mechanismen zur Datenintegration. Dies betrifft insbesondere den *Component Editor*, der es Nutzern erlaubt, eigenständig Komponenten zu entwickeln. Auf Grundlage eines *Visual Integration-Model* wird dem Nutzer ermöglicht, mehrere Datenquellen zu integrieren, datenbankorientierte Operationen zur Mediation anzuwenden und Abbildungen zwischen Datenschemata und visueller Darstellung in UI-Templates per Drag-and-Drop zu definieren. PEUDOM offeriert daneben eine Laufzeitumgebung zur Live-Entwicklung von Mashups, die Werkzeuge zur Kollaboration und für das EUD bereitstellt. Auf einer Canvas ist die Anwendung zu sehen und kann assistiert manipuliert werden, siehe linker Teil von Abbildung 3.8. Weiterhin existiert eine Palette von Komponenten. Letztere verfügen über parametrisierte Events und Operationen als Schnittstellen und können auf diese Weise miteinander verknüpft werden. Die Annotation von Komponenten und Mashups mit Qualitätseigenschaften und die Nutzung dieser im Rahmen eines Empfehlungssystems stellt ein Alleinstellungsmerkmal des Ansatzes dar. Das hybride Empfehlungssystem schlägt dem Nutzer auf Nachfrage Kompositionsschritte vor, siehe rechts oben in Abbildung 3.8. Dies umfasst empfohlene erweiternde Komponenten sowie Alternativkomponenten. Für weitere Details sei auf Abschnitt 3.2 verwiesen. Weiterhin kann der Nutzer mit Hilfe des *Binding Dialog*, siehe Abbildung 3.8 rechts unten, Komponenten verknüpfen. Dazu wählt er Komponenten und deren Events und Operationen aus. Dabei assistiert ihm das System, indem eine Farbkodierung von Einträgen entsprechend ihrer Kompatibilität stattfindet, kurze Sätze gebildet werden, um den kausalen Zusammenhang zu beschreiben, und die zugehörigen Komponenten in der Canvas hervorgehoben werden. Eine schematische Darstellung der Komposition verschafft einen Überblick der Komponenten und der Kommunikationsbeziehungen. **Fazit:** Die ★*Entwicklungsmethodik* wird dem Großteil der Anforderungen gerecht, jedoch ist die vorgesehene Assistenz auf die Kompositionsphase begrenzt. PEUDOM bietet einen guten ★*Abstraktionsgrad*, etwa durch WYSIWYG, allerdings wird der Nutzer insbesondere bei der Binding-Erstellung mit technischer Terminologie konfrontiert. Auffällig ist ein Mangel an ★*Unterstützung bei Einstieg und Suche*, da letztlich nur eine Palette verfügbarer Komponenten auftaucht. Der Nutzer erfährt durchschnittliche ★*Unterstützung*

bei der Komposition. Zwar sind die Einflussmöglichkeiten adäquat und Fehler werden durch Kompatibilitätsprüfungen vermieden. Jedoch entspricht die Kompositonsmetapher dem »Verdrahten« und es mangelt an Datenmediation sowie automatisiertem Hinzufügen komplexer Kompositionsfragmente. Übertragbare Konzepte liefert das hybride ★*Empfehlungssystem*, vor allem die Nutzung von Qualitätsmetriken. Es weist jedoch im Detail zahlreiche Schwächen auf, was in Abschnitt 3.2 näher ausgeführt wird. Bis auf die Darstellung von Kausalsätzen bei der Binding-Erstellung und den Kompositionsgraph existiert keine ★*Unterstützung bei Nutzen und Verstehen*. Die Techniken des *Component Editor* finden allerdings bei der Mashup-Komposition keine Anwendung, sodass der Bezug zum UI fehlt. Darüber hinaus wird der zur Laufzeit auftretende Datenfluss nicht veranschaulicht. Damit ist dieses Kriterium nur unbefriedigend erfüllt.

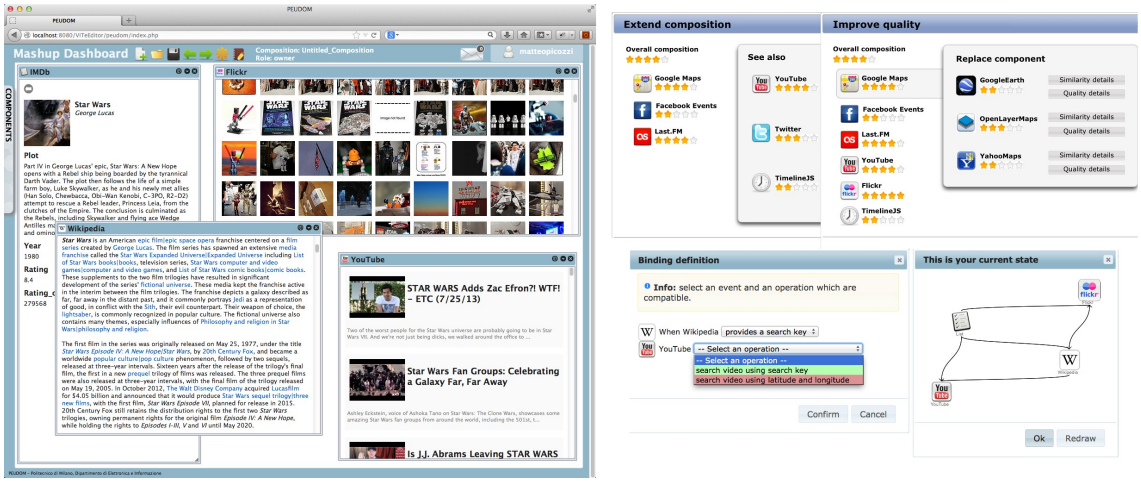


Abbildung 3.8: Die Kompositionsplattform PEUDOM [Pic13]

Die Mashup-Plattform **FAST-Wirecloud** [Liz+08; Liz+11; Liz+14; Liz+16] zielt auf das EUD durch technisch nicht versierte Nutzer, ähnlich Nicht-Programmierern, ab. Die konzeptionelle Grundlage des Ansatzes stellen Blackbox-Komponenten dar, deren Ein- und Ausgaben sowie Vorbedingungen und Effekte semantisch annotiert sind. Die Plattform kennt verschiedene Arten von Komponenten zur Erstellung von Mashups: *Widgets* und *Operators*. Erstere können von Endnutzern in einem dedizierten Autorenwerkzeug aus Teilelementen wie *Forms* und *Resources* komponiert werden. Die Mashup-Erstellung erfolgt auf Ebene von *Screens* (Widgets mit Layouts und Navigation) sowie auf Ebene der Kommunikationsbeziehungen. Als für diese Arbeit besonders relevant erweisen sich die entsprechenden Werkzeuge. Das *Workspace Editing Component* dient dem visuellen Entwurf von Mashups. Dazu können Nutzer nach Widgets und Operators suchen sowie diese aus einem Katalog auswählen und dem Arbeitsbereich hinzufügen. Die Verknüpfung von Widgets findet in einem extra Werkzeug, dem *Wiring Editing Component*, statt. Wie Abbildung 3.9 verdeutlicht, kommt dabei die Kompositionsmetapher des »Verdrahtens« zum Einsatz. Ein- und Ausgabeports von Widgets und Operators können durch Kommunikationskanäle miteinander verbunden werden, wobei Nutzer durch semantikbasierte Kompatibilitätsprüfung und entsprechende Hervorhebungen unterstützt werden. Hierbei werden Granularitätsunterschiede unterstützt, das heißt, aus einem Konzept können Teilkonzepte projiziert werden, was in Abbildung 3.9 angedeutet ist. Eine weitere Assistenzfunktion bietet der Ansatz in Form von *Behaviors*. Diese fassen einen Ausschnitt

des Kompositionsmodells zusammen, der eine gewisse Funktionalität erbringt. Behaviors dienen im *Wiring Editing Component* zur Partitionierung der Kompositionslogik, um die Funktionsweise der Anwendung auf abstrakter Ebene besser nachvollziehen zu können. Einzelne Behaviors können ein- und ausgeblendet werden, um die Übersichtlichkeit zu erhöhen. Zusätzliche Hilfe erhalten Nutzer bei der Parametrisierung und Adaption von Komponenten durch textuelle, manuell vom Entwickler bereitgestellte Beschreibungen der Auswirkungen von Konfigurationsmöglichkeiten. **Fazit:** Die ★*Entwicklungsmethodik* ist iterativ, generisch und für eine passende Zielgruppe ausgelegt. Abzüge ergeben sich jedoch aus nicht durchgehender Assistenz sowie aus der Trennung von Nutzung und Entwicklung bei der Erstellung von Verknüpfungen. Bezüglich ★*Abstraktionsgrad* zeigen sich Defizite, da Nutzer mit technischen Begriffen und Kompositionsdetails konfrontiert werden und da WYSIWYG nur eingeschränkt zum Einsatz kommt. ★*Unterstützung bei Einstieg und Suche* bietet FAST-WireCloud nur begrenzt. Neben einer Palette und einer Stichwortsuche existieren weder Ansätze zur Anforderungseingabe noch zur explorativen Suche. ★*Unterstützung bei der Komposition* umfasst adäquate Einflussmöglichkeiten, semantische Typprüfung zur Fehlervermeidung und eingeschränkte Bereitstellung von Kompositionswissen (*Behaviors*). Als Kompositionsmetapher wird allerdings »Verdrahten« genutzt und Automatisierung wird nicht für beliebige Kompositionsfragmente angeboten. Datenmediation wird limitiert unterstützt, vergleiche Kapitel 3.4. Zwar wird angedeutet, dass Nutzer Empfehlungen erhalten, das Konzept eines ★*Empfehlungssystems* bleibt jedoch unbekannt. ★*Unterstützung bei Nutzen und Verstehen* wird lediglich eingeschränkt geboten. Die Kompositionslogik wird visuell präsentiert. Behaviors bieten Potenzial, werden jedoch ausschließlich in der Kompositionsphase verwendet. Weder ein Aufmerksammachen auf auftretenden Datenaustausch, noch eine genaue Funktionalitätsbeschreibung, noch Bedienungsanleitungen sind vorgesehen.

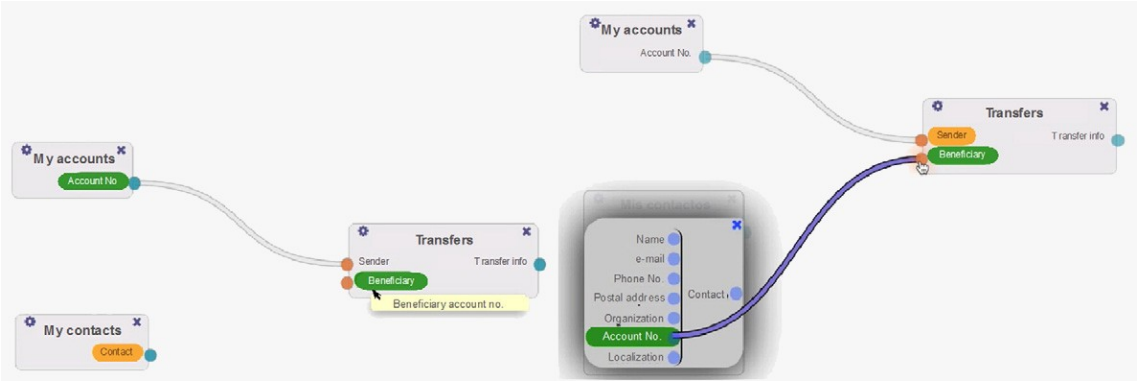


Abbildung 3.9: Semantik-gestützte Komposition in FAST-Wirecloud [Liz+16]

Die **Linked Widgets Platform** [Tri+14; Tri+15] ermöglicht das interaktive, visuelle und kollaborative Verknüpfen von Widgets zu Mashups und orientiert sich an Prinzipien von *Linked Open Data*. Die Grundlage hierfür ist das *Linked Widget Model*, welches eine semantikgestützte Beschreibung und Komposition von Komponenten erlaubt, indem Ein- und Ausgaben semantisch annotiert sind. Weiterhin arbeiten Komponenten mit Daten auf Basis von Resource Description Framework (RDF). Nutzer wählen zunächst Widgets aus, platzieren diese auf der Kompositionsfläche und sind anschließend gefordert, Verknüpfungen vorzunehmen. Bei diesem Prozess assistiert die Plattform auf verschiedene

Weisen. Es existiert eine semantische Widget-Suche. Bei dieser spezifiziert der Nutzer eine Art Schablone für eine benötigte semantische Komponentenschnittstelle und wird durch Beispiele und tolerante, Äquivalenzbeziehungen ausnutzende Algorithmen unterstützt. Weiterhin stehen Kollektionen von Mashups und Widgets für den Direkteinstieg zur Verfügung. Ein *interaktiver, tag-basierter Agent* adressiert als Zielgruppe »Novizen« mit wenig oder keiner Erfahrung in der Mashup-Erstellung. Im Agenten gilt es, die Zielstellung des Mashups textuell zu definieren. Die Eingabe erfolgt entweder nach einem vorgegebenen Schema oder als Freitext. In letzterem Fall obliegt es dem Nutzer, den Text semantisch zu annotieren, wobei er vom System durch Begriffsvorschläge unterstützt wird. Das Annotieren erfolgt mittels Konzepten und Individuen aus Ontologien. Auf Knopfdruck werden schließlich durch semantischen Abgleich passende Widgets identifiziert und automatisch komponiert. Nachdem der Nutzer Komponenten oder Anwendungen durch zuvor genannte Assistenzmechanismen ausgewählt hat, findet ein Wechsel in die Kompositionsansicht statt. Diese ist in Abbildung 3.10 veranschaulicht. Der Mechanismus *Auto-Matching* steht als Kompositionshilfe bereit und berechnet mögliche Kopplungen ausgehend von einer Komponente, siehe Abbildung unten links. Zudem verhindert *Terminal Matching*, dass Nutzer bei der manuellen Verdrahtung inkompatible Verbindungen etablieren. Die Ausführung von Widgets erfolgt über Schaltflächen im unteren Bereich des Komponentenrahmens. **Fazit:** Die ★*Entwicklungsmethodik* ist generisch und zugänglich für eine passende Zielgruppe. Inwiefern Iterationen bei der Nutzung des Agenten unterstützt werden, bleibt unklar. Die Assistenz ist nicht durchgehend und Entwurf sowie Nutzung sind getrennt. Der ★*Abstraktionsgrad* wird nicht allen Anforderungen gerecht. Zwar finden WYSIWYG-Prinzipien Verwendung, Nutzer werden jedoch vor allem in der Kompositionsansicht mit technischer Terminologie und technischen Konzepten konfrontiert. Die ★*Unterstützung bei Einstieg und Suche* zeigt gute Ansätze, da verschiedene Einstiegspunkte angeboten werden. Der Agent hat allerdings deutliche Defizite, wie eine fehlende Ergebnisvorschau und stark begrenzte Nutzerführung. Außerdem mangelt es an explorativen Konzepten. Der Nutzer erfährt durchschnittliche ★*Unterstützung bei der Komposition*. Der Ansatz erlaubt adäquate Einflussmöglichkeiten und bietet Fehlervermeidung sowie gute Teilautomatisierung durch Generierung. Defizite treten durch fehlende Datenmediation, die genutzte Kompositionsmetapher des »Verdrahtens« und begrenzte Bereitstellung von Kompositionswissen auf. Das ★*Empfehlungssystem* basiert auf semantischem Matching und lässt unter anderem Kontextsensitivität und eine Einbeziehung von Nutzerfeedback missen. ★*Unterstützung bei Nutzen und Verstehen* wird kaum geboten. Außer der Anzeige von Kommunikationskanälen existieren keinerlei Ansätze zur Erklärung der Funktionsweise und Bedienung. Awareness-Mechanismen für aktiven Datenaustausch sind beschränkt auf die Indikation der Aktivität von Widgets.

Der Mashup-Plattform **iMashup** [Liu+10; Liu+11; Ma+13; Liu+14; Liu+15] liegt ein hierarchisches *tag*-basiertes Modell der Semantik von Komponenten, Kompositionen und des Nutzerziels zugrunde. Wesentlicher Architekturbestandteil ist das Kompositionswerkzeug, das Abbildung 3.11 illustriert. Das Konzept ist stark an **MARIO**, den Ansatz von Riabov et al. [Ria+08], angelehnt und erweitert diesen. Daher wird MARIO nicht näher vorgestellt. Eine wesentliche Rolle spielt die *Tag Cloud*, welche Nutzern initial präsentiert wird. Sie visualisiert relevante *Tags* der vorhandenen Komponenten (produzierbare oder konsumierbare Daten). *Feature Tags* stellen Cluster dar, die Tags semantisch zusammenfassen, zum Beispiel »Weather« sowie »News«, und werden generiert. Nutzer können Tags aus der *Tag Cloud* auswählen, woraufhin diese dem aktuellen *Goal* des Nutzers

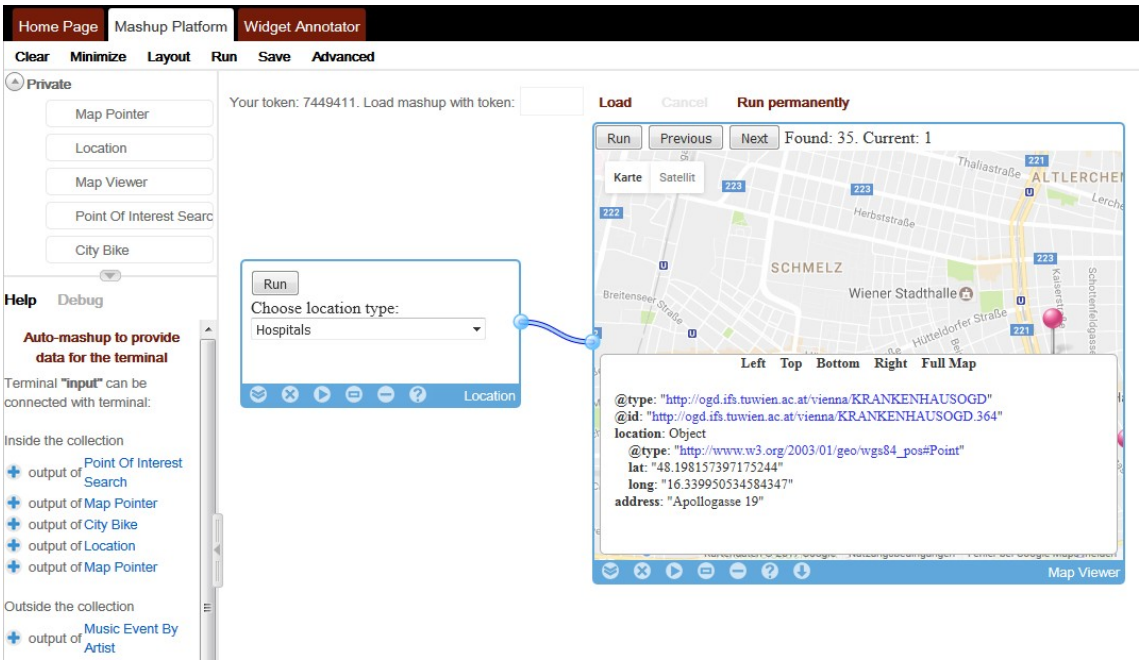


Abbildung 3.10: Linked Widgets Platform (selbst erstellter Screenshot)

hinzugefügt werden. Alternativ besteht die Möglichkeit, mittels *Tag Search* schlüsselwortbasiert nach Tags zu suchen und diese hinzuzufügen. Entsprechend der Nutzerziele werden passende Mashups generiert und nach Relevanz sortiert angezeigt, wobei das beste Ergebnis live angezeigt wird (Bereich *Visualization* in der Abbildung). Die Generierung basiert auf Planungsalgorithmen und einer formalen Tag-Semantik. Durch Hinzufügen und Entfernen von Tags verfeinert der Nutzer iterativ seine Anfrage. So hervorgerufene Änderungen der *Goals* führen zur Anpassung sowohl der *Tag Cloud* als auch der Auflistung passender Mashups. Weiterhin steht es dem Nutzer frei, direkt eine Schlüsselwortsuche nach Komponenten durchzuführen oder aus dem *Services Catalog* auszuwählen. Schließlich wird der Nutzer durch Empfehlungen unterstützt, siehe Abbildung 3.11 unten. Ein datenflussorientierter Kompositionsgraph veranschaulicht die aktuelle Anwendung. **Fazit:** Die ★*Entwicklungsmethodik* ist iterativ, generisch und Entwurf sowie Nutzung sind eng verzahnt. Allerdings ist die Assistenz in der Nutzungsphase stark limitiert und die genaue Charakterisierung der Zielgruppe bleibt offen. Durch Tags und WYSIWYG-Prinzipien sind die Anforderungen an den ★*Abstraktionsgrad* erfüllt. Der Ansatz offeriert mit der Palette und dem tag-basierten Werkzeug mehrere Einstiegspunkte, sodass die ★*Unterstützung bei Einstieg und Suche* viele Kriterien abdeckt. Es fehlt jedoch sowohl an Konzepten für einen explorativen als auch einen direkten Einstieg. Zur ★*Unterstützung bei der Komposition* bietet iMashup einen Planungsalgorithmus, der Fehler vermeidet und Teilautomatisierung liefert. Allerdings beschränkt er die Einflussmöglichkeiten. Zudem fehlt es an semantischer Datenmediation. Das hybride ★*Empfehlungssystem* unterbreitet Vorschläge zu komplexen Kompositionsfragmenten und baut diese automatisch in die Anwendung ein. Wie in Abbildung A.5 veranschaulicht, zeigen sich jedoch Defizite zum Beispiel bei der Durchgängigkeit, der Kontextsensitivität und der nachvollziehbaren Darstellung von Vorschlägen. Für die ★*Unterstützung bei Nutzen und Verstehen* steht ein generierter Text sowie ein Kompositionsgraph bereit. Defizite sind darin begründet, dass die Beschreibung

der Funktion kleinteilig und wenig fachlich ist, was das Erfassen der Gesamtfunktion erschwert. Zudem fehlt es an Bezug zum UI sowie an Ansätzen, die Nutzer auf den Datenaustausch zur Laufzeit aufmerksam machen oder die Bedienung erläutern.

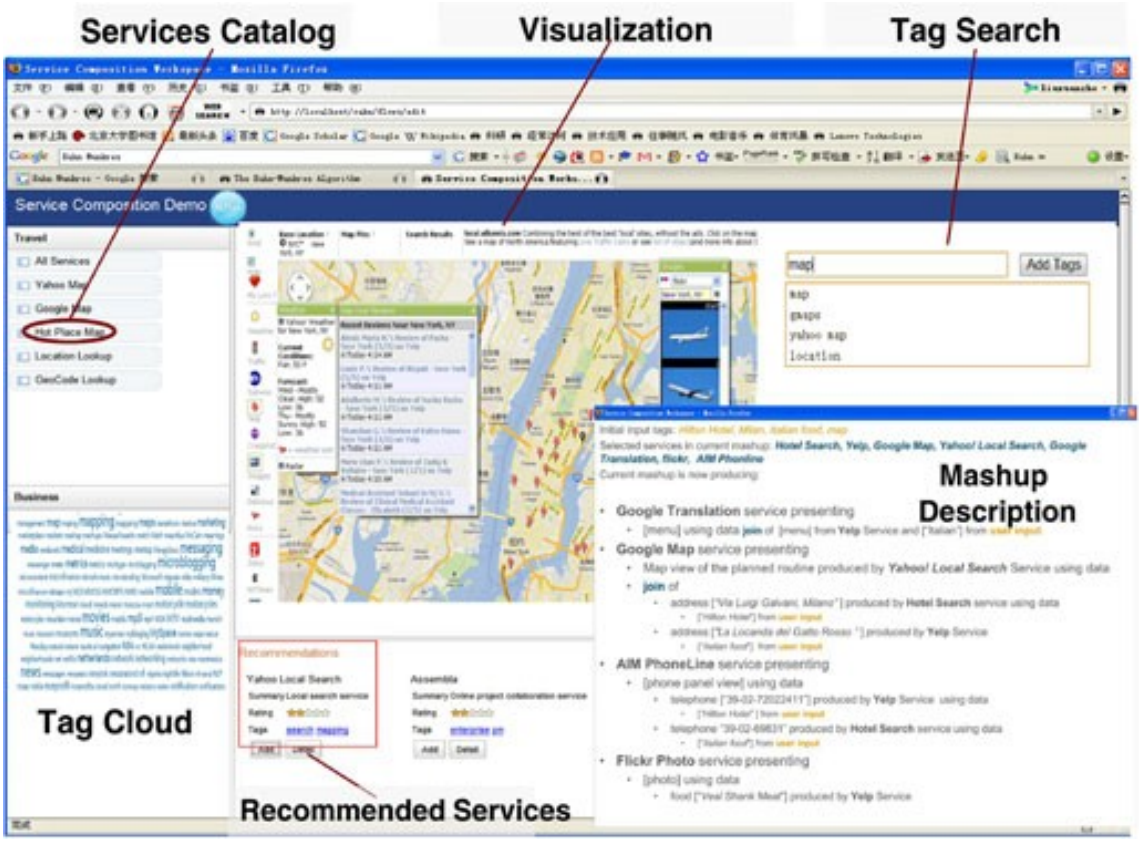


Abbildung 3.11: Die Benutzerschnittstelle des iMashup-Ansatzes [Liu+15]

Mit **sMash** [Lu+09; Che+09] wurde eine Mashup-Plattform vorgestellt, die auf semantisch annotierten Komponenten beruht. Als Alleinstellungsmerkmal des Ansatzes erzeugen Nutzer ein Mashup durch die Navigation in einem Graphen. Dessen Knoten entsprechen Web-APIs und seine Kanten repräsentieren die Kombinierbarkeit der Komponenten. Ob APIs verknüpfbar sind, wird auf Basis von semantischer Typisierung sowie Heuristiken, welche die funktionale Sinnhaftigkeit reflektieren, festgestellt. Nutzer bekommen initial den Graphen an Web-APIs präsentiert, vergleiche Abbildung 3.12. In diesem können sie direkt navigieren und Knoten selektieren. Weiterhin werden Empfehlungen aufgelistet (Abbildung rechts) und es besteht die Möglichkeit, per Schlüsselwörtern nach Komponenten zu suchen, deren Knoten daraufhin fokussiert werden. In Abhängigkeit von den ausgewählten APIs werden häufig mit diesen zusammen genutzte Komponenten näher herangerückt, und die Knotenmenge auf die verknüpfbaren limitiert (wobei diese Filterung explizit deaktivierbar ist). Im Graphen können Informationen zu APIs in Erfahrung gebracht werden, etwa hinsichtlich deren Kombinierbarkeit, und Details zur API erscheinen in einem Dialogfenster. Zudem werden sämtliche ausgewählte Kanten hervorgehoben, sodass Datenflüsse ersichtlich sind. Weiterhin wird wenn möglich versucht, selektierte Komponenten automatisch zu verknüpfen. Unterstützung erhält der Nutzer durch drei Arten von Empfehlungen, die auf das Hinzufügen von Knoten und Kanten abzielen. Die Visualisierung von Vorschlägen

erfolgt listenartig und ist technisch geprägt. Das Empfehlungssystem stützt sich auf existierende Mashups, um statistische Aussagen zu Pfaden zu treffen, sowie auf semantisches Schlussfolgern, um zusätzlich weniger populäre Verknüpfungen zu bestimmen. **Fazit:** Der Ansatz bietet eine iterative, generische ★*Entwicklungsmethodik* für »Novizen«, und somit für eine passende Zielgruppe. Allerdings erfolgt eine Trennung von Entwurf und Nutzung und es fehlt an Assistenz bei Nutzen und Testen der Anwendung. Deutliche Abzüge ergeben sich beim ★*Abstraktionsgrad*. Es treten häufig technische Terminologie sowie Konzepte auf und es mangelt an WYSIWYG-Prinzipien. Für die ★*Unterstützung bei Einstieg und Suche* ist lediglich eine Schlüsselwortsuche angedacht. Der Graph bietet gute Ansätze zum explorativen Einstieg, jedoch fehlt es an fachlicher Abstraktion, sodass der Nutzer zum Beispiel Komponentennamen kennen muss. Eine Eingabe fachlicher Anforderungen ist nicht vorgesehen. Die ★*Unterstützung bei der Komposition* zeichnet sich durch eine gute Kompositionsmetapher aus: Mittels Navigation im Graphen entsteht eine Mashup-Komposition, was abstrahiertem »Verdrahten« entspricht. Es gibt adäquate Einflussmöglichkeiten, Fehler werden durch Typprüfung vermieden und es findet Teilautomatisierung statt. Datenmediation fehlt jedoch. Das ★*Empfehlungssystem* ist hybrid und bezieht den Kompositionskontext ein. Bemängelt werden muss die fehlende Durchgängigkeit sowie die technische Darstellung von Empfehlungen. Auch werden keine komplexen Kompositionsfragmente vorgeschlagen. Zudem wird nur implizites Feedback erfasst. Hinsichtlich ★*Unterstützung bei Nutzen und Verstehen* bietet der Graph die Möglichkeit bestehende Verbindungen nachzuvollziehen. Darüber hinausgehende Konzepte zur Erklärung der Funktionsweise, insbesondere zur Laufzeit, fehlen.

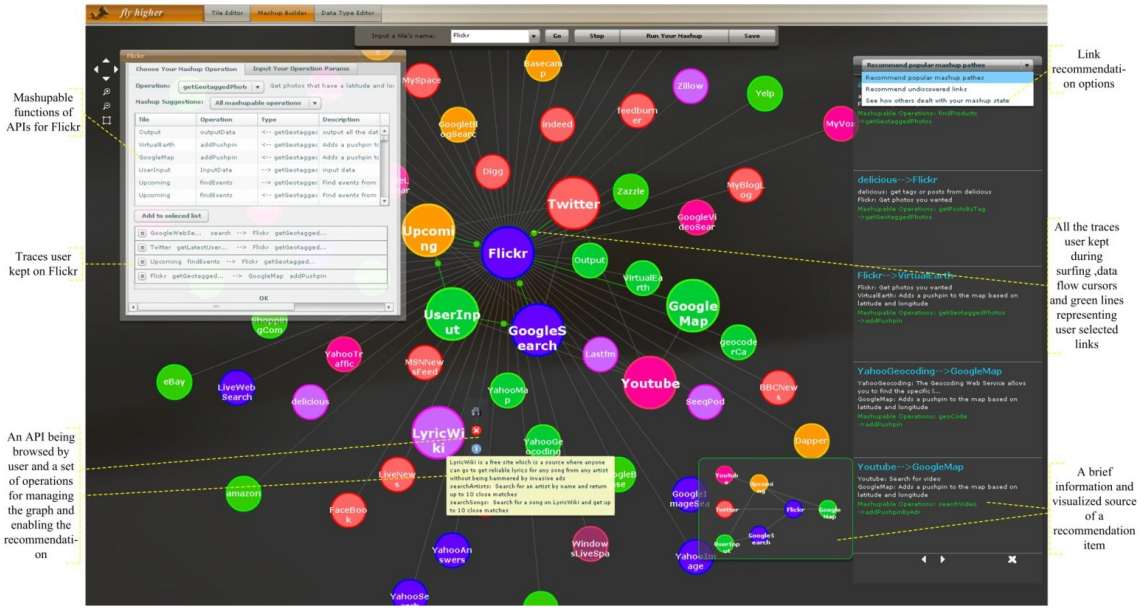


Abbildung 3.12: Benutzerschnittstelle der Mashup-Plattform sMash [Che+09]

3.1.3 Fazit

Zahlreiche Mashup-Plattformen haben sich herausgebildet und wurden anhand der Darstellungen in der jeweils zitierten Literatur untersucht. Nur in Ausnahmefällen wie der Linked

Widgets Platform konnte ein online verfügbarer Prototyp ausprobiert werden. Insgesamt handelt es sich bei sämtlichen Vertretern um Forschungsprojekte, deren Prototypen jedoch kaum eine praktische Bedeutung erlangen konnten.

Tabelle 3.13 zeigt eine genaue Aufschlüsselung der Untersuchungsergebnisse für die behandelten Kompositionsplattformen. Wie deutlich wird, kann keiner der Vertreter den Anforderungen in vollem Umfang gerecht werden. Vielversprechende Ansätze zeigen sich hinsichtlich der Kriterien ★*Entwicklungsmethodik* und ★*Abstraktionsgrad*. Für die übrigen Anforderungen sind punktuell brauchbare Lösungen vorzufinden, die allerdings deutliche Defizite besitzen. Auffällig ist ein Mangel an ★*Unterstützung bei Nutzen und Verstehen*. Wie [Spe06; BMC93] zeigen, gibt es verschiedenste Vorgehensweisen bei der Informationssuche. Diesem Umstand wird in aktuellen Vertretern zu wenig Rechnung getragen. Im weiteren Verlauf der Arbeit werden daher für spezielle Fragestellungen passende Lösungen in der Forschung untersucht. Schließlich gilt es, ein eigenes Konzept zu entwickeln, das die aufgedeckten Defizite aktueller Mashup-Plattformen adressiert.

	SOA4All	BOP	ServFace	OMELETTE	DEMISA	ResEvalMash	VizBoard	MashupEditor	SMASHAKER	RUBIK	PEUDOM	NaturalMash	Fast-Wirecloud	Linked Widgets Platform	iMashup	sMash
EM	o	+	o	++	+	-	+	+	o	+	++	++	++	+	++	+
AG	+	+	++	++	+	++	++	++	-	++	+	++	o	+	++	-
UE	-	-	-	o	-	-	-	--	-	-	-	o	o	+	+	-
UK	+	o	o	o	o	o	o	-	--	-	o	o	o	o	o	+
ES	--	--	--	+	-	--	o	--	+	-	+	--	--	--	o	o
UN	-	--	--	o	--	-	o	-	--	--	-	-	-	-	-	-

Legende:

EM	Entwicklungsmethodik
AG	Abstraktionsgrad
UE	Unterstützung bei Einstieg und Suche
UK	Unterstützung bei Komposition
ES	Empfehlungssystem
UN	Unterstützung bei Nutzen und Verstehen

--	0% - 19%
-	20% - 39 %
o	40% - 59%
+	60% - 79%
++	80% - 100%

Abbildung 3.13: Bewertungsergebnisse sämtlicher betrachteter Kompositionsplattformen

3.2 Empfehlungssysteme im Mashupkontext

Empfehlungssysteme dienen grundsätzlich dazu, Nutzern solche Items vorzuschlagen, die ihnen bisher unbekannt und höchstwahrscheinlich für sie relevant sind. Zielstellung ist demnach, zu prognostizieren, wie ein Nutzer ein Item bewerten würde. Ein klassisches Einsatzgebiet ist elektronischer Handel, wobei insbesondere wirtschaftliche Interessen der Betreiber im Vordergrund stehen. Abhängig von der Datenbasis, auf der ein Empfehlungssystem aufbaut, können **kollaboratives Filtern**, **inhaltsbasierte** und **hybride Ansätze** unterschieden werden [AT05]. Erstere basieren auf Nutzerprofilen, die Aufschluss über Nutzer-Item-Relationen geben, beispielsweise welche Items bereits gekauft oder bewertet wurden. Anhand dieser Profile werden Nutzer identifiziert, die dem aktuellen Nutzer ähnlich sind. Anschließend werden die Nutzer-Item-Relationen dieser Profile verrechnet, um eine Prognose für die Relevanz von Items zu bestimmen. Kollaboratives Filtern hat den Vorteil, dass es ohne inhaltliche Beschreibung von Items auskommen kann. Allerdings

treten Kaltstartprobleme auf, zum Beispiel im Fall eines bisher unbekannten Nutzers sowie bei neuen Items. Das Grundprinzip inhaltsbasierter Empfehlungssysteme ist die Ausnutzung von Ähnlichkeiten zwischen Items. Diese sind anhand von Metadaten inhaltlich beschrieben, zum Beispiel Autoren und Genres von Büchern. Ein Nutzerprofil umfasst wiederum Nutzer-Item-Relationen oder direkt einen Vektor von Interessen (aus Metadaten der Items). Beide Informationen werden anschließend kombiniert, um Items zu berechnen, die ähnlich zu den bisherigen Items des Nutzers sowie zu den abgeleiteten Interessen sind. Vorteilhaft ist, dass Kaltstarts ein geringeres Problem darstellen und höchstens bei neuen Nutzern auftreten. Es werden jedoch Metadaten für sämtliche Items benötigt. Zudem gilt es, Überspezialisierung entgegen zu wirken.

Empfehlungssysteme gehören auch in Kompositionsplattformen für EUD oft zur Ausstattung, vergleiche Tabelle 3.13, um Nutzern beispielsweise Kompositionsschritte, Komponenten oder Anwendung vorzuschlagen. Gemäß [Maa+11] lassen sich diese dahingehend unterscheiden, ob Vorschläge auf eine schrittweise oder vollständige Komplettierung einer Anwendung abzielen. Nachfolgend werden ausgewählte Vertreter aus den Bereichen Mashups und Webservices anhand der Anforderung 8 bewertet.

3.2.1 Empfehlungsansätze in Kompositionsplattformen

Im **Webservice-Bereich** existieren zahlreiche Ansätze zur Berechnung von Vorschlägen basierend auf strukturellen Merkmalen von Kompositionsgraphen, zum Beispiel [Maa+11; CGT11], semantischer Ähnlichkeit und Koppelbarkeit von Webservices, wie [Zha+10; LLM11; Meh+10b], oder Abgleich von Webservice und Nutzerkontext, beispielsweise [Cho+14]. Die Arbeiten [Maa+11; CGT11] stellen konkrete Algorithmen zur Berechnung von Kompositionsempfehlungen vor. Chan et al. [CGT11] berechnen für einen gegebenen Service Alternativen, indem in vorhandenen Kompositionsmodellen nach Services mit ähnlichem Kompositionskontext gesucht wird. [Maa+11] schlagen vor, wiederkehrende Sequenzen von Services zu identifizieren und damit Erweiterungen für eine gegebene Komposition abzuleiten. Diese Ansätze zeigen gut, wie der aktuelle Kompositionskontext genutzt werden kann, sind allerdings nicht hybrid und beziehen den Nutzerkontext nicht oder nur sehr limitiert ein. Da algorithmische Fragestellungen im Vordergrund stehen, sind die meisten Kriterien nicht erfüllt beziehungsweise nicht anwendbar.

Zahlreiche Vertreter wie [Zha+10; LLM11] setzen hybride Empfehlungssysteme ein. In [Zha+10] werden semantische Abgleiche von Ein- und Ausgaben von Services mit Überprüfung des Nutzerkontextes und globaler Wichtigkeit von Services kombiniert. Stellvertretend wird der Ansatz von [LLM11] genauer untersucht, da dieser grundsätzlich ähnlich funktioniert und weiterführende Konzepte bietet. Das Empfehlungssystem von [LLM11] umgeht Kaltstartprobleme durch Kombination von semantischen Ansätzen und kollaborativem Filtern. Webservices werden hierzu in Beschreibungslogik unter anderem anhand Kategorie, Ein- und Ausgaben beschrieben. Durch semantikbasierten Vergleich der Beschreibungen können Aussagen über Ähnlichkeiten von Services vorberechnet werden. Daneben findet kollaboratives Filtern statt. Dazu existieren Nutzerprofile, die explizit in einem Bewertungsformular erfasste Präferenzen hinsichtlich Qualitätseigenschaften von Services beinhalten. Weiterhin werden Nutzungshistorien gepflegt und semantisch modelliert, wobei der Nutzungskontext anhand von Dimensionen wie dem Verwendungszweck und dem Ort hinterlegt wird. Schließlich findet eine Verrechnung statt, sodass anfangs vorrangig inhaltsbasierte Empfehlungen zum Tragen kommen, während Ergebnisse des

kollaborativen Filterns zunehmend Einfluss erhalten sobald diese vorliegen. **Fazit:** Das Konzept ist hybrid und bietet Ansätze zur Erfassung und Nutzung von Nutzerfeedback. Kontextsensitivität kann hinsichtlich des Nutzerkontextes überzeugen, während der Kompositionskontext und die funktionale Relevanz auf Basis semantischer Technologien nur limitiert einbezogen werden. Defizite zeigen sich bezüglich der übrigen Kriterien, z. B. werden nicht komplexe Kompositionsfragmente sondern einzelne Services empfohlen.

Im **Mashup-Bereich** wurden ähnliche Konzepte aufgegriffen und erweitert. Nachfolgend wird ein Überblick gegeben und ausgewählte Vertreter werden vertiefend behandelt.

Zahlreiche Empfehlungssysteme nutzen Kompositionsmuster, die aus bestehenden Mashups extrahiert werden, um Vorschläge für einen gegebenen Kontext abzuleiten. Zugehörige Vertreter sind beispielsweise [Roy+10; Wan+15; Yan+12]. Nachfolgend wird der Ansatz von OMELETTE als Referenzvertreter untersucht. Dessen Grundgedanke ist die Unterstützung des Nutzers durch Vorschläge auf Basis des kollektiven Wissens der Entwicklergemeinschaft [Roy+10; RDC14]. Dazu wird Kompositionswissen in Form von *Patterns* aus bestehenden Mashups extrahiert, zum Beispiel durch statistische Analysen. Verschiedene Pattern-Klassen werden unterstützt, beispielsweise:

- *Connector Patterns* repräsentieren zwei Komponenten, die über einen bestimmten Kommunikationskanal verknüpft sind.
- *Component Co-Occurrence Patterns* beschreiben, dass zwei Komponenten statistisch häufig zusammen auftreten.
- *Multi-Component Patterns* umfassen mehrere Komponenten, inklusive Konfiguration und Kommunikationskanälen.

Unter bestimmten Umständen, sogenannten *Triggern*, wird der Empfehlungsprozess angestoßen. Trigger sind dabei stets an explizite Nutzeraktionen gekoppelt. Auf Basis einer Assoziation von Triggern und zugehörigen Pattern-Klassen werden anschließend Kandidaten bestimmt und daraufhin sortiert. Für das Ranking wurden drei Algorithmen vorgestellt, die sich dahingehend unterscheiden, welche Faktoren einbezogen werden. Beispielsweise stützen sich personalisierte Empfehlungen auf implizites Nutzerfeedback (Nutzungshäufigkeiten). Vorgeschlagene Patterns werden dem Nutzer in einem Fenster angezeigt, siehe Abbildung 3.4, indem vorrangig technische Metadaten visualisiert werden. Wählt ein Nutzer ein empfohlenes Pattern aus, sorgt das *Pattern Weaving* dafür, dass es in die aktuelle Komposition integriert wird. **Fazit:** Das Empfehlungssystem schlägt auch komplexe Kompositionsfragmente vor und integriert diese automatisch. Durch Trigger ist ein Ansatz vorhanden, der Durchgängigkeit ermöglicht, auch wenn stets eine Kompositionsaktion des Nutzers vorliegen muss. Die Kontextsensitivität ist nur teilweise erfüllt, da zwar die aktuelle Komposition beachtet wird, der Nutzerkontext allerdings sehr rudimentär ausfällt und es kein Konzept für funktionale Relevanz gibt. Erfassung und Nutzung ist nur für implizites Feedback vorgesehen. Die Zuordnung von Triggern zu Pattern-Klassen, deren Herkunft offen bleibt, kann als stark limitierte Konfigurierbarkeit aufgefasst werden. Letztlich mangelt es der Darstellung von Empfehlungen an Mitteln zum Herstellen von Nachvollziehbarkeit.

Semantikbasierte Ansätze mit Abgleich von Ein- und Ausgaben wurden zuerst im Umfeld von SWS vorgestellt und dort auch im Rahmen von EUD-Plattformen erprobt, siehe zum Beispiel [Meh+10b]. Zahlreiche Mashup-Vertreter haben das Konzept aufgegriffen [Elm+08; BDM10; PRM11b; Liz+14; Nus+12; Pic+10; Voi+12; Tie15]. Aufgrund der

höchsten Anforderungsabdeckung wird das für die Plattform SMASHAKER entwickelte Konzept von Bianchini et al. im Detail vorgestellt [BDM10; BDM13b; BDM13c; BDM13a]. Grundlage des Ansatzes ist ein *Web-API-Modell*, das Komponenten anhand von einer Kategorie und semantischen *Tags* beschreibt. Weiterhin werden Mashups zugeordnet und vom Entwickler manuell mit semantischen Tags ausgestattet. Darüber hinaus können Komponenten von Nutzern bewertet werden, wobei im System die Expertise des Nutzers zusammen mit der Bewertung abgelegt wird. Der Prozess von Matching und Ranking kann konfiguriert werden. Dazu existieren *Web API Search Patterns*, die Suchfunktionen und -szenarien deklarativ charakterisieren. Darin werden die Suchanfragen, zu verwendende Metriken, Aggregationsfunktionen, Gewichtungen und Schwellenwerte festgelegt. *Suchszenarien* werden hinsichtlich des Suchziels (Auswahl einer einzelnen Komponente, Komplettierung eines Mashups, Austausch einer vorhandenen Komponente) und der Suchtopologie (einfache Suche, erweiterte Suche, proaktive Suche) unterschieden. Entsprechend unterscheidet sich jeweils die Struktur der Suchanfrage. Als Beispiele für Metriken seien genannt: semantikbasierte Ähnlichkeit von Kategorien und semantischen Tags sowie strukturelle Ähnlichkeit von Mashups. Je nach Ranking-Funktion des *Search Pattern* erfolgt schließlich die Aggregation der einzelnen Metriken. **Fazit:** Der Ansatz ist hybrid und bietet ein gutes Konzept für Konfigurierbarkeit. Es werden jedoch keine komplexen Kompositionsfragmente vorgeschlagen, Nachvollziehbarkeit wird nicht behandelt und eine automatische Integration von Vorschlägen fehlt. Durchgängigkeit ist teilweise erfüllt, da zwar viele, auch proaktive Situationen beachtet werden, jedoch starr kodiert sind. Kontextsensitivität ist mit leichten Abzügen bei funktionaler Relevanz und Nutzerkontext erfüllt. *Web API Search Patterns* erlauben Konfigurierbarkeit, jedoch nur bezüglich der Berechnung von Empfehlungen. Feedback wird implizit und explizit gesammelt und verwertet, es mangelt jedoch an Kontextbezug.

Diverse Vertreter verfolgen den Ansatz, unter Zuhilfenahme von Planungstechniken aus dem Bereich der künstlichen Intelligenz Anwendungen zu generieren und in einem Schritt automatisch zu vervollständigen, zum Beispiel [GMP09; Ma+13; Elm+08; Liu+15]. Aufgrund der algorithmischen Komplexität und vergleichsweise geringen Anforderungsabdeckung, die Abbildung A.5 im Anhang belegt, erfolgt keine genauere Behandlung. Daneben haben sich verschiedene hybride Ansätze herausgebildet, wie [Pic+10; TTA11; Che+09; GMP09; Voi+12]. Qualitätsbasierte Empfehlungen bilden einen wesentlichen Teil der Plattform von PEUDOM [Pic13]. Das Empfehlungssystem bezieht mehrere Metriken ein: syntaktische und semantische Kompatibilität sowie Ähnlichkeit von Komponenten, Qualität und Mehrwert. Ähnlich zu CRUISE findet eine deklarative Beschreibung von Komponenten statt, unter anderem der Schnittstellen, die semantisch annotiert sind. Auf dieser Basis kann die Kompatibilität von Komponenten unter technischen und semantischen Gesichtspunkten ermittelt werden. Ein Qualitätsmodell definiert nicht-funktionale Eigenschaften von Komponenten und Mashups. Dadurch kann unter qualitativen Aspekten berechnet werden, wie ähnlich Komponenten untereinander sind und welchen Einfluss sie auf ein vorliegendes Mashup besitzen. Das Konzept des *Added Value* soll ausdrücken, welchen Mehrwert ein Kandidat auf Ebene von Funktionalität, Daten und Visualisierung bietet. Schließlich drücken Assoziationsregeln aus, welche Kategorien von Komponenten häufig zusammen auftreten. Das Empfehlungssystem kommt in zwei Fällen zum Einsatz: bei der Erstellung von Kommunikationskanälen und in dedizierten Fenstern, die bei Bedarf alternative und erweiternde Komponenten vorschlagen, siehe Abbildung 3.8. Bezüglich der Darstellung von Empfehlungen ist hervorzuheben, dass angedacht wurde, Details zu

Metriken pro Kandidat aufzuführen. **Fazit:** Der hybride Ansatz sieht eine automatische Integration von Vorschlägen vor. Erfassung und Nutzung von Feedback ist durch Assoziationsregeln und Nutzerbewertungen gegeben. Empfehlungen werden in mehreren, fest definierten Situationen unterbreitet, sodass die Durchgängigkeit eingeschränkt ist. Weiterhin mangelt es an der Unterstützung komplexer Kompositionsfragmente. Die Kontextsensitivität ist nur teilweise erfüllt, da der Kompositionskontext beachtet wird, der Nutzerkontext jedoch auf Bewertungen begrenzt ist. Added Value kommt als Ansatz für funktionale Relevanz in Frage, allerdings bleiben die Ausführungen dazu sehr vage, es fehlt an konkreten Konzepten. Selbiges gilt für die Konzepte zur Schaffung von Nachvollziehbarkeit, zum Beispiel bleibt offen, welche Details die Nutzer tatsächlich zu den Metriken eines Kandidaten präsentiert bekommen.

Für VizBoard wurde ein hybrides Empfehlungssystem entwickelt [Voi+12; VFM13]. In einer Vorselektion werden Komponenten hinsichtlich Nutzerkontextparametern wie Geräteeigenschaften gefiltert. Anschließend erfolgt das Matching, das auf einem semantikbasierten Vergleich von Komponentenbeschreibungen und Nutzeranforderungen, wie darzustellenden Datenkonzepten, beruht. Visualisierungsspezifische Heuristiken kommen zusätzlich zum Einsatz, beispielsweise beim Abgleich von Skalenniveaus. Die Sortierung von Kandidatenkomponenten übernimmt das Ranking. Hierzu sieht das Konzept vor, dass Nutzerfeedback erfasst, modelliert und im Ranking einbezogen wird. Genauer werden Zuordnungen von visualisierten Datenkonzepten und genutzten Komponenten als Visualisierungswissen abgespeichert. Diese Nutzerprofile dienen schließlich dazu, das Ranking der Ergebnisse durch kollaboratives Filtern zu bestimmen. **Fazit:** Wie gefordert ist das Empfehlungssystem hybrid. Zur Erhöhung der Nachvollziehbarkeit von Empfehlungen wird die Anforderungserfüllung angezeigt. Hervorzuheben ist die Erfassung und Verwendung von Nutzerfeedback sowie die Beachtung des Nutzerkontextes. Das Konzept weist jedoch Defizite auf, zum Beispiel werden keine komplexen Kompositionsfragmente vorgeschlagen, sondern nur einzelne Komponenten, und der aktuelle Kompositionskontext wird gänzlich missachtet. Weiterhin existiert kein Ansatz für Konfigurierbarkeit und Empfehlungen werden nur bei der Entwicklung der Anwendung unterbreitet.

3.2.2 Nutzerfeedback in Empfehlungssystemen

Zur Verbesserung von Prognosen basieren Empfehlungssysteme zumeist auf Nutzermodellen, welche Aufschluss über Interessen und Präferenzen gewähren. Dazu werden Interaktionen des Nutzers mit Items der Datenbasis gespeichert, die implizite und explizite Informationen über die Präferenzen und Interessen des Nutzers beinhalten. Solche Informationen werden typischerweise als **explizites** und **implizites Feedback** bezeichnet [JWK14]. Eine ausführliche Charakterisierung und Gegenüberstellung von implizitem und explizitem Nutzerfeedback in Empfehlungssystemen nehmen Jawaheer et al. [JWK14] vor. Wesentliche Aspekte daraus werden im Folgenden beschrieben und mit weiteren Forschungsergebnissen angereichert.

Kognitiver Aufwand: Explizites Feedback erfordert stets einen zeitlichen und kognitiven Aufwand vom Nutzer. Er wird in seinem Arbeitsfluss unterbrochen, sodass die Aufforderung Feedback zu geben stört oder ignoriert wird und ihr somit nur wenige Nutzer folgen [Cla+01]. Implizites Feedback kann hingegen im Hintergrund gesammelt werden und ist dadurch weniger aufdringlich. Diese Unabhängigkeit von der Bereitschaft von Nutzern geht jedoch mit höherer Unsicherheit der Ergebnisse einher.

Messskala: Explizites Nutzerfeedback wird in der Regel in Form von Bewertungen und Rezensionen abgegeben und daher ordinal skaliert, auf Basis n-stufiger Likert-Skalen. Ein zentraler Untersuchungsgegenstand von Nutzerstudien, wie [SS11], sind geeignete Bewertungsskalen, insbesondere die Frage wie n zu wählen ist. Dabei gilt es, einen adäquaten Kompromiss aus zeitlichen sowie kognitivem Aufwand und Aussagekraft der Bewertung zu identifizieren. Die benötigte Zeit für die Abgabe einer Bewertung scheint zwar mit höherer Granularität der Skala zuzunehmen, die kognitive Belastung jedoch annähernd konstant zu bleiben [SS11]. Zudem wurde deutlich, dass es zeitaufwändiger ist, Bewertungen im mittleren Bereich der Skala zu vergeben als in den Extrema. Die Zufriedenheit von Nutzern mit den Skalen scheint bei fünfstufigen Skalen am ausgeprägtesten [SS11]. In der Praxis sind unäre Systeme, wie bei Facebook, binäre Systeme, zum Beispiel bei Youtube, und Ansätze mit $n = 5$ zu finden, etwa bei Amazon. Bei implizitem Feedback werden in der Regel Verhaltensweisen gezählt und es kommen Verhältniswerte und Kardinalskalen zum Einsatz.

Anfälligkeit für Rauschen: Sowohl explizites als auch implizites Feedback sind anfällig für Rauschen, verursacht durch Nutzer, das System oder Erfassungsmechanismen bei implizitem Feedback. Als Spezialfall von Rauschen gilt gezielte Manipulation, die insbesondere bei explizitem Feedback auftritt, indem Items gezielt auf- oder abgewertet werden. Diesbezüglich können zum Beispiel Schwellenwerte eingesetzt werden [OHS06].

Polarität: Explizites Feedback kann positiv und negativ ausfallen, da es Präferenzen des Nutzers widerspiegelt. Implizites Feedback gilt verbreitet als ausschließlich positiv [JWK14]. Es existieren jedoch Ansätze zur Identifikation von Verhaltensweisen, die als negative Bewertung interpretiert werden, und zum qualitätssteigernden Einsatz des inferierten negativen Feedbacks in Empfehlungssystemen, zum Beispiel [PV13].

Reichweite: Während typischerweise lediglich eine Teilmenge aller Nutzer explizit Feedback gibt, kann implizites Feedback von allen Nutzern gesammelt werden.

Transparenz: Explizites Feedback ist für Nutzer transparent, das heißt, diese sind sich darüber im Klaren, dass das Empfehlungssystem die Daten nutzt und wozu. Dadurch steigen Manipulierbarkeit und Erklärbarkeit von Empfehlungen. Implizites Feedback ist intransparent, da meist unbekannt ist, welche Nutzeraktionen beobachtet werden. Somit ist die Manipulierbarkeit geringer, jedoch fällt es schwerer Empfehlungen zu erklären.

Verzerrung: Bei Nutzung von explizitem Feedback können Empfehlungssysteme eine Grundtendenz hin zur Meinung der aktivsten Nutzer aufweisen und somit verzerrte Prognosen hervorbringen. Zwar sind Bewertungen ein zuverlässiges Mittel, da, wie Cosley et al. [Cos+03] zeigen konnten, bei erneuter Wertung von Items durch Nutzer in 60% der Fälle ein identischer Wert vorliegt. Dies trifft allerdings nur zu, falls das bereits abgegebene Rating nicht sichtbar ist. Sonst tendieren Nutzer dazu, sich in ihrer Meinung dem angezeigten Wert anzunähern, werden somit in ihrer Entscheidung beeinflusst.

Feedback basiert auf Interaktionen des Nutzers. Zahlreiche Klassifikation von beobachtbaren Verhaltensweisen, welche potentiell zur Schlussfolgerung von Feedback geeignet sind, wurden vorgestellt. An dieser Stelle wird auf die Klassifikation von Oard et al. [OK01] eingegangen, da sie vorhandene konsolidiert und erweitert. Darin werden vier Kategorien unterschieden und diesen konkrete Aktivitäten zugeordnet. *Examine* umfasst Aktionen, wie das Ansehen, Anhören und Auswählen von Items. Einer Untersuchung zufolge handelt es sich bei dem Scrollen, der Verweilzeit und der Mausbewegung auf Webseiten um geeignete Indikatoren für Nutzerinteresse und somit um geeignete Metriken für implizites Feedback [Cla+01]. *Retain* gruppiert Aktionen, welche auf eine zukünftige Nutzung eines

Items hindeuten und somit Interesse andeuten. Beispiele sind ein Lesezeichen zu erstellen und etwas zu speichern. *Reference* beinhaltet Aktivitäten, die Items in Relation setzen, zum Beispiel Zitieren und Verlinken. Die Kategorie *Annotate* beschreibt Aktivitäten, deren Ziel es ist, zu den Werten von Informationsobjekten beizutragen, wie etwas zu bewerten. Sie umfasst somit Aktivitäten, die für explizites Feedback relevant sind.

Die Forschung im Bereich von Empfehlungssystemen beschäftigt sich verstärkt mit explizitem Feedback [PV13]. Den aktuellen Stand zu Feedback im Mashup-Bereich skizziert Abschnitt 3.2.1. Deutlich wurde, dass explizites Feedback vergleichsweise wenig zum Einsatz kommt. Implizit werden Historien von erstellten Anwendungen und Auswahlstatistiken in hybriden Ansätzen wie [Roy+10; VFM13; TTA11] angelegt und ausgewertet. Im Ergebnis werden nicht-personalisiertes Kompositionswissen, jedoch keine Nutzerprofile abgeleitet. Des Weiteren spielt der Kontext, in dem das Feedback abgegeben und gesammelt wurde, bei den wenigsten Vertretern eine Rolle. Einzig die Ansätze [VFM13; BDM13b] berücksichtigen dies, jedoch zu begrenzt, um den Anforderungen aus Kapitel 2.3 zu genügen. In anderen Forschungszweigen zeigen sich diesbezüglich punktuell vielversprechende Lösungsansätze. Die Plattform HyperService [Zha+10] profiliert Nutzer, indem deren Verhaltensweisen analysiert werden. Bei Suche, Browsen und Navigation schneidet die Plattform Zustandsinformationen und implizites Feedback mit. Dazu zählen zum Beispiel die zur Suche eingegebenen Stichworte, das Anklicken von Ergebnissen, die Verweilzeit auf Detailseiten, der Navigationspfad und das Navigationsverhalten, das unterscheidet, ob Pfade selbst hergestellt wurden oder Empfehlungen gefolgt wurde. Wie genau die Erfassung erfolgt und das Modell formal repräsentiert ist, bleibt jedoch unklar. In [Khr15] wird ein Rahmenwerk zur Erfassung und Auswertung von semantisch angereichertem Feedback vorgestellt. Dieses basiert auf semantisch annotierten Inhalten, zum Beispiel Produktbeschreibungen, und einem semantischen Feedback-Modell. Neben dem Zweck und der Zielgruppe des Feedbacks deckt dieses unter anderem Zeit, Ort und Nutzerinformationen ab. Zudem kann eine Menge von Feedback-Elementen genutzt werden, um die Struktur einer Umfrage, eines Quiz oder einer Bewertung zu repräsentieren. Der Ansatz fokussiert auf explizites Feedback und ein formales Modell wird nicht präsentiert. In der widget-basierten Plattform des ROLE-Projekts [Gov+15] erfolgt das Mitschneiden von Nutzeraktivitäten durch ein spezielles Widget, das dazu relevante Events von anderen Widgets registriert. Als Modell kommen *Contextualized Attention Metadata* zum Einsatz, die Tupel aus Entität, Event und Sitzungsinformationen beschreiben. [VGG13] verfolgt dies unter Zuhilfenahme von *Activity Streams* [@ASWG]. Eine *Activity* umfasst hierbei das Datum der Aktivität, den Akteur, die eigentliche Aktion, das Objekt, auf dem die Aktion ausgeführt wurde, und ein *Target*, das eine aktionsspezifische Bedeutung besitzt. Vorteil von *Activity Streams* ist die Verbreitung, beispielsweise in Social-Media-Plattformen, und die klare Semantik von Aktionen aus einem vordefinierten Katalog. Der Kontext von Aktionen ist jedoch nur begrenzt modellierbar, sodass zum Beispiel Erweiterungen zur Abbildung des Kompositionskontextes notwendig sind, um den Ansatz für die vorliegende Arbeit zu verwenden.

3.2.3 Fazit

Wie die Untersuchung zeigt, existieren zahlreiche Ansätze zur Berechnung von Kompositionsvorschlägen. Deren jeweilige Defizite lassen erkennen, dass kein Ansatz alle hier relevanten Anforderungen erfüllt. Insbesondere mangelt es an Konzepten zur durch-

gehenden Bereitstellung von Empfehlungen und zur bedarfsgerechten Anpassung des Empfehlungssystems. Dennoch kann auf vielversprechende Konzepte zurückgegriffen werden. Zu benennen sind hier *Trigger* und *Patterns* aus dem OMELETTE-Ansatz, *Web API Search Patterns* nach Bianchini et al. und die Erfassung und Verwendung von Nutzerfeedback wie in VizBoard und PEUDOM. Abbildung A.5 im Anhang zeigt eine genaue Aufschlüsselung der Analyseergebnisse für die untersuchten Ansätze.

Bezüglich Feedback kann resümiert werden, dass eine Kombination von explizitem und implizitem Feedback als vielversprechende Lösung erscheint. Die vorgestellte Kategorisierung von Nutzeraktivitäten gibt Aufschluss über geeignete, beobachtbare Verhaltensweisen, aus denen Feedback abgeleitet werden kann. Sie stellt insgesamt eine gute Basis für Ansätze dar, die sowohl implizites als auch explizites Feedback vereinen. Als zentrale Forschungsfrage stellt sich, inwieweit solche Interaktionen auf die Mashup-Domäne übertragbar sind, insbesondere angesichts der zugrundeliegenden Blackbox-Komponenten.

3.3 Eingabe funktionaler Anforderungen

Bei der Informationssuche stehen Nutzer vor der zentralen Herausforderung, dem System ihr Anliegen mitzuteilen. In Anbetracht heterogener Zielgruppen und Anwendungsfälle haben sich vielzählige Ansätze, die Nutzern dies erlauben sollen, herausgebildet. In Anlehnung an die in [Hea09; MW07; Tri+15; WLZ13] aufgeschlüsselten Optionen werden ausgewählte Konzepte in diesem Abschnitt gruppiert und bewertet.

Textuelle Anfragen: Ansätze in dieser Kategorie können hinsichtlich Freiheits- und Formalisierungsgrad folgendermaßen unterschieden werden: Schlüsselwörter, Freitext, restriktive natürliche Sprache und spezielle Kompositionssprachen, wie in NaturalMash [AP14].

Formular-basierte Suche: Hierbei stehen zahlreiche Filter für die Anfrageformulierung zur Verfügung. Dieser Ansatz tritt zum Beispiel als erweiterte Suche in Suchmaschinen auf.

Query-by-Example: Hierbei handelt es sich um die Suche anhand eines gegebenen Items. Zum Beispiel werden Vorlagen für Komponenten mit den Mitteln des jeweiligen Komponentenmodells verwendet, etwa in CRUISe [PRM11b] und SMASHAKER [BDM10]; deren Fokus liegt jedoch auf technischen Aspekten und eine fachliche Anforderungsspezifikation findet nicht statt, sodass sie im weiteren Verlauf nicht näher betrachtet werden.

Graphische Anfrageformulierung: Diese erfolgt zum Beispiel durch Aufgabenmodellierung in DEMISA [Tie15] und mittels visueller Erstellung von SPARQL-Anfragen in [Tri+15].

Facettierte und hierarchische Suche: Solche Konzepte kommen zum Beispiel in VizBoard [Voi14] und in [BDM17] zum Einsatz.

Assistenten und Dialogsysteme: In diese Kategorie ist beispielsweise der Ansatz aus OMELETTE [Roy+13] einzuordnen.

Programming-by-Demonstration und -by-Example: Bei solchen Ansätzen werden Nutzerinteraktionen im Kontext der Anwendung interpretiert, um Rückschlüsse auf Anforderungen zu erlangen, wie in [Chu+13; Sir+09]. Nutzer demonstrieren somit ihre Ziele durch »Vormachen«. Derartige Ansätze sehen keine explizite fachliche Anforderungsspezifikation vor und werden daher nicht näher betrachtet.

Nachfolgend werden ausgewählte Vertreter für wesentliche Kategorien unter Bezugnahme auf Anforderung 5 (siehe Kapitel 2.3) und deren Unterpunkte analysiert.

3.3.1 Textuelle Ansätze

Viele aktuelle Ansätze im Web, insbesondere Suchmaschinen wie Google [@Goo], und im Mashupbereich, siehe Abschnitt 3.1, setzen auf Stichwortsuche. Am Beispiel der Google-Suche wird deutlich, dass zusätzliche Assistenzfeatures, wie beispielsweise Suchanfragevorschläge, entwickelt wurden. Zudem werden typischerweise die Ergebnisse sofort sichtbar und die Relevanz wird zumindest ansatzweise begründet, zum Beispiel durch Ergebnissortierung und durch Hervorhebung von Treffern im Dokument. Auch kann der Nutzer jederzeit die aktuelle Anfrage einsehen und ändern. In geringem Maße sind verschiedene Einstiegspunkte vorhanden, zum Beispiel existieren die Funktion »Auf gut Glück« und eine erweiterte Suche. Solche Ansätze erfordern vom Nutzer, sein Informationsbedürfnis in eine für das System verständliche textuelle Form zu transformieren. Jedoch bestehen Lücken zwischen Nutzer und System: Während die Probleme des Nutzers durch Unsicherheit und Konfusion charakterisiert sind, nimmt das System Sicherheit und Ordnung an [Kuh91]. Weiterhin existiert oftmals ein Unterschied zwischen dem Vokabular des Nutzers und dem des Systems sowie der darin hinterlegten Items. Der Nutzer benötigt demnach eine Vorstellung, welche Begriffe einzugeben sind und wie das zugrundeliegende Schema der Daten aussieht. Eine Darstellung verfügbarer Suchkriterien und im System vorhandener Items sowie explorative Ansätze fehlen hingegen. Freitextuelle Konzepte erlauben einen hohen Freiheitsgrad, die Nutzerführung ist allerdings begrenzt, was insbesondere in explorativen Szenarien und bei vagem Informationsbedürfnis nachteilig ist. Zahlreiche Ansätze kombinieren reine Stichwortsuche mit einem erhöhten Grad der Nutzerführung, wie [KC13; LLS02; @WA]. In der *Goal-oriented Search Engine* [LLS02] kann der Nutzer aus Kategorien von Zielen auswählen, welche als Schablonen dienen, um dem System die Intention des Nutzers näher zu bringen, zum Beispiel »I want help solving this problem«. Die semantische Suchmaschine *Wolfram|Alpha* [@WA] bietet als weiteren Einstiegspunkt zahlreiche Beispiele, die auf verschiedene, teilweise explorative Weise zugänglich sind. Auf Grundlage semantischer Wissensbasen verdeutlicht der Ansatz dem Nutzer, wie die Anfrage interpretiert wurde und zeigt Alternativen. Zudem kann zwischen den Ergebnissen und Konzepten navigiert werden. Die Grundprobleme textueller Ansätze bleiben allerdings bestehen. Zahlreiche Ansätze beschäftigen sich mit komplett natürlichsprachlichen Eingabemöglichkeiten, welche auf Basis von Wissensmodellen ausgewertet, angereichert sowie umgeschrieben und schließlich in technische, formale Anfragen an Datenbanken übersetzt werden, zum Beispiel [KA11]. Da diese Ansätze vor allem den Übersetzungsprozess und algorithmische Aspekte betrachten, die UI-Seite jedoch nicht oder nur sehr reduziert behandeln, werden sie an dieser Stelle nicht genauer vorgestellt. Andere Vertreter umgehen aufwändige Analysen des eingegebenen Textes durch Nutzung von eingeschränkter Sprache mit vordefinierter Syntax. Als Beispiel sei das bereits in Kapitel 3.1 behandelte *NaturalMash* [AP12; AP14] genannt.

3.3.2 Graphische Anfrageformulierung

Bei Konzepten dieser Kategorie stellen Nutzer ihre Anfrage aktiv graphisch zusammen. Für eine Übersicht klassischer Ansätze sei auf [Hea99] verwiesen. Als Beispiel aus dem

Mashup-Bereich kann sMash [Lu+09; Che+09] genannt werden, siehe Kapitel 3.1. Es erlaubt Nutzern einen explorativen Navigationsansatz zur Suche nach Komponenten, bietet jedoch keinerlei Möglichkeit funktionale Anforderungen zu definieren. In der Linked Widgets Platform [Tri+14; Tri+15] können Nutzer nach Komponenten suchen, indem ein benötigtes Datenmodell im Sinne von semantischen Ein- und Ausgabedatentypen definiert wird. Da Nutzer hierzu mit technischen Konzepten und Datenstrukturen konfrontiert werden, ist die Eignung für Nicht-Programmierer kaum gegeben. Ein Ansatz, der für Nicht-Programmierer besser geeignet erscheint, ist die Aufgabenmodellierung, wie sie beispielsweise in DEMISA [Tie15] vorgeschlagen wurde. Abbildung 3.14 zeigt das konzipierte Werkzeug, den *Aufgabeneditor*. Dieser dient Nutzern dazu, die fachlichen Anforderungen in Form eines hierarchischen Aufgabenmodells zu definieren. Der Aufgabeneditor umfasst mehrere Ansichten, welche die jeweiligen Aspekte und Schritte der Aufgabenmodellierung abdecken: die Strukturierung und Dekomposition von Aufgaben in der *Relation View*, die Spezifikation von Detailinformationen zu Aufgaben in der *Attribute View* sowie die Definition von Datenflüssen zwischen Aufgaben, wofür die *Data Flow View* bereitsteht. In letzterer können Aufgaben anhand ihrer semantisch typisierten Ein- und Ausgaben miteinander verbunden werden. Ein weiterer zentraler Bestandteil des Aufgabeneditors präsentiert passende Komponenten und komplette Mashups, die zur Erledigung der geforderten Aufgabe und einzelner, selektierter Aufgaben dienen können. Um die Annotation von Aufgaben zu vereinfachen, können Nutzer auf einen *Ontology Browser* zurückgreifen, der die Exploration verfügbarer Ontologien ermöglicht. Der Ansatz bietet die geforderte Spezifikation auf fachlicher Ebene und in iterativer Vorgehensweise. Ergebnisse werden dynamisch angezeigt und die aktuelle Anfrage (Aufgabenmodell) ist stets sichtbar. Allerdings wird nur ein Vorgehen angeboten: die Dekomposition von Aufgaben. Zudem lässt die vorgesehene Assistenz zu wünschen übrig. Nutzer werden zum Beispiel kaum dabei unterstützt, Aufgaben in passender Granularität zu spezifizieren und zu verfeinern. Weiterhin erhalten Nutzer keinen Einblick, welche Aufgaben tatsächlich durch verfügbare Komponenten erfüllbar sind, sodass es nach der Modellierung vorkommen kann, dass keine Ergebnisse vorliegen. Hinsichtlich Relevanzindikation der Ergebnisse erfahren Nutzer zumindest, für welche der Aufgaben Komponenten bereitstellen.

3.3.3 Hierarchische und facettierte Suche

Ansätze zur hierarchischen Suche sehen vor, dass Nutzer anhand einer vorgegebenen Struktur oder Taxonomie die verfügbaren Items durchsuchen können. Als Referenzvertreter aus dem Bereich SWS und CWA dient SOA4All [Meh+10a]. Wie in Abschnitt 3.1 bereits angedeutet, steht Nutzern eine Taxonomie von Templates von Webservice-Kompositionen zur Verfügung. In dieser können Nutzer zunächst zur benötigten Kategorie navigieren und erhalten nach Selektion eine Auflistung zugehöriger Templates. Sofern der Nutzer sein Problem der Taxonomie entsprechend klassifizieren kann, mag der Ansatz ausreichend sein. Zudem ähneln die Kategorien Aufgaben, sodass die Angabe auf fachlicher Ebene erfolgt, und die tatsächlich im System verfügbaren Templates und Kategorien sind ersichtlich. Hilfe bei der Dekomposition des Problems sowie bei Vokabularunterschieden erhält der Nutzer nicht. Weiterhin fehlen Empfehlungen zu Anfragen sowie eine Indikation der Relevanz von Ergebnissen.

Als Referenzvertreter für facettierte Suche wird der Facettenbrowser von VizBoard [Voi14] gewählt, da dieser für die passende Anwendungsdomäne konzipiert und erprobt

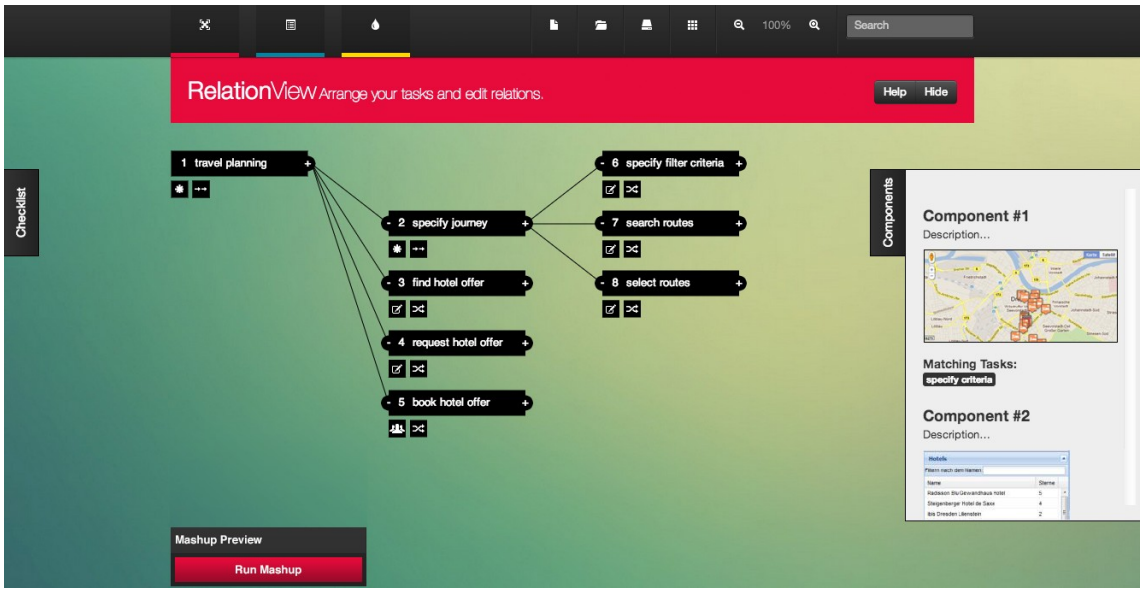


Abbildung 3.14: Aufgabeneditor aus dem DEMISA-Projekt [Tie15]

wurde. Abbildung 3.15 zeigt den Aufbau. Wie im oberen Bereich zu erkennen ist, werden Visualisierungsanforderungen anhand von Facetten wie den darzustellenden Daten ①, der graphischen Repräsentation, dem Detailgrad und durchführbaren Aktionen ② formuliert. Die verfügbaren Werte pro Facette ergeben sich aus den zuvor selektierten Datensätzen und der genutzten Visualisierungsontologie. Hinter den Werten wird die zu erwartende Ergebnisanzahl angedeutet. Ausgewählte Facettenwerte sind im Bereich ③ ersichtlich und als Muss- oder Kannkriterium definierbar. Auf Basis eines Empfehlungsansatzes (siehe Kapitel 3.2) präsentiert das System jederzeit passende Visualisierungskomponenten in einer Liste ④, wobei der Erfüllungsgrad der Anforderungen durch die Länge und die Farbe der Linien unter dem Komponentennamen vermittelt werden soll. Bei Bedarf öffnet sich ein Detailbereich zur ausgewählten Komponente ⑤. Die Forderung nach fachlich funktionaler Ebene wird nur teilweise erfüllt, da zwar Terminologie aus der Visualisierung, jedoch nicht aus anderen Domänen auftritt. Kriterien können iterativ geändert werden, sind stets sichtbar und es existiert eine dynamische Ergebnisanzeige mit Relevanzindikatoren, die allerdings eine Vorschau vermissen lässt. Assistenz ist begrenzt auf die Anzeige der zu erwartenden Ergebnisanzahl. Vorschläge zu Suchanfragen und dergleichen fehlen. Ebenso mangelt es an explorativen Ansätzen und an Unterstützung verschiedener Suchstrategien.

3.3.4 Assistenten und dialogbasierte Ansätze

Mehrere Vertreter gestatten es Nutzern ihre Anforderungen basierend auf **Tags** auszudrücken, wie [Tri+15; Ria+08; Liu+15]. Der Ansatz von [Tri+15] im Rahmen der Linked Widgets Platform ist letztlich textbasiert und es mangelt unter anderem an Ergebnisvorschau und Nutzerführung durch Vorschläge. Der Ansatz von iMashup [Liu+15] ist nahezu identisch zu dem von MARIO [Ria+08] und wurde bereits in Kapitel 3.1 behandelt. Demnach wird die iterative Anforderungseingabe auf fachlicher Ebene unterstützt, Ergebnisse inklusive Vorschau werden dynamisch angezeigt und die aktuelle Anfrage ist stets sichtbar.

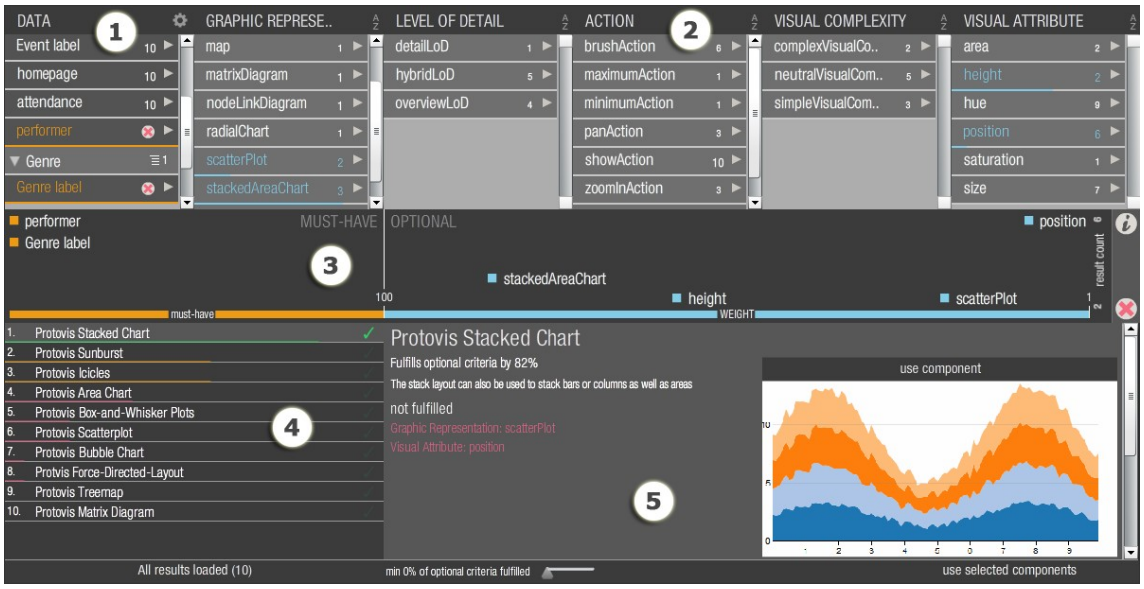


Abbildung 3.15: Facettenbrowser zur Angabe von Visualisierungsanforderungen [Voi14]

Zumindest ein Ausschnitt der verfügbaren Funktionalitäten wird dem Nutzer in der *Tag Cloud* angezeigt. Relevanz wird anhand der Rangfolge indiziert. Jedoch ist ein manueller Anforderungsabgleich notwendig. Zudem mangelt es an Assistenz.

Bildbasierte Ansätze sind im Kontext der Produktsuche etablierte Lösungen, um Nutzer mit vagem Informationsbedürfnis und ungenauen Zielvorstellungen zu inspirieren und ihnen zu assistieren. Dies veranschaulichen beispielsweise die Angebote *Picture your Holiday* von British Airways [@BA] und *Inspire Me* der Firma Iberia [@Iberia] in der Reisedomäne und die Mediathek des ZDF [@ZDF] im Bereich medialer Inhalte.

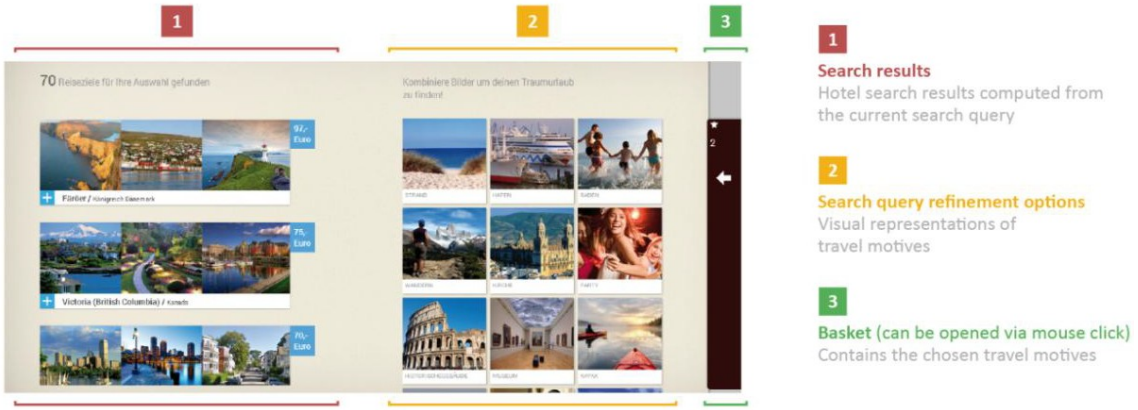


Abbildung 3.16: Bildbasierte Suche gemäß dem Ansatz von Get Inspired! [Bot+14]

Ein wissenschaftlicher Ansatz, der die Eignung eines bildbasierten Ansatzes zur Produktsuche in der Reisedomäne untersucht, ist *Get Inspired!* [Bot+14], siehe Abbildung 3.16. Zielgruppe sind Personen mit vorhandener, jedoch nicht auf Expertenniveau befindlicher Fachkenntnis und einem vagen Informationsbedürfnis. Die Autoren prägen den Begriff *motivbasierte Suche*, die dadurch gekennzeichnet ist, dass sie oft auf Grundlage unbewus-

ster Motive und Erwartungen initiiert wird, welche nur schwer in eine konkrete Anfrage an das System transformiert werden können. Basierend auf einer Polyhierarchie von Items, die dem Nutzer unbekannt sind, findet eine schrittweise Reduktion des Ergebnisraums statt. Dazu wählt der Nutzer bildhaft repräsentierte Domänenkonzepte ② aus. Daraufhin werden die Ergebnisse aktualisiert ① und zudem neue Domänenkonzepte in ② zur Auswahl gestellt. Der zugrundeliegende Algorithmus strebt dabei an, diese Konzepte so anzubieten, dass die Ergebnismenge möglichst schnell reduziert wird. Bereits selektierte Motive beinhaltet Bereich ③. Der Ansatz sieht die iterativ verfeinernde Spezifikation fachlicher Anforderungen vor. Ergebnisse werden dynamisch angezeigt und die bereits zusammengetragene Anfrage ist zugänglich. Zumindest ein Ausschnitt der verfügbaren Kriterien oder Items (Motive) wird geboten. Exploratives Vorgehen wird ansatzweise unterstützt. Zudem fehlt es an Relevanzindikation der Ergebnisse.

Agenten und Wizards erheben in einem zumeist schrittweisen Prozess die Anforderungen des Nutzers, wobei gezielt bestimmte Aspekte und Facetten abgefragt werden, um die Systemrepräsentation der Anforderungen stufenweise zu verfeinern. Derartige Ansätze sind typisch für die Produktkonfiguration und Produktsuche. Ein Wizard zur Erstellung von Webanwendungen wird in [CDP16] vorgestellt. Dieser leitet Nutzer, die von den Autoren nicht genauer charakterisiert werden, in fünf Schritten durch den Entwicklungsprozess. Zuerst erfolgt die Eingrenzung der Anwendungsdomäne, wobei Nutzer aus vorgefertigten Domänen auswählen können. Im darauffolgenden Schritt gilt es, benötigte Dienste und Geschäftsprozesse zu definieren. Hierbei können vorgegebene Funktionalitäten gewählt oder neue spezifiziert werden, etwa durch graphische Modellierung. In nachfolgenden Schritten werden weitere Anwendungsaspekte behandelt, zum Beispiel Rollen inklusive ihrer Rechte sowie das Layout. Der Ansatz geht über die Eingabe funktionaler Anforderungen hinaus und bildet vielmehr den gesamten Entwicklungsprozess einer Webanwendung ab. Als übertragbar zeigt sich der schrittweise Ansatz zur Trennung der Belange, wobei insbesondere die ersten zwei Schritte relevant sind. Ein *Mashup Recommender Widget* als Teil einer Plattform für personalisierte Lernumgebungen schlagen [Nus+12] vor. Es erlaubt Nutzern nach Komponenten zu suchen, die für bestimmte Lernstrategien als geeignet gelten. Aufgrund der oberflächlichen Darstellungen entzieht sich das Konzept jedoch der Bewertung.

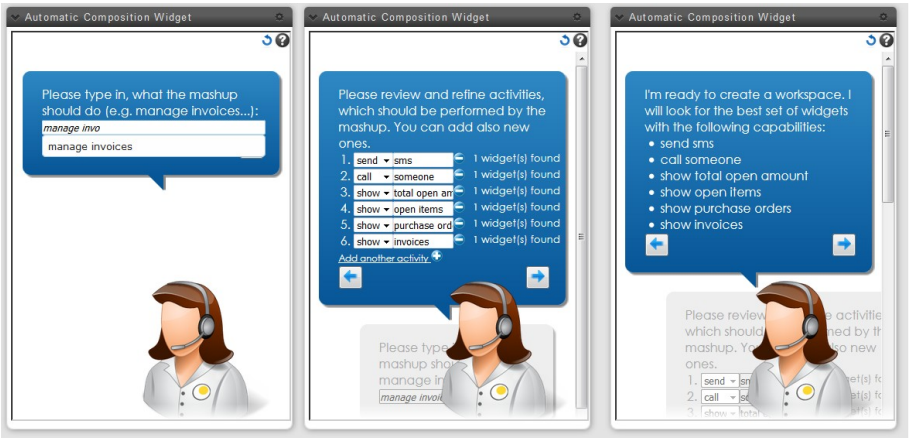


Abbildung 3.17: Agent zur Eingabe funktionaler Anforderungen aus OMELETTE [Con13a]

In OMELETTE wurde ein dialogbasierter Agent entworfen [Con13b; Roy+13]. Konzeptionell fußt dieser auf einer semantischen Wissensbasis aus Domänen- und Zielmodell. Für letzteres kommt das Aufgabenmodell aus DEMISA zum Einsatz. Nutzereingaben stammen aus der Interaktion mit dem Assistenten. Innerhalb des Dialogs werden die Ziele verfeinert, indem Nutzern Fragen gemäß einer vordefinierten *Conversation Strategy* gestellt werden. Eine beispielhafte Strategie klärt zunächst komposite Aufgaben und hinterfragt anschließend Details der Aufgaben. Das Ergebnis des Dialogs stellt ein *Overlay Model* dar, das die Interessen des Nutzers beinhaltet. Praktisch umgesetzt wurde der in Abbildung 3.17 illustrierte Stand. Zuerst gibt der Nutzer in ein Textfeld eine zu lösende Aufgabe ein, wobei er durch Vervollständigungen unterstützt wird, siehe linker Teil der Abbildung. Falls das System die Aufgabe kennt, wird im nächsten Schritt die Eingabe dargestellt – für atomare Aufgaben erscheint genau ein Eintrag und bei kompositen sämtliche Unteraufgaben (Abbildung mittig). Ist dem System die Aufgabe unbekannt, obliegt es dem Nutzer in diesem Schritt die Unteraufgaben zu benennen. Die Anzahl passender Komponenten wird für Teilaufgaben, jedoch nicht für die Gesamtaufgabe vermerkt. Zwischen den Teilschritten kann navigiert werden. Abschließend prüft der Nutzer die Angaben und erhält bei Bestätigung eine Anwendung. Der Ansatz operiert auf fachlicher Ebene und lässt Iterationen zu. Die eingegebenen Anforderungen sind sichtbar. Assistenz ist auf Vervollständigungen und die Anzahl von Ergebnissen limitiert. Es fehlt an einer Darstellung passender Ergebnisse, einer Vorschau, Relevanzindikatoren und an explorativen Ansätzen. Wichtige Konzepte, wie *Conversation Strategy*, sind nur oberflächlich erklärt und der Ansatz wurde nicht evaluiert.

3.3.5 Fazit

Zusammenfassend lässt sich sagen, dass kein singulärer Ansatz als vollständige Lösung prädestiniert ist. Die Konzepte unterscheiden sich stark hinsichtlich des Grades an Nutzerführung. Dieser ist bei Get Inspired! und SOA4All als hoch einzuschätzen, während er zum Beispiel bei OMELETTE mäßig ausfällt und bei klassischen Suchmaschinen und DEMISA gering erscheint. Nahezu invers dazu gestaltet sich der Freiheitsgrad, der jedoch angesichts der Zielgruppe weniger maßgeblich ist. Insgesamt scheint eine Kombination von Techniken zielführend, die mehrere Einstiegspunkte erlauben und somit verschiedenen Präferenzen und Expertisegraden gerecht werden, und die kontextsensitiv ausgewählt werden. Beispielsweise kann ein Wizard zur geführten, explorativen Suche angeboten werden, daneben Direkteinstiege wie Lesezeichen, zuletzt genutzte Items sowie Beispiele und eine Schnellsuche wie in gängigen Suchmaschinen. Allerdings muss die Anforderungsspezifikation auf fachlicher Ebene stattfinden, wie beispielsweise in DEMISA, um die Zugänglichkeit für Nicht-Programmierer sicherzustellen.

3.4 Ansätze zur Datenmediation

Typischerweise stammen Komponenten, welche in Kompositionsplattformen für CWA bereitgestellt werden, von verschiedenen Drittanbietern. Wie in Kapitel 1.1 erläutert, ist daher mit Heterogenität der Schnittstellen auf syntaktischer sowie semantischer Ebene zu rechnen. Dies beschränkt schließlich die Verknüpfbarkeit von Komponenten oder bedeutet zusätzlichen Aufwand für den Anwendungsentwickler. Insbesondere im Rahmen

von EUD durch Nicht-Programmierer sind daher automatisierte Ansätze gefordert. In diesem Abschnitt werden Konzepte untersucht und anhand der Anforderung 6.2 bewertet.

3.4.1 Ontology Mediation

Ontology Mediation verfolgt das Ziel, Abbildungen zwischen Konzepten aus verschiedenen Ontologien zu identifizieren, zu beschreiben und falls notwendig anzuwenden. Das Teilgebiet *Ontology Alignment* behandelt den semi-automatischen Prozess der Identifikation von Beziehungen zwischen Ontologien. Dazu existiert eine Vielzahl an Techniken, über welche [SE13; Jan12; Che+16] einen umfassenden Überblick gewähren. Der Alignment-Prozess resultiert in der Regel in Abbildungen, *Ontology Mappings* genannt, mit einer gewissen Unsicherheit, sodass Domänenexperten im Zweifelsfall zur Validierung oder Berichtigung in den Vorgang einbezogen werden müssen, um Abbildungen schließlich automatisiert anwenden zu können. Wie in [Che+16] attestiert wird, erreichen aktuelle Ansätze ein F-Maß[1] bis 0.94, was für eine hohe Zuverlässigkeit der identifizierten Abbildungen spricht. Es herrscht kein Konsens darüber, wie Abbildungsvorschriften zwischen Ontologien formalisiert werden sollten. Hierfür existieren verschiedene Ansätze, wie die Nutzung dedizierter Sprachen, zum Beispiel in [SB05], und mittels Konstrukten der jeweiligen Ontologiesprache wie in [KH13; Mae+02].

Die eigentliche Ausführung von Mappings wird nicht in allen Ansätzen unterstützt, z. B. in [KH13]. Ein Vertreter, der dies berücksichtigt, ist MAFRA [Mae+02] und wird deshalb stellvertretend genauer betrachtet. Das konzeptionelle Rahmenwerk von MAFRA umfasst neben Modulen zum halbautomatischen Ontology Alignment auch die Ausführung von Abbildungen, um Instanzen der Quell- in die Zielontologie zu transformieren. Dabei wird sowohl die einmalige Ausführung als auch eine kontinuierliche Ausführung unterstützt. Letztere wird zusätzlich aktiv sobald Änderungen in der A-Box der Quell-Ontologie auftreten. *Semantic Bridges* beschreiben Abbildungen von Konzepten einer Ontologie in eine andere und werden ebenfalls semantisch modelliert, sodass sie als Individuen vorliegen. Sie können zudem komponiert und ihre Anwendbarkeit an die Erfüllung von Bedingungen geknüpft werden. Die Klasse *Service* dient dazu, Ressourcen zu referenzieren, die Transformationen durchführen können. Zum Aufruf des Dienstes werden technische Schnittstelleneigenschaften, wie Eingabeparameter, modelliert. Insgesamt zeigen sich Defizite bei der ★*Unterstützung von Kommunikationsbeziehungen*. Zwar existieren die Service-Klasse und die *Argument Mappings*, deren Zweck jedoch nicht die Beschreibung von Ausgangs- und Zielschnittstelle von Komponenten ist. Die Komposition von Semantic Bridges kann als übertragbarer Ansatz genannt werden. Die Mapping-Sprache bedient alle wesentlichen Fälle zu ★*unterstützender Heterogenität*, fokussiert jedoch auf die Beschreibung der mediierten Konzepte und weniger auf die des genauen Transformationsprozesses, welcher in dedizierten Konzepten wie *Service* und *Transformation* versteckt ist. Das Kriterium der ★*Laufzeitunterstützung* ist ebenfalls nicht zufriedenstellend erfüllt, da die Ausführung von Abbildungen nur im Falle von Schemaänderungen stattfindet und keine Performanzbetrachtungen durchgeführt wurden.

[1] Ein Gütemaß, das Präzision und Trefferquote auf Basis des harmonischen Mittels kombiniert.

3.4.2 Vertreter aus dem Bereich Webservices

In der Webservice-Domäne basierten erste Ansätze auf Schema-Matching, siehe [SE05] für einen Überblick, um Ähnlichkeiten zwischen den in Web Service Description Language (WSDL)-Beschreibungen verwendeten Extensible Markup Language (XML)-Schemata herzuleiten. Die Ergebnisse des Abgleichs können automatisch angewendet werden, zum Beispiel im Ansatz von Bleul et al. [BZG08] veranschaulicht. Allerdings unterstellt dies eine absolute Zuverlässigkeit der Zuordnungen, was – wie im Prozess des Ontology Alignment – nicht in jedem Fall gewährleistet werden kann. Als weitere Einschränkung werden keine semantikbasierten Techniken unterstützt.

Im Bereich der SWS existieren diverse Ansätze [CLB07; SPM06; Nag+07], welche auf Semantic Annotations for WSDL (SAWSDL) [@SAWSDL] aufbauen, das heißt semantisch annotieren WSDL-Beschreibungen, um Kompatibilität zu etablierten Standards zu gewährleisten und um automatisierte Mediation zu unterstützen. Als wesentliche Konzeptbestandteile kommen *Lifting* und *Lowering* zum Tragen. Hierbei werden ausgehende Daten gemäß dem jeweiligen XML-Schema eines Webservice durch das Lifting, etwa mittels XSLT, auf die semantische Ebene in semantische Individuen überführt. Anschließend finden die Mediationstechniken unter Einbeziehung von Abbildungsvorschriften statt. Letztlich erfolgt ein Lowering der resultierenden Individuen zurück auf die syntaktische Ebene in XML-Dokumente. Die Ansätze [SPM06] und [Nag+07] stellen darauf aufbauend Ausführungsplattformen für mediierte Webservicekomposition bereit. Defizite zeigen sich vor allem bei der ★*Unterstützung von Kommunikationsbeziehungen*, da N:m-Konstellationen nicht möglich sind, und bei der ★*unterstützten Heterogenität*, da weder Multi-Ontologie-Szenarien noch Projektion abdeckt werden. Bezüglich ★*Laufzeitunterstützung* sind fehlende Performanzanalysen zu bemängeln.

Andere Kompositionsansätze im Bereich SWS verwenden dedizierte ontologiebasierte Beschreibungssprachen für Webservices, wie WSMO und OWL-S. Während letztere kein eigenständiges Konzept für Mediation bereitstellt, sondern Mediatoren vielmehr als Webservices angeboten werden können, umfasst das konzeptionelle Rahmenwerk von WSMO mit den *Mediators* eine Reihe dedizierter Sprachkonstrukte für diesen Belang [@WSMO; Sto+06]. Für Datenmediation sind die Mediatoren *Ontologie-zu-Ontologie* und *Webservice-zu-Webservice* relevant. Diese werden ebenfalls als Webservices bereitgestellt und operieren vollständig auf semantischer Ebene, sodass Lifting und Lowering entfallen. Aus *Ontology Matching* gewonnene Abbildungsvorschriften sind wiederverwendbar und werden zur Laufzeit auf semantische Instanzdaten des Quell-Webservice angewendet. Dabei werden nahezu alle geforderten Fälle von ★*Heterogenität unterstützt*, nicht jedoch syntaktische Schnittstellendifferenzen und der Umgang mit Kollektionen. Die ★*Laufzeitunterstützung* zeigt Abstriche, da kein Nachweis der Performanz vorliegt. Als ★*unterstützte Kommunikationsbeziehung* kann nur die 1:1-Konstellation genannt werden.

RDFT [Ome02] stellt ein RDF-basiertes Vokabular bereit, welches es erlaubt, *Bridges* zu definieren, die sowohl Aspekte des Lifting, der Transformationen als auch des Lowering beschreiben. Konzeptionell vorgesehen sind auch 1:n- und N:1-Bridges. Weiterhin dienen Bridges zur Komposition von Webservices. Hinsichtlich ★*unterstützter Kommunikationsbeziehungen* wird theoretisch auch N:m-Kommunikation angeboten, jedoch beschreibt der Ansatz die Ausführung der Mediation nicht näher. Daher ist auch das Kriterium der ★*Laufzeitunterstützung* nicht zufriedenstellend erfüllt. Die ★*unterstützte Heterogenität* zeigt Defizite bei der Behandlung von Kollektionen und von syntaktischen Schnittstellendifferenzen.

3.4.3 Datenmediation in Mashup-Plattformen

In Kompositionsplattformen für Mashups mangelt es bisher an Ansätzen für die automatisierte Überbrückung von semantischer Heterogenität [Di +09]. Abbildungen zwischen Schnittstellenparametern sowie Funktionen zur Manipulation von Datenstrukturen wie Filterung oder Sortierung sind zwar bei vielen Vertretern als Teil der Kompositionssprache vorgesehen, z. B. [Pru07; Max+07; Liz+14]. Diese sind jedoch auf die syntaktische Ebene begrenzt und für jede Anwendung manuell zu definieren. Die Semantik ausgetauschter Daten wird bei derartigen Ansätzen daher nicht ausgenutzt.

Semantikbasierte Konzepte sehen lediglich zwei Vertreter mit deutlichen Limitierungen vor. Der Ansatz von Pietschmann et al. [PRM11b] vermag es, syntaktische Schnittstellendifferenzen wie unterschiedliche Benennung oder Reihenfolgen von Schnittstellenelementen und Parametern durch den Einsatz generierter Wrapper zu überbrücken. Als einzige semantische Technik stehen *Upcasts* bereit, die bei Subklassenbeziehungen der Parametertypen entlang des Datenflusses verwendet werden. Das Basisvorgehen umfasst ein Lifting, die Anwendung des Upcast und schließlich ein Lowering. Weitere Techniken, die über Subklassenbeziehung hinausgehende semantische Zusammenhänge ausnutzen, werden nicht unterstützt. Auch FAST-Wirecloud nutzt Ontologien zur Beschreibung von Ein- und Ausgaben von Komponenten. Darauf aufbauend erlauben es Operatoren, den Datenfluss pipeline-artig anzupassen [Liz+14]. Diese kapseln spezielle Services, die z. B. Daten aggregieren, filtern oder iterieren können. Als einzige semantikbasierte Technik wird Projektion bei der Erstellung von Datenfluss-Konnektoren angeboten, sodass unterschiedliche Granularität von Aus- und Eingabekonzepten überbrückt werden kann. Das Fazit beider Vertreter fällt wie folgt aus. Die ★*unterstützten Kommunikationsbeziehungen* sind auf 1:1-Konstellationen beschränkt und die ★*unterstützte Heterogenität* der Mappingsprache ist limitiert auf syntaktische Aspekte, Upcast und Projektion. Das Kriterium ★*Laufzeitunterstützung* wird zumindest bei [PRM11b] gut erfüllt, da auch Performanzbetrachtungen stattfinden. Allerdings liegen Mappings nicht wiederverwendbar vor, sondern sind integraler Bestandteil der jeweiligen Kompositionsbeschreibung.

3.4.4 Fazit

Insgesamt lässt sich feststellen, dass ein substantielles Defizit an Ansätzen zur automatischen Datenmediation in Mashup-Plattformen herrscht. Lediglich das in [PRM11b] vorgestellte Konzept bietet erste Ansatzpunkte, die allerdings den in Abschnitt 2.3 formulierten Anforderungen nicht gerecht werden. Ergebnisse aus dem Bereich der SWS, in Kombination mit Ontology Mediation, lassen vielversprechende Ansätze erkennen. Ungeklärt ist jedoch die Anwendbarkeit in Mashups, was insbesondere Performanzaspekte betrifft. Weitere offene Fragestellungen dieser Vertreter zeigen sich bei der Unterstützung von N:m-Kommunikationsbeziehungen sowie der Mediation auf Ebene der Syntax von Schnittstellen. Abbildung A.4 im Anhang zeigt eine genaue Aufschlüsselung der Untersuchungsergebnisse für wesentliche Vertreter zur Datenmediation.

3.5 Fazit zum Stand von Forschung und Technik

Kapitel 3 stellt den aktuellen Stand von Forschung und Technik in den relevanten Forschungsbereichen dar. Den Schwerpunkt bildet dabei die Untersuchung von Kompositions-

plattformen für das EUD. Wie deutlich wurde, bieten neuere Vertreter vielversprechende Ansätze hinsichtlich der ★*Entwicklungsmethodik* und des ★*Abstraktionsgrades*. Insbesondere Konzepte der »Live-Entwicklung« in NaturalMash sowie OMELETTE und die Verwendung fachlicher Abstraktion der Kompositionslogik wie in DEMISA sowie SOA4All liefern substantielle Anregungen für den eigenen Ansatz.

Hinsichtlich der ★*Unterstützung beim Einstieg und bei der Suche* zeigen sich in den meisten Vertretern deutliche Defizite. Hier mangelt es an verschiedenartigen Einstiegspunkten, an geeigneten Möglichkeiten funktionale Anforderungen zu formulieren und an explorativen Ansätzen für Nutzer mit vagem Informationsbedürfnis. Daher wurden hinsichtlich dieses Aspekts in Abschnitt 3.3 weitere Lösungsmöglichkeiten vorgestellt. Insgesamt erscheinen eine Kombination und eine kontextsensitive Zugänglichkeit mehrerer Ansätze, wie verschiedenen Direkteinstiegspunkten auf einer Art Dashboard, einer facettierten erweiterten Suche sowie einem assistierendem Wizard vielversprechend.

Die ★*Unterstützung bei der Komposition* zeigt ebenfalls Defizite. Zumeist verwenden Kompositionsplattformen eine Variante der Metapher des »Verdrahtens«. Diese erlaubt zwar die gezielte, feingranulare Anpassung der Kompositionslogik, kann allerdings für Nicht-Programmierer mit weniger technischem Hintergrund eine Hürde darstellen. Schwächen zeigen sich bei der Sicherstellung einer korrekten Komposition. Zwar bieten die meisten Vertreter Fehlervermeidung durch Typprüfung oder basieren auf Kompositionsmustern. Jedoch fehlt es an Ansätzen, die den Aufgaben- und Kompositionskontext einbeziehen. Hier zeigen Planungsalgorithmen wie in iMashup und DEMISA Potenzial, allerdings mangelt es auch hier an Konzepten wie Nutzer die Korrektheit einschätzen können. Als weiteres Defizit muss ein Fehlen an Mechanismen zur semantischen Datenmediation genannt werden. Keiner der untersuchten Mashup-Ansätze vermag es, die gestellten Anforderungen zu erfüllen. Diesbezüglich zeigen jedoch die Ergebnisse aus dem Bereich SWS hohes Potenzial, insbesondere *WSMO Mediators* [Sto+06]. Einige Vertreter bieten Automatisierung von Kompositionsaktionen, beispielsweise auf Basis von ★*Empfehlungssystemen*, und Kompositionswissen an, zum Beispiel in Form von Vorlagen oder Kompositionsmustern. Die Untersuchung in Abschnitt 3.2 legt dar, dass nur wenige Plattformen umfassende, flexibel einsetzbare Empfehlungssysteme bereitstellen, zum Beispiel SMASHAKER [BDM13a]. Trotzdem zeigen die vorgestellten Konzepte wesentliche Defizite. Es fehlt zum Beispiel eine generische, konfigurierbare Architektur und eine endnutzergerechte Visualisierung von Empfehlungen zu Kompositionsschritten.

Als besonders defizitär erweist sich die ★*Unterstützung bei Nutzen und Verstehen* von kompositen Webanwendungen. Zwar bieten die meisten Plattformen zumindest externe Hilfen an, etwa Einführungen und Anleitungen. Wie in Abschnitt 3.1 verdeutlicht wurde, sind extrinsische und intrinsische Ansätze jedoch nicht vorhanden oder nur unzureichend. Hinsichtlich der Erklärung von Kommunikationsverbindungen kann bereits das genutzte Kompositionsparadigma einen Ansatz liefern. Oftmals kommt dabei eine Ableitung von *Wenn-Dann* zum Einsatz, wie in IFTTT [@IFTTT], Apiant [@Apiant], PEUDOM [Pic13] und *NaturalMash* [AP14]. In simplen Szenarien scheint die erzeugte Kompositionslogik geeignet, um Nutzern im Nachhinein die Funktionsweise verständlich zu machen. Grafisch setzen derartige Ansätze zumeist auf eine pfeilbasierte Darstellung, wie [Ghi+16; Tsc+14]. Wenige Vertreter sehen generierte Anleitungen [Voi14] oder Methoden zur Präsentation von auftretendem Datentransfer zur Laufzeit [Tsc+14] vor. Es empfiehlt sich daher diese Konzepte aufzugreifen. Wegen der vorhandenen Defizite gilt es allerdings, substantielle Erweiterungen vorzunehmen.

Die vorangegangenen Ausführungen betonen und detaillieren die in Kapitel 1.1 herausgearbeiteten Problemfelder und Herausforderungen, welchen sich diese Dissertation widmet. Wie insgesamt deutlich erkennbar wird, existieren für diverse Fragestellungen punktuelle Teillösungen. Ein ganzheitlicher Ansatz fehlt allerdings. Daher wird es im weiteren Verlauf das Ziel sein, geeignete vorhandene Konzepte aufzugreifen und neuartige Ansätzen zu schaffen, um die Vision und Anforderungen der Arbeit zu verwirklichen. Nachfolgend steht ein systematischer Überblick des Gesamtkonzepts zum assistierten EUD von kompositen Webanwendungen durch Nicht-Programmierer im Mittelpunkt der Betrachtung. Dabei werden sowohl die Forschungsschwerpunkte als auch die wesentlichen neuen Konzepte des Ansatzes betont.

4

Assistiertes EUD von CWA durch Nicht-Programmierer

In den bisherigen Kapiteln wurden Herausforderungen identifiziert, denen Nicht-Programmierer beim EUD kompositer Webanwendungen potentiell gegenüberstehen. Weiterhin konnten aus existierenden Ansätzen gewonnene Erkenntnisse und erkannte Defizite aufgezeigt werden. Zur Lösung der genannten Herausforderungen wird in dieser Arbeit ein neuer, umfassender Ansatz für Nicht-Programmierer zur Ad-hoc-Entwicklung und Nutzung von kompositen Webanwendungen vorgeschlagen. Dieser zeichnet sich unter anderem wie folgt aus [RM18]:

Iterative Entwicklung: Ein hochiteratives Vorgehen mit kontinuierlicher Assistenz und sofortigem Feedback zu Kompositionsschritten wird angeboten.

Fachliche Abstraktion: Technische Details und Terminologie werden vom Nutzer ferngehalten. Stattdessen erfolgt die Kommunikation anhand des fachlichen Vokabulars, repräsentiert durch Ontologien und Capabilities.

Assistierter Einstieg: Verschiedene Werkzeuge zum Zugriff auf und zur explorativen Suchen von Kompositionsfragmenten sind verfügbar.

Assistierte Komposition: Als Kompositionsmetapher gilt das Verknüpfen von Komponenten anhand von Capabilities, die fachliche Aufgaben und Begriffe repräsentieren. Automatisierung und semantische Datenmediation helfen Nutzern zusätzlich.

Empfehlungssystem: Nutzer werden prozessbegleitend durch kontextsensitive Empfehlungen, die sowohl explizit angefordert als auch proaktiv angeboten werden, unterstützt. Empfehlungsstrategien dienen der bedarfsgerechten Maßschneiderung des Empfehlungssystems.

Assistiertes Nutzen und Verstehen: Spezielle Werkzeuge erlauben es Nutzern, die Funktionsweise von Mashups zu explorieren, anhand von Tutorien zu verstehen und sich des funktionalen Zusammenspiels von Komponenten bewusst zu werden.

Im Mittelpunkt dieses Kapitels steht die überblicksartige Beschreibung des Gesamtansatzes für das assistierte EUD für Nicht-Programmierer [RM18]. Dazu wird das vorgeschlagene Vorgehensmodell vorgestellt, ein Überblick zugrundeliegender Modelle sowie wesentlicher Assistenzmechanismen gegeben und schließlich grobe Architekturkomponenten einer entsprechenden Kompositionsplattform eingeführt.

4.1 Assistiertes EUD von Mashups

Abbildung 4.1 veranschaulicht den Gesamtansatz des assistierten EUD von kompositen Webanwendungen durch Nicht-Programmierer. Wie darin zu erkennen ist, umfasst das Konzept eine Ebene vorrangig semantischer Modelle, auf deren Grundlage eine Reihe von Algorithmen und Funktionsblöcken angesiedelt sind. Beide bilden das Fundament für Werkzeuge und Assistenten, gemäß dem Vorgehensmodell der *Live-Sophistication*.

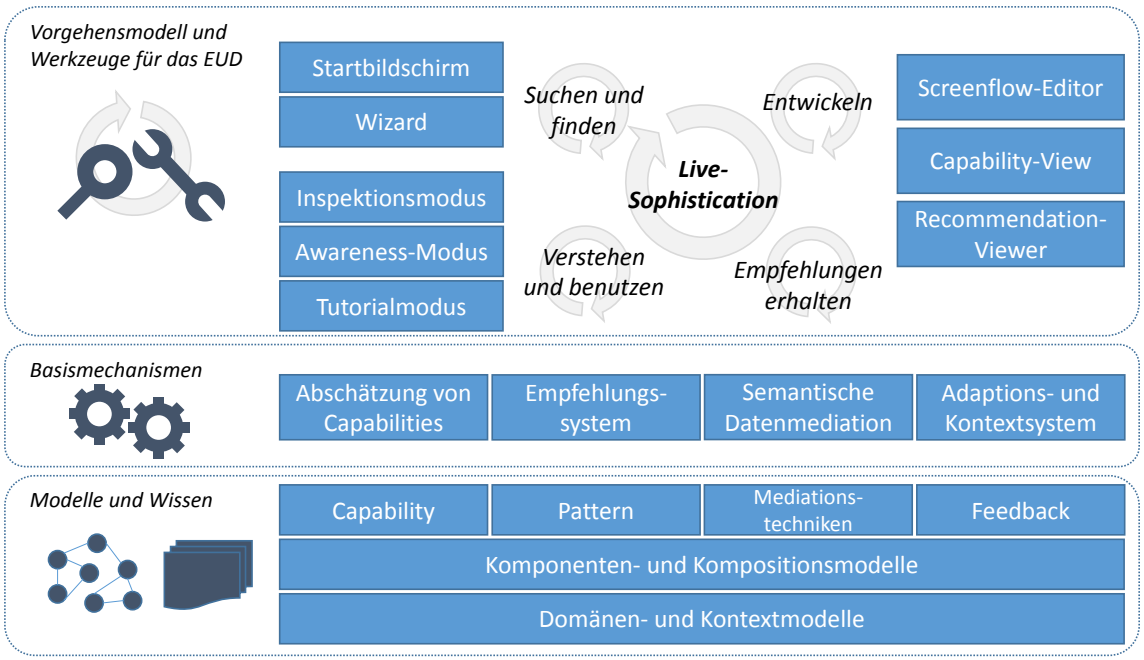


Abbildung 4.1: Überblick des geschichteten Aufbaus der Konzepte für das assistierte EUD von kompositen Webanwendungen durch Nicht-Programmierer

Wie im Rahmen der Problemanalyse in Kapitel 1.1 beschrieben, erschwert eine Trennung von Entwicklung und Nutzung einer Anwendung das EUD für Nicht-Programmierer deutlich, da diese stark iterativ bei der Entwicklung vorgehen. Um Anforderung 1 gerecht zu werden, liegt dem Ansatz die Live-Sophistication [Rüm+11] zugrunde. Dabei findet ein Verschmelzen der Nutzung und der Entwicklung von CWA statt. Ein MRE liefert die notwendigen Entwicklungswerkzeuge direkt mit, sodass Nicht-Programmierer an einer in Ausführung befindlichen Anwendung Änderungen vornehmen können und somit sofortiges Feedback zu durchgeführten Kompositionsschritten erhalten. Gemäß der Live-Sophistication führen Nutzer mehrere Aktivitäten in hochiterativer Weise durch, vergleiche auch Abbildung 4.1. Diese im Folgenden vorgestellten Aktivitäten sind größtenteils optional, untereinander stark durch Transitionen vernetzt, das heißt nicht linear geordnet, und in sich iterativ.

Suchen passender Kompositionsfragmente: Hierbei gilt es, Nutzer dabei zu unterstützten, möglichst einfach und zielgerichtet jene Kompositionsfragmente zu finden, welche ihren fachlichen Anforderungen und Vorstellungen genügen. Diese werden dann typischerweise zum *Benutzen* in eine CWA integriert.

Benutzen: Nachdem eine komposite Webanwendung gestartet wurde, interagiert der Nutzer mit den UI-Komponenten der Anwendung, um seine fachlichen Aufgaben zu bewältigen. Wird die aktuelle CWA beendet, beispielsweise weil sie nicht den Anforderungen entspricht oder der Nutzer seine Aufgabe erledigt hat, kann eine Transition zum *Entwickeln* oder zum *Suchen* stattfinden.

Untersuchen: Es kann vorkommen, dass der Nutzer die Funktionsweise der aktuellen CWA, einzelner Komponenten oder empfohlener Kompositionsschritte nicht nachvollziehen kann. Dann werden ihm geeignete Werkzeuge zur Verfügung gestellt, mit denen er Zweck und Funktionsweise zum besseren Verständnis untersuchen kann. Danach *benutzt* ein Nicht-Programmierer die CWA oder erkennt Anpassungsbedarf, sodass er zum *Entwickeln* übergeht oder *Empfehlungen anfordert*.

Entwickeln: Das Anpassen einer CWA kann direkt durch Nutzerhandlungen induziert werden, etwa durch Entfernen oder Hinzufügen von Komponenten und Kommunikationskanälen. Die korrekte Umsetzung eines Kompositionsschrittes übernimmt das MRE. Zur Reflektion der Auswirkungen eines vorgenommenen Entwicklungsschrittes kann ein Nutzer in die Aktivität *Benutzen* oder *Untersuchen* übergehen. Zudem finden Übergänge zum *Suchen* und zum *Empfehlungen erhalten* statt.

Empfehlungen erhalten: Nutzer können explizit Empfehlungen anfordern, etwa zu sinnvollen Anwendungserweiterungen oder zu Alternativkomponenten. Daneben werden auf Systeminitiative, das heißt ohne explizite Aufforderung durch den Nutzer, Empfehlungen unterbreitet, die ein Nicht-Programmierer annehmen kann.

Diese im Vorgehensmodell enthaltenen Aktivitäten von Nicht-Programmierern sind generischer Natur und in dieser Form prinzipiell auf andere EUD-Plattformen übertragbar. Erst die Feinspezifikation von Aktivitäten lässt Eigenheiten von CWA deutlich werden, zum Beispiel hinsichtlich Komponentenorientierung. Auch Art und Umfang resultierender Werkzeuge, Basismechanismen sowie Modelle, welche nachfolgend skizziert werden, sind spezifisch, insbesondere im Hinblick auf die Gruppe der Nicht-Programmierer.

4.1.1 Modellebene

Die Basisschicht des Konzepts bilden verschiedene Metamodelle, die in Kapitel 5.1 genauer vorgestellt werden. Gemäß den Grundlagen aus Abschnitt 2.1 liegen deklarative *Komponenten- und Kompositionsmodelle* zugrunde, stellvertretend in Ausprägung der CRUISE-Konzepte. Daneben dienen semantische Modelle von Domänen und des Nutzerkontextes zur semantischen Annotation von *Kompositionsfragmenten*, sowohl bezüglich der ein- und ausgehenden Daten als auch ihrer funktionalen und qualitativen Eigenschaften. Das *Capability-Metamodell* erlaubt die hierarchische Beschreibung der funktionalen Fähigkeiten von Kompositionsfragmenten. Basierend auf Erkenntnissen aus dem Bereich

der Aufgabenmodellierung werden dazu Tupel aus einer Aktivität und einem Domänenobjekt definiert, wie »Hotels suchen«. Zudem können Capabilities mit zugehörigen Interaktionsschritten und deren Bezug zum Komponenten-UI ausgestattet werden.

Kompositionswissen wird durch mehrere komplementäre Aspekte repräsentiert. Sogenannte *Patterns* modellieren bedeutsame oder statistisch signifikante Kompositionsfragmente. Neben diesem vorrangig strukturellen Wissen, bieten Domänenmodelle semantische Zusammenhänge von Fachbegriffen. Sie werden verwendet, um die Capabilities von Kompositionsfragmenten abzuleiten sowie miteinander in Relation zu setzen und um Patterns zu abstrahieren und nach Relevanz zu gewichten. Als dritte Säule dient eine modellhafte Repräsentation von explizitem und implizitem *Feedback*, das mit Kompositionsfragmenten sowie dem Feedback-Kontext verknüpft wird, um community-basierte Techniken zum Validieren von Kompositionswissen zu unterstützen. Schließlich beschreiben *Mediationstechniken* auflösbare Schnittstelleninkompatibilitäten und deren semantikbasierte Überbrückung, wobei wiederum Ontologien eine tragende Rolle spielen.

4.1.2 Basismechanismen

Das Konzept des assistierten EUD von CWA sieht verschiedene Basismechanismen vor. Diese bauen auf die zuvor skizzierten Modelle auf, bieten die notwendige Funktionalität für Werkzeuge und Assistenten und sind Nutzern typischerweise komplett verborgen.

Datenmediation: Mediationstechniken ermöglichen es diesem Funktionsblock, unterschiedlich typisierte Signaturen von Schnittstellenelementen aufeinander abzubilden, sofern semantisch möglich. Zum Berechnen von Abbildungsvorschriften, welche Mediationstechniken einbeziehen, wird maßgeblich auf Ontologiewissen zurückgegriffen. Zur Ausführungszeit einer CWA sorgt semantische Datenmediation für das Anwenden der Abbildungsvorschriften auf zwischen Komponenten auszutauschende Nachrichten. Das Konzept wird in Abschnitt 5.2 im Detail vorgestellt.

Abschätzung von Capabilities: Aus den Capabilities von Komponenten und aus Kommunikationsbeziehungen in einem Kompositionsmodell schätzt dieser Algorithmus übergeordnete, potentiell komposite Capabilities beliebiger Kompositionsfragmente ab. Der Ansatz bildet den Schwerpunkt von Abschnitt 5.3.

Empfehlungssystem: Zur Nutzerunterstützung werden Empfehlungen zu Kompositionsschritten bereitgestellt, siehe Kapitel 6. Als Neuheit kann das Empfehlungssystem mit *Empfehlungsstrategien* ausgestattet werden, die konfigurieren, wann welche Art von Empfehlungen unterbreitet und wie diese dargestellt werden sollen. Zuständig für diese Aspekte sind *Trigger* (definieren Bedingungen), *Empfehlungsmethoden* (legen den Typ vorzuschlagender Kompositionsfragmente fest und führen Matching und Ranking durch) und *Recommendation-Viewer* (stellen Empfehlungen dar und sammeln Feedback). Empfehlungsmethoden stützen sich auf Patterns, Kompositionsfragmente sowie Feedback. Wird eine Empfehlung vom Nutzer akzeptiert, erfolgt deren automatische Integration in eine CWA.

Adaptions- und Kontextsystem: Dieses dient dem Verwalten semantischer Kontextmodelle und dem geordneten Anpassen eines Mashups zur Durchführung von Kompositionsschritten, zum Beispiel bei der Integration von Patterns.

4.1.3 Werkzeuge

Werkzeuge für das EUD bilden die Schnittstelle zum Nutzer und setzen auf Modelle und Basismechanismen auf, um die obengenannten Aktivitäten zu unterstützen. Allen Werkzeugen ist gemein, dass zur Kommunikation mit Nutzern fachliche, natürlichsprachliche Texte, die aus Capabilities generiert werden, zum Einsatz kommen. Detaillierte Ausführungen zu den Werkzeugen befinden sich in Kapitel 7.

Als wichtiges Assistenzmerkmal hilft eine Mashup-Plattform Nicht-Programmierern dabei, in den Kompositionsprozess einzusteigen. Ein personalisierter **Startbildschirm** offeriert hierzu mehrere Einstiegspunkte. Verschiedene Auflistungen, zum Beispiel anhand der Relevanz für den Nutzer und anhand inhaltlicher Kategorien, erlauben es, schnell auf Komponenten und Anwendungen zuzugreifen diese zu explorieren. Zudem stehen gängige Suchmöglichkeiten zur Verfügung, wie eine Stichwortsuche, ein Facettenbrowser und ein **Wizard**. Letzterer unterstützt Nicht-Programmierer bei der Eingabe und der Dekomposition funktionaler Anforderungen und Ziele durch einen geführten Dialog.

Ein MRE ist mit Kompositionswerkzeugen ausgestattet. Es stehen mehrere Ansichten auf ein Mashup bereit: Die *Live-View* stellt ausschließlich Komponenten-UIs gemäß dem Anwendungslayout dar. Sie dient dem Benutzen von CWA und versteckt nahezu sämtliche technische Details. Weitere Ansichten können der Live-View überlagert werden und somit, passend zu den Erfahrungen und Fertigkeiten der Zielgruppe, Details zur zugrundeliegenden Komposition preisgeben, wie die Schnittstellen von Komponenten und deren Verknüpfungen. Die **Capability-View (CapView)** lässt die Komposition von Komponenten für den Nutzer als Verknüpfung von Capabilities erscheinen und Kausalsätze veranschaulichen die Funktionalität von Komponenten und deren Zusammenspiel.

Nutzer erhalten prozessbegleitend Empfehlungen. Als Schnittstelle zum Nutzer dienen **Recommendation-Viewer**, die sowohl dediziert bereitgestellt werden, zum Beispiel als ein *Empfehlungsmenü*, als auch integraler Bestandteil anderer Werkzeuge, wie der CapView, sein können. Empfehlungen werden anhand von Capabilities, die sie zur Anwendung beitragen, angezeigt. Der Nutzer bekommt die Möglichkeit, explizit **Feedback** zur Qualität von Komponenten, Anwendungen sowie angenommenen Vorschlägen hinsichtlich mehrerer Kriterien zu geben. Ein MRE bietet zu diesem Zweck entsprechende UI-Dialoge an. Daneben sammeln Recommendation-Viewer implizites Feedback.

Erklärungstechniken helfen beim Untersuchen und beim Nutzen von CWA. Der **Inspektionsmodus** erlaubt die Exploration der Capabilities von Komponenten und Anwendungen direkt im UI. Schrittweise interaktive Tutorien generiert und präsentiert der **Tutorialmodus**. Zur Schaffung eines Bewusstsein für auftretenden Datenaustausch zwischen Komponenten dient der **Awareness-Modus**.

4.2 Grobarchitektur

Ein wesentlicher Bestandteil einer Plattform für Live-Sophistication ist eine *Laufzeitumgebung (MRE)* gemäß der CRUISE-Referenzarchitektur aus Kapitel 2.1.3. Das Konzept wird im Rahmen dieser Dissertation aufgegriffen und um zahlreiche Architekturkomponenten erweitert. Wie in Abbildung 4.2 verdeutlicht, stützt sich ein MRE auf externe Repositorien, die zum Beispiel SMCDL-Dokumente (*Komponentenrepositorium*) und Kompositionsmodelle (*Anwendungsrepositorium*) verwalten. Ein *Adaptionssystem* erlaubt die Anpassung einer CWA und das *Kontextsystem* bietet lesenden und schreibenden

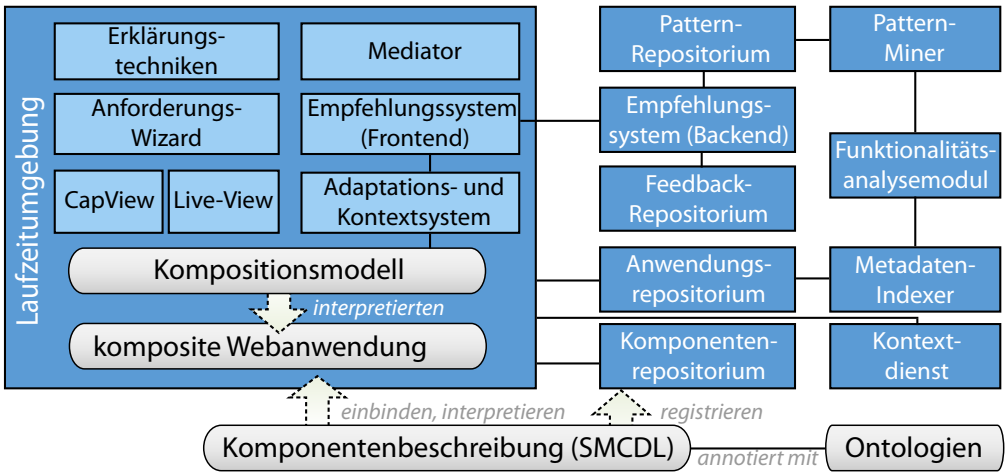


Abbildung 4.2: Grobarchitektur einer Kompositionsplattform für assistiertes EUD

Zugriff auf Kontextmodelle, welche im *Kontextdienst* verwaltet werden. Der *Mediator* setzt die zuvor genannte semantische Datenmediation um. Wie in Kapitel 5.2 vertieft wird, ist er in der Lage, Abbildungsvorschriften zur Laufzeit auf Instanzdaten in Kommunikationsnachrichten zwischen Komponenten anzuwenden. Damit wird er vom Event-Bus beauftragt, sofern ein Kommunikationskanal eine Abbildungsvorschrift aufweist.

Die Architektur umfasst ein *Empfehlungssystem*, welches im Detail in Kapitel 6 behandelt wird. Wie in Abbildung 4.2 zu erkennen ist, wird es auf ein Frontend pro Laufzeitumgebung und auf ein MRE-übergreifendes Backend verteilt. Einem Frontend obliegt es, festzustellen, wann Empfehlungen notwendig sind, passende Anfragen an das Backend zu stellen und erhaltene Empfehlungen zu visualisieren. Dazu werden *Trigger* als Softwaremodule konzipiert, mit welchen ein MRE gemäß deklarativen Beschreibungen von Empfehlungsstrategien ausgestattet werden kann. Das Backend des Empfehlungssystems verfügt über verschiedene Empfehlungsmethoden, die sich auf Repositorien wie das *Pattern-Repositorium* stützen und Matching und Ranking unter Einbeziehung des *Kontextdienstes* und des *Feedback-Repositoriums* durchführen. Das *Adaptionssystem* trägt Verantwortung für das Einbinden vorgeschlagener Kompositionsfragmente in ein Mashup. *Pattern-Miner* kapseln Mechanismen zum Ableiten von Pattern-Instanzen auf Grundlage semantikbasierter oder statistischer Algorithmen. Eng gekoppelt mit dem Empfehlungssystem sind Mechanismen zum Erfassen von Nutzerfeedback zur Qualität von Komponenten, Anwendungen und Empfehlungen. Ein MRE bietet dafür Eingabedialoge sowie Tracking-Mechanismen und übermittelt die Werte an das *Feedback-Repositorium*. Das *Funktionalitätsanalysemodul* stellt den Algorithmus zum Abschätzen der Capabilities für beliebige Kompositionsfragmente bereit [RBM16; RBM17]. Er wird von Pattern-Minern sowie einem *Metadaten-Indexer*, der CWA analysiert und extrahierte Metadaten indexiert, benutzt. Zusätzlich obliegt dem MRE das Bereitstellen von EUD-Werkzeugen, wie der Capability-View, den Erklärungstechniken, dem Wizard und dem Startbildschirm.

Die vorgeschlagene Erweiterung der CRUISE-Architektur zeichnet sich durch hochgradige Trennung von Belangen aus. Insbesondere das Auslagern von Modulen aus der Laufzeitumgebung erlaubt deren MRE-übergreifende Wiederverwendung.

Die hier skizzierten Modelle, Algorithmen und Architekturkomponenten für das assistierte EUD von CWA werden in den nachfolgenden Kapiteln sukzessive detailliert.

5 Basiskonzepte

Nachdem der Gesamtansatz in Kapitel 4 überblicksartig vorgestellt wurde, ist dieses Kapitel der Beschreibung von grundlegenden architektonischen und algorithmischen Konzepten des Ansatzes gewidmet. Hierzu werden Modelle in Kapitel 5.1, semantikbasierte Techniken zur Datenmediation in Kapitel 5.2 und schließlich der Algorithmus zum Abschätzen der Capabilities eines Kompositionsfragments in Kapitel 5.3 behandelt.

5.1 Grundlegende Modelle

Die Basis der geschichteten Struktur des Gesamtkonzepts, siehe Abbildung 4.1, bilden verschiedene Metamodelle, zum Beispiel zur Modellierung von Fähigkeiten und zur Beschreibung von Kompositionsmustern. Diese werden daher in diesem Abschnitt definiert.

5.1.1 Capability-Metamodell

Capabilities [RBM13; RBM16] beschreiben die funktionale Semantik von Kompositionsfragmenten, das heißt von Komponenten und darauf aufbauend von Anwendungen sowie Pattern-Instanzen. Es kann demnach formal definiert werden, was ein Komponente leisten kann und welche Funktionalität sie bereitstellt, etwa das Darstellen von Orten und das Suchen sowie Anzeigen von Hotels. Um dies zu erreichen und die Hürde der Annotation für Komponentenentwickler nicht zu hoch ausfallen zu lassen, sind Capabilities im Wesentlichen Tupel aus *Activity* und *Entity*. Damit wird ausgedrückt, welche Aktivität oder Aufgabe auf oder mit welchem Domänenobjekt durchgeführt wird. Referenzen zu Konzepten aus Ontologien, die in der Web Ontology Language (OWL) [@OWL] formalisiert sind, wie Klassen, Instanzen und Properties, hinterlegen Capabilities mit formaler Semantik. Capabilities weisen folgende **Kernattribute** auf, vergleiche Abbildung 5.1.

Activity: Die Aktivität oder Aktion, wie Filtern, Suchen, Darstellen. In der entwickelten Ontologie existieren drei zentrale Klassen, die Aktivitäten gemäß einem systemtheoretischen Ansatz gruppieren [Tie+13]: `Input`, `Manipulate`, `Output`.

Activity-Modifier: Dieser präzisiert optional die Beschreibung der Aktivität, wodurch vermieden wird, viele Individuen oder Unterklassen in Ontologien definieren zu

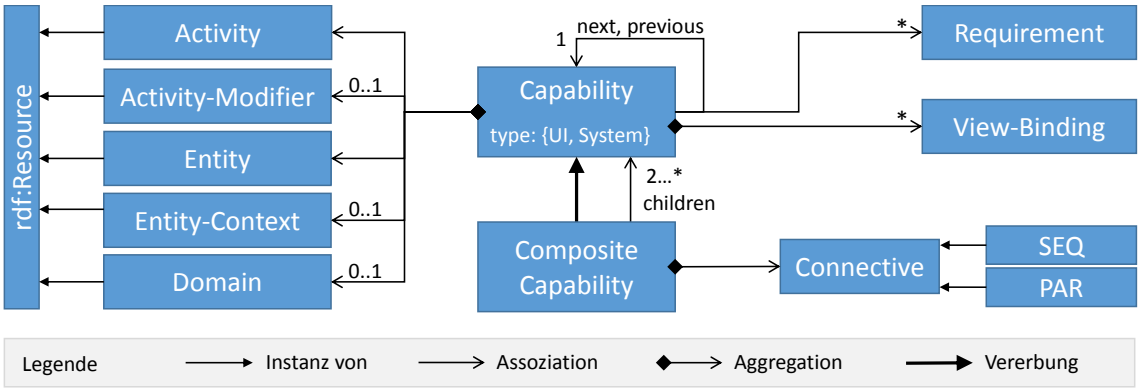


Abbildung 5.1: Capability-Metamodell: Schematischer Überblick

müssen. Als ein Beispiel soll `Sort by name` dienen. Dies kann entweder direkt in einer Ontologie als Instanz oder Subklasse von `Sort` beschrieben und als *Activity* genutzt werden. Oder es wird das generische Konzept `Sort` als *Activity* der Capability annotiert und die Spezialisierung erfolgt über den *Activity-Modifier*, zum Beispiel durch Referenz auf eine OWL-Property `hasName`. Verglichen mit einer alleinigen Nutzung der *Activity*, mag die Interpretierbarkeit bei diesem Ansatz zwar fallweise geringer sein und es existieren Seiteneffekte auf Matching-Algorithmen, die primär auf Subsumption von Klassen setzen, um Ähnlichkeiten zu bestimmen. Aber dafür kann ein »Aufblähen« von Ontologien vermieden werden.

Entity: Das Domänenobjekt auf dem oder mit dem eine Aktivität ausgeführt wird, beispielsweise `Movie`, `Person`, `hasStartLocation`.

Entity-Context: Analog zum Activity-Modifier dient der optionale *Entity-Context* zur genaueren Adressierung der Entität. Wichtigster Anwendungsfall ist die Angabe des Wertebereichs, falls es sich bei der Entität um eine Property gemäß Resource Description Framework Schema (RDFS) [@RDFS] handelt. Ist zum Beispiel eine RDFS-Property `hasName` als Entity annotiert, kann mittels Entity-Context klargestellt werden, dass es sich um den Namen eines `Hotels` und nicht um den einer `Person` handelt.

UI- und System-Capabilities: Um eine durch die Capability repräsentierte Funktionalität bereitzustellen, ist möglicherweise Nutzerpartizipation durch Interaktion mit dem Komponenten-UI notwendig. Diesen Sachverhalt reflektiert in Abbildung 5.1 das Attribut *type*, das mit *UI* und *System* belegt werden kann.

Domain: Dies beschreibt optional, in welche Domäne eine Capability eingeordnet werden kann. Zum Beispiel entstammt `Search Route` der Domäne »Reiseplanung«. Mit diesem Attribut können Capabilities semantisch klassifiziert werden, was sowohl ein übersichtliches Auflisten als auch ein eingegrenztes Suchen erleichtert.

Unter Verwendung des Entwurfsmusters *Kompositum* [Gam+94] ist es weiterhin möglich, komposite Capabilities und somit Eltern-Kind-Beziehungen zu beschreiben. Auf Modellebene steht dafür die Unterklasse *Composite Capability* bereit. Durch die Assoziation *children* zur Klasse Capability werden mindestens zwei Kinder angegeben und

es können rekursiv hierarchische Strukturen gebildet werden. In welcher Relation die Capabilities einer Kinderebene zueinander stehen, spezifiziert das *Connective*. Dabei werden sequentielle (`SEQ`) und parallele Abfolgen (`PAR`) unterstützt. Angelehnt an Aufgabenmodelle erlaubt es dieses Instrument beispielsweise, auszusagen, dass eine Capability `Search route` aus den Capabilities `Select start location`, `Select destination location`, `Search route` und `Display route` komponiert ist. Im Fall von Sequenzen können darüber hinaus Capabilities entsprechend ihrer Kausalitätsbeziehung verkettet werden. Die Referenzierung erfolgt dabei mittels der Assoziationen *next* und *previous*.

Die optionale Relation zwischen Capabilities und *Requirements* erlaubt es, die Erbringung von Fähigkeiten an Bedingungen im Bezug auf den Nutzer-, Geräte- und aktuellen Ausführungskontext zu koppeln. Die Semantik ist in einem solchen Fall wie folgt definiert: Die Capability kann erbracht werden, wenn alle Anforderungen erfüllt sind. Dies spielt insbesondere bei der Suche nach Kompositionsfragmenten anhand funktionaler Anforderungen und Zielen eine Rolle. Als Beispiel sei die Capability `Take Picture` einer Komponente genannt, deren Bereitstellung an die Nutzbarkeit einer Kamera im jeweiligen Laufzeitumgebungskontext gebunden ist.

Durch Verkettung von Capabilities und Kopplung an Kontextbedingungen gelingt eine begrenzte Art der Beschreibung der Verhaltenssemantik von Kompositionsfragmenten.

Ein weiteres, speziell für UI-Capabilities relevantes Konzept sind sogenannte *View-Bindings*. Sie stellen die Verbindung zwischen Capabilities und der grafischen Benutzerschnittstelle der jeweiligen Komponente dar. Dazu werden UI-Elemente unter Zuhilfenahme von Selektoren, wie zum Beispiel aus Cascading Style Sheet (CSS) bekannt, referenziert und notwendige Interaktionsschritte angegeben. Letztere werden anhand von Operationen beschrieben, welche atomar oder komposit (sequentielle oder parallele Gruppierung atomarer Operationen) ausgeprägt sein können. Mehrere View-Bindings einer Capability sind als Alternativen anzusehen. Zum Beispiel kann definiert werden, dass ein Nutzer für die Capability `Select Location` einen Marker auf der Karte per Drag-and-drop verschieben muss oder eine Tastatureingabe in ein Textfeld nötig ist.

Das vorgestellte Modell basiert in Teilen auf ähnlichen Arbeiten. Zur inhaltlichen Verschlagwortung von Ressourcen wurden *Purpose-Tags* [Str08] und *Intent-Tags* [KKS10] vorgestellt. Diese repräsentieren Aspekte des Vorhabens des Nutzers. Sie beschreiben den Kontext, in dem eine Ressource genutzt werden kann, und die Ziele und Absichten eines Nutzers. Capabilities verfolgen ein ähnliches Ziel: das Beschreiben von Aufgaben, für die ein Kompositionsfragment verwendet werden kann. Allerdings sind Capabilities durch den Annotationsansatz mit formaler Semantik ausgestattet. *Aufgabenmodelle* dienen der formalen Repräsentation von Aufgaben. Sie erlauben es grobgranulare Aufgaben in zunehmend detailliertere zu dekomponieren und somit hierarchische Strukturen zu beschreiben. Neben klassischen Ansätzen, siehe [Tie+13; Tie15] für einen Überblick, wurde eine Kombination mit semantischen Technologien vorgeschlagen, um Schwächen bei der Domänenmodellierung und der semantischen Klarheit zu begegnen [Tie15]. Das semantische Aufgabenmodell von DEMISA (vergleiche Abschnitt 3.1) sieht dazu vor, dass Aktionen, Ein- und Ausgaben sowie Rollen von Aufgaben anhand von Ontologiekonzepten formal definiert werden. Das Capability-Metamodell ist stark von diesem Ansatz beeinflusst. Allerdings wurden Konstrukte auf eine notwendige Mindestmenge begrenzt, wodurch die Ausdrucksmächtigkeit eingeschränkt sein mag, die Kompliziertheit für Algorithmen und den Entwickler jedoch reduziert wird. Zudem werden mashup-spezifische Anforderungen berücksichtigt, was sich insbesondere in den View-Bindings äußert.

Ein wesentlicher Grundgedanke bei der Einführung des Capability-Modells ist, dass Komponenten typischerweise zur Erfüllung von Aufgaben (Capabilities) dienen, und dass ein Verbund von Komponenten entsprechend komplexe Aufgaben (komposite Capabilities) erledigen kann. Dazu benötigen Komponenten Eingaben und produzieren Ausgaben (mit Capabilities assoziierte Schnittstellenelemente). Die Capabilities eines Kompositionsfragments ergeben sich demnach maßgeblich aus denen der verknüpften Komponenten. Letztere sind daher von besonderer Bedeutung für den Ansatz zum Abschätzen der Capabilities von Kompositionsfragmenten, um treffende Ableitungen vorzunehmen.

5.1.2 Erweiterungen von Komponentenmodell und SMCDL

In diesem Abschnitt wird exemplarisch aufgezeigt, wie die im vorherigen Kapitel vorgeschlagenen Konzepte in ein Modell und eine Beschreibungssprache für Komponenten gemäß universeller Komposition eingefügt werden können. Hierzu dienen die in Kapitel 2.1.1 analysierten Konzepte als Grundlage. Zunächst wurden neben den bekannten Abstraktionen des CRUISE-Komponentenmodells zwei neue hinzugefügt:

Capabilities: Von einer Komponente typischerweise unter Nutzereinwirkung erbrachte Fähigkeiten. Sie ersetzen die bisherigen Mittel zur Angabe funktionaler Semantik.

Anforderungen: Komponenten können Anforderungen an den Ausführungskontext stellen, beispielsweise an Eigenschaften des Nutzers und der zugrundeliegenden Plattform. Dadurch kann der Heterogenität des Kontextes beim Matching und Ranking von Kompositionsfragmenten genauer Rechnung getragen werden.

Wie Abbildung 5.2 verdeutlicht, können Capabilities in Bezug auf Komponenten an zwei Stellen vorkommen. Erstens können Capabilities auf Ebene der gesamten Komponente platziert werden. Neben UI-Capabilities, wie `Display Location`, betrifft dies auch System-Capabilities, die nicht ausschließlich von Schnittstellenelementen abhängen, zum Beispiel `Search Route` einer Komponente, die Eingabemöglichkeiten für die Suchparameter auf ihrem UI anbietet. Zweitens sind System-Capabilities, die exklusiv bei Aufruf einer Operation oder beim Setzen einer Property erbracht werden, direkt an den zugehörigen Schnittstellenbestandteilen annotiert.

Als konkrete Syntax zur Beschreibung von Komponenten dient auch im Rahmen dieser Arbeit die SMCDL. Die Grundstruktur veranschaulicht Abbildung 5.2.

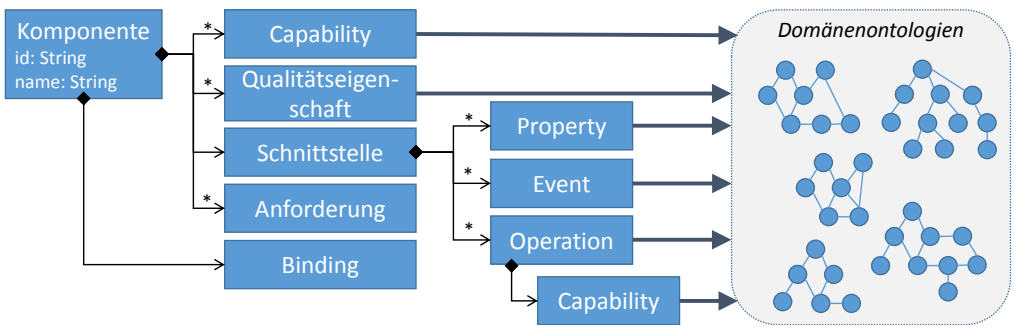


Abbildung 5.2: Abstraktes Schema der Komponentenbeschreibung

Jede Komponente besteht aus Attributen wie einer ID und einem Namen. Weiterhin erfolgt die Definition von Capabilities, von Qualitätseigenschaften, von Anforderungen der Komponente an ihren Kontext und von der technischen Komponentenschnittstelle bestehend aus Operations, Events und Properties. Zudem beinhaltet das *Binding* alle Informationen zur Integration und Ausführung einer Komponente in einer Laufzeitumgebung. Dazu gehören Referenzen auf Ressourcen, wie Quellcode- und CSS-Dateien, sowie die Anweisungen zum Instantiieren einer Komponente.

Das Schema wurde in dieser Dissertation im Wesentlichen um Capabilities sowie die Anforderungen von Komponenten an ihren Kontext erweitert. Mit letzteren kann beispielsweise angegeben werden, dass eine Komponente bestimmte Technologien von der Laufzeitumgebung benötigt, um ordnungsgemäß zu funktionieren. Auch Anforderungen an den Nutzer, wie das Vorhandensein von Accounts zwecks Zugriff auf Services, werden unterstützt. Der Zusammenhang zwischen Anforderungen und Capabilities der Komponente wird mittels Referenzierung hergestellt. Somit kann beschrieben werden, dass Funktionalitäten nur unter bestimmten Kontextbedingungen verfügbar sind. Technologisch dienen SPARQL-Queries zur Formulierung der Anforderungen und werden beim Matching von Komponenten gegen die semantische Komponentenbeschreibung und das semantische Kontextmodell, siehe Abschnitt 5.1.3, abgeglichen.

Die SMCDL enthält Konstrukte zur Umsetzung des Großteils des Capability-Metamodells, was anhand von Listing 5.1 erläutert wird. Capabilities auf Komponentenebene werden in Form von `capability`-Elementen direkt unterhalb des Wurzelelements `component` platziert, siehe Zeilen 2–15. Ein solches Element enthält für die Kernattribute Entity, Entity-Context, Activity und Activity-Modifier eine Entsprechung als XML-Attribut. Das Attribut `requiresInteraction` dient der Unterscheidung von UI- und System-Capabilities. Jede Capability erhält eine ID zu Referenzierungszwecken. Capabilities auf Komponentenebene können `viewbinding`-Elemente beinhalten, um das Konzept der View-Bindings zu unterstützen, siehe Zeilen 9–11. Darin kann zum Beispiel ein Element `atomicoperation` deklariert werden, welches im Attribut `element` mit CSS-Selektoren entsprechende Zielelemente referenziert und in `modifier` die vom Nutzer durchzuführende Interaktionstechnik definiert, zum Beispiel Tastatureingabe. Verkettung erlauben die Elemente `causes` und `causedBy` (Zeile 6). Operationen definieren eigene Capabilities, was in den Zeilen 22–24 veranschaulicht wird. Im Unterschied zu denen auf Komponentenebene ist bei diesen Capabilities das Attribut `requiresInteraction` auf `false` festgesetzt. Events und Properties verweisen auf andernorts definierte Capabilities per Element `causedBy` (Events und Properties) und `causes` (nur Properties). Dabei kann eine einzelne Capability-ID verwendet werden, siehe Zeile 19, oder ein *Capability-Selektor* wie in Zeile 6. Letzterer dient der Umsetzung der im Capability-Metamodell vorgesehenen Möglichkeit mehrere Capabilities mit einem Connective zu referenzieren. Dazu können IDs mit den Schlüsselworten »and« und »or« verknüpft werden.

```
1  <component id="..." name="Google Maps" version="1.0" isUI="true" ...>
2    <capability id="capDispLoc" activity="a:Display" entity="g:Location">
3      <viewbinding>
4        <atomicoperation element="div[id$='_mapcanvas']" id="..." />
5      </viewbinding>
6      <causedBy>capInpLoc or cap02 or capInpLocDet</causedBy>
7    </capability>
8    <capability id="cap02" activity="a:Select" entity="g:Location">
9      <viewbinding>
10       <atomicoperation element="..." id="..." interaction="i:TypeOperation" />
11     </viewbinding>
12     <viewbinding>
```

```
13        <atomicoperation element="..." id="..." interaction="i:DragNDrop" />
14      </viewbinding>
15    </capability> ...
16    <interface> ...
17      <event name="locationSelected">
18        <parameter name="location" type="g:Location" />
19        <causedBy>cap02</causedBy>
20      </event>
21      <operation name="showLocation">
22        <capability entity="g:Location" activity="a:Input" id="capInpLoc">
23          <causes>capDispLoc</causes>
24        </capability>
25        <parameter name="location" type="geo:Location" />
26      </operation> ...
27    </interface>
28  </component>
```

Listing 5.1: Ausschnitt der SMCDL-Beschreibung einer Kartenkomponente

Bei Properties wird aus Gründen der Einfachheit in der SMCDL nicht gefordert, dass eine Capability definiert wird, obwohl dies gemäß den Ausführungen aus Abschnitt 5.1.1 vorgesehen ist. Es wird von impliziten Capabilities ausgegangen, welche an notwendigen Stellen, wie dem Algorithmus in Kapitel 5.3, expliziert werden. Ebenso wird aus Gründen der Einfachheit vom Komponentenentwickler nicht die Definition kompositer Capabilities gefordert, sondern lediglich die Verkettung atomarer Capabilities.

Zum Erhöhen der Qualität von Annotationen werden Annahmen getroffen, siehe Anhang A.1, die zudem als Leitlinien für Komponentenentwickler aufgefasst werden können.

5.1.3 Nutzer- und Kontextmodell

Das Nutzerkontextmodell ist ein weiteres wesentliches Modell, das für das Bestimmen von Vorschlägen und für die Konfiguration der Plattform herangezogen wird. Insbesondere beim Ranking von Kompositionsfragmenten wird es genutzt, um die Kandidatenmenge nach Relevanz zu sortieren, was in Kapitel 6.3.3 ausführlicher behandelt wird.

Wesentliche Klassen und Beziehungen des Kontextmodells stellt Abbildung 5.3 dar. *Nutzer* werden anhand von Attributen, wie Name und geographischer Position, beschrieben. Zudem werden Eigenschaften des *aktuellen Geräts* des Nutzers modelliert, zum Beispiel hinsichtlich des *Zustands* und der Software- und Hardware*ausstattung*. Weiterhin können Nutzer als *Mashup-Entwickler* auftreten, wobei zwischen *Laien* und *Experten* unterschieden wird. Die erzeugten CWA sind in diesem Fall mit dem jeweiligen Nutzer assoziiert. Die *Intention* des Nutzers innerhalb einer Sitzung wird ebenfalls modellhaft hinterlegt, in erster Näherung in Form von Capabilities. Ausgefeiltere Konzepte, beispielsweise aus dem InfoApp-Projekt [MRM17], können perspektivisch hinzugefügt werden, sind jedoch nicht Schwerpunkt der vorliegenden Arbeit. *Präferenzen* umfasst das Modell in mehrerlei Aspekten. Zunächst kann ein Nutzer *Qualitätsanforderungen* besitzen, welche die Suche und die Empfehlung von Kompositionsfragmenten beeinflussen. Weiterhin können Präferenzen und Einstellungen hinsichtlich der Assistenzwerkzeuge der Kompositionsplattform hinterlegt werden (*Werkzeugpräferenz*). Dazu gehören bevorzugte Empfehlungsstrategien, vergleiche Kapitel 6, und Assistenzsysteme, zum Beispiel die präferierte Art der Suche und ob der Awareness-Modus initial aktiviert wird, siehe Kapitel 7. Zusätzlich werden beliebte Kompositionsfragmente anhand einer Nutzungshistorie modelliert. Letztere umfasst im Wesentlichen die vom Nutzer erstellten CWA (Assoziation *hat erstellt*) und sämtliches vom Nutzer abgegebenes *Feedback*. Die genaue Vorstellung des Feedback-Metamodells findet in Kapitel 5.1.4 statt.

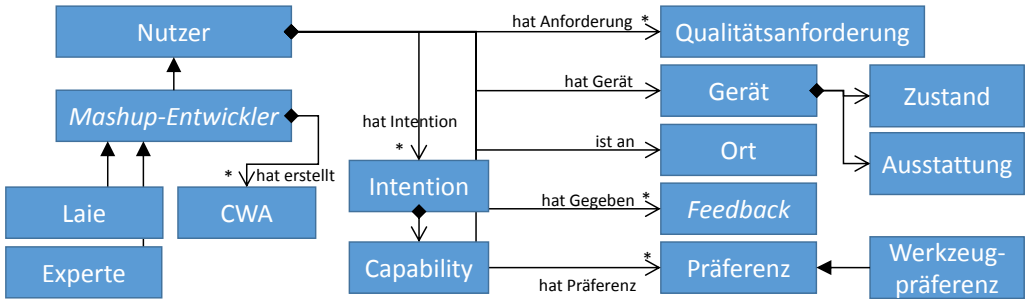


Abbildung 5.3: Überblick des Nutzerkontextmodells

Zur Instanziierung des Nutzerkontextmodells dienen verschiedene Techniken. Zunächst kommt explizites Befragen von Nutzern in Betracht, insbesondere hinsichtlich einer Selbsteinschätzung der Entwicklungsexpertise und präferierter Assistenzsysteme. Weiterhin kann das Nutzerverhalten beobachtet werden, um eine automatische Ableitung von Eigenschaften vorzunehmen. Darüber hinaus sind regelbasierte Techniken zum Schlussfolgern konzeptionell angedacht, die zum Beispiel anhand des Gerätetyps und dessen Zustands Qualitätsanforderungen zum Bevorzugen von Kompositionsfragmenten mit geringem Energieverbrauch hinzufügen. Ähnliche Prinzipien dienen dazu, anhand der Entwicklungsexpertise des Nutzers dessen Qualitätsanforderungen und bevorzugte Assistenzsysteme initial mit Werten auszustatten. Für den Aspekt des Nutzerfeedbacks wird die Thematik der Befüllung des Nutzermodells in Kapitel 6 näher untersucht.

5.1.4 Metamodell für kontextualisiertes Feedback

Zur Verbesserung der Qualität von Empfehlungen sammelt eine Mashup-Plattform Feedback von Nutzern. Es nimmt eine wesentliche Stellung in der Wissensbasis ein, da es strukturelles und semantisches Wissen um tatsächliche Nutzungsdaten ergänzt.

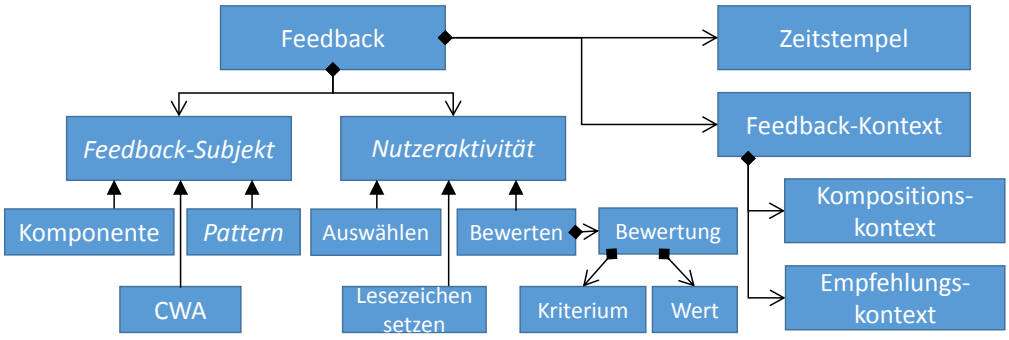


Abbildung 5.4: Überblick des Metamodells für kontextualisiertes Feedback

Abbildung 5.4 veranschaulicht das konzipierte Metamodell. Wie zu erkennen ist, bezieht sich ein *Feedback*-Eintrag stets auf ein zugehöriges *Feedback-Subjekt*, das heißt Komponenten, Anwendungen und Patterns im Rahmen des EUD von CWA. Um sowohl explizites als auch implizites Feedback zu unterstützen, sieht das Konzept entsprechende *Nutzeraktivitäten* vor, die ein Nutzer mit dem *Feedback-Subjekt* durchgeführt hat, wie *Auswählen* und *Bewerten*. Optional besitzen solche Aktivitäten spezifische Eigenschaften.

Im Fall der Aktivität *Bewerten* wird beispielsweise die mehrdimensionale Bewertung als Tupel aus Bewertungskriterium und zugehörigem Wert hinterlegt. Das zugrundeliegende Blackbox-Komponentenmodell erschwert das Übertragen von Forschungsergebnissen aus Kapitel 3.2.2 zu Nutzeraktivitäten, die geeignete Indikatoren für Interesse darstellen. Denn zahlreiche dieser Interaktionen finden innerhalb des Komponenten-UI statt und entziehen sich somit einer unmittelbaren Erfassung. Zwar wäre ein Erweitern des Komponentenmodells zum Offenlegen solcher Informationen denkbar, jedoch würde dies den Aufwand für Komponentenentwickler weiter steigern. Daher werden zunächst vorrangig Aktivitäten unterstützt, die über das Plattform-UI erfassbar sind. Allerdings bietet das grundlegende Modell die nötige Erweiterbarkeit um zusätzliche Aktivitäten, wobei die Herausforderung darin besteht, entsprechende Erfassungsmechanismen anzubieten. Darüber hinaus gibt ein *Zeitstempel* über den Zeitpunkt Auskunft, an welchem das Feedback gesammelt wurde. Als Neuheit beinhaltet das Metamodell Konstrukte zur Beschreibung des *Feedback-Kontextes* anhand mehrerer Dimensionen:

Kompositionskontext: Dieser umfasst optional die CWA, welche der Nutzer zum Zeitpunkt des Feedbacks nutzte. Dabei werden neben der ID auch der aktuelle Kompositionszustand inklusive Komponenten und Kommunikationskanälen erfasst.

Empfehlungskontext: Er reflektiert unter welchen Umständen und zu welchem Zweck eine Empfehlung ausgesprochen wurde, indem Trigger und Empfehlungsmethode benannt werden, die zur Berechnung des *Feedback-Subjekts* führten. Zudem erfolgt das Abspeichern einer relevanten Teilmenge der Parameterbelegung der Empfehlungsmethode, um essentielle, feingranulare Unterschiede zuzulassen. Beispiele hierfür sind funktionale Anforderungen und eingegebene Stichworte.

Diese Modellierung bezieht die Erkenntnisse aus der Analyse verwandter Arbeiten in Kapitel 3.2.2 ein. Das Konzept ist generischer als [VFM13] und [BDM13b], da sowohl Subjekte als auch Feedback-Kontext umfassender modelliert werden können. Die Kernattribute sind inspiriert von Activity-Streams [@ASWG] und CAM [Gov+15], allerdings sowohl hinsichtlich Subjekt als auch Kontext an die Mashup-Domäne adaptiert worden.

Der gewählte Ansatz erlaubt es, den Feedback-Kontext detailliert zu beschreiben. Dadurch kann Feedback identifiziert werden, das zu Gegebenheiten erfasst wurde, die zum aktuellen Kontext passen. Zudem ist die Grundlage für unscharfe Suche nach Feedback-Einträgen vorhanden, indem gezielt einzelne Eigenschaften, etwa des *Empfehlungskontextes*, vernachlässigt werden. Durch die Assoziation mit Nutzerprofilen können Techniken des kollaborativen Filterns eingesetzt werden. Dies stellt eine wesentliche Grundlage für das Empfehlungssystem dar, das in Abschnitt 6 genauer vorgestellt wird.

5.2 Semantische Datenmediation

Die konzipierten Techniken zur semantischen Mediation von Komponentenschnittstellen [Rad+14] dienen dazu, Heterogenität und Inkompatibilitäten bei der semantischen Typisierung von Schnittstellenelementen zu überbrücken. Sie basieren auf existierenden Ansätzen aus dem Bereich SWS, siehe Kapitel 3.4, und übertragen diese in erweiterter Weise auf CWA. Die Techniken erlauben deutlich mehr Spielraum bei dem Verknüpfen von Komponenten als in bisherigen Kompositionsansätzen für Mashups üblich.

5.2.1 Vorbetrachtungen und Definitionen

Mediationstechniken nutzen Domänenwissen aus, das in den Ontologien formuliert ist, welche zur Annotation von Komponenten verwendet werden. Unterstellt wird dabei, dass es sich um Ontologien in RDFS oder OWL-DL handelt. Im Wesentlichen können daher Klassen, Datatype- und Object-Properties sowie Individuen referenziert werden. Je nach Anforderungen und angedachter Nutzung existieren verschiedene Möglichkeiten, Zusammenhänge einer Domäne zu modellieren. Deshalb werden Annahmen an die Art der Modellierung und Annotation getroffen, die gängige Best-Practices einbeziehen. Grundsätzlich können Klassen direkt annotiert werden, wie `Location`, oder indirekt über eine OWL-Object-Property, deren Wertebereich diese Klasse umfasst, wie `hasCenter`. Letzteres kann dazu genutzt werden, spezifischere Bedeutungen eines Parametertyps hervorzuheben. Unter Umständen ist es sinnvoller, Individuen anstatt Klassen zu modellieren und zu annotieren, wie bei Einheiten und anderen konvertierbaren Maßen. Dies gilt auch für Klassen, die auf erster Property-Ebene mit solchen Einheiten assoziiert sind: Beispielsweise sollte anstatt der Subklasse `DistanceInMiles` ein konkretes Individuum der Klasse `Distance`, dessen Object-Property `hasUnit` auf `mile` zeigt, annotiert werden.

Vor der detaillierten Spezifikation der Mediationstechniken werden zunächst grundlegende Begrifflichkeiten definiert. *Schnittstellenelemente*, wie Operations, Events und Properties, erlauben den parametrisierten Datentransfer zwischen Komponenten. Parameter sind eindeutig identifizierbar und durch ein annotiertes semantisches Konzept typisiert. Jedes Schnittstellenelement besitzt einen oder mehrere Parameter. Von einem datenorientierten Standpunkt fassen Komponenten eine Menge von Schnittstellenelementen zusammen. Zwecks Datenaustausch verbinden gerichtete *Kommunikationskanäle* ein Schnittstellenelement einer Quellkomponente QK mit einem einer Zielkomponente ZK. Eine Zuweisungsvorschrift muss existieren, welche die n Ausgabeparameter P_{out} von QK auf die m Zielparameter P_{in} von ZK abbildet. Sind für sämtliche Zuweisungen die Parametertypen identisch, handelt es sich um eine perfekte Übereinstimmung. P_{out} and P_{in} sind *semantisch kompatibel*, falls ein *semantischer Konnektor* $SeCo$ definiert werden kann. Ein semantischer Konnektor entspricht einer Menge von Kommunikationskanälen und Mediationstechniken. Als eine Zuweisungsvorschrift stellt ein $SeCo$ sicher, dass alle Parameter P_{in} eines Schnittstellenelements von ZK verknüpft sind und mit Daten vom benötigten semantischen Typ befüllt werden. Ein $SeCo$ kann somit auch mehrere Kanäle beinhalten, welche von Schnittstellenelementen mehrerer Quellkomponenten ausgehen, und trägt in diesem Fall Sorge für die Synchronisation der eingehenden Daten. Um die semantische Kompatibilität herzustellen, kann ein $SeCo$ mehrere Mediationstechniken kombinieren. Eine *Mapping-Definition* spezifiziert schließlich, wie welche Mediationstechniken in einem $SeCo$ benutzt, konfiguriert und komponiert werden.

5.2.2 Techniken zur semantischen Datenmediation

Nachfolgend werden Techniken zur semantischen Datenmediation spezifiziert. Abbildung A.2 im Anhang liefert weitere Beispiele zu den hier genannten.

Upcast Bei der Mediationstechnik *Upcast* handelt es sich um eine Erweiterung zuvor entwickelter Konzepte [PRM11b]. Sie dient dem Überbrücken abweichender Generalisierungsgrade annotierter Konzepte. Im Fall von Klassen bedeutet dies, dass ein Individuum der spezifischeren Subklasse, zum Beispiel `AppointmentLocation` in eine »Sicht« gemäß

einer ihrer Superklassen, beispielsweise `Location`, überführt wird. Der Gedanke dahinter ist, dass alle Individuen einer Subklasse auch den Superklassen zugeordnet werden können. Prinzipiell sind *Upcasts* auch für annotierte OWL-Object-Properties anwendbar, sofern zwischen der Object-Property des Quellparameters und des Zielparameters die Beziehung `subPropertyOf` herrscht. Dieser Fall kann so gehandhabt werden, als wären die Wertebereiche annotiert worden. Für annotierte Datatype-Properties wird angenommen, dass der Wertebereich im Kern unverändert bleibt, zum Beispiel String oder Integer, sodass ein *Upcast* einer Weiterleitung entspricht. *Upcast* ist nur anwendbar, falls die Generalisierung der Konzepte mit der Richtung des Kommunikationskanals übereinstimmt. Das heißt, dass lediglich »Casts« aufwärts in der Vererbungshierarchie möglich sind, da ansonsten nicht sichergestellt ist, dass Erweiterungen oder Einschränkungen von Subklassen Rechnung getragen wird.

Conversion Die Mediationstechnik *Conversion* hat zwei hauptsächliche Anwendungsgebiete. Zunächst dient sie dem Auflösen von Inkompatibilitäten bei annotierten konvertierbaren Konzepten, wie Maßeinheiten. Typischerweise wird für solche Konvertierungen domänenspezifisches Wissen benötigt.

- *Einheiten, Maßeinheiten, Datentypen*: Als Beispiel hierfür sei `kilometer` → `mile` aus dem Szenario in Abbildung 2.2 genannt. Dabei sind oft primitive Datentypen involviert. Das spezifische Wissen zu notwendigen Transformationen kann zum Beispiel in speziellen Ontologien wie QUDT [@QUDT] formalisiert werden. QUDT umfasst eine Menge von Ontologien und bietet insgesamt ein Modell für messbare Einheiten, die in Domänen wie `LengthUnit` eingeordnet sind. Durch die Verwendung von Basiseinheiten pro Domäne und die Angabe von Konvertierungsfaktoren können innerhalb einer Domäne Einheiten ineinander umgerechnet werden.

- *Anpassung von Skalen*: Darunter fallen weitere domänenspezifische Transformationen, wie das Umformen eines Fünfsterne- in ein Zehnpunkte-Rating.

Weiterhin dient *Conversion* zum Umwandeln äquivalenter Klassen, wie `Location` und `Ort`. Zwar reichen OWL-eigene Sprachkonstrukte wie `equivalentClass` auf Modellebene aus, die konkrete Umsetzung und Detektion solcher Fälle hängt jedoch stark von der Definition von »Äquivalenz« ab. Aus Gründen der Einfachheit und zum Senken der algorithmischen Komplexität wird konzeptionell von einer strikten Definition ausgegangen. Wenn zwei Klassen als äquivalent deklariert sind, muss jede zugehörige OWL-Property der Ausgangsklasse eine `equivalentProperty` in den OWL-Properties der Zielklasse besitzen. Zudem sind dadurch generische Algorithmen weiterhin anwendbar.

Semantic Split Ein *Semantic Split* projiziert Teilkonzepte eines grobgranulareren Konzepts aus P_{out} auf einen oder mehrere P_{in}. Übertragen auf OWL bedeutet dies, dass von einem Individuum, das als Parameter oder als Property auf einem Kommunikationskanal transferiert wird, die Werte bestimmter Datatype- oder Object-Properties ausgelesen und als Parameter an die Zielschnittstellenelemente weitergegeben werden. Diese Mediationstechnik ist demzufolge nur anwendbar, wenn für die Annotation die folgenden OWL-Konstrukte verwendet werden:

- Klasse: Die Werte von Datatype- und Object-Properties einer Klasse können Zielparametern zugewiesen werden, wenn letztere mit einem semantisch zum

Wertebereich der OWL-Property passenden Konzept annotiert sind. Als Beispiel kann `Appointment` → `Location` unterstützt werden, unterstellt, dass die Object-Property `hasLocation` von `Appointment` den Wertebereich `Location` besitzt.

- Object-Property: Ein Fall wie `hasCenter` → `{hasLatitude, hasLongitude}`, wird derart gehandhabt als wäre der Wertebereich als Parametertyp annotiert.

Semantic Join Ein *Semantic Join* dient dem Zusammenführen mehrerer Teilkonzepte, gegeben als Parameter eines Quellschnittstellenelements in P_{out}, zu einem grobgranulareren Konzept. Letzteres wird als ein Parameter in P_{in} an das Zielschnittstellenelement weitergeleitet. Übertragen auf OWL bedeutet dies, dass aus gegebenen Literalen und Individuen ein neues Individuum erzeugt wird, wobei sichergestellt ist, dass letzteres die Restriktionen seiner Klasse erfüllt. Angenommen eine Karte veröffentlicht ein Event mit zwei Parametern vom Typ `{hasLatitude, hasLongitude}` und in der Anwendung befindet sich eine Komponente zur Suche nach Point of Interest (POI), welche eine Operation mit einem Parameter vom Typ `Location` besitzt. Dann ist ein *Semantic Join* möglich, falls gewährleistet ist, dass das erzeugte Individuum sämtliche Bedingungen der Zielklasse erfüllt. Im Beispiel betrifft dies etwa die Bedingung, dass `hasLatitude` und `hasLongitude` einer `Location` mit je genau einem Wert assoziiert sein müssen.

Partial Substitution Diese Technik dient dem Aktualisieren von Individuen, die als Property einer Komponente repräsentiert sind. Dabei werden passende OWL-Datatype- oder Object-Properties aktualisiert, indem deren Werte mit Literalen oder Individuen, übertragen als Parameter aus P_{out}, ersetzt werden. Dies bedingt, dass als Parametertyp eine Klasse oder eine OWL-Object-Property, deren Wertebereich dann ausschlaggebend ist, annotiert sein muss. Bezüglich des Ausschnitts der Ontologie in Abbildung 2.2 des Szenarios ist *Partial Substitution* zwischen `AppointmentLocation` und `Appointment` möglich, angesichts der Object-Property `hasLocation` von `Appointment`. Das bedeutet für ein Individuum vom Typ `Appointment`, repräsentiert als Zielparameter, dass dessen aktueller Wert der Object-Property `hasLocation` ersetzt wird, sobald eine `Location` als Parameter aus P_{out} übertragen wird. *Partial Substitution* ist exklusiv auf Properties von Komponenten anwendbar, da nur diese per Definition einen Ausschnitt des Komponentenzustands direkt offenlegen, siehe Kapitel 2.1.1. Das bedeutet, dass bei der Abfrage des aktuellen Werts einer Property ein Zugriff auf die Datenschicht stattfindet und somit direkt das zugrundeliegende Individuum zugänglich ist. Analog wird angenommen, dass sich das Setzen einer Property direkt auf das in der Datenschicht repräsentierte Individuum auswirkt. Diese Mediationstechnik erhöht die Möglichkeiten, Properties von Komponenten in beide Richtungen zu synchronisieren, auch wenn die Typisierung nicht identisch ist. Für Events und Operations von Komponenten ist *Partial Substitution* hingegen nicht geeignet, da bei ihnen gemäß Spezifikation keine derart enge Relation zum Komponentenzustand unterstellt werden kann wie bei Properties. Zudem sind Events und Operations, im Gegensatz zu Properties, nur entweder ein- oder ausgehend.

Collection-Conversion Das Komponentenmodell aus Abschnitt 2.1.1 sieht vor, dass Parameter von Operationen und Events sowie Properties mit dem Attribut `isCollection` angeben können, ob einzelne Instanzen des entsprechenden semantischen Typs veröffentlicht werden oder eine ganze Kollektion. Ein Beispiel illustriert Abbildung A.2. Ziel dieser

Mediationstechnik ist das Überbrücken von Heterogenität in Fällen, bei denen sich die Annotation `isCollection` zwar unterscheidet, die semantische Typisierung ansonsten aber, gegebenenfalls durch Anwenden zusätzlicher Mediationstechniken, kompatibel ist. Es existieren in RDFS mit `rdf:Bag`, `rdf:Seq` und `rdf:Alt` verschiedene Konstrukte zur Definition von Kollektionen, die allesamt Subklassen von `rdfs:Collection` [@RDFS] sind. Diese Unterscheidung indiziert dem Menschen zusätzliche Information, während die formale Semantik der Konstrukte identisch ist [@RDFSe]. Im Konzept wird daher vorgesehen, dass ein Parameter oder eine Property, die mit `isCollection=true` attributiert ist, zur Laufzeit als Instanzen von `rdf:Collection` von ihrer Komponente veröffentlicht werden. Sofern nicht direkt semantische Daten, das heißt RDF-Graphen, veröffentlicht werden, muss ein Grounding sowie Lifting und Lowering stattfinden. In JavaScript Object Notation (JSON) bietet sich beispielsweise die Verwendung des inhärenten Array-Datentyps an, während in XML ein dediziertes Container-Element die einzelnen Objekte aufnehmen muss. *Collection-Conversion* bietet für folgende Konstellationen Datenmediation an.

- *Kollektion* → *Individuum*: In diesem Fall wird ein Element aus der übergebenen `rdfs:Collection` extrahiert und ausgegeben. Es stehen mehrere *Extraktionsmodi* zur Auswahl: 1. *Dequeue*: Das erste Element der Kollektion wird extrahiert. Dies ist der voreingestellte Modus. 2. *Random-Pick*: Ein zufälliges Element wird gewählt. 3. *Defined-Index*: Das Element an einer konfigurierbaren Indexposition wird herausgenommen.
- *Individuum* → *Kollektion*: Eine neue `rdfs:Collection` mit dem übermittelten Individuum als einzigem Element wird erzeugt.
- *Mehrere Individuen (und Kollektionen)* → *Kollektion*: Werden mehrere Parameter eines kompatiblen Typs bereitgestellt und sollen einem einzelnen Zielparameter zugeordnet werden, wird eine neue `rdfs:Collection` erzeugt und sämtliche einzeln oder als Kollektion übertragenen Ressourcen in diese eingefügt.

Der Vollständigkeit halber sei erwähnt, dass der Fall Kollektion → Kollektion existiert, jedoch aufgrund der fehlenden Annotierbarkeit der Größe von Kollektion nicht relevant ist. Theoretisch sind zwar auch Operationen auf Kollektionen wie Teilausschnitte und Invertierung denkbar. Diese sind jedoch in hohem Maße anwendungsspezifisch und entziehen sich daher dem automatischen Ableiten generischer Mapping-Definitionen.

Syntactic Join Neben den bisher behandelten Mediationstechniken, welche auf semantischer Ebene operieren und somit die Typisierung von Parametern beeinflussen, agiert der *Syntactic Join* auf syntaktischer Ebene und verändert die Typisierung nicht. Der hauptsächliche Zweck eines *Syntactic Join* ist die Synchronisation von Parametern aus P_{out}, die von mehreren Quellschnittstellenelementen einer oder mehrerer Quellkomponenten stammen. Diese werden gemeinsam an ein einzelnes Zielschnittstellenelement übertragen. Wie diese Zusammenführung erfolgt, kann je nach Anwendungsfall variieren und wird über verschiedene *Synronisationsmodi* geregelt.

Kombination von Mediationstechniken Wie bereits angedeutet, können mehrere Mediationstechniken in einem semantischen Konnektor kombiniert werden. Dies tritt

besonders dann auf, wenn *Syntactic Joins* stattfinden. Da diese lediglich auf syntaktischer Ebene der Parameter agieren, dienen sie oft der Vorbereitung von Techniken auf semantischer Ebene. Beispielsweise stellt ein *Syntactic Join* zunächst sicher, dass alle P_{out} der Quellkomponenten zusammengetragen werden, bevor diese durch einen *Semantic Join* zu einem grobgranulareren Konzept eines einzelnen Parameters aus P_{in} aggregiert werden. Abbildung 5.5 zeigt ein Beispiel, bei dem aus dem einzelnen Parameter vom Typ `Appointment` des Kalenders zunächst durch einen *Semantic Split* der Ort (`AppointmentLocation`) und Zeitpunkt (`hasTime`) projiziert werden. Der Ort wird anschließend per *Upcast* in eine `Location` umgewandelt. Schließlich sorgt ein *Syntactic Join* dafür, dass beide Parameter mit der `Location` aus einer Karte zusammen an eine Operation der Flugsuchkomponente übertragen werden. Als weiteres Beispiel sei die Kombination mit einem nachgeschalteten *Collection-Conversion* genannt, wodurch Parameter aus mehreren Schnittstellenelementen gesammelt, in eine Kollektion zusammengefasst und an ein einzelnes Zielschnittstellenelement weitergereicht werden können.

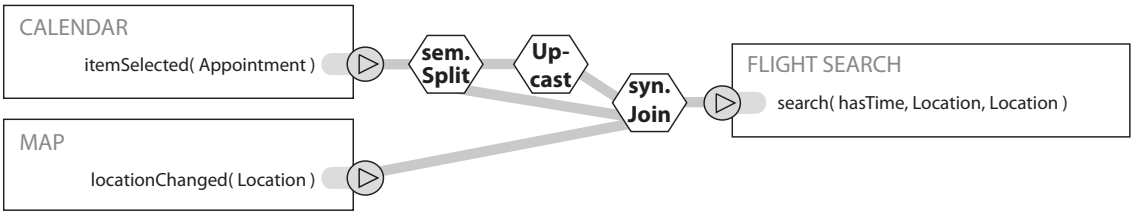


Abbildung 5.5: Exemplarische Kombination mehrerer Mediationstechniken

5.2.3 Architektonische Implikationen und Abläufe

Anhand von Abbildung 5.6 erörtert dieser Abschnitt architektonische Erweiterungen einer Kompositionsplattform zur Bereitstellung semantischer Mediationstechniken.

Ontologien spielen eine zentrale Rolle, da diese zur Annotation von Schnittstellenelementen dienen und das Wissen über semantische Zusammenhänge von Domänenkonzepten formalisieren. Neben domänenspezifischen Ontologien können Upper-Level-Ontologien (zum Beispiel für Einheiten), Ontologien, die Transformationen zwischen Konzepten beschreiben, und Mapping-Ontologien, welche Ähnlichkeitsbeziehungen zwischen Konzepten in verschiedenen Wissensmodellen ausweisen, unterschieden werden.

Im Konzept wird angenommen, dass ein State-of-the-Art-Prozess zum Ontology-Alignment, vergleiche Übersichtspapier [SE13], vorangestellt wird, falls Komponenten einer Kompositionsplattform unterschiedliche Vokabulare für semantisch übereinstimmende Konzepte benutzen. Dabei wird es sich in der Regel um einen semiautomatischen Vorgang handeln. Denn zwar können aktuelle Algorithmen zum Matching und Alignment von Ontologien in Abhängigkeit vom Datensatz qualitativ hochwertige Ergebnisse mit einer Zuverlässigkeit von über 90 % erzielen [Che+16], jedoch sind Fälle kaum zu vermeiden, bei denen ein Algorithmus nur unsichere Zuordnungen identifiziert oder komplett scheitert, sodass manuelle Kontrolle und Validierung notwendig ist. Im Ergebnis liegen ebensolche Mapping-Ontologien vor, die lediglich bestätigte Aussagen über `equivalentClass`, `equivalentProperty`, `sameAs` und so weiter treffen.

Im Bezug auf die Unterstützung semantischer Datenmediation sind zwei wesentliche Aufgaben durch eine Mashup-Plattform bereitzustellen: *Mapping-Discovery* zum Ableiten

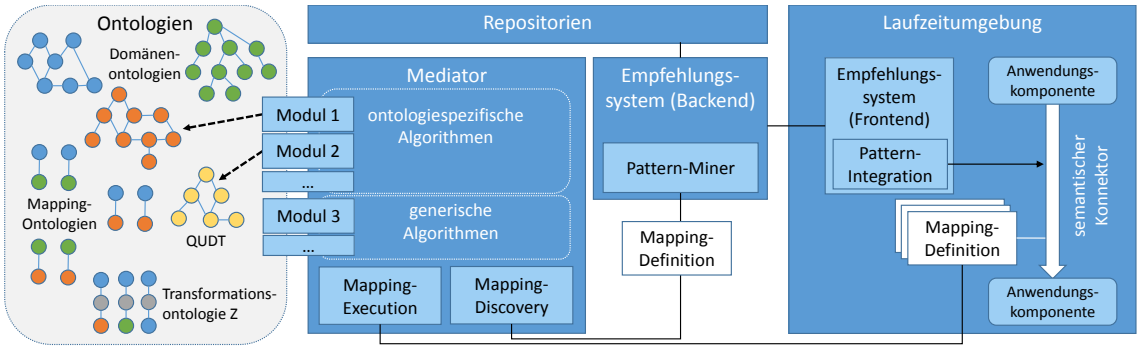


Abbildung 5.6: Architektonische Einordnung der Konzepte zur Datenmediation

von Mapping-Definitionen für gegebene Signaturen und *Mapping-Execution* für das eigentliche Anwenden von Mapping-Definitionen auf Nutzdaten zur Laufzeit.

Konzeptionell übernimmt diese Aufgaben der *Mediator*. Zur Mapping-Discovery bietet er über seine Schnittstelle die Möglichkeit, für zwei gegebene Signaturen mögliche Mapping-Definitionen abzufragen. Dabei können mehrere valide Ergebnisse vorliegen, da ein lineares Zuordnungsproblem zu lösen ist. Alle resultierenden Mapping-Definitionen werden vom Mediator wertungsfrei zurückgeliefert. Zum Bestimmen und Ausführen von Mapping-Definitionen existieren im Mediator zwei Arten von Algorithmen:

- **Generische** Algorithmen nutzen unmittelbar von RDFS und OWL bereitgestellte Modellkonstrukte und Verfahren zum Schlussfolgern, zum Beispiel Subsumption, Definitions- und Wertebereiche von Object-Properties und Äquivalenz.
- **Ontologiespezifische** Algorithmen basieren auf speziellen Konstrukten eines Domänenvokabulars, beispielsweise zur Konvertierung von Längeneinheiten auf Basis der *Quantities, Units, Dimensions and Data Types Ontologies (QUDT)*. Diese Art der Algorithmen wird konsultiert, sofern generische Algorithmen keine Abbildungsvorschrift ableiten können.

Sämtliche Algorithmen sind zur Einbindung in den *Mediator* in Module gekapselt und mit einer klar definierten Schnittstelle ausgestattet. Somit ist der Mediator um Mechanismen zur Mapping-Discovery und -Execution erweiterbar. Diese Schnittstelle umfasst Methoden, die Aussage darüber treffen, ob ein Modul zwei gegebene Signaturen unterstützt. Zusätzlich existieren Methoden zum Ausführen der Mapping-Discovery beziehungsweise Mapping-Execution. Als konkretes Beispiel eines ontologiespezifischen Moduls sei das für die QUDT genannt. Es unterstützt Signaturen, falls diese auf QUDT-Konzepte aus der selben Domäne verweisen, sodass eine Umrechnung möglich ist, und liefert ein entsprechend konfiguriertes *Convert*-Mapping zurück. Zur Ausführung des *Convert*-Mappings wird zur Laufzeit der Umrechnungsfaktor auf Instanzdaten angewendet.

Die Mapping-Discovery des Mediators wird insbesondere durch Architekturkomponenten zum semantikbasierten Ableiten von Patterns genutzt, um die Verknüpfbarkeit von Komponenten über semantische Konnektoren zu bestimmen. Die dabei vom Mediator erhaltenen Mapping-Definitionen werden in den Pattern-Instanzen hinterlegt und stehen somit einem MRE zur Verfügung, um empfehlungsgestützte Kompositionsschritte automatisiert vorzunehmen.

Zum Bereitstellen von Datenmediation bietet eine Laufzeitumgebung die notwendigen Mittel. Für die Umsetzung von Mediationstechniken auf Konzepte gemäß dem MCM

existieren dabei mehrere, gleich ausdrucksstarke Optionen, die nachfolgend anhand von Abbildung 5.7 diskutiert werden.

Mediationskomponenten: Spezielle Komponenten stellen Mediationstechniken bereit und können in Form konfigurierbarer Schablonen durch ein MRE in die Komposition eingebunden und gemäß den Mapping-Definitionen konfiguriert werden. Da Mediationskomponenten dem Komponentenmodell entsprechen, können sie zwischen die Komponenten einer CWA geschaltet werden. Diese konsistente Handhabung kann als vorteilhaft angesehen werden. Nachteilig ist hingegen die mit zunehmender Komponentenanzahl steigende Komplexität von Kompositionsmodellen, die sowohl in Auswertungsalgorithmen als auch in Werkzeugen behandelt werden muss.

Erweiterte Komunikationskanäle: Diese werden direkt mit Mapping-Definitionen ausgestattet. Dazu weisen Kanäle genau eine Sender- und eine Empfängerkomponente auf, deren Signaturen nicht identisch sein müssen. Lediglich *Syntactic Join* wird als Mediationskomponente repräsentiert. Dadurch steigt die Komplexität von Kompositionsmodellen weniger. Zwar werden nicht alle Mediationstechniken konsistent behandelt, jedoch reflektiert das deren unterschiedlichen Zweck.

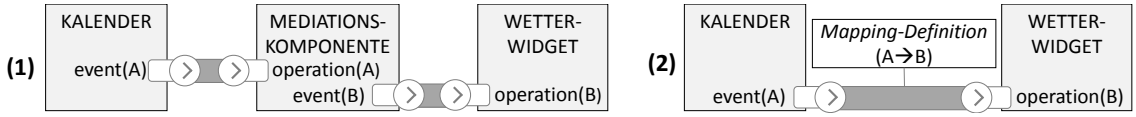


Abbildung 5.7: Umsetzung von Mediationstechniken als Mediationskomponenten (1) und mit erweiterten Kommunikationskanälen (2)

Konzeptionell ist die zweite Option zur Umsetzung von Mediationstechniken vorgesehen. *Syntactic Joins* werden als Mediationskomponenten vom MRE bereitgestellt und alle anderen Techniken in Form von Mapping-Definitionen an Kommunikationskanäle geheftet. Sobald Events auf Kommunikationskanälen auftreten, werden die Mapping-Definitionen mithilfe der Mapping-Execution des Mediators auf die Instanzdaten angewendet. Anschließend werden die mediierten Daten an die Zielkomponente weitergeleitet.

5.3 Ableiten von Capabilities

In diesem Kapitel wird ein Algorithmus vorgestellt, der die Capabilities eines beliebigen Kompositionsfragments automatisch abschätzt [RBM16; RBM17], um Anforderung 7.4 gerecht zu werden. Vorarbeiten zum Konzept liefert der Große Beleg von Ronny Kursawe [Kur13]. Wie die Referenzszenarien aus Kapitel 2.2 sowie der Konzeptüberblick in Abschnitt 4 veranschaulichen, können der in diesem Kapitel vorgestellte Algorithmus und das spezifizierte Capability-Metamodell in vielfältiger Weise in einer Mashup-Plattform für EUD eingesetzt werden. Vorrangiges Ziel ist dabei, die Kommunikation mit den Nutzern auf der Ebene von Funktionalitäten und Aufgaben anzusiedeln, anstatt sie mit technischen Begriffen zu belasten. Weiterhin kann das Empfehlungssystem von Wissen über das Ziel des Nutzers profitieren.

5.3.1 Anforderungen und verwandte Ansätze

An dieser Stelle werden zunächst wesentliche Anforderungen an den Algorithmus zur Abschätzung der Funktionalität von Kompositionsfragmenten und an dessen architektonische Einordnung gelistet. Diese Kriterien verfeinern Anforderung 7.4 und stammen aus dem in Kapitel 2.1 skizzierten Ist-Stand, dem Capability-Metamodell aus Kapitel 5.1.1 und den Referenzszenarien aus Kapitel 2.2.

1. Der Algorithmus muss für jedes beliebige Kompositionsfragment eine erwartungskonforme Beschreibung der Funktionalität in Form von Capabilities liefern.

2. Es müssen komponentenübergreifende und komponenteninterne Funktionalitäten erkannt, strukturiert und weiterführend verarbeitet werden.
   a) Datengrundlage müssen die Capabilities von Komponenten mit deren Kernattributen und Verkettung sein. Zusammenhänge von Capabilities gemäß den Kommunikationskanälen müssen beachtet werden, wobei beliebig strukturierte semantische Konnektoren zu unterstützen sind.
   b) Es müssen beliebig viele funktional zusammenhängende »Inseln« in einem Kompositionsfragment möglich sein und zu plausiblen Ergebnissen führen. Das heißt, es müssen in erster Linie pro »Insel« die Capabilities bestimmt und falls möglich übergeordnete Capabilities abgeleitet werden.
   c) Es ist notwendig, übergeordnete Capabilities aus einer Menge miteinander in Beziehung stehender Capabilities abzuleiten, um deren gemeinsame Funktionalität abzuschätzen. So muss beispielsweise eine Abgrenzung von Teilfunktionen in sequentiellen Abfolgen vorgesehen sein.

3. Bestehende Wissensbasen müssen genutzt werden, wie referenzierte Ontologien und Ergebnisse aus vorherigen Durchläufen, im Sinne von maschinellem Lernen.

4. Der Algorithmus muss einen *Confidence*-Wert liefern, der ausdrückt, wie sicher oder unsicher die Ergebnisse sind.

Zur Identifikation von Lösungsmöglichkeiten für die vorliegende Herausforderung wurden spezifische Ansätze aus Forschung und Technik analysiert. Die Autoren von [KKS10] schlagen einen Ansatz zur automatischen Generierung von *Intent-Tags* für textuelle Ressourcen vor. Intent-Tags werden dabei satzweise abgeleitet, basierend auf einer Taxonomie von menschlichen Zielen, für die typische Phrasen hinterlegt werden, die wiederum mit den Sätzen abgeglichen werden. Die Annotation des gesamten Dokuments entspricht schließlich der aggregierten Menge der einzelnen Tags. Der in der Dissertation vorgestellte Ansatz nutzt semantisches Wissen und Heuristiken, um übergeordnete Fähigkeiten aus dem Zusammenspiel von Komponenten abzuleiten. Ein auf *Tags* basierendes Konzept zur Annotation, Suche und Komposition von Komponenten stellen Bouillet et al. [Bou+08] vor. Taxonomien von Tags werden darin genutzt, um Mehrdeutigkeiten zu vermeiden und flexibleres Matching zu erlauben. Das Tag-Modell ist jedoch weniger formal und weniger ausdrucksstark als Capabilities. Weiterhin entspricht die Funktionalität einer Anwendung der Vermengung sämtlicher Komponenten-Tags, das heißt, es finden keine vergleichbaren semantischen Analysen wie in dem hier vorgestellten Ansatz statt. In [BYW12] wird ein ontologie-basiertes Modell für Mashups beschrieben. Es sieht auch Konstrukte für

die Funktionalität von Mashups vor, wobei es sich jedoch um eine textuelle Definition handelt, während Capabilities formal und semantisch hinterlegt sind. Ein Algorithmus zur Erzeugung von Modell-Instanzen für gegebene Komponenten wurde vorgeschlagen. Dabei finden lexikalische Analysen der Funktionalitätsbeschreibung statt. Im Vergleich zum hier vorgestellten Ansatz fehlt es an hierarchischer Strukturierung, an Subsequenzierung und an Ableitung übergeordneter funktionaler Zusammenhänge im Mashup. Im OMELETTE-Projekt dienen *Tags* zur Klassifikation von Komponenten [Con13b]. Der Ansatz sieht vor, zunächst gegebene Klassifikationen aus Quell-Repositorien wie dem *ProgrammableWeb* einzubinden. Falls diese fehlen, jedoch zumindest Tags existieren, findet das automatische Kategorisieren von Komponenten auf Basis der Tags statt. Dieser Ansatz zeigt die inhärenten Probleme von Tags, insbesondere Mehrdeutigkeiten, die Kategorisierung ist grobgranular und letztlich mangelt es an einer Evaluation des Ansatzes. SMASHAKER [BDM12] nutzt *semantische Tags* zur groben funktionalen Kategorisierung von Komponenten. Sie referenzieren Konzepte in *WordNet* als Vokabular, sodass Mehrdeutigkeiten weniger ein Problem darstellen. Die Ausdrucksmächtigkeit ist allerdings begrenzt, da zum Beispiel die Relation mehrerer Tags nicht definiert werden kann. Semantische Tags werden statisch bereitgestellt, wobei Techniken wie Entity-Recognition helfen, um aus Beschreibungen passende WordNet-Konzepte zu identifizieren. Darauf aufsetzend werden Mashups durch Vermengen der Komponenten-Tags kategorisiert. Es fehlt an Automatisierung und Mechanismen zum Ableiten übergeordneter Funktionalitäten. Wie deutlich wird, kann keiner der Ansätze die Anforderungen erfüllen, sodass ein neuartiges Konzept benötigt wird.

5.3.2 Definitionen und Grundlagen

Zunächst gilt es, wichtige Begriffe und grundlegende Konzepte zu beleuchten. Anhand von Abbildung 5.8 werden in diesem Abschnitt wesentliche Datenstrukturen dargestellt, die im Rahmen der Abarbeitung des Algorithmus aufgebaut werden.

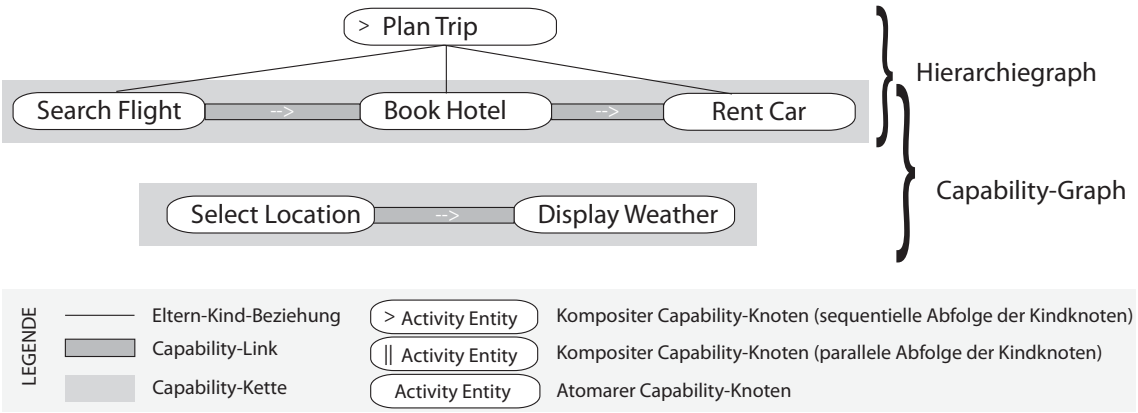


Abbildung 5.8: Beispielhaftes Schema eines Capability-Graphen mit überlagerter Hierarchie

Ein **Capability-Graph** ist eine Menge von *Capability-Knoten* und gerichteten Kanten, den *Capability-Links*. Er kann Zyklen beinhalten und repräsentiert die Capabilities eines Kompositionsfragments und deren Verbindungen. Dazu wird für jeden Kommunikationskanal sowie jede `causes`- und jede `causedBy`-Relationen eine Kante zwischen den assoziierten Capabilities beziehungsweise deren Knotenrepräsentation erzeugt. Abbildung

5.8 illustriert einen beispielhaften Capability-Graphen. *Capability-Knoten* kapseln die eigentliche Capability sowie strukturelle Informationen über den Capability-Graphen, wie ein- und ausgehende Kanten, und identitätsstiftende Merkmale, wie eine ID und die Komponentenzugehörigkeit. Ein *Capability-Link* repräsentiert den kausalen Zusammenhang zweier Capabilities, die per `causes` oder `causedBy` innerhalb einer Komponente oder per Kommunikationskanal durch ihre zugehörigen Schnittstellenelemente in Beziehung gesetzt wurden. Ein Capability-Link umfasst somit einen Start- und einen Ziel-Capability-Knoten. Liegt ein Kommunikationskanal zugrunde, können zusätzliche, für die Abarbeitung des Algorithmus notwendige Informationen hinterlegt werden. Hierzu gehören das Wissen, ob eine Transition zwischen *Views* gemäß dem Screenflow-Modell des Mashups stattfindet, und die Mapping-Definition des Kommunikationskanals.

In einem Capability-Graphen können Knoten in strukturell besonderer Position auftreten. *Quell-Capability-Knoten* sind Capability-Knoten mit ausgehenden und keinen eingehenden Capability-Links, zum Beispiel `Search Flight` in Abbildung 5.8. Analog dazu sind *Senke-Capability-Knoten* solche mit eingehenden und keinen ausgehenden Capability-Links, beispielsweise `Display Weather`. Analog zum Capability-Metamodell aus Kapitel 5.1.1 können Capability-Knoten kompositer Natur sein, um komposite Capabilites zu repräsentieren. Diese vermögen es, sequentielle und parallele Abläufe der Kind-Capabilities auszudrücken. Beispielsweise ist `Plan Trip` eine Sequenz.

Neben der kausalitäts- und datenflussgetriebenen Perspektive des Capability-Graphen existiert ein überlagerter **Hierarchiegraph**, siehe Abbildung 5.8. Dieser entsteht durch das Bilden kompositer Capabilities und reflektiert somit Eltern-Kind-Beziehungen von Capability-Knoten und zugrundeliegenden Capabilities. An Capability-Knoten werden zu diesem Zweck Referenzen hinterlegt, um auf Kinder- und den Elternknoten zuzugreifen.

Besteht ein Capability-Graph aus mehreren getrennten, in sich zusammenhängenden Teilgraphen, werden letztere als *Capability-Ketten* bezeichnet. Intuitiv handelt es sich dabei um verschiedene Funktionsblöcke eines Kompositionsfragments.

Das Capability-Metamodell aus Kapitel 5.1.1 sieht vor, dass Konzepte aus Ontologien in OWL als Entity referenziert werden. Solche Konzepte können in vielfältiger Weise in Relation zueinander stehen, etwa als Subklassen und als Werte- oder Definitionsbereich von OWL-Properties. Im Rahmen des Bildens kompositer Capabilities ist es notwendig, auch eine möglichst aussagekräftige Entity zu benennen. Diese wird als **bestimmende Entity** bezeichnet. Sie ergibt sich durch die Analyse der semantischen Annotationen, vor allem der Entities, aller direkten Kinder des betrachteten Capability-Knotens. Das ist ausreichend, da das Bestimmen kompositer Capabilities bottom-up erfolgt, was in Abschnitt 5.3.4 detailliert wird. Bei der Analyse werden Vererbungsrelationen (`subClassOf`, `subPropertyOf`) genutzt, um grobgranularere Konzepte zu identifizieren, die andere Entities subsumieren. Weiterhin wird angenommen, dass eine Ontologieklasse C_1 eine andere C_2 aggregiert, falls OWL-Properties existieren, deren Definitionsbereich C_1 und Wertebereich C_2 ist. Das Szenario in Abbildung 2.2 liefert für C_1 als Beispiel `Appointment` und für C_2 `AppointmentLocation`, aufgrund der OWL-Property `hasLocation`. Abschnitt A.5.1 im Anhang illustriert ein Beispiel zur Berechnung der bestimmenden Entity.

Prinzipiell wird folgendermaßen vorgegangen, um für eine Menge von Entities E, die bestimmende Entity abzuschätzen. Zunächst werden aus E alle Einträge entfernt, die eine »untergeordnete Entity« eines anderen Eintrags sind. Dazu wird für jedes $e \in E$ die Menge L_e in zwei Schritten wie folgt berechnet:

1. Die maßgebliche Ontologie-Klasse c von e wird bestimmt, da auch Individuen ($c =$

Klasse von e) und OWL-Properties (c = Wertebereich von e) als Entity annotiert werden können.

2. Nachfolgende Ontologie-Konzepte werden L_e hinzugefügt: (a) Subklassen von c; (b) Falls es sich bei e um eine OWL-Property handelt, werden alle OWL-Properties q hinzugefügt, für die gilt q `subPropertyOf` e; (c) OWL-Properties, bei denen c als Definitionsbereich spezifiziert ist, und deren Wertebereich.

Schließlich wird jedes Element von L_e aus E entfernt, falls es in E vorkommt. L_e ist wegen Schritt 2 nicht zwangsläufig eine Teilmenge von E. Nachfolgend wird versucht, eine allen verbliebenen Elementen von E »übergeordnete Entity« zu identifizieren. Hierzu wird für jedes $e \in E$ die Menge U_e wie folgt berechnet:

1. Die maßgebliche Ontologie-Klasse c von e wird bestimmt, da auch Individuen (c = Klasse von e) und OWL-Properties (c = Wertebereich von e) als Entity annotiert werden können.

2. Folgende Ontologie-Konzepte werden in die Menge U_e hinzugefügt: (a) Superklassen von c, wobei mengentheoretische Klassendefinitionen, zum Beispiel durch Vereinigung und Enumeration, auf die einzelnen Konzepte reduziert werden. (b) Falls es sich bei e um eine OWL-Property handelt, werden alle OWL-Properties q hinzugefügt, für die gilt e `subPropertyOf` q; (c) OWL-Properties, bei denen c als Wertebereich spezifiziert ist, und deren Definitionsbereich.

Darauf aufsetzend werden mögliche bestimmende Entities für die Menge E per Schnittmengenbildung aller U berechnet, siehe Beispiel in Abbildung A.3. Dabei ist zu beachten, dass insbesondere auch die leere Menge als Ergebnis auftreten kann.

Der grundlegende Ansatz bringt Entities außer über die reine Vererbungshierarchie auch durch Object-Properties in eine Ordnung hinsichtlich der Granularität. Object-Properties können eine aggregierende Semantik besitzen, sodass das aggregierende Konzept das aggregierte »subsummiert«. Deswegen wird, jeweils in Schritt 2, in L_e der Wertebereich und in U_e der Definitionsbereich einer OWL-Property hinzugefügt. Bei Object-Properties ist jedoch zu bedenken, dass Zyklen auftreten können, wenn deren Definitions- und Wertebereich gleich oder wechselseitig sind. Dies wird erreicht durch Nutzung von *Symmetric-Properties* beziehungsweise durch Definition zweier einzelner Object-Properties, die als `inverseOf` zueinander deklariert wurden. Solche Fälle können detektiert werden und finden während der Berechnung von L_e und U_e Beachtung, indem solche OWL-Properties übersprungen werden. Sind derartige Zusammenhänge nicht explizit modelliert, muss angenommen werden, dass beide Richtungen eine eigenständige Semantik aufweisen und keine bestimmende Entity vorhanden ist.

5.3.3 Übersicht über den Algorithmus

Grundgedanken und Annahmen des verfolgten Ansatzes lassen sich wie folgt kurz umreißen. Der Kern der Anwendungsfunktionalität wird durch Capabilities von Komponenten und deren Zusammenspiel basierend auf Capability-Links erbracht. Durch transitive Verbindungen entstehen komplexere funktionale Zusammenhänge (Capability-Graph und Capability-Ketten) innerhalb der Anwendung. Basierend auf semantischen Informationen

aus den einzelnen Capabilities, Heuristiken und gelernten Daten können zusammengefasste Capabilities hergeleitet werden, welche die Funktionalität auf Ebene des jeweiligen Kompositionsfragments beschreiben.

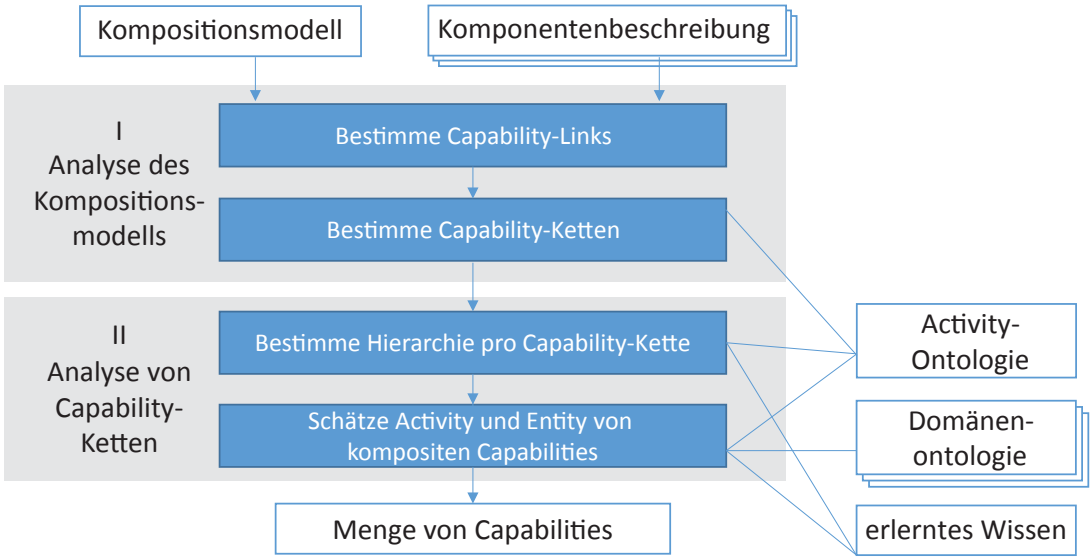


Abbildung 5.9: Eingaben, wesentliche Schritte und Ausgaben des Algorithmus

Daraus leitet sich die Arbeitsweise des Algorithmus ab, die Abbildung 5.9 veranschaulicht. Als Eingaben für den Algorithmus dienen das zu untersuchende Kompositionsmodell sowie die Beschreibungen sämtlicher darin auftretenden Komponenten. Es sei angemerkt, dass auch Pattern-Instanzen unterstützt werden, da aus diesen ein passendes Kompositionsmodell erzeugt werden kann. Weitere Datenquellen, insbesondere Ontologien und erlerntes Wissen, werden während der Abarbeitung einbezogen. In der ersten Phase wird das Kompositionsmodell analysiert. Dabei werden basierend auf dem Communication-Model, dem Conceptual-Model und dem Screenflow-Model sowie den Komponentenbeschreibungen Capability-Links bestimmt. Der entstandene Capability-Graph wird anschließend hinsichtlich Capability-Ketten untersucht. Pro Capability-Kette wird nachfolgend versucht, einen Hierarchiegraphen aufzubauen. Dieser stellt die Voraussetzung für das Abschätzen zusammengefasster Capabilities dar. Dabei wird das Wissen zu den in Entity und Activity referenzierten Konzepten aus den jeweiligen Ontologien herangezogen. Zusätzlich wird nach bereits bekannten Konstellationen in aus früheren Durchläufen erlerntem Wissen gesucht. Im Ergebnis liefert der Algorithmus eine (möglicherweise leere) Menge von Capabilities, welche die Funktionalität des gegebenen Kompositionsmodells repräsentieren. Dabei handelt es sich in der Regel um komposite Capabilities.

5.3.4 Detaillierter Ablauf

Nachdem das grundlegende Vorgehen skizziert wurde, folgt in diesem Kapitel die Detailspezifikation des Ansatzes zum Abschätzen der Capabilities eines beliebigen Kompositionsfragments. Abbildung 5.10 zeigt auf der linken Seite die Schrittfolge samt anfallender Artefakte und rechts ein schematisches Beispiel, das den jeweiligen Stand der Datenstrukturen und somit die sukzessive Herleitung von Capabilities veranschaulicht.

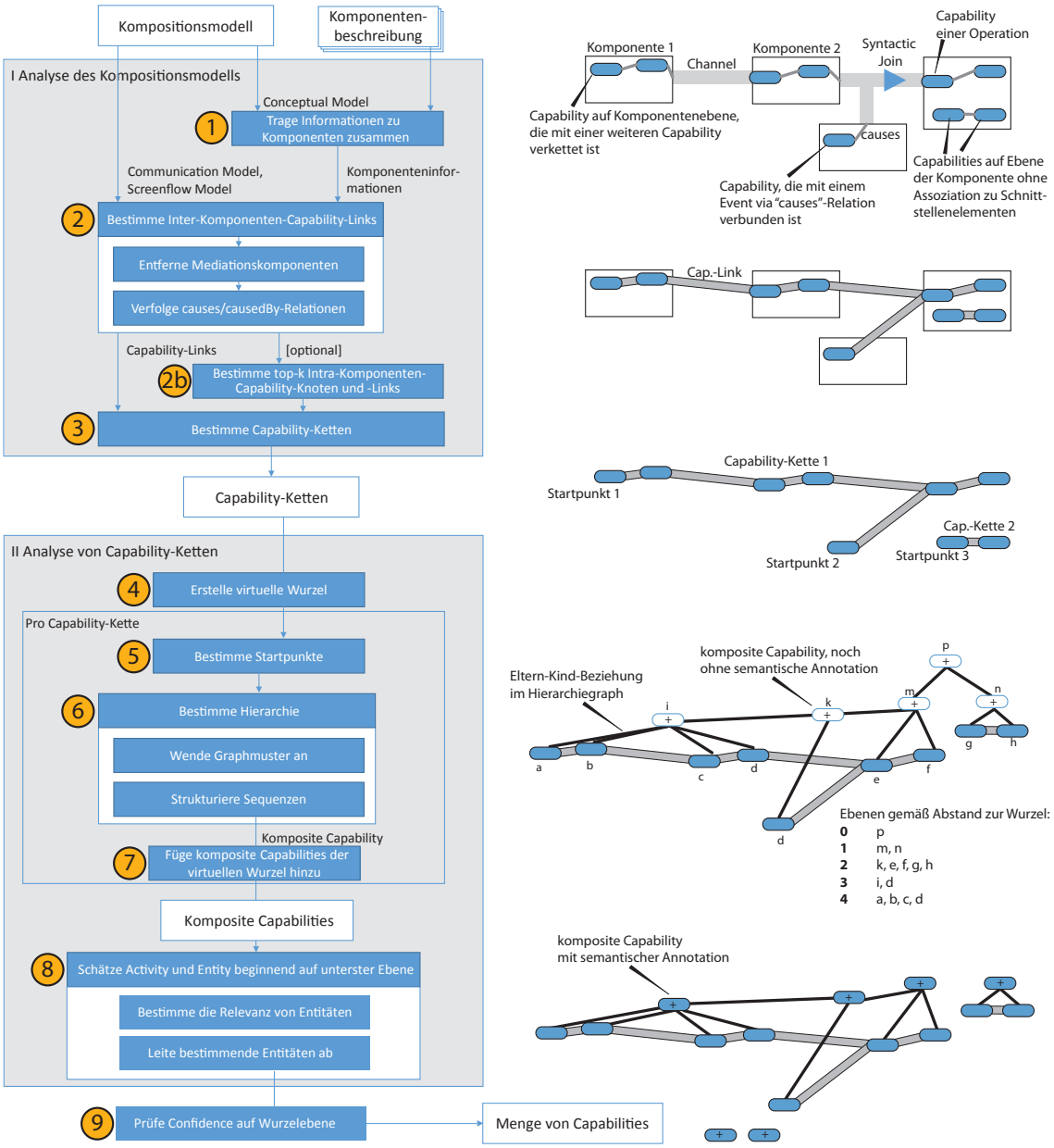


Abbildung 5.10: Detaillierter Ablauf des Algorithmus inklusive schematischem Beispiel

Phase I Analyse des Kompositionsmodells In der ersten Phase des Algorithmus wird das übergebene Kompositionsmodell und die Beschreibungen der darin enthaltenen Komponenten analysiert. Dazu werden zunächst *Komponenteninformationen* gemäß den Konzepten aus Kapitel 5.1.2 zusammengetragen und für ein effizientes Nachschlagen optimiert in Datenstrukturen abgelegt ①. Dies umfasst im Wesentlichen die Capabilities auf Komponentenebene und die auf Ebene von Komponentenschnittstellen mit der Zuordnung zu den entsprechenden Schnittstellenelementen. Begonnen wird dabei mit den Capabilities auf Komponentenebene gefolgt von denen auf Operationsebene, damit die Capability-Selektoren von Events und Properties erfolgreich ausgewertet werden können. Im Falle von Events werden die `causedBy`-Referenzen aufgelöst. Properties von Komponenten bedürfen ebenfalls einer gesonderten Behandlung. Ähnlich zu Events

besitzen diese lediglich Referenzen zu Capabilities mittels der Relationen `causes` und `causedBy`. Anders als Events wird jedoch davon ausgegangen, dass Properties eine generische Funktionalität bereitstellen. Daher wird eine implizite Capability cap_p mit Activity `Set` und Entity entsprechend dem semantischen Typ der Property erzeugt und mit den referenzierten Capabilities assoziiert, indem die `causes`/`causedBy`-Referenzen der Property auf cap_p übertragen werden.

Eine weitere Besonderheit sind *Mediationskomponenten*, die Syntactic Join durchführen. Da deren Beschreibung je nach Erfordernissen der aktuellen Anwendung generiert wird und ihnen lediglich eine syntaktische – weniger eine semantische – Bedeutung in der Anwendung zukommt, werden temporär »Pseudo-Capabilities« erzeugt. Diese haben keine semantischen Annotationen, erlauben es jedoch, Mediationskomponenten analog zu UI- und Nicht-UI-Komponenten in der weiteren Verarbeitung zu behandeln. In einem späteren Schritt werden sie ohnehin aus dem Capability-Graphen entfernt.

Auf Basis der gewonnenen Komponenteninformationen sowie des Communication-Model und des Screenflow-Model erfolgt in Schritt ② das Bestimmen von Capability-Links zwischen Komponenten. Dazu wird über alle Kommunikationskanäle im Communication-Model des Kompositionsmodells iteriert. Hierbei wird unterstellt, dass es nur Kanäle gibt, die genau ein Publisher- mit einem Subscriber-Schnittstellenelement verbinden. Alle anderen Konstellationen, wie ein Publisher und mehrere Subscriber, lassen sich auf diesen atomaren Fall zurückführen und würden die Komplexität des Algorithmus unnötig erhöhen. Für jeden Kanal werden relevante Publisher- und Subscriber-Capabilities bestimmt. Sollten mehrere Capabilities auf einer Seite vorhanden sein, finden folgende Filterschritte statt. Wie in Kapitel 5.1.2 eingeführt, können Operations mehrere verkettete Capabilities besitzen, zum Beispiel `Set Location` → `Search Hotel`. Diese werden dahingehend überprüft, ob sie ein Attribut `causes`, jedoch kein `causedBy` besitzen, um die erste Capability in der komponenteninternen Capability-Kette zu identifizieren. Auf Publisher-Seite findet zusätzlich eine Filterung von Capabilities statt, sofern der Capability-Selektor mehrere Capabilities mit `or` aufzählt, siehe Kapitel 5.1.1. Dann wird angenommen, dass die referenzierten Capabilities semantisch übereinstimmen und lediglich auf verschiedene Weisen erbracht werden, zum Beispiel `Select Location` als Texteingabe und mittels Marker auf einer Karte. Zudem werden dann Capabilities auf Komponentenebene bevorzugt, da sie ohne Kommunikationskanal als funktional angesehen werden können. Im Gegensatz dazu ist bei Auftreten eines Capability-Selektors, der Capabilities mit `and` verknüpft, keine Filterung vorgesehen, da angenommen wird, dass sich die Capabilities ergänzen. Als Beispiel hierfür kann ein Event dienen, das im Attribut `causedBy` auf die Capabilities `Select Location` und `Set Date` verweist.

Oft sind nach diesen Schritten für den Publisher und den Subscriber jeweils eine Capability als relevant erkannt. Im Allgemeinen können jedoch mehrere Capabilities auftreten, insbesondere auf Publisher-Seite. Letztlich wird für jede Kombination von relevanten Capabilities des Publishers und des Subscribers ein Capability-Link erzeugt. Zudem wird am Capability-Link gespeichert, ob eine View-Transition ausgelöst wird. Diese Information dient in der späteren Hierarchisierung zum Festlegen der Connectives von kompositen Capabilities.

Bisher enden Capability-Links an den Capability-Knoten der direkt mit einem Publisher oder Subscriber assoziierten Capabilities. Durch das Nachverfolgen der komponenteninternen Relationen `causes` und `causedBy` wird der Capability-Graph ergänzt. Dies stellt einen wesentlichen Schritt dar, um funktionale Zusammenhänge zu repräsentieren. Das

prinzipielle Vorgehen für einen Capability-Link cl gestaltet sich dabei wie folgt:

- Zuerst wird für die Capability des Ziel-Capability-Knotens von cl die Relation `causes` ausgewertet, falls vorhanden. Dann werden folgende Anweisungen für jede referenzierte Capability abgearbeitet: (1) Der zugehörige Capability-Knoten wird im Capability-Graphen identifiziert oder ein neuer erzeugt, falls die Capability bisher nicht enthalten ist. (2) Ein Capability-Link wird zwischen beiden Capability-Knoten erzeugt, sofern nicht schon vorhanden. Er wird zum Capability-Graphen hinzugefügt und die soeben definierten zwei Schritte für ihn ausgeführt.
- Für die Capability des Start-Capability-Knotens von cl wird die Relation `causedBy` ausgewertet, falls vorhanden. Grundsätzlich wird dabei analog zum vorherigen Schritt vorgegangen. Sollten jedoch mehrere Capabilities mit `or` verknüpft referenziert werden, findet eine Duplikaterkennung statt, wobei Capabilities auf Komponentenebene bevorzugt erhalten bleiben.

Sind alle Capability-Links abgearbeitet, ist dieser Zwischenschritt abgeschlossen. Anschließend werden die temporär erzeugten Pseudo-Capabilities von Mediationskomponenten aus dem Capability-Graphen entfernt. Dies erfolgt durch die in Abbildung 5.11 beispielhaft dargestellte Regel. Im Prinzip werden die Capability-Knoten der Mediationskomponente gelöscht und die vor- beziehungsweise nachgelagerten Capability-Knoten direkt verbunden. Damit wird die Natur von Syntactic Join auf die semantische Ebene des Capability-Graphen transferiert, indem ein Capability-Knoten mit mehreren eingehenden Capability-Links, und somit ein Synchronisationspunkt, entsteht.

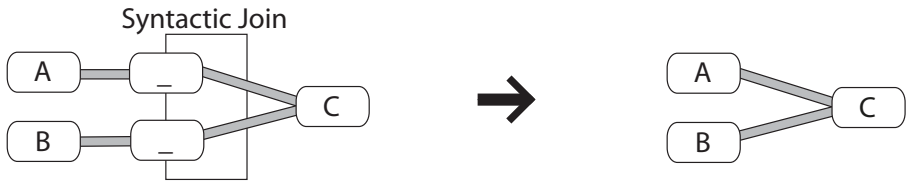


Abbildung 5.11: Entfernen einer Mediationskomponente aus einem Capability-Graphen

Optional werden komponenteninterne Funktionalitäten mit in den Capability-Graphen aufgenommen (2b). Dies ist vor allem notwendig, wenn keinerlei Kanäle zwischen Komponenten existieren, etwa bei einer in Entwicklung befindlichen CWA. Dann strebt der Algorithmus an, zumindest die wesentlichen Funktionalitäten der einzelnen Komponenten zu verarbeiten. Dazu wird für jede Komponente des Mashups nach Capabilities gesucht, die noch nicht im Capability-Graphen auftauchen und keine Operation-Capabilities sind. Zudem wird sichergestellt, dass die Capabilities nicht exklusiv von Operationsaufrufen abhängen. Einzelne unverkettete Capabilities mit einer Activity, die Subklasse von `Input` ist, z. B. Eingeben von Text, werden aufgrund des geringen Beitrags zur Anwendungsfunktionalität verworfen. Anschließend werden wie oben beschrieben die Relationen `causes` und `causedBy` nachverfolgt, um Verkettungen im Capability-Graphen abzubilden.

Danach wird der Capability-Graph dahingehend geprüft, ob Quell-Capability-Knoten zu finden sind, die mehrere ausgehende Capability-Links aufweisen. Solche Knoten werden pro Capability-Link kopiert und nur mit diesem assoziiert, sodass mehrere Capability-Ketten entstehen können. Dies vereinfacht nachfolgende Analyseschritte, insbesondere die Herleitung kompositer Capability-Knoten im Hierarchiegraphen.

Im Anschluss werden Capability-Ketten im Capability-Graph identifiziert ③. Dazu ist es notwendig, sämtliche Quell-Capability-Knoten zu iterieren und solange den Capability-Links zu folgen bis entweder ein Ziel-Capability-Knoten keine ausgehenden Capability-Links mehr aufweist oder Teil einer anderen Capability-Kette ist. In letzterem Fall wird die momentan im Aufbau befindliche Capability-Kette mit der vorhandenen verschmolzen.

Phase II Analyse von Capability-Ketten Am Ende von Phase I liegt ein Capability-Graph vor, der potentiell aus mehreren Capability-Ketten besteht, siehe rechte Seite, dritter Graph von oben in Abbildung 5.10. Die Semantik dieser funktionalen Zusammenhänge ist an dieser Stelle jedoch noch nicht explizit bekannt und wird daher in Phase II des Algorithmus abgeschätzt.

Zunächst wird eine *virtuelle Wurzel* in Form eines kompositen Capability-Knotens erzeugt ④, der sämtliche komponenteninternen Capability-Knoten als Kinder mit paralleler Abfolge beinhaltet. Anschließend wird für jede Capability-Kette folgende Prozedur durchgeführt. Zuerst ist die Bestimmung der Quell-Capability-Knoten der Kette notwendig ⑤. Diese sind Ausgangspunkt für das Erstellen der überlagerten Hierarchie ⑥, wobei das Konzept aus [Kur13] aufgegriffen und erweitert wird. Im Kern umfasst es das Identifizieren bestimmter Muster im Capability-Graphen, welche vordefinierte Auswirkungen auf den entstehenden überlagerten Hierarchiegraphen haben. Diese Muster sind angelehnt an *Workflow Patterns* [Aal+03] und in Abbildung 5.12 samt der resultierende Struktur im überlagerten Hierarchiegraphen illustriert.

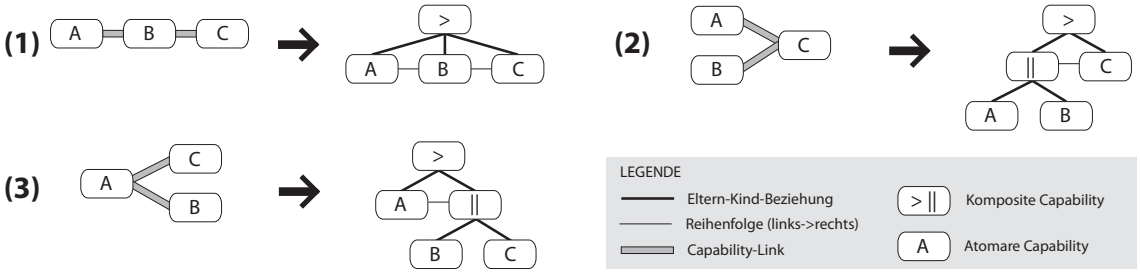


Abbildung 5.12: Unterstützte Graphmuster zur Strukturierung kompositer Capabilities

(1) Sequence: Im Datenfluss oder durch `causes`-Beziehungen aufeinander folgende Capability-Knoten bilden Sequenzen und werden einem kompositen Capability-Knoten mit Connective `SEQ` zugeordnet. Die Reihenfolge bleibt dabei erhalten. Einer Sequenz sind mindestens zwei Capability-Knoten zugeordnet.

(2) Synchronization: Treffen an einem Capability-Knoten mehrere Capability-Links zusammen, handelt es sich um einen Synchronisationspunkt. Die Start-Capability-Knoten der eingehenden Capability-Links werden einem kompositen Capability-Knoten unterstellt, dessen Connective auf `PAR` gesetzt wird. Zudem findet ein Gruppieren dieses kompositen Capability-Knotens und des Ziel-Capability-Knotens (C in der Abbildung) unterhalb eines weiteren kompositen Capability-Knotens mit Connective `SEQ` statt.

(3) Parallel-Split: Dieses Muster ist durch einen Capability-Knoten gekennzeichnet, der mehrere ausgehende Capability-Links aufweist. Hierbei werden alle Ziel-Capabi-

lity-Knoten einem kompositen Capability-Knoten mit Connective `PAR` zugeordnet. Dieser wiederum bildet zusammen mit dem Start-Capability-Knoten (A in der Abbildung) eine Sequenz.

Die identifizierten Sequenzen werden weiter untergliedert, indem anhand der annotierten Activity-Konzepte nach Teilfunktionen gesucht wird. Der Grundgedanke entspricht dabei dem bereits erwähnten systemtheoretischen Ansatz. Daher wird nach potentiellen Übergängen zwischen Subsequenzen gesucht. Solche Punkte befinden sich nach Capabilities, deren Activity-Konzepte Subklasse von `Output` sind und die nicht von weiteren Capabilities mit solchen Activity-Konzepten gefolgt werden, zum Beispiel `Display Location` → `Search Route`. Weitere Übergangspunkte befinden sich nach Capabilities, deren Activity-Konzepte von `Transform` erben und die vor einer Capability stehen, deren Activity-Konzept Subklasse von `Input` ist, wie `Search Hotel` → `Add Hotel`. Eine Unterteilung findet nur statt, falls alle der potentiellen Subsequenzen nach Unterteilung aus mindestens zwei Capability-Knoten bestehen würden. Abbildung 5.13 zeigt im unteren Teil beispielhaft das Ergebnis der Unterteilung einer längeren Ausgangssequenz (oben). Die Übergangspunkte sind dabei mit ⓐ und ⓑ gekennzeichnet.

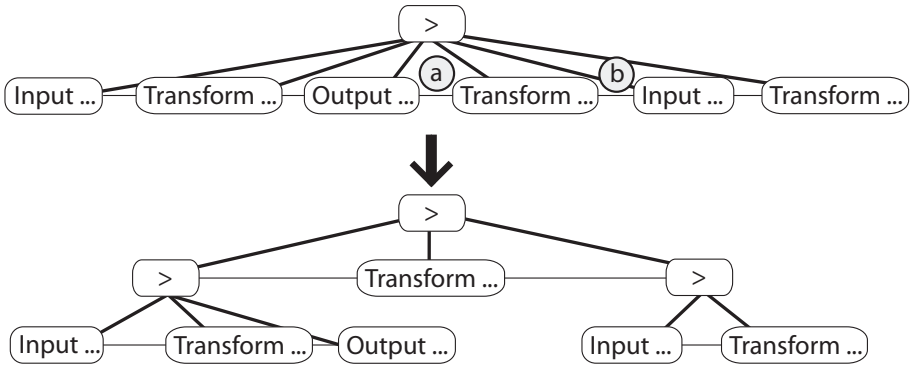


Abbildung 5.13: Beispiel der Unterteilung einer Sequenz in Teilsequenzen

In oben genanntem Regelwerk findet der Screenflow noch keine Beachtung. Das Konzept sieht vor, dass basierend auf den an Capability-Knoten hinterlegten View-Zugehörigkeiten und den Informationen zu View-Transitionen an Capability-Links (vergleiche Kapitel 5.3.2) der Screenflow in die Hierarchisierung mit einfließt. Die Regeln (2) und (3) werden deshalb wie folgt erweitert. Das *Connective*, vergleiche Kapitel 5.1.1, des kompositen Capability-Knotens, der die Start-Capability-Knoten im Fall von Synchronization beziehungsweise die Ziel-Capability-Knoten im Fall von Parallel-Split beinhaltet, wird in Abhängigkeit eines View-Wechsels gesetzt. Findet ein solcher nicht statt, gelten die Regeln unverändert und das Connective wird auf `PAR` gesetzt. Ansonsten wird `SEQ` als Connective angegeben. Beispielhaft illustriert dies Abbildung 5.14 für eine Synchronisation an `Search Flight`. Hauptantrieb hinter diesem Ansatz ist die Sicht des Nutzers. Sind Funktionalitäten über verschiedene Ansichten auf die Anwendung verteilt, etwa bei begrenzter Bildschirmgröße auf einem mobilen Endgerät, kann der Nutzer diese typischerweise nur nacheinander abarbeiten, auch wenn laut Datenfluss keine Reihenfolge vorgeschrieben wäre.

Schließlich wird der abgeleitete komposite Capability-Knoten, welcher die abgeleitete Hierarchie repräsentiert, den Kindknoten der virtuellen Wurzel hinzugefügt ⑦.

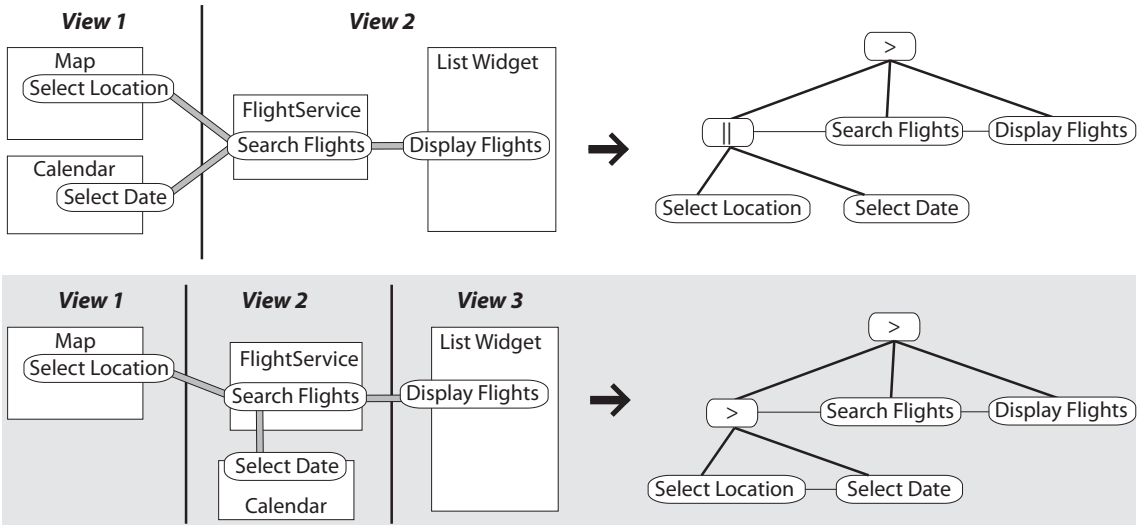


Abbildung 5.14: Beispiel für den Einfluss des Screenflows auf den Hierarchiegraph

Dem Capability-Graphen wurde zwar der Hierarchiegraph überlagert, jedoch sind zu diesem Zwischenstand die resultierenden kompositen Capability-Knoten noch nicht mit Semantik hinterlegt. Letztere steckt zu diesem Zeitpunkt implizit in den Kind-Elementen und deren Strukturierung. Ein derartiger Zustand ist in Abbildung 5.10 rechts im Beispiel an vierter Stelle zu sehen, wobei die fehlende Semantik der kompositen Capability-Knoten durch fehlende Formfüllung angedeutet ist. Deshalb wird in Schritt ⑧ die Semantik abgeschätzt. Dabei werden sämtliche kompositen Capabililty-Knoten in Ebenen gemäß dem Abstand zur Wurzel eingeordnet. Anschließend wird auf der untersten Ebene mit dem größten Abstand begonnen. Ziel ist es, durch das Auswerten der Capabilities aller Kindknoten $caps_{children}$, die wahrscheinlichste Capability des Elternknotens herzuleiten. Dazu notwendiges externes Wissen stammt aus den Activity- und Domänenontologien sowie aus Trainingsdaten und erlerntem Wissen, das in Form von Hierarchie- und Capability-Graphen aus vorherigen Durchläufen bekannt ist.

Zunächst wird nach bekannten Lösungen aus den Trainingsdaten und erlerntem Wissen gesucht. Dabei wird geprüft, ob es einen Hierarchiegraphen gibt, der eine komposite Capability enthält, die mit den aktuell untersuchten strukturell übereinstimmt. Dazu müssen im Wesentlichen das Connective und die Capabilities sowie gegebenenfalls die Reihenfolge der Kindknoten auf Gleichheit untersucht werden. Kann ein passender Knoten identifiziert werden, wird direkt dessen Capability als Ergebnis cap_{result} übernommen.

Ansonsten wird die bestimmende Entity $entity_{dom}$ für $caps_{children}$ gesucht. Anschließend wird für jedes Entity-Konzept, das in $caps_{children}$ vorkommt, ein Rating berechnet. Dieses ergibt sich aus den nachfolgend gelisteten Kriterien. Analog zum *TRank*-Ansatz [Ton+13] werden verschiedene Metriken angewendet, welche sich auf Individuen- und Klassenebene der Domänenmodelle (r_{sem}) beziehen oder statistische Zusammenhänge auswerten (r_h). Darüber hinaus finden Metriken Anwendung, die Charakteristika von Capabilty-Graphen einbeziehen (r_a und r_s).

- Das Activity-Rating r_a entspricht dem Maximum der Ratings für die Activities, mit denen die Entity auftritt. Die Ratings orientieren sich an den Superklassen für Activities und folgen der Ordnung `Transform` > `Output` > `Input`, zum Beispiel

$r(search) = 1$, $r(display) = 0.7$ und $r(select) = 0.4$. Besitzen mehrere Activities das gleiche Gewicht, wird erlerntes Wissen einbezogen.

- Strukturrating r_s trifft Aussagen über die Relevanz einer Entity gemessen an der Einordnung im Datenfluss. Folgende Gegebenheiten wirken steigernd auf r_s: (1) die Stellung am Ende von Sequenzen, (2) das Auftreten an kompositen Capability-Knoten und (3) an Capability-Knoten, die an Capability-Links teilhaben, welche wiederum von Kommunikationskanälen abgeleitet wurden.

- Das Häufigkeits-Rating r_h gibt die relative Häufigkeit der Entity in der zu untersuchenden Menge aller Entities an.

- Das Semantik-Rating r_{sem} gibt an, ob es sich um die bestimmende Entity handelt.

Das Gesamtrating für eine Entity ergibt sich gemäß folgender Formel.

$$rating_{entity} = r_a + r_s + r_h + r_{sem}$$

Sollten letztlich mehrere Entities mit dem gleichen Rating auftreten, wird, wie in Kapitel 5.3.2 beschrieben, untersucht, ob eine Entity die bestimmende bezüglich dieser Menge ist, und deren Rating gegebenenfalls erhöht. Zudem wird im erlernten Wissen nachgesehen, ob eine Lösung für die verbliebene Menge an Entities bekannt ist.

Schließlich wird eine komposite Capability cap_{result} erzeugt mit der höchstbewerteten Entity, der Activity mit dem höchsten Gewicht und dem zugehörigen Activity-Modifier. Weiterhin wird für cap_{result} die Domäne bestimmt. Diese entspricht in erster Näherung der Domäne, die durch die von der Entity referenzierte Ontologie beschrieben wird. Daneben wird unter bestimmten Umständen auch der Entity-Context von cap_{result} hergeleitet und festgelegt. Dabei werden zunächst Capability-Links identifiziert, denen ein Kommunikationskanal zugrunde liegt, der eine Mapping-Definition mit mindestens einem Semantic Split aufweist, vergleiche Kapitel 5.2.2. Für jeden betroffenen Capability-Link wird danach geprüft, ob dessen Start-Capability c_s eine Eltern-Capability c_p im Hierarchiegraphen besitzt. Ist dies der Fall, entspricht der Entity-Context von c_p der Entity von c_s. Dies ist in Abbildung 5.15 illustriert. Angenommen der Capability-Link auf der linken Seite der Abbildung entstammt einem Kommunikationskanal, bei dem ein Semantic Split die `Location` des `POI` projiziert. Aus dem Capability-Link wird wie bekannt eine Sequenz abgeleitet und weiterhin der Entity-Context auf `POI` gesetzt. Dies wird durch den Zusatz »of POI« in Abbildung 5.15 verdeutlicht und dient dazu, die Semantik der Capability der Sequenz zu schärfen, die ansonsten lediglich `Show Location` lauten würde.

Abbildung 5.15: Beispiel für das Ableiten des Entity-Context einer kompositen Capability

Zudem wird ein *Confidence*-Wert unter Verwendung der Ratings berechnet und bei cap_{result} hinterlegt. Das Konzept dabei ist, dass dieser Wert direkt proportional zum

Abstand des höchsten und zweithöchsten $rating_{entity}$ ist. Existiert nur eine Entity, ist der *Confidence*-Wert hoch, falls nur eine Activity existiert, ansonsten tendenziell niedrig, wobei erlerntes Wissen einbezogen werden kann.

Als Ergebnis liegt zu diesem Zeitpunkt eine einzelne Hierarchie vor, deren oberster Knoten die virtuelle Wurzel ist. Um die Plausibilität zu erhöhen, wird geprüft, ob der vom Algorithmus hinterlegte *Confidence*-Wert am Wurzelknoten einen Schwellenwert t_{min} übersteigt. Letzterer ist in Abhängigkeit von den gewählten Gewichtungen in Schritt ⑧ konkret zu definieren und empirisch zu validieren. Kann t_{min} nicht überschritten werden, wird die Wurzelebene aufgelöst und es resultiert eine Menge von Capabilities. Ansonsten bleibt die Wurzelebene erhalten und als Ergebnis liegt eine einzelne Capability vor.

5.3.5 Architekturüberblick

Schwerpunkt dieses Kapitels ist die Konzeption einer Architektur zur Bereitstellung des Algorithmus und die Einordnung dieser in die bestehende Plattform CRUISE. Abbildung 5.16 illustriert wesentliche Module, die zum Erbringen des zuvor spezifizierten Algorithmus beitragen oder diesen verwenden.

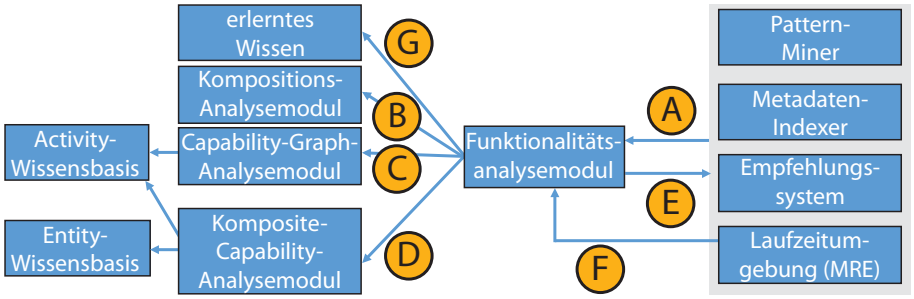


Abbildung 5.16: Architekturüberblick des Subsystems zur Capability-Abschätzung

Eine zentrale Stellung nimmt das *Funktionalitätsanalysemodul* ein. Es stellt eine *Fassade* [Gam+94] für das gesamte Subsystem zur Funktionalitätsabschätzung dar und nimmt sämtliche Anfragen durch Clients entgegen. Dazu bietet das Modul eine Schnittstelle, anhand derer für ein gegebenes MCM oder Pattern eine Menge von Capabilities geliefert wird. Es orchestriert sämtliche anderen Module zum Erbringen des Algorithmus und somit zum Beantworten der Anfrage. Weiterhin ist es zuständig für vor- und nachbereitende Aufgaben, wie die Transformation von Datenstrukturen, für das Aufbauen von neuen Datenstrukturen wie bei der Feststellung von Ebenenzugehörigkeiten, und für Schritt ⑨ des Algorithmus. Eingehende Anfragen von Clients nimmt das Funktionalitätsanalysemodul entgegen Ⓐ und führt gegebenenfalls Konvertierungen durch. Danach Ⓑ wird das *Kompositions-Analysemodul* damit beauftragt, die Schritte ①, ② und optional ②b abzuarbeiten. Als Ergebnis liegen sämtliche relevanten Capability-Links der Anwendung vor. Diese werden an das *Capability-Graph-Analysemodul* überreicht Ⓒ, das Capability-Ketten identifiziert ③ und für jede die überlagerte Hierarchie ableitet ④ – ⑦. Es benötigt zusätzlich Wissen über Activity-Konzepte, um Sequenzen in Schritt ⑥ zu strukturieren. Dazu dient die *Activity-Wissensbasis*. Sie kapselt den Zugriff auf die Activity-Ontologie und dient dazu, die Klassifikation einer gegebenen Activity hinsichtlich der Superklassen `Input`, `Transform` und `Output` vorzunehmen. Die Hierarchie beinhaltet

zu diesem Zeitpunkt noch komposite Capabilities ohne semantische Annotationen, das heißt, Activity, Entity und so weiter sind noch undefiniert. Basierend darauf, organisiert das Funktionalitätsanalysemodul danach die Abschätzung der semantischen Annotationen ⑧ durch das *Komposite-Capability-Analysemodul*. Als Vorbereitung analysiert das Funktionalitätsanalysemodul die Ebenenzugehörigkeit sämtlicher kompositen Capabilities und beauftragt – beginnend bei der untersten Ebene kompositer Capabilities – pro Ebene und pro darin befindlicher Capability das Komposite-Capability-Analysemodul Ⓓ. Letzteres erbringt alle notwendigen Teilschritte, wie das Ranking von Entities. Dabei nutzt es die *Entity-Wissensbasis*, welche die Funktionalität zum Berechnen von bestimmenden Entities bereitstellt. Schließlich führt das Funktionalitätsanalysemodul Schritt ⑨ durch und übergibt dem Client die resultierende Menge an Capabilities Ⓔ.

Infrage kommende Clients sind das MRE, das Empfehlungssystem, der Metadaten-Indexer des Anwendungsrepositoriums und diverse Pattern-Miner. Die Verbindung Ⓕ zwischen MRE und Funktionalitätsanalysemodul deutet den Rückfluss von Feedback an. Abhängig vom jeweiligen Anwendungsfall ist es vorgesehen, dass Nutzer angeben können, ob die abgeleitete Funktionalität treffend ist oder nicht, zum Beispiel bei Vorschlägen zur Anwendungsklassifikation im Referenzszenario aus Kapitel 2.2.1. Dieses Feedback wird an das Funktionalitätsanalysemodul übermittelt, das anschließend die entsprechende Konstellation aus Kompositionsfragment und abgeleiteter Capability in das erlernte Wissen übernimmt Ⓖ.

5.4 Erzeugung eines Capability-Wissensgraphen

Dieser Abschnitt stellt eine graphbasierte Wissensquelle vor, auf welche insbesondere im Rahmen der Berechnung von Empfehlungen zugegriffen wird, vergleiche Kapitel 6.3.3. Ziel ist das Modellieren von erkannten, möglicherweise hypothetischen Relationen zwischen Capabilities. Dazu wird Wissen expliziert, welches in Capabilities von Kompositionsfragmenten implizit enthalten ist. Das hat den Vorteil, dass das Wissen effizienter abgefragt werden kann. Durch das Separieren in ein zusätzliches Modell bleiben zudem die ursprünglichen Ontologien und Capability-Graphen unberührt.

5.4.1 Struktur des Wissensgraphen

Das Wissen beinhaltet ein gewichteter Multigraph $(V, E, f(E))$, dessen Knoten V Capabilities repräsentieren, die über gerichtete, gewichtete Kanten E verbunden sind. Funktion f weist einer Kante $(c_1, c_2) \in E$ ein Tupel (r, v) zu, wobei r eine Relation und $v = r(c_1, c_2)$ ein Gewicht oder Wert der Relation mit $v \in [0, 1]$ ist. Diese Relationen beschreiben statistische und semantische Zusammenhänge von Capabilities und können zwischen zwei Capabilities c_a und c_b jeweils höchstens einmal auftreten. Abbildung 5.17 veranschaulicht die Struktur des Wissensgraphen sowie dessen Bezüge zu Domänenontologien einerseits und Capability- sowie Hierarchiegraphen von Kompositionsfragmenten andererseits. Der Wissensgraph umfasst kondensiertes Wissen, indem von Capability-Details wie verschiedenen zugrundeliegenden Schnittstellenelementen und von Hierachiegraphen abstrahiert wird. Beispielsweise existiert für eine atomare und eine komposite Capability `Search Hotel` ein gemeinsamer Knoten im Graphen. Dass keine Dopplungen von Knoten auftreten, wird anhand der Tupel aus den semantischen Annotationen zu Activity und Entity als identitätsstiftendes Merkmal durchgesetzt. Diese Tupel sind geeignet, da sich die

Referenzen in Form von Uniform Resource Identifiers (URIs) gemäß den Konzepten des Semantischen Webs durch Eindeutigkeit auszeichnen. Nachfolgend werden die verschiedenen Relationen beschrieben, während Ansätze zur Ermittlung der zugehörigen Werte in Kapitel 5.4.2 behandelt werden.

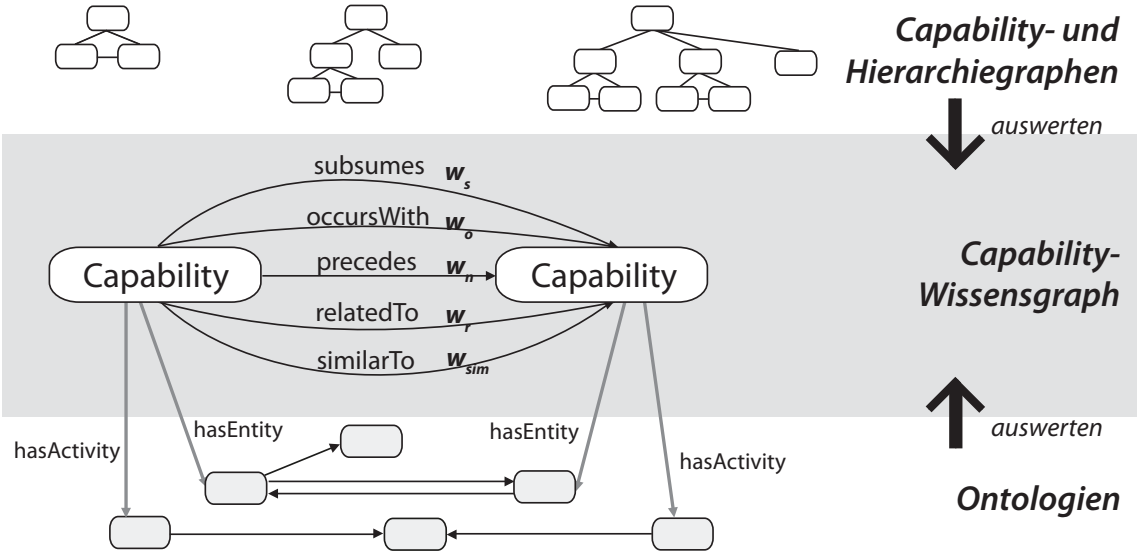


Abbildung 5.17: Aufbau eines Capability-Wissensgraphen und Bezug zu Domänenontologien

precedes: Die Relation beschreibt die Wahrscheinlichkeit w_n, dass c_a in einer sequentiellen Abfolge vor c_b auftritt.

subsumes: Diese Relation modelliert die Wahrscheinlichkeit w_s dafür, dass c_b in c_a enthalten ist, d. h., dass eine Eltern-Kind-Beziehung vorherrscht.

occursWith: Die Relation beschreibt den Umstand, dass c_a mit Wahrscheinlichkeit w_o mit c_b zusammen auftritt. Intuitiv gilt aufgrund der schwachen Bedingung immer $w_o \geq w_s$ und $w_o \geq w_n$ für jedes Paar (c_a,c_b).

Semantische Relationen dienen dem Materialisieren von Wissen, das in den semantischen Konzepten von Activity, Entity, etc. der Capabilities kodiert ist. Dabei werden einerseits Äquivalenz- und Vererbungsbeziehungen ausgewertet, um hierarchische Bezüge herzustellen. Andererseits gilt es, inhaltliche Zusammenhänge zu identifizieren, die durch Assoziationen, etwa via OWL-Properties, modelliert sind. Damit findet analog zu SKOS [@SKOS] eine Trennung in hierarchische und andere inhaltliche Beziehungen statt.

relatedTo: Diese Relation drückt aus, dass c_a und c_b gemessen an den semantischen Annotationen einen inhärenten inhaltlichen Bezug aufweisen, der jedoch nicht auf Vererbung sondern auf Assoziationen zwischen Ontolgiekonzepten zurückzuführen ist.

similarTo: Die Relation umschreibt den Grad der semantischen Ähnlichkeit w_{sim} von c_a zu c_b, basierend auf Vererbungsbeziehungen der Entities und Activities.

Insgesamt steht mit dem spezifizierten Graph eine wichtige zusätzliche Wissensquelle zur Verfügung, die das Domänenwissen aus Ontologien anreichert und darüber hinaus statistische Zusammenhänge repräsentiert. Der nächste Abschnitt geht der Frage nach, wie ein solcher Wissensgraph mit Informationen angereicht wird.

5.4.2 Instanziierung des Wissensgraphen

Wie in Abbildung 5.17 angedeutet ist, wird der Graph aus zwei Datenquellen befüllt: annotierte Capability- und Hierarchiegraphen und die zur Annotation genutzten semantischen Konzepte in den jeweiligen Domänenontologien. Beim Hinzufügen und beim Aktualisieren von Informationen wird aus den ausgewerteten Capabilities das oben beschriebene identitätsstiftende Tupel berechnet und daraufhin im Graph ein neuer Knoten angelegt oder ein vorhandener identifiziert.

Statistische Auswertung von Capability-Graphen

Aus Komponentenbeschreibungen und den darin vorhandenen Capabilities werden Informationen zu den Relationen *precedes* und *occursWith* gewonnen. Die in SMCDL verfügbaren Attribute *causes* und *causedBy*, vergleiche Kapitel 5.1.2, tragen zum Wert der Relation *precedes* bei. Alle innerhalb einer SMCDL-Instanz vorkommenden Capabilities werden auf die Relation *occursWith* abgebildet. Da in SMCDL keine kompositen Capabilities vorzufinden sind, entfällt die Anreicherung der Relation *subsumes*.

Die statistische Analyse der Capabilities von CWA und Pattern-Instanzen stellt eine weitere essentielle Datenquelle dar. Capability- und Hierarchiegraphen werden hinsichtlich sämtlicher statistischer Relationen untersucht. Analog zu Komponentenbeschreibungen erfolgt die Analyse bezüglich der Relation *occursWith*. Informationen zur Relation *precedes* sind aus den Capability-Ketten ablesbar. Basierend auf Hierarchiegraphen werden Zusammenhänge zu *subsumes* aufgedeckt.

Semantische Auswertung von annotierten Konzepten

Unter Verwendung semantischer Technologien sind die Ähnlichkeit von Capabilities und die semantische Nähe von Konzepten explizierbar. Hierzu findet ein paarweiser Vergleich sämtlicher Knoten des Wissensgraphen statt. Für jedes Paar (c_a, c_b) werden wie folgt die semantischen Relationen *similarTo* sowie *relatedTo* qualifiziert.

similarTo Die Ähnlichkeit von atomaren Capabilities sim_{atomic} basiert gemäß untenstehenden Formeln vorrangig auf Vererbungsbeziehungen der annotierten Konzepte. Den Grundbaustein für die Metrik stellt ein semantikbasierter Vergleich der in den Kernattributen einer Capability, siehe Kapitel 5.1.1, annotierten Ontologiekonzepte dar. Dazu dient die Funktion $semSim$, die für zwei Ontologiekonzepte die Ähnlichkeit, analog zu Ansätzen wie [PRM11b; Tie+12; BDM13b], basierend auf Subsumption bestimmt. Sollen zwei Konzepte, die keine Ontologieklassen sind, verglichen werden, wird zunächst die maßgebliche Klasse wie in Abschnitt 5.3.2 bestimmt. Für zwei Ontologieklassen prüft $semSim$ auf Identität, Äquivalenz und Subsumption.

$$semSim(Class_1, Class_2) = \begin{cases} 1 & \Leftrightarrow Class_1 = Class_2 \\ 0.9 & \Leftrightarrow Class_1 \equiv Class_2 \\ 0.7 & \Leftrightarrow Class_1 \sqsubseteq Class_2 \vee Class_2 \sqsubseteq Class_1 \\ 0 & sonst \end{cases}$$

$$semSim_{act}(A_1, A_2) = \begin{cases} 0.5 & super(A_1) = super(A_2) \\ semSim(A_1, A_2) & sonst \end{cases}$$

Konzepte, die als *Activity* annotiert wurden, sind ein Sonderfall, da die genutzte Ontologie bekannt ist und einer Referenzstruktur folgt. Daher wird neben den zuvor genannten Beziehungen in $semSim_{act}$ zusätzlich geprüft, ob die Konzepte den gleichen übergeordneten Klassen angehören, und dies gegebenenfalls mit einem geringeren Ähnlichkeitsgrad berücksichtigt. Mithilfe der Funktion $super$ erfolgt die Zuordnung eines Activity-Konzepts zu den drei Klassen `Input`, `Output` und `Manipulate`. Die Ähnlichkeit atomarer Capabilities sim_{atomic} ist wie folgt definiert ($\alpha + \beta = 1$):

$$sim_{atomic}(c1, c2) = \Big(\alpha * semSim_{act}(c1_{act}, c2_{act}) \ + \beta * semSim(c1_{actMod}, c2_{actMod})\Big) * \Big(\alpha * semSim(c1_{ent}, c2_{ent}) \ + \beta * semSim(c1_{entCtx}, c2_{entCtx})\Big)$$

Diese Formel hat den Vorteil, dass nur ein hoher Wert zustande kommt, wenn sowohl *Entity* als auch *Activity* in hohem Maße übereinstimmen. Durch das Einbeziehen von Activity-Modifier und Entity-Context wird zudem feinen semantischen Unterschieden von Capabilities Rechnung getragen. Die Faktoren $\alpha, \beta \in [0, 1]$ dienen dazu, den Einfluss der Modifikatoren zu mindern. Beispiele zur Veranschaulichung der Funktionsweise von sim_{atomic} mit $\alpha = \frac{2}{3}$ und $\beta = \frac{1}{3}$ enthalten die Tabellen A.1 und A.2 im Anhang A.

relatedTo Für das Abschätzen des inhärenten inhaltlichen Bezugs von Capabilities dienen in erster Linie die in Domänenontologien modellierten semantischen Zusammenhänge. Im Rahmen dieser Arbeit wird von einer Wissensmodellierung in OWL ausgegangen. Daneben existieren Ansätze wie SKOS. Zum Unterstützen dieser müsste das Hauptaugenmerk auf die passenden Ontologie-Properties gelegt werden, zum Beispiel `related` im Fall von SKOS. Der Wert $relatedTo(c_a, c_b)$ ergibt sich direkt aus dem semantischen Bezug zwischen den Entities e_a und e_b, notiert als $entityRelatedness(e_a, e_b)$, der wie folgt berechnet wird. Falls es OWL-Properties mit e_a als Definitions- und e_b als Wertebereich gibt, wird der höchste Bezug angenommen ($entityRelatedness(e_a, e_b) = 1$). Ist dies nicht gegeben, wird geprüft, ob OWL-Properties mit e_b als Definitions- und e_a als Wertebereich existieren und gegebenenfalls mit $entityRelatedness(e_a, e_b) = 0.8$ ein geringerer Bezug attestiert. Dieser Ansatz stellt einen Teilaspekt der Definition einer *bestimmenden Entity* dar, vergleiche Kapitel 5.3.2. Sollte auf diese Weise noch kein semantischer Bezug festgestellt werden können, wird getestet, ob e_a und e_b zumindest der selben Domäne angehören. Wie in Abschnitt 5.3.4 beschrieben, gilt dies in Näherung, sobald sie in der selben Ontologie modelliert sind. In diesem Fall gilt $entityRelatedness(e_a, e_b) = 0.5$ ansonsten $entityRelatedness(e_a, e_b) = 0$. Beispielsweise gilt $entityRelatedness(Hotel, Location) = 1$ unter der Annahme, dass das Konzept `Hotel` über die OWL-Property `atLocation` mit `Location` assoziiert ist. Um dem Umstand gerecht zu werden, dass nur eine Verbindung entgegen der Assoziationsrichtung existiert, ergibt $entityRelatedness(Location, Hotel) = 0.8$.

5.5 Zusammenfassung

Bezugnehmend auf den geschichteten Aufbau des Gesamtkonzepts in Abbildung 4.1 wurden in diesem Kapitel weite Teile der Modellgrundlage sowie zentrale Algorithmen und Funktionsblöcke spezifiziert. An dieser Stelle werden die entwickelten Konzepte in erster Linie hinsichtlich der Anforderungen abgeglichen. Eine tiefgehende Diskussion der wissenschaftlichen Beiträge, Stärken und Grenzen der gewählten Ansätze erfolgt hingegen in Kapitel 8. Die vorgeschlagenen Basismodelle bestehend aus Capabilities, annotierten Komponenten und Nutzer- sowie Feedbackmodell bieten eine ausdrucksstarke Grundlage für ein Empfehlungssystem sowie für EUD-Werkzeuge. Insbesondere Capabilities dienen im weiteren Verlauf der Arbeit aufgrund ihres inhärenten Bezugs zum Domänenwissen sowie zum UI und durch die Deklaration von Interaktionen als zentrales Hilfsmittel zur Kommunikation mit Nicht-Programmierern. Wie in Anforderung 2 vorgesehen, umfasst das Konzept semantische Datenmediation, siehe Abschnitt 5.2. Diese erfüllt sämtliche gestellte Teilanforderungen. Zur Erfüllung von Anforderung 7 tragen mehrere Konzeptbestandteile in Kombination bei. Patterns entsprechen einem Modell für strukturelles Kompositionswissen. Es wird von Nutzerfeedback und dem neuartigen Capability-Wissensgraphen um kontextsensitive Nutzungsinformationen und um semantische Zusammenhänge ergänzt. Erfassungsmechanismen zur Instanziierung der jeweiligen Modelle sind konzipiert. Diese wurden für den Capability-Wissensgraphen bereits erläutert, während die Ansätze zum Erfassen von Pattern-Instanzen und Nutzerfeedback in Kapitel 6 behandelt werden.

Gegenstand des nachfolgenden Kapitels 6 ist das konzipierte Empfehlungssystem und Konzepte zum Erfassen und zum Modellieren von Nutzerfeedback. Damit schließt die Beschreibung der zwei unteren Ebenen aus Abbildung 4.1, bevor Kapitel 7 neuartige Assistenzmechanismen zur Unterstützung von Nicht-Programmierern bei dem Entwickeln und dem Verstehen von CWA behandelt.

6
Empfehlungssystem

Ein zentraler Bestandteil einer Kompositionsplattform für Live-Sophistication ist ein Empfehlungssystem, das Nicht-Programmierer durchgehend bei dem Erstellen von CWA unterstützt. Zwar existieren zahlreiche konzeptionelle Ansätze. Die Analyse in Kapitel 3.2 verdeutlicht jedoch substanzielle Defizite. Gegenstand dieses Kapitels sind deshalb die neuen Ansätze des konzipierten Empfehlungssystems [Rad+12; RM17a].

Zur durchgängigen Bereitstellung von Empfehlungen ist es notwendig, unter vielfältigen Umständen variierende Arten von Empfehlungen zu bestimmen und geeignet anzuzeigen. Die daraus resultierende Menge an **Empfehlungsszenarien** muss systematisiert und vom Empfehlungssystem unterstützt werden. Zur Illustration seien als beispielhafte Empfehlungsszenarien genannt:

- explizite, facettierte Suche nach CWA durch einen Nicht-Programmierer
- proaktives Vorschlagen von Vervollständigungen der aktuellen CWA
- Vorschlagen von Alternativkomponenten sobald ein Service nicht erreichbar ist

Wie zu erkennen ist, zeichnen sich die skizzierten Empfehlungsszenarien durch verschiedene Ausprägungen in Dimensionen wie der Art des Auslösens der Empfehlung, den vorzuschlagenden Artefakten und dem Zweck von Empfehlungen aus. Das Grundgerüst notwendiger Aktivitäten beziehungsweise Phasen eines Empfehlungssystem zur Erbringung solcher Szenarien ist generell, konstant und wird konzeptionell durch den **Empfehlungskreislauf** charakterisiert.

① ***Empfehlungsgründe identifizieren*:** Zuerst ist festzustellen, wann der Endnutzer Hilfestellungen benötigt oder benötigen könnte. Dabei ist es grundsätzlich möglich, dass der Nutzer explizit nachfragt, das heißt, selbst aktiv wird. Zudem können implizit Probleme festgestellt werden, zum Beispiel indem ein MRE das Kompositionsmodell, das Kontextmodell oder das Nutzerverhalten überwacht.

② ***Empfehlungen berechnen*:** Wurden Bedingungen festgestellt, die darauf hindeuten, dass Empfehlungen bestimmt werden sollten, werden angemessene Artefakte, wie Komponenten, CWA, Kompositionsschritte und Anfragevorschläge, unter Beachtung von Nutzer- und Nutzungskontext sowie Kompositionswissen berechnet.

③ ***Empfehlungen darstellen*:** Anschließend werden die berechneten Empfehlungen visuell aufbereitet und Nutzern in geeigneter Weise dargestellt. Dies geschieht in verständlicher, nicht-technischer Weise, indem Nicht-Programmierern kommuniziert wird, welche Funktionalität die Empfehlung bereitstellt.

④ ***Empfehlungen integrieren*:** Akzeptiert ein Nutzer eine Empfehlung, wird diese automatisch integriert. Die Semantik von »Integrieren« hängt dabei von der jeweiligen Plattform und der Art des empfohlenen Items ab. In der Live-Sophistication bedeutet dies, dass ein Kompositionsfragment in eine vorhandene oder eine neue CWA eingewoben wird. Solche Änderungen des Kompositionsmodells können in Schritt ① zu Empfehlungsgründen führen, sodass sich der Kreis schließt.

Insgesamt weist der Prozess somit Ähnlichkeiten zu einer Adaptionsschleife auf: Separiert von der eigentlichen Anwendung wird kontinuierlich überwacht ①, analysiert, geplant ② – ③ und adaptiert ④. Implikationen des Empfehlungskreislaufs auf die Architektur und die Abläufe einer Kompositionsplattform für das EUD von CWA durch Nicht-Programmierer sind Gegenstand von Kapitel 6.3. Die Aktivitäten des Empfehlungskreislaufs sind generisch und gelten für alle Empfehlungsszenarien. Die spezifische Ausprägung der jeweiligen Aktivitäten hängt typischerweise vom Empfehlungsszenario ab. Daher variiert die erforderliche Herangehensweise des Empfehlungssystems hinsichtlich zu nutzender Algorithmen und einzubeziehender Kriterien sowie deren Gewichtung entsprechend der Ausprägungen. **Empfehlungsstrategien** spezifizieren eine konkrete Konfiguration des generischen Rahmenwerks, das der Empfehlungskreislauf vorgibt, indem die jeweiligen Aktivitäten zweckmäßig konfiguriert werden. Empfehlungsstrategien beschreiben wann, welche Art von Items empfohlen werden, wie die Berechnung erfolgt und in welcher Weise Empfehlungen visualisiert werden. Welche Strategien für eine Kompositionsplattform relevant sind, hängt von deren intendierten Anwendungsfällen, Anwendungsdomänen und Nutzerzielgruppen ab. Das vorgestellte Empfehlungssystem ermöglicht die Anpassung an solche kontextspezifischen Bedürfnisse.

6.1 Gesamtansatz im Überblick

Dieser Abschnitt thematisiert die architektonische Einbettung von Empfehlungskreislauf und Empfehlungsstrategien innerhalb einer Plattform für assistiertes EUD von CWA.

Abbildung 6.1 illustriert die Module eines MRE, die dem Empfehlungssystem zuzuordnen sind. *Trigger* sind für ① *Empfehlungsgründe identifizieren* zuständig, indem Events aus verschiedenen Quellen empfangen und Bedingungen ausgewertet werden. Sie stützen sich dabei auf Informationsquellen, wie das Kompositionsmodell, das Nutzerprofil und Ontologien, die zur semantischen Annotation von Komponenten genutzt werden. Wurden Bedingungen festgestellt, die darauf hindeuten, dass Empfehlungen bestimmt werden sollten, wird die Empfehlungsberechnung ② initiiert. Hierfür ist ein *Empfehlungsmanager* vorgesehen, der Empfehlungsmethoden bereitstellt. Er interpretiert die Ausgabedatenstruktur von Triggern und führt die zugehörigen Empfehlungsmethoden durch. Der grundsätzliche Ablauf umfasst die Schritte des Matching und Ranking von Kompositionsfragmenten und wird je nach konkreter Empfehlungsstrategie parametrisiert. Diverse Datenbestände aus der Modellebene, vergleiche Kapitel 6.2 sowie 5.1, werden einbezogen, insbesondere Pattern-Instanzen und Domänenwissen. Zudem fließt

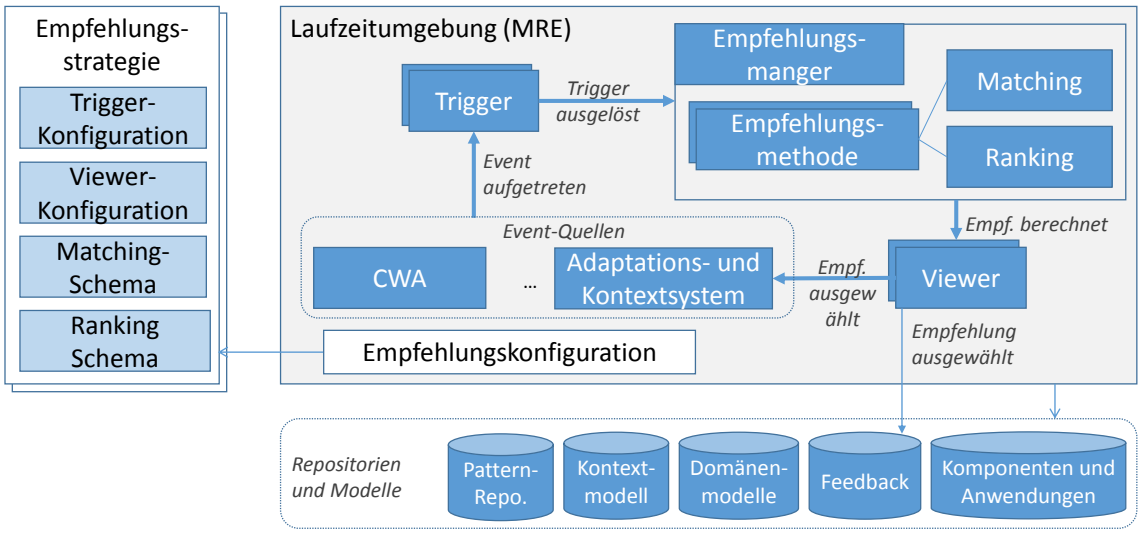


Abbildung 6.1: Überblick der architektonischen Einordnung des Empfehlungskreislaufs

das Wissen aus der vorherigen Phase meist mit ein, da Lösungen für die identifizierten Empfehlungsgründe gesucht werden. Anschließend stellen *Recommendation-Viewer* dem Nutzer die berechneten Empfehlungen in geeigneter Weise dar und erlauben es, diese auszuwählen. Weiterhin sammeln sie implizites Feedback in Form von Auswahlstatistiken. Einem MRE obliegt es nach Selektion einer Empfehlung durch den Nutzer, implizites Feedback in das Nutzerprofil zu übertragen, sodass es in späteren Empfehlungsberechnungen die Ergebnisse verbessern kann. Weiterhin gilt es, das hinter der Empfehlung stehende Kompositionsfragment in die aktuelle CWA zu integrieren. Hierzu werden notwendige Adaptionsschritte bestimmt und das *Adaptionssystem* mit deren transaktionsorientierter Ausführung beauftragt.

Das konzipierte Empfehlungssystem basiert auf **Empfehlungsstrategien**. Diese beschreiben konkret unter welchen Umständen (*Wann*), welche Art von Artefakten (*Was*) vorgeschlagen werden sollen, wie die Berechnung vorgenommen wird und auf welche Weise die Ergebnisse visualisiert werden (*Wie*). Dieses Konzept ist generisch auf Empfehlungssysteme anwendbar. Bezugnehmend auf den vorgestellten Empfehlungskreislauf und die Architekturbestandteile wird eine Empfehlungsstrategie wie folgt charakterisiert:

- Optionale *Bedingungen* drücken aus, in welchen Kontexten eine Strategie einsetzbar ist. Dabei spielen bspw. Präferenzen und Fähigkeiten des Nutzers eine Rolle.
- Zu nutzende *Trigger* zur Initiierung des Empfehlungskreislaufs werden festgelegt.
- Eine Teilmenge der verfügbaren *Empfehlungsmethoden*, die zu verwenden sind, wird angegeben. Empfehlungsmethoden behandeln *was* vorzuschlagen ist und *wie* die Berechnung vonstattengeht. Sie spiegeln daher den Zweck von Empfehlungen in einem Szenario wider. Sie sind Vorgehensweisen des Empfehlungsmanagers und gekennzeichnet durch benötigte Eingabeparameter, ein *Matching-Schema* und ein *Ranking-Schema*, die in Abschnitt 6.3.3 vertieft vorgestellt werden.
- Zu nutzende *Recommendation-Viewer*, die den Nutzern Empfehlungen zugänglich machen und verständlich visualisieren, werden festgelegt und optional konfiguriert.

- Die *Konfiguration der Integration* von Kompositionsfragmenten legt beispielsweise fest, wie vorzugehen ist, falls eine Komponente bereits im Mashup vorkommt.

Empfehlungsstrategien definieren somit eine konkrete Ausprägung des generellen Ablaufs des Empfehlungskreislaufs, indem die Schritte gemäß dem angedachten Zweck der Empfehlungen konfiguriert werden. Empfehlungsstrategien weisen somit eine Ähnlichkeit zu *Web-API-Search-Patterns* [BDM13c] auf, sind jedoch weiter gefasst und generischer. Letztere entsprechen zwar Empfehlungsmethoden, die Belange *wann* und *wie* in Bezug auf Visualisierungsaspekte lassen sie jedoch missen. Empfehlungsstrategien erlauben die bedarfsgerechte Konfiguration des gesamten Empfehlungskreislaufs je nach Empfehlungsszenario.

Eine *Empfehlungskonfiguration* deklariert die in einem MRE zu nutzenden Empfehlungsstrategien. Während der Initialisierung wertet der Empfehlungsmanager diese Konfiguration aus und validiert die Bedingungen in Relation zum aktuellen Kontext, um die Menge der *aktiven Empfehlungsstrategien* zu ermitteln. Anschließend werden die benötigten Trigger dynamisch eingebunden. Ein derartiger Plug-In-Mechanismus kann konzeptionell ebenfalls auf Empfehlungsmethoden und Recommendation-Viewer angewendet werden. Allerdings wird dies nicht vertiefend behandelt, sondern vorgesehen, dass vordefinierte Empfehlungsmethoden und Recommendation-Viewern existieren, die strategiespezifisch einbezogen und konfiguriert werden. Trigger deklarieren welche Empfehlungsmethoden sie ansprechen können. Sowohl Matching- als auch Ranking-Schema von Empfehlungsmethoden können durch Empfehlungsstrategien konfiguriert werden.

Welche Empfehlungsstrategien für eine Kompositionsplattform für CWA von Belang sind, wird maßgeblich von den anvisierten Anwendungsdomänen und Nutzergruppen bestimmt. Daneben spielen individuelle Vorlieben und Fähigkeiten von Nutzern eine Rolle. Dazu werden im Nutzermodell nicht gewünschte Empfehlungsstrategien hinterlegt.

6.2 Empfehlungssystemspezifische Metamodelle

Mehrere Modelle formen die Grundlage des vorgeschlagenen Empfehlungssystems: Trigger als Initiatoren des Empfehlungskreislaufs und Patterns, die im Zusammenspiel mit Feedback (Abschnitt 5.1.4) und dem Capability-Wissensgraphen (Abschnitt 5.4) Kompositionswissen repräsentieren.

6.2.1 Trigger-Metamodell

Zur Erbringung von Aktivität ① *Empfehlungsgründe identifizieren* wird ein vereinheitlichtes Konzept von **Triggern** vorgeschlagen, das auf Ergebnissen des Großen Belegs von Clemens Große [Gro13] beruht. Ein Trigger spezifiziert Kontextbedingungen, die festlegen, wann ein Nutzer mit Empfehlungen zu unterstützen ist. Durch Kontextüberwachung sowie Auswertung von Bedingungen stößt ein Trigger den Prozess der Generierung von Vorschlägen an. Ein Trigger entspricht einem Tupel (k, S, B, E).

Identifikator k: Jeder Trigger besitzt einen einzigartigen Identifikator, der zum Verwalten und Referenzieren, etwa im Rahmen einer Empfehlungsstrategie, dient.

Menge von Startevents S: In S werden die Events aufgeführt und optional definiert, welche notwendig sind, um den Trigger zu aktivieren. Grundlegende Annahme ist

dabei, dass das MRE eventbasiert ist. Da das Nutzen und Entwickeln von CWA verschmelzen, werden lediglich Events zugelassen, die über ein MRE erfassbar sind. S enthält entweder Referenzen auf existierende Events eines MRE oder die Definition eines neuen *komplexen Events* unter Zuhilfenahme existierender. Eine genaue Spezifikation und eine konkrete Syntax der schematischen Mittel zur Adressierung von Events findet sich in Kapitel 6.3.2. Zum Abdecken einer hohen Bandbreite an Situationen bei der Live-Sophistication werden unterschiedliche Arten von Events unterstützt, die sich gemäß ihrer Quelle kategorisieren lassen:

MRE-Events stammen von Modulen eines MRE. Annahme hierbei ist, dass eine Menge MRE-übergreifend standardisierter Events existiert, auf die Low-Level-Events, etwa aus dem DOM, abgebildet werden. Dadurch wird eine Abhängigkeit von plattformspezifischen Implementierungen vermieden, was die Wiederverwendbarkeit von Triggern erhöht. Hierzu zählen *Lebenszyklusevents* aus der Umsetzung des Lebenszyklus von Komponenten und Anwendungen, zum Beispiel die Initialisierung einer Komponente anzeigend. *Kompositionsevents* spiegeln Änderungen am Kompositionsmodell wider. *Timer-Events* signalisieren den Ablauf zeitlicher Intervalle. *Fehlerevents* repräsentieren Fehlerereignisse innerhalb eines MRE, wie fehlgeschlagene Service-Anfragen und Fehler bei der Integration von Komponenten. *UI-Events* indizieren Ereignisse im UI eines MRE, wie die Eingabe in ein Suchfeld.

Kontextevents werden vom Adaptationssystem propagiert, falls sich relevante Parameter im Kontextmodell verändert haben, zum Beispiel der Gerätezustand.

Anwendungsevents werden von Komponenten einer CWA gemäß ihrer SMCDL über die im MCM definierten Kommunikationskanäle veröffentlicht.

Menge von Bedingungen B: Ein aktivierter Trigger prüft optional weitere Bedingungen. Diese entsprechen Wenn-Dann-Regeln und beziehen Informationen aus dem Nutzdatenbereich von Events, aus Modellen und aus Webservices ein. Sind sämtliche Bedingungen erfüllt, löst ein Trigger aus.

Empfehlungsmethoden E: Wenn ein Trigger auslöst, zieht er eine Reihe von Empfehlungsmethoden nach sich und konfiguriert diese situationsspezifisch.

Konzeptionell lassen sich Trigger hinsichtlich ihres Verhaltens und der Intention des Nutzers, Empfehlungen zu erhalten, unterscheiden. Bei **expliziten Triggern** wird das Auslösen der Empfehlungsberechnung bewusst durch den Nutzer veranlasst, sodass UI-Events vorausgehen. Komponentenspezifische UI-Interaktionen werden hierbei bewusst ausgeklammert, da nicht vorgesehen ist, dass über Interaktionselemente in Komponenten expliziter Zugriff auf Plattformfunktionalitäten erfolgen kann. Dennoch können komponentenspezifische Events den Empfehlungsprozess auslösen, jedoch über die nachfolgenden Trigger-Arten. **Implizite Trigger** decken Situationen ab, in denen das Auslösen der Empfehlungsberechnung nicht bewusst durch den Nutzer forciert wird.

- **Reaktive Trigger**: Das Auslösen erfolgt ohne explizite Veranlassung des Nutzers als Reaktion auf eine Zustandsänderung, die entweder von außerhalb an das MRE kommuniziert wird, zum Beispiel vom Kontextdienst, oder direkt dem MRE entspringt, zum Beispiel nach Änderungen der Komposition. Die Zustandsänderung ist dabei gegebenenfalls direkter Grund für die Empfehlungsberechnung.

- **Proaktive Trigger**: Das Auslösen erfolgt weder durch Veranlassen des Nutzers noch als unmittelbare Reaktion auf Zustandsänderungen. Das System beginnt die Empfehlungsberechnung durch interne Regeln, wie zyklischer Überprüfungen bestimmter Bedingungen. Diese Bedingungen betreffen typischerweise die aktuelle Komposition. Zwar geht im Regelfall ein MRE-Event voraus, das zum Beispiel die Erweiterung der Komposition anzeigt. Es stellt jedoch nicht den unmittelbaren Grund für die Empfehlungsberechnung dar, sondern dieser ergibt sich erst durch weitere Startevents, wie Timer-Events, sowie die Bedingungsprüfung des Triggers, zum Beispiel hinsichtlich des Vorhandenseins von Kommunikationskanälen.

Die Unterscheidung in *implizit* und *explizit* spiegelt die Intention des Nutzers bezüglich des Erhaltens von Vorschlägen wider. Die Einteilung in *reaktiv* und *proaktiv* reflektiert Unterschiede im Systemverhalten.

Der grundlegende Ansatz der Trigger bezieht Ergebnisse aus der Arbeit von Chowdhury et al. [RDC14] ein. Allerdings ist das hier vorgestellte Konzept generischer, da es nicht auf Kompositionsaktivitäten von Nutzern beschränkt ist, sondern eine Reaktion auf allgemeine Kontextänderungen sowie proaktives Systemverhalten unterstützt. Dies lässt sich anhand der nachfolgenden Beispiele verdeutlichen.

- Wenn ein Nutzer mit der Capability-View interagiert, um eine Komponenten mit anderen zu verknüpfen, wird die Berechnung passender Kommunikationskanäle für die selektierte Komponente in Auftrag gegeben (*explizit*).
- Alle 30 s oder nach Änderungen des Kompositionsmodells und falls bereits mindestens zwei Komponenten in der aktuellen Anwendung integriert sind, wird die Berechnung von Komponenten und Kommunikationskanälen induziert, die die CWA hinsichtlich der aktuellen Nutzeraufgabe sinnvoll ergänzen (*proaktiv*).
- Falls eine Komponente ein Event publiziert, das noch über keinen Kommunikationskanal an andere Komponenten transferiert wird, wird die Empfehlung passender Verknüpfungen zu vorzugsweise bereits im Mashup enthaltenen Komponenten angestoßen (*reaktiv*).
- Falls Anfragen einer Komponente an einen Webservice mehrfach fehlschlagen oder falls der Nutzer in einer Qualitätsanforderung »schnelle Antwortzeiten« fordert und der Webservice länger als zwei Sekunden zur Antwort benötigt, wird die Bestimmung alternativer Komponenten in Auftrag gegeben (*reaktiv*).

6.2.2 Pattern-Metamodell

Das konzipierte **Pattern-Metamodell** [Rad+12] gibt die Struktur und die Semantik von Pattern-Instanzen vor. Es basiert auf dem Ansatz von Chowdhury [Roy+10] und wurde gemäß den Anforderungen des assistierten Mashup-EUD erweitert, beispielsweise um funktionale und qualitative Aspekte. Patterns repräsentieren Kompositionswissen und kapseln dazu Wissen über strukturelle oder logische Beziehungen zwischen Komponenten. Folgende Patternklassen sind vorgesehen und generisch insofern, als im Wesentlichen Konstrukte des MCM zur Einteilung dienen.

- **Co-Occurrence** beschreibt das statistische häufige Auftreten zweier Komponenten zusammen in einer Mashup-Anwendung.

- **Coupling** repräsentiert einen Kommunikationskanal inklusive Mapping-Definition, vergleiche Kapitel 5.2, zwischen Schnittstellenelementen zweier Komponenten.
- **Occlusion, Exchangeability** drückt Kompositionswissen über eine semantische, gerichtete Relation zweier Komponenten aus. Während austauschbare Komponenten gleichwertige Funktionalität erbringen, tritt bei Überdeckung zusätzliche Funktionalität auf. Neben den beiden an der Relation beteiligten Komponenten wird am Pattern eine Mapping-Definition hinterlegt, die beschreibt, wie Komponentenschnittstellen aufeinander abzubilden sind, zum Beispiel durch Umbenennung von Operationen oder durch semantische Datenmediation von Parametertypen.
- **Complex** modelliert Kompositionswissen, das durch ein umfangreicheres Kompositionsfragment bestehend aus Komponenten, Kommunikationskanälen und Anwendungssichten inklusive Layouts beschrieben ist.

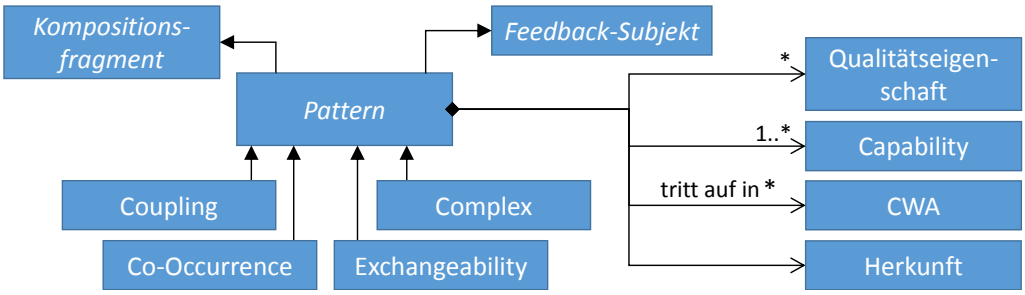


Abbildung 6.2: Schematische Darstellung des Pattern-Metamodells

Von diesen Pattern-Klassen können Instanzen gebildet werden. Domänenspezifische Pattern-Instanzen lassen sich abbilden, da der Domänenbezug über die semantischen Annotationen von Komponenten und abgeleitete Annotationen ganzer Teilkompositionen gegeben ist. Alle Pattern-Klassen sind wie folgt charakterisiert, siehe dazu Abbildung 6.2.

- *Kompositionsfragment:* Jede Pattern-Instanz ist ein Kompositionsfragment, wie die Vererbung zwischen *Kompositionsfragment* und *Pattern* zeigt. Je nach Pattern-Klasse umfasst eine Pattern-Instanz somit einzelne Komponenten, wie bei Exchangebility, Kanäle inklusive den verknüpften Komponenten im Fall von Coupling bis hin zu beliebigen Ausschnitten von Kompositionsmodellen bei Complex.
- *Community-Feedback:* Patterns können Gegenstand von Nutzerfeedback sein, weshalb sie von *Feedback-Subjekt* erben. Dies gilt sowohl für explizite Bewertung als auch implizite Nutzungsstatistiken und leistet dadurch einen wesentlichen Beitrag zur Validierung von Pattern-Instanzen. Details zum Modell und zur Erfassung von Feedback enthält Kapitel 6.3.4.
- Die *Herkunft* informiert darüber, woher eine Pattern-Instanz stammt. Prinzipiell sind Pattern-Instanzen unabhängig von der Art ihrer Detektion, das heißt, sie können auf mehrerlei Weise abgeleitet werden. Die Herkunft des Kompositionswissens wird vorgehalten, um dies dem Nutzer bei der Visualisierung zu kommunizieren

und somit Vertrauen herzustellen. Dazu wird eine Kennung des genutzten Algorithmus hinterlegt, zusammen mit einem optionalen *Confidence*-Wert. Details zur Mustererkennung beinhaltet Abschnitt 6.3.1. Im Fall von Pattern-Instanzen, die aus vorhandenen Mashups extrahiert wurden, werden die Anwendungsmodelle referenziert, in denen das Muster entdeckt wurde. Schließlich beschreibt die Assoziation *tritt auf in* in welchen Mashups eine Pattern-Instanz vorkommt.

- Die *Funktionalität* einer Pattern-Instanz wird als eine Menge von Capabilities aufgefasst. In Abbildung 6.2 ist dies durch die Relation zu *Capability* modelliert. Die Capabilities bezeichnen die domänenspezifische Funktionalität, die eine Pattern-Instanz in einer CWA erbringen kann. Sie gehen zum Beispiel aus der algorithmischen Bestimmung gemäß dem in Kapitel 5.3 vorgestellten Konzept hervor.

- Analog zu Komponenten und Mashups besitzen Pattern-Instanzen nicht-funktionale Eigenschaften, was durch die Assoziation zu *Qualitätseigenschaft* notiert ist. Diese ergeben sich je nach Pattern-Klasse durch Verrechnen der Qualitätseigenschaften involvierter Komponenten. Hierbei wird auf vorhandene Forschungsansätze aufgesetzt [Rüm+13; Rüm+14; Tsc14].

Das vorgestellte Pattern-Metamodell erlaubt das Beschreiben von Kompositionswissen, wobei die Belange Erfassung und Repräsentation separiert sind. Erkenntnisse aus existierenden Arbeiten wurden aufgegriffen und erweitert, zum Beispiel um semantische Annotationen sowie Qualitätseigenschaften. Zudem erlaubt es das hier vorgestellte Konzept, im Vergleich zu existierenden Ansätzen, Anwendungssichten und Layouts durch Complex-Pattern zu unterstützen. Pattern-Instanzen spielen eine zentrale Rolle für das Empfehlungssystem, was im nachfolgenden Kapitel 6.3 vertieft wird, und für Werkzeuge zum assistierten EUD von CWA, die Kapitel 7 behandelt.

6.3 Architektur und Abläufe des Empfehlungssystems

In diesem Kapitel wird diskutiert, wie die in Abschnitt 6.1 genannten Phasen des Empfehlungskreislaufs auf die Architektur einer Mashup-Plattform abgebildet werden und wie die resultierenden Architekturbestandteile miteinander in Wechselwirkung stehen.

Wie in Abbildung 6.3 zu sehen ist, sind Trigger als Softwaremodule eines MRE der Ausgangspunkt, indem sie Events aus verschiedenen Quellen entgegennehmen und ① *Empfehlungsgründe identifizieren*. Falls ein Trigger auslöst, übermittelt er eine *Triggerausgabe* an den *Empfehlungsmanager*. Der Empfehlungsmanager verwaltet die ankommenden Daten und verarbeitet diese. Im ersten Schritt identifiziert er anhand der Triggerausgabe die zugehörige *Empfehlungsstrategie*. Danach wird ein *Recommendation-Job* pro Triggerausgabe instantiiert. Dabei handelt es sich um eine Datenstruktur zur Schaffung eines Kontextes, der für die strukturierte Abarbeitung der nachfolgenden Phasen des Empfehlungskreislaufs benötigt wird. Die Umsetzung der Phasen realisiert der Empfehlungsmanager unter Zuhilfenahme verschiedener weiterer Architekturkomponenten, welche nachfolgend detailliert werden. Im Zuge dessen dient der *Recommendation-Job* zur Haltung von Daten und zum Austausch von Zustandsinformationen. Er wird mit der Triggerausgabe befüllt. Strategiespezifische Konfigurationsparameter ergänzen und überschreiben gegebenenfalls diese Daten. Für nicht definierte Parameter gelten Standardwerte. Anschließend wird der Recommendation-Job in eine Warteschlange eingefügt.

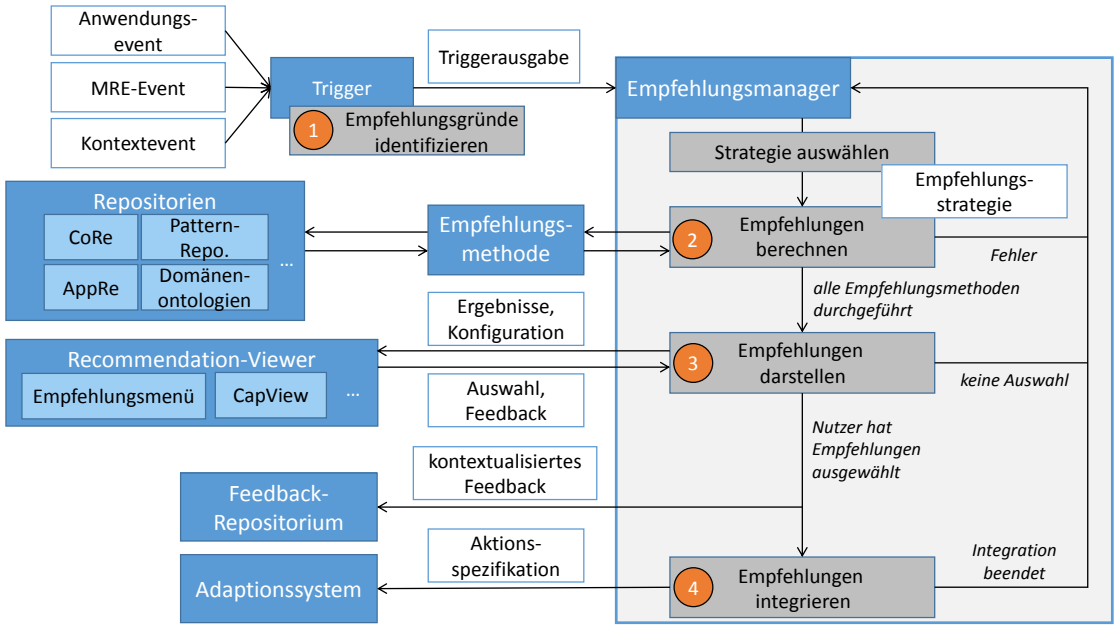


Abbildung 6.3: Architektur und Grundablauf des Empfehlungssystems

Das Vorhandensein »gleicher« Einträge wird unter Zuhilfenahme einer Äquivalenzrelation für Triggerausgaben geprüft, und gemäß der Konfiguration gehandhabt. Anschließend erfolgt die Sortierung der Warteschlange anhand der Priorität der jeweiligen Triggerklasse, siehe Abschnitt 6.3.2.

Zur *Berechnung von Empfehlungen* ② nutzt der Empfehlungsmanager die jeweiligen Empfehlungsmethoden, welche wiederum auf verschiedene Repositorien zugreifen, insbesondere das *Pattern-Repositorium*, vergleiche Abschnitt 6.3.1. Diese Aspekte werden in Abschnitt 6.3.3 vertieft. Welche Empfehlungsmethoden für den aktuellen Recommendation-Job relevant sind, folgert der Empfehlungsmanager anhand der Empfehlungsstrategie und zugehöriger Trigger. Empfehlungsmethoden sind dafür zuständig, Anfragen an Repositorien aus den Daten im Recommendation-Job zu formulieren, und transformieren die Ergebnisse in ein normiertes Format. Die resultierenden Empfehlungen pro Methode und möglicherweise aufgetretene Fehler legt der Empfehlungsmanager im Recommendation-Job ab. Je nach Zustandsinformation im Recommendation-Job wird der aktuelle Empfehlungsprozess beendet oder wechselt in die nächste Phase.

Zur Erbringung der Phase ③ *Empfehlungen darstellen* stützt sich der Empfehlungsmanager auf konfigurierbare *Recommendation-Viewer* der Laufzeitumgebung, vergleiche Abschnitt 6.3.4. Er identifiziert anhand der aktuellen Empfehlungsstrategie die passenden Recommendation-Viewer und übergibt diesen anschließend die Ergebnisse der Empfehlungsmethoden sowie optional Konfigurationsparameter aus dem Recommendation-Job. Recommendation-Viewer erlauben die Auswahl von Empfehlungen sowie das Aufzeichnen von Nutzerverhalten, um implizites Feedback zur Güte der Empfehlungen zu erheben. Zudem melden sie dem Empfehlungsmanager das Beenden der Darstellung, sei es durch Selektion oder Verwerfen von Empfehlungen. Implizites Feedback, das die Recommendation-Viewer übermitteln, zum Beispiel die getroffene Auswahl, wird vom Empfehlungsmanager gesammelt und an das *Feedback-Repositorium* übergeben. Anschließend wechselt der Empfehlungsmanager in die Phase ④ *Empfehlungen integrieren* unter Zuhilfenahme des

Adaptionssystems, was Abschnitt 6.3.5 behandelt.

Spezifische Aspekte dieses generellen Ablaufs sowie deren architektonische Auswirkungen werden in den folgenden Kapiteln im Detail erläutert.

6.3.1 Ableitung von Pattern-Instanzen

Wie in Kapitel 6.2.2 beschrieben, bilden Pattern-Instanzen die Datengrundlage für die Empfehlung von Kompositionsschritten. Sie repräsentieren Kompositionswissen, das möglichst bewährte Lösungen für bestimmte Problemstellungen und Situationen kapselt.

Pattern-Instanzen können auf verschiedene Weisen abgeleitet werden. Einerseits besteht die Möglichkeit das Wissen, welches in semantischen Annotationen von Komponenten steckt, auszunutzen und zum Beispiel die Austauschbarkeit oder Verknüpfbarkeit von Komponenten zu bestimmen. Andererseits spielen Techniken zur statistischen Auswertung und zum Datamining vorhandener Kompositionsmodelle sowie zum maschinellen Lernen eine Rolle. Dabei ist typischerweise die Semantik weniger relevant als der syntaktische Aufbau der Modelle. Wie in Kapitel 1.2.3 angedeutet, wird davon ausgegangen, existierende Ansätze wie [Yan+12; Rod+14; Maa+11] zu adaptieren und einzubinden. Semantikbasierte Techniken eignen sich, um ein Basiswissen zu erzeugen ohne auf die Existenz von Kompositionsmodellen angewiesen zu sein. Eine Validierungsinstanz ist jedoch notwendig, da das gewonnene Wissen eher Vermutungen entspricht und daher nicht zwangsläufig »nützlich« sein muss, weil es stark von der Qualität semantischer Annotationen abhängt. Community-basierte Techniken weisen hingegen Kaltstartprobleme auf. Es kann jedoch bei entsprechender Quantität davon ausgegangen werden, dass nützliche und funktionierende Pattern-Instanzen zutage treten. Weiterhin ist die Möglichkeit zu nennen, dass Anbieter oder Betreiber einer Kompositionsplattform passend zur anvisierten Anwendungsdomäne bereits eine Menge von Pattern-Instanzen bereitstellen.

Im Rahmen dieser Arbeit wird vorgesehen, dass Pattern-Instanzen der Klassen Coupling und Exchangeability initial semantisch bestimmt werden. Zusätzlich existieren Mechanismen zum Auswerten von Kompositionsmodellen, wodurch insbesondere Instanzen der Pattern-Klassen Complex und Co-Occurrence abgeleitet werden. Als Validierungsinstanz treten die Nutzer auf, indem *Feedback* zu empfohlenen Pattern-Instanzen implizit und explizit gesammelt wird. Auf Basis des Einbeziehens des Feedbacks in das Ranking wird erreicht, dass sich korrekte Pattern-Instanzen mit der Zeit herauskristallisieren.

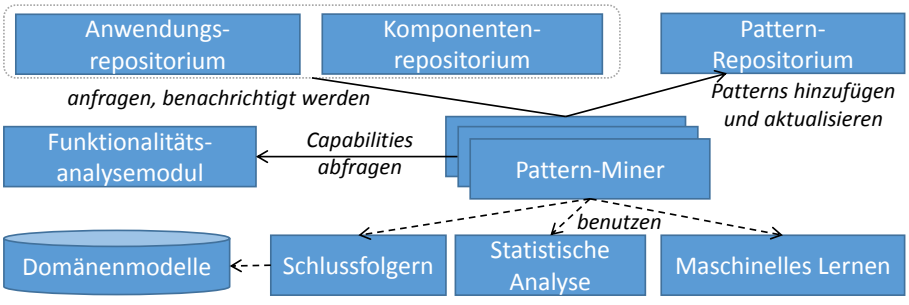


Abbildung 6.4: An der Ableitung von Pattern-Instanzen beteiligte Architekturkomponenten

Architektonisch sind *Pattern-Miner* für die Ableitung von Pattern-Instanzen zuständig, siehe Abbildung 6.4. Sie analysieren die entsprechenden Repositorien aktiv oder bei Bedarf,

beispielsweise nach Änderung der Datenbasis, und bestimmen unter Zuhilfenahme oben angedeuteter Algorithmen neue Pattern-Instanzen. Im Rahmen dessen stützen sie sich auf Funktionalitäten weiterer Module. Insbesondere wird das *Funktionalitätsanalysemodul* mit dem automatischen Ableiten der Capabilities von Pattern-Instanzen beauftragt, siehe dazu Kapitel 5.3. Zusätzliche Abhängigkeiten von Modulen zum semantischen Schlussfolgern, zur statistischen Analyse und zum maschinellen Lernen können bestehen. Schließlich veranlassen Pattern-Miner die persistente Speicherung von Pattern-Instanzen im Pattern-Repositorium, sodass diese daraufhin dem Empfehlungsmanager zur Verfügung stehen. Dazu stellt das Pattern-Repositorium eine Reihe generischer Schnittstellen bereit, zum Beispiel zur Abfrage aller Pattern-Instanzen einer bestimmten Klasse. Darüber hinaus obliegt es dem Pattern-Repositorium, die Konsistenz und Aktualität der Pattern-Instanzen sicherzustellen. Obgleich sich entsprechende Mechanismen nicht im Fokus der vorliegenden Arbeit befinden, existieren konzeptionelle Ansatzpunkte. Zunächst werden sowohl das Pattern-Repositorium als auch die Pattern-Miner über Änderungen im Komponenten- und im Anwendungsrepositorium in Kenntnis gesetzt. Dadurch wird das Pattern-Repositorium in die Lage versetzt, auf das Entfernen von Komponentenbeschreibungen und Kompositionsmodellen adäquat zu reagieren. Sinnvolle Maßnahmen umfassen zum Beispiel das Löschen von Pattern-Instanzen, die solche Komponenten beinhalten, und das Neuberechnen der Signifikanz von statistisch aus Kompositionsmodellen gewonnenen Pattern-Instanzen. Pattern-Miner können durch die Benachrichtigungen veranlasst werden, aktualisierte Artefakte nochmals zu analysieren und gegebenenfalls veraltete Pattern-Instanzen zu ersetzen.

6.3.2 Empfehlungsgründe identifizieren durch Trigger

Trigger können bedarfsgerecht in eine Laufzeitumgebung integriert werden. In diesem Abschnitt werden die dazu nötigen Konzepte detailliert, insbesondere der Lebenszyklus, die Beschreibung und die architektonische Einordnung von Triggern.

Lebenszyklus von Triggern

Jeder Trigger durchläuft einen definierten Lebenszyklus, dessen Realisierung dem MRE obliegt. In Abbildung 6.5 sind die Zustände und – falls vorhanden – die Aktivitäten, die zu Transitionen führen, dargestellt.

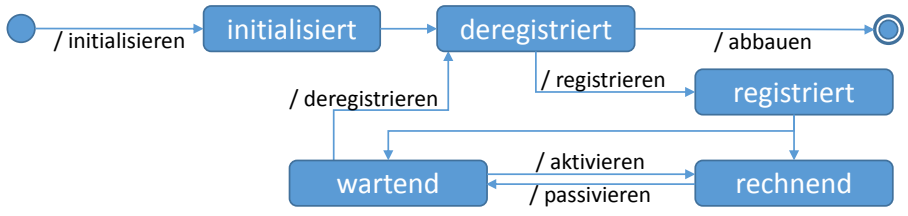


Abbildung 6.5: Lebenszyklus von Triggern

Zuerst erfolgt das **Initialisieren** des Triggers. Er wird instantiiert und über vordefinierte Schnittstellen vom MRE konfiguriert. Anschließend befindet sich der Trigger im Zustand *initialisiert*. Er ist jedoch noch *deregistriert*, sodass er keinerlei Events empfangen kann. Im Zuge der Aktivität **Registrieren** wird der Trigger mit seinen Eingabequellen gekoppelt,

indem er an den entsprechenden Kanälen am Event-Bus als Empfänger registriert wird. Die Registrierung auf Anwendungsevents kann verzögert geschehen. In einem solchen Fall verharrt der Trigger im Zustand *deregistriert* und wird mit den passenden Kommunikationskanälen verbunden, sobald solche in der Anwendung vorhanden sind. Daraufhin sind Trigger *registriert* und bereit, ihrer Aufgabe nachzugehen. Standardmäßig findet ein Übergang in den Zustand *wartend* statt. Eine Transition in den Zustand *rechnend* kann auf zwei Weisen erfolgen: (1) durch **Aktivieren** im Zustand *wartend* nach dem Eintreten der notwendigen Startevents und (2) durch initiale Bedingungsprüfung. In diesem Fall wechselt der Trigger direkt von *registriert* in den Zustand *rechnend*, um unabhängig von Startevents Bedingungen aus B zu prüfen, gegebenenfalls unter initialer Abfrage von Modellen. Als Beispiel seien Trigger genannt, die Werte aus dem Kontextmodell zur Auswertung von Bedingungen benötigen. Abhängig von der Granularität des Kontextparameters und dessen Änderungshäufigkeit kann es sein, dass keine Änderung vom Kontextdienst während der Anwendungsnutzung propagiert wird. In solchen Fällen ist eine Abfrage von Kontextparametern und eine initiale Bedingungsprüfung notwendig. Im Zustand *rechnend* wertet ein Trigger im Wesentlichen die Bedingungen B unter Nutzung der bereitgestellten Schnittstellen und Nutzdaten der Startevents aus. Können die Bedingungen nicht positiv evaluiert werden, bricht die Berechnung ab und es findet ein **Passivieren** statt. Anderenfalls löst der Trigger aus, baut die Ausgabedatenstruktur auf und übermittelt sie schließlich an den Empfehlungsmanager, sodass die Empfehlungsberechnung eingeleitet ist. Nach dem Auswerten der Bedingungen und gegebenenfalls nach Auslösen des Triggers geht dieser durch **Passivieren** in den Zustand *wartend*.

Falls ein Kommunikationskanal der Anwendung, auf den ein Trigger registriert ist, entfernt wird, auf Nutzerinitiative oder aus Gründen der Fehlerbehandlung, zum Beisiel im Fall des mehrfachen Auftretens von Ausnahmen im Zustand *wartend*, wird der Trigger von seinen Eingabequellen entkoppelt und somit isoliert. Im Rahmen dieser Aktivität **Deregistrieren** erfolgt ein Abmelden beim Event-Bus für sämtliche Startevents. **Abbauen** zieht nach sich, dass der Trigger endgültig aus dem aktuellen Ausführungskontext des MRE entfernt wird, sodass alle benötigten Ressourcen freigegeben werden. Dazu muss der Trigger bereits *deregistriert* sein.

Beschreibungssprache für Trigger

Zur einfachen Erweiterung eines MRE mit Triggern und zur Erhöhung der Wiederverwendbarkeit von Triggern in verschiedenen MRE existiert ein hybrides Beschreibungsformat, das Ähnlichkeiten zur Komponentenbeschreibungssprache SMCDL aufweist. Die Trigger-Beschreibung umfasst einen deklarativen Teil in XML ①, der in Listing 6.1 veranschaulicht wird, und einen imperativen Teil ② in Form einer Implementierung. Teil ① beschreibt die Startevents (siehe Zeilen 2–3 in Listing 6.1), die Empfehlungsmethoden (Zeilen 4–6) und die Kennzeichnung des Triggers (Zeile 1). Außerdem wird der Implementierungsteil ② referenziert (benötigte Quellcodedateien, Zeilen 7–11) und über den Konstruktor angebunden (Zeilen 12–14).

```
1 <tr:trigger class="..." id="..." ...>
2   <tr:startevents>...
3   </tr:startevents>
4   <tr:recommendationtypes>
5     <tr:recommendationtype>getPossibleComplexPatterns</tr:recommendationtype>
6   </tr:recommendationtypes>
7   <smcdl:dependencies>
8     <smcdl:dependency>
```

```
9         <smcdl:url>http://tempuri.org</smcdl:url>
10      </smcdl:dependency>
11    </smcdl:dependencies>
12    <smcdl:constructor>
13      <smcdl:code>new org.example.ExampleTrigger()</smcdl:code>
14    </smcdl:constructor>
15  </tr:trigger>
```

Listing 6.1: Beispiel des XML-Teils einer Trigger-Beschreibung

Zur Deklaration der Startevents können vorhandene Events referenziert werden via Tupel aus Kanal und Event (siehe Zeile 1 in Listing 6.2). Im Fall von Anwendungsevents ist die Definition einer Signatur vorgesehen, die ein passendes Event aufweisen muss, um für den Trigger von Belang zu sein, vergleiche Zeilen 8–13. Grundsätzlich ist es zulässig, mehrere Events anzugeben, die dann in keiner zeitlichen oder strukturellen Beziehung stehen. Stattdessen führt das Eintreten jedes dieser Events zur Aktivierung des Triggers.

```
1  <tr:startevent xsi:type="tr:SimpleEvent" name="componentRemoved" channel="
       componentLCChannel" />
2
3  <tr:startevent xsi:type="tr:ComplexEvent" operator="Seq">
4    <tr:event xsi:type="..." >...</tr:event>
5    <tr:event xsi:type="..." >...</tr:event>
6  </tr:startevent>
7
8  <tr:startevent xsi:type="tr:ApplicationEvent" name="AppEventWithLocationParam">
9    <tr:correlation unit="ms" value="500" action="useLast"/>
10   <tr:signature>
11     <tr:parametertype>geo:Location</tr:parametertype>
12   </tr:signature>
13 </tr:startevent>
```

Listing 6.2: Beispiele für verschiedene Möglichkeiten zur Definition von Startevents

Neue komplexe Events können ebenfalls definiert werden (siehe Zeilen 3–6). Dabei kommen zeitliche und logische Verknüpfungen von atomaren oder wiederum komplexen Events zum Einsatz. Weiterhin ist es möglich, *Event-Correlation* [Fül+10] zu konfigurieren, sodass sehr häufig auftretende Events über einen bestimmten Zeitraum gefiltert oder zusammengefasst werden und nur ein korreliertes Event an den Trigger geschickt wird, siehe Zeile 9 in Listing 6.2. Wie dabei die zusammenzufassenden Events behandeln werden, gibt die `action` an. Der Standardfall ist das Nutzen des letzten Events.

Der imperative Anteil ② in Form einer Implementierung wird in ① referenziert, siehe Zeilen 7–11 in Listing 6.1. Er deckt die Bedingungen B sowie die Umsetzung des Zustands *rechnend* ab. Hierzu zählt das Erstellen und Veröffentlichen der Ausgabedatenstruktur, welche in Kapitel 6.3.2 behandelt wird. Somit ist gewährleistet, dass pro Auslösen spezifische Gründe und Konfigurationsparameter an den Empfehlungsmanager übertragen werden können. Beispielsweise kann ein proaktiver Trigger je nach aktuellem Kompositionskontext und dessen Änderung entscheiden, ob Überschreiben oder Beibehalten bereits angezeigter Ergebnisse sinnvoller ist. Zusätzlich existieren obligatorische Methoden, die der Realisierung der Transitionen im Lebenszyklus aus Kapitel 6.3.2 dienen. Nachfolgende Auflistung beschreibt sie, wobei in Klammern die zugehörigen Aktivitäten gemäß dem Lebenszyklus notiert sind, falls zutreffend.

1. Konstruktor zur Erzeugung einer Instanz des Triggers
2. Initialisierungsmethode, mittels derer ein Trigger Zugang zu den Schnittstellen nach außen erlangt (Initialisieren)

3. Eine Rückrufmethode, die dem Trigger signalisiert, dass er sich im Zustand *registriert* befindet. Eine wichtige Anwendungsmöglichkeit für einen Triggerentwickler ist der direkte Übergang in den Zustand *rechnend*, um initiale Abfrage von Kontextinformationen und dergleichen durchzuführen.

4. Destruktor zur strukturierten Freigabe interner Datenstrukturen (Abbauen)

Obgleich ein Trigger komplett deklarativ beschrieben werden könnte, das heißt ohne den MRE-spezifischen Teil ② und dessen Einbindung im deklarativen Teil ①, wurde von einer solchen Lösung abgesehen. Dies würde einen komplizierten XML-Dialekt erfordern, der in der Lage ist, Bedingungen, Kontroll- und Datenflüsse zu verfassen. Um den Entwicklungsaufwand durch Verzicht auf das Einarbeiten in einen derartigen XML-Dialekt, geringer zu halten, und weil Kenntnisse über gängige Programmiersprachen seitens der Triggerentwickler unterstellt werden können, wurde der vorgestellte Ansatz gewählt.

Architektonische Betrachtungen

Dieser Abschnitt thematisiert die trigger-spezifischen Architekturbestandteile und deren Einordnung in eine Mashup-Plattform anhand von Abbildung 6.6.

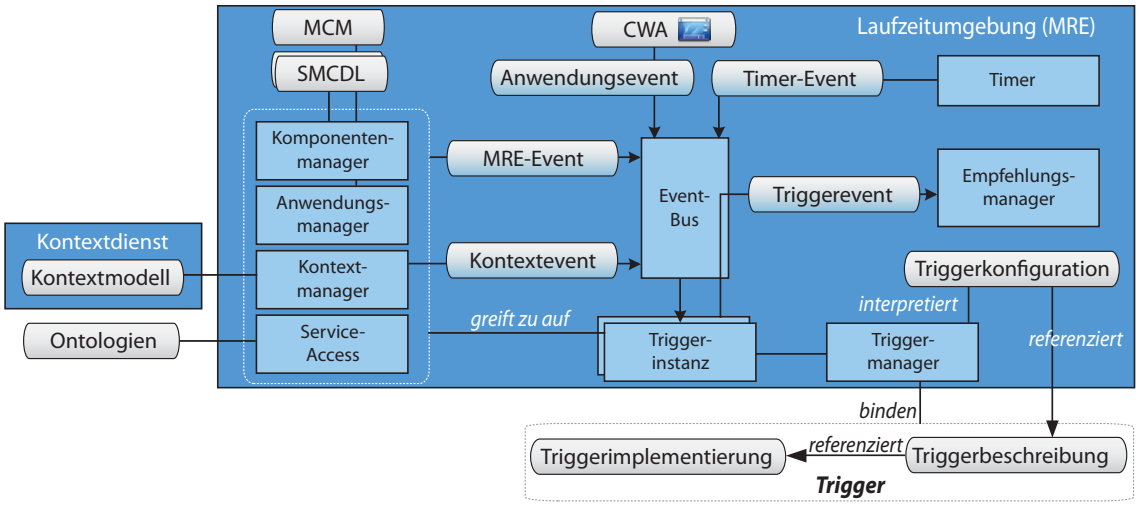


Abbildung 6.6: Triggerspezifischer Ausschnitt der Architektur

Schnittstellen zum MRE Als Softwareeinheit besitzen Trigger Schnittstellen zur Laufzeitumgebung, in die sie integriert sind. Diese dienen einerseits dem Erhalten von Informationen durch aktive Abfrage oder passiv mittels Benachrichtigungen. Andererseits erlauben die Schnittstellen einem Trigger, Daten zu veröffentlichen, falls er auslöst. Wesentliche **Eingaben** für einen Trigger sind seine Startevents. Zeitgleich sind diese mit ihren transferierten Nutzdaten typischerweise eine wichtige Informationsquelle. Als einfaches Beispiel sei hier das MRE-Event `componentInitialized` genannt, das die ID der initialisierten Komponente enthält. Weiterhin erhalten Trigger Zugriff auf Webservices und Webressourcen via *Service-Access* sowie Zugang zu folgenden Modellen, um ihre Bedingungen auszuwerten: das Kompositionsmodell der aktuellen Anwendung über

den *Anwendungsmanager*; die Komponentenbeschreibungen der aktuell vorhandenen Komponenten über den *Komponentenmanager*; das Kontextmodell inklusive Profil des aktuellen Nutzers über den *Kontextmanager*, der sowohl synchrone Kontextabfragen als auch Möglichkeiten zur Registrierung auf asynchrone Benachrichtigungen bietet; sowie Domänen-Ontologien, die zur Annotation der Komponenten verwendet werden, über den *Service-Access*. Löst ein Trigger aus, sammelt er optional weitere Informationen und erzeugt schließlich eine **Ausgabedatenstruktur**. Diese ist in ihrem grundsätzlichen Aufbau vordefiniert und wird als ein MRE-Event über den Event-Bus an den Empfehlungsmanager geschickt, sodass eine lose Kopplung zum Trigger gegeben ist und Trigger kaskadiert werden können. Grundlegend ist die Ausgabedatenstruktur aus folgenden drei Teilen aufgebaut.

- Die **Kennung des Triggers** wird zur Behandlung durch den Empfehlungsmanager benötigt, zum Beispiel für die Zuordnung der Empfehlungsmethoden.
- **Grund**: Auch eine menschenlesbare Beschreibung des identifizierten Empfehlungsgrunds wird geliefert, was insbesondere zur Herstellung von Nachvollziehbarkeit und Vertrauen beim Nutzer eingesetzt werden kann. Dies ist vor allem bei impliziten Triggern relevant, während es bei expliziten weniger eine Rolle spielt, da das Einholen von Empfehlungen in dem Fall der Intention des Endnutzers entspricht. Die Beschreibung wird vom Trigger jeweils passend zur konkreten Situation maßgeschneidert, indem zum Beispiel der Name der betroffenen Komponente eingesetzt wird. Dies ist notwendig, da Bedingungen typischerweise unabhängig von bestimmten Instanzen sind oder nur grobe Filter definieren.
- **Konfiguration** der nachfolgenden Schritte des Empfehlungskreislaufs: Dabei ergänzen trigger-spezifische Werte die Konfiguration der Empfehlungsstrategie oder werden von letzterer überschrieben. Dies umfasst drei Aspekte:
  1. Konfiguration der Handhabung wiederholter Triggerauslösungen, falls ein vom selben Trigger zuvor initiierter Empfehlungskreislauf noch nicht abgearbeitet wurde, sodass beispielsweise zugehörige Empfehlungen noch berechnet oder noch dargestellt werden. Dies ist notwendig, da der Trigger losgelöst von den nachfolgenden Schritten operiert. Es kann definiert werden, dass das neue Auslösen des Triggers (1) ignoriert werden soll, sodass vorhandene Ergebnisse erhalten bleiben, (2) zu einer erneuten Empfehlungsberechnung führt und vorhandene dargestellte Empfehlungen zu ersetzen sind, oder (3) zu einer erneuten Empfehlungsberechnung führt, deren Ergebnis neben den vorhandenen Empfehlungen anzuzeigen ist.
  2. Konfiguration der Empfehlungsberechnung: Ein Trigger liefert Eingabeparameter, welche die anzusprechenden Empfehlungsmethoden benötigen.
  3. Trigger konfigurieren zudem die Integration von Kompositionsfragmenten in eine CWA, da sie Informationen über den Empfehlungskontext liefern, wie die Kennung einer Komponente, für die Alternativen berechnet wurden.

Ein Mitsenden der mit dem Trigger assoziierten Empfehlungsmethoden ist nicht erforderlich, da der Empfehlungsmanager bereits eine Zuordnungstabelle pflegt, was im folgenden Unterkapitel näher behandelt wird. Diese Zuordnung wird dynamisch in Abhängigkeit der zu verwendenden Trigger befüllt, sodass keine statische Rekonfiguration des Empfehlungsmanagers nötig wird.

Involvierte Architekturkomponenten und Abläufe Die *Triggerkonfigurationen* der aktiven Empfehlungsstrategien deklarieren die im aktuellen MRE zu nutzenden Trigger via URIs, unter denen die jeweiligen Triggerbeschreibungen abrufbar sind. Als zentrale Instanz zur Verwaltung von Triggern dient der **Triggermanager**. Er bezieht die Triggerbeschreibungen und interpretiert diese. Er orchestriert nachfolgend alle notwendigen Schritte, um anhand der Informationen in der Beschreibungsdatei alle weiteren involvierten Architekturkomponenten des MRE zu konfigurieren und somit den Lebenszyklus von Triggern zu verwirklichen. In jedem Schritt kann es im Fehlerfall zum Abbruch kommen, beispielsweise falls eine URI nicht erreichbar oder die Triggerbeschreibung invalide ist.

1. Laden aller benötigten Quellcode-Dateien
2. Instantiieren des Triggers durch Aufruf des Konstruktors
3. Aufrufen der Initialisierungsmethode. Dabei wird ein Kontext für den Trigger eingerichtet, über den er auf die Schnittstellen nach außen zugreifen kann.
4. Registrieren von Startevents und Ausgabeevent des Triggers am Event-Bus
5. Hinzufügen der Zuordnung von Triggerkennung zu Empfehlungsmethoden beim Empfehlungsmanager
6. Aufrufen der Rückrufmethode, um den Abschluss der Registrierung zu signalisieren.

Für Trigger, zu deren Startevents Kontextevents gehören, erfolgt eine Registrierung auf Benachrichtigungen der jeweiligen Kontextparameter beim Kontextmanager. Trigger können im Rahmen ihres Lebenszyklus über die Schnittstelle zum Kontextmanager eine spätere Registrierung auf Änderungen weiterer Kontextparameter veranlassen.

Der Triggermanager ist zudem dafür verantwortlich, eine verzögerte Registrierung von Triggern durchzuführen. Betroffen sind hierbei Trigger, deren referenzierte Kanal-Event-Tupel noch nicht vorhanden sind oder für die noch kein Event mit passenden Signatur identifiziert wurde. Solche Trigger verbleiben im Zustand *deregistriert* und der Triggermanager überprüft beim Hinzukommen neuer Kanäle, ob eine Registrierung notwendig ist und nimmt diese gegebenenfalls vor. Dazu erfolgt ein Vergleich der Beschreibung von Startevents pro betroffenen Trigger mit den Signaturen hinzugekommener Kanäle. Ein weiterer Fall sind Trigger, die im Modell des aktuellen Nutzers als zu deaktivieren markiert sind. Auch diese verharren im Zustand *deregistriert*, sodass die Schritte 4–6 entfallen. Entscheidet sich der Nutzer im Laufe der Sitzung um, kann jederzeit eine verzögerte Registrierung des Triggers stattfinden. Des weiteren dient der Triggermanager dem Deregistrieren von Triggern, etwa im gehäuften Fehlerfall, bei expliziter Deaktivierung durch den Nutzer oder beim Entfernen der Kanäle, auf denen der Trigger angemeldet ist.

Der **Event-Bus** leitet Startevents über die entsprechenden Kommunikationskanäle zur Triggerinstanz. Neben dem reinen Verteilen von Events führt er *Complex Event Processing (CEP)* [EB09] durch, gemäß der Spezifikation komplexer Startevents in Triggerbeschreibungen. Beim CEP werden atomare Events nicht konsumiert, sodass diese auch an weitere Empfänger propagiert werden können. Zusätzlich führt der Event-Bus die in Triggerbeschreibungen geforderte Korrelation von Events durch.

Jedem proaktiven Trigger wird ein eigener **Timer** zugeordnet, der im Intervall laut Triggerbeschreibung ein Event auf einem pro Timer-Trigger-Paar eingerichteten Kanal veröffentlicht. Ein Timer wird vom Triggermanager instantiiert. Er ist an den Lebenszyklus des Triggers gekoppelt, sodass er mit dem Trigger zusammen abgebaut wird und die Registrierung beziehungsweise Deregistrierung des Triggers die Aktivierung beziehungsweise Deaktivierung des Timers nach sich zieht.

Der **Empfehlungsmanager** nimmt Triggerausgaben entgegen und pflegt eine Zuordnung von Triggerkennung zu Empfehlungsmethoden, die beim Initialisieren und Registrieren der Trigger durch den Triggermanager entsprechend der aktiven Empfehlungsstrategien gefüllt wird. Anhand dieser Zuordnung stellt der Empfehlungsmanager fest, welche Empfehlungsmethoden für einen ausgelösten Trigger relevant sind.

Zur Vermeidung der Überlastung von Nutzern mit zu vielen Empfehlungen bietet das Konzept bereits auf Ebene von Triggern mehrere Ansätze. Lösen mehr Trigger zeitnah aus als vom MRE parallel bedient werden können, wird durch *Priorisierung* eine Reihenfolge der zugehörigen Recommendation-Jobs hergestellt. Die initiale Priorität ergibt sich gemäß der Ordnung *proaktiv* $<$ *reaktiv* $<$ *explizit*, damit der Intention des Nutzers, Empfehlungen zu erhalten, Rechnung getragen wird. Danach erfolgt das *Filtern* der Jobs. Dabei wird geprüft, ob es einen Recommendation-Job gibt, der aufgrund eines vorhergegangenen Auslösens des selben Triggers vorliegt, und ob die zugehörigen Triggerausgaben gleich sind. Wurden betroffene Recommendation-Jobs identifiziert, entscheidet der Empfehlungsmanager anhand des Konfigurationsteils der Triggerausgabe oder anhand empfehlungsstrategiespezifischer Werte wie mit dem aktuellen Recommendation-Job weiter zu verfahren ist. Im Fall des Ignorierens wird der neue Recommendation-Job verworfen. Ansonsten wird er in die Warteschlange eingefügt. Beim Ersetzen sind betroffene Recommendation-Jobs zusätzlich abzubrechen. Für den am höchsten priorisierten neuen Recommendation-Job beginnt anschließend die Empfehlungsberechnung.

6.3.3 Empfehlungen berechnen

Als eine zentrale Verantwortlichkeit übernimmt der Empfehlungsmanager die Phase ② *Empfehlungen berechnen*, vergleiche Abbildung 6.3. Dazu verwendet er eine Menge von Empfehlungsmethoden. Wie in Abschnitt 6.1 eingeführt, stellen diese eine konzeptionelle Erweiterung der *Web-API-Search-Patterns* aus [BDM13b] dar. Jede Empfehlungsmethode dient einem wohldefinierten Zweck und legt dazu folgende Aspekte fest, die im Rahmen der Beschreibung der Prozessschritte Matching und Ranking im Verlauf dieses Kapitels detailliert werden:

- *Eingabeparameter*: Zur Erbringung der Funktionalität werden typischerweise Informationen zum aktuellen Anfragekontext benötigt, die sich in zwei Kategorien einordnen lassen: empfehlungsmethoden-spezifische Parameter, zum Beispiel Suchbegriffe oder ein Komponentenevent, für das Couplings vorzuschlagen sind, sowie Parameter, welche grundsätzlich übergeben werden. Bei letzteren handelt sich um 1. die Nutzerkennung zum Zugriff auf das Nutzerprofil, 2. den Kompositionskontext (Kompositionsfragment des Nutzers), 3. die Anforderungen des Nutzers sowie 4. Modifikatoren der Ergebnismenge, zum Beispiel ein Limit.

- *Matching-Schema*: Dieses definiert die Arten von Kandidaten, das heißt, Komponenten, CWA und Pattern-Klassen. Weiterhin beinhaltet es anzuwendende Metriken, deren Gewichtungen wm_i und Schwellenwerte tm_i. Zudem wird ein Operator Ω festgelegt, der definiert, wie die einzelnen Metriken verrechnet werden, und standardmäßig dem arithmetischen Mittel entspricht.

- *Ranking-Schema*: Es dient der Konfiguration des Rankings, indem es festlegt, welche Kriterien in das Rating einfließen sollen. Die Gewichtungen wr_i und die

Schwellenwerte tr_i von Kriterien sowie die Verrechnungsvorschrift Θ zum Bilden des Gesamtratings, zum Beispiel anhand des Mittelwerts, werden zudem definiert.

Der Empfehlungsmanager nimmt die Trigger-Ausgabedatenstruktur entgegen und realisiert die zugehörigen Empfehlungsmethoden. Wie in Kapitel 6.3.2 beschrieben wurde, definieren Trigger, welche Empfehlungsmethoden sie ansprechen, und sind dafür verantwortlich, sämtliche empfehlungsmethoden-spezifischen Eingabeparameter bereitzustellen. Aus dem Konfigurationsteil in der Triggerausgabe werden essentielle Daten extrahiert und gegebenenfalls mit Standardwerten und strategiespezifischen Werten angereichert oder überschrieben. Zusammen mit dem Matching- und Ranking-Schema gemäß der aktuellen Empfehlungsstrategie dient dies schließlich als Eingabe für die Empfehlungsmethoden. Diese beziehen diverse weitere Datenquellen ein, siehe Abbildung 6.3: die Repositorien für Pattern-Instanzen, Komponenten, Anwendungen und Feedback sowie den Kontextdienst zum Zugriff auf Nutzerkontextmodelle.

Im Grunde führt jede Empfehlungsmethode die Schritte **Matching** und **Ranking** durch. Dieses generische Grundgerüst zur Empfehlungsberechnung wird entsprechend der Empfehlungsstrategie sowie der Anfrage konfiguriert und vom Empfehlungsmanager koordiniert. Nachfolgend werden beide Schritte im Detail behandelt.

Matching

Hauptverantwortlichkeit des Matchings ist es, zur Anfrage passende Kompositionsfragmente zu identifizieren. Die Passgenauigkeit wird dabei in erster Linie dadurch definiert, dass ein Kandidat strukturell und funktional zur Anfrage passt. Dies umfasst rein filternde Schritte und solche, bei denen Abstufungen bezüglich des Übereinstimmungsgrades zulässig sind. Letztlich wird eine Menge von Tupeln aus Kandidaten und deren Passgenauigkeit zur Anfrage geliefert. Die prinzipiellen Schritte des Grundablaufs werden nachfolgend vorgestellt, wobei die genaue Ausgestaltung einzelner Schritte je nach Kandidatentyp variieren kann.

1. Bestimmung der Ausgangsmenge an Kandidaten K In Abhängigkeit von der Empfehlungsmethode und der benötigten Ergebnisartefakte werden aus den jeweiligen Repositorien sämtliche Kandidaten bezogen.

2. Vorselektion Anschließend wird die potentiell große Kandidatenmenge anhand boolescher Kriterien vorgefiltert. Als Resultat liegt eine Menge $K' \subseteq K$ vor.

Zuerst werden **obligatorische funktionale und nicht-funktionale Anforderungen** aus der Anfrage und dem Nutzerprofil überprüft und Kandidaten entfernt, die diese nicht erfüllen. Werden beispielsweise Erweiterungsvorschläge ausgehend von einem bestimmten Event einer Komponente benötigt, kommen lediglich Couplings in Frage, welche die Komponente und das geforderte Event beinhalten. Eine filternde Qualitätsanforderung kann etwa darin bestehen, dass ein Nutzer nur kostenlose Komponenten akzeptiert.

Für die verbleibenden Kandidaten wird die Eignung im Ausführungskontext des aktuellen Nutzers bewertet. Dazu werden die Anforderungen des Kandidaten an den Ausführungskontext bestimmt und gegen das Kontextmodell abgeglichen. Beispiele für solche Anforderungen sind zum Beispiel das genutzte MRE, Software- und Hardwareeigenschaften des Geräts sowie Eigenschaften des Nutzers wie vorhandene Accounts.

Weiterhin findet bei vielen Empfehlungsmethoden eine Vorfilterung anhand des Kompositionskontextes statt. Je nach Kandidatenart fällt dies unterschiedlich aus.

- *Coupling*: Es wird sichergestellt, dass durch Pattern-Instanzen keine Zyklen entstehen. Optional kann ein Kopplungstest vorsehen sein, der sichergestellt, dass Pattern-Instanzen mit der aktuellen Komposition überlappen. Das ist der Fall, sobald eine gemeinsame Komponente auftritt. Weiterhin werden Pattern-Instanzen mit vorhandenen Kommunikationskanälen und Komponenten abgeglichen, um auszuschließen, dass sie bereits Teil der Anwendung sind (Inklusionstest).
- *Co-Occurence*: Es wird geprüft, ob nicht bereits beide Komponenten der Pattern-Instanz in ähnlicher Weise in der Komposition vorhanden sind. Dies wird die Suche nach entsprechenden Exchangeability-Patterns verwirklicht.
- *Complex*: Analog zu Couplings findet ein Kopplungs- und Inklusionstest statt.

3. Berechnung von Metriken Im nächsten Schritt gilt es, die Eignung jedes Kandidaten aus K' zu berechnen. Welche Metriken hierzu relevant sind, hängt von der Empfehlungsstrategie ab und wird im Matching-Schema der Empfehlungsmethode festgelegt. Im Folgenden werden konzeptionell vorgesehene Metriken aufgeführt, die etablierte Konzepte einbeziehen. Der Fokus liegt auf neuen Metriken wie der funktionalen Relevanz. Das Konzept lässt es zu, existierende Ansätze ergänzend hinzuzunehmen. Dazu wären diese in das Matching-Schema und in diesen sowie den nächsten Schritt, dem Verrechnen der einzelnen Metriken, einzubeziehen.
I. Erfüllungsgrad nicht-obligatorischer funktionaler Anforderungen $match_{req}$: Dazu werden funktionale Anforderungen, die zum Beispiel vom Nutzer als optional deklariert wurden, gegen die verfügbare Kandidatenmenge ausgewertet. Das umfasst **Capability-** und **Freitextanforderungen**, die beispielsweise vom Nutzer mithilfe des Wizards formuliert wurden. Zur Erläuterung des Vorgehens sei auf Abbildung 6.7 verwiesen. Diese illustriert, wie für eine Freitext- und eine Capability-Anforderung der Erfüllungsgrad durch zwei Kandidaten, auf der linken Seite für eine Komponente und rechts für eine Anwendung, bestimmt wird.

Zunächst werden **Capability-Anforderungen**, im Beispiel `Search Route`, gegen die Capabilities von potentiellen Kandidaten evaluiert, vergleiche Punkt 3 in Abbildung 6.7. Um dies zu gewährleisten wird nachfolgend ein Maß für die Ähnlichkeit von Capabilities hergeleitet. Für atomare Capabilities wurde dazu in Anlehnung an semantikbasierte Ansätze, wie [PRM11b; BDM13b], bereits in Kapitel 5.4.2 die Funktion sim_{atomic} spezifiziert. Die Ähnlichkeit von zwei kompositen Capabilities wird rekursiv anhand der strukturellen Übereinstimmung und sim_{atomic} gemessen. Es wird jedoch davon ausgegangen, dass vorrangig atomare Capabilities als funktionale Anforderungen vom Client gestellt werden, vergleiche Abschnitt 7.4. Der Abgleich einer atomaren Capability ac_{req}, im Beispiel `Search Route`, und einer von einem Kompositionsfragment offerierten kompositen Capability cc_{adv}, im Beispiel `Plan Trip`, beginnt bei der Wurzel von cc_{adv}. Es wird $sim_{atomic}(ac_{req}, cc_{adv})$ berechnet und in eine Menge L_1 eingefügt. Anschließend erfolgt ein analoges Vorgehen für die Ebenen der Kinder und Kindeskinder von cc_{adv} (im Beispiel existiert nur eine Kinderebene). Pro Ebene werden alle Capabilities über sim_{atomic} mit ac_{req} verglichen und der Wert in Menge L_2 beziehungsweise L_3 hinterlegt. Das höchste Ergebnis einer Ebene wird durch zwei (L_2) respektive drei (L_3) dividiert, sodass Treffer auf oberer

Hierarchieebene stärker gewichtet werden. Der Wert $sim_{composite}$ ist darauf aufsetzend definiert als der höchste der maximal drei Einzelwerte, in Abbildung 6.7 rechts demnach das Maximum von 0.22 und 0.5. Basierend auf diesen Formeln wird pro Kandidat jede seiner Capabilities mit jeder geforderten Capability verglichen. Daraus ergibt sich pro geforderter Capability ein maximaler Übereinstimmungsgrad. Im Beispiel existiert daher pro Kandidat genau ein Wert, $match_{SearchRoute}$.

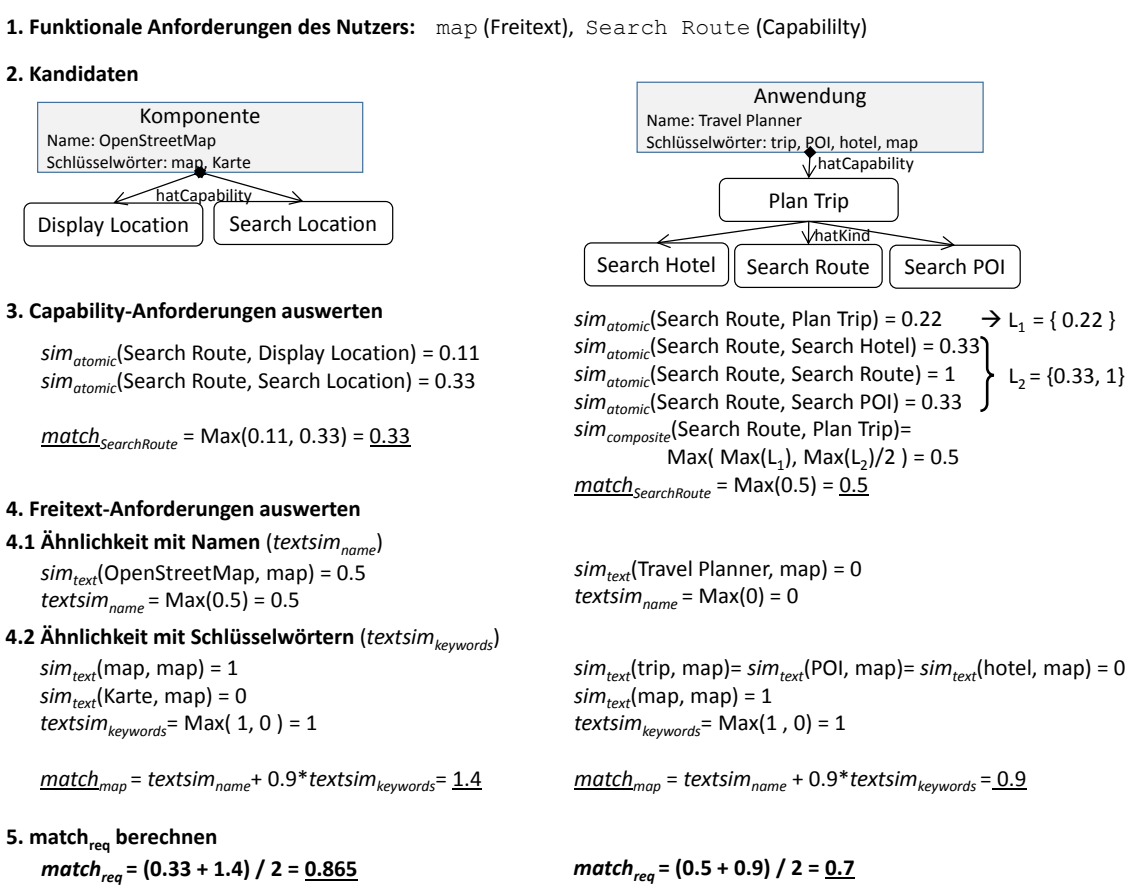


Abbildung 6.7: Beispiel zur Berechnung von $match_{req}$

Weiterhin werden **Freitextanforderungen** evaluiert, siehe Punkt 4 in Abbildung 6.7. Der Grundansatz zur Auswertung besteht hierbei darin, dass eine Menge geeigneter Metadaten pro Kandidatentyp, wie der Name und die Schlüsselworte von Komponenten und Anwendungen, anhand eines spezifischen Faktors w_{md} relevanzgewichtet und gegen die Anfrage ausgewertet werden. Dazu werden diese Metadaten eines Kandidaten mit allen Einzelwörtern pro Anforderung, »map« im Beispiel aus Abbildung 6.7, abgeglichen. Dabei wird die textuelle Ähnlichkeit wie in Suchmaschinen üblich berechnet und die maximale Übereinstimmung wird pro untersuchtem Metadatum gespeichert, im Beispiel $textsim_{name}$ und $textsim_{keywords}$. Die Einzelergebnisse werden entsprechend mit w_{md} zwecks Gewichtung multipliziert, im Beispiel gilt $w_{name} = 1$ und $w_{keywords} = 0.9$. Anschließend werden die resultierenden Werte zu einem Einzelwert aufaddiert, der den Erfüllungsgrad der textuellen Anforderung darstellt, im Beispiel $match_{map}$. Durch das Summieren werden Kandidaten bevorzugt, bei denen in mehreren Metadaten-Feldern

Treffer hinsichtlich der Freitextanforderung vorliegen. Im Beispiel in Abbildung 6.7 wird dies auf der linken Seite im Fall der Komponente verdeutlicht.

Pro funktionaler und pro textueller Anforderung liegen Evaluationsergebnisse vor, die zu $match_{req}$ für den aktuellen Kandidaten gemittelt werden (siehe 5. in Abbildung 6.7).
II. Funktionale Relevanz $match_{relevance}$: Ziel dieses Schrittes ist die Bewertung der funktionalen Relevanz eines Kandidaten in Bezug auf den vorliegenden Kompositionskontext. Die Berechnung findet statt, falls es sich bei der Art von Kandidaten um Instanzen der Pattern-Klassen Coupling, Co-Occurrence und Complex handelt, da nur diese direkten Einfluss auf die Anwendungsfunktionalität besitzen.

Grundlage für die Bestimmung von $match_{relevance}$ ist der in Kapitel 5.4 beschriebene Wissensgraph, der gewichtete Relationen zwischen Capabilities modelliert. Die Berechnung erfolgt unter Einbeziehung dieses Wissens auf folgende Weise.

Gegeben seien die Capabilities C_{app} des Kompositionskontextes und C_{cand} des Kandidaten. Dann werden für sämtliche Capability-Paare (c_a, c_b) aus $C_{app} \times C_{cand}$ die nachfolgenden Schritte durchgeführt. Erst werden sämtliche Relationen zwischen den Knoten, die c_a und c_b repräsentieren, aus dem Wissensgraphen ausgelesen. Darauf aufbauend kann $relevance_{a,b}$ nach folgender Formel kalkuliert werden.

$$relevance_{a,b} = \frac{k_1 \cdot Max(subsumes, precedes, occursWith) + k_2 \cdot relatedTo - k_3 \cdot similarTo}{\sum_{i=1}^{3} k_i}$$

Die Relationen beziehungsweise deren Werte im Wissensgraph werden gemäß obiger Formel linear kombiniert, mit $k_i + k_2 + k_3 > 0$. Der Wert ergibt sich demnach aus drei Bestandteilen: dem Maximum der drei Werte *subsumes*, *precedes* und *occursWith*, die strukturelle Zusammenhänge von Capabilities beschreiben, dem Wert von *relatedTo*, der inhaltliche Bezüge von Capabilities repräsentiert, sowie der auf Vererbungsbeziehungen beruhenden Ähnlichkeit in *similarTo*. Für eine Definition dieser Relationen sei auf Kapitel 5.4.1 verwiesen. Der Wert für *similarTo* wird subtrahiert. Der Grundgedanke dabei ist, dass ein Kandidat weniger funktional relevant scheint, wenn er Capabilities beisteuert, die ähnlich zu bereits im Kompositionskontext vorhandenen sind. Zum Beispiel sei c_a = `Select Location` und c_b = `Display Weather`. Angenommen für c_a und c_b gelte $subsumes = 0$, $occursWith = 0.4$, $precedes = 0.5$, da beide Capabilities recht häufig zusammen auftreten und sogar oft verknüpft werden, $relatedTo = 0.7$, da `Location` im Domänenmodell mit `Weather` assoziiert ist, und $similarTo = 0$, da keine Vererbungsbeziehungen auftreten. Dann ergibt sich mit $k_i = 1$ für $relevance_{a,b} = 0.4$. Das Maximum sämtlicher $relevance_{i,j}$ bildet schließlich $match_{relevance}$.
III. Strukturelle Ähnlichkeit $match_{composition}$: Diese Metrik spielt vor allem für Empfehlungsmethoden eine Rolle, deren Kandidatenart Pattern-Klassen umfasst. Analog zu [BDM13c] wird im Prinzip kalkuliert, wie stark das Kompositionsfragment einer Pattern-Instanz mit dem aktuellen Kompositionskontext übereinstimmt. Neben einer reinen Betrachtung von Komponenten wie in [BDM13c] finden zusätzlich Kommunikationskanäle Berücksichtigung. Der Wert ergibt sich nach folgender Vorschrift.

$$match_{composition} = \frac{components_{covered} + channels_{covered}}{components_{app} + channels_{app}}$$

Die Variablen $components_{covered}$ und $channels_{covered}$ bezeichnen die Anzahl von Komponenten und Kommunikationskanälen, die sowohl im Kompositionskontext als auch im Kandidat auftreten; die Gesamtanzahl der Komponenten und Kommunikationskanäle im aktuellen Kompositionskontext repräsentieren $components_{app}$ respektive $channels_{app}$.

4. Berechnung des finalen Übereinstimmungsgrades Für Kandidaten, welche alle Filterschritte passieren, erfolgt schließlich die Aggregation oben definierter Werte anhand des Operators Ω und der Gewichtungen wm_i aus dem Matching-Schema, mit $\sum_{i=1}^{3} wm_i = 1$, wie folgt:

$$rating_{match} = \Omega(wm_1 \cdot match_{relevance}, wm_2 \cdot match_{req}, wm_3 \cdot match_{composition})$$

Dieser Ansatz hat den Vorteil, dass komplementäre Aspekte durch dedizierte Metriken repräsentiert und je nach Empfehlungsmethode bedarfsgerecht gewichtet werden können. Auch können Metriken durch Setzen des Gewichts auf Null ausgeschlossen werden.

5. Nachbearbeitung Abhängig von der Empfehlungsmethode wird optional die verbleibende Kandidatenmenge nochmals gefiltert. Beispielsweise findet bei der Suche nach Complex-Patterns eine Gruppierung hinsichtlich der in Pattern-Instanzen enthaltenen Komponenten und der erbrachten Funktionalität statt. Das heißt, dass verschiedene technische Realisierungen der selben Funktionalität zusammengefasst werden und lediglich die höchstbewerteten Pattern-Instanzen aus den Gruppen in der Ergebnismenge verbleiben. Zudem werden in diesem Schritt die Schwellenwerte tm_i durchgesetzt und Kandidaten verworfen, die bezüglich eines Teilratings oder $rating_{match}$ unter dem geforderten Mindestwert liegen.

Als Ergebnis des Matchings liegen Tupel (Kandidat, $rating_{match}$) vor, welche im nachfolgenden Schritt des Rankings anhand mehrerer Kriterien sortiert werden.

Ranking

Im anschießenden Schritt des Rankings werden die verbleibenden Kandidaten aus dem Matching in eine Reihenfolge gebracht. Die Bestimmung dieser Reihenfolge basiert dabei – unabhängig von der Art der Kandidaten – auf folgenden Dimensionen.

- $rating_{match}$: Das Ergebnis aus dem Matching trifft eine Aussage darüber, wie funktional passgenau ein Kandidat hinsichtlich des Anfragekontextes ist.
- $rating_{usage}$: Dieser Wert spiegelt wider, mit welcher Wahrscheinlichkeit ein Nutzer angesichts seiner bisherigen Nutzungshistorie einen Kandidaten erneut auswählen würde. Dies stützt sich auf das Feedback-Metamodell aus Abschnitt 5.1.4. Durch Übertragung des Konzepts *Frecency* [@MDN] gelingt es, Häufigkeit und Neuheit in einem einzelnen Wert zu kombinieren. Zudem werden die Nutzeraktivitäten von Feedback-Einträgen berücksichtigt. Im Vergleich zu Ansätzen, die ausschließlich die zeitliche Dimension beachten, erlaubt dies, die Relevanz anhand der Häufigkeit zu gewichten. Umgekehrt ermöglicht es, verglichen mit rein häufigkeitsbasierter Berechnung, aktuelle Trends zu beachten, da der Einfluss von Feedback exponentiell über die Zeit abgeschwächt wird. Zum Bestimmen der Häufigkeit werden sämtliche Feedback-Einträge $f \in F_u$ des aktuellen Nutzers, deren Subjekt der jeweilige Kandidat ist, identifiziert und mittels $sim(f, rec)$ die Ähnlichkeit von deren Feedback-Kontext zum Empfehlungskontext rec, vergleiche Kapitel 5.1.4, berechnet. Die Definition des Ähnlichkeitsmaßes sim hängt dabei von der jeweiligen Empfehlungsmethode ab. Sollte keine Nutzerhistorie vorliegen oder falls die aktuelle

Empfehlungsstrategie dies grundsätzlich vorsieht, wird das Feedback anderer Nutzer mit einbezogen. Analog zur zuvor beschriebenen Prozedur erfolgt die Identifikation passender Feedback-Einträge F_g aus den Nutzerprofilen.

Der letztliche Wert $rating_{usage}$ ergibt sich nach [@MDN] gemäß folgender Formel:

$$rating_{usage} = \frac{1}{|F|} \sum_{f \in F} (sim(f, rec) \cdot score(f_{activity}) \cdot \mathrm{e}^{\lambda \cdot age})$$

Es gelte $F = F_u \cup F_g$. Die Funktion $score$ ermittelt einen Wert zwischen 0 und 1, der die Nutzeraktivität gewichtet. $\mathrm{e}^{\lambda \cdot age}$ spiegelt die Aktualität des Feedback-Eintrags wider, indem exponentieller Zerfall kalkuliert wird. Dabei fließen das Alter age eines Feedback-Eintrags in Millisekunden, berechnet durch Subtraktion des Zeitstempels des Eintrags vom aktuellen Zeitpunkt, und eine Zerfallsrate λ ein. Für eine Halbwertszeit von 30 Tagen gilt zum Beispiel $\lambda = \frac{\ln 2}{30 \cdot 24 \cdot 60 \cdot 60 \cdot 1000 ms}$.

- $rating_{users}$: Repräsentiert die durchschnittliche explizite Bewertung eines Kandidaten, die aus Einträgen gemäß dem Feedback-Metamodell 5.1.4 abgeleitet wird.
- $rating_{quality}$: Bewertet die Qualität des Kandidaten, indem der Erfüllungsgrad der Qualitätsanforderungen des aktuellen Nutzers bestimmt wird. Dabei wird auf Ergebnisse anderer Arbeiten wie [Rüm+14] verwiesen. Auch das Berechnen eines Qualitätsmaßes für den Kandidaten, wie in [Pic+10], kann in diesen Wert aufgenommen werden, indem eine Menge von Standard-Qualitätsanforderungen gestellt wird, die für Qualitätseigenschaften eine Optimierungsrichtung vorgeben.
- $rating_{unexpectedness}$: Diese zusätzliche Variable zielt darauf ab, die *Serendipität* von Empfehlungsmethoden zu erhöhen. Per Definition müssen Empfehlungen dazu nützlich, unerwartet und neuartig für den Nutzer sein [KVW16; AT14]. Nützlichkeit beziehungsweise Relevanz wird bereits durch $rating_{match}$ ausgedrückt. Basierend auf Ansätzen wie [KB14; AT14] kann abgeschätzt werden, wie unerwartet ein Kandidat c für den Nutzer ist.

Auf Basis dieser Kriterien erfolgt im Zuge des Rankings die Berechnung von $rating_{final}$ jedes Kandidaten. Die Gewichtung und die Berücksichtigung von Kriterien kann je nach Empfehlungsstrategie in der Empfehlungsmethode konfiguriert werden, um verschiedene Intentionen zu reflektieren. Beispielsweise spielt es nicht in jedem Fall eine Rolle, unerwartete Ergebnisse zu erhalten, sodass in diesem Fall $rating_{unexpectedness}$ nicht bestimmt oder mit dem Einfluss $wr_5 = 0$ eingerechnet wird. Schließlich wird abhängig von der Empfehlungsstrategie nach Verrechnungsvorschrift Θ vorgegangen. Standardmäßig handelt es sich dabei um ein gewichtetes arithmetisches Mittel. Eine andere Option stellt *Dominance-Ranking* dar, was im Rahmen einer studentischen Arbeit [Wec16] durch Übertragung des Ansatzes von [Sko+09] auf die CRUISE-Plattform erprobt wurde.

$$\begin{aligned} rating_{final} = \Theta(& wr_1 \cdot rating_{match}, wr_2 \cdot rating_{usage}, wr_3 \cdot rating_{users}, \\ & wr_4 \cdot rating_{quality}, wr_5 \cdot rating_{unexpectedness}) \end{aligned}$$

Anschließend werden die Teilratings und $rating_{final}$ gegen die Schwellwerte tr_i geprüft und es wird aus den verbleibenden Kandidaten eine absteigend nach $rating_{final}$ sortierte Liste erzeugt. Je nach Anfrageparametern werden die Einträge limitiert, gegebenenfalls beginnend bei dem Eintrag an Stelle `offset`. Neben $rating_{final}$ werden sämtliche weiteren Teilratings und die durchschnittlichen Nutzerbewertungen, vergleiche Kapitel 5.1.4, zurückgeliefert. Dies erlaubt es Recommendation-Viewern, die Sortierung anzupassen oder durch den Nutzer beeinflussen zu lassen.

6.3.4 Präsentation von Empfehlungen

Anschließend werden in Phase ③ *Empfehlungen darstellen* die berechneten Empfehlungen in geeigneter Weise visuell aufbereitet und dem Endnutzer angezeigt. Ein zentraler Punkt hierbei ist, dass Nicht-Programmierern kommuniziert wird, welche Funktionalität eine Empfehlung bereitstellt beziehungsweise zum Mashup beiträgt. Dies basiert auf den Capabilities von Kompositionsfragmenten und ist ein Novum gegenüber existierenden Ansätzen. Weiterhin erfahren Nutzer, warum Empfehlungen erscheinen und woher diese stammen. Diesen Prozessschritt übernehmen *Recommendation-Viewer*, die Bestandteile eines MRE sind. Sie bieten eine Schnittstelle, über die sie in den Empfehlungskreislauf eingebunden werden können. Dazu gehört einerseits ein Eingang, der darzustellende Empfehlungen entgegennimmt, und ein Ausgang, der die Abarbeitung von Aktivität ③ *Empfehlungen darstellen* kommuniziert. Recommendation-Viewer wenden verschiedene Visualisierungstechniken an und halten Mittel zur Auswahl von Empfehlungen bereit. Der Empfehlungsmanager übermittelt die berechneten Empfehlungen zusammen mit der optionalen strategiespezifischen Viewer-Konfiguration, vergleiche Kapitel 6.1, an sämtliche Recommendation-Viewer gemäß der aktuellen Empfehlungsstrategie. Es existieren verschiedene Recommendation-Viewer, die in Kapitel 7 vorgestellt werden. Sie können sowohl dedizierte Module, zum Beispiel das Empfehlungsmenü, als auch integraler Bestandteil anderer Werkzeuge sein, wie der Capability-View. Neben der Visualisierung von Empfehlungen sind sie zudem dafür verantwortlich, den Abschluss von Phase ③ *Empfehlungen darstellen* an den Empfehlungsmanager zu melden. Gegebenenfalls umfasst dies die selektierten Empfehlungen. Zusätzlich können Recommendation-Viewer implizites und explizites Nutzerfeedback zur Qualität von Empfehlungen sammeln. Der Empfehlungsmanager erzeugt daraus Instanzen gemäß dem Feedback-Metamodell aus Abschnitt 5.1.4 und überträgt es zum Feedback-Repositorium.

6.3.5 Integration von Patterns

Im Zuge der Aktivität ④ *Empfehlungen integrieren* ist der Empfehlungsmanager in Kooperation mit dem Adaptionssystem einer Laufzeitumgebung für die Integration von Pattern-Instanzen in eine in Ausführung befindliche CWA zuständig. Abbildung 6.8 veranschaulicht den prinzipiellen Ablauf.

Die vorangegangenen Phasen eines Empfehlungsprozesses seien erfolgreich abgearbeitet worden. Im Recommendation-Job ist folglich eine vom Nutzer ausgewählte Pattern-Instanz notiert, welche es in das MCM der aktuell ausgeführten CWA einzubauen gilt. In einem ersten Schritt erfolgt das Bestimmen des Unterschieds zwischen aktuellem Kompositionsmodell und dem Kompositionsfragment der Pattern-Instanz. Wie in Abschnitt 6.3.2 beschrieben, liefert der vom Trigger ausgegebene Kontext dazu wichtige Anhaltspunkte zu

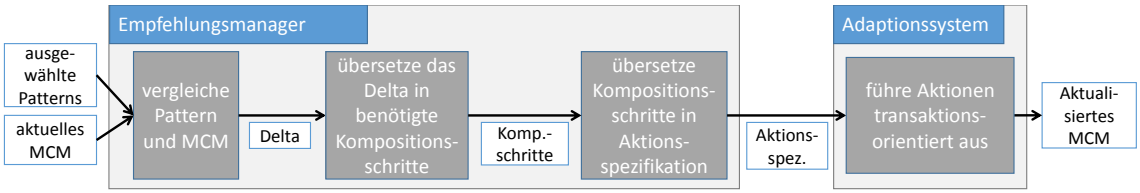


Abbildung 6.8: Details zum Ablauf der Integration von Pattern-Instanzen in eine CWA

Einfügepunkten einer Pattern-Instanz. Im beispielhaften Fall eines expliziten Triggers, der Empfehlungen zu Couplings aller Operationen einer Komponente $C1$ nach sich zieht, steht zum Beispiel fest, dass der Subscriber am Kommunikationskanal des Couplings $C1$ sein muss. Allerdings kann es vorkommen, dass keine eindeutige Lösung identifiziert werden kann, zum Beispiel falls $C2$ des gewählten Couplings bereits im Mashup instantiiert ist. Die Empfehlungsstrategie definiert das Vorgehen in solchen Fällen, beispielsweise, dass vorrangig vorhandene Komponenten zu nutzen sind oder dass der Nutzer diese Entscheidung treffen muss. Letzteres gilt es, angesichts der Zielgruppendefinition, möglichst selten in Erwägung zu ziehen und auf solche Fälle zu begrenzen, die sich auf Komponenten und nicht auf feingranularere Kompositionsdetails beziehen.

Darauf aufsetzend wird gefolgert, welche Kompositionsschritte durchzuführen sind, um das identifizierte Delta zu überbrücken. Angenommen im obigen Beispiel besteht der Unterschied aus dem Kommunikationskanal und $C2$. Dann wären zwei Kompositionsschritte durchzuführen: Hinzufügen von $C2$ und Etablieren des Kommunikationskanals.

Kompositionsschritte werden anschließend auf Adaptionstechniken des Adaptionssystems abgebildet. Die Komplexität dieser Abbildung hängt von verfügbaren Adaptionstechniken und deren Abhängigkeiten untereinander sowie von deren Granularität ab. Im Fall von CRUISE [PRM11a] umfassen die zur Verfügung stehenden Adaptionstechniken unter anderem das Hinzufügen, Entfernen, Austauschen und Rekonfigurieren von Komponenten, sowie das Hinzufügen, Konfigurieren und Entfernen von Kommunikationskanälen und das Anpassen von Layout sowie Screenflow.

Pattern-Klasse	**Kompositionsfragment**	**Adaptionstechniken**
Occlusion, Exchangeability	Komponente	Komponente austauschen
Co-Occurrence	Zwei Komponenten	Komponente hinzufügen
Coupling	Zwei Komponenten, ein Kanal	Komponente hinzufügen, Kanal hinzufügen, Publisher und Subscriber zum Kanal hinzufügen
Complex	`n` Komponenten, `m` Kanäle, `k` Layouts, ...	Aggregation von Adaptionstechniken

Tabelle 6.1: Abbildung von Pattern-Klassen auf Adaptionstechniken

Tabelle 6.1 illustriert die resultierende Abbildung von Pattern-Klasse auf eine Menge anzuwendender Adaptionstechniken, die vom Adaptionssystem angeboten werden. Ergebnis ist eine *Aktionsspezifikation*, die alle notwendigen Techniken definiert. Das Adaptionssystem trägt die Verantwortung für die Herstellung einer geeigneten Reihenfolge der anzuwendenden Adaptionstechniken. Dazu werden diese priorisiert und sortiert, um die Korrektheit des Ergebnisses sicher zu stellen. Die Prioritäten werden im Zuge des Abbildungsvorgangs definiert und können je nach Implementierung der Laufzeitumgebung

variieren. Zudem ist das Adaptionssystem für das transaktionsorientierte Umsetzen der notwendigen Adaptionstechniken zuständig. Das bedeutet, dass entweder alle Schritte fehlerfrei realisiert werden können oder die gesamte Transaktion abgebrochen wird. Letzteres ist zum Beispiel bei Auftritt eines Fehlers, wie dem Scheitern der Instanziierung einer Komponente, nötig und stellt sicher, dass der Zustand der CWA dem vor der Transaktion entspricht und insbesondere keine inkonsistenten Zustände erreicht werden. Das Adaptionssystem informiert ereignisbasiert in jedem Fall über den Ausgang der Transaktion, was beispielsweise genutzt wird, um den Nutzer in Kenntnis zu setzen.

6.4 Zusammenfassung

Bestandteil dieses Kapitels ist ein Empfehlungssystem für das EUD von CWA durch Nicht-Programmierer. Als Hauptbeitrag zum Stand der Technik wurde eine generische, konfigurierbare Empfehlungsinfrastruktur konzipiert. Diese trennt die Aspekte, *wann* Items *welcher* Art zu empfehlen sind, *wie* die Empfehlungen zu berechnen und *wie* sie zu visualisieren sind. Somit werden Aspekte adressiert, die in existierenden Empfehlungssystemen für Mashups nicht oder nur unzureichend beachtet wurden. Um den hochgradig iterativen Ad-hoc-Charakter der CWA-Entwicklung zu berücksichtigen, um den Anforderungen der Zielgruppe gerecht zu werden, und, um die vielfältigen, empfehlungsrelevanten Szenarien zu unterstützen, wurden die Konzepte von Empfehlungskreislauf und Empfehlungsstrategien entwickelt. Auf Basis von Empfehlungsstrategien kann der Gesamtprozess, den der Empfehlungskreislauf beschreibt, maßgeschneidert werden. Auf diese Weise ist das resultierende Empfehlungssystem hochgradig konfigurierbar und an den jeweiligen Anwendungskontext bedarfsgerecht anpassbar.

Das Konzept erfüllt sämtliche gestellte Unterpunkte von Anforderung 8. Auf Basis des Empfehlungskreislaufs und konkreten Ausprägungen dessen in Form von Empfehlungsstrategien werden Nicht-Programmierer durchgängig unterstützt. Insbesondere Trigger erlauben diesbezüglich das reaktive und proaktive Vorschlagen von Kompositionsfragmenten. Diese Arbeit beinhaltet die erste umfangreiche Systematisierung von Triggern. Verwandte Ansätze sind zuweilen auf wenige, statisch vorgesehene Auslösemechanismen beschränkt und im Vergleich zu [BDM13a; Roy+13] stechen insbesondere reaktive Trigger hervor. Hybridität und Kontextsensitivität werden durch die Metriken im Matching und Ranking von Empfehlungsmethoden realisiert. Diese basieren auf State-Of-The-Art-Techniken und adaptieren diese auf die Mashup-Domäne. Als Beispiele sind Frecency, Serendipität und fallbasiertes Schließen durch Pattern-Matching sowie durch Identifikation relevanter Feedback-Einträge zu nennen. Erstmals werden Nutzungskontext und funktionale Relevanz derart einbezogen. Hinsichtlich Kontextsensitivität kann zudem die neuartige Auswahl aktiver Empfehlungsstrategien genannt werden. Nachvollziehbarkeit von Empfehlungen wird durch die Visualisierung anhand von Capabilities, die Nennung des Grundes des Auslösens von Triggern, sowie die Herkunft von Pattern-Instanzen erreicht. Dies geht weit über den im Mashup-Bereich üblichen Rahmen hinaus. Wie gefordert werden Pattern-Instanzen automatisch in Anwendungen überführt, wobei ein dem OMELETTE-Ansatz vergleichbares »Einweben« stattfindet. Schließlich wird implizites und explizites Feedback gesammelt. Zum Einen in Form von Einträgen im Nutzerprofil gemäß dem Feedbackmetamodell, zum Anderen als Pattern-Instanzen.

7 Methoden zur Nutzerführung

Eine überblicksartige Vorstellung wesentlicher Werkzeuge und deren Einordnung in die Live-Sophistication beinhaltet bereits Abschnitt 4.1.3. Ziel von Kapitel 7 ist die detaillierte Konzeption ausgewählter EUD-Werkzeuge.

7.1 Der Startbildschirm als zentraler Einstiegspunkt

Initial präsentiert eine EUD-Kompositionsplattform für CWA dem Nutzer einen **Startbildschirm**, der einen personalisierten Einstiegspunkt in die *Live-Sophistication* bietet. Somit stellt er das wesentliche Instrument zur Erfüllung von Anforderung 5 dar.

Abbildung 7.1 zeigt den Entwurf des Startbildschirms. In der oben angebrachten Werkzeugleiste befindet sich ein Anwendungsmenü mit gängigen Verwaltungsfunktionen ①. Sie bietet ferner Informationen zum angemeldeten Nutzer und den Zugriff auf dessen Profil ② sowie eine Schaltfläche zum Abmelden ③.

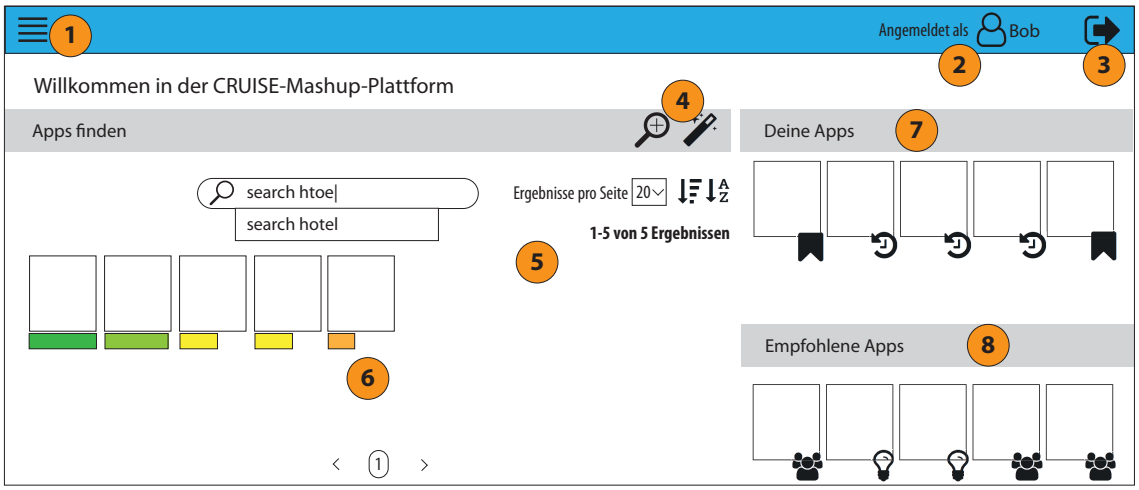


Abbildung 7.1: UI-Entwurf des Startbildschirms

Der Startbildschirm offeriert verschiedene Auflistungen von Anwendungen und Komponenten zum Schnellzugriff und zum Recherchieren. Zum Beispiel werden im Bereich

⑦ Komponenten und Anwendungen gewichtet nach der Historie des Nutzers gemäß dem *Frecency*-Ansatz aufgelistet, vergleiche Abschnitt 6.3.3. Die verschiedenen Nutzeraktivitäten, die dem Feedback zugrunde liegen, zum Beispiel *Setzen eines Lesezeichens*, *Auswählen*, *Nutzen* und *Erstellen*, werden durch Icons visualisiert. Zudem erfolgt die Darstellung von Komponenten und Anwendungen anhand verschiedener Kategorien wie dem Hersteller und der erbrachten Funktionalität. Weiterhin umfasst der Startbildschirm je nach den aktiven Empfehlungsstrategien eine Menge an Recommendation-Viewern ⑧. Dazu zählen zum Beispiel Empfehlungen anhand der Relevanz für den Nutzer und dessen aktueller Intention, anhand der semantischen Ähnlichkeit zu Items aus der Nutzerhistorie sowie anhand kollaborativen Filterns. Die Herkunft von Empfehlungen wird durch Icons kenntlich gemacht. Solche Auflistungen unterstützen ein Weiterblättern, etwa durch Karussells, und erlauben es, selektierte Anwendungen und Komponenten in der Live-View zu öffnen und somit zu verwenden. Weiterhin ist das Anzeigen sämtlicher zu dieser Kategorie gehöriger Items möglich. Dazu öffnet sich seitenfüllend der Facettenbrowser mit passend vorkonfigurierten Facetten und Filtern. Derartige Ansätze sind aus dem Bereich der Produktsuche bekannt und etabliert.

Zudem stehen gängige Suchmöglichkeiten im Bereich ⑤ zur Verfügung, zum Beispiel eine Stichwortsuche, ein Facettenbrowser sowie ein Wizard, der Gegenstand von Kapitel 7.4 ist. Gemäß dem Profil des aktiven Nutzers wird das präferierte Suchwerkzeug initial eingeblendet. Bei Bedarf sind die übrigen Optionen jederzeit zugänglich ④. Die Abbildung skizziert die Stichwortsuche mit gängigen Funktionen. Dazu gehört eine gekachelte Ergebnisdarstellung inklusive Relevanzindikatoren ⑥, deren Farbe und Länge entsprechend der Relevanz der Items angepasst werden. Wie in Kapitel 6.3.3 beschrieben, wird die Relevanz je nach Empfehlungsstrategie aus dem Übereinstimmungsgrad eines Items mit der Anfrage und weiterer kontextspezifischer Kriterien gebildet.

Nach Auswahl eines Kompositionsfragments in einem der beschriebenen Bereiche des Startbildschirms wird es von der Laufzeitumgebung instantiiert und dem Nutzer als eine CWA in der Live-View bereitgestellt. Letztere wird im nächsten Abschnitt spezifiziert.

7.2 Live-View

Die *Live-View* erlaubt es Nicht-Programmierern, Mashup-Anwendungen zu nutzen. Wie bei zahlreichen Mashup-Plattformen, vergleiche Kapitel 3.1, zeigt sie dazu im Anwendungsbereich ⑨ das UI der aktuellen Komposition gemäß deren MCM an, siehe Abbildung 7.2. Sämtliche UI-Komponenten werden entsprechend der aktuellen Ansicht und dem zugehörigen Layout visualisiert. Icons repräsentieren Nicht-UI-Komponenten in Bereich ⑫. Alle Komponenten besitzen einen vom MRE bereitgestellten Kopfbereich mit weiteren Werkzeugen. Mit diesen kann die Komponente entfernt sowie vergrößert werden und es sind Empfehlungen zur Komponente abrufbar. Eine Werkzeugleiste zeigt den Namen ② des Mashups, ein Menü mit Verwaltungsoptionen, die Möglichkeit in der Historie der Kompositionsschritte zu navigieren (Aktion rückgängig und wiederherstellen) sowie den Zugang zum Startbildschirm ①. Weiterhin bietet die Live-View Zugriff auf sämtliche Werkzeuge der Live-Sophistication, siehe Kapitel 4.1.3.

Der Nutzer kann per Stichwortsuche ③ sowie geführt per Wizard ④, siehe Kapitel 7.4, geeignete Kompositionen und Komponenten recherchieren. Erklärungstechniken unterstützen Nutzer beim Verständnis der Anwendungsfunktionalität und werden in Kapitel 7.5 genauer vorgestellt. Dazu gehört der Inspektionsmodus ⑤ zum Inspizieren

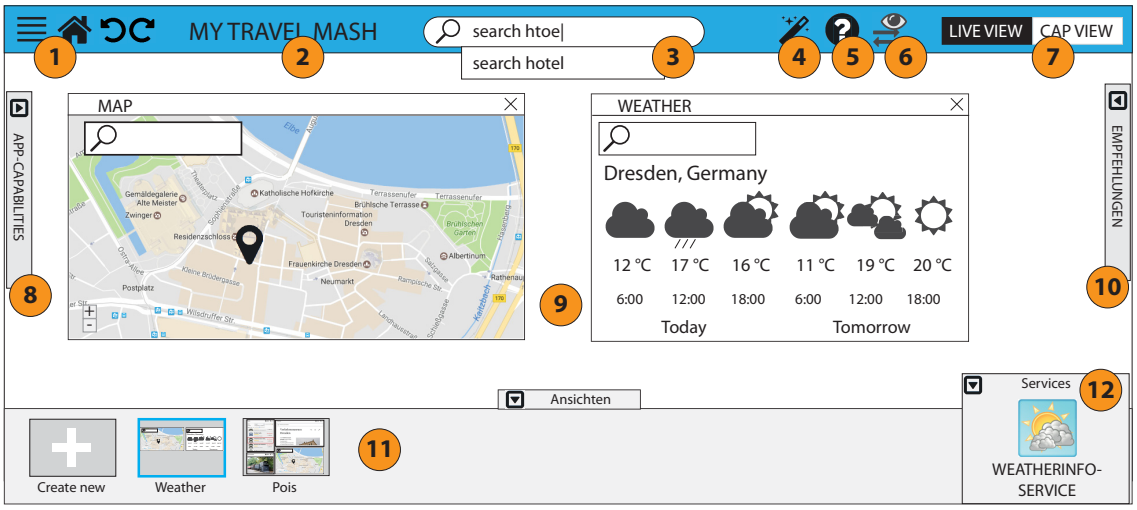


Abbildung 7.2: UI-Entwurf der Live-View

von Capabilities direkt im UI von Komponenten. Der Tutorialmodus liefert Bedienungsanleitungen für Capabilities und der Awareness-Modus ⑥ macht Nutzer auf Datentransfer zur Laufzeit aufmerksam. Ein Capability-Panel ⑧ dient der Exploration von Capabilities der Anwendung. Das Empfehlungsmenü ⑩ ist ein Recommendation-Viewer zur Anzeige von empfohlenen Kompositionsschritten, vergleiche Kapitel 6.3.4. Details dazu beinhaltet Abschnitt 7.5.3. Zudem können Nutzer in Anwendungsansichten navigieren. Dazu visualisiert ein separater, ein- und ausklappbarer Bereich ⑪ eine Vorschau pro Ansicht gemäß dem *Screenflow-Model*, siehe Kapitel 2.1.2. Standardmäßig werden Komponenten, die per Suchfeld, Wizard oder Empfehlungsmenü hinzugefügt werden, in der aktiven Ansicht platziert. Der *Screenflow-Editor* dient dem Erstellen neuer und dem Editieren existierender Ansichten. Er erlaubt zudem, Komponenten den Ansichten zuzuordnen und Transitionen zwischen letzteren zu definieren. Um Anforderung 2 gerecht zu werden, erfolgt das Erstellen von Transitionen auf Ebene von Capabilities. Hierzu bilden Nutzer, wie in der CapView, in PEUDOM und in NaturalMash, Wenn-Dann-Konstrukte, beispielsweise *Wenn ein Ort in der Karte ausgewählt wurde, wechsle in die Ansicht »Pois«*. Die Capability-View kann via ⑦ eingeblendet werden. Hierbei handelt es sich um eine den Anwendungsbereich ⑨ überlagernde Ansicht, die das Komponieren von Komponenten abstrahiert zur Verknüpfung von Capabilities.

7.3 Capability-View

Typische Kompositionswerkzeuge wie sie in vorhandenen Ansätzen aus Kapitel 3.1 auftauchen, stellen ein hohes Maß an Kompositionsdetails dar, sodass Nutzer technische Konzepte, Datenstrukturen et cetera verstehen müssen. Inspiriert durch Ansätze wie den Aufgabeneditor von DEMISA, vergleiche Kapitel 3.3, bietet die **Capability-View** (kurz CapView) hingegen eine aufgabenorientierte Abstraktion des Kompositionsmodells. Dazu dienen Capabilities von Komponenten als verknüpfbare Elemente und werden anhand natürlichsprachlicher Beschriftungen im Sinne von Fachbegriffen visualisiert. Dadurch werden Nicht-Programmierer unterstützt, wenn sie die Funktionsweise einzelner Kompo-

nenten oder eines Mashups nachvollziehen wollen und wenn sie die Funktionalität eines Mashups an ihre spezifischen Bedürfnisse anpassen oder erweitern möchten, zum Beispiel auf Basis von Empfehlungen. Die Ausführungen in diesem Abschnitt entsprechen einem überarbeiteten Stand des veröffentlichten Entwurfs [RBM13].

Die CapView ist eine überlagerte Ansicht auf ein in Ausführung und Entwicklung befindliches Mashup mit dem Schwerpunkt auf dem funktionalen Zusammenspiel von Komponenten. Das Komponentenmodell, die Mediationstechniken sowie das Capability-Metamodell bieten die Basis zur Umsetzung der grundlegenden Idee der CapView. Diese besteht darin, die Komposition von Komponenten auf die visuelle Verknüpfung von Capabilities zu abstrahieren, um Anforderung 6 gerecht zu werden. Technische Eigenschaften von Komponenten, wie Operationen und Events, werden nicht direkt angezeigt. Stattdessen werden Capabilities, die Aufgaben repräsentieren, die eine Komponente erledigen kann, visualisiert und sind interaktiv nutzbar. Damit Nutzer den Zusammenhang zwischen Live-View und CapView leichter erkennen, überlagern Capabilities das UI der zugehörigen Komponente im Anwendungsbereich der Live-View, siehe Kapitel 7.2.

Grundlegende Annahme ist, dass eine CWA und deren Komponenten zur Erledigung von Aufgaben dienen können. Letztere benötigen dazu Eingaben und produzieren Ausgaben, die von anderen Aufgaben bereitgestellt beziehungsweise konsumiert werden können. Capabilities modellieren dies weitgehend, werden jedoch mit den optionalen Mengen $P_{in|out}$ ergänzt. Diese beinhalten die Parameter der zur Capability gehörigen Operationen (P_{in}) oder Events (P_{out}). Capabilities gelten als *verknüpfbar*, falls die zugehörigen P_{out} und P_{in} semantisch kompatibel sind und somit ein semantischer Konnektor, siehe Kapitel 5.2, definiert werden kann. Des Weiteren stellt die CapView die Komponenten-Properties dar. Obgleich dies die Aufgabenorientierung aufweicht, wird angenommen, dass es für Nicht-Programmierer intuitiv ist, dass Objekte Eigenschaften besitzen.

Abbildung 7.3 illustriert in Bereich ① die überlagerte CapView, die Capabilities ③ und Properties ② von Komponenten auflistet und etablierte Verbindungen ④ präsentiert. Am rechten Bildrand befindet sich ein Empfehlungsmenü ⑦, das Patterns mit noch nicht im Mashup befindlichen Komponenten vorschlägt. Wie anhand von Farbe und Beschriftungen erkennbar ist, werden diese Patterns konsistent als Capabilities dargestellt.

7.3.1 Interaktive Exploration von Capabilities

Visuelle *Repräsentationen* von Capabilities und Properties sind der zentrale Bestandteil der CapView. Sie sind pro UI-Komponente gruppiert, überlagern diese und können zur Platzersparnis ein- und bei Bedarf wieder ausgeklappt werden. Zur Verknüpfung besitzt eine Repräsentation ein- und ausgehende *Ports*, insofern die Menge P_{in} respektive P_{out} nicht leer ist. Capability-Repräsentationen kapseln ganze Intra-Komponenten-Ketten von Capabilities, vergleiche Kapitel 5.3, wobei lediglich die resultierende komposite Capability auftaucht. Zum Beispiel wird eine Kette `Add Location` (Capability einer Operation) → `Display Weather` zu `Display Weather` reduziert, um Einfachheit und Übersichtlichkeit zu fördern. Komponenten können mehrere ähnliche Capabilities aufweisen, die sich lediglich in den zugrundeliegenden Schnittstellenelementen und deren Parametersignatur unterscheiden. In diesem Fall findet eine *Gruppierung* statt. Dabei werden alle Repräsentationen von Capabilities mit gleicher Activity und Entity zusammengefasst, um Implementierungsdetails zu verbergen. Zum Beispiel offeriert die Wetter-Komponente aus Abbildung 7.3 zwei Operationen mit der Capability `Display Weather`. Eine benö-

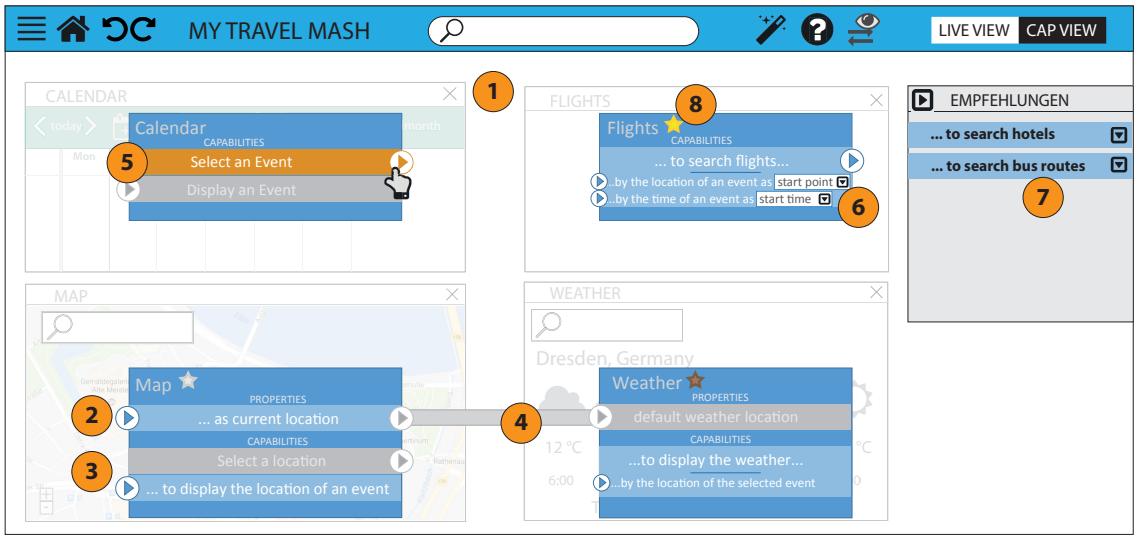


Abbildung 7.3: Beispielhafter Überblick der Capability-View

tigt einen Ort, die andere zusätzlich ein Datum als Eingabe. Verschiedene Ausgaben in Form von Events einer Capability-Repräsentation einer Gruppe werden vor Nutzern verborgen und transparent gehandhabt. Zum Beispiel werden die Events bei der Berechnung von verknüpfbaren Capabilities separat untersucht und das passende Event wird entsprechend der Pattern-Instanz bei dem Erzeugen eines Kommunikationskanals gewählt. Durch die genannten Konzepte werden technische Schnittstellendetails effektiv von Nicht-Programmierern ferngehalten, was vorteilhaft hinsichtlich Anforderung 3 ist.

Zudem weist jede Repräsentation eine *natürlichsprachliche Beschriftung* auf, die gemäß dem im nächsten Abschnitt vorgestellten Regelwerk gebildet wird. Während im ursprünglichen Entwurf eine Farbkodierung vorgesehen war, entfällt dieser aufgrund der Evaluationsergebnisse (Kapitel 8.5.1) im finalen Konzept. Lediglich Properties und Capabilities werden optisch separiert und zusammengefasst. Repräsentationen von *Properties* sind ebenfalls mit Ausgabeports ausgestattet. Ein Eingabeport erscheint nur, falls die Property nicht als *nur lesend* in der SMCDL deklariert ist.

Die vom Nutzer selektierte Repräsentation ⑤ wird im weiteren Verlauf als r_0 bezeichnet. Die Menge L beinhaltet alle direkt mit r_0 verknüpfbaren Repräsentationen. $S \subseteq L$ sind dabei solche, die Eingaben liefern können, und $T \subseteq L$ solche, die Ausgaben von r_0 konsumieren können. Zur Vermeidung von Zyklen wird angenommen, dass $S \cap L = \emptyset$ gilt. Reflexives Verknüpfen von r_0 wird ebenfalls ausgeschlossen. Nach der Auswahl von r_0 folgt ein Hervorheben sämtlicher $r_i \in L$. Weiterhin werden deren Beschriftungen regelbasiert adaptiert, um Kausalsätze zu bilden und somit die funktionalen Zusammenhänge für Nutzer verständlich anzuzeigen. Auch die angepassten Beschriftungen werden zeitweise hervorgehoben, um Nutzern diese Änderungen zu verdeutlichen. Direkt und transitiv an r_0 anliegende Kanäle werden betont. Alle übrigen Kanäle werden zur Steigerung der Übersichtlichkeit ausgegraut. Das Hervorheben verknüpfbarer Repräsentationen und deren Ports entspricht der nahtlos integrierten Visualisierung von Empfehlungen, das heißt, die CapView hält einen Recommendation-Viewer bereit. Medaillen geben Aufschluss über die am höchsten gewichteten Empfehlungen ⑧.

Sobald ein Nutzer die Auswahl von r_0 oder eines dessen Ports aufhebt, wird jegliche Hervorhebung zurückgesetzt und die initialen Beschriftungen angezeigt.

7.3.2 Kontextsensitive Erzeugung von Beschriftungen

Die automatische Erstellung von Beschriftungen ist ein Kernbestandteil der CapView und wird vom *Label-Generator* übernommen. Dieser baut auf einer Menge generischer Regeln auf und bezieht dabei das Wissen aus Ontologien, aus einem Wörterbuch sowie aus Mediationstechniken ein. Damit wird erreicht, dass mit Nutzern im Einklang mit Anforderung 2 auf fachlicher Ebene kommuniziert wird. Die folgenden Ausführungen bedienen sich dem Englischen als Sprache der erzeugten Beschriftungen. Das zugrundeliegende Regelwerk lässt sich jedoch auf andere Sprachen übertragen, während die spezifische Grammatik unter Umständen anzupassen wäre.

Die Notation des Regelwerks zur Erzeugung der Beschriftungen basiert auf mehreren Funktionen und Indizes. $dLabel()$ und $aLabel()$ geben eine menschenlesbare Bezeichnung für ein Ontologiekonzept zurück, das als Property-Typ, Entity ($dLabel()$) oder Activity ($aLabel()$) annotiert ist. Dabei wird entweder der Name des Konzepts aus dessen URI extrahiert oder das `rdfs:label` genutzt. Funktion $art()$ fügt einen passenden Artikel ein. Index pp bezeichnet die Partizip-Form, die aus einem Wörterbuch ausgelesen wird. Index $norm$ bezeichnet das »normalisierte« Konzept. Dabei handelt es sich um den Wertebereich im Fall einer OWL-Object-Property und ansonsten das Konzept selbst. $map_{P_1 \to P_2}$ beschreibt die Mapping-Definition für zwei Parametersignaturen P_1 und P_2. $Type_{prop}$ bezeichnet den semantischen Typ einer Property und Val_{prop} den Wert der Property.

Der Generierungsprozess unterscheidet zwei wesentliche Fälle. Zum einen kommt eine **Basiskonfiguration** für Beschriftungen zum Einsatz, falls keine Selektion vorliegt. Für Properties wird neben dem Namen des Typkonzepts eine Kurzform des aktuellen Wertes verwendet, um das Verständnis für Nutzer zu erleichtern, zum Beispiel `center location (Dresden)`. Die Beschriftung folgt stets dem Aufbau:

$$dLabel(Type_{prop})\ dLabel(Val_{prop})$$

Für eine Capability setzt sich die Beschriftung im Basisfall aus einer Phrase aus Konzeptnamen von Activity und Entity zusammen, zum Beispiel `search a route`. Nachfolgend werden Capabilities Tupel <*A, E, iR*> notiert, mit Activity A, Entity E und der booleschen Aussage iR, ob es sich um eine UI-Capability handelt.

$$aLabel(A)\ art()\ dLabel(E)$$

Zum anderen erfolgt eine **kontextsensitive Anpassung von Beschriftungen** entsprechend der aktiven Selektion des Nutzers, um Ursache-Wirkungs-Prinzipien durch kurze Satzkonstrukte zu verdeutlichen. Wie nachfolgend erläutert wird, basiert der Ansatz auf einem generischen Regelwerk, benötigt L von r_0 als Eingabe und bestimmt die Beschriftungen für jedes $r_k \in L$, wobei $r_j \in S$ und $r_i \in T$ jeweils speziell behandelt werden. Angefügte oder vorangestellte Punkte stellen die Leserichtung klar.

Eine Property ist fokussiert

Falls es sich bei r_0 um eine Property handelt, wird unterschieden, ob $r_k \in L$ entweder eine Property oder eine Capability ist.

Falls $r_j \in S$ oder $r_i \in T$ eine Property repräsentieren, hängt es davon ab, ob r_0 Empfänger oder Sender der Verknüpfung ist. Im ersten Fall wird die Beschriftung von $r_j \in S$ gemäß dem Schema

$$\texttt{Use } dLabel(Val^j_{prop}|Type^j_{prop}) \texttt{ as } \ldots$$

aufgebaut. Andernfalls kommt eine leicht modifizierte Regel für $r_i \in T$ zum Einsatz. Zur Veranschaulichung sei auf nachfolgende Tabelle verwiesen.

$$\texttt{Use } dLabel(Val^0_{prop}|Type^0_{prop}) \texttt{ as } dLabel(Type^i_{prop})$$

$r_j \in S$	r_0	$r_i \in T$
$Type_{prop} = Location$	$Type_{prop} = hasCenter$	$Type_{prop} = Location$
`Use location (Dresden) as ...`	`Center`	`Use center as location`

Die selektierte Property r_0 kann als Eingabe für eine Capability $r_i \in T$ mit $r_i = <<A^i, E^i, iR^i>, P^i_{in}, P^i_{out}>$ dienen, wobei $map_{Type_{prop} \rightarrow P^i_{in}}$ injektiv ist und $Type_{param}$ den semantischen Typ des einzigen, passenden Operationsparameters kennzeichnet.

Falls die »normalisierte« Entity und $Type_{param}$ nicht semantisch kompatibel sind und falls $Type_{param,norm}$ gleich $Type_{prop,norm}$ oder ein Superkonzept dessen ist, entspricht die Beschriftung folgender Struktur:

$$aLabel(A^i)\ art()\ dLabel(E^i) \texttt{ using } art()\ dLabel(Type_{prop})$$

r_0	$r_i \in T$
$Type_{prop} = hasCurrentLocation$	<<Search, Hotel,false >, {Location, Time} >
`Current location`	`Search a hotel using the current location`

Ein weiterer Fall ist, dass $Type_{param}$ vom Konzept $Type_{prop,norm}$ aggregiert wird, sodass ein Semantic Split angewendet werden kann (**SplitRule**).

$$aLabel(A^i)\ art()\ dLabel(E^i) \texttt{ using } art()\ dLabel(Type_{param}) \texttt{ of } art()\ dLabel(Type_{prop})$$

r_0	$r_i \in T$
$Type_{prop} = Event$	<<Display, Hotel, true >, {Location, Time} >
`Event`	`Search a hotel using the location of the event`

Im Gegensatz dazu kommt eine kürzere Regel zum Einsatz, falls Entity und Parametertyp semantisch kompatibel sind, um kompaktere Beschriftungen zu generieren (**CompRule**). In diesem Fall wird bei Bedarf ebenfalls die *SplitRule* angewendet.

$$aLabel(A^i)\ art()\ dLabel(Type_{prop})$$

r_0	$r_i \in T$
$Type_{prop} = hasCenter$	<<Display, Location, true >, Location >
`Center location`	`Display the center location`
$Type_{prop} = Event$	<<Display, Location, true >, Location >
`Event`	`Display the location of the event`

Falls die von r_i repräsentierte Capability mehrere Parameter offeriert ($|P^i_{in}| > 1$), kann eine Vielzahl valider Mapping-Definitionen zwischen Property und Parametern existieren. In dieser Situation werden die Optionen als Suffix deklariert (**SuffixRule**):

$$\texttt{as}\ dLabel(Type_{param})$$

r_0	$r_i \in T$
$Type_{prop} = hasCenter$	<<Search, Route, false >, {hasStart, hasDest} >
`Center`	`Search a route using the center as start`
	`Search a route using the center as destination`

Eine Property r_0 kann die Ausgabe einer Capability $r_j \in S$ mit $r_j = << A^j, E^j, iR^j>, P^j_{in}, P^j_{out} >$ konsumieren, wobei $map_{P^j_{out} \rightarrow Type_{prop}}$ injektiv ist und $Type_{param}$ das Typkonzept des Parameters bezeichnet, der mit der Property verknüpft ist.

$$\texttt{Use}\ art()\ aLabel(A^j)_{pp}\ dLabel(E^j)\ \texttt{as}\ art()\ \ldots$$

$$\texttt{Use}\ art()\ dLabel(Type_{prop})\ \texttt{of}\ art()\ aLabel(A^j)_{pp}\ dLabel(E^j)\ \texttt{as}\ art()\ \ldots$$

$r_j \in S$	r_0
<<Select, Location, true>, Location >	$Type_{prop} = hasCenter$
`Use the selected location as the ...`	`center location`
<<Select, Event, true>, Event >	$Type_{prop} = hasCenter$
`Use the location of the selected event as the ...`	`center location`

Eine Capability ist selektiert

Sofern es sich bei $r_j \in S$ oder $r_i \in T$ um eine Property handelt, gelten die oben eingeführten Regeln. Deshalb bedarf es nachfolgend lediglich einer genaueren Betrachtung des Falls, dass $r_j \in S$ oder $r_i \in T$ Capabilities sind.

Wie im vorherigen Abschnitt wird zunächst geprüft, ob die Bedingungen für die *CompRule* zutreffen. Dazu müssen beide Entities identisch oder äquivalent sein oder zueinander in Subklassenbeziehung stehen. Auch für die aufeinander abgebildeten Parametertypen wird dies überprüft. Sind die Bedingungen erfüllt, resultiert folgender Aufbau:

$$\ldots\texttt{to}\ aLabel(A^i)\ art()\ aLabel(A^0)_{pp}\ dLabel(E^0)$$

r_0	$r_i \in T$
<<Select, Location, true>, Location >	<<Display, Location, true>, Location >
`Select a location`	`...to display the selected location`

Sofern *CompRule* nicht anwendbar ist, kommt das unten formalisierte Schema zum Einsatz. *SplitRule* und *SuffixRule* werden ebenfalls getestet und bei Bedarf angewendet. Abhängig von der Mapping-Definition $map_{P_1 \rightarrow P_2}$ können prinzipiell n by- und k as-Teile (letztere werden weggelassen, falls $k = 1$) in der resultierenden Beschriftung auftreten. Hierbei entspricht n der Anzahl der passenden Parameter in P^0_{out} und k der in P^i_{in}.

$$\ldots\texttt{to}\ aLabel(A^i)\ art()\ dLabel(E^i)\ \Bigg(\ \texttt{by}\ art()\ \Big[\ dLabel(Type^{0,n}_{param})\ \texttt{of}\ art()\ \Big]$$
$$aLabel(A^0)_{pp}\ dLabel(E^0)\ \Big[\ \texttt{as}\ dLabel(Type^{i,k}_{param})\ \Big]_k\ \Bigg)_n$$

r_0	$r_i \in T$
<<Select, Location, true>, Location >	<<Search, Hotel, false >, Location >
`Select a location`	`...to search a hotel by the selected location`
<<Select, Event, true>, Event >	<<Search, Hotel, false>, Location >
`Select an event`	`...to search a hotel by the location of the selected event`
<<Select, Location, true>, Location >	<<Search, Route, false>, {hasStart, hasDest} >
`Select a location`	`...to search a route by the selected location as start`

Eine ähnliche Behandlung erfahren Capability-Repräsentationen, die Eingaben für r_0 bereitstellen. Allerdings bezeichnen n und k diesmal die Anzahl passender Parameter in $P_{out,j}$ bzw. in $P_{in,0}$.

$$\left(\ldots \texttt{by}\ art() \left[dLabel(Type^{j,n}_{param})\ \texttt{of}\ art() \right] aLabel(A^j)_{pp}\ dLabel(E^j) \left[\texttt{as}\ dLabel(Type^{0,k}_{param}) \right]_k \right)_n$$

$r_j \in S$	r_0
<<Select, Location, true>, Location >	<<Display, Location, true>, Location >
`...by a selected location`	`Display location`
<<Select, Event, true>, Location >	<<Search, Hotel, false>, Location >
`...by the location of a selected event`	`Search a hotel`

Dieses generische Regelwerk erlaubt es, aus semantischem Wissen über Komponentenannotationen verständliche Beschriftungen und kurze Satzkonstrukte zu erzeugen, um Funktionalitäten von Komponenten und funktionale Zusammenhänge zwischen diesen zu erläutern. Zwar verursacht dieser Ansatz stärkere Abhängigkeiten von Annotationen und die resultierenden Texte mögen weniger präzise und umfangreich ausfallen als manuell bereitgestellte Dokumentationen. Allerdings besticht das Konzept durch Generizität, wodurch unvorhergesehene Konstellationen von Komponenten abgedeckt werden können. Dies ist angesichts der Eigenheiten von CWA besonders wichtig. Zudem verringert sich die Abhängigkeit von manuell erstellten Dokumentationen für Komponenten und Empfehlungen, da semantisches Wissen wiederverwendet wird.

7.3.3 Verknüpfen von Capabilities

Existiert eine vom Nutzer ausgewählte Repräsentation r_0, kann eine neue Verbindung mit hervorgehobenen, verknüpfbaren Repräsentationen vorgenommen werden. Zudem können bestehende Verbindungen verändert sowie gelöscht werden. Das Herstellen eines Kanals kann daraufhin durch Auswahl und somit durch Aktivieren sowohl eines Eingabe- als auch eines Ausgabeports eingeleitet werden. Falls r_0 nur einen Port besitzt, wird dieser automatisch aktiviert. Entsprechend dem aktivierten Port werden die möglichen Verknüpfungen auf S bzw. T von r_0 eingeschränkt. Somit können Nutzer ausschließlich kompatible Komponentenschnittstellen verbinden und fehlerhafte Zustände der CWA

vermieden werden. Verbindungen werden via Drag-and-drop oder aufeinanderfolgendes Anklicken der Ports erstellt. Dabei sind beide Richtungen (Quelle $\leftrightarrow$ Ziel) möglich.

Falls Gruppierung vorgenommen wurde und mehrere Parameter oder mehrere Optionen für deren Abbildung vorliegen, wird standardmäßig die Belegung entsprechend der höchstbewerteten empfohlenen Pattern-Instanz angewendet. Entspricht diese Zuordnung nicht den Wünschen des Nutzers, obliegt es ihm diese zu adaptieren. Dazu werden auf Basis der zuvor genannten Regeln diverse »by«- und »as«-Abschnitte gemäß der auftretenden Optionen generiert. Diese werden als Bestandteil der Repräsentation visualisiert, siehe ⑥ in Abbildung 7.3. Aus Gründen der Übersichtlichkeit und um den örtlichen Zusammenhang von Repräsentation und Komponente beizubehalten, enthält die Beschriftung in einem zugeklappten Zustand zunächst nur essentielle Informationen. Nach Aufklappen werden die vollständigen Details und Auswahlmöglichkeiten offengelegt. Im Beispiel ist zu sehen, dass der Ausgabeparameter vom Typ `Event` nicht komplett zur Flugsuche genutzt wird, sondern lediglich die Zeit und der Ort des `Events`. Dies wird durch einzelne Eingabeports in Abbildung 7.3 gekennzeichnet. Zudem benötigt die Flugsuche mehrere Eingabeparameter (Zeitpunkt, Start-, Zielort), sodass es mehrere Möglichkeiten gibt, den Ort des Events abzubilden, was das Auswahlfeld mit zwei Einträgen in der Beschriftung des ersten Eingabeports andeutet. Falls Gruppierung vorgenommen wurde, ist es Aufgabe der Laufzeitumgebung auf die passenden Operationen beziehungsweise Events zu schließen. Die Laufzeitumgebung trägt schließlich Sorge dafür, dass alle notwendigen Details gemäß dem Kompositionsmodell korrekt instanziiert werden, sodass die Verbindung auf Implementierungsebene und somit in der Live-View funktionstüchtig ist. Funktionstüchtig bedeutet, dass sämtliche Ein- bzw. Ausgabeparameter des aktiven Ports von r_0 belegt sind. Dies gewährleistet das Empfehlungssystem, da Coupling- und Complex-Patterns stets komplette semantische Konnektoren bereitstellen.

Im Mittelpunkt des nachfolgenden Kapitels steht, wie Nicht-UI-Komponenten in der Capabililty-View behandelt werden.

7.3.4 Handhabung von Komponenten ohne UI

Als eine wichtige Erkenntnis der Nutzerstudie zum ursprünglichen Konzept der CapView [RBM13] zeigten sich Verständnisprobleme bezüglich Nicht-UI-Komponenten. Diese wurden ursprünglich analog zu UI-Komponenten durch eine gedachte Positionierung im Randbereich inklusive Auflistung von Capabilities behandelt. Der finale Entwurf sieht folgenden in Abbildung 7.4 veranschaulichten Ansatz vor, um dem zu begegnen.

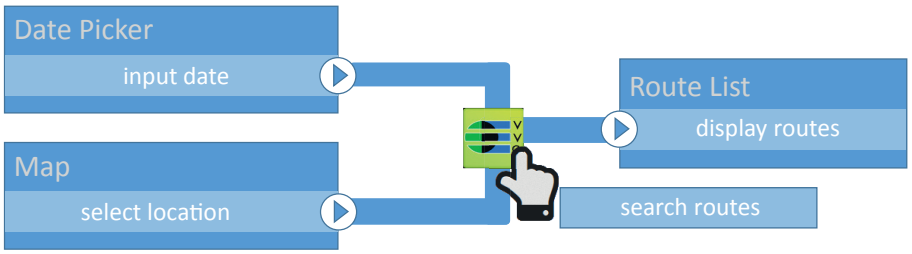


Abbildung 7.4: Beispiel zur Handhabung von Nicht-UI-Komponenten in der CapView

Nicht-UI-Komponenten werden nicht als Komponentenrepräsentation aufgeführt, sondern lediglich durch Icons symbolisiert. Diese Icons werden an die semantischen Konnek-

toren angeheftet, an denen die Komponenten beteiligt sind. Die maßgebliche Capability der Nicht-UI-Komponente wird optisch dem semantischen Konnektor zugeordnet und angezeigt, wenn sich der Mauszeiger über dem Icon befindet. Zur Bestimmung dieser maßgeblichen Capability wird davon ausgegangen, dass jede Nicht-UI-Komponente Capability-Ketten aufweist, die mindestens eine Capability mit einer Activity, die von `Manipulate` erbt, beinhaltet. Diese wird visualisiert, während Capabilities mit einer Activity vom Typ `Input` oder `Output` als Teil der Kette nicht dargestellt werden. Beim Anklicken des Icons erfolgt ein Fixieren der Capability und es bestehen die zuvor definierten Möglichkeit zur Konfiguration von Parameterabbildungen. Falls mehrere Nicht-UI-Komponenten in einer Reihe auftreten, wird das Prinzip pro Komponente angewendet und entsprechend viele Icons gezeichnet.

Zwar wären alternative Konzepte möglich gewesen, wie ein Zuordnen der Icons oder Capabilities zu UI-Komponenten. Der Ansatz wurde jedoch gewählt, da die Vorteile überwiegen. Eine Zuordnung zu UI-Komponenten ist nicht notwendig. Eine solche wäre nicht allgemeingültig festlegbar und je nach mentalem Modell des Nutzers als verwirrend einzuschätzen. Die CapView wird im Vergleich zum ursprünglichen Konzept vereinfacht, da weniger Repräsentationen auftauchen und Services nicht als Komponenten verdeutlicht werden. Weiterhin eignet sich die Lösung für kaskadierte Nicht-UI-Komponenten. Als Einschränkung ist die limitierte Konfigurierbarkeit zu nennen, da zwar Parameterabbildungen, jedoch kein direktes Verknüpfen von Nicht-UI-Komponenten unterstützt wird. Vielmehr dienen Complex-Patterns, siehe Kapitel 6.2.2, dem automatischen Hinzufügen und Exchangeability-Patterns dem Austausch von Nicht-UI-Komponenten. Allerdings scheint dies angesichts der Zielgruppe ein akzeptabler Kompromiss.

Insgesamt ermächtigt die CapView Nicht-Programmierer dazu, die Funktionalitäten von Mashups und von deren Komponenten zu explorieren und insbesondere die Komposition zu manipulieren, indem Capabilities visuell miteinander verknüpft werden.

7.4 Wizard zur Eingabe funktionaler Anforderungen

Wie in Anforderung 2 festgehalten, bedarf es an Werkzeugen, mit denen Nicht-Programmierer einer Kompositionsplattform ihre funktionalen und qualitativen Ziele und Anforderungen mitteilen können. Live-Sophistication sieht hierzu mehrere komplementäre Ansätze vor, wie die Stichwortsuche auf dem Startbildschirm, Recommendation-Viewer im Live-View sowie die Capability-View. Als ein geführter Ansatz existiert zudem ein Wizard, der im Fokus dieses Abschnitts steht und Nutzern bei der Spezifikation ihrer Anforderung und bei dem Auffinden geeigneter Kompositionsfragmente assistiert. Als Entwurfsgrundsätze gelten Einfachheit, Aufgabenangemessenheit und andere Kriterien zur Gebrauchstauglichkeit. Zudem wird auf Generizität geachtet, sodass nicht nur CWA aus bestimmten Anwendungsdomänen erstellt werden können. Eine andere Option wäre ein anpassbares oder austauschbares Regelwerk, das für Anwendungsdomänen zugeschnittene Dialogabläufe und Fachwissen beschreibt. Allerdings wäre es dazu nötig, derartige Regelwerke bereitzustellen und situationsbezogen zu applizieren. Daher sieht das Konzept eine feste Dialogführung vor und bezieht Domänenwissen mit ein.

In jedem Schritt werden *Icons* dargestellt, die von Nutzern ausgewählt werden können. Jedes Icon besteht nicht nur aus einem Bild, das Emotionen wecken und Nutzer inspirieren soll, sondern auch aus einer Beschriftung, um eine eindeutige Identifizierung zu gewährleisten. Die aktuelle Anfrage, das heißt die Gesamtheit der selektierten Icons, wird

jederzeit deutlich. Dazu beherbergen die Überschriftsbereiche eines jeden Schritts eine Repräsentation der zugehörigen Icons, welche zwecks Platzersparnis auf ihre Beschriftung reduziert sind, siehe ④ in Abbildung 7.6. Die Repräsentation lässt es zu, Icons und somit die zugehörige Anforderung zu entfernen. Daneben können Anforderungen im jeweiligen Schritt abgewählt werden, indem das zugehörige Icon deselektiert wird.

Zur Anbindung an das Empfehlungssystem im Rahmen einer definierten Empfehlungsstrategie trägt der Wizard dafür Verantwortung, alle für die Empfehlungsmethode benötigten Parameter bereitzustellen und das notwendige Startevent des zugehörigen Triggers zu erzeugen. In diesem Zusammenhang verwaltet der Wizard eine Datenstruktur, welche die Anforderungen des Nutzers in den Kategorien *Themen*, *Capabilities*, *Freitext* und *Qualitätsanforderung* aufnimmt. Zwecks Erhalt und Darstellung der Empfehlungen entspricht der Wizard einem *Recommendation-Viewer*, vergleiche Kapitel 6.3.4.

Für das Konzept wurden drei generische Schritte entwickelt, die idealerweise sequentiell abgearbeitet werden. Allerdings sind sie allesamt optional, sodass ein hohes Maß an verschiedenen Herangehensweisen unterstützt wird. Durch textuelle Hinweise und optisches Hervorheben werden Nutzer über die Zielstellung jedes Schritts aufgeklärt und auf mögliche Abarbeitungsreihenfolgen hingewiesen.

Schritt 1 – Auswahl von Themen Zunächst können Nutzer ihre Vorstellungen eingrenzen, indem sie Themenbereiche auswählen, siehe ① in Abbildung 7.5. Welche Themen zur Verfügung stehen, ergibt sich aus den Domänen der Capabilities aller im System vorhandenen Kompositionsfragmente. Somit kann gewährleistet werden, dass stets Capabilities (in Schritt 2) und Ergebnisse vorliegen. Wählt ein Nutzer ein Themen-Icon aus, wird dieses optisch hervorgehoben und der Anfragerepräsentation ① hinzugefügt. Zusätzlich wird ein neuer Empfehlungskreislauf ausgelöst, wodurch schließlich die Ergebnisliste ③ sowie die im nachfolgenden Schritt verfügbaren Icons aktualisiert werden.

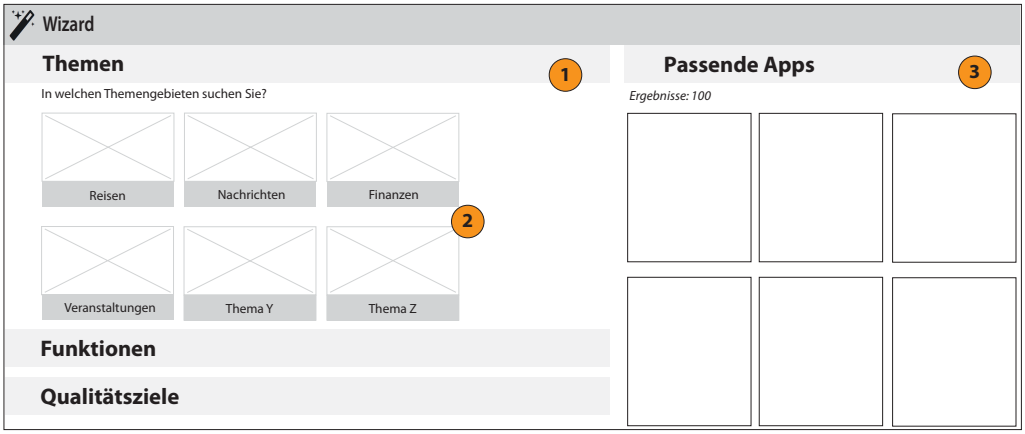


Abbildung 7.5: Wizard: Auswahl von Themengebieten (links), Ergebnisliste (rechts)

Schritt 2 – Auswahl gewünschter Funktionalitäten Im zweiten Schritt können Nutzer ausdrücken, welche Funktionalitäten sie benötigen. Dazu beinhaltet der Bereich eine Menge an Icons ⑥, die Capabilities und andere Metadaten von Komponenten und Anwendungen darstellen. Die Einträge werden initial gemäß der Themenauswahl gefiltert und nach Relevanz sortiert. Neben dieser Direktauswahl von Funktionalitäten

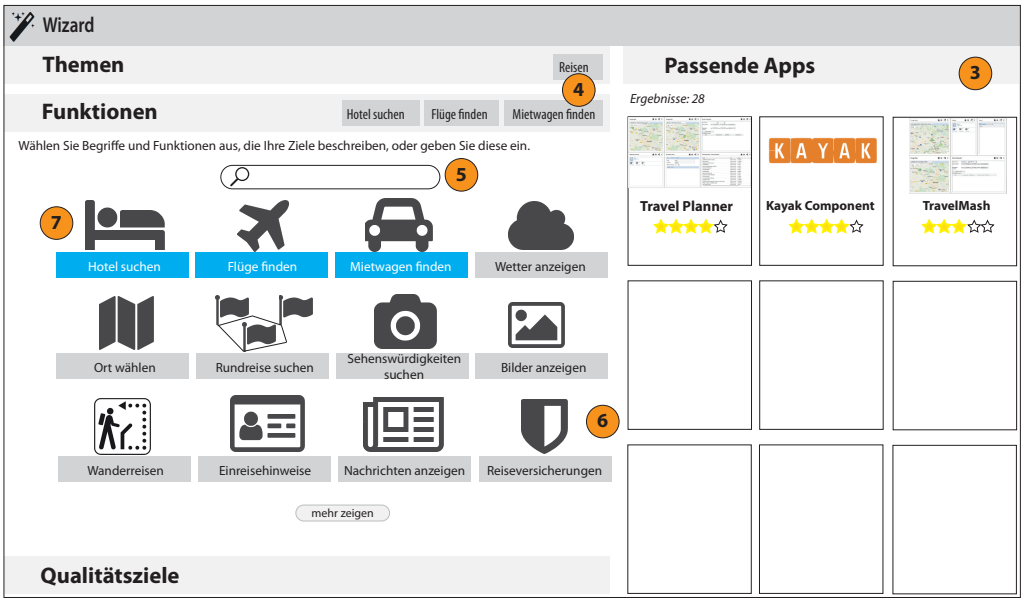


Abbildung 7.6: Wizard: Auswahl benötigter Funktionalitäten (links), Ergebnisliste (rechts)

können Nutzer Begriffe eingeben ⑤. Hierbei assistieren ihnen Vervollständigungen und Vorschläge. Der Wizard erkennt, ob es sich bei der Eingabe um eine Capability oder Freitext handelt und ordnet diese der entsprechenden Anforderungskategorie zu, sodass der Empfehlungsmanager die Anfrage richtig interpretiert. Bei Icons, die Capabilities repräsentieren, kommen die Konzepte aus Abschnitt 7.3 zur Generierung von Beschriftungen zum Einsatz. Somit kann mit Nutzern auf Ebene fachlicher Begriffe kommuniziert und Anforderung 2 Rechnung getragen werden. Erneut sind Icons selektierbar, erweitern daraufhin die Anfrage ④ und werden als erstes im Capability-Bereich angezeigt ⑦. In Abbildung 7.6 sind die drei blau hinterlegten Icons ausgewählt. Daraufhin erfolgt wiederum eine Neuberechnung der Ergebnisse, welche kachelartig visualisiert werden ③. Sie sind absteigend nach Übereinstimmung mit der Gesamtanfrage sortiert und werden anhand ihres Namens, eines Bilds und der durchschnittlichen Nutzerbewertung präsentiert. Sobald sich die Maus über einem Ergebnis befindet, zeigt der Wizard die Relevanz des Treffers an, indem die Icons in der Anfragerepräsentation den Übereinstimmungsgrad mit der jeweiligen Anforderung durch Farbkodierung kennzeichnen.

Nutzer werden bei der Anforderungsspezifikation durch Empfehlungen zu Funktionalitäten unterstützt. Diese helfen sowohl bei der Komposition (Bottom-up-Vorgehen) als auch der Dekomposition (top-down) von Funktionalitäten, zum Beispiel indem übergeordnete Capabilities, Spezialisierungen und Erweiterungen angeboten werden. Dabei identifiziert ein Algorithmus ausgehend von den Capabilities der jeweiligen Kandidatenmenge sowie der aktuellen Selektion und unter Einbeziehung des in Kapitel 5.4 vorgestellten Wissensgraphen, Capabilities, welche die aktuelle Anfrage näher eingrenzen oder verbreitern. Ersteres entfaltet seinen Wert beim Top-down-Vorgehen. Als Beispiel sei `Search Route` selektiert worden und es wird `Search CarRoute` vorgeschlagen, oder bei Selektion von `Search Hotel` und `Display Route` wird `Rent Car` angeboten. Verbreiternde Vorschläge sind insbesondere bei Bottom-up-Herangehensweisen hilfreich, zum Beispiel `Plan Trip` bei bisheriger Auswahl von `Search Hotel` und `Search POI`. Auch andere Metadaten wie Schlüsselworte von Komponenten werden in Betracht gezogen, indem die höchbewerteten

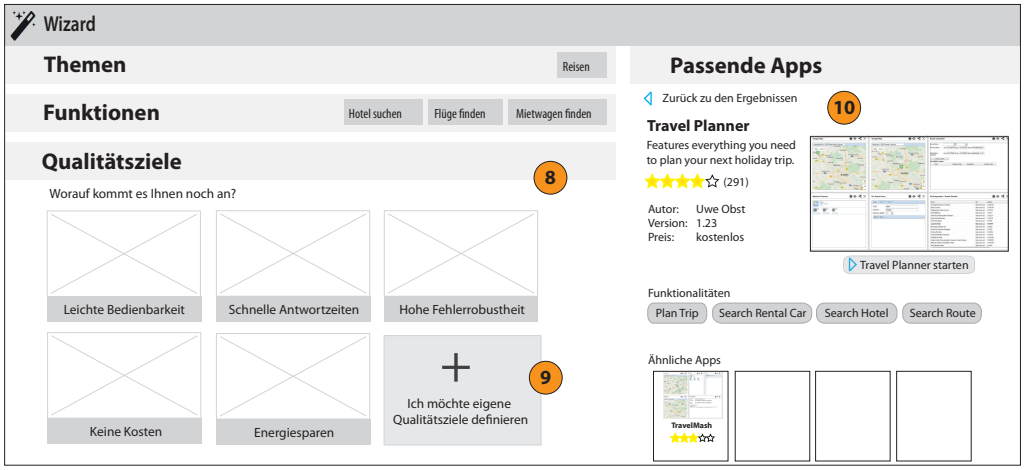


Abbildung 7.7: Wizard: Auswahl von Qualitätsanforderungen (links), Detailseite (rechts)

Kompositionsfragmente nach statistisch häufigen Einträgen untersucht werden. Empfehlungen werden nach Änderung der Anforderungsmenge in einem der Schritte berechnet und in Bereich ⑥, den selektierten Icons nachgeordnet, visualisiert. Auf diese Weise wird domänenspezifisches Wissen in den Wizard einbezogen.

Insgesamt sind notwendige Hilfsmittel vorhanden, um sowohl Bottom-up- als auch Top-down-Vorgehen bei der Anforderungsspezifikation zuzulassen. Im Bezug auf den Stand von Forschung und Technik in Kapitel 3.3 stellt dies ein Novum dar. Der Ansatz nutzt semantische Domänenmodelle und findet auf fachlicher Ebene statt. Dahingehend zeigen sich Ähnlichkeiten zum Aufgabeneditor von DEMISA, der allerdings stets Dekomposition erfordert und weniger stark in die Laufzeitumgebung integriert ist. Die Nutzung von Bildern ist von GetInspired! [Bot+14] inspiriert, das jedoch lediglich auf Taxonomien beruht und nur Dekompositon unterstützt. Auch UI-Konzepte aus icon-basierten Ansätzen sind in den Entwurf des Wizards eingeflossen.

Schritt 3 – Auswahl von Qualitätszielen Weiterhin können Qualitätsziele im dritten Schritt definiert werden, vergleiche Abbildung 7.7. Hinsichtlich des Modells und der Auswertung von Qualitätsanforderungen wird auf existierende Konzepte [Rüm+14] aufgesetzt. Im Wizard bekommen Nicht-Programmierer eine Auswahl vordefinierter Qualitätsanforderungen präsentiert ⑧. Die Qualitätsvorlieben aus dem Nutzerprofil reichern die Ansicht an, indem passende Anforderungen bereits selektiert werden. Daneben steht Nutzern die Möglichkeit offen, individuelle Qualitätsanforderungen zu definieren ⑨. Das erfolgt entsprechend der Konzepte aus [Rüm+14] und ist deshalb hier nicht Gegenstand genauerer Betrachtungen. Wie in den vorherigen Schritten, ändert sich die Ergebnismenge sobald die Icons vom Nutzer (de)selektiert werden. Das Betätigen einer Kachel in der Ergebnisliste startet die gewählte Anwendung respektive eine neue im Fall von Komponenten. Zudem besteht die Möglichkeit, eine Detailseite zu Kompositionsfragmenten anzeigen zu lassen. Abbildung 7.7 skizziert hierzu einen Entwurf. Hervorzuheben ist die Anzeige ähnlicher Kompositionsfragmente, zu deren Detailseite ebenfalls navigiert werden kann. Somit steht ein weiterer Ansatz zur explorativen Recherche bereit.

Im vorgestellten Konzept spielt der Wizard zur initialen Suche eine zentrale Rolle. Grundsätzlich kann er zudem dazu verwendet werden, um aus dem Kontext einer vorhandenen Anwendung nach Komponenten zu suchen. Das Konzept sieht diesbezüglich

mehrerlei valide Optionen vor. Zunächst kann die gleiche Empfehlungsstrategie genutzt werden und die besondere Situation anhand des Kompositionskontextes als Teil der Eingabeparameter für die Empfehlungsmethode berücksichtigen. Zudem besteht die Möglichkeit, eine dedizierte Empfehlungsstrategie zu definieren.

Der Wizard wird allen Aspekten von Anforderung 2 gerecht. Die Kommunikation mit Nutzern findet auf fachlicher Ebene statt, Assistenzmechanismen wie Komplettierungen und Vorschläge sind vorhanden, die Schritte können iteriert werden und unterstützen verschiedenartige Vorgehensweisen. Ergebnisse sind stets sofort sichtbar und deren Relevanz ist gekennzeichnet.

7.5 Erklärungstechniken

Wie in Anforderung 9 beschrieben, müssen Nicht-Programmierer beim Untersuchen, Nutzen und Nachvollziehen von Mashups unterstützt werden. Zu diesem Zweck wurden komplementäre Ansätze konzipiert, die nach einer Betrachtung von Anforderungen und verwandten Ansätzen im Detail vorgestellt werden.

7.5.1 Anforderungen und verwandte Ansätze

In der Bachelorarbeit [Pfl15] wurde, mit Unterstützung des Autors dieser Dissertation, zur Anforderungsanalyse eine kleine Nutzerstudie durchgeführt. Diese zielte drauf ab, Herausforderungen für Nicht-Programmierer hinsichtlich des Verständnisses von CWA und ihre Probleme mit aktuellen EUD-Werkzeugen zu untersuchen, um Schwachstellen in der Konzeption und der Gebrauchstauglichkeit zu erkennen. Details dazu sind auch [RM17b] zu entnehmen. Die Studie lieferte zahlreiche Erkenntnisse. So wurden bekannte Probleme der CapView als Mittel zum Nachvollziehen der Funktionsweise von CWA bestätigt und neue identifiziert. Hauptprobleme stellen das Umschalten in eine überlagerte Ansicht und Schwierigkeiten von Nutzern, den Zusammenhang von CapView-Elementen und zugehörigen UI-Bestandteilen zu erkennen, dar. Dies führte dazu, dass Probanden eher in der Live-View experimentierten als die CapView zu nutzen. Insgesamt kann zusammengefasst werden, dass CapView gute konzeptionelle Ansätze als Kompositionswerkzeug bietet, allerdings nur eingeschränkt zur Erklärung der Funktionsweise dienen kann. Daher gilt es, für diesen Zweck geeignetere Hilfsmittel zu konzipieren. Gemäß Anforderung 9 und unter Beachtung der dem Ansatz zugrundeliegenden Metamodelle, gilt es grundlegend, die Funktionsweise einzelner Komponenten und des Mashups zu verdeutlichen. Dazu sollten diese aufgelistet, im Kontext der Komponenten-UIs visualisiert, textuell erklärt und bei Bedarf sämtliche Interaktionsschritte in interaktiver Form animiert und vom Nutzer exploriert werden können. Es gilt, geeignete Konzepte zur Handhabung von kompositen Capabilities und von CWA mit mehreren Anwendungssichten zu entwickeln. Für eine umfassende Anforderungsanalyse sei auf [Pfl15] verwiesen.

Wie aus Kapitel 3.1 hervorgeht, stellen aktuelle Kompositionsplattformen keine vollumfänglich nutzbaren Ansätze zur Erklärung der Funktionsweise von Mashups und zur Visualisierung der IWC zur Verfügung. Deshalb wird nachfolgend ein kurzer Überblick von Lösungsmöglichkeiten aus weiter entfernten Forschungsbereichen vorgestellt.

Hilfestellungen, die Nutzern ein produktives Arbeiten mit einer Anwendung ermöglichen sollen, können in vielfältiger Darreichungsform konzipiert werden. Der Begriff

Performance-Support bezieht sich dabei auf Werkzeuge oder Ressourcen, die im Bedarfsfall in passendem Detailgrad bereitstehen und in Verbindung mit der Durchführung der Aufgabe eingesetzt werden. Gemäß [Ger95] können drei Kategorien solcher Hilfestellungen unterschieden werden. *Intrinsische Unterstützung* ist inhärenter Bestandteil des Systems und daher nahtlos in dessen UI und Verhalten integriert. Als Vorteil solcher Lösungen tritt kein Bruch im Arbeitsablauf auf. Die Unterstützungsmöglichkeiten sind jedoch aufgrund von Platzbeschränkungen im UI limitiert. *Extrinsische Unterstützung* wird ebenfalls direkt von der Anwendung bereitgestellt, ist jedoch nicht Teil des primären Arbeitsbereichs. Typischerweise findet eine Anpassung an den Aufgabenkontext statt und der Nutzer muss die Hilfe explizit aufrufen oder annehmen. Beispiele extrinsischer Hilfen sind Erklärungen, Demonstrationen, Wizards und Tipps. Durch diese Techniken findet ein geringer Bruch im Arbeitsablauf statt. *Externe Unterstützung* ist nicht Teil des Systems und des Arbeitsbereichs, sondern wird separat bereitgestellt. Beispiele dieser Kategorie sind Foren, Hilfe-Webseiten und Video-Anleitungen. Dadurch können zwar umfangreiche Informationen angeboten werden, diese sind jedoch selten auf den konkreten Aufgabenkontext angepasst, sodass Nutzer das erworbene Wissen auf den konkreten Fall übertragen müssen. Zudem findet ein starker Bruch im Arbeitsablauf statt. In Bezug auf diese Arbeit sind externe Hilfen gut geeignet um statische Informationen zu Komponenten und der Plattform bereitzustellen. Für situationsspezifische, kurzlebige CWA mit unvorhersehbaren Kombinationen von Komponenten und mit dynamischem Anwendungskontext lassen die Ansätze jedoch Praktikabilität missen.

Nachfolgend werden konkrete Ansätze hinsichtlich Anforderung 9 und ihrer Teilanforderungen untersucht. Der *Idea Garden* [Cao13] zielt darauf ab, Endnutzer bei der Überwindung von Entwurfsbarrieren durch extrinsische Hilfen zu unterstützen. Hierzu werden auf Veranlassen des Nutzers oder reaktiv auf Ereignisse strategische Vorschläge zur Problemlösung und zu Kompositionswissen und -mustern angezeigt. Die Hinweise sollen Nutzer anregen und umfassen schrittweise textuelle Instruktionen, wie eine Strategie durchgeführt werden kann. Die Vorschläge sind allerdings oftmals generell und kontextfrei, sodass ein konkreter Bezug zur Anwendung sowie deren UI nicht gegeben ist. Die textuellen Erklärungen können detailliert ausfallen und erfolgen schrittweise, werden jedoch statisch bereitgestellt. Ein Konzept zur Vermittlung von Kompositionswissen wird nicht näher ausgeführt.

Die in [KM04] vorgestellte *Whyline* unterstützt Nutzer bei der Identifikation und Behebung von Fehlern in ihren Programmen. Dazu konstruieren Nutzer Fragen nach dem Schema »why did« sowie »why did not« über Menüeinträge und unter Referenzierung von Objekten, welche im Algorithmus genutzt werden. Antworten werden anschließend anhand einer Graphdarstellung von Aktionen, die zur Ausführungszeit aufgetreten sind, und Pfeilen, welche deren kausale Zusammenhänge andeuten, präsentiert. Dieser Ansatz operiert nah am Quellcode und bietet Erklärungen für Fragen zur Funktionsweise einer Anwendung. Ein Bezug zum UI ist eingeschränkt gegeben, zumal eine Trennung von Lauf- und Entwicklungszeit besteht. Textuelle Erklärungen werden vorgenommen. Die übrigen Anforderungen sind kaum anwendbar, da keine CWA vorliegen. Auch fehlt es an Konzepten zur Darstellung von Datenaustausch zwischen Komponenten zur Laufzeit.

Kuttal et al. [KSR13] stellen ein Werkzeug zum Debugging von Mashups am Beispiel von *Yahoo Pipes!* vor. Die Autoren klassifizieren Arten von Anomalien, die in solchen Mashups auftreten können, in *Intra-module-Bugs* und *Inter-module-Bugs*, wie fehlende Verbindungen zwischen Modulen und Verknüpfung inkompatibler Datentypen. Solche

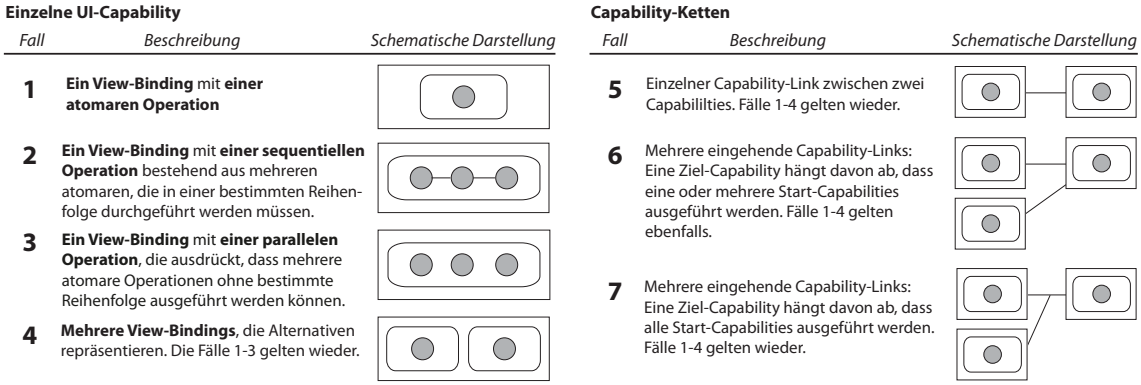


Abbildung 7.8: Wesentliche zu betrachtende Fälle und Konstellationen (nach [RM18])

Fehler werden in der Kompositionslogik automatisch detektiert, zum Beispiel durch statische Codeanalysen, und Nutzern aufgelistet. Die Liste hebt betroffene Module hervor, beschreibt das Problem und gibt Hinweise zu Lösungsoptionen. Somit erfüllt der Ansatz teilweise die Forderung nach Bezug zum UI und textuellen Erklärungen. Allerdings liegt der Fokus auf Erklärung von Problemen und Lösungen und nicht wie benötigt auf der Funktionsweise von Komponenten und Kompositionen.

7.5.2 Kernkonzepte

In Anbetracht der Modellierungskonzepte aus Kapitel 5.3 müssen verschiedene Konstellationen von Capabilities und View-Bindings durch Erklärungstechniken, die auf Capability-Graphen basieren, berücksichtigt werden. Abbildung 7.8 fasst die wichtigsten Fälle zusammen, welche im Zuge der gesamten Konzeptbeschreibung einbezogen werden.

Um die Interaktionsschritte für eine komposite Capability zu identifizieren, ist es notwendig, den Hierarchiegraphen bis zum Erreichen atomarer Capabilities zu traversieren. Dies führt zu den entsprechenden Capability-Ketten, die es erlauben, die oben genannten Fälle anzuwenden. Die vorgeschlagenen Erklärungstechniken operieren direkt innerhalb des UI von Komponenten und nicht in abstrakten Ansichten. Um dies im Kontext eines Blackbox-Komponentenmodells zu erreichen, definieren View-Bindings deklarativ die Verknüpfungen zwischen UI und Capabilities. Innerhalb der Erklärungstechniken werden konsistent drei grundlegende visuelle Elementtypen verwendet.

1. **Rahmen** heben UI-Elemente hervor, die in View-Bindings referenziert wurden. 2. **Pfeile** verbinden Rahmen und repräsentieren Capability-Links, wobei sie die Datenflussrichtung angeben. 3. In **Beschreibungsfenstern** werden textuelle Erklärungen zu Capabilities und Interaktionsschritten angezeigt. Sie werden in der Nähe von Rahmen oder Pfeilen positioniert, wodurch Überlappungen minimiert werden. Der Text wird kontextsensitiv aus den Annotationen der Capabilities und View-Bindings generiert.

Standardmäßig wird für jedes Element des Document Object Model (DOM), das von den Selektoren der atomaren View-Binding-Operationen einer Capability referenziert wird, ein Rahmen erstellt. Im speziellen Fall von Nicht-UI-Komponenten wird ein Beschreibungsfenster erzeugt und eingerahmt. UI-Elemente, die von View-Bindings referenziert werden, jedoch zum benötigten Zeitpunkt nicht sichtbar sind, stellen eine weitere Herausforderung dar. Zwar mag es technisch möglich sein, solche Elemente sichtbar zu

machen, beispielsweise durch deklarative Anweisungen in SMCDL oder mit imperativem Quellcode, der vom MRE aufgerufen wird. Dies führt jedoch zu einem hohen Aufwand für Komponentenentwickler. Zudem erscheint es fraglich, ob solche Lösungen generisch funktionieren. Daher sieht das Konzept eine vereinfachte, generische Lösung vor: Sollte ein referenziertes Element unsichtbar sein, wird die gesamte Komponenten eingerahmt und ein textueller Hinweis in einem Beschreibungsfenster angezeigt.

7.5.3 Assistenzwerkzeuge

Das Konzept stellt eine Reihe generischer Erklärungstechniken vor, die auf den in SMCDL beschriebenen Capabilities, den Capability- und Hierarchiegraphen einer CWA und den oben vorgestellten visuellen Elementtypen basieren. **Capability-Panels** zählen die Fähigkeiten von Mashups und Komponenten auf. Ein **Empfehlungsmenü** zeigt empfohlene Pattern-Instanzen an. Der **Inspektionsmodus** dient der Untersuchung von Capabilities und Capability-Links direkt in der Benutzeroberfläche von Komponenten. Der **Tutorialmodus** präsentiert Schritt-für-Schritt-Anweisungen und der **Awareness-Modus** visualisiert den Datenfluss zur Laufzeit.

Exploration von Funktionalitäten im UI mit dem Inspektionsmodus

Der Inspektionsmodus ermöglicht es Nicht-Programmierern, die Funktionalität von Komponenten und CWA zu erforschen. Sein Hauptzweck ist es, die Capability-Ketten eines Mashups darzustellen, ihre Funktionalität zu beschreiben und entsprechende UI-Elemente hervorzuheben. Auf diese Weise kann der Inspektionsmodus als eine in die Live-View integrierte Capability-View verstanden werden.

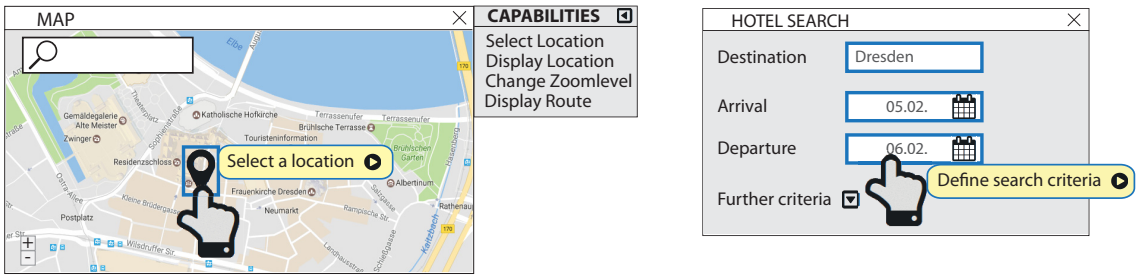


Abbildung 7.9: Inspektionsmodus am Beispiel einzelner Komponenten (nach [RM18])

Der Inspektionsmodus kann über einen Menübutton aktiviert und deaktiviert werden. Sobald er aktiv ist, werden alle Verbindungen zwischen den Komponenten durch das Zeichnen von Rahmen und Pfeilen visualisiert. Capability-Links innerhalb von Komponenten werden zunächst ausgeblendet, falls sie nicht transitiv mit Inter-Komponenten-Links verbunden sind, und bei Bedarf angezeigt. Dadurch wird die Übersichtlichkeit verbessert. Zudem sind den Nutzern die Capabilities einzelner Komponenten oft bekannt.

Wenn ein Nutzer das UI einer Komponente mittels Mauszeiger durchsucht, erscheint das zur Komponente gehörige Capability-Panel. Darüber hinaus werden Rahmen sichtbar sobald sich der Zeiger über zugehörigen UI-Elementen befindet, siehe linker Teil von Abbildung 7.9. In den Fällen 2 und 3 aus Abbildung 7.8 werden auch alle anderen Elemente, auf die in Geschwistern einer atomaren Operationen verwiesen wird, hervorgehoben, siehe rechte Seite von Abbildung 7.9. Zusätzlich erscheint ein Beschreibungsfenster, das über

einen »Play-Button« den Zugang zum Tutorialmodus für diese Capability ermöglicht. Es enthält einen generierten Text, der unter Wiederverwendung der Konzepte aus Kapitel 7.3.2 von Capabilities abgeleitet wird. Sofern die Capability, über deren View-Binding der Mauszeiger schwebt, Teil einer Capability-Kette ist, werden alle entsprechenden Pfeile sowie Rahmen hervorgehoben und Intra-Komponenten-Ketten werden sichtbar. Zusätzlich erwähnt der Beschreibungstext alle Capabilities eines Capability-Links und weist auf Ursache und Wirkung hin. Dazu werden Sätze aus den beteiligten Capabilities und aus den Kommunikationskanälen gebildet, wobei auf die Kommunikationsrichtung geachtet wird. Das Regelwerk zur Erzeugung von Beschriftungen aus Kapitel 7.3.2 wird dahingehend erweitert, auch Komponentennamen aufzunehmen und mehrere Beschriftungen mit »und« (im Fall 7 aus Abbildung 7.8) und »oder« (im Fall 6) zu verbinden. Beim Schweben mit dem Mauszeiger über speziellen Textfragmenten, wie Komponentenname und Capability-Beschriftung, wird die komplette Komponenten oder der Rahmen des View-Bindings hervorgehoben.

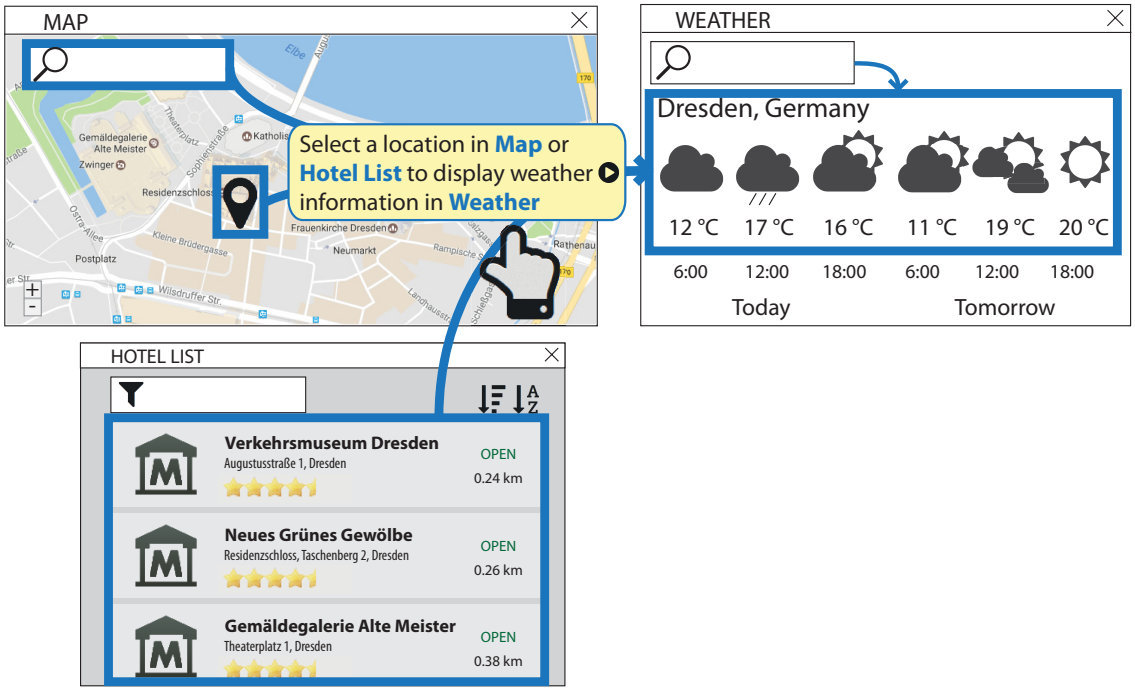


Abbildung 7.10: Inspektionsmodus im Fall verknüpfter Capabilities [RM17b]

Wie Abbildung 7.10 zeigt, werden bei der Auswahl eines Pfeils dieser und alle anderen Pfeile sowie Rahmen, die zum dargestellten Capability-Link gehören, hervorgehoben und es wird ein Beschreibungsfenster angezeigt. Auch hier können Nutzer ein Tutorial starten. In Abschnitt 2.1.2 wurde erwähnt, dass Komponenten in verschiedenen Anwendungsansichten angeordnet werden können. Um dem Rechnung zu tragen, werden im Inspektionsmodus Komponenten, die an einer Capability-Kette teilnehmen, jedoch derzeit nicht sichtbar sind, durch ihr Icon dargestellt. Dieses wird am Rand des Anwendungsbereichs positioniert und dient als Ankerelement für zu dieser Komponente gehörende Rahmen. Weiterhin ermöglicht das Beschreibungsfenster in solchen Fällen den Wechsel zum entsprechenden Anwendungsbildschirm.

Bedienungserklärungen via Tutorialmodus

Der Tutorialmodus legt das Hauptaugenmerk auf die Erläuterung notwendiger Interaktionsschritte und nicht auf die vollständigen Details einer Capability. Daher sind UI-Capabilities von besonderem Interesse und werden benötigt, um schrittweise Anweisungen zu generieren. Die Tutorials können über Schaltflächen in den Capability-Panels und in den Beschreibungsfenstern des Inspektionsmodus aufgerufen werden.

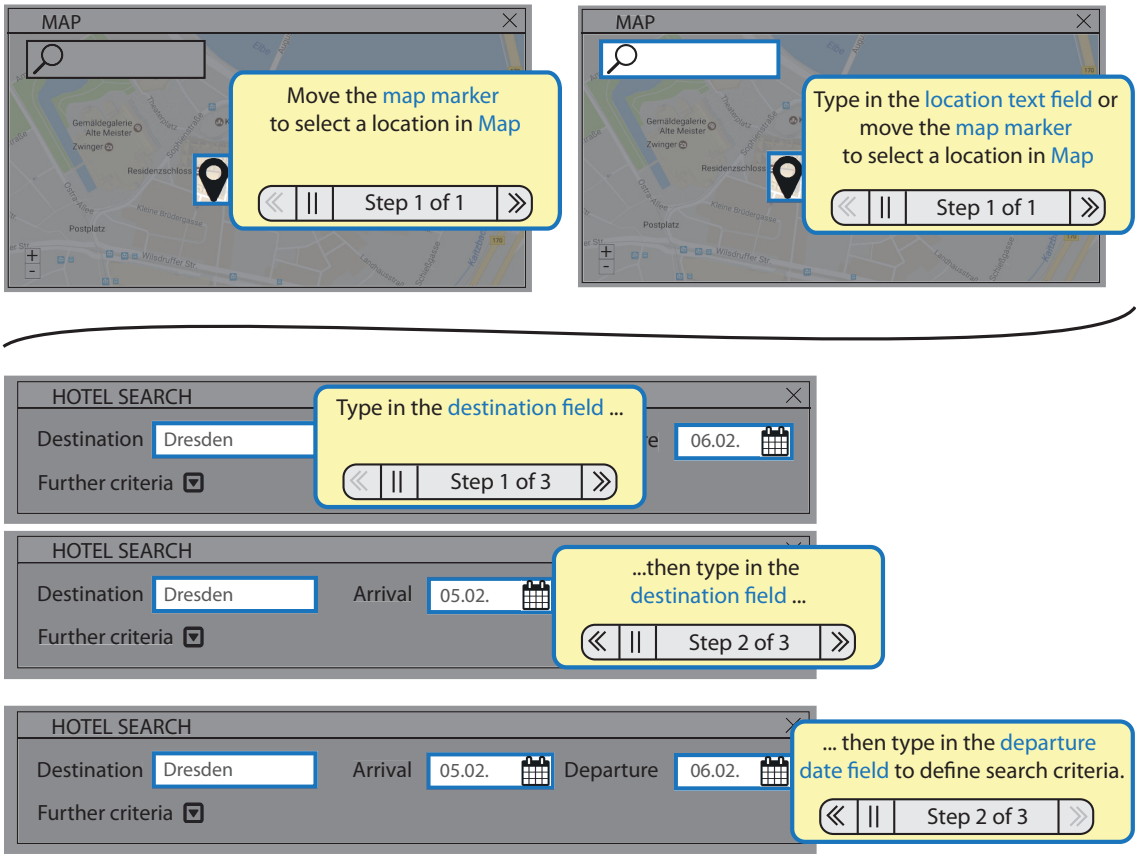


Abbildung 7.11: Beispiel eines Tutorials für eine einzelne Komponente (nach [RM17b])

Wenn der Tutorialmodus für eine Capability oder eine Capability-Kette aktiviert ist, visualisiert er die Rahmen des ersten Interaktionsschrittes und hebt diese hervor, indem er die übrige CWA ausgraut. Dies wird in Abbildung 7.11 veranschaulicht und lenkt die Aufmerksamkeit des Nutzers. Zudem erscheinen Beschreibungsfenster, die einen generierten Text sowie eine Navigationsleiste enthalten. Letztere bietet Schaltflächen zur automatischen Wiedergabe, zum Schließen, Pausieren sowie zur schrittweisen Navigation im Tutorial. Die generierten Beschreibungstexte benennen die Interaktionstechnik und den Elementnamen, wie er im View-Binding definiert ist, die zugehörige Capability sowie den Komponentennamen. Auch hier hängt die Satzstruktur von der Position der Capability in einem Capability-Link ab, um Ursache und Wirkung widerzuspiegeln.

Abbildung 7.11 zeigt im oberen Teil Beispiele für die Fälle 1 und 4 (vergleiche Abbildung 7.8). In Fall 2 existiert ein Tutorialschritt pro atomarer Operation, siehe unterer Teil von Abbildung 7.11, während in Fall 3 ein einzelner Schritt sowie ein Text, der alle atomaren Operationen zusammen benennt, vorhanden sind. Ebenso werden in Fall 6 alle Quell-Capabilities eines Capability-Links in einem Schritt erwähnt, mit einem

Beschreibungsfenster pro Quell-Capability. Im Gegensatz dazu führt eine Konstellation wie in Fall 7 zu getrennten Schritten pro Quell-Capability.

Abbildung 7.12: Erzeugtes Tutorial für eine Inter-Komponenten-Capability (nach [RM17b])

Tutorials beziehen auch Nicht-UI-Komponenten mit ein. Dazu wird ein zusätzlicher Schritt eingefügt, der ein Beschreibungsfenster visualisiert. Dieses präsentiert den Namen, das Icon und die Capabilities der Komponente, die im Zusammenhang mit der zu erklärenden Capability-Kette zur Verfügung gestellt werden. Ein Beispiel befindet sich in Abbildung 7.12. Analog dazu werden Komponenten behandelt, die zwar in einem Schritt erklärungsbedürftig sind, jedoch in der aktuellen Anwendungssicht nicht angezeigt werden. In diesem Fall ermöglicht das Beschreibungsfenster das Umschalten in die entsprechende Ansicht und zurück.

Hervorheben von Datentransfer zur Laufzeit: der Awareness-Modus

Der Awareness-Modus hat zum Ziel, Nutzer auf den Datenaustausch via IWC aufmerksam zu machen, wenn er zur Laufzeit einer beliebigen CWA stattfindet. Dazu baut es auf den vorgestellten Visualisierungskonzepten für Inter-Komponenten-Capabilities auf. Über eine speziellen Menübutton kann der Awareness-Modus aktiviert werden.

Sobald ein Event auf einem Kommunikationskanal auftritt, wird die zugehörige Capability c aus der SMCDL der publizierenden Komponente ermittelt und als Start-Capability

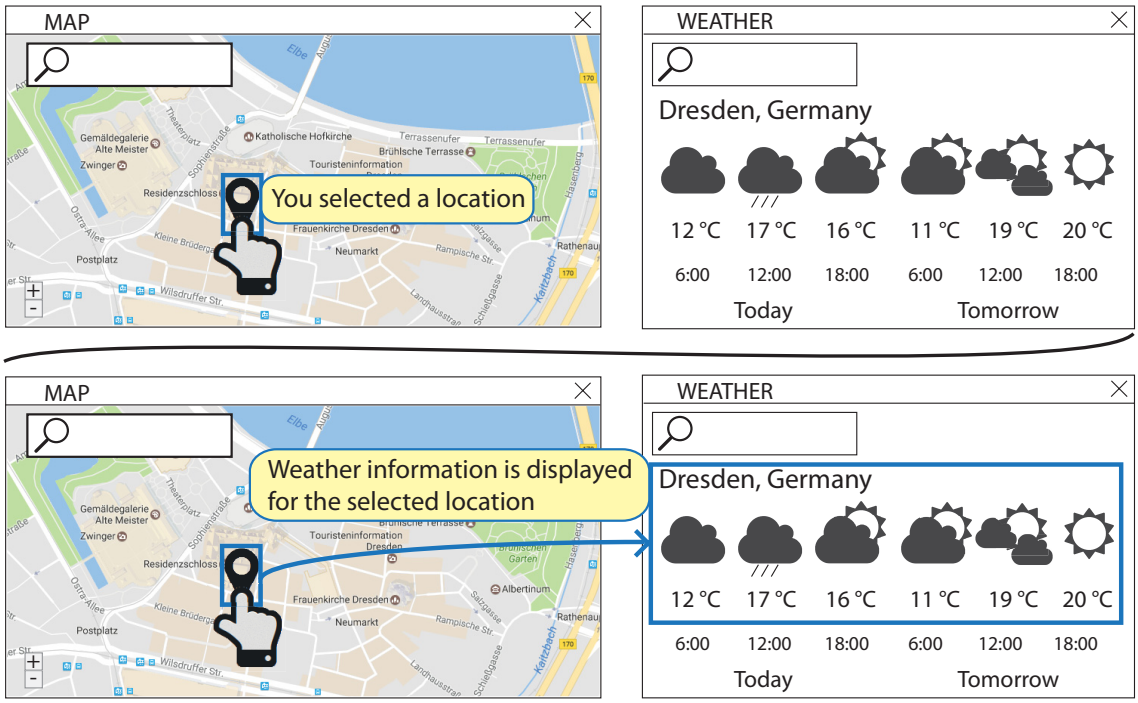


Abbildung 7.13: Beispielhafter Awareness-Modus für zwei Zeitpunkte [RM17b]

c_s gespeichert. Falls c keine UI-Capability ist, wird die Capability-Verkettung via *causedBy* von c ausgehend, falls vorhanden, solange innerhalb der Komponente transitiv zurückverfolgt, bis eine UI-Capability gefunden wird und diese als c_s definiert. Analog dazu wird die Ziel-Capability c_t eines Capability-Links unter Nutzung der Verkettung über *causes* bestimmt. Anschließend startet eine Animation von konfigurierbarer Dauer. Sie zeigt zunächst die Rahmen von c_s, wie im oberen Teil von Abbildung 7.13 dargestellt. Sollte c_s keine UI-Capability sein oder keine View-Bindings besitzen, wird die komplette Komponente eingerahmt. Ein Beschreibungsfenster mit einem generierten Text zeigt außerdem die gerade aktivierte Capability c_s an. Als nächstes wird ein Pfeil angezeigt, gefolgt von den View-Binding-Rahmen von c_t und einem entsprechenden Beschreibungsfenster, siehe unterer Teil von Abbildung 7.13. Fall 7, das heißt, wenn mehrere Interaktionen notwendig sind, um alle Daten zusammenzuführen, bevor c_t erbracht werden kann, erfordert besondere Betrachtung. Der letzte Animationsschritt wird zunächst hinausgezögert. Stattdessen werden die Rahmen der benötigten anderen Capabilities für eine Weile gezeigt und hervorgehoben, jeweils ergänzt durch ein Beschreibungsfenster, das notwendige Interaktionen benennt. Sobald alle Eingaben vorliegen, wird die Animation mit den Rahmen und Beschreibungsfenstern von c_t fortgesetzt.

Falls eine Komponente, die c_s oder c_t bereitstellt, nicht Teil der aktuellen Anwendungsansicht ist, kommt der vom Inspektionsmodus bekannte Ansatz zum Tragen: Ein Icon repräsentiert die Komponente und fungiert als Ankerelement für Rahmen und Pfeile.

Auflistung von Capabilities in Capability-Panels

Capability-Panels geben einen Überblick und Einblick in die Capabilities von Komponenten und Mashups. Ein Capability-Panel präsentiert die baumartigen Hierarchiegraphen einer CWA beziehungsweise listet die Fähigkeiten einer einzelnen Komponente, wie es beispiel-

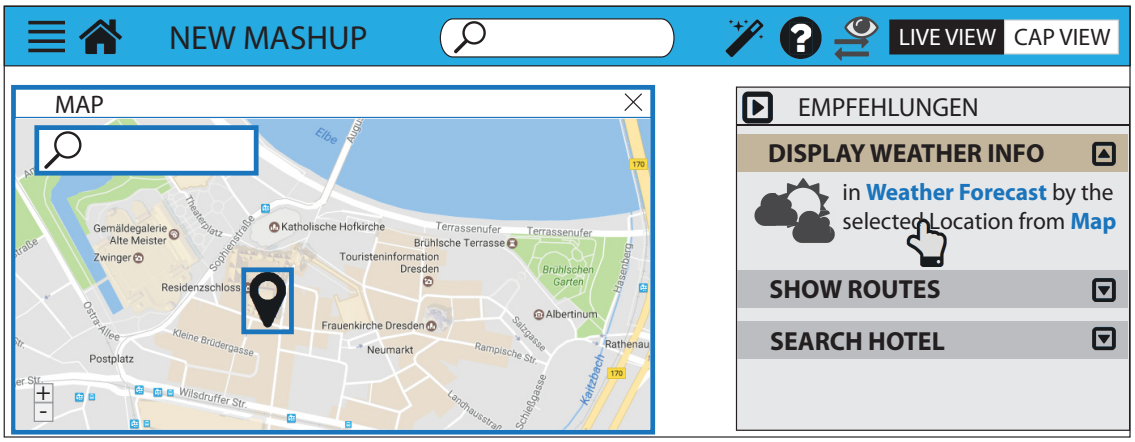


Abbildung 7.14: Empfehlungsmenü mit Kompositionsvorschlägen (nach [RBM17])

haft Abbildung 7.9 dargestellt. Darin werden die Capabilities mit Hilfe von generierten Beschriftungen textuell beschrieben. Sobald ein Nutzer eine Capability auswählt, kann der Tutorialmodus für diese aktiviert werden. Capability-Panels werden mit dem Inspektionsmodus synchronisiert: Wenn der Mauszeiger über einem View-Binding schwebt, heben betroffene Capability-Panels die zugehörige Capability hervor. Zudem bieten Capability-Panels die Möglichkeit, Knoten des Hierarchiegraphen zu fokussieren. Das bewirkt, dass lediglich die zum Knoten (und dessen untergeordneten Knoten) gehörigen Capability-Ketten im Inspektionsmodus visualisiert werden. Dies erhöht die Übersichtlichkeit und ist angelehnt an die in Kapitel 3.1.2 beschriebenen *Behaviours* von WireCloud.

Erklären von empfohlenen Kompositionsschritten im Empfehlungsmenü

Das *Empfehlungsmenü*, ein Recommendation-Viewer gemäß Kapitel 6.3.4, zeigt Nutzern empfohlene Kompositionsschritte an. Wie in Abbildung 7.14 auf der rechten Seite zu sehen, findet ein neuartiger zweistufiger Ansatz zur Visualisierung von Empfehlungen statt. Dieser basiert weitgehend auf den Fähigkeiten der zugrundeliegenden Pattern-Instanzen. In einem Vorverarbeitungsschritt werden die Pattern-Instanzen nach ihren Capabilities gruppiert. Daraufhin werden alle Gruppen mit Hilfe von generierten Beschriftungen präsentiert, zum Beispiel `Display Weather Info`. Sobald der Nutzer seine Wahl getroffen hat, erscheinen Beschreibungen für jede Pattern-Instanz aus der selektierten Gruppe. Diese umfassen Icons der enthaltenen Komponenten und zusätzliche Textfragmente, welche die Beschriftung der Gruppe zu einem kurzen Satz ergänzen. Wie in Abbildung 7.14 rechts ersichtlich ist, werden im Text Komponentennamen und die entsprechenden Capabilities erwähnt. Beim Positionieren der Maus über diesen Textbestandteilen wird die Komponente hervorgehoben beziehungsweise die Rahmen für die View-Bindings der Capability eingeblendet.

8 Implementierung und Evaluation

In diesem Abschnitt wird die prototypische Umsetzung und Validierung wesentlicher Konzepte beschrieben. Das grundlegende Rahmenwerk besteht dabei aus den praktischen Ergebnissen vorheriger Forschungsarbeiten aus dem CRUISE-Umfeld, wie Referenzimplementierungen der clientseitigen Laufzeitumgebung Thin-Server Runtime (TSR), der Client-Server Runtime (CSR), des Kontextdienstes CroCo, einem Komponentenrepositorium und eines ersten Mediators. Diese Artefakte werden einbezogen und an konzeptrelevanten Stellen substantiell erweitert. Eine Annahme für die Implementierung ist, dass UI-Komponenten ein DOM besitzen. Plugin-Technologien werden ausgeklammert, was angesichts deren schwindender Relevanz und dem Vorantreiben aktueller Webstandards wie HTML5 keine Einschränkung der praktischen Anwendbarkeit der Konzepte ist.

Sämtliche in diesem Kapitel erwähnten Performanzmessungen basieren auf einem Testsystem, das hardwareseitig mit Intel Core i7-4900 2.8 GHz, 32 GB Arbeitsspeicher sowie einer SSD-Festplatte und softwareseitig mit Windows 7, jeweils aktueller Java-JRE und aktuellem Apache Tomcat oder TomEE ausgestattet ist. Serverseitige Implementierungen erfolgen in Java unter Verwendung der Entwicklungsumgebung Eclipse mit Apache Maven als Werkzeug für die Verwaltung des Erstellungsprozesses der Softwareartefakte.

8.1 Umsetzung der Modelle und der Basisarchitektur

Die Komponentenbeschreibungssprache SMCDL wurde entsprechend der vorgeschlagenen Konzepte angepasst. Das zugehörige XML-Schema ist dazu insbesondere um Capabilities erweitert worden. Listing A.1 im Anhang stellt ein umfassendes Beispiel dar. Insgesamt wurden 64 Komponenten unter Mitwirken des Autors dieser Dissertation entwickelt und semantisch annotiert, um die Praktikabilität der Konzepte zu verdeutlichen. Zur Annotation benötigte Ontologien wurden teils vom Autor der vorliegenden Arbeit oder in Kooperation implementiert, etwa `travel.owl`, `finance.owl`, `consulting.owl`, teils wiederverwendet, wie QUDT, foaf und dublin core. Die Komponenten decken eine große Bandbreite an Themenbereichen ab, zum Beispiel Reiseplanung, Unterhaltung, Terminplanung und Finanzberatung, sodass sie eine geeignete Grundlage für vielfältige CWA bieten. Zudem verfügen sie über zahlreiche Schnittstellenelemente zur Sicherstellung

der Verknüpfbarkeit der Komponenten untereinander, wie die Ausführungen in Kapitel 8.4.1 belegen. Die Nutzung von View-Bindings konnte in kollaborativen Szenarien gezeigt werden [Bli+16]. Dennoch muss angemerkt werden, dass der gewählte Ansatz semantisch annotierter Komponenten einen nicht zu vernachlässigenden Aufwand für Komponentenentwickler verursacht, wie die Richtlinien in Abschnitt A.1 erkennen lassen. Diesbezüglich bedarf es geeigneter Werkzeuge und Assistenzmechanismen. Allerdings bieten erst Annotationen die Möglichkeit der Automatisierung wie im Konzept vorgesehen. Das vollständige Capability-Metamodell zur Annotation von Kompositionen sowie Pattern-Instanzen, das Feedback-Metamodell, Metadaten zu Kompositionen sowie die Gesamtheit aller Pattern-Klassen wurden jeweils als Java-Objektmodell implementiert. Der Capability-Wissensgraph aus Kapitel 5.4 wurde als RDF-Graph umgesetzt und wird durch Implementierung der vorgeschlagenen Ansätze befüllt. Die praktische Erprobung der realisierten Modelle durch Instantiieren und Verwenden innerhalb der prototypischen Mashup-Plattform belegt ihre Angemessenheit und Praktikabilität. Dies erfolgte im Kontext verschiedener Anwendungsszenarien, wie Reiseplanung, Finanzwesen, Unterhaltung und elektronischer Handel. In Kapitel 8.3.1 werden Beispielanwendungen erläutert. Zudem trägt die Verwendung von etablierten Ontologien sowie W3C-Standards der Forderung nach Standardkonformität Rechnung.

Das Architekturkonzept wurde auf Basis von Webtechnologien für das Frontend und Java auf Serverseite realisiert. Das bestehende Komponentenrepositorium wurde substantiell erweitert und beherbergt nun ein Anwendungsrepositorium, Teile des Empfehlungssystems, ein Feedback-Repositorium, und den Algorithmus zum Abschätzen von Capabilities. Historisch bedingt existieren zwei Laufzeitumgebungen. Die TSR operiert rein clientseitig und dient dem Validieren der Capability-View, des Empfehlungssystems sowie des Screenflow-Editor. Die zwischen Client und Server verteilte CSR bietet Funktionsmuster insbesondere des Startbildschirms, der Erklärungstechniken und des Wizards. Der Mediator wurde verteilt implementiert, wie Abschnitt 8.2 diskutiert.

Obgleich der Prototyp der konzipierten Mashup-Plattform auf Grundlage von Webtechnologien umgesetzt wurde, zeichnen sich die Konzepte dadurch aus, diesbezüglich nicht eingeschränkt zu sein. Grundsätzlich sind Implementierungen mithilfe anderer Komponententechnologien wie EJB oder OSGI möglich.

8.2 Realisierung der Mediationskonzepte

In diesem Abschnitt wird die prototypische Implementierung der in Kapitel 5.2 vorgestellten Ansätze zur semantischen Datenmediation näher erläutert.

8.2.1 Erweiterung des Kompositionsmodells

Da Mediationsvorschriften ein integraler Bestandteil der Kommunikation von Komponenten innerhalb einer CWA sind, wurde das MCM erweitert. Gemäß der Trennung der Zuständigkeiten betrifft dies vor allem das *Communication-Model*. `Channels` weisen nun das Element `mapping` auf, welches die zugehörige *Mapping-Definition* beinhaltet, siehe untere linke Seite von Abbildung 8.1. Für letztere wurde ein separates XSD entwickelt, das in das MCM importiert wird. Durch die Verwendung von XSD kann von diversen Frameworks profitiert werden, zum Beispiel bei der Generierung von Java-Klassen via Java Architecture for XML Binding (JAXB). Die obere und rechte Seite von Abbildung

8.1 illustriert die oberste Ebene der Typdefinition für Mapping-Definitionen. Eine solche umfasst demnach eine Identifikation zu Verwaltungs- und Referenzierungszwecken, eine Signatur bestehend aus den Quell- und Zielparametertypen sowie alle notwendigen Mediationstechniken im Element `mappings`. Folgende Elemente sind vorgesehen und decken alle kanalgebundenen Techniken ab.

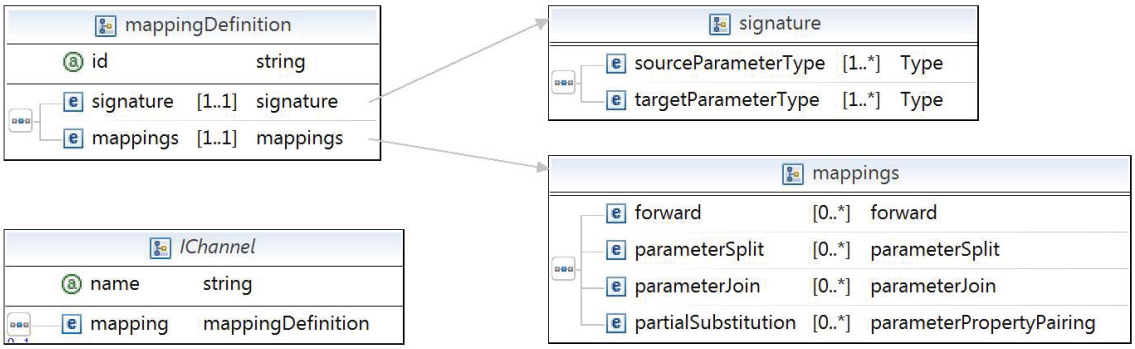


Abbildung 8.1: Übersicht des umgesetzten XML-Schemas für Mapping-Definitionen

- Das Element `forward` dient der Beschreibung von Parameterzuweisungen und bietet die Möglichkeit, *Upcast*, *Convert* und *Collection-Conversion* abzubilden.
- Die Elemente `parameterSplit`, `parameterJoin` und `partialSubstitution` decken die Techniken *Semantic Split*, *Semantic Join* und *Partial Substitution* ab.

Wie bereits konzeptionell angedacht, werden *Syntactic Joins* durch Mediationskomponenten realisiert. Dadurch manifestieren sie sich nicht im vorgestellten XSD, sondern direkt im *Communication-* und *Conceptual-Model*. Die im MCM vorhandene Beschreibung von Syntactic Join nutzen die TSR und die CSR zum Konfigurieren von bereitgestellten Schablonen für die Komponentenimplementierung und für die SMCDL, sodass Mediationskomponenten analog zu den übrigen Komponenten gehandhabt werden können.

8.2.2 Implementierung des Mediators

Konzeptionell umfasst der *Mediator* die Funktionalitäten zur *Mapping-Discovery* und zur *Mapping-Execution*, vergleiche Kapitel 5.2. Deren Verteilung und Umsetzung in der Referenzimplementierung behandelt dieses Kapitel.

Die *Mapping-Discovery* wurde in den Prototyp des Komponentenrepositoriums integriert, da dort Empfehlungsmechanismen und Pattern-Mining angesiedelt sind, die substantiell darauf aufsetzen. Die Klasse `DataSemanticsMatcher` verfügt über die entsprechende Funktionalität, um für zwei gegebene Signaturen eine `MappingDefinition` zu bestimmen, sofern möglich. Zusätzlich wurde das Komponentenrepositorium um die Klasse `MediationRuleStorage` erweitert. Diese dient zum persistenten Verwalten von Mapping-Definitionen im Dateisystem durch Mechanismen von JAXB. Weiterhin wurde die Hilfsklasse `MediationRuleSerializer` implementiert, die dem Serialisieren von `MappingDefinition`-Objekten nach XML und JSON dient. Analog dazu bietet der `MediationRuleDeserializer` Funktionen zum Instantiieren von Java-Objekten aus

serialisierten `MappingDefinitions`. In einem eigenständigen Modul wurden Konverter-Funktionalitäten umgesetzt. Das Komponentenrepositorium bezieht es zur *Mapping-Discovery* ein. Allen Konverter implementieren die Java-Schnittstelle `IConverter`, die zwei Methoden vorsieht. Die erste trifft eine Aussage darüber, ob zwei gegebene semantische Konzepte mit diesem Konverter umgewandelt werden können. Die zweite führt die Konvertierung durch. Als konkrete Subklassen wurden `EquivalentClassConverter` und `QUDTConverter` prototypisch erprobt. Ersterer prüft im Wesentlichen, ob zwischen beiden gegebenen Ontologieklassen die Beziehung `owl:hasEquivalenceClass` vorhanden ist oder geschlussfolgert werden kann. Die Konvertierung läuft auf das Erstellen eines Individuums der Zielklasse und das Kopieren sämtlicher OWL-Properties unter Beachtung von `owl:hasEquivalentProperty` hinaus. Der `QUDTConverter` macht die Konvertierbarkeit davon abhängig, ob beide übergebenen Ontologiekonzepte Längeneinheiten, das heißt vom Typ `qudt:LengthUnit`, sind. In der QUDT besitzen sämtliche Domänen, wie Längeneinheiten, eine Referenzeinheit, auf die konkrete Maßeinheiten umgerechnet werden können. Daher ist im Rahmen der Konvertierung zunächst das Bestimmen der jeweiligen Umrechnungsfaktoren zur Referenzeinheit notwendig. Danach kann der gegebene Wert mit den Faktoren in die Zieleinheit umgerechnet werden.

Bei der Implementierung von Algorithmen zur *Mapping-Discovery* wurde aus Gründen der Komplexität darauf geachtet, dass bei dem Untersuchen von Ontologiekonzepten, zum Beispiel hinsichtlich Möglichkeiten zum *Convert* und *Semantic Split*, nicht beliebig tief in den Wissensgraph eingetaucht wird, sondern nur direkt mit Klassen assoziierte OWL-Properties beachtet und nicht transitiv nachverfolgt werden.

Die *Mapping-Execution* wurde im Prototyp zwischen MRE, sowohl der TSR als auch der CSR, und einem dedizierten Webservice, dem *Mediationsdienst* verteilt. Der im MRE verortete Teil dient beispielsweise zur Transformation von Instanzdaten einer Komponente in das Standard-Grounding und zur Bereitstellung von *Syntactic Joins*. Der Webservice hingegen ist zuständig für die Durchführung von Mediationstechniken, die auf semantischer Ebene stattfinden. Diese Trennung ist aus vielerlei Hinsicht vorteilhaft. Sie erlaubt eine Performanzsteigerung, indem nicht-semantische Mediationsaspekte direkt in der Laufzeitumgebung behandelt werden können. Weiterhin muss ein MRE nicht sämtliche Techniken selbst bereitstellen, was zudem den Umgang mit semantischen Technologien erfordern würde. Letzteres wäre besonders in der TSR problematisch, da diese komplett im Browser ausgeführt wird. Im Rahmen dieser Arbeit wurde die in der TSR vorhandene Implementierung dahingehend erweitert, dass der `Mediator` den Mediationsdienst per *Service-Access* anbindet. Analog zum Server existieren Funktionen zum Serialisieren und Deserialisieren, bereitgestellt von der Klasse `MediationUtil`. Weiterhin müssen die vom Server im Zuge von vorgeschlagenen Pattern-Instanzen mitgelieferten `MappingDefinitions` an die jeweiligen Komponentenschnittstellen maßgeschneidert werden, da serverseitig aus Effizienzgründen pro `Signatur` nur eine `MappingDefinition` hinterlegt wird. Der Mediationsdienst wurde in Java implementiert und stützt sich auf Apache Jena [@Jena] zum Handhaben semantischer Modelle. Dazu stellt es Mechanismen zur Persistierung und Verwaltung, zur Validierung, zur programmatischen Manipulation von Modellen, zum Reasoning und zum Abfragen von RDF-Graphen via SPARQL Protocol and RDF Query Language (SPARQL) zur Verfügung. Zwecks Konvertierungen bezieht der Mediationsdienst das Konverter-Paket mit ein. Es existieren zwei Klassen, die über Apache Axis2 [@Axis2] als SOAP-Webservice angeboten werden. Die eine Variante ist nutzbar, wenn die zu mediierenden Instanzdaten im Standard-Grounding in XSD vorliegen und Lifting

und Lowering anzuwenden sind. Abbildung 8.2 veranschaulicht dies für einen *Semantic Split* der `Location` aus einem `Appointment`. Daneben gibt es einen direkten Zugang für MRE, bei welchen Instanzdaten in RDF zwischen Komponenten ausgetauscht werden.

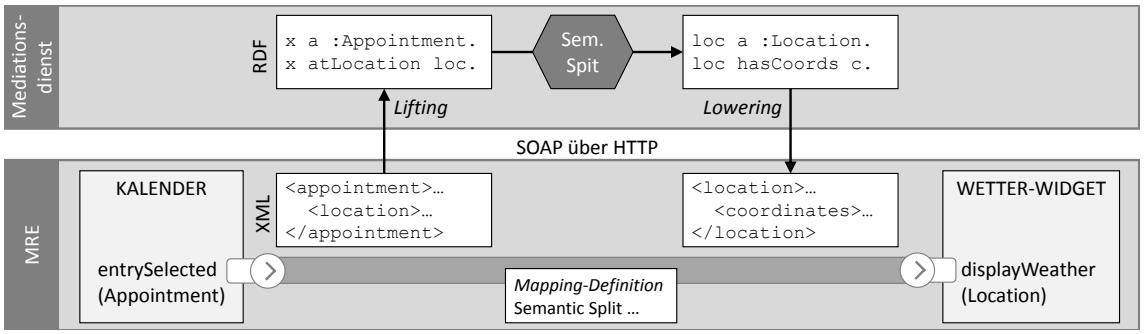


Abbildung 8.2: Exemplarischer Ablauf bei der Ausführung eines *Semantic Split*

8.2.3 Evaluation und Diskussion

Auf Basis der in Kapitel 8.1 beschriebenen Anwendungsszenarien und der dabei verwendeten Ontologien konnte die Praxistauglichkeit der Mediationstechniken erprobt werden, indem beispielsweise die in Tabelle 8.1 aufgeführten Konstellationen getestet wurden. Dazu wurden automatisierbare Testfälle sowohl zum korrekten Ableiten von Mapping-Definitionen als auch zu deren korrekter Anwendung auf Instanzdaten definiert und durchgeführt. Weiterhin wurden Mapping-Discovery und -Execution durch die in Kapitel 8.2.2 beschriebene Anbindung an das Empfehlungssystem validiert. Auf Basis der in Kapitel 8.4 beschriebenen Implementierung, erfolgte dazu Pattern-Mining von Coupling-Patterns. Deren Mapping-Definitionen wurden auf Plausibilität überprüft und durch Integration der Pattern-Instanzen in selbst erstellten CWA validiert. Analog dazu bildeten manuell bereitgestellte Complex-Patterns die Grundlage zum Erproben der Kombination von Mediationstechniken unter Verwendung von Syntactic Join.

Um die Praktikabilität der Konzepte zu zeigen, wurden Performanzmessungen durchgeführt. Ziel war dabei, den Einfluss von Mediationstechniken auf die Bedienbarkeit von CWA abzuschätzen. Dazu wurden Beispiele aus dem Referenzszenario aufgegriffen und mittels SOAP-UI [@SoapUI] in automatisierte Testfälle überführt. Diese wurden jeweils 100 mal ausgeführt und die Ergebnisse hinsichtlich syntaktischer und semantischer Korrektheit geprüft. Zudem erfolgte dabei das Aufzeichnen der durchschnittlichen Antwortzeit des Mediationsdienstes. Die Ergebnisse sind in Tabelle 8.1 aufgelistet.

Mediationstechnik	$T_{\varnothing}$
Upcast (`AppointmentLocation` → `Location`)	≈15 ms
Conversion (`kilometers` → `miles`)	≈14 ms
Conversion (für zwei äquivalente Klassen: `ont1:Hotel` → `ont2:Hotel`)	≈40 ms
Semantic split (`Location` → {`hasLat, hasLng`})	≈23 ms
Semantic join ({`hasLat, hasLng`} → `Location`)	≈13 ms
Partial substitution (`Location` → `Event.hasLocation`)	≈23 ms

Tabelle 8.1: Durchschnittliche Antwortzeit eines lokal ausgeführten Mediationsdienstes bei 100 Durchläufen pro Mediationstechnik

Die erzielten Ergebnisse sind vielversprechend und kaum zeitkritisch, insbesondere in Anbetracht der losen Kopplung von Komponenten. Zwar kann die noch hinzuzurechnende Netzwerkverzögerung verursacht durch SOAP über HTTP zu spürbaren Antwortzeiten führen. Technologisch kann dem jedoch entgegengesteuert werden, indem WebSockets [@WSoc] genutzt oder der Mediator komplett in ein hochoptimiertes MRE verlagert wird. Der Einfluss weiterer Faktoren wie der Größe und Komplexität von Ontologien – vor allem mit aktivem Reasoner – kann die Bearbeitungszeit zusätzlich erhöhen.

Einer der Hauptvorteile, den die Anwendung semantischer Datenmediationstechniken mit sich bringt, ist die deutlich erhöhte Koppelbarkeit von Komponentenschnittstellen. Die Interoperabilität basiert auf vielfältigen semantischen Zusammenhängen der zugrundeliegenden Ontologiekonzepte, anstatt auf rein syntaktischer Übereinstimmung wie in den meisten existierenden Mashup-Ansätzen oder der auf Vererbung beschränkten Semantik in CRUISE. Dadurch können Komponenten eher in unvorhergesehenen Konstellationen verknüpft werden, was insbesondere angesichts globaler Komponenten-Marktplätze vorteilhaft ist. Dies fördert wiederum die Wiederverwendung von Komponenten. Besonders im Rahmen von EUD durch Nicht-Programmierer vereinfachen die Automatismen das Erstellen von Anwendungen erheblich. Zudem können potenziell mehr Nischenanforderungen von Nutzern erfüllt werden, ohne spezielle neue Komponenten bereitstellen zu müssen. Wie die Ausführungen aufzeigen, ist es möglich, die konzipierten Mediationstechniken sowie die damit einhergehenden Konzepte zur Mapping-Discovery und Mapping-Execution zu implementieren und performant anzubieten. Zudem zeigen sich Mediationstechniken im Empfehlungssystem als zentral, vergleiche Kapitel 6.3.1 sowie 6.3.3, und spielen eine wichtige Rolle bei der Kommunikation mit Nicht-Programmierern, etwa bei dem Erzeugen von erklärenden Beschriftungen an diversen Stellen des Frontends, siehe Kapitel 7. Wie die Untersuchung der Werkzeuge in Kapitel 8.5 herausstellen wird, gelingt es, die Darstellung der inhärenten technischen Details und die Kompliziertheit der Verknüpfung heterogener Komponentenschnittstellen sowie der Definition von Abbildungsvorschriften Nutzern gegenüber auf ein Minimum zu reduzieren. Das Automatisieren von Mapping-Discovery und -Execution spielt hierbei eine zentrale Rolle. Die Praktikabilität des Ansatzes vor dem Hintergrund des EUD ist somit gegeben. Über diese Forschungsarbeit hinaus wird parallel an der Nutzung der vorgeschlagenen Techniken in kollaborativen Szenarien geforscht [Bli+15], etwa zur bedarfsgerechteren Synchronisation von Properties auf Basis von *Partial Substitution* und *Semantic Split*.

Grenzen der gewählten Ansätze stellen insbesondere die algorithmische Komplexität bei der *Mapping-Discovery* und *Mapping-Execution* sowie die Sicherstellung der Nützlichkeit der identifizierten Abbildungen dar. Letzteres ist für das Ableiten von Pattern-Instanzen eine Herausforderung, da potentiell eine Vielzahl von Kombinationsmöglichkeiten der Komponenten besteht. Die dafür entwickelten Lösungsansätze sind in Kapitel 6.3 erörtert. Komplexität entsteht durch die mögliche kombinatorische Explosion beim Vergleich von Komponentenschnittstellen sowie den darin referenzierten Ontologiekonzepten und durch die Möglichkeit, Mediationstechniken zu verknüpfen. Würden beispielsweise bei *Semantic Split* mehr als eine Ebene der OWL-Properties von Konzepten analysiert werden, würde die Koppelbarkeit zwar steigen, das Bestimmen und Ausführen von Mapping-Definitionen jedoch deutlich aufwändiger ausfallen. Neben der algorithmischen Komplexität besteht auch die Gefahr kognitiver Überlastung des Nutzers durch eine große Menge an Kombinationsmöglichkeiten und tiefe Schachtelung von Abbildungen. Zwar könnte argumentiert werden, dass dem mit geeigneten UI- und Empfehlungskonzepten

begegnet werden kann. Ob der möglicherweise resultierende Mehrwert, die entstehenden Aufwände rechtfertigt, ist allerdings fraglich. Vor diesem Hintergrund wurde die restriktive Definition von Äquivalenz bei Ontology-Mappings und die Begrenzung auf Tiefe eins der OWL-Properties untersuchter Ontologiekonzepte konzeptionell vorgesehen. Hinsichtlich der Korrektheit der identifizierten Abbildungsvorschriften hängt das Konzept von der Qualität des formalisierten Wissens ab. An dieser Stelle empfiehlt sich das konsequente Nutzen etablierter Ontologien und systematisches Ontology-Engineering.

Insgesamt bilden die Konzepte zur semantischen Datenmediation ein wichtiges Rückgrat einer Kompositionsplattform für Live-Sophistication. Anhand der zuvor beschriebenen prototypischen Erprobung und Evaluation ließ sich die Praktikabilität und Angemessenheit der Mediationskonzepte zeigen. Im Hinblick auf Anforderung 2 lässt sich deren vollständige Erfüllung attestieren. Automatisierung ist durch Mapping-Discovery und Mapping-Execution gewährleistet und performant. Mediationstechniken sind per Definition Bestandteil von semantischen Konnektoren, sodass jederzeit sämtliche benötigte Zielparameter vorliegen. Durch Kombination von Mediationstechniken können komplizierte Kommunikationsbeziehungen hergestellt werden, mit den genannten konzeptionellen Begrenzungen. Die Techniken sind semantikbasiert und decken die geforderten Fälle von Heterogenität vollständig ab. Mapping-Definitionen werden in wiederverwendbarer Form im Mapping-Repositorium hinterlegt. Die Umsetzung des Architekturkonzepts belegt dessen Eignung für EUD-Plattformen. Zudem konnten Indizien für die Eignung der Ansätze für Nicht-Programmierer gefunden werden, denen in längerfristig angelegten Studien weiter nachgegangen werden sollte.

8.3 Algorithmus zur Abschätzung von Capabilities

Dieses Kapitel widmet sich der Evaluation des Algorithmus zum Abschätzen der Capabilities von Kompositionsfragmenten aus Kapitel 5.3. Dazu wird ein Prototyp zum Nachweis der Umsetzbarkeit beschrieben und anschließend der Frage nach der Qualität der Ergebnisse durch eine Experten-Evaluation nachgegangen.

8.3.1 Prototypische Umsetzung

Angesichts der den Algorithmus nutzenden Systemkomponenten, siehe Kapitel 5.3.5, bietet sich als Umgebung für den Prototyp das existierende Repositorium der Plattform CRUISE an. Dieses beinhaltet die Funktionalität eines Komponenten- und Anwendungsrepositoriums und stellt Zugriff auf Empfehlungsmechanismen bereit. Darin wurde ein eigenständiges Modul bestehend aus Java-Paketen und -Klassen integriert. Zentraler Einstiegspunkt in das entwickelte Modul ist die Klasse `FunctionalAnalyzer`. Diese bietet im Wesentlichen eine Methode, die für ein gegebenes Kompositionsfragment eine Menge an Capabilities liefert. Hierzu delegiert der `FunctionalAnalyzer` Teilschritte an die zuständigen Klassen, die gemäß dem Konzept aus Kapitel 5.3.5 umgesetzt wurden. Der `FunctionalAnalyzer` wird auf Basis von Apache Axis2 [@Axis2] über eine SOAP-Schnittstelle für Laufzeitumgebungen per entferntem Zugriff erreichbar.

Die Klassen `EntityKnowledge` und `ActivityKnowledge` kapseln, wie konzeptionell vorgesehen, den Zugriff auf Domänenmodelle und die Action-Klassifikation. Zum Verwalten und Arbeiten mit semantischen Modellen wurde, wie bereits beim Mediator in Kapitel

8.2, Apache Jena verwendet. Dieses bietet vielfältige Implementierungen für verschiedene Reasoner. Aus Performanzgründen wird lediglich RDFS-Reasoning für ausgewählte Domänenmodelle durch `EntityKnowledge` unterstützt.

Auf Basis einer Vielzahl von Testfällen, welche unter Zuhilfenahme des Frameworks JUnit [@JUnit] entstanden, wurden verschiedenste Situationen abgedeckt, sodass die Erwartungskonformität der Ergebnisse umfangreich erprobt werden konnte. Entsprechend möglicher Fälle bei der Abschätzung der Capabilities von Kompositionsfragmenten wurden Tests für verschieden komplexe funktionale Zusammenhänge entwickelt. Nachfolgende Übersicht erläutert die dazu genutzten Kompositionsfragmente.

- Kompositionsfragmente mit einer einzelnen Komponente, zum Beispiel eine einzelne Karten- oder POI-Suchkomponente. Diese Tests prüfen insbesondere, ob Intra-Komponenten-Capabilities erwartungskonform bestimmt werden.
- Kompositionsfragmente mit zwei unverbundenen Komponenten wie beispielsweise einer Wikipedia- und einer SoundCloud-Komponente
- Kompositionsfragmente mit zwei Komponenten, die über einen einzelnen Kommunikationskanal verbunden sind. Diese Klasse von Kompositionsfragmenten umfasst Coupling-Patterns und einfache CWA, wie die nachfolgenden.
  - Wetter-App: Diese CWA umfasst eine Karte und ein Wetter-Widget. Sie ermöglicht das Anzeigen der Wetterprognose für die in der Karte gewählte Position. Dabei wurden teilweise Variationen der Verbindung ausgenutzt, um herauszufinden, ob die abgeleitete Funktionalität auf den oberen Hierarchieebenen trotzdem übereinstimmt. Im konkreten Beispiel bietet sowohl die Karte mehrere Optionen eine `Location` zu publizieren (Event, Property) als auch das Wetter-Widget eine solche entgegenzunehmen (Operation, Property), wobei jeweils der gleiche Senke-Capability-Knoten erreicht wird.
  - Nachrichten-App: Sie beinhaltet eine Komponente zum Auflisten von Nachrichten und eine zum Darstellen von Bildern. Bei Auswahl eines Nachrichteneintrags werden zugehörige Tags und Schlüsselworte zur Bildkomponente übermittelt, welche daraufhin von *flickr* passende Bilder abfragt und anzeigt.
- Kompositionsfragmente aus verschiedenen Anwendungsdomänen mit 3–8 Komponenten und 2–7 Kommunikationskanälen, wie die nachfolgenden Beispiele. Wie zuvor existieren teilweise strukturelle Varianten, um zu prüfen, ob auf oberster Ebene die gleiche Funktionalität abgleitet wird.
  - Reiseplaner I: Diese Anwendung besteht aus 7 Kommunikationskanälen und 7 Komponenten, die im Wesentlichen drei Funktionsblöcke bereitstellen: Routensuche, Wetteranzeige und Hotelsuche.
  - POI-Suche: Das Mashup umfasst 6 Komponenten und 4 Kommunikationskanäle. Es erlaubt dem Nutzer, POIs zu suchen, den Ort auf einer Karte zu visualisieren und weiterhin Wikipedia-Artikel zu einem POI anzuzeigen.
  - Routensuche: 4 Komponenten und 5 Kommunikationskanäle dienen zur Suche nach und Visualisierung von Routen.
  - Veranstaltungsplaner: Das Mashup erlaubt das Suchen und Verwalten von Terminen basierend auf 4 Komponenten und 2 Kommunikationskanälen.

– Hotelsuche: Ein Kompositionsfragment zur ortsbasierten Suche und Auflistung von Hotels. Es enthält 3 Komponenten und 2 Kommunikationskanäle.
– Reiseplaner II: Diese CWA weist zwei getrennte Capability-Ketten auf. Sie ist in Abbildung 8.3 dargestellt und ermöglicht es Routen zu suchen (Komponenten ①, ②, ⑤ und ⑦), das Wetter für den Zielort anzuzeigen (②, ⑥) und POIs zu suchen (③, ④ und ⑧).

Derartige Kompositionsfragmente dienen als Grundlage zur Betrachtung der Abarbeitungsgeschwindigkeit des konzipierten Algorithmus in Kapitel 8.3.1 sowie zur Validierung der Plausibilität der Ergebnisse im Rahmen einer Expertenevaluation, siehe Kapitel 8.3.2.

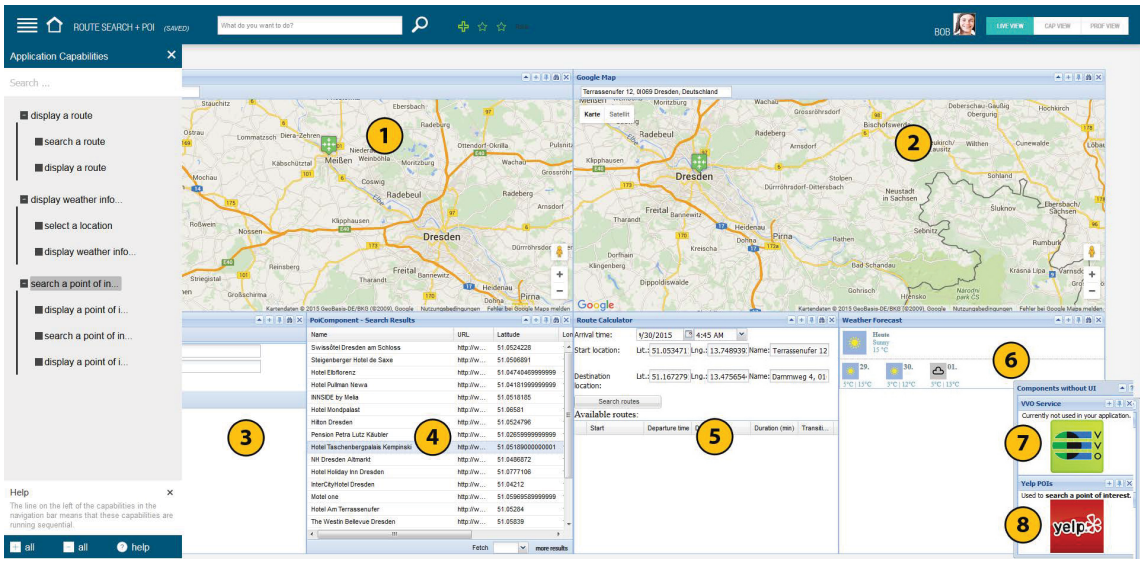


Abbildung 8.3: Screenshot einer CWA mit mehreren Capability-Ketten. Am linken Rand ist das Widget zur Anzeige der Capabilities der Anwendung zu sehen.

Verwendung des Prototyps des Algorithmus durch Architekturkomponenten

Die zuvor erläuterte prototypische Umsetzung wird, wie in Kapitel 5.3.5 beschrieben, durch diverse Architekturkomponenten genutzt. In diesem Kapitel werden diese umrissen, während Details zu diesen in den jeweiligen Abschnitten aus Kapitel 7 zu finden sind.

Im **Komponentenrepositorium** existieren mehrere Komponenten, die den Algorithmus zur Abschätzung der Funktionalität von Kompositionsfragmenten nutzen. Zunächst werden Capabilities von Patterns im Rahmen des Pattern-Mining abgeleitet. Konkret nutzen den Algorithmus Pattern-Miner wie der `SemanticCouplingMiner`, ein semantikbasierter Miner von Coupling-Patterns, und der `ComplexMiner`, welcher Complex-Pattern bereitstellt. Weiterhin tritt die Klasse `MetaDataIndexer` als Client auf. Sie bereitet Metainformationen zu existierenden Kompositionsmodellen im Anwendungsrepositorium auf und hält diese für einen effizienten Zugriff bereit. Dabei werden die Capabilities der Anwendung abgeleitet. Auch der Empfehlungsmanager nutzt indirekt die Ergebnisse des Algorithmus. Auf Implementierungsebene besitzt der Empfehlungsmanager einen serverseitigen Teil, der als SOAP-Webservice an ein MRE gebunden werden kann, siehe

Kapitel 8.4. Dieser `RecommendationService` stellt dabei alle notwendigen Empfehlungsmethoden bereit, die teilweise auch funktionale Anforderungen in Form von Capabilities als Parameter akzeptieren. Sind Capabilities teil der Anfrage werden diese gegen die an Pattern-Instanzen hinterlegten Capabilities geprüft.

In **Laufzeitumgebungen**, sowohl in der TSR als auch der CSR, wird der SOAP-Webservice des `FunctionalAnalyzer` angebunden, um Nutzungsszenarien des Algorithmus zu erproben. Folgende in Abschnitt 7 spezifizierten und prototypisch implementierten Werkzeuge nutzen die Ergebnisse des Webservice: das Empfehlungsmenü, die Erklärungstechniken, das Capability-Panel der Anwendung und der Wizard.

Performanzmessungen

In Experimenten wurde die Performanz der prototypischen Implementierung des Algorithmus zur Abschätzung von Capabilities gemessen, um dessen Nutzbarkeit im praktischen Einsatz zu belegen. Dazu wurde die durchschnittlich benötigte Berechnungszeit des Algorithmus für zunehmend komplexe Kompositionsfragmente bestimmt. Um softwaregeschuldete Abweichungen auszugleichen, zum Beispiel durch *Garbage-Collection* der *Java Virtual Machine*, wurden jeweils 100 Durchläufe in einem einzelnen Thread durchgeführt. In den Tests wurden lokale Aufrufe innerhalb der *Java Virtual Machine* abgesetzt, sodass in den Ergebnissen keine Netzwerkverzögerung enthalten ist. Tabelle 8.2 listet die gemessenen durchschnittlichen Berechnungszeiten.

Testfall	$N_{components}$	$N_{channels}$	$N_{caplinks}$	$T_{\varnothing}$
Nachrichten-App	2	1	1	205 ms
Veranstaltungsplaner	4	2	2	303 ms
Hotelsuche	3	2	2	250 ms
POI-Suche	6	4	7	422 ms
Reiseplaner II	8	7	12	526 ms

Tabelle 8.2: Ergebnisse der Performanzmessung. $T_{\varnothing}$ ist die durchschnittliche Zeit, die der Prototyp zur Berechnung der Capabilities für eine gegebene Komposition benötigt, und enthält das Deserialisieren von MCM- (≈100 ms) und SMCDL-Instanzen.

Die erzielten Ergebnisse verdeutlichen, dass erwartungskonform mit steigender Kompliziertheit des Kompositionsmodells die Berechnungszeit ansteigt. Die Werte sind jedoch wenig kritisch, da für keine der den Algorithmus nutzenden Architekturkomponenten (vergleiche Kapitel 5.3.5) eine Echtzeitfähigkeit mit besonders niedriger Zeitschranke relevant ist. Zudem kann durch Optimieren der Implementierung noch ein Senken der Berechnungszeit erreicht werden. Auch bei umfangreichen Anwendungen, wie in Zeile 5 von Tabelle 8.2, ist die Antwortzeit in einem akzeptablen Bereich, der im Kontext von Suchmaschinen einen Höchstwert von 3 Sekunden nicht überschreiten sollte [BHS08].

8.3.2 Experten-Evaluation

Um sowohl das Capability-Metamodell, siehe Kapitel 5.1.1, als auch den vorgeschlagenen Algorithmus, vergleiche Kapitel 5.3, zu validieren, wurde eine Expertenevaluation durchgeführt. Nachfolgend wird zunächst die angewendete Methodik genauer beschrieben. Anschließend erfolgt die Vorstellung und Diskussion der Ergebnisse.

Methodik

Der Teilnehmerkreis umfasste insgesamt sieben wissenschaftliche Mitarbeiter und Masterstudenten, die im Fachgebiet von Mashups oder serviceorientierten Architekturen forschen. Alle Teilnehmer besitzen tiefgreifende Kenntnis über das Erstellen sowie das Nutzen komponentenbasierter Anwendungen. Es wurden neun CWA mit zunehmender Kompliziertheit – diese gehören zur in Abschnitt 8.3.1 eingeführten Testumgebung, weisen zwei bis zu acht Komponenten auf und decken Anwendungsdomänen wie Reiseplanung, Unterhaltung und Terminverwaltung ab – auf Papier skizziert. Dabei wurden Komponenten, deren Capabilities und Capability-Links zwischen diesen schematisch präsentiert. Dieser Ansatz wurde gewählt, um im Vergleich zu in Ausführung befindlichen CWA die Übersichtlichkeit zu erhöhen und den Fokus auf die relevanten Capabilities der Komponenten zu lenken, sodass Fehlinterpretationen der Fähigkeiten vermieden oder vermindert werden können. Sofern notwendig, wurde eine theoretische Einführung in Konzepte der Aufgabenmodellierung und in das Capability-Metamodell gegeben. Anschließend wurden die Experten gebeten, sukzessive pro CWA die folgenden Fragen zu beantworten, indem Capability-Graphen auf Papier gezeichnet wurden.

F1: *Wie würden Sie die Gesamtfunktionalitäten dieser CWA mit den Mitteln des Capability-Metamodells beschreiben?*

F2: *Würden Sie diese Capabilities weiter dekomponieren? Falls ja, wie?*

Zusätzlichen wurden die Teilnehmer dazu angehalten, Erklärungen und Gedanken zu äußern, die vom Interviewer notiert wurden. Die vorrangige Zielsetzung der Studie war es, zu zeigen, dass das vorgestellte Capability-Modell gut zum Beschreiben von Funktionalitäten und Fähigkeiten geeignet ist, und dass der Algorithmus in der Lage ist, für ein gegebenes Kompositionsfragment adäquate Modellinstanzen abzuleiten. Dazu wurden die Erwartungswerte der Experten mit den Ergebnissen des Algorithmus verglichen.

Ergebnisse und Diskussion

Die Experten waren in nahezu allen Fällen in der Lage mit dem Capability-Metamodell zu modellieren, was sie ausdrücken wollten. Es kam vor, dass Activity oder Entity qualifiziert wurden, zum Beispiel »Search Hotel for Location«. Das kann einerseits grundsätzlich auf Activity-Modifier sowie Entity-Context abgebildet und andererseits als Frage des Erzeugens von Beschriftungen und weniger des Metamodells verstanden werden. Folgende Anmerkungen und Vorschläge wurden wiederholt geäußert. Beispielsweise kann eine Capability wie `Select Location` Quelle für mehrere Capability-Links sein oder es existieren mehrere Quell-Capabilities, gegebenenfalls in verschiedenen Komponenten. Mancher Experte merkte an, dass in letzterem Fall eine Unterscheidung der Capabilities möglich sein sollte, da verschiedene Instanzdaten, zum Beispiel von `Location` auftreten. Dieser Umstand wird im Capability-Metamodell implizit abgedeckt, indem Capabilities eine ID besitzen und Komponenteninstanzen zugeordnet sind. Daher besteht bereits die Möglichkeit, diese Differenzierung vorzunehmen, eine entsprechende Auswertung durch das Frontend unterstellt.

Bezüglich *F1* zeigen sich die Ergebnisse vielversprechend. Es wurde ein Übereinstimmungsgrad zwischen erwarteten und abgeleiteten Capabilities berechnet, wobei semantisch

ähnliche Konzepte, wie `Show` und `Display`, als 50 %-iger Treffer gewertet wurden. Weiterhin wurde bei der Auswertung die vergleichbarste Hierarchiestufe geprüft, sollte ein Experte beispielsweise eine flachere oder tiefere Hierarchie aufgestellt haben. Insgesamt wurden im Kontext der Testfälle eine Übereinstimmung von 96,83 % für *Entities* und von 80,16 % für *Activities* erreicht, was im Durchschnitt 88,5 % entspricht. Diese Werte sind vielversprechend. Sie lassen allerdings nur begrenzt verallgemeinernde Schlüsse zu, sodass in anderen Testumgebungen, die sich zum Beispiel hinsichtlich Komponenten, semantischen Annotationen und befragten Experten unterscheiden, eine abweichende Ergebnisgüte auftreten kann. Wie in den Ausführungen zur Methodik betont wurde, kamen Anwendungen aus verschiedenen Domänen und mit unterschiedlich kompliziertem Aufbau zum Einsatz. Somit kann bereits eine gewisse Bandbreite an Szenarien abgedeckt werden, weiterführende Untersuchungen scheinen jedoch angebracht, um die Validität der Konzepte weiter zu untermauern.

Hinsichtlich der Frage, ob Activitys der Klasse *Transform* oder *Output* die Funktionalität mehr beeinflussen, bestand kein Konsens. Tendenziell scheinen erstere als wichtiger erachtet zu werden (5 von 7 Teilnehmern), was die Heuristik des Algorithmus bestärkt. Treten mehrere Capabilities mit der gleichen Art von Activity in einer Sequenz auf, zum Beispiel `Search Song` → `Search Article`, haben 6 von 7 Experten die weiter hinten stehende Entity, im Beispiel `Article`, stärker gewichtet. Dies ist konsistent zum Ansatz des Algorithmus, welcher die Richtung des Datenflusses beachtet.

Da vordergründig die grundlegende Arbeitsweise und Heuristiken des Algorithmus validiert werden sollten, wurden in der Evaluation keine erlernten Daten einbezogen. Daher gelang es dem Algorithmus in einigen Fällen nicht, eine sinnvolle Capability auf Wurzelebene abzuleiten, während dies für Experten kein Problem darstellte. Beispielsweise errechnete er für die CWA in Abbildung 8.3 drei komposite Capabilities. Zwar decken sich diese mit den Erwartungswerten, jedoch haben 6 von 7 Experten zusätzlich einen Wurzelknoten `Plan Trip` oder sinngemäß definiert. Im Testfall »Terminplaner« schätzt der Algorithmus – basierend auf den semantischen Annotationen – `Edit` als Activity der Wurzel, während Experten oft ähnliche Begriffe vergaben, zum Beispiel `Manage` oder `Plan`, was auf Annahmen und zusätzlichem Wissen beruhte. Etwas derartiges algorithmisch abzuleiten gestaltet sich als hochgradig kompliziert. An dieser Stelle scheint eine Kombination aus kollektivem und semantisch beschriebenem Wissen der erfolgversprechendste Ansatz. Jedoch hilft semantisches Wissen bereits dabei, einen Kaltstart zu vermeiden, und ist im Stande, plausible Resultate zu erzielen.

Die Analyse der Antworten auf Frage *F2* verdeutlichen, dass die von Experten aufgezeichneten Capability-Graphen prinzipiell ähnlich zu den vom Algorithmus berechneten ausfallen. Es wird jedoch deutlich, dass die Teilnehmer dazu tendierten, Capabilities zu subsumieren und wegzulassen. Beispielsweise sagten Experten aus, dass es für sie klar sei, dass etwas zu suchen (`Search`) impliziert, dass zuvor die Suchkriterien eingegeben (`Input`) werden. Angesichts der Vielzahl an Anwendungsfällen für die Ergebnisse des Algorithmus behält letzterer alle Capabilities bei und es obliegt einem Client, derartige Filterungen bei Bedarf durchzuführen. Im kompliziertesten Szenario fiel es Experten schwer, die Strukturierung des Graphen bis zu den Blättern vorzunehmen. Die oberen Ebenen bis etwa Tiefe drei stellten hingegen kein Problem dar. Dies untermauert die Notwendigkeit eines automatisierten Ansatzes. Weiterhin war zu beobachten, dass Substrukturierung sehr ähnlich zum vorgestellten Konzept angewendet wurde, wenn auch nicht in jedem Fall, in dem der Algorithmus dies vorsieht. Dies ändert jedoch weniger

etwas an der resultierenden Semantik als mehr an der Hierarchietiefe. Bezüglich der Wichtigkeit von Capabilities auf Komponentenebene, die nicht in einem Capability-Link eingebunden sind, gingen die Meinungen auseinander. Einige Experten ignorierten diese komplett, während andere sie subsumierten oder in eine extra komposite Capability mit Activity `Display` oder ähnlich gruppierten. Generell war zu bemerken, dass Experten von Erfahrungen mit Webanwendungen beeinflusst wurden und deshalb abhängig vom Komponentennamen Funktionalitäten annahmen, selbst wenn kein passendes Pendant zu sehen war. Dies betrifft auch nicht vorhandene Capability-Links, die teilweise als gegeben angenommen wurden. Dies untermauert die kritische Rolle sorgfältiger semantischer Auszeichnung von Komponenten in der SMCDL. Da das Annotieren eine aufwändige und fehleranfällige Aufgabe darstellt, müssen Komponentenentwickler durch geeignete Werkzeuge und Richtlinien unterstützt werden. Auch die Qualität von Ontologien hat direkte und wesentliche Auswirkungen auf die Ergebnisse. Daher sollten gut strukturierte, bewährte und etablierte Vokabulare verwendet werden. Dennoch bieten semantische Annotationen eine Fülle an Vorteilen. Zudem erlaubt der vorgestellte Algorithmus, die Annotation von Kompositionsfragmenten und Anwendungen hinsichtlich Capabilities zu automatisieren, ohne dass Entwickler oder Nutzer aktiv werden müssen.

Die Evaluation zeigt, dass das vorgestellte Konzept tragbar sowie praktikabel ist und weiterhin Anforderung 7.4 und die Kriterien aus Abschnitt 5.3.1 vollständig abdeckt.

8.4 Umsetzung des Empfehlungskreislaufes

Die Empfehlungsinfrastruktur weist sowohl client- als auch serverseitige Anteile auf. Die Laufzeitumgebungen TSR und CSR wurden dahingehend erweitert, dass sie zunächst die Triggerkonzepte umsetzen. Die zu verwendenden Trigger werden dabei anhand der URI ihrer Beschreibungsdatei per JavaScript (TSR) oder Java (CSR) deklariert. Dazu wurde das konzipierte XML-Schema implementiert und vom `TriggerManager` ausgewertet. Somit kann die Anpassbarkeit des Empfehlungssystems erreicht werden. Allerdings wird im Prototyp von einer festen Menge an Empfehlungsmethoden und Recommendation-Viewern ausgegangen und diese sind fest zu Empfehlungsstrategien verknüpft. Sämtliche Triggerarten konnten erprobt werden, wie in nachfolgender Beschreibung deutlich wird.

- **Search issued** reagiert auf Eingaben in die Stichwortsuche, vergleiche Kapitel 7.1.
- **Unwired event** löst aus, falls ein aufgetretenes Komponenten-Event noch von keiner anderen Komponente empfangen wird.
- **Cap selection** erfordert, dass ein Nutzer eine Repräsentation in der Capability-View selektiert, siehe Kapitel 7.3.
- **Unavailable service** löst aus, wenn das MRE-Event `serviceUnavailable` mehrere fehlgeschlagene Webservice-Aufrufe einer Komponente signalisiert.
- **Auto-completion** prüft alle 30 Sekunden, ob die aktuelle Komposition geändert wurde und mindestens zwei Komponenten enthält, und löst in diesem Fall aus.
- **Explicit completion** triggert, falls ein Nutzer explizit Vorschläge zu Komplettierungen anfragt. Die TSR bietet dazu einen dedizierten Dialog, in dem Komponenten selektiert werden können, für die Empfehlungen benötigt werden.

- **Dashboard recs required** löst aus, falls der Startbildschirm einen Empfehlungsbereich umfasst und dieser initialisiert wird.
- **Component reqs failed** reagiert auf `contextChanged`-Events und testet, ob die Kontextänderung dazu führt, dass Komponentenanforderungen nicht mehr erfüllt sind. Falls ja, löst der Trigger aus.

Insgesamt konnte anhand der prototypischen Umsetzung von acht Triggern die Anwendbarkeit der erarbeiteten Konzepte an praxisnahen Beispielen verdeutlicht werden. Die Praktikabilität sämtliche Trigger-Klassen konnte erprobt und aufgezeigt werden. Die Abhängigkeit bereitgestellter Trigger bezüglich zugehöriger Empfehlungsmethoden und Recommendation-Viewer kann zwar durch die definierten Schnittstellen beherrscht werden, erfordert jedoch genaue Kenntnis über verfügbare Konfigurationsparameter seitens des Trigger-Entwicklers. Es konnte gezeigt werden, dass ein MRE hinsichtlich der zu nutzenden Trigger konfiguriert werden kann und dass die dynamische, späte Bindung von Triggern auf Basis der deklarativen Beschreibung, losen Kopplung und Schnittstellendefinitionen realisierbar ist. Dies trägt zur angestrebten domänenspezifischen Konfigurierbarkeit einer Kompositionsplattform bei. Die Verwendung des deklarativen Teils der Triggerbeschreibungen sowohl in TSR als auch CSR illustriert die Wiederverwendbarkeit. Die Implementierungsteile von Triggern, vergleiche Kapitel 6.3.2, sind in den entwickelten Prototypen von TSR und CSR aufgrund von Unterschieden in Klassenstrukturen und weiterer Details der MRE-Umsetzungen nicht unmittelbar wiederverwendbar.

Listing 8.1 zeigt die Standardkonfiguration in JSON für das Handhaben einer Triggerausgabe, falls sich für diesen Trigger noch Recommendation-Jobs im Empfehlungskreislauf befinden. Wie in den Zeilen 2 und 3 zu erkennen ist, wird dabei unterschieden, ob die Triggerausgaben gleich gemäß der in Kapitel 6.3 genannten Äquivalenzrelation sind oder ob lediglich der selbe Trigger ausgelöst hat. Zulässige Werte sind in beiden Fällen `IGNORE`, `REPLACE` und `ADD` entsprechend den drei Optionen aus der Spezifikation der Trigger-Ausgabedatenstruktur in Kapitel 6.3.2

```
1 {  "handleRecurrence": {
2     "inCaseOfEquality": "IGNORE",
3     "otherwise": "ADD" }, ...}
```

Listing 8.1: Auszug einer Konfiguration des Empfehlungskreislaufs als Teil einer Triggerausgabe

Nachdem ein Trigger auslöst, erfolgt das Filtern und Sortieren der Recommendation-Jobs in einer Warteschlange unter Anwendung dieser Konfiguration. Auf Clientseite implementiert der `RecommendationManager` die Abarbeitung eines `RecommendationJob` anhand von `Handler`-Klassen, deren Kontroll- und Datenfluss er koordiniert. Sie verfügen über eine fest definierte, eventbasierte Schnittstelle, um das Verlassen einer Phase zu signalisieren. `Handler` benutzen den `RecommendationJob`, um Daten darin abzulegen sowie zum Austausch von Zustandsinformationen. Die Klasse `QueryPhaseHandler` organisiert Aktivität ② *Empfehlungen berechnen*, vergleiche Kapitel 6, unter Zuhilfenahme des *Empfehlungsdienstes*. Welche Empfehlungsmethoden für den aktuellen Recommendation-Job relevant sind, entnimmt er der Triggerbeschreibung. Außerdem ist er dafür zuständig, die Anfrage aus der in der Triggerausgabe enthaltenen Konfiguration und ggf. aus weiteren, generischen Parameter zu formulieren. Weiterhin sorgt der `QueryPhaseHandler` dafür, die Ergebnisse in ein normiertes JSON-Format zu transformieren. Der `DisplayPhaseHandler` ordnet den Ergebnissen jeder Empfehlungsmethode die passenden Recommendation-Viewer zu, liest die optional vorhandene Konfiguration aus der Triggerausgabe aus

Name	Pattern-Klasse	Beschreibung
ComplexMiner	Complex	Konvertiert MCM zu Complex-Pattern und umfasst vordefinierte Patterns.
ExchangeabilityMiner	Exchangeability	Bestimmt austauschbare Komponenten gemäß einer modifizierten Variante von [PRM11b].
SemanticCouplingMiner	Coupling	Untersucht sämtliche Komponenten hinsichtlich semantisch kompatibler Schnittstellenelemente.
SemanticComplexPattern-Materializer	Complex	Erstellt neue aus bestehenden Complex-Pattern, indem Komponenten gemäß vorhandener Exchangeability-Patterns ausgetauscht werden.

Tabelle 8.3: Prototypisch umgesetzte Pattern-Miner

und delegiert das Darstellen der Ergebnisse an die Recommendation-Viewer. Folgende Recommendation-Viewer wurden prototypisch in CSR und TSR implementiert:

- Die **Ergebnisliste** zeigt relevanzsortierte Ergebnisse auf dem Startbildschirm an, die semantisch zu eingegebenen Stichworten passen.
- Der **Startbildschirm** umfasst mehrere Empfehlungsbereiche, siehe Kapitel 7.1.
- Der **Wizard** präsentiert passende Kompositionsfragmente im Ergebnisbereich.
- Die **Capability-View** aus Kapitel 7.3 stellt empfohlene Patterns in Form von hervorgehobenen Ports und mittels Beschriftungen dar.
- Das **Empfehlungsmenü** visualisiert empfohlene Patterns, siehe Kapitel 7.5.3.

Falls eine Auswahl einer Empfehlung vorliegt, wird dies erfasst und mit den notwendigen Kontextinformationen an den Feedback-Dienst übermittelt. Anschließend übernimmt der `IntegrationPhaseHandler` das Hinzufügen des Kompositionsfragments gemäß dem Konzept aus Kapitel 6.3.5. In einem dedizierten Bewertungsdialog können Nutzer anhand einer Sterne-Metapher Komponenten, Anwendungen und angenommene Empfehlungen bewerten. Zunächst wird nur die Eingabe von textuellem Feedback und der Gesamtbewertung eingefordert. Weitere Bewertungskriterien sind optional einblendbar.

Auf Serverseite wurden das Komponenten- und das Anwendungsrepositorium erweitert. Der `RepositoryManager` als zentrale Klasse koordiniert die Instanziierung und Konfiguration des prototypischen Anwendungsrepositoriums. Dazu zählen neben den nachfolgend beschriebenen Bestandteilen das `IPatternStore`, das für die Persistenz von Patterns (implementiert als Java-Objektmodell) zuständig ist. Der `MiningManager` orchestriert die Mining-Infrastruktur und macht diese zugänglich. Analysemodule, die von `AbstractMiner` und `Thread` erben, folgen dem Entwurfsmuster Observer [Gam+94], um auf Änderungen der Datenbasis in den Repositorien zu reagieren. Wie in Kapitel 1.2 verdeutlicht, zielt diese Arbeit nicht darauf ab, neuartige Analyse-Algorithmen zu erforschen. Daher wurden lediglich simple Pattern-Miner implementiert, die in Tabelle 8.3 erläutert werden. Dem Konzept entsprechend nutzen diese sowohl semantische als auch community-basierte Ansätze. Die zu nutzenden Pattern-Miner können deklarativ konfiguriert werden und werden entsprechend vom `MiningManager` instantiiert und verwaltet. Der Prozess des Pattern-Mining kann durch mehrere Modi gesteuert werden: *initial* zur initialen Analyse, *full* zur zusätzlichen reaktiven Analyse und *off* zur Deaktivierung. Ein `ConsistencyChecker` setzt Mechanismen zur Konsistenzsicherung um, indem er auf

Änderungen der Datenbasis reagiert und pro Pattern-Instanz prüft, ob die benötigten Komponenten und Mapping-Definitionen existieren. Die Klasse `RecommendationManager` setzt das Konzept des Empfehlungsmanagers um und bildet das MRE-unabhängige Gegenstück zu den clientseitigen Empfehlungsmanagern. Er stellt im Prototyp folgende Menge an Empfehlungsmethoden bereit.

- **getCompositionFragments** liefert Komponenten und Anwendungen, die mit gegebenen Kriterien am besten übereinstimmen. Für Freitextanforderungen wird geprüft, ob diese sich auf Capabilities abbilden lassen. Dabei kommt unter Einbeziehung von WordNet lexikalisches und semantisches Wissen zum Einsatz. Gibt der Nutzer zum Beispiel »show position« ein, führt der Abgleich mit Ontologiekonzepten dazu, dass die Capability `Display Location` identifiziert wird.
- **getPossibleCouplings** und **getPossibleComplexPatterns** geben Coupling beziehungsweise Complex-Patterns für eine gegebene Komponente sowie für gegebene Schnittstellenelemente dieser zurück.
- **recommendAlternatives** beantwortet Anfragen nach Komponenten, die eine gegebene Komponente ersetzen können.
- **getCompletionPatterns** schlägt Coupling und Complex-Patterns vor, die einen gegebenen Kompositionskontext sinnvoll erweitern.
- **getRecommendedCompositionFragments** berechnet Kompositionsfragmente, die für einen Nutzer relevant sein können. Dabei werden mehrere, verschiedene Arten von Relevanz berücksichtigende Listen kalkuliert. Dies basiert maßgeblich auf den Feedback-Einträgen im Nutzerprofil und auf dem semantischen Vergleich von Komponenten und Anwendungen.
- **recommendComponentsByEvents** und **recommendComponentsByOperations** berechnen Coupling- und Complex-Patterns für eine gegebene Komponente und für eine Menge von Events beziehungsweise Operations dieser.
- **getSimilarFragments** bestimmt für gegebene Komponenten und Anwendungen ähnliche Kompositionsfragmente. Im Fall von Komponenten erfolgt dies anhand des semantischen Vergleichs von Capabilities. Bei Anwendungen wird zusätzlich die Übereinstimmungen der darin befindlichen Komponenten einbezogen.
- **getFrecencyFragments** liefert Komponenten und Anwendungen des Nutzers gewichtet nach dem Konzept der *Frecency* aus Kapitel 6.3.3.

Die Abarbeitung von Matching und Ranking basiert auf Implementierungen der Klasse `AbstractMatcher` für die jeweilige Kandidatenart. Für das Ranking verwendet der `RecommendationManager` im Prototyp eine fest definierte `RankingStrategy`-Instanz, die Gewichte und Schwellwerte kodiert und $rating_{final}$ berechnet.

Der `FeedbackManager` kapselt Funktionen zum Verwalten von Feedback gemäß dem Modell aus Kapitel 5.1.4, das als Java-Objektmodell umgesetzt wurde. Pro Nutzer pflegt er dazu eine Kollektion von Feedback-Einträgen für jedes *Feedback-Subjekt*. Zudem berechnet er Auflistungen anhand der *Frecency*.

Die Empfehlungsinfrastruktur bietet zwei Webservice-Schnittstellen zu einem MRE via SOAP: den Feedback-Dienst und den Empfehlungsdienst. Zu Administrationszwecken

Kenngröße	Datensatz 1	Datensatz 2
Anzahl von Komponenten	64	129
Anzahl von Anwendungen	20	39
Anzahl von Patterns	2937	21627
Größe des Capability-Wissensgraphen	298390 Tripel	nicht genutzt
Nutzerprofile und Anzahl Feedback-Einträge	Bob: 131 Einträge, Alice: 0, Charlie: 3	
Semantische Modelle	28 Ontologien genutzt	

Tabelle 8.4: Wesentliche Kenngrößen der verwendeten Datensätze

Empfehlungsmethode	$T_{\varnothing}$ (Datensatz 1)	$T_{\varnothing}$ (Datensatz 2)
getCompletionPatterns (Anfrage 1)	394 ms	1074 ms
getCompletionPatterns (Anfrage 2)	113 ms	368 ms
getCompositionFragments (Anfrage 1)	11 ms	19 ms
getCompositionFragments (Anfrage 2)	67 ms	99 ms
getPossibleCouplings	10 ms	26 ms
getPossibleComplexPatterns	6 ms	11 ms
getCuppleableComponentsByOperations	3 ms	3 ms
getCuppleableComponentsByEvents	10 ms	19 ms
getRecosForUser	541 ms	1146 ms
getSimilarFragments (Anfrage 1)	54 ms	108 ms
getSimilarFragments (Anfrage 2)	5 ms	8 ms
getFrecencyFragments	7 ms	7 ms

Tabelle 8.5: Durchschnittliche Antwortzeiten des Empfehlungsdienstes für Nutzer Bob

wurde eine Webanwendung entwickelt, die Aufschluss über den aktuellen Status der Empfehlungsinfrastruktur gibt und diese von außen steuern lässt.

8.4.1 Performanzbetrachtungen

Um die Praktikabilität von neun genutzten Empfehlungsmethoden zu belegen, fand eine Performanzmessung statt. Dazu wurde die durchschnittliche Antwortzeit des Empfehlungsdienstes bestimmt. In den Tests wurden lokale Anfragen via SOAP-UI abgesetzt, sodass in den Ergebnissen keine Netzwerkverzögerung enthalten ist. Ein Testdurchgang bestand aus 30 Durchläufen. Jeder Durchlauf beinhaltet 12 Anfragen aus einem einzelnen Thread. Jeder Testdurchgang erfolgte in einer neu gestarteten JVM mit zirka 5 Durchläufen pro Anfrage »zur Aufwärmung«. Die Testdurchgänge unterscheiden sich hinsichtlich des Nutzers oder des Datenbestands. Tabelle 8.4 führt die beiden Datensätze auf. Am Beispiel von Datensatz 1 sollen anhand ausgewählter Metriken die Kombinationsmöglichkeiten und somit die Praxisrelevanz dargelegt werden. Die 64 Komponenten weisen im Schnitt 1,28 (0–8) Operationen, 1,1 (0–6) Events, 2,84 (0–6) Properties und 4,63 (0–20) Capabilities auf. Die 20 Anwendungen bestehen durchschnittlich aus 4,25 (2–8) Komponenten und 3,75 (1–9) Kommunikationskanälen. Datensatz 2 wurde aus dem ersten durch Duplizierung von Komponenten und Anwendungen synthetisiert.

Zunächst wurden die Antwortzeiten für Datensatz 1 bestimmt. Wie Tabelle 8.5 veranschaulicht, bewegen sich die Werte durchweg deutlich unterhalb einer Sekunde. Insbesondere Methoden, die durch explizite Trigger angesteuert werden, können mit Zeiten von unter 100 ms aufwarten. Dies kann darauf zurückgeführt werden, dass meist eine lineare Komplexität in der Anzahl der jeweiligen Kandidaten vorliegt und dass Pattern-Instanzen im Prototyp vorberechnet werden. Weiterhin wurde untersucht, inwiefern die Größe der

zugrundeliegenden Datenbasis die Antwortzeiten beeinflusst. Angesichts der Komplexität der Empfehlungsmethoden, die linear zur Anzahl der zu untersuchenden Kandidaten steigt, ist davon auszugehen, dass die durchschnittlichen Antwortzeiten für Datensatz 2 durchgehend über denen von Datensatz 1 liegen. Die in Tabelle 8.5 gezeigten Resultate bestätigen dies. Auch die Kompliziertheit der Eingabeparameter verursacht klare Differenzen. Beispielsweise unterscheiden sich die beiden Anfragen zu *getCompletionPatterns* im übergebenen Kompositionsfragment (Zeile 1 in Tabelle 8.5: 4 Komponenten, 4 Kanäle, 3 Capabilities; Zeile 2: 2 Komponenten, 1 Kanal, 1 Capability).

Ein Vergleich der Antwortzeiten für Datensatz 1 in Abhängigkeit der drei Nutzerprofilen zeigt, dass die Anzahl der Einträge einer Nutzerhistorie einen Einfluss auf die Berechnungszeit ausübt. Bei der Mehrheit der implementierten Methoden treten allerdings nur geringe Unterschiede auf, da die Historie zur Kalkulation von $rating_{usage}$, siehe Kapitel 6.3.3, verwendet wird (lineare Komplexität in Abhängigkeit der Anzahl der Einträge). Dies verdeutlicht die Praktikabilität des Feedback-Metamodells und des gewählten Verfahrens zur Einbeziehung von Feedback in das Ranking. Besonders sticht allerdings die Anfrage *getRecosForUser* hervor, bei der für Bob 540,8 ms, für Alice 4,06 ms und für Charlie 108,9 ms gemessen wurden. Dies ist der Funktionsweise der Methode geschuldet, die auch vorsieht, pro Feedback-Subjekt inhaltsbasiert ähnliche Komponenten und Anwendungen zu identifizieren. Daher sind hier bei steigender Anzahl an Feedback-Einträgen, Komponenten und Anwendungen höhere Berechnungszeiten zu erwarten. Allerdings können hier Maßnahmen wie Vorberechnungen ergriffen werden. Insgesamt sind die erzielten Antwortzeiten als unbedenklich und akzeptabel einzuschätzen, vergleiche Argumentation in Abschnitt 8.3.1. Längere Antwortzeiten treten zwar auf, allerdings werden die zugehörigen Empfehlungsmethoden durch implizite Trigger angestoßen, sodass für Nutzer keine beeinträchtigende Wartezeit entsteht. Als Einschränkung des Versuchsaufbaus muss der Umfang der Datensätze genannt werden, der beispielsweise im Vergleich zu `programmableweb.org` gering ausfällt. Allerdings können aufgrund der linearen Komplexität der implementierten Empfehlungsmethoden auch bei einer um Faktor 100 vergrößerten Datenbasis akzeptable Antwortzeiten, das heißt im Kontext von Suchmaschinen kleiner als 3 Sekunden [BHS08], erzielt werden.

8.4.2 Evaluation und Diskussion

Basierend auf den oben genannten implementierten Triggern, Empfehlungsmethoden und Recommendation-Viewern sowie unter Beachtung der Szenarien aus Kapitel 2.2 konnten mehrere Empfehlungsstrategien für Nicht-Programmierer entwickelt werden. Jede Zeile in Tabelle 8.6 repräsentiert eine Empfehlungsstrategie.

Um die Diversität der implementierten Empfehlungsstrategien zu verdeutlichen, werden nachfolgend ausgewählte erläutert. Zum Beispiel sieht Empfehlungsstrategie 1 vor, dass sobald ein Nutzer eine Suche nach Kompositionsfragmenten auslöst, zum Beispiel über die Stichwortsuche, siehe Kapitel 7.1, zu den eingegebenen Kriterien passende Kompositionsfragmente berechnet und in der Ergebnisliste, vergleiche Kapitel 7.1, dargestellt werden. Bei Strategie 2 werden für ein auftretendes Komponentenevent, das noch nicht mit einem Kommunikationskanal verbunden ist, verknüpfbare Komponenten anhand von Coupling- und Complex-Patterns vorgeschlagen und im Empfehlungsmenü visualisiert. Empfehlungsstrategie 3 legt fest, dass bei Auswahl einer Repräsentation in der Capability-View, vergleiche Kapitel 7.3, anhand mehrerer Empfehlungsmethoden Coupling- und

	Trigger-ID	Empfehlungsmethoden	Rec.-Viewer
1	Search issued	`getCompositionFragments`	Ergebnisliste
2	Unwired event	`recommendComponentsByEvents`, `getPossibleComplexPatterns`	Empfehlungsmenü
3	Cap selection	`getPossibleCouplings`, `getPossibleComplexPatterns`, `recommendComponentsByOperations`, `recommendComponentsByEvents`	CapView, Empfehlungsmenü
4	Unavailable service	`recommendAlternatives`	Empfehlungsmenü
5	Auto-completion	`getCompletionPatterns`	Empfehlungsmenü
6	Dashboard recs required	`getRecommendedCompositionFragments`	Startbildschirm
7	Component reqs failed	`recommendAlternatives`	Empfehlungsmenü
8	Explicit completion	`recommendComponentsByOperations`, `recommendComponentsByEvents`, `recommendAlternatives`, `getCompletionPatterns`	Empfehlungsmenü

Tabelle 8.6: Abstrakte Beschreibung implementierter Empfehlungsstrategien

Complex-Patterns berechnet werden, die Verknüpfungen mit der gewählten Komponente und dem entsprechenden Schnittstellenelement bereitstellen. Das Visualisieren von Pattern-Instanzen erfolgt dabei in der CapView, wie in Kapitel 7.3 beschrieben, und im Empfehlungsmenü für Pattern-Instanzen mit Komponenten, die noch nicht Teil der Anwendung sind. Empfehlungsstrategie 4 definiert, dass nach mehrfach fehlgeschlagenen Aufrufen von Webservices einer Komponente nach austauschbaren Komponenten mittels Exchangeability-Patterns gesucht wird und diese im Empfehlungsmenü präsentiert werden sollen. Die Strategie in Zeile 5 umfasst den proaktiven Trigger *Auto-completion*, der zeitgesteuert auslöst, woraufhin Coupling- und Complex-Patterns, die die Anwendung erweitern können, bestimmt und im Empfehlungsmenü angezeigt werden.

Diese Strategien decken eine hohe Bandbreite an proaktiven und reaktiven Empfehlungsszenarien ab. Durch das Erproben der Strategien im Prototyp konnte deren Plausibilität festgestellt werden. Allerdings sollte die wahrgenommene Nützlichkeit von Nicht-Programmieren im Rahmen von Nutzerstudien validiert werden, zum Beispiel mittels A/B-Tests, bei denen eine Gruppe mit, die andere ohne zu evaluierende Strategien Aufgaben lösen. Darüber hinaus ist davon auszugehen, dass die wahrgenommene Nützlichkeit stark von den gelieferten Empfehlungen und deren Präsentation abhängt. Da dies von einer Vielzahl von Faktoren beeinflusst wird, wie den vorhandenen Komponenten, der Qualität von Annotationen, den genutzten Algorithmen, der Nutzerhistorie und subjektiven Erwartungen, scheint eine derartige Evaluation bereits ausreichend für eine nachfolgende Forschungsarbeit. Trotz dieser Limitierungen wird aus den umgesetzten Strategien die Umsetzbarkeit und Eignung der vorgeschlagenen Konzepte ersichtlich. Auf Grundlage einer modularen Architektur sowie dem neuartigen Ansatz von Empfehlungsstrategien kann das Empfehlungssystem dem hochgradig iterativen Ad-hoc-Charakter der CWA-Entwicklung sowie den Anforderungen der hier betrachteten Zielgruppe gerecht werden und die Bandbreite empfehlungsrelevanter Szenarien bewältigen. Konzeptionell können Trigger, Empfehlungsmethoden und Recommendation-Viewer dynamisch eingebunden, konfiguriert und kombiniert werden, sodass das Empfehlungssystem hochgradig anpassbar an domänen- und plattformspezifische Gegebenheiten ist. Dies gestaltet die Definition von Empfehlungsstrategien als herausfordernd, da passende Module identifiziert, konfiguriert sowie sinnvoll kombiniert werden müssen und da geeignete Kontextbedingungen für die

Strategie zu formulieren sind. Bei der Entwicklung neuer Module besteht stets eine Abhängigkeit von den vorhandenen Modulen. Dem gilt es zukünftig durch geeignete Autorenwerkzeuge zu begegnen. Der Prototyp zeigt Beschränkungen. Zum Beispiel wurde lediglich mit dynamisch integrierbaren Triggern experimentiert, während eine feste Menge an Empfehlungsmethoden und Recommendation-Viewern angeboten wurde. Weiterhin fehlt die Auswertung von Kontextbedingungen für Empfehlungsstrategien, was allerdings durch die Anbindung an den Kontextdienst gut umsetzbar ist.

Die vorgeschlagenen, validierten Konzepte erfüllen Anforderung 8 und gehen sogar stellenweise darüber hinaus. Geschaffen wurde ein Architekturkonzept für ein hochgradig anpassbares Empfehlungssystem für komponentenbasierte Kompositionsplattformen. Auf Basis von kontextsensitiv auswählbaren Empfehlungsstrategien gelingt es, die komplette Nutzung und Entwicklung von CWA proaktiv und reaktiv zu unterstützen. Empfehlungsmethoden stehen Datenquellen zur Verfügung, mit denen hybride und den aktuellen Kontext berücksichtigende Algorithmen realisierbar sind. Dies belegen die zahlreichen implementierten Beispiele. Die neuartige funktionale Relevanz unter Zuhilfenahme des Capability-Wissensgraphen ermöglicht es, im Zusammenspiel mit strukturellem, semantischem und statistischem Kompositionswissen, relevante Ergebnisse zu generieren. Allerdings stand die Konzeption und Validierung neuer Empfehlungsmethoden nicht im Fokus der Arbeit. Vielmehr wurde ein Rahmenwerk geschaffen, in das diese eingefügt werden können. Empfehlungen zu Komponenten, Anwendungen und Kompositionsschritten werden dem Nutzer in konfigurierbarer Weise in Recommendation-Viewern präsentiert, die Mechanismen zur Schaffung von Nachvollziehbarkeit aufweisen. Inwiefern dies erreicht wurde, klärt der nachfolgende Abschnitt. Trigger liefern die notwendigen Informationen und somit fundamentale Voraussetzungen. Das Konzept sieht vor, dass unter Zuhilfenahme einer Adaptionsinfrastruktur die Empfehlungen nach Nutzerauswahl automatisch integriert werden können. Die transaktionsorientierte Integration von Kompositionsfragmenten sichert valide Zustände und fördert die Undo/Redo-Funktion. Schließlich ist vorgesehen, dass von Recommendation-Viewern gesammeltes Feedback im Feedback-Repositorium hinterlegt wird und somit Empfehlungsmethoden als Datenquelle bereitsteht. Die erfolgreiche Implementierung unterstreicht die Tragfähigkeit der Konzepte.

8.5 Evaluation von EUD-Werkzeugen

Zentraler Bestandteil einer Plattform für das assistierte EUD von Mashups durch Nicht-Programmierer sind geeignete Werkzeuge. Den Schwerpunkt dieses Kapitels bildet die Validierung der in Abschnitt 7 konzipierten Ansätze durch Prototypen und Nutzerstudien.

Im Rahmen der CSR wurde die Konzeption des Startbildschirms in vollem Umfang implementiert. Abbildung 8.4 zeigt im oberen Teil den initialen Zustand. Standardmäßig wird die Stichwortsuche eingeblendet ① und via ② öffnet sich der Wizard. In ③ werden Kompositionsfragmente angezeigt und ④ umfasst die Kompositionsfragmente des Nutzers gemäß *Frecency*, vergleiche Kapitel 6.3.3, wobei die Empfehlungsmethode `getFrecencyFragments` zum Einsatz kommt. Sobald der Nutzer Stichworte eingibt, wird die Empfehlungsstrategie aus Zeile 1 in Tabelle 8.6 befolgt und in ⑤ werden Ergebnisse aufgelistet, welche die Anforderungen erfüllen. Balken unterhalb einzelner Ergebnisse deuten den Übereinstimmungsgrad anhand der Farbe und der Länge an. Empfehlungen für den Nutzer beinhaltet ⑥ entsprechend der Strategie in Zeile 6 der Tabelle 8.6.

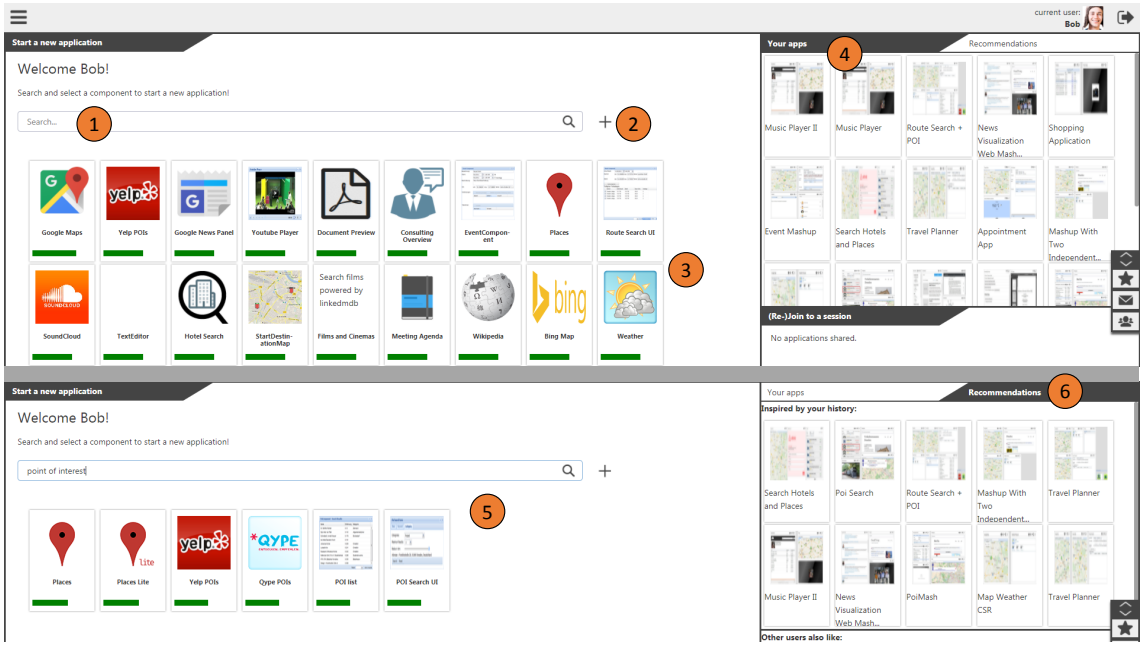


Abbildung 8.4: Prototypisch umgesetzter Startbildschirm

Abbildung 8.5 zeigt die Live-View mit einer CWA. Die Menüleiste bietet Zugang zur Stichwortsuche ①, zum Wizard ②, zu den Erklärungstechniken ③ und zur CapView ④. Weiterhin existiert das Empfehlungsmenü ⑤ und ein Bewertungsdialog ⑥.

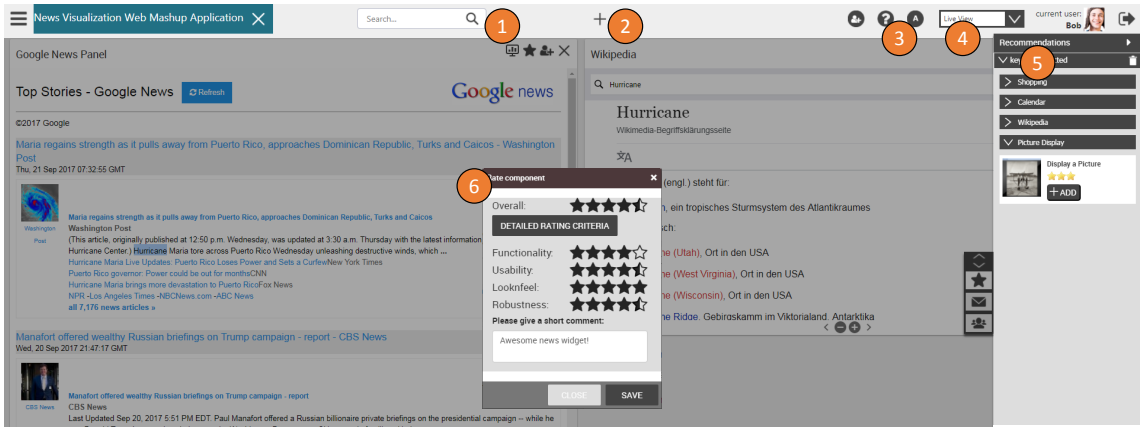


Abbildung 8.5: Nutzungs- und Entwicklungsansicht im Prototyp

Nachfolgend werden ausgewählte Teile der Implementierung sowie die Evaluation vertiefend behandelt.

8.5.1 Evaluation der Capability-View

Kapitel 7.3 beinhaltet das Konzept der *Capability-View*. Nachfolgend wird die Umsetzbarkeit sowie die Eignung des Entwurfs für Nicht-Programmierer analysiert.

Prototyp

Unter Zuhilfenahme studentischer Hilfskräfte wurden die in Abschnitt 7.3 hergeleiteten Konzepte prototypisch realisiert. Die Implementierung wurde zunächst in der TSR vorgenommen und erreichte dort einen Stand, der die Konzeption nahezu bis ins Detail erfüllt. Ein Nachbau des Prototyps unter teilweiser Wiederverwendung von Quellcodes erfolgte im Kontext der CSR, bietet jedoch aus Kapazitätsgründen einen reduzierten Funktionsumfang. Die Programmierung in der TSR stützt sich angesichts ihrer rein clientseitigen Natur auf etablierte Web-Technologien, wie HTML, CSS und JavaScript. Der Funktionsumfang deckt sämtliche Basisfunktionen ab, siehe dazu Abbildung 8.6. Ein semi-transparentes `div`-Element wird über der Live-View erzeugt. Darauf werden gruppierte und zusammenklappbare Capability- sowie Property-Repräsentationen pro Komponente zentriert platziert. Ein- und ausgehende Ports sind gemäß dem Konzept umgesetzt und dienen als Ausgangspunkt für das Erstellen von Verbindungen, wobei sowohl einzelne Klicks auf Ports als auch Drag-and-drop unterstützt werden. Liniensegmente können verschoben werden, um die Platzierung manuell zu beeinflussen. Das spezifizierte Regelwerk zum kontextsensitiven Generieren von Beschriftungen konnte durch den `LabelGenerator` erfolgreich umgesetzt werden. Auch Details und Auswahllisten für Parameter-Zuweisungen bei Vorhandensein mehrerer Parameter und Optionen erlaubt der Prototyp. Zusätzlich wurde das Konzept zur Handhabung von Nicht-UI-Komponenten vollständig eingearbeitet. Um Empfehlungen zu passenden Ports anzuzeigen, gliedert sich die CapView in die Empfehlungsinfrastruktur ein: Der `RecViewer` veröffentlicht ein Event, das Empfehlungsstrategie 3 aus Tabelle 8.6 auslöst, und ist für das Darstellen der erhaltenen Empfehlungen im Sinne eines Recommendation-Viewer verantwortlich. Das zieht das Hervorheben passender Capabilities und Ports nach sich. Dabei kennt der Prototyp neben Coupling- auch Complex-Patterns. Letztere werden vorzugsweise gewählt, falls mehrere Pattern-Instanzen für einen Port anwendbar sind.

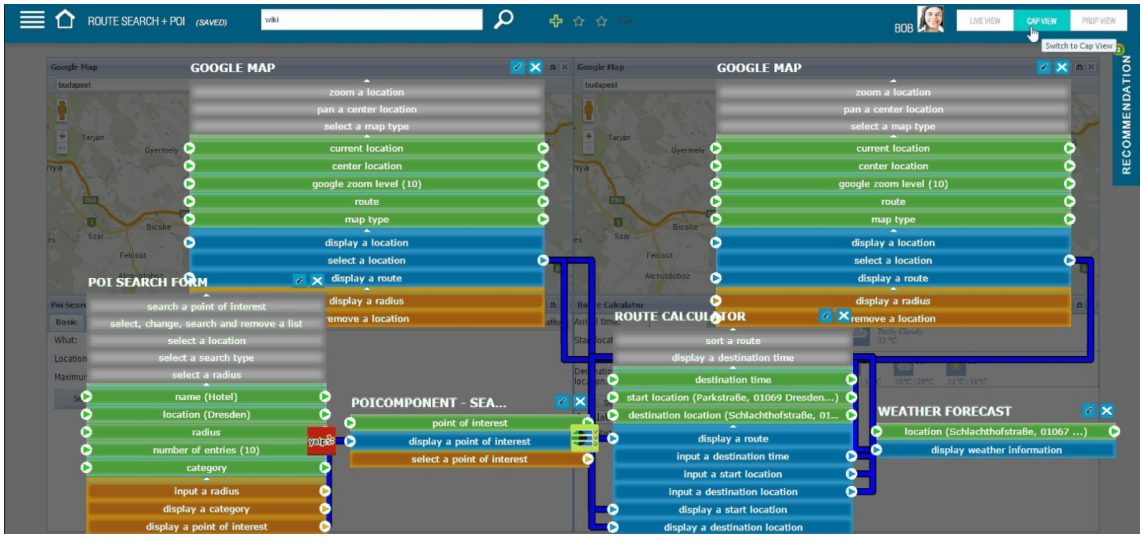


Abbildung 8.6: Screenshot des Prototyps der Capability-View in der TSR

Das Experimentieren mit dem Prototyp deckte Limitierungen des Ansatzes auf, die teilweise konzeptioneller Natur sind. Zunächst stellt die Linienführung eine erhebliche Herausforderung dar. Trotz zahlreicher angewendeten Heuristiken treten bei vielen und großen

Komponenten-Repräsentationen und zahlreichen Linien schnell Engpässe hinsichtlich Platz und Überschneidungsfreiheit von Verbindungen auf. Das ist in obiger Abbildung deutlich erkennbar. Diesbezüglich sorgt das Konzept zum Handhaben von Nicht-UI-Komponenten sowie das Auf- und Zuklappen von Repräsentationen für Entspannung. Zukünftig gilt es, dahingehend weitere Mechanismen zur Reduzierung zu unterstützen, zum Beispiel durch bedarfsorientiertes Anzeigen und Ausblenden in Kombination mit dem Capability-Panel der Anwendung, vergleiche Kapitel 7.5.3. Dadurch kann die Übersichtlichkeit länger gewährleistet werden.

Nutzerstudie

Zur Evaluation der in Kapitel 7.3 vorgeschlagenen Konzepte der *Capability-View* wurde eine Nutzerstudie auf Basis eines vorläufigen Prototyps durchgeführt.

Methodik Die Nutzerstudie erfolgte gemäß der etablierten Methode *Thinking Aloud* [Jor90]. Dabei werden Teilnehmer gebeten, eine Reihe von Aufgaben selbstständig zu lösen. Zudem sind sie aufgefordert, während der Abarbeitung der Aufgaben ihre Gedanken zu äußern, etwa hinsichtlich ihrer Lösungsstrategie, Handlungen sowie erwartetem und tatsächlichem Systemverhalten. Währenddessen protokolliert der Studienleiter die Aussagen der Probanden. Zehn Teilnehmer nahmen an der Studie teil und wurden zunächst gebeten, einen Fragebogen zu demografischen Angaben sowie zu einer Selbsteinschätzung ihrer Fähigkeiten auszufüllen. Die Probanden waren zwischen 22 und 37 Jahre alt und Studenten aus verschiedenen Fachrichtungen, wie Medieninformatik, Informationssystemtechnik, Maschinenbau und Computational Logic. Alle Teilnehmer gaben an, keine oder nur Basiskenntnisse zu Mashups zu besitzen und täglich Webanwendungen zu nutzen. Fünf von ihnen benannten »durchschnittliche« Programmierfähigkeiten.

Nach einer kurzen Einführung zu Mashups sowie zur CapView durch den Interviewer wurden den Probanden zwei Szenarien steigender Kompliziertheit in der Domäne Reiseplanung mit jeweils fünf Aufgaben präsentiert. Die Studie erfolgte auf Basis eines Klickprototyps. Dieser ähnelte dem letztlichen Prototyp optisch stark und deckte grundlegende UI- und Interaktionskonzepte ab. Allerdings wurde die Anbindung an das Empfehlungssystem simuliert und die Handhabung von Nicht-UI-Komponenten entsprach noch nicht dem finalen Konzeptstand. Viel mehr haben die Ergebnisse der Studie zur Anpassung des Konzepts und des in Abschnitt 8.5.1 dargestellten Prototyps geführt. Pro Szenario wurde eine CWA aus UI- und Nicht-UI-Komponenten dargestellt. Das erste Szenario, das vier UI- und eine Nicht-UI-Komponente umfasst, zielte auf die Untersuchung des grundlegenden Verständnisses der Explorations- und Interaktionstechniken ab. Dabei sollten von den Probanden Aufgaben gelöst werden, zum Beispiel Komponenten identifizieren, die zur Capability `Search Flights` beitragen können, und Capabilities so verknüpfen, dass das Suchen und Buchen von Hotels möglich ist. Im Fokus des zweiten Szenarios standen Konzepte zum Erstellen und Modifizieren von Verknüpfungen unter Zuhilfenahme von Parameter-Mappings. Das Mashup erweitert das des ersten Szenarios. Die Teilnehmer wurden gebeten, das Mashup derart zu erweitern, dass Veranstaltungen am Zielort gesucht werden können, dass Routen mit dem ÖPNV vom Hotel zum Veranstaltungsort berechnet werden können und dass eine Wettervorhersage eingeblendet wird. Zur Lösung der Aufgaben galt es, Verbindungen anzupassen und mehrere Parameter zu handhaben.

Zielsetzung der Studie war es, die Eignung der CapView für Nicht-Programmierer zu untersuchen. Daher war von Interesse, ob die Probanden die Aufgaben lösen können. Weiterhin füllten die Teilnehmer einen Fragebogen zur Ermittelung der *System Usability Scale (SUS)* [Bro96] sowie des *Task Load Index (TLX)* [HS88] aus. Die SUS wurde gewählt, da diese ein weitverbreitetes und akzeptiertes Maß für die Usability einer Benutzerschnittstelle darstellt. Der TLX stellt ein viel genutztes Instrument dar, mit dem die wahrgenommene Arbeitslast subjektiv und multidimensional beurteilt werden kann. Weiterhin stand es den Teilnehmer offen, Kommentare niederzuschreiben.

Ergebnisse und Auswertung Als eine wichtige Beobachtung konnten sämtliche Teilnehmer die ihnen gestellten Aufgaben lösen. Geschwindigkeit und Effizienz differierte in Abhängigkeit individueller Kenntnisse. Diesbezüglich wurden jedoch keine belastbaren quantitativen Daten erhoben, da Thinking Aloud hierfür methodisch nicht geeignet ist. Generell wurden Schlüsselkonzepte der CapView positiv von den Probanden wahrgenommen. Die Grundidee einer capability-basierten Abstraktion der Kompositionslogik sowie der örtliche Bezug von Capabilities zu Komponenten im Live-View wurden von den Probanden gutgeheißen. Weiterhin betrachteten die Teilnehmer die natürlichsprachlichen Beschriftungen als ausreichend intuitiv, um Funktionalitäten von Komponenten zu verstehen und Mashups als aufgabenlösende Entitäten zu begreifen (7 mal genannt). Ein Großteil der Probanden schätzte das Hervorheben passender Ports als hilfreich ein (8 mal). Einige merkten an, dass sie ohne diese Funktion nicht erfolgreich gewesen wären. Damit einhergehend unterstützte das kontextsensitive Anpassen der Beschriftungen das Nachvollziehen der Verknüpfbarkeit von Ports. In Kombination erleichterten sämtliche Explorations-Features (Fokussierung, Highlighting, Erzeugen von Beschriftungen und Sätzen, Komponentennamen) die Aufgabenbewältigung erheblich. Aufgrund einer reduzierten Komplexität und erhöhten Klarheit bewertete die Mehrheit der Teilnehmer die Überblick-und-Detail-Metapher der zusammenklappbaren Capabilities als positiv (8 mal). Es konnte beobachtet werden, dass beide Ansätze zum Erstellen von Verknüpfungen, das heißt beginnend bei Eingabe- oder Ausgabeport, von den Teilnehmern verfolgt wurden. Dies unterstreicht die Notwendigkeit beide Möglichkeiten bereitzustellen. Neutral bewerteten Nutzer die verwendete Portmetapher sowie die Farbkodierung.

Die Probanden standen wiederholt folgenden Schwierigkeiten gegenüber. Das Konzept von Nicht-UI-Komponenten ist für Nicht-Programmierer nicht zugänglich und führte zu Fehlinterpretationen von Capabilities. Daher wurde das Konzept wie in Abschnitt 7.3 überarbeitet, sodass Nicht-UI-Komponenten lediglich als Icons an Verbindungen auftauchen. Dies war jedoch im genutzten Prototyp noch nicht umgesetzt, sodass jede Nicht-UI-Komponente eine vollständige Repräsentation wie ihre UI-behafteten Konterfeis aufwies. Zusätzlich wurde offensichtlich, dass die Erwartungen an die Funktionalitäten von Komponenten stark von Erfahrungen der Nutzer im Umgang mit Webanwendungen beeinflusst wurden. Dementsprechend müssen aussagekräftige Capabilities bereitgestellt werden. Es wurde deutlich, dass Probanden Ein- und Ausgabeports unterschiedlich interpretieren: In einer eher »menschenorientierten« Perspektive erwarteten sie zum Beispiel, das `Select an event` nur eine Eingabe hat. Dies widersprach teilweise der systemorientierten Perspektive, die bei der Annotation von Komponenten zum Einsatz kam und bei der `Select an event` eine Ausgabe liefert. Nach kurzer Eingewöhnung waren die Nutzer mit dieser Perspektive vertraut. Wenigen Probanden fiel es schwer zu verstehen, dass CapView von Instanzdaten abstrahiert. Um ihnen das Verständnis zu erleichtern,

sollten Live-View und Capability-View stärker verwoben sein. Wie bereits in Abschnitt 7.5 erläutert, wurde im Rahmen einer weiteren kleinen Nutzerstudie festgestellt, dass die Capability-View als Werkzeug zum Nachvollziehen der Anwendungsfunktionalität nur begrenzt geeignet ist. Das scheint im Wesentlichen der Abstraktion von den Komponenten-UIs und von Instanzdaten geschuldet zu sein. Es gelingt Nutzern zwar, zu verstehen, welche Funktionalitäten bereitstehen und wie Komponenten zusammenhängen. Nachzuvollziehen wie Capabilities durch Nutzerinteraktionen mit Komponenten tatsächlich ausgeführt werden können, fiel den Teilnehmern allerdings schwer.

Die Auswertung der SUS-Fragebögen ergibt einen Mittelwert von 78,5 (70–92,5). Damit wird dem Prototyp eine gute Usability bescheinigt, was angesichts des vorläufigen Charakters des Prototyps ein zufriedenstellendes Ergebnis darstellt. Genauer würden die Probanden das System gerne öfter nutzen (8 mal genannt), sahen keine unnötige Komplexität (8 mal), fanden es einfach zu nutzen (8 mal) und attestierten eine schnelle Erlernbarkeit (6 mal). Das tendenziell positive Feedback kann durch die erhobene Arbeitslast bestätigt werden. Zum Beispiel wurde *geistige Anforderung* und *Anstrengung* zwischen niedrig und mittel bewertet und *Frustration* als sehr gering. Bezüglich ihrer Gesamtleistung waren die Probanden sehr zufrieden. Angesichts der Natur der Studie sind die Dimensionen *physikalische* und *zeitliche Anforderung* wenig aussagekräftig.

Als Beschränkungen der Validität der Ergebnisse zählt zunächst die geringe Anzahl an Teilnehmern, die zudem nicht alle der Zielgruppe entsprechen. Gemäß [Nie00] genügt jedoch bereits eine kleine Nutzergruppe, um den Großteil der Usability-Probleme zu identifizieren. Auch kann die gewährte Einführung zu Verzerrungseffekten geführt haben und das eigentliche Testen im Live-UI war nicht Teil der Studie. Daher scheint es angebracht, einen überarbeiteten Prototyp in einem größeren Rahmen zu evaluieren, wobei insbesondere eine idealerweise repräsentative Bandbreite an Nutzern einbezogen wird, da die Vielfalt der Branchen sowie die Altersstruktur der Studienteilnehmer begrenzt scheint.

Fazit

Mit der Capability-View als graphischem Kompositionswerkzeug wird das Konzept der Forderung 6 nach Unterstützung bei der Komposition gerecht. Die vorgesehene hybride Kompositionsmetapher setzt vorrangig auf *Orchestrierung* durch den Nutzer, indem dieser Capabilities verbindet. Prinzipien der Choreographie treten insofern auf, dass Complex-Patterns komplett integriert werden. Durch diese Metapher wird die Komposition von Komponentenschnittstellen auf die weniger technische Komposition von Capabilities abstrahiert, was Anforderung 3 zugute kommt. Weiterhin trägt das Generieren von Beschriftungen zu Anforderungen 2 und 3 bei, da Begriffe der Fachdomänen verwendet werden. Experimente mit dem Prototyp und die Ergebnisse der Nutzerstudie zeigen die Eignung des Ansatzes, obgleich genannte Limitierungen weiterführende Betrachtungen notwendig machen.

8.5.2 Prototyp und Nutzerstudie des Wizards

Dieser Abschnitt behandelt die Implementierung und Evaluation des in Kapitel 7.4 konzipierten Wizards zur Recherche nach Kompositionsfragmenten.

Prototyp

Die prototypische Umsetzung erfolgte in einem studentischen Komplexpraktikum[1], das durch den Autor dieser Dissertation betreut und mit notwendigen Zuarbeiten, wie der Bereitstellung eines umfangreichen Repertoires an Kompositionsfragmenten und der Back-End-Funktionalitäten, versorgt wurde. Dazu wurde die CSR insbesondere clientseitig erweitert, indem Technologien wie HTML5, CSS3 und JavaScript eingesetzt wurden. Die Implementierung bezieht den parallel weiterentwickelten Prototyp zur unscharfen Eingabe von Qualitätsanforderungen [Rüm+14] mit ein. Das Frontend greift das gängige MVC-Muster auf und nutzt die bereitgestellte SOAP-Schnittstelle des Empfehlungsdienstes, siehe Abschnitt 8.4, um sowohl notwendige Daten für den Anfragebereich (verfügbare Themen, Ziele, Vervollständigungen für Texteingaben) als auch die empfohlenen Kompositionsfragmente abzurufen. Der Prototyp konzentriert sich auf den Anwendungsfall der initialen Anforderungserfassung, sodass er nicht aus dem Kontext einer laufenden Anwendung zu deren Erweiterung dienen kann. Dementsprechend ist der Wizard vom Startbildschirm aus über eine Schaltfläche erreichbar. Wie Abbildung 8.7 verdeutlicht, erscheint der Wizard als modaler Dialog, um mehr Bildschirmplatz nutzen zu können.

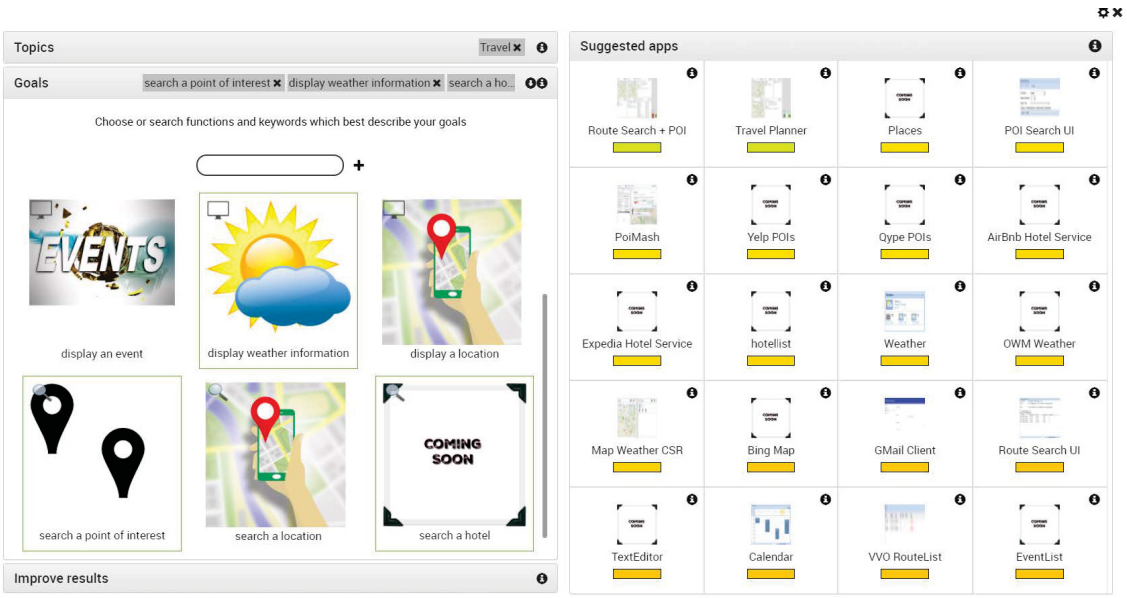


Abbildung 8.7: Screenshot des Prototyps des Wizards

Nutzerstudie

Die Evaluation erfolgt im Rahmen einer Nutzerstudie, die von den Praktikumsteilnehmern mit externen Probanden durchgeführt wurde. Die Methodik, die Rahmenbedingungen für die Auswahl der Teilnehmer sowie die Aufgaben wurden in Absprache mit dem Autor dieser Dissertation festgelegt.

Methodik Die Studie verfolgte das Ziel, die Gebrauchstauglichkeit des konzipierten Ansatzes für Nicht-Programmierer zu belegen. Insgesamt nahmen sechs Probanden teil.

[1] https://mmt.inf.tu-dresden.de/Lehre/Sommersemester_17/KP1/

Details zu diesen sind in Tabelle 8.7 notiert. Fünf der Teilnehmer gelten als Nicht-Programmierer. Die Evaluation folgte dem Prinzip der Methode *Thinking Aloud*. An eine kurze Einführung inklusive einer Vorstellung des Prototyps schloss sich eine etwa zweiminütige freie Explorationsphase an, in der sich Nutzer mit dem Wizard vertraut machen konnten. Anschließend galt es, individuell zwei Aufgaben mit steigender Schwierigkeit zu lösen. In der ersten sollte ein kostenloser Musikplayer gefunden werden. In der zweiten Aufgabe galt es, eine Anwendung zur Anzeige von Sehenswürdigkeiten sowie zur Routenplanung auf einer Karte zu identifizieren. Die Aufgaben decken somit sämtliche Schritte des Wizards ab. Letztlich füllte jeder Proband einen SUS-Fragebogen aus. Darüber hinaus wurden die Nutzer gezielt bezüglich einzelnen Aspekten des Konzepts befragt, beispielsweise zur Gliederung des Wizards und zur Nutzerführung.

Ergebnisse und Auswertung Alle Probanden konnten die Aufgaben ohne Probleme oder Fragen lösen. Tabelle 8.7 veranschaulicht das Resultat der SUS-Fragebögen. Ein SUS-Wert von durchschnittlich 82 attestiert dem Prototyp gute bis sehr gute Gebrauchstauglichkeit. Zwar beträgt der geringste Wert 72,5. Dieser entspricht jedoch einer brauchbaren Usability und wurde zudem vom einzigen Teilnehmer, der nicht der Zielgruppe zuzuordnen ist, vergeben. Daher ist das Ergebnis als gut zu bewerten, da eine zielgruppengerechte Usability bestätigt werden konnte. Die Analyse der Protokolle liefert weitere Erkenntnisse. Die Nutzerführung wurde als unnötig (3 mal genannt) oder nur für die erste Nutzung als nötig (1 mal) erachtet, da der Wizard selbsterklärend sei. Die Gliederung des Wizards wurde als nachvollziehbar beschrieben (2 mal). Schritt drei zur Auswahl von Qualitätszielen, siehe Kapitel 7.4, wurde als unnötig kompliziert für Nicht-Programmierer angesehen (3 mal) und für einen Nutzer wäre nur die Option »kostenlos« wichtig. Die flüssige Bedienung auf Basis von Animationen und die Nützlichkeit der Anfragerepräsentation, siehe Schriftfelder zum Beispiel neben Überschrift »Goals« in Abbildung 8.7, wurde hervorgehoben. Zu gewünschten Funktionen zählen eine Vergleichsfunktion und direkte Anzeige der Nutzerbewertungen von Ergebnissen. Zu den Einschränkungen der Validität der Ergebnisse zählt die geringe Nutzeranzahl, die gemäß [Nie00] allerdings ausreichend ist, um einen Großteil von Usability-Problemen zu identifizieren. Jedoch scheint es angebracht einen überarbeiteten Prototyp in einem größeren Rahmen zu evaluieren, wobei eine repräsentative Bandbreite an Nutzern einbezogen werden sollte. Dennoch gelingt es anhand der Ergebnisse eine tendenziell gute Gebrauchstauglichkeit des Konzepts für die in die Untersuchung einbezogene Nutzergruppe aufzuzeigen. Zudem spielt der Wizard eine wichtige Rolle bei der Anforderungserfüllung. Insbesondere leistet er einen substantiellen Beitrag zu Anforderung 6, indem Nutzer mittels fachlicher Begriffe iterativ ihre Ziele verfeinern können. Ergebnisse werden stets dargestellt, ebenso ihre Übereinstimmung mit den Zielen. Hierdurch werden auch weitere Kriterien gefördert, zum Beispiel WYSIWYG-Prinzipien (Anforderung 4) durch die Vorschau, Anforderung 9.3 durch den Relevanzindikator sowie das Verbergen technischer Details und Sprache des Nutzers (Anforderungen 3 und 2).

8.5.3 Prototyp und Nutzerstudie zu den Erklärungstechniken

Thema dieses Abschnitts sind Details zum Prototyp und zur Nutzerstudie, die zur Validierung und Evaluation wesentlicher Konzepte aus Kapitel 7.5 durchgeführt wurde.

Teilnehmer	Branche / Beruf	Geschlecht	Alter	SUS
P1	Wirtschaftswissenschaften (Studium)	weiblich	22	82,5
P2	Wirtschaftswissenschaften (Studium)	weiblich	27	82,5
P3	IT-Manager	männlich	26	72,5
P4	Psychologie (Studium)	männlich	25	80
P5	Psychologie (Studium)	weiblich	24	92,5
P6	Psychologie (Studium)	männlich	28	82,5
		Mittelwert:	**25**	**82**

Tabelle 8.7: Ergebnisse der durchgeführten Nutzerstudie zum Wizard

Prototyp

Der Prototyp der *Erklärungstechniken* basiert auf studentischen Arbeiten, vor allem [Pfl15; Mic15], und ist Teil des CSR-Client. Dementsprechend kommen standardisierte Technologien wie HTML5 und CSS3 zum Einsatz. Weiterhin werden mehrere JavaScript-Bibliotheken verwendet. Beispielsweise dient *Bootstrap Tour*[2] zum Implementieren von schrittweisen Tutorien. Die Capability- und Hierarchiegraphen der aktuellen CWA werden vom Webservice, der in Abschnitt 8.3.1 vorgestellt wurde, berechnet und clientseitig als JSON-Datenstruktur verwaltet. Die Datenstruktur wird analysiert und für die Konfiguration der Empfehlungstechniken aufbereitet, zum Beispiel um die tatsächlichen Interaktionsschritte für Capability-Ketten zu identifizieren und gemäß der API an *Bootstrap Tour* zu übergeben. Der Inspektionsmodus kann über Menü-Icons aktiviert werden. Daraufhin erscheint das *Capability-Panel* der Anwendung, das im Rahmen der Bachelorarbeit [Mic15] implementiert und durch [Pfl15] erweitert wurde. Es visualisiert den Hierarchiegraphen der Anwendung und stellt integrierte Suchfunktionen bereit. Davon ausgehend kann der Tutorialmodus für ausgewählte Capabilities gestartet werden, vergleiche Abbildung 8.8. Dieser erzeugt ein transparentes `div`-Element über Komponenten-UIs als Zeichenfläche. Weiterhin konnte die vorgeschlagene zweistufige Darstellung von Empfehlungen im *Empfehlungsmenü* erprobt werden.

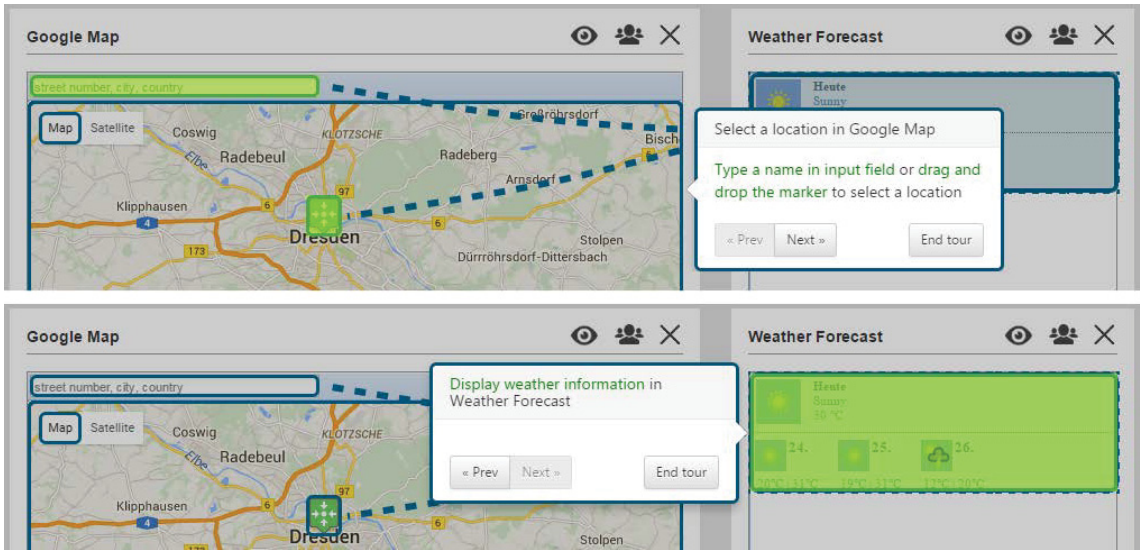


Abbildung 8.8: Prototypischer Tutorialmodus im Rahmen der CSR

[2] https://bootstraptour.com/

Nutzerstudie

Im Rahmen der vom Autor dieser Dissertation betreuten Bachelorarbeit von Konrad Michalik [Mic15] wurden die Konzepte zur *CompCapView* evaluiert. Sechs Personen nahmen teil, die im Durchschnitt 23 Jahre alt waren und keine Kenntnisse über Mashups und Programmierung hatten. Sie arbeiteten oder studierten in verschiedenen Gebieten wie Medizintechnik, Einzelhandel, Wirtschaft und Sozialpädagogik. Obgleich die Konzepte der CompCapView sich als begrenzt geeignet für Nicht-Programmierer herausstellten, ging als wichtiges Ergebnis hervor, dass das Anwendungs-Capability-Panel von den Probanden als hilfreich eingestuft wurde [RBM16]. In der Bachelorarbeit von Johannes Pflugmacher [Pfl15] wurde unter Betreuung des Autors dieser Dissertation eine Nutzerstudie zu den Erklärungstechniken Inspektionsmodus und Tutorialmodus vorgenommen.

Methodik An der Studie nahmen zehn Nicht-Programmierer teil, die aus verschiedensten Fachrichtungen stammen, wie aus den Tabellen 8.8 und 8.9 hervorgeht. Im Mittel waren die Probanden 31 Jahre alt. Im Sinne eines A/B-Tests wurden die Teilnehmer auf zwei Gruppen verteilt: Gruppe A nutzte die Mashup-Plattform ohne, Gruppe B mit den implementierten Erklärungstechniken. Methodische Grundlage stellte *Thinking Aloud* dar. Die Probanden wurden gebeten, mehrere Aufgaben mit ansteigender Schwierigkeit im Kontext von drei CWA zu bearbeiten. Die erste Anwendung dient der Anzeige einer Wettervorhersage und bestand dazu aus Karte und Wetter-Widget. Ein weiteres Mashup erlaubte es, basierend auf zwei Karten und einem Routensuche-Widget, Routen zu bestimmen. Die umfangreichste Komposition war aus acht Komponenten aufgebaut, siehe Abbildung 8.3. Sie erbrachte drei Capabilities: Routensuche, Wetteranzeige und Suche nach Sehenswürdigkeiten.

Einfachere Aufgaben verlangten von den Teilnehmern, Komponenten-Capabilities zu identifizieren und auszuführen, zum Beispiel einen Ort auf der Karte auszuwählen, was auf verschiedene Weisen erreicht werden kann. Weiterhin wurde geprüft, ob Nutzer in der Lage sind, Capabilities zu identifizieren und zu verwenden, die aus dem Zusammenspiel von Komponenten entstehen. Zum Beispiel wurden die Probanden gebeten, in der zweiten CWA Routen zu finden und mittels Mashup drei Sehenswürdigkeiten zu suchen. Eine weitere Aufgabe bestand darin, die übergeordneten Capabilities der dritten Anwendung sowie die dabei partizipierenden Komponenten zu benennen. Zudem zielten Aufgaben darauf ab, zu testen, ob Nutzer sich über aktiven Datenaustausch zwischen Komponenten bewusst sind, beispielsweise zwischen POI-Liste und Karte in Anwendung drei. Neben der Erzeugung von Thinking-Aloud-Protokollen wurde die Anzahl der gelösten Aufgaben gemessen. Zusätzlich sollten die Teilnehmer einen SUS-Fragebogen ausfüllen, um ein weitverbreitetes und akzeptiertes Maß für die Usability zu erheben. Letztlich wurden die Nutzer ermuntert, Aspekte, die ihnen gefallen oder missfallen haben, zu kommentieren.

Das Hauptziel der Nutzerstudie war es, nachzuweisen, dass die Erklärungstechniken Nutzern helfen, die Funktionsweise und die Bedienung von für sie unbekannten CWA besser zu verstehen. Dies sollte sich darin äußern, dass die Anzahl gelöster Aufgaben in Gruppe B ansteigt. Um dies zu prüfen, wurden die Ergebnisse beider Gruppen verglichen.

Ergebnisse und Auswertung Die Resultate des SUS-Fragebogens sowie die Anteile gelöster Aufgaben A_g sind in Tabelle 8.8 für Gruppe A und in Tabelle 8.9 für Gruppe B zusammengefasst. Wie erkennbar ist, differiert A_g der beiden Gruppen deutlich. Ohne die

Teilnehmer	Branche / Beruf	Geschlecht	Alter	A_g	SUS
P1	Techniker (Schüler)	männlich	25	55%	60
P2	Lagerlogistik	männlich	56	36%	48
P3	Arzthelferin	weiblich	45	55%	58
P4	Fotografie	weiblich	23	55%	65
P5	Maschinenbau (Student)	männlich	22	59%	63
		Mittelwert:	34	**50%**	**57**

Tabelle 8.8: Ergebnisse für Gruppe A (ohne Erklärungstechniken)

Teilnehmer	Branche / Beruf	Geschlecht	Alter	A_g	SUS
P6	Feinmittelmechaniker	männlich	23	73%	73
P7	Verkäuferin	weiblich	52	64%	73
P8	Fotografie	weiblich	25	100%	78
P9	Forstwissenschaft (Studentin)	weiblich	23	73%	75
P10	Wirtschaft (Schüler)	männlich	16	86%	70
		Mittelwert:	28	**79%**	**74**

Tabelle 8.9: Ergebnisse für Gruppe B (Nutzung der Erklärungstechniken)

Hilfe der Erklärungstechniken waren die Teilnehmer von Gruppe A im Durchschnitt in der Lage, 50 % der Aufgaben erfolgreich zu absolvieren. Die individuellen Werte reichen dabei von 36 % bis 59 %. Probanden aus Gruppe B waren dahingehend erfolgreicher, da im Mittel 79 % (64 %–100 %) der Aufgaben gelöst werden konnten. Zudem attestierten die Teilnehmer von Gruppe B mit einem SUS-Wert von durchschnittlich 74 eine höhere Usability als die Probanden von Gruppe A mit einem Mittel von 57.

Diese Ergebnisse deuten darauf hin, dass die vorgeschlagenen Erklärungstechniken für Nicht-Programmierer nützlich sind und diesen helfen können, unbekannte CWA besser zu verstehen und zielführender zu bedienen. Der SUS-Wert von 74 indiziert eine gute Gebrauchstauglichkeit der implementierten Werkzeuge, was angesichts des prototypischen Status als zufriedenstellend gewertet werden kann. Ein direkter Vergleich der SUS-Werte scheint kaum sinnvoll, da von den Gruppen verschiedene UIs bedient wurden.

Aus den Protokollen und Kommentaren deutete sich Verbesserungspotenzial an, vorrangig hinsichtlich Prototyp, weniger in Bezug auf das Konzept. Beispielsweise versuchten mehrere Nutzer aus Gruppe B im Tutorialmodus die Schritte direkt auszuführen, was angesichts der transparenten `div`-Elemente scheiterte. Zudem wurden kleinere Usability-Schwächen wie unpassende Icons und ungünstige Positionierung von Rahmen offensichtlich. Wenig überraschend schenkten die Teilnehmer den Capability-Panels von Komponenten, die sie bereits kannten, wenig Beachtung. Weiterhin deckten wenige Nutzer die limitierte Unterstützung des Prototyps für parallele Sequenzen auf. Solche Mängel müssen in nachfolgenden Iterationen der Implementierung behoben werden.

Insgesamt sind die Ergebnisse vielversprechend und deuten an, dass die vorgeschlagenen Konzepte das Verständnis von Nutzern über die Mashup-Funktionalität und das Nachvollziehen von Datenaustausch verbessern können. Assistierte Teilnehmer vermochten es besser, Capabilities zu identifizieren, funktionale Zusammenhänge zwischen Komponenten zu erkennen und notwendige Interaktionen durchzuführen. Als Beschränkung der Validität der Ergebnisse zählt die geringe Teilnehmerzahl, die jedoch genügt, um den Großteil an Usability-Problemen zu identifizieren [Nie00]. Zusätzlich wurden mögliche Seiteneffekten durch andere Plattformfunktionen nicht untersucht. Solche können insbesondere die Nutzer von Gruppe A betroffen haben. Die Erklärungstechniken weisen Grenzen auf, wie

die Lösung für versteckte Elemente. Zusätzlich müssen Komponentenbeschreibungen umfangreich annotiert werden. Das gleicht einer fordernden Aufgabe, vor allem aufgrund verschiedener möglicher Annotationsstile (Anzahl und Granularität von Capabilities und View-Bindings). Auch das Wechselspiel mit von Komponenten bereitgestellten Hilfen wurde nicht betrachtet. Dennoch bieten die Konzepte zahlreiche Vorteile. Sie funktionieren für Blackbox-Komponenten und beliebige ad hoc erstellte CWA. Weiterhin ist es nun erstmals möglich, die Fähigkeiten von Komponenten und Anwendungen direkt im UI systematisch zu explorieren, anstatt Versuch-Irrtum-Strategien anzuwenden. Tutorien können generiert werden, was den Aufwand für Komponentenentwickler reduziert und für ein konsistentes Erscheinungsbild sorgt. Für Anwendungsentwickler bieten die Tutorien kontextualisierte Bedienungsanleitungen direkt anhand der Komponenten-UIs. Wie die Nutzerstudie aufzeigt, helfen die Erklärungstechniken Nicht-Programmierern dabei, sich über Capabilities von Anwendungen klar zu werden und diese besser zu verstehen. Damit gelingt es Anforderungen 9.1 und 9.2 in Gänze zu erfüllen.

8.6 Fazit

Gegenstand dieses Kapitels bildete die praktische Erprobung und die Validierung der in den Kapiteln 4 – 7 vorgestellten wissenschaftlichen Konzepte zum assistierten EUD kompositer Webanwendungen durch Nicht-Programmierer. Die einzelnen vorherigen Abschnitte zeigten die Umsetzbarkeit und Praktikabilität der Konzepte anhand von Prototypen und diskutierten die Ergebnisse von durchgeführten Analysen, zum Beispiel Performanzmessungen, Beispielanwendungen und Nutzerstudien.

Hinsichtlich der **nicht-funktionalen Anforderungen** geben die Nutzerstudien darüber Aufschluss, dass die entwickelten Werkzeuge eine gute Gebrauchstauglichkeit besitzen. Die geringe Teilnehmerzahl genügt, den Großteil potenziell vorhandener Usability-Probleme aufzudecken [Nie00]. Um verallgemeinernde Schlüsse hinsichtlich der Eignung für die gesamte Zielgruppe zu ziehen, sollten allerdings repräsentative Studien mit heterogener Nutzerbasis durchgeführt werden. Zur Erweiterbarkeit des Ansatzes tragen vielfältige Konzepte bei, wie das Hinzufügen von Empfehlungsstrategien, von Triggern sowie von Mapping-Modulen für die Datenmediationen. Auch der Komponentenansatz und die Referenzierung von semantischen Konzepten in SMCDL gehören dazu. Eine derart umfassende Konfigurierbarkeit und Erweiterbarkeit weist kein bisher existierender Ansatz auf. Der geschichtete und modulare Aufbau des Architekturkonzepts sowie der darauf fußenden prototypischen Implementierung, zum Beispiel durch Bereitstellung des Mediators und des Komponentenrepositoriums als Webservices, gewährleistet die geforderte Trennung von Belangen.

Wie gezeigt wurde, können die gestellten **funktionalen Anforderungen** von den Prototypen erfüllt werden. Die nur in der Gesamtsicht bewertbaren Anforderungen werden im Folgenden betrachtet. Wie gefordert, entspricht die in Kapitel 4 vorgestellte Live-Sophistication einem Vorgehensmodell, dessen Aktivitäten hochiterativ durchlaufen werden. Sämtliche Aktivitäten werden durch Werkzeuge unterstützt, insbesondere das Untersuchen von CWA – ein Alleinstellungsmerkmal das hinsichtlich seiner Nützlichkeit positiv evaluiert wurde. Aufgrund der weitgehend nahtlosen Integration der Werkzeuge mit der Live-View erhalten Nutzer das geforderte sofortige Feedback zu Kompositionsschritten. Dadurch wird der Ansatz den Eigenheiten der Zielgruppe Nicht-Programmierer sowie dem Ad-hoc-Charakter von CWA gerecht. Anhand der Implementation von Komponenten

und Anwendungen in verschiedenen Anwendungsdomänen sowie anhand der Domänenunabhängigkeit der konzipierten Modelle und Mechanismen wird die Forderung nach Generizität erfüllt. Dadurch ist es möglich, eine generische Mashup-Plattform bereitzustellen, mit der sich CWA unterschiedlichster Anwendungsdomänen erstellen lassen. Allerdings können Plattformen auch so konfiguriert werden, dass lediglich domänenspezifische CWA entwickelt werden können. Insgesamt erfüllt das Konzept somit Anforderung 1. Bezüglich der Kompositionsmetapher bietet das Konzept eine hybride Lösung. Vorrangig wird ein orchestrierender Ansatz verfolgt. Besonders deutlich wird dies durch die graphische Verknüpfung in der CapView. Um den dadurch entstehenden Aufwand für Nutzer zu verringern, werden Ansätze der Choreographie verwendet, insbesondere beim Integrieren empfohlener Kompositionsfragmente. Die Evaluation liefert Indizien dafür, jedoch sollte zusätzlich eine umfassende Studie durchgeführt werden, um zu belegen, dass dieser Ansatz für ein breites Spektrum an Nutzern geeignet ist. Die Live-View fungiert als zentraler Ort für die Nutzung und die Entwicklung von Mashups, bietet direkt integrierte Werkzeuge an und trägt somit entscheidend zur Erfüllung von Anforderung 4 bei. Zudem wird in Werkzeugen wie dem Wizard und dem Empfehlungsmenü konzeptionell eine Vorschau von Komponenten und Anwendungen gezeigt, was allerdings prototypisch nicht umgesetzt wurde. Die an die Aufgabenmodellierung angelehnten Capabilities finden Verwendung zur funktionalen Annotation von Kompositionsfragmenten. Durch ihre konsequente Nutzung in Werkzeugen zur Generierung von Beschriftungen, die in Nutzerstudien als verständlich bewertet wurden, erfüllt der Ansatz Anforderung 2. Damit einhergehend bieten die Werkzeuge auf Basis von Mediationstechniken und Empfehlungen einen sehr hohen Grad an Abstraktion von technischen Kompositionsdetails (Anforderung 3). Allerdings kann das Aufgreifen der Kompositionsmetapher »Verdrahten« in der CapView als Einschränkung angesehen werden, da diese Metapher außerhalb des Ingenieurbereichs möglicherweise weniger gängig ist. Dies entkräftend sei angeführt, dass hier im Gegensatz zu gängigen Mashup-Ansätzen Capabilities und nicht Komponentenschnittstellen verknüpft werden, wodurch die Komposition auf Ebene fachlicher Begriffe und Aktivitäten stattfindet. Um Anforderung 7 gerecht zu werden, sieht das Konzept die Nutzung von Kompositionswissen vor. Dabei ergänzen sich strukturelles, semantisches und statistisches Wissen aus Patterns, Ontologien (darauf aufsetzend auch aus Capabilities und Mediationstechniken) und Feedback. Für diese Aspekte existieren Metamodelle und passende Mechanismen zur Detektion und Verwaltung von Instanzen. Ein neuartiges Metamodell für die Funktionalität von Kompositionsfragmenten stellen Capabilities dar, die Hierarchien bilden und einen Bezug zum zugehörigen UI herstellen können. Ein neuer Algorithmus kann passende Capabilities für beliebige Kompositionsfragmente berechnen. Vor- und Nachteile hierzu wurden bereits in Abschnitt 8.3.2 ausführlich diskutiert.

Zwar existieren Beschränkungen im Funktionsumfang oder kleine Mängel der Gebrauchstauglichkeit einzelner Prototypen, die es künftig zu beheben gilt. Trotz der genannten Einschränkungen wird aber insgesamt die Tragfähigkeit der gewählten Ansätze deutlich. Um das entwickelte Vorgehensmodell sowie den Gesamtansatz vergleichend beurteilen zu können, wäre eine repräsentative Nutzerstudie mit ähnlichen Mashup-Plattformen zwar wünschenswert. Dies scheitert jedoch an der mangelnden Verfügbarkeit entsprechender Prototypen.

Das nachfolgende Kapitel widmet sich einem Fazit zu den Ergebnissen und den wissenschaftlichen Beiträgen der vorliegenden Dissertation. Außerdem werden potenzielle Schwerpunkte für anknüpfende Forschungsaktivitäten beleuchtet.

9 Zusammenfassung, Diskussion und Ausblick

Im Rahmen der vorliegenden Dissertation wurde ein neuer, umfassender EUD-Ansatz vorgestellt, der Nicht-Programmierer bei der Entwicklung von kompositen Webanwendungen unterstützt. Zugrunde liegen eine gründlichen Literaturrecherche und Erkenntnisse aus Nutzerstudien. Etablierte Konzepte wurden einbezogen und um neuartige Ansätze ergänzt. Zuletzt verschaffte Kapitel 8 einen Einblick in die praktische Umsetzung der Konzepte im Rahmen der CRUISE-Mashup-Plattform. Zudem wurden die Konzepte anhand der Resultate von Nutzerstudien, Benchmarks sowie einer Expertenevaluation soweit mit den Prototypen möglich beurteilt und hinsichtlich der Anforderungen abgeglichen.

Im weiteren Verlauf werden die Ergebnisse der Dissertation kapitelweise zusammengefasst. Daraufhin bietet Abschnitt 9.2 eine Einschätzung der Ergebnisse, der Grenzen und der Beiträge der gesamten Dissertation hinsichtlich der Forschungsziele und Thesen. Abschließend gibt Abschnitt 9.3 einen Ausblick auf weiterführende Forschungsthemen.

9.1 Zusammenfassung und Beiträge der Kapitel

Kapitel 1 – Einleitung

Das erste Kapitel motivierte die Thematik der Dissertation, indem die maßgeblichen Trends des EUD sowie der zunehmenden Verbreitung von kompositen Webanwendungen illustriert wurden. Insbesondere wenn Nicht-Programmierer in die Lage versetzt werden, eigenständig Mashups aus vorhandenen Komponenten zu entwickeln, sind die Herausforderungen für geeignete Kompositionsplattformen hoch. Das wurde anhand einer genauen Zielgruppendefinition und im Zuge einer systematischen Problemanalyse unter Einbeziehung von Forschungsergebnissen verdeutlicht. Wie zur Problemlösung beigetragen werden soll, formulieren die Forschungsthesen dieser Arbeit sowie die daraus abgeleiteten Forschungsziele. Diese lassen sich in folgende Schwerpunkte untergliedern: (1) Entwurf plattformunabhängiger Metamodelle, unter anderem zur Repräsentation von Kompositionswissen, (2) Konzeption generischer Basismechanismen, wie eines Empfehlungssystems, (3) Definition eines generisches Vorgehensmodells inklusive endnutzergerechter Werkzeuge

und (4) Schaffung einer Architektur, welche die vorherigen Teilaspekte in eine Kompositionsplattform für das EUD von CWA durch Nicht-Programmierer zusammenführt. Zusätzlich nimmt das Kapitel eine Abgrenzung von Fragestellungen vor, die nicht im Fokus der Dissertation stehen, und erläutert den Aufbau der Arbeit.

Kapitel 2 – Grundlagen und Anforderungsanalyse

Dieses Kapitel diente der Klärung von Grundlagen, der fachlichen Einordnung der Arbeit und der Anforderungsanalyse. Das konzeptionelle Rahmenwerk der Arbeit wurde stellvertretend anhand der CRUISE-Mashup-Plattform vorgestellt. Basierend darauf und auf den thematisierten Defiziten der Plattform führen praxisnahe Referenzszenarien zur systematischen Ableitung eines umfassenden Anforderungskatalogs. Dieser stellte zugleich den Maßstab bei der Validierung sowohl existierender als auch der eigenen Konzepte dar.

Forschungsbeiträge: Dieses Kapitel liefert mit der Herleitung eines umfassenden Anforderungskatalogs ein Rahmenwerk zur Evaluation von Kompositionsplattformen für das EUD durch Nicht-Programmierer.

Kapitel 3 – Stand von Forschung und Technik

In Kapitel 3 wurde ausgehend von dem Anforderungskatalog als Bewertungsmaßstab eine Bestandsaufnahme des aktuellen Stands von Forschung und Technik aus den Blickwinkeln der Forschungsfelder EUD, Semantisches Web und komposite Webanwendungen vorgenommen. Den Schwerpunkt bildet dabei die Analyse verwandter Kompositionsplattformen für Mashups und Webservices. Des Weiteren finden Ansätze zu Empfehlungssystemen, zur Eingabe von Nutzeranforderungen und zu Datenmediation besondere Beachtung. Wie die Diskussion geeigneter Lösungsansätze und Defizite herausstellt, zeigen sich vielversprechende Konzepte für einzelne Fragestellungen. Allerdings fehlt bisher ein ganzheitlicher Ansatz, der notwendige Automatismen und Werkzeuge vorsieht, um Nicht-Programmierern im gesamten Prozess von Nutzung und Entwicklung einer CWA angemessen zu assistieren.

Forschungsbeiträge: Als wesentlichen Beitrag liefert dieses Kapitel eine systematische Untersuchung des Standes von Forschung und Technik im interdisziplinären Forschungsfeld der vorliegenden Dissertation.

Kapitel 4 – Assistiertes EUD von CWA durch Nicht-Programmierer

Die aus der Analyse des Ist-Standes in den vorherigen Kapiteln gewonnenen Erkenntnisse bildeten die Basis für die eigene Konzeption. Entsprechend liefert Kapitel 4 einen Überblick der in dieser Dissertation entwickelten, wissenschaftlichen Konzepte des Gesamtansatzes zum assistierten EUD von CWA durch Nicht-Programmierer. Im Zentrum des gewählten Lösungsansatzes steht das Vorgehensmodell der Live-Sophistication, das eine hochiterative Abfolge der Aktivitäten *Suchen* nach Komponenten, *Entwickeln*, *Empfehlungen erhalten* sowie *Benutzen* und *Untersuchen* vorsieht. Die entwickelte Architektur sieht zur bedarfsgerechten Unterstützung von Nicht-Programmierern bei diesen Aktivitäten eine Palette neuer Werkzeuge vor, die von einer Mashup-Laufzeitumgebung angeboten werden, um eine hochgradige Verschmelzung der Aktivitäten zu erreichen. Als Fundament

von Werkzeugen und Laufzeitumgebung fungieren Basismechanismen zur semantischen Datenmediation, ein hybrides Empfehlungssystem sowie ein Algorithmus zur Abschätzung der Funktionalität beliebiger CWA, die allesamt auf neuen wissenschaftlichen Ansätzen beruhen. Diese Mechanismen stützen sich wiederum auf eine Schicht aus Modellen, die unter anderem Kompositionsfragmente und deren Funktionalitäten, Domänenwissen, den Kontext, Kompositionswissen und Nutzerfeedback formal beschreiben.

Forschungsbeiträge: Das vorgestellte Vorgehensmodell sowie das Zusammenwirken der Teilkonzepte im neuen Gesamtkonzept tragen wesentlich dazu bei, Nicht-Programmierer in die Lage zu versetzen, komposite Webanwendungen zielgerichtet und selbstständig zu entwickeln und zu nutzen. Somit stellt das Architekturkonzept einen zentralen Forschungsbeitrag dieser Dissertation dar und konnte in [RM18] im Rahmen eines erweiterten Konferenzartikels veröffentlicht werden. Wie nachfolgend aufgeführt, bieten auch die einzelnen Teilkonzepte Beiträge zum Forschungsgebiet.

Kapitel 5 – Basiskonzepte

In Kapitel 5 wurden weite Teile der Modellgrundlage und Funktionsblöcke im Detail spezifiziert. Mit dem vorgestellten Capability-Metamodell zur Beschreibung von Funktionalitäten, einem Wissensgraph über Capabilities, einer erweiterten semantischen Komponenten- und Kompositionsbeschreibung, Kontextmodellen und einem Feedbackmodell steht eine ausdrucksstarke Datengrundlage zur Verfügung. Zur Erleichterung des Kompositionsprozesses wurden Techniken zur semantischen Datenmediation konzipiert, die auf Basis von Ontologiewissen automatisch bestimmt und ausgeführt werden können. Neu ist auch der Algorithmus zur Abschätzung von Capabilities beliebiger Kompositionsfragmente, dessen Ergebnisgüte durch eine Expertenevaluation belegt wurde. Zudem geht das Kapitel auf die Struktur und die Befüllung eines gerichteten Multigraphen ein, der semantische und statistische Relationen von Capabilities beschreibt.

Forschungsbeiträge: Wesentliche Beiträge dieses Kapitels stellen die Konzepte zu Capabilities, zu Mediationstechniken und zur Abschätzung von Capabilities dar. Sie sind in dieser Form neu, umfassend validiert worden und konnten auf internationalen Tagungen veröffentlicht werden [Rad+14; RBM16; RBM17; RM17a]. Capabilities sowie Mediationstechniken wurden in anderen Forschungsvorhaben aufgegriffen [Bli+15; Bli+16].

Kapitel 6 – Empfehlungssystem

Mit dem neuen Empfehlungssystem widmete sich Kapitel 6 einem zentralen Bestandteil des Gesamtkonzepts. Der Ansatz zeichnet sich durch eine durchgängige Unterstützung von Nicht-Programmierern und eine hochgradige Anpassbarkeit an den Nutzer- und Nutzungskontext anhand von Empfehlungsstrategien aus. Das Hauptaugenmerk lag auf der Vorstellung von Triggern als Auslöser des Empfehlungsprozesses, auf Patterns, die strukturelles Kompositionswissen repräsentieren, sowie auf Empfehlungsmethoden, die in konfigurierbarer Form Matching und Ranking von Kompositionsfragmenten vornehmen und sich dabei auf die zur Verfügung stehende Datenbasis an Kompositionswissen stützen. Letzteres ergänzt strukturelles Wissen aus Pattern-Instanzen mit semantischem und statistischem Wissen, das aus Ontologien, Mediationstechniken, dem Capability-Wissensgraphen sowie Nutzerfeedback stammt.

Forschungsbeiträge: Der Kernbeitrag dieses Kapitels besteht in einem Konzept eines Empfehlungssystems für EUD-Kompositionsplattformen, das es erstmals erlaubt, auf Grundlage von Empfehlungsstrategien in hohem Maße an den Nutzer- und Nutzungskontext angepasst zu werden. Es konnte gezeigt werden, dass die wesentlichen Bestandteile des Empfehlungskreislaufs in modularer Form separiert und bedarfsgerecht zusammengesetzt werden können. Das Konzept und die Evaluationsergebnisse wurden auf einschlägigen internationalen Konferenzen [Rad+12; RM17a] präsentiert. Zentrale Ansätze greift ein Forschungsprojekt im Umfeld von Enterprise-Search [MRM17] auf.

Kapitel 7 – Methoden zur Nutzerführung

Schwerpunkt von Kapitel 7 bildete die Präsentation der konzipierten Benutzeroberfläche einer Kompositionsplattform für Nicht-Programmierer. Diese fügt die entwickelten EUD-Werkzeuge in einen harmonischen Rahmen zur Realisierung von Live-Sophistication ein. Hierzu wurden sukzessive der Startbildschirm, die Live-View, die Capability-View, der Wizard sowie die Erklärungstechniken spezifiziert. Durch die Verknüpfung der Werkzeuge untereinander und durch deren Aufsetzen auf den Basismechanismen und den Metamodellen wird das Gesamtkonzept abgerundet. Insgesamt wird Nicht-Programmierern somit erstmals bei der Nutzung und Entwicklung von CWA durchgängig assistiert.

Forschungsbeiträge: Der Beitrag des Kapitels liegt in der Spezifikation einer in sich zusammenhängenden endnutzergerechten Werkzeugpalette. Dabei wurden sowohl neue Konzepte geschaffen als auch existierende Ansätze erfolgreich auf das EUD von kompositen Webanwendungen übertragen. Die Ergebnisse wurden auf einschlägigen internationalen Tagungen veröffentlicht [RBM13; RM17b; RM18].

Kapitel 8 – Implementierung und Evaluation

Kapitel 8 verschafft einen Einblick in die Evaluation der vorgeschlagenen wissenschaftlichen Konzepte. Dazu wurde die technische Umsetzung in Form von prototypischen Implementierungen in der CRUISE-Plattform erörtert, welche die praktische Umsetzbarkeit der Konzeption untermauern. Zuerst stand die Realisierung der Metamodelle sowie der Basisarchitektur im Mittelpunkt. Anhand einer Vielzahl von Komponenten und Anwendungen aus vielfältigen Domänen zeigt sich die Angemessenheit der erweiterten Komponentenbeschreibung sowie des Capability-Metamodells. Die erfolgreiche, modulare Erweiterung der CRUISE-Plattform um die neuen Konzepte belegt die Umsetzbarkeit letzterer. Im weiteren Verlauf wurde im Detail auf die Implementierung der Mediationskonzepte eingegangen. Dabei steht mit dem Mediationsdienst ein Webservice bereit, der von TSR und CSR genutzt wird. Die Performanzuntersuchung ergab angemessene Antwortzeiten, was die Praktikabilität der vorgestellten Konzepte unterstreicht. Auch die nachgewiesenermaßen performante Implementierung des Algorithmus zur Capability-Abschätzung wurde erörtert. Die Ergebnisse einer Expertenevaluation lassen die Plausibilität des Konzepts erkennen. Schließlich konnte die Umsetzbarkeit, die Performanz und ansatzweise die Konfigurierbarkeit des vorgeschlagenen Empfehlungssystems demonstriert werden. Im Zusammenspiel der konzipierten Modelle und Werkzeuge gelang es, zahlreiche Empfehlungsstrategien zu erproben. Letztlich thematisierte das Kapitel die Implementierung wesentlicher Werkzeuge für das EUD. Auf Basis von Nutzerstudien, deren Methodik und Ergebnisse jeweils diskutiert wurden, fand die Evaluation der Werkzeuge statt.

Forschungsbeiträge: Durch die prototypische Implementierung und die praktische Erprobung sowie durch Performanzuntersuchungen gelang es, die Umsetzbarkeit, Validität und Praktikabilität der vorgeschlagenen Konzepte nachzuweisen. Zudem zeigen die Ergebnisse mehrerer Nutzerstudien die tendenzielle Nützlichkeit und die Gebrauchstauglichkeit der Ansätze für Nicht-Programmierer. Evaluationsergebnisse von Teilkonzepten wurden in den zugehörigen Publikationen veröffentlicht.

9.2 Einschätzung der Ergebnisse

Den Schwerpunkt dieses Abschnitts bildet die zusammenfassende Einschätzung der Ergebnisse. Dazu werden diese zunächst in Relation zu den Zielen und Thesen beurteilt. Daraufhin folgt die Zusammenfassung der wissenschaftlichen Beiträge der Arbeit.

9.2.1 Diskussion der Erreichung der Forschungsziele

Die in Kapitel 1.2 aufgestellten Thesen und Forschungsziele dienten durchweg als Leitfaden für die Strukturierung der Dissertation. Auf diese bezugnehmend werden die Ergebnisse der Arbeit in diesem Abschnitt reflektiert. Um fundierte Aussagen zu den Forschungsthesen treffen zu können, sind umfangreiche Untersuchungen notwendig, wozu die vorliegende Arbeit einen Grundstein legt. Daher orientiert sich die Einschätzung vorrangig an den formulierten Forschungszielen. Teilweise kann dahingehend auf die bereits in Kapitel 8 dargestellte Erfüllung der Anforderungen verwiesen werden.

Schaffung von Modellen

Das Konzept beinhaltet eine Reihe von Metamodellen. Capabilities stellen einen an Aufgabenmodelle angelehnten Ansatz zur Beschreibung von Funktionalitäten dar und können hierarchisch strukturiert werden. Sie verfügen über formale Semantik durch Referenzierung von Ontologiekonzepten. Zudem werden sie den Besonderheiten von CWA dadurch gerecht, dass sie eine Verknüpfung zum UI von Komponenten etablieren können. Damit stellen Capabilities eine geeignete Grundlage für die Kommunikation mit Nicht-Programmierern dar, wie anhand der Verwendung zur Erzeugung von Beschriftungen in diversen Werkzeugen und anhand zugehöriger Nutzerevaluationen belegt werden konnte. Zusätzlich dienen Capabilities in Kombination mit einem Modell für funktionale Anforderungen nach [Rüm+14] als wichtiger Baustein für das EUD. Darüber hinaus wurden Erweiterungen für Komponenten- sowie Kompositionsbeschreibungen entwickelt und in die Metamodelle der CRUISE-Plattform integriert. So verfügen diese erstmals über systematische Annotationen von Capabilities und Anforderungen an den Nutzungskontext. Es gelang weiterhin, Kompositionswissen durch mehrere komplementäre Ansätze auszudrücken. Unter Einbeziehung verwandter Arbeiten wurden Patterns spezifiziert, die strukturellem Wissen entsprechen, das vorrangig aus Kompositionsmodellen extrahiert wurde. Als Alleinstellungsmerkmal verfügen Patterns über Annotationen mit Capabilities sowie Angaben zur Herkunft und sind mit Feedback assoziiert. Dadurch liegen Bewertungen und Informationen dazu vor, in welchem Kontext sie verwendet wurden. Weiterhin existiert ein Wissensgraph über Capabilities, der semantische und statistische Relationen beherbergt, sowie ein Kontextmodell, das unter anderem Nutzerprofile enthält. Keine der

bisherigen Kompositionsplattformen bietet ein derart umfassendes Repertoire an Informationsquellen. Mit ihm wurde der Grundstein für intelligente Automatismen und Werkzeuge gelegt. Bei der Konzeption der Modelle wurde konsequent auf Plattformunabhängigkeit und Standardkonformität geachtet. Durch die Erprobung der Ansätze im implementierten Prototyp konnte eine große Bandbreite an Einsatzszenarien und Anwendungsdomänen abgedeckt werden, sodass von ausreichender Generizität auszugehen ist. Damit wurde dieses Ziel insgesamt erreicht.

Mechanismen und Algorithmen

Als wesentlicher Beitrag zum Erreichen des Gesamtziels einer EUD-Plattform für Nicht-Programmierer beschäftigte sich die Dissertation mit der Konzeption und Validierung mehrerer neuer Basismechanismen. Der Algorithmus zum Abschätzen von Capabilities erlaubt es erstmals, Anwendungen automatisch bezüglich ihrer Domäne und Funktionalität zu klassifizieren sowie mit einem plausiblen Capability-Modell auszustatten. Das Konzept konnte erfolgreich umgesetzt sowie validiert werden und dient im Gesamtansatz als Fundament für weitere Algorithmen und nicht zuletzt für Werkzeuge, wie die Erklärungstechniken. Basierend auf semantischen Komponentenannotationen ermöglichen es Mediationstechniken, semantische Konnektoren zwischen Komponentenschnittstellen zu definieren. Wie in Kapitel 8.2 anhand von Beispielszenarien und Performanzmessungen nachgewiesen, weiten sich die Möglichkeiten zur Kopplung von Komponenten erheblich aus, und die schnelle Anwendung von Mediationstechniken gewährleistet die Gebrauchstauglichkeit von CWA. Weiterhin erfolgt das Ableiten und Ausführen von Abbildungsvorschriften automatisiert, sodass Nutzer von diesen Aspekten weitgehend abgeschirmt werden. Der Ansatz verfügt über ein hybrides Empfehlungssystem, das unter Zuhilfenahme des Kompositionswissens Vorschläge zu Kompositionsfragmenten ermittelt. Als Alleinstellungsmerkmal wurde auf eine Architektur geachtet, deren Module anhand von Empfehlungsstrategien konfigurierbar sind. Somit kann ein Empfehlungssystem erstmals an die jeweilige Situation und den Nutzungskontext angepasst werden. Zudem sieht der Ansatz die konsequente Verwendung von Triggern vor, um Nutzer proaktiv und reaktiv zu unterstützen. Trigger bieten ferner, durch Zugriff auf sämtliche Modelle und auf Instanzdaten ausgetauschter Kommunikationsnachrichten, die Möglichkeit, domänenspezifische Situationen zu detektieren. Durch das Sammeln und Einbeziehen von kontextualisiertem Nutzerfeedback sowie durch das generische Rahmenwerk für Matching und Ranking sind gute Grundlagen für die Berechnung von hochqualitativen Empfehlungen gelegt. Zwar stand die Evaluation der Ergebnisqualität von Empfehlungsmethoden nicht im Fokus der Betrachtung. Jedoch ist anzunehmen, dass durch die konzipierte Kombination aus semantischen und community-basierten Techniken ausreichend valide Pattern-Instanzen vorhanden sind. Dass die Konzepte praktikabel und angemessen sind, zeigen Benchmarks sowie die Entwicklung verschiedener Empfehlungsstrategien. Da die Algorithmen auf den zuvor genannten Metamodellen aufsetzen und generisch funktionieren, sind sie als unabhängig von konkreten Anwendungen und Domänen einzustufen.

Vorgehensmodell und Werkzeuge

Als zentrale Konzepte dieser Arbeit sind der vorgeschlagene Entwicklungsprozess und die zugehörigen Werkzeuge zu nennen. Gemäß Live-Sophistication umfasst der hochiterative Prozess mehrere Aktivitäten, die in sich wiederum iterative Strukturen aufweisen können,

um der Natur von CWA und den Eigenheiten der Zielgruppe gerecht zu werden. Durch die konsequente Durchsetzung von WYSIWYG-Prinzipien und die hohe Werkzeugintegration können besonders kurze Feedback-Schleifen erreicht werden. Werkzeuge schirmen Nutzer vor technischen Details ab und nutzen fachliches Vokabular durch die automatisierte Generierung von Beschriftungen aus Capabilities. Wie angestrebt, wurde erreicht, dass Nicht-Programmierer bei sämtlichen Aktivitäten unterstützt werden. Folgende Punkte kennzeichnen den Gesamtansatz.

- Ein hoher Automatisierungsgrad konnte auf Basis von Mediationstechniken und der Integration von Pattern-Instanzen erzielt werden.
- Erstmals bietet ein Ansatz durchgängig Assistenz durch das Empfehlungssystem an und das sowohl proaktiv als auch reaktiv.
- Capabilities dienen zur Generierung von Beschriftungen und Sätzen in verschiedenen Werkzeugen. Es wurde gezeigt, dass die Resultate für Nicht-Programmierer verständlich sind.
- Erstmals werden generische Techniken zur Unterstützung von Nicht-Programmierern beim Untersuchen und Verstehen von CWA vorgestellt. Dabei leisten Capability-Graphen einen wesentlichen Beitrag zum Erzeugen von Bedienungsanleitungen und Erklärungen.
- Mit dem Wizard steht Nicht-Programmierern ein valides Werkzeug zur Recherche nach Kompositionsfragmenten zur Verfügung. Die Kombination mit weiteren Unterstützungsmechanismen wie dem Startbildschirm, der Stichwortsuche und Empfehlungen senkt die Hürde beim Einstieg in den Kompositionsprozess.
- Die Verknüpfung von Capabilities als Kompositionsmetapher unterstützt Nicht-Programmierer, muss allerdings noch weiterführend evaluiert werden.

Insgesamt konnte somit der erste ganzheitliche Ansatz zum assistierten EUD von CWA durch Nicht-Programmierer auf Basis valider Konzepte geschaffen werden. Dieser löst zudem viele der Probleme aus Abschnitt 1.1. Durch mehrere Nutzerstudien wurde die Validität von vorgeschlagenen Werkzeugen deutlich.

Referenzarchitektur

Basierend auf der CRUISE-Plattform konnte erfolgreich eine Referenzarchitektur für das assistierte EUD kompositer Webanwendungen konzipiert und vorgestellt werden. Existierende Bestandteile wie das Komponentenrepositorium und die Laufzeitumgebung wurden aufgegriffen und konzeptgemäß erweitert. Zudem wurden neue Architekturkomponenten etabliert. Diese betreffen insbesondere das Empfehlungssystem, die Mediationsinfrastruktur und die Repositorien für CWA und Nutzerfeedback. Zudem bieten Laufzeitumgebungen eine Reihe neuer EUD-Werkzeuge. Die Referenzarchitektur konnte erfolgreich prototypisch validiert werden. Mit den zwei Laufzeitumgebungen TSR und CSR ließ sich die Übertragbarkeit der Konzepte demonstrieren. Zentrale Funktionsblöcke, wie das Komponentenrepositorium, der Empfehlungs-, Feedback- und Mediationsdienst, wurden serviceorientiert realisiert und dienten zum Nachweis der Wiederverwendbarkeit anhand der Nutzung in beiden MRE. Auch dieses Ziel konnte somit erreicht werden.

9.2.2 Diskussion der Forschungsthesen

Insgesamt kann zu den Forschungsthesen folgendermaßen Stellung genommen werden.

These 1: Wie gezeigt wurde, stellt das Capability-Metamodell eine valide Beschreibung von Komponentenfähigkeiten dar. Zudem wurde ein Algorithmus konzipiert, der die Capabilities eines beliebigen Kompositionsfragments bestimmt. Die Expertenevaluation lieferte Indizien für die Plausibilität der Ergebnisse. Angesichts dessen kann trotz der genannten Einschränkungen und des Aufwands für Komponentenentwickler die These als belegt gewertet werden.

These 2: Live-Sophistication entspricht einem hochiterativen Prozess, der in jeder Phase Werkzeuge vorsieht, um Nicht-Programmierer durchgängig zu unterstützen. Wie gezeigt wurde, können Capabilities als Basis für verständliche Beschriftungen dienen. Zudem erlaubt der Wizard es Nutzern, ihre Anforderungen mit fachlichem Vokabular auszudrücken. Die Capability-View nutzt die abstrahierte Metapher der Verknüpfung von Capabilities, welche in Kombination mit Mediationstechniken und automatisierter Pattern-Integration hochgradig technische Details aus dem Kompositionsprozess nimmt. Allerdings fehlt an dieser Stelle eine umfassendere Nutzerstudie, um die Eignung belastbar zu belegen. Erklärungstechniken unterstützen schließlich die Nutzung von Mashups, indem direkt im UI von Blackbox-Komponenten Bedienungsanleitungen eingeblendet werden. Die durchgeführte Validierung legt den Schluss nah, dass die angestrebte bessere Verständlichkeit der Funktionalität von CWA für Nicht-Programmierer erreicht wurde. Auch wenn die These nicht zweifelsfrei belegt ist, zeigt sich ansatzweise, dass Capabilities ein geeignetes zentrales Instrument für das EUD darstellen.

These 3: Auf konzeptioneller Ebene wurde die These belegt. Das Empfehlungssystem zeichnet sich durch lose gekoppelte Module für die Belange Identifikation von Empfehlungsgründen, Berechnung, Präsentation und Integration von Empfehlungen aus. Diese können via Empfehlungsstrategien in kontextspezifischen Konstellationen kombiniert werden. Damit sind auch domänenspezifische Szenarien umsetzbar. Einschränkend sei angemerkt, dass diese Flexibilität nicht komplett prototypisch erprobt werden konnte und eine Evaluation hinsichtlich der Eignung umgesetzter Strategien für Nutzer noch aussteht. Die Durchgängigkeit konnte anhand von umgesetzten Empfehlungsstrategien aufgezeigt werden.

9.2.3 Wissenschaftliche Beiträge

Kapitel 9.1 enthält bereits eine detaillierte Aufstellung der Forschungsbeiträge der vorliegenden Dissertation. Wie an selbiger Stelle erwähnt, wurden die wissenschaftlichen Erkenntnisse in Tagungsbänden und Büchern veröffentlicht. Zusammenfassend lassen sich folgende **Kernbeiträge** nennen.

- Konzeption eines hochiterativen Vorgehensmodells und zugehöriger Werkzeuge für das EUD von kompositen Webanwendungen für Nicht-Programmierer
- Entwurf der modularen Softwarearchitektur einer Plattformen für das vorgeschlagene Vorgehensmodell und Integration dieser in die CRUISE-Plattform

- Konzeption des Capability-Metamodells zur Beschreibung der Funktionalität von Kompositionsfragmenten
- Formalisierung von Modellen zur komplementären Beschreibung von strukturellem, statistischem und semantischem Kompositionswissen
- Konzeption und Validierung automatisierter, semantikbasierter Datenmediation zur Auflösung von Inkompatibilitäten zwischen Komponentenschnittstellen
- Konzeption und Validierung eines Algorithmus zum Ableiten von Capabilities beliebiger Kompositionsfragmente
- Konzeption und Implementierung eines Empfehlungssystems, das durch Empfehlungsstrategien an heterogene Einsatzkontexte angepasst werden kann
- Konzeption und Evaluation von Werkzeugen zur Komposition, insbesondere der Capability-View, sowie zur geführten Recherche nach Kompositionsfragmenten
- Konzeption und Evaluation von Erklärungstechniken, die auf Basis von Komponentenannotationen generiert werden können

In der Gesamtheit zeigt sich der erste ganzheitliche Ansatz zum EUD kompositer Webanwendungen durch Nicht-Programmierer. Mittels Kombination bewährter Lösungen mit neuartigen Konzepten zu semantischen Modellen, Algorithmen und intelligenten Werkzeugen gelingt es, Nicht-Programmierer bei dem Erstellen, Modifizieren und Benutzen von CWA proaktiv und reaktiv zu unterstützen. Nachfolgend stehen Limitierungen des vorgeschlagenen Ansatzes und ein Ausblick auf zukünftige Arbeiten im Mittelpunkt.

9.2.4 Grenzen der geschaffenen Konzepte

Grenzen und offene Punkte der jeweiligen Teilkonzepte wurden bereits umfangreich im Rahmen von Kapitel 8 diskutiert. Daher zielt dieser Abschnitt auf die zusammenfassende Darstellung übergeordneter Einschränkungen der geschaffenen Konzepte.

Grenzen bezüglich des Komponentenmodells Wie erwähnt, stützen sich viele Konzepte des Gesamtansatzes auf semantische Annotationen von Komponenten (und daraus abgeleitet von Kompositionsfragmenten) in Form von Capabilities. Besonders hervorzuheben sind die Datenmediation, die Abschätzung von Capabilities, das Matching in Empfehlungsmethoden und Werkzeuge wie die Capability-View und die Erklärungstechniken. Es erfordert einen nicht zu vernachlässigenden Aufwand für Komponentenentwickler, um die benötigten Annotationen sowohl in ausreichender Quantität als auch Qualität vorzunehmen. Dies wiederum hängt maßgeblich von der Qualität der semantischen Wissensrepräsentation ab. Hier gilt es, etablierte Ontologien einzubeziehen und Autorenwerkzeuge bereitzustellen. Insgesamt wurde jedoch gezeigt, dass auf Basis der semantischen Annotationen mächtige Mechanismen und Werkzeuge für das EUD durch Nicht-Programmierer, wie die oben genannten, geschaffen werden können. Zudem liefert das Konzept einen Ansatz, der es erlaubt, aus Capabilities von Komponenten automatisiert auf Capabilities von Anwendungen zu schließen, sodass die Bereitstellung der Annotationen von Anwendungen und Pattern-Instanzen erheblich erleichtert wird.

Sicherstellung funktionaler Korrektheit Das Konzept sieht zur Sicherstellung plausibler Kompositionsergebnisse mehrere komplementäre Ansätze vor: auf Basis von Ontologiewissen, Mediationstechniken und kollektivem Wissen in Form von Pattern-Instanzen und Feedback. Dies bedingt, dass initial, wenn kollektives Wissen noch spärlich vorhanden ist, vorrangig auf Semantik zurückgegriffen wird. Wie in der Problemanalyse bereits thematisiert, sind derartige Ansätze eher als Versprechen aufzufassen. Allerdings muss unterstellt werden, dass Komponentenentwickler eine gewisse Sorgfalt bei der Annotation und Implementierung ihrer Komponenten walten lassen. Unter dieser Annahme erlaubt der Ansatz ein schrittweises Komponieren unter Sicherstellung von semantisch geprüften Signaturen. Das heißt, dass Nutzer lediglich von der Plattform angebotene Kompositionsschritte vornehmen können. An Grenzen stößt der Einsatz von Annotationen sobald Komponenten zwar hinsichtlich Capabilities und Parametertypen passen, jedoch aufgrund kontextbezogener oder instanzdaten-bezogener Probleme nicht harmonieren. Hier können die Anforderungen von Komponenten zumindest hinsichtlich antizipierbarer Kontexteinschränkungen Abhilfe leisten. Durch Ausprobieren, Verwenden der Erklärungstechniken sowie der abgeleiteten Anwendungsfunktionalität können Nutzer nachvollziehen, ob die Funktionsweise der erwarteten entspricht. Zur Gewährleistung der »funktionalen Korrektheit« (im Sinne einer erwartungskonformen Funktionsweise) einer Gesamtanwendung wird zudem auf kollektives Filtern gesetzt.

Gebrauchstauglichkeit für Nicht-Programmierer Als Kompositionsmetapher sieht die Capability-View das »Verdrahten« in einer überlagerten Ansicht vor. Zwar nehmen Nutzer dabei im UI eine Komposition von Capabilities und nur implizit von Komponentenschnittstellen vor, was eine wesentliche Neuerung darstellt. Dennoch zeigten die Nutzerstudien, dass diese Metapher Probleme bei Nutzern aus nicht-technischen Branchen verursachen kann, von Skalierungsproblemen bei begrenzter Auflösung oder vielen Linien abgesehen. Wie daher bereits motiviert und wie im Zuge der Entwicklung des Inspektionsmodus erfolgreich validiert wurde, wäre eine stärkere Einbindung des Kompositionswerkzeugs in die Live-View erstrebenswert. Grundsätzlich besteht bei kompositen Webanwendungen die Herausforderung, dass verschiedene UI- und Interaktionsmetaphern in Komponenten einer Anwendung auftreten können. Zwar kann unterstellt werden, dass wenig brauchbare Komponentenkonstellationen durch das kollektive Filtern nach einer Aufwärmphase des Empfehlungssystems nicht mehr auftauchen. Allerdings fehlen dedizierte Konzepte, dies nach der Initialisierung der Plattform sicherzustellen. Diesbezüglich bietet sich aufbauend auf Capabilities und deren View-Bindings ein angepasstes Matching als Lösungsansatz an. Zwar wurden bereits wesentliche Teilkonzepte des UI in Nutzerstudien hinsichtlich Gebrauchstauglichkeit und Nützlichkeit evaluiert. Jedoch bedarf es weiterer Iterationszyklen sowie einer Gesamtevaluation der Kompositionsplattform, insbesondere um den gewählten Kompromiss aus Mächtigkeit der Werkzeuge und Einfachheit sowie Angemessenheit für Nicht-Programmierer längerfristig zu evaluieren.

9.3 Laufende und weiterführende Arbeiten

Unterstützung durch Autorenwerkzeuge Zur Sicherstellung einer hohen Ergebnisgüte der konzipierten Algorithmen und der Werkzeugunterstützung spielt die Qualität der semantischen Annotationen von Komponenten im vorgeschlagenen Ansatz eine kritische

Rolle. Komponentenentwicklern wurde dahingehend ein Leitfaden an die Hand gegeben, allerdings sollten auch Autorenwerkzeuge entwickelt werden, die bei der Auswahl von Ontologiekonzepten und der Erstellung der Komponentenbeschreibung behilflich sind. Ebenso relevant gestaltet sich die Konfiguration der Kompositionsplattform, insbesondere des Empfehlungssystems. Wie zuvor thematisiert, bieten sich hier Autorenwerkzeuge, wie mehrstufige Assistenten, zur Vereinfachung an.

Überwachung und Durchsetzung von nicht-funktionalen Anforderungen Im Rahmen des Wizards wurden nicht-funktionale Anforderungen in symbolhaft abstrahierter Darreichungsform in einem extra Schritt vorgesehen und zudem ein Anknüpfungspunkt zum Anforderungseditor [Rüm+14] eingebettet. Die Auswertung solcher Anforderungen beschränkt sich im Konzept allerdings auf das Ranking im Rahmen von Empfehlungsmethoden. Als Erweiterung des Ansatzes böten Maßnahmen zur Anforderungsüberwachung sowie geeignete Strategien bei Nichteinhalten von Anforderungen einen Mehrwert und werden parallel zur vorliegenden Dissertation erforscht [Rüm+13; Rüm+14].

Weiterentwicklung der Assistenzfunktionen Als Ergebnis der Nutzerstudien scheint eine Kombination der Konzepte zur Capability-View mit denen des Inspektionsmodus vielversprechend, um ein stärker in die Live-View integriertes Kompositionwerkzeug bereitzustellen. Weiterentwicklungspotenzial bieten auch proaktive Hinweise zu Problemen in der Komposition. Hier wäre eine Erweiterung des Trigger-Ansatzes vorstellbar, wobei ein Auslösen neben Empfehlungsberechnung weitere Aktionen nach sich ziehen kann, wie das Einblenden von Hinweisen und das Starten eines Tutorials. Zudem können zukünftig erweiterte Ansätze für das Testen von CWA untersucht werden. Dabei wäre vorstellbar, dass Nicht-Programmierer bspw. Nutzdaten, die zwischen Komponenten ausgetauscht werden, inspizieren und modifizieren können oder Operationen aufrufen. Zwar sind derartige Konzepte aus dem Bereich der Softwareentwicklung hinlänglich bekannt, im Rahmen von Mashups und EUD durch Nicht-Programmierer stellen sich allerdings bezüglich UI-, Interaktionsmetaphern und Abstraktionsgrad sowie durch die Anbindung von Komponenten an Webservices umfassende Fragestellungen. Weiterhin bieten die geschaffenen Konzepte ein geeignetes Rahmenwerk für Weiterentwicklungen im Geschäftsumfeld, zum Beispiel für Enterprise-Search- und Business-Analytics-Anwendungen sowie zur Bereitstellung bedarfsgerecht anpassbarer Arbeitsabläufe. Diesbezüglich sind weiterführende Forschungsfragen, etwa hinsichtlich Datensicherheit, kollaborativer Nutzung und Korrektheit der Komposition, zu beantworten. Der Fokus der Arbeit lag auf Single-User-Szenarien, sodass synchrone Multi-User- und Multi-Device-Szenarien ausgeklammert wurden. Diesbezüglich gilt es, zahlreichen Forschungsfragen nachzugehen, etwa hinsichtlich Multi-User-Empfehlungen, kollaborativer Nutzung von Werkzeugen und Konfliktmanagement. Beispielsweise widmet sich das Projekt DoCUMA [@DoC] Verteilungsmechanismen von Komponenten inklusive Autorenwerkzeugen in Multi-Device-Szenarien. Weitere Herausforderungen wie die Freigabe von Sichten auf Komponenten und die Synchronisation von Komponenten in kollaborativen Szenarien werden parallel untersucht [Bli+15; Bli+16].

A Anhänge

A.1 Richtlinien für die Annotation von Komponenten

- Capability-Ketten haben, hinsichtlich der Zuordnung ihrer Activities zu den drei in Kapitel 5.1.1 vorgestellten Superklassen, die Grundstruktur `Input` → `Manipulate` → `Output`. Variationen davon sind möglich, etwa mehrere `Input`-Capabilities für eine `Manipulate`-Capability, mehrere sequentielle `Manipulate`-Capabilities oder mehrere parallele `Output`-Capabilities.
- Zumindest eine vollständige *causes*-Verkettung existiert innerhalb von Komponenten. Eine bidirektionale Verkettung ist nicht verpflichtend.
- Annotation von Operationen:
  - Optional wird eine Operation mit `Input`-Capabilities ausgestattet. Es wird entweder für alle Parameter eine `Input`-Capability mit Entity gleich dem Parametertyp erstellt oder eine einzelne, deren Entity ein semantisch grobgranulareres Konzept ist. Parameter einer Operation und die Entities der zugehörigen Capabilities stehen in semantischer Relation.
  - Eine `Transform`-Capability wird annotiert, wenn keine Nutzerinteraktion gefordert ist und die Capability nicht auch über das UI erbracht werden kann. `Input`-Capabilities werden mit dieser Capability verknüpft. Zudem sollte eine weitere Verkettung dieser Capability mit anderen Capabilities auf Komponentenebene in Erwägung gezogen werden.
  - Ist keine `Transform`-Capability an der Operation sinnvoll, wird eine `Input`-Capability hinterlegt und mit Capabilities auf Komponentenebene verknüpft.
- Zur Annotation von Events wird mindestens eine `Manipulate`- oder `Output`-Capability auf Komponenten- oder Operationsebene via `causedBy` referenziert.
  - Falls das Event unter mehreren Umständen ausgelöst wird, werden alle entsprechenden Capabilities im Capability-Selektor im Attribut *causedBy* aufgeführt und `OR`-verknüpft.
  - Sind mehrere Voraussetzungen für das Auslösen notwendig, ist das Connective `AND` im Capability-Selektor zu nutzen.

A.2 Fragebogen zur System Usability Scale

System Usability Scale

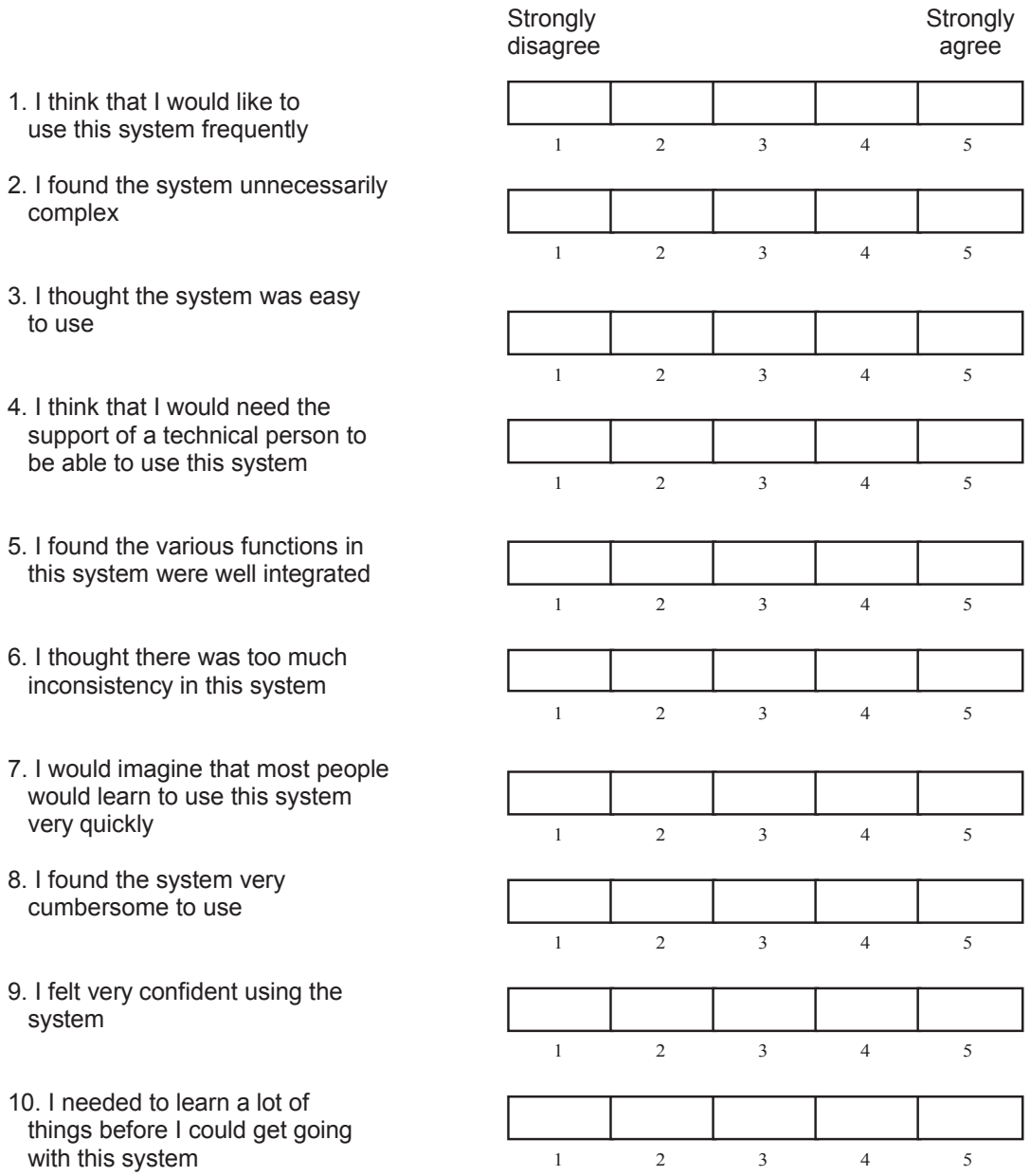


Abbildung A.1: Original Fragebogen zur Erhebung der SUS aus [Bro96]

A.3 Illustration von Mediationstechniken

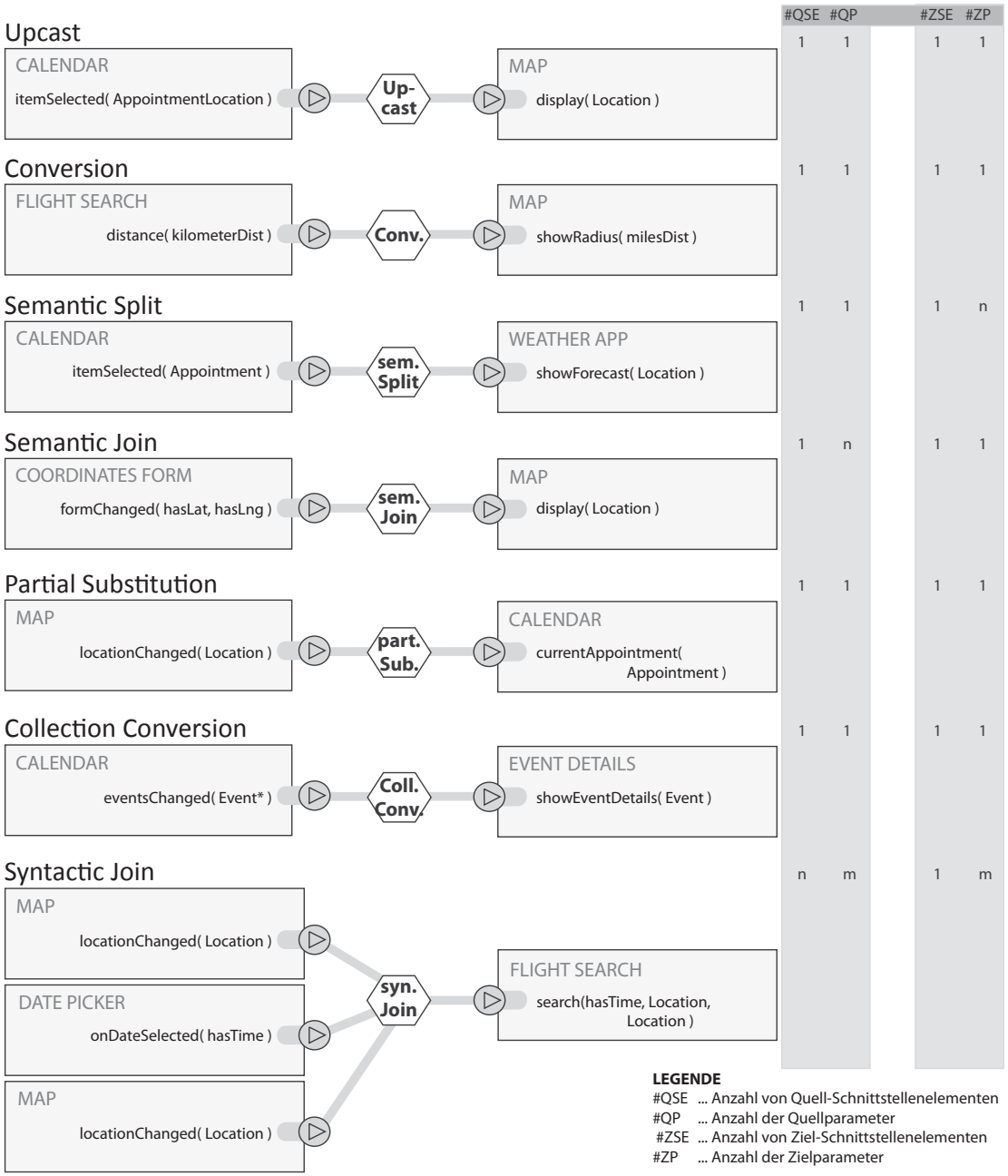


Abbildung A.2: Illustration konzipierter Mediationstechniken

A.4 Komponentenbeschreibung in SMCDL (Beispiel)

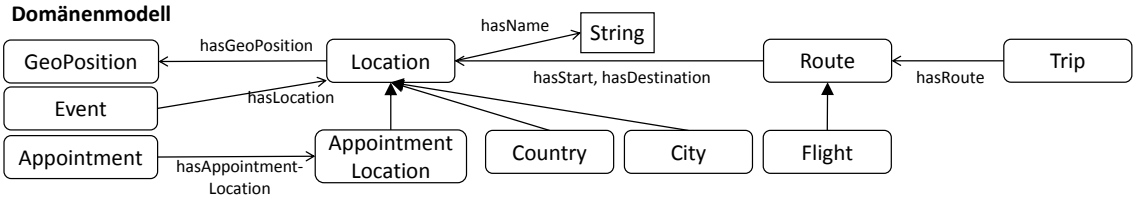
```
<component id="http://mmt.inf.tu-dresden.de/EDYRA/prototype/Map" name="Google
    Maps" version="1.0" isUI="true" ...>
    <capability id="capDispLoc" activity="act:Display" entity="travel:Location">
        <viewbinding>
            <atomicoperation element="div[id$='_mapcanvas']" id="atoOpDispLoc" />
        </viewbinding>
        <causedBy>capInpLoc or cap02 or capInpLocDet</causedBy>
    </capability>
    <capability id="cap01" activity="act:Zoom" entity="travel:Location">
        <viewbinding>
            <atomicoperation element="div[id$='gMapZoomControl']" id="atomicOp01"
                 modifier="interaction:Click" />
        </viewbinding>
        <viewbinding>
            <atomicoperation element="div[id$='_mapcanvas']" id="atomicOp02"
                modifier="interaction:DoubleClick" />
        </viewbinding>
        <viewbinding>
            <atomicoperation element="div[id$='_mapcanvas']" id="atomicOp03"
                modifier="interaction:MouseWheel" />
        </viewbinding>
    </capability>
    <capability id="cap02" activity="act:Select" entity="travel:Location">
        <viewbinding>
            <atomicoperation element="input[id$='mapTextField']" id="atomicOp04"
                modifier="interaction:TypeOperation" />
        </viewbinding>
        <viewbinding>
            <atomicoperation element="div[id$='gMapCurrentLocationIcon']" id="
                atomicOp05" modifier="interaction:DragNDrop" />
        </viewbinding>
        <viewbinding>
            <atomicoperation element="div[id$='currentLocationIcon']" id="
                atomicOp10" modifier="interaction:DragNDrop" />
        </viewbinding>
    </capability>
    <capability entity="travel:Route" activity="act:Display" id="capShowRoutes">
        <viewbinding>
            <atomicoperation element="div[id$='_mapcanvas']" id="atomicOp14"
                modifier="interaction:RightButton" />
        </viewbinding>
    </capability>...
    <metadata xmlns="http://mmt.inf.tu-dresden.de/smcdl/1.15/metadata">
        <keywords>
            <keyword>map</keyword>
            <keyword>google</keyword>
            <keyword>location</keyword>
        </keywords>...
        <documentation>Map utilizing Google Maps API. It features a draggable
            marker and the visualization of routes.</documentation>...
    </metadata>
    <requirements>
        <runtimes>
            <runtime id="TSR" version=">=1.8" />
        </runtimes>
    </requirements>
    <interface>
        <property type="mcdl:hasTitle" required="true" name="title" configOnly="
            true">
            <default>Google Map</default>
        </property>
        <property name="width" type="mcdl:hasWidth" required="true" configOnly="
            true">
            <default>400</default>
        </property>
        <property name="height" type="mcdl:hasHeight" required="true" configOnly=
            "true">
            <default>325</default>
        </property>
```

```
58          <property name="currentLocation" type="travel:hasCurrentLocation"
                required="false">
59              <changeEvent name="locationSelected" />
60          </property>
61          <property name="center" type="travel:hasCenterLocation" required="false">
62              <causedBy>cap02a</causedBy>
63          </property>
64          <property name="zoomLevel" type="travel:GoogleZoomLevel">
65              <default>10</default>
66              <causedBy>cap01</causedBy>
67          </property>
68          <property name="route" type="travel:Route" required="false">
69              <causedBy>capShowRoutes</causedBy>
70          </property>
71          <property name="mapType" type="travel:hasMapType" required="false">
72              <causedBy>cap02b</causedBy>
73          </property>
74          <event name="locationSelected">
75              <parameter name="location" type="travel:Location" />
76              <causedBy>cap02</causedBy>
77          </event>
78          <operation name="showRadius">
79              <capability id="cap03" activity="act:Display" entity="travel:
                    hasRadius" />
80              <parameter name="long" type="travel:hasLongitude" />
81              <parameter name="lat" type="travel:hasLatitude" />
82              <parameter name="radius" type="travel:hasRadius" />
83          </operation>
84          <operation name="showLocation">
85              <capability entity="travel:Location" activity="act:Input" id="
                    capInpLoc">
86                  <causes>capDispLoc</causes>
87              </capability>
88              <parameter name="location" type="travel:Location" />
89          </operation>
90          <operation name="showLocationDetails">
91              <capability id="capInpLocDet" activity="act:Input" entityContext="
                    travel:Location" entity="travel:hasDescription">
92                  <causes>capDispLoc</causes>
93              </capability>
94              <parameter name="location" type="travel:Location" />
95              <parameter name="description" type="travel:hasDescription" />
96          </operation>
97          <operation name="showRoute">
98              <capability id="cap07" activity="act:Input" entity="travel:Route">
99                  <causes>capShowRoutes</causes>
100             </capability>
101             <parameter name="route" type="travel:Route" />
102         </operation>...
103     </interface>
104     <binding bindingtype="mapping_simplewrapper">
105         <dependencies>
106             <dependency language="javascript">
107                 <url>http://maps.google.com/maps/api/js?sensor=false</url>
108             </dependency>
109             <dependency language="javascript">
110                 <url>https://svn.mmt.inf.tu-dresden.de/repositories/CRUISe/
                        components/trunk/UI_GoogleMap/GoogleMap.js</url>
111             </dependency> ...
112         </dependencies>
113         <constructor>
114             <code>new EDYRA.components.MapComponent()</code>
115         </constructor>
116     </binding>
117 </component>
```

Listing A.1: Beispielhafte Beschreibung einer Kartenkomponente in SMCDL (Auszug)

A.5 Beispiele zu Algorithmen

A.5.1 Berechnung einer bestimmenden Entity



Ablauf

E = { Location, AppointmentLocation, hasLocation, Route }
Berechnung „untergeordneter Entities“ (Mengen *L*)

für Location (maßgebliche Ontologieklasse *c* = Location)	$L_{Location}$ = { AppointmentLocation, Country, City, hasName, hasGeoposition }
für AppointmentLocation (*c* = AppointmentLocation)	$L_{AppointmentLocation}$ = { hasName, hasGeoposition }
für hasLocation (*c* = Location)	$L_{hasLocation}$ = { hasAppointmentLocation, AppointmentLocation, Country, City, hasName, hasGeoposition }
für Route (*c* = Route)	L_{Route} = { Flight, hasStart, hastDestination, Location }
Entferne alle Konzepte aus den Mengen *L* aus *E*:	*E*= { hasLocation, Route }

Berechnung von „übergeordneten Entities“ (Mengen *U*)

für Location (*c* = Location)	$U_{Location}$ = { Route, hasStart, hastDestination }
für hasLocation (*c* = Location)	$U_{hasLocation}$ = { Route, hasStart, hastDestination }
für Route (*c* = Route)	U_{Route} = { Trip, hasRoute }

$(\{Location\} \cup U_{Location}) \cap (\{Location\} \cup U_{hasLocation}) \cap (\{Route\} \cup U_{Route}) = \{ Route \}$
Bestimmende Entity = Route

Abbildung A.3: Beispielhafte Berechnung der bestimmenden Entity

A.5.2 Berechnung der Ähnlichkeit atomarer Capabilities

	Activity	Activity-Modifier	Entity	Entity-Context
a	Display	-	Location	Event
b	Display	-	Location	-
c	Select	-	Location	-
d	Sort	-	Hotel	-
e	Sort	hasName	Hotel	-
f	Sort	hasName	Event	-

Tabelle A.1: Datenbasis zur Illustration der semantischen Ähnlichkeit atomarer Capabilities

	Activity	Activity-Modifier	Entity	Entity-Context	sim_{atomic}
a → a	1	1	1	1	**1**
a → b	1	1	1	0	**0,67**
b → c	0	1	1	1	**0,33**
a → c	0	1	1	0	**0,22**
d → e	1	0	1	1	**0,67**
e → f	1	1	0	1	**0,33**

Tabelle A.2: Illustration der semantischen Ähnlichkeit atomarer Capabilities

A.6 Bewertung verwandter Ansätze

In diesem Abschnitt werden Ergebnisse der Untersuchung verwandter Ansätze tabellarisch aufgeführt. Kriterien sind dabei die Anforderungen aus Kapitel 2.3 unter Verwendung folgender Skala: +. . . unterstützt; o. . . teilweise unterstützt; –. . . nicht unterstützt oder Kriterium nicht anwendbar.

Abbildung A.4: Analyseergebnisse betrachteter Ansätze zur Datenmediation

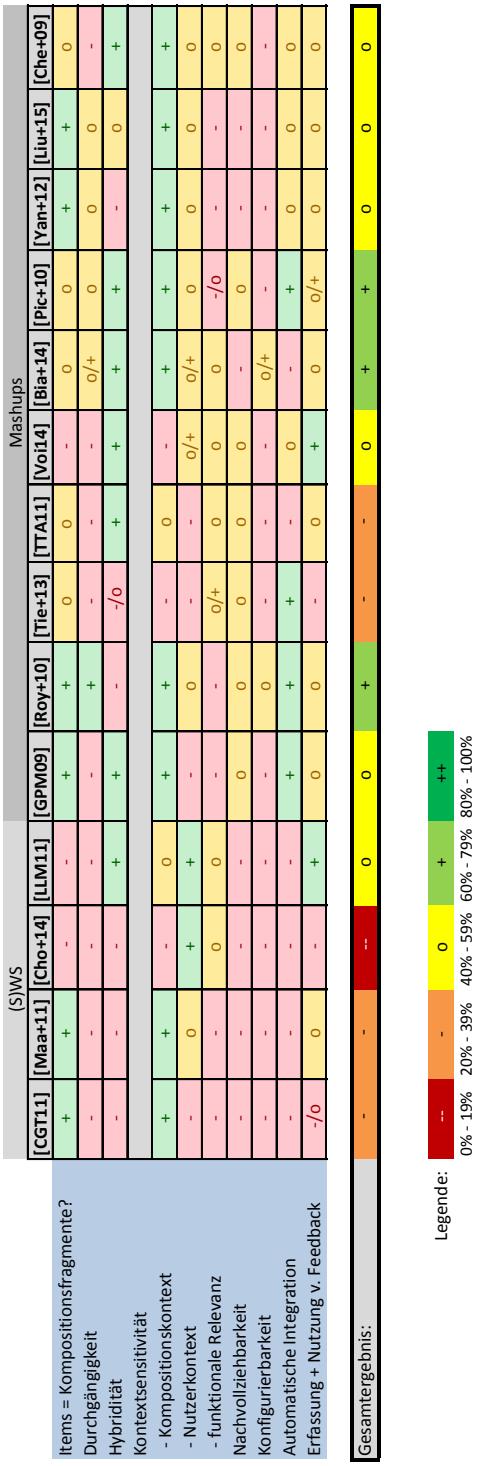


Abbildung A.5: Analyseergebnisse betrachteter Empfehlungssysteme

Literaturverzeichnis

[Aal+03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski und A.P. Barros. »Workflow Patterns«. In: *Distributed and Parallel Databases* 14.1 (2003), Seiten 5–51.

[AP12] Saeed Aghaee und Cesare Pautasso. »EnglishMash: Usability Design for a Natural Mashup Composition Environment«. In: *Current Trends in Web Engineering*. Herausgegeben von Michael Grossniklaus und Manuel Wimmer. Band 7703. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, Seiten 109–120.

[AP13] Saeed Aghaee und Cesare Pautasso. »Guidelines for Efficient and Effective End-User Development of Mashups«. In: *End-User Development*. Herausgegeben von Yvonne Dittrich, Margaret Burnett, Anders Mørch und David Redmiles. Band 7897. LNCS. Springer Berlin Heidelberg, 2013, Seiten 260–265.

[AP14] Saeed Aghaee und Cesare Pautasso. »End-User Development of Mashups with NaturalMash«. In: *Journal of Visual Languages & Computing* 25.4 (2014), Seiten 414–432.

[AT05] G. Adomavicius und A. Tuzhilin. »Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions«. In: *IEEE Transactions on Knowledge and Data Engineering* 17.6 (2005), Seiten 734–749.

[AT14] Panagiotis Adamopoulos und Alexander Tuzhilin. »On Unexpectedness in Recommender Systems: Or How to Better Expect the Unexpected«. In: *ACM Trans. Intell. Syst. Technol.* 5.4 (Dez. 2014), 54:1–54:32.

[Bat90] Marcia J. Bates. »Where should the person stop and the information search interface start?« In: *Information Processing & Management* 26.5 (1990), Seiten 575–591.

[BDM10] Devis Bianchini, Valeria De Antonellis und Michele Melchiori. »A Recommendation System for Semantic Mashup Design«. In: *2010 Workshops on Database and Expert Systems Applications*. IEEE, Aug. 2010, Seiten 159–163.

[BDM12] Devis Bianchini, Valeria De Antonellis und Michele Melchiori. »Web Engineering: 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings«. In: Herausgegeben von Marco Brambilla, Takehiro Tokuda und Robert Tolksdorf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Kapitel Semantic Collaborative Tagging for Web APIs Sharing and Reuse, Seiten 76–90.

[BDM13a] Devis Bianchini, Valeria De Antonellis und Michele Melchiori. »A Framework for Guided Search of Mashup Components«. In: *Proceedings of the 3rd International Workshop on Semantic Search Over the Web*. SS@ '13. New York, NY, USA: ACM, 2013, 3:1–3:4.

[BDM13b] Devis Bianchini, Valeria De Antonellis und Michele Melchiori. »A Multi-perspective Framework for Web API Search in Enterprise Mashup Design«. In: *Proceedings of the 25th International Conference on Advanced Information Systems Engineering*. CAiSE'13. Valencia, Spain: Springer-Verlag, 2013, Seiten 353–368.

[BDM13c] Devis Bianchini, Valeria De Antonellis und Michele Melchiori. »Advanced Web API Search Patterns Adding Collective Knowledge to Public Repository Facets«. In: *Proceedings of International Conference on Information Integration and Web-based Applications & Services*. IIWAS '13. Vienna, Austria: ACM, 2013, 211:211–211:219.

[BDM17] Devis Bianchini, Valeria De Antonellis und Michele Melchiori. »Exploratory Search of Web Data Services Based on Collective Intelligence«. In: *Web Engineering: 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings*. Herausgegeben von Jordi Cabot, Roberto De Virgilio und Riccardo Torlone. Springer International Publishing, 2017, Seiten 378–385.

[Bez09] Jean Bezivin. *Advances in Model Driven Engineering – Achievements and challenges*. Keynote bei der JISBD 2009, http://www.mondragon.edu/jisbd2009/Documentos/JeanBezivin_JISBD09.pdf. letzter Zugriff: 04.01.2016. 2009.

[BHS08] Jake D. Brutlag, Hilary Hutchinson und Maria Stone. »User Preference and Search Engine Latency«. In: *JSM Proceedings, Qualtiy and Productivity Research Section*. 2008.

[Bia+14] Devis Bianchini, Silvana Castano, Valeria De Antonellis u. a. »Transactions on Large-Scale Data- and Knowledge-Centered Systems XIII«. In: Herausgegeben von Abdelkader Hameurlain, Josef Küng und Roland Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. Kapitel RUBIK: Proactive, Entity-Centric and Personalized Situational Web Application Design, Seiten 123–157.

[Bli+15] Gregor Blichmann, Carsten Radeck, Sergej Hahn und Klaus Meißner. »Component-based Workspace Awareness Support for Composite Web Applications«. In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWas 2015)*. 2015.

[Bli+16] Gregor Blichmann, Carsten Radeck, Robert Starke und Klaus Meißner. »Triple-based Sharing of Context-Aware Composite Web Applications for Non-programmers«. In: *Proceedings of the 12th International Conference of Web Information Systems and Technologies (WEBSIST 2016)*. 2016.

[BMC93] N.J. Belkin, P.G. Marchetti und C. Cool. »BRAQUE: Design of an interface to support user interaction in information retrieval«. In: *Information Processing & Management* 29.3 (1993), Seiten 325–344.

[Bot+14] Andreas Both, Viet Nguyen, Mandy Keck u. a. »Motive-based Search using a Recommendation-driven Visual Divide and Conquer Approach«. In: *IADIS International Journal on WWW/Internet* 12.2 (2014), Seiten 115–130.

[Bou+08] Eric Bouillet, Mark Feblowitz, Zhen Liu, Anand Ranganathan und Anton Riabov. »A Tag-based Approach for the Design and Composition of Information Processing Applications«. In: *SIGPLAN Not.* 43.10 (Okt. 2008), Seiten 585–602.

[Bro96] J. Brooke. »Usability evaluation in industry«. In: Herausgegeben von P. W. Jordan, B. Thomas, B.A. Weerdmeester und I. L. McClelland. Taylor & Francis., 1996. Kapitel SUS: A "quick and dirtyüsability scale, Seiten 189–194.

[BYW12] Lin Bai, Dan Ye und Jun Wei. »A Goal Decomposition Approach for Automatic Mashup Development«. In: *Enterprise Interoperability*. Herausgegeben von Marten van Sinderen, Pontus Johnson, Xiaofei Xu und Guy Doumeingts. Band 122. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2012, Seiten 20–33.

[BZG08] Steffen Bleul, Michael Zapf und Kurt Geihs. »Self-Integration of Web Services in BPEL Processes«. In: *Proceedings of the Workshop SAKS*. März 2008.

[Cao+10] Jill Cao, Yann Riche, Susan Wiedenbeck, Margaret Burnett und Valentina Grigoreanu. »End-user mashup programming: through the design lens«. In: *Proceedings of the 28th international conference on Human factors in computing systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, Seiten 1009–1018.

[Cao13] Chen Cao. »Helping End-User Programmers Help Themselves – The Idea Garden Approach«. Dissertation. Oregon State University, 2013.

[CDP16] Loredana Caruccio, Vincenzo Deufemia und Giuseppe Polese. »A Wizard Based EUDWeb Development Process«. In: *Empowering Organizations: Enabling Platforms and Artefacts*. Herausgegeben von Teresina Torre, Alessio Maria Braccini und Riccardo Spinelli. Cham: Springer International Publishing, 2016, Seiten 173–185.

[CGT11] Nguyen Ngoc Chan, Walid Gaaloul und Samir Tata. »Composition Context Matching for Web Service Recommendation«. In: *2011 IEEE International Conference on Services Computing* (Juli 2011), Seiten 624–631.

[Che+09] Huajun Chen, Bin Lu, Yuan Ni u. a. »Mashup by surfing a web of data APIs«. In: *Proc. VLDB Endow.* 2.2 (Aug. 2009), Seiten 1602–1605.

[Che+16] Michelle Cheatham, Zlatan Dragisic, Jérôme Euzenat u. a. »Results of the Ontology Alignment Evaluation Initiative 2015«. In: *10th ISWC workshop on ontology matching (OM)*. cheatham2016a. Bethlehem, United States, Okt. 2016, Seiten 60–115. URL: https://hal.archives-ouvertes.fr/hal-01254907.

[Cho+14] Zahira Chouiref, Karim Benouaret, Allel Hadjali und Abdelkader Belkhir. »Web Engineering: 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings«. In: Herausgegeben von Sven Casteleyn, Gustavo Rossi und Marco Winckler. Springer International Publishing, 2014. Kapitel Multi Matchmaking Approach for Semantic Web Services Selection Based on Fuzzy Inference, Seiten 440–449.

[Chu+12] Olexiy Chudnovskyy, Tobias Nestler, Martin Gaedke u. a. »End-user-oriented Telco Mashups: The OMELETTE Approach«. In: *Proceedings of the 21st International Conference on World Wide Web*. WWW '12 Companion. New York, NY, USA: ACM, 2012, Seiten 235–238.

[Chu+13] Olexiy Chudnovskyy, Stefan Pietschmann, Matthias Niederhausen u. a. »Awareness and Control for Inter-Widget Communication: Challenges and Solutions«. In: *Web Engineering*. Herausgegeben von Florian Daniel, Peter Dolog und Qing Li. Band 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, Seiten 114–122.

[Cla+01] Mark Claypool, Phong Le, Makoto Wased und David Brown. »Implicit Interest Indicators«. In: *Proceedings of the 6th International Conference on Intelligent User Interfaces*. IUI '01. New York, NY, USA: ACM, 2001, Seiten 33–40.

[CLB07] Oscar Corcho, Silvestre Losada und Richard Benjamins. »Mediation: Bridging between Heterogeneous Web Service Systems«. In: *Semantic Web Services*. Herausgegeben von Rudi Studer, Stephan Grimm und Andreas Abecker. Springer Berlin Heidelberg, 2007, Seiten 287–308.

[Con13a] OMELETTE Consortium. *Deliverable D2.3 Final Specification of Mashup Description Language and Telco Mashup Architecture*. http://www.ict-omelette.eu/c/document_library/get_file?uuid=adc3270d-4521-480e-996e-1b0c4a96307d&groupId=48739. 2013.

[Con13b] OMELETTE Consortium. *Deliverable D5.3 - Final Automatic Discovery and Composition Report*. http://www.ict-omelette.eu/c/document_library/get_file?uuid=7df36613-76be-4714-a862-397084c50eb6&groupId=48739. 2013.

[Cos+03] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan und John Riedl. »Is Seeing Believing?: How Recommender System Interfaces Affect Users' Opinions«. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. New York, NY, USA: ACM, 2003, Seiten 585–592.

[Dan+09] Florian Daniel, Fabio Casati, Boualem Benatallah und Ming-Chien Shan. »Hosted Universal Composition: Models, Languages and Infrastructure in mashArt«. In: *Proceedings of the 28th International Conference on Conceptual Modeling*. Nov. 2009.

[Dan+10] Lars Dannecker, Marius Feldmann, Tobias Nestler u. a. »Rapid Development of Composite Applications Using Annotated Web Services«. In: *Current Trends in Web Engineering: 10th International Conference on Web Engineering ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers*. Herausgegeben von Florian Daniel und Federico Michele Facca. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, Seiten 1–12.

[Di +09] Giusy Di Lorenzo, Hakim Hacid, Hye-young Paik und Boualem Benatallah. »Data Integration in Mashups«. In: *SIGMOD Rec.* 38.1 (Juni 2009), Seiten 59–66.

[EB09] Michael Eckert und François Bry. »Complex Event Processing (CEP)«. In: *Informatik-Spektrum* 32.2 (März 2009), Seiten 163–167.

[Elm+08] H. Elmeleegy, A. Ivan, R. Akkiraju und R. Goodwin. »Mashup Advisor: A Recommendation Tool for Mashup Development«. In: *International Conference on Web Services (ICWS 2008)*. IEEE, Sep. 2008, Seiten 337–344.

[FBN08] Thomas Fischer, Fedor Bakalov und Andreas Nauerz. *Towards an Automatic Service Composition for Generation of User-Sensitive Mashups*. 2008.

[Fel+09] Marius Feldmann, Tobias Nestler, Klemens Muthmann u. a. »Overview of an End User enabled Model-driven Development Approach for Interactive Applications based on Annotated Services«. In: *4th Workshop on Emerging Web Services Technology, ACM International Conference Proceeding Series*. ACM, 2009.

[Fül+10] LJ Fülöp, G Tóth, R Rácz und J Pánczél. *Survey on Complex Event Processing and Predictive Analytics*. Technischer Bericht. Nokia Siemens Networks, 2010.

[Gam+94] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[Ger95] Gloria Gery. »Attributes and Behaviors of Performance-Centered Systems«. In: *Performance Improvement Quarterly* 8.1 (1995), Seiten 47–93.

[Ghi+16] Giuseppe Ghiani, Fabio Paternò, Lucio Davide Spano und Giuliano Pintori. »An environment for End-User Development of Web mashups«. In: *International Journal of Human-Computer Studies* 87 (2016), Seiten 38–64.

[GMP09] Ohad Greenshpan, Tova Milo und Neoklis Polyzotis. »Autocompletion for mashups«. In: *Proc. of the VLDB Endowment* 2.1 (1 Aug. 2009), Seiten 538–549.

[Gov+15] Sten Govaerts, Katrien Verbert, Evgeny Bogdanov u. a. »Lessons Learned from the Development of the ROLE PLE Framework«. In: *Responsive Open Learning Environments: Outcomes of Research from the ROLE Project*. Herausgegeben von Sylvana Kroop, Alexander Mikroyannidis und Martin Wolpers. Springer International Publishing, 2015, Seiten 185–217.

[Gro13] Clemens Große. »Beschreibung von Auslösern für Vorschläge zur Anwendungserweiterung«. Großer Beleg. Technische Universität Dresden, 2013.

[Hea09] Marti A. Hearst. *Search User Interfaces*. Cambridge University Press, 2009.

[Hea99] Marti A. Hearst. »Modern Information Retrieval«. In: Band 463. ACM press, 1999. Kapitel User Interfaces and Visualization, Seiten 257–324.

[HS88] Sandra G. Hart und Lowell E. Staveland. »Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research«. In: *Advances in Psychology* 52 (1988). Human Mental Workload, Seiten 139–183.

[HZ15] F. Hang und L. Zhao. »Supporting End-User Service Composition: A Systematic Review of Current Activities and Tools«. In: *2015 IEEE International Conference on Web Services*. Juni 2015, Seiten 479–486.

[Imr13] Muhammad Imran. »An Effective end-user development approach through domain-specific mashups for Research Impact Evaluation«. Dissertation. Qatar Computing Research Institute, 2013.

[Jan+09] T. Janner, R. Siebeck, C. Schroth und V. Hoyer. »Patterns for Enterprise Mashups in B2B Collaborations to Foster Lightweight Composition and End User Development«. In: *Proceedings of the IEEE International Conference on Web Services (ICWS 2009)*. IEEE, 2009, Seiten 976–983.

[Jan12] Manuel Jany. »Auflösung der Inkompatibilität von Schnittstellen bzw. Daten in Mashups«. Diplomarbeit. Technische Universität Dresden, 2012.

[Jor90] Anker Helms Jorgensen. »Thinking-aloud in user interface design: a method promoting cognitive ergonomics«. In: *Ergonomics* 33.4 (1990), Seiten 501–507.

[JWK14] Gawesh Jawaheer, Peter Weller und Patty Kostkova. »Modeling User Preferences in Recommender Systems: A Classification Framework for Explicit and Implicit User Feedback«. In: *ACM Trans. Interact. Intell. Syst.* 4.2 (Juni 2014), 8:1–8:26.

[KA11] Nicolas Kuchmann-Beauger und Marie-Aude Aufaure. »A Natural Language Interface for Data Warehouse Question Answering«. In: *Natural Language Processing and Information Systems: 16th International Conference on Applications of Natural Language to Information Systems, NLDB 2011, Alicante, Spain, June 28-30, 2011. Proceedings*. Herausgegeben von Rafael Muñoz, Andrés Montoyo und Elisabeth Métais. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, Seiten 201–208.

[KB14] Marius Kaminskas und Derek Bridge. »Measuring surprise in recommender systems«. In: *Proceedings of the Workshop on Recommender Systems Evaluation: Dimensions and Design (Workshop Programme of the 8th ACM Conference on Recommender Systems)*. 2014.

[KC13] Rima Kilany und Maroun Chamoun. »Smart: Semantically mashup rest web services«. In: *ArXiv e-prints* (Nov. 2013).

[Kec+13] Mandy Keck, Martin Herrmann, Andreas Both, Ricardo Gaertner und Rainer Groh. »Improving Motive-Based Search«. In: *Distributed, Ambient, and Pervasive Interactions*. Herausgegeben von Norbert Streitz und Constantine Stephanidis. Band 8028. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, Seiten 439–448.

[KH13] Sri Krishna Kumar und J.A. Harding. »Ontology mapping using description logic and bridging axioms«. In: *Computers in Industry* 64.1 (2013), Seiten 19–28.

[Khr15] Oleksiy Khriyenko. »Customer Feedback System - Evolution towards Semantically-enhanced Systems«. In: *Proceedings of the 11th International Conference on Web Information Systems and Technologies (WEBIST 2015)*. 2015, Seiten 518–525.

[KKS10] Mark Kröll, Christian Körner und Markus Strohmaier. »iTAG: Automatically Annotating Textual Resources with Human Intentions«. In: *Journal of Emerging Technologies in Web Intelligence* 2.4 (2010), Seiten 333–342.

[KM04] Andrew J. Ko und Brad A. Myers. »Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior«. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. New York, NY, USA: ACM, 2004, Seiten 151–158.

[KPW06] Markus Klann, Fabio Paternó und Volker Wulf. »Future Perspectives in End-User Development«. In: *End User Development*. Herausgegeben von Henry Lieberman, Fabio Paternò und Volker Wulf. Band 9. Human-Computer Interaction Series. Springer Netherlands, 2006, Seiten 475–486.

[KSR13] Sandeep Kaur Kuttal, Anita Sarma und Gregg Rothermel. »Debugging Support for End User Mashup Programming«. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. New York, NY, USA: ACM, 2013, Seiten 1609–1618.

[Kuh91] Carol C. Kuhlthau. »Inside the Search Process: Information Seeking from the User's Perspective«. In: *Journal of the American Society for Information Science* 42.5 (Juni 1991), Seite 361.

[Kur13] Ronny Kursawe. »Automatische Klassifizierung von Kompositionsfragmenten hinsichtlich Anwendungsdomäne und Funktionalität«. Großer Beleg. Technische Universität Dresden, 2013.

[KVW16] Denis Kotkov, Jari Veijalainen und Shuaiqiang Wang. »Challenges of Serendipity in Recommender Systems«. In: *Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016)*. 2016, Seiten 251–256.

[LDB15] Angel Lagares Lemos, Florian Daniel und Boualem Benatallah. »Web Service Composition: A Survey of Techniques and Tools«. In: *ACM Comput. Surv.* 48.3 (Dez. 2015), 33:1–33:41.

[Lie+06] Henry Lieberman, Fabio Paternó, Markus Klann und Volker Wulf. »End-User Development: An Emerging Paradigm«. In: *End User Development*. Herausgegeben von Henry Lieberman, Fabio Paternó und Volker Wulf. Band 9. Human-Computer Interaction Series. Springer Netherlands, 2006, Seiten 1–8.

[Liu+10] Xuanzhe Liu, Zhao Qi, Huang Gang, Jin Zhi und Mei Hong. »iMashup: Assisting End-user Programming for the Service-oriented Web«. In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ASE '10. New York, NY, USA: ACM, 2010, Seiten 285–288.

[Liu+11] X. Liu, Q. Zhao, G. Huang, H. Mei und T. Teng. »Composing Data-Driven Service Mashups with Tag-Based Semantic Annotations«. In: *Web Services (ICWS), 2011 IEEE International Conference on*. Juli 2011, Seiten 243–250.

[Liu+14] XuanZhe Liu, Gang Huang, Qi Zhao, Hong Mei und M. Brian Blake. »iMashup: a mashup-based framework for service composition«. In: *Science China Information Sciences* 57.1 (2014), Seiten 1–20.

[Liu+15] X. Liu, Y. Ma, G. Huang u. a. »Data-Driven Composition for Service-Oriented Situational Web Applications«. In: *IEEE Transactions on Services Computing* 8.1 (Jan. 2015), Seiten 2–16.

[Liz+08] David Lizcano, Javier Soriano, Marcos Reyes und Juan J. Hierro. »EzWeb/FAST: reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services«. In: *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*. iiWAS '08. New York, NY, USA: ACM, 2008, Seiten 15–24.

[Liz+11] David Lizcano, Fernando Alonso, Javier Soriano und Genoveva López. »End-User Development Success Factors and their Application to Composite Web Development Environments«. In: *ICONS 2011, The Sixth International Conference on Systems*. 2011, Seiten 99–108.

[Liz+14] D. Lizcano, F. Alonso, J. Soriano und G. López. »A component- and connector-based approach for end-user composite web applications development«. In: *Journal of Systems and Software* 94 (2014), Seiten 108–128.

[Liz+16] David Lizcano, Genoveva López, Javier Soriano und Jaime Lloret. »Implementation of end-user development success factors in mashup development environments«. In: *Computer Standards & Interfaces* 47 (2016), Seiten 1–18.

[LLM11] Liwei Liu, F. Lecue und N. Mehandjiev. »A Hybrid Approach to Recommending Semantic Software Services«. In: *Intl. Conf. on Web Services (ICWS 2011)*. IEEE, Juli 2011, Seiten 379–386.

[LLS02] Hugo Liu, Henry Lieberman und Ted Selker. »Adaptive Hypermedia and Adaptive Web-Based Systems: Second International Conference, AH 2002 Málaga, Spain, May 29–31, 2002 Proceedings«. In: Herausgegeben von Paul De Bra, Peter Brusilovsky und Ricardo Conejo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. Kapitel GOOSE: A Goal-Oriented Search Engine with Commonsense, Seiten 253–263.

[Lu+09] Bin Lu, Zhaohui Wu, Yuan Ni u. a. »sMash: Semantic-based Mashup Navigation for Data API Network«. In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. New York, NY, USA: ACM, 2009, Seiten 1133–1134.

[Ma+13] Yun Ma, Xuan Lu, XuanZhe Liu, XuDong Wang und M. Brian Blake. »Data-driven synthesis of multiple recommendation patterns to create situational Web mashups«. In: *Science China Information Sciences* 56.8 (2013), Seiten 1–16.

[Maa+11] A. Maaradji, H. Hacid, R. Skraba und A. Vakali. »Social Web Mashups Full Completion via Frequent Sequence Mining«. In: *World Congress on Services (SERVICES 2011)*. IEEE, Juli 2011, Seiten 9–16.

[Mae+02] Alexander Maedche, Boris Motik, Nuno Silva und Raphael Volz. »MAFRA — A MApping FRAmework for Distributed Ontologies«. In: *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web: 13th International Conference, EKAW 2002 Sigüenza, Spain, October 1–4, 2002 Proceedings*. Herausgegeben von Asunción Gómez-Pérez und V. Richard Benjamins. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, Seiten 235–250.

[Mar95] Gary Marchionini. *Information seeking in electronic environments*. 9. Cambridge university press, 1995.

[Mat+13] Maristella Matera, Matteo Picozzi, Michele Pini und Marco Tonazzo. »PEUDOM: A Mashup Platform for the End User Development of Common Information Spaces«. In: *Web Engineering*. Herausgegeben von Florian Daniel, Peter Dolog und Qing Li. Band 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, Seiten 494–497.

[Max+07] E. Michael Maximilien, Hernan Wilkinson, Nirmit Desai und Stefan Tai. »A Domain-Specific Language for Web APIs and Services Mashups«. In: *Service-Oriented Computing – ICSOC 2007: Fifth International Conference, Vienna, Austria, September 17-20, 2007. Proceedings*. Herausgegeben von Bernd J. Krämer, Kwei-Jay Lin und Priya Narasimhan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, Seiten 13–26.

[Meh+10a] N. Mehandjiev, F. Lecue, U. Wajid und A. Namoun. »Assisted Service Composition for End Users«. In: *8th European Conference on Web Services (ECOWS 2010)*. IEEE, Dez. 2010, Seiten 131–138.

[Meh+10b] N. Mehandjiev, A. Namoune, U. Wajid, L. Macaulay und A. Sutcliffe. »End User Service Composition: Perceptions and Requirements«. In: *Web Services (ECOWS), 2010 IEEE 8th European Conference on*. Dez. 2010, Seiten 139–146.

[Meh+14] Nikolay Mehandjiev, Abdallah Namoun, Freddy Lécué, Usman Wajid und Georgia Kleanthous. »End Users Developing Mashups«. In: *Web Services Foundations*. Herausgegeben von Athman Bouguettaya, Quan Z. Sheng und Florian Daniel. New York, NY: Springer New York, 2014, Seiten 709–736. ISBN: 978-1-4614-7518-7.

[Mel11] Michele Melchiori. »Hybrid techniques for web APIs recommendation«. In: *Proceedings of the 1st International Workshop on Linked Web Data Management*. LWDM '11. New York, NY, USA: ACM, 2011, Seiten 17–23.

[Mic15] Konrad Michalik. »Visuelle Zusammenfassung atomarer Capabilities«. Bachelorarbeit. Technische Universität Dresden, 2015.

[MRM17] Oliver Mroß, Carsten Radeck und Klaus Meißner. »Towards Exploratory Search Mashups based on Strategic Knowledge«. In: *Proceedings of the Twelfth International Conference on Internet and Web Applications and Services (ICIW 2017)*. XPS, 2017, Seiten 1–6.

[MRT07] E. Michael Maximilien, Ajith Ranabahu und Stefan Tai. »Swashup: Situational Web Applications Mashups«. In: *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion*. OOPSLA '07. New York, NY, USA: ACM, 2007, Seiten 797–798.

[MW07] Gary Marchionini und Ryen White. »Find What You Need, Understand What You Find«. In: *International Journal of Human–Computer Interaction* 23.3 (2007), Seiten 205–237.

[Nag+07] M. Nagarajan, K. Verma, A. P. Sheth und J. A Miller. »Ontology Driven Data Mediation in Web Services«. In: *Intl. Journal of Web Services Research* 4.4 (2007), Seiten 104–126.

[NND10a] A. Namoun, T. Nestler und A. De Angeli. »Service Composition for Non-programmers: Prospects, Problems, and Design Recommendations«. In: *Web Services (ECOWS), 2010 IEEE 8th European Conference on*. Dez. 2010, Seiten 123–130.

[NND10b] Abdallah Namoun, Tobias Nestler und Antonella De Angeli. »Conceptual and Usability Issues in the Composable Web of Software Services«. In: *Current Trends in Web Engineering*. Herausgegeben von Florian Daniel und Federico Facca. Band 6385. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, Seiten 396–407.

[NNS11] Tobias Nestler, Abdallah Namoun und Alexander Schill. »End-user Development of Service-based Interactive Web Applications at the Presentation Layer«. In: *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS '11. New York, NY, USA: ACM, 2011, Seiten 197–206.

[Nus+12] Alexander Nussbaumer, Marcel Berthold, Daniel Dahrendorf u. a. »A Mashup Recommender for Creating Personal Learning Environments«. In: *Advances in Web-Based Learning - ICWL 2012: 11th International Conference, Sinaia, Romania, September 2-4, 2012. Proceedings*. Herausgegeben von Elvira Popescu, Qing Li, Ralf Klamma, Howard Leung und Marcus Specht. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, Seiten 79–88.

[NWM10] Abdallah Namoun, Usman Wajid und Nikolay Mehandjiev. »Service Composition for Everyone: A Study of Risks and Benefits«. In: *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*. Band 6275. LNCS. Springer, 2010, Seiten 550–559.

[OHS06] Michael P. O'Mahony, Neil J. Hurley und Guénolé C.M. Silvestre. »Detecting Noise in Recommender System Databases«. In: *Proceedings of the 11th International Conference on Intelligent User Interfaces*. IUI '06. New York, NY, USA: ACM, 2006, Seiten 109–115.

[OK01] Douglas W. Oard und Jinmook Kim. »Modeling Information Content Using Observable Behavior«. In: *Proceedings of the 64th Annual Conference of the American Society for Information Science and Technology*. 2001, Seiten 481–488.

[Ome02] Borys Omelayenko. »RDFT: A mapping meta-ontology for business integration«. In: *Proceedings of the Workshop on Knowledge Transformation for the Semantic Web (KTSW2002)* (2002).

[Pfl15] Johannes Pflugmacher. »Erklärung funktionaler Zusammenhänge von Komponenten in ad-hoc Mashups«. Bachelorarbeit. Technische Universität Dresden, 2015.

[Pic+10] Matteo Picozzi, Marta Rodolfi, Cinzia Cappiello und Maristella Matera. »Quality-Based Recommendations for Mashup Composition«. In: *Current Trends in Web Engineering*. LNCS. Springer, Juli 2010, Seiten 360–371.

[Pic13] Matteo Picozzi. »End-User Development of Mashups: Models, Composition Paradigms and Tools«. Dissertation. Politecnico di Milano, 2013.

[Pie12] Stefan Pietschmann. »Modellgetriebene Entwicklung adaptiver, komponentenbasierter Mashup-Anwendungen«. Dissertation. Technische Universität Dresden, 2012.

[PRM11a] Stefan Pietschmann, Carsten Radeck und Klaus Meißner. »Facilitating Context-Awareness in Composite Mashup Applications«. In: *ADAPTIVE 2011, The Third International Conference on Adaptive and Self-Adaptive Systems and Applications*. XPS, 2011, Seiten 1–8.

[PRM11b] Stefan Pietschmann, Carsten Radeck und Klaus Meißner. »Semantics-Based Discovery, Selection and Mediation for Presentation-Oriented Mashups«. In: *Proceedings of the 5th International Workshop on Web APIs and Service Mashups - Mashups '11*. New York, New York, USA: ACM Press, 2011, Seite 1.

[Pru07] Mark Pruett. *Yahoo! Pipes*. First. O'Reilly, 2007. ISBN: 9780596514532.

[PV13] Ladislav Peska und Peter Vojtas. »Negative Implicit Feedback in e-Commerce Recommender Systems«. In: *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*. WIMS '13. New York, NY, USA: ACM, 2013, 45:1–45:4.

[Rad+12] Carsten Radeck, Alexander Lorz, Gregor Blichmann und Klaus Meißner. »Hybrid Recommendation of Composition Knowledge for End User Development of Mashups«. In: *ICIW 2012, The Seventh International Conference on Internet and Web Applications and Services*. XPS, 2012, Seiten 30–33.

[Rad+14] Carsten Radeck, Gregor Blichmann, Mroß Oliver und Klaus Meißner. »Semantic Mediation Techniques for Composite Web Applications«. In: *Web Engineering*. Herausgegeben von Sven Casteleyn, Gustavo Rossi und Marco Winckler. Band 8541. Lecture Notes in Computer Science. Springer International Publishing, 2014, Seiten 450–459.

[RBM13] Carsten Radeck, Gregor Blichmann und Klaus Meißner. »CapView – Functionality-Aware Visual Mashup Development for Non-programmers«. In: *Web Engineering*. Herausgegeben von Florian Daniel, Peter Dolog und Qing Li. Band 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, Seiten 140–155.

[RBM16] Carsten Radeck, Gregor Blichmann und Klaus Meißner. »Estimating the Functionality of Mashup Applications for Assisted, Capability-centered End User Development«. In: *Proceedings of the 12th International Conference on Web Information Systems and Technologies - Volume 2: WEBIST*. SciTePress, 2016, Seiten 109–120.

[RBM17] Carsten Radeck, Gregor Blichmann und Klaus Meißner. »Modeling and Calculating Capabilities of Composite Web Applications for Assisted End User Development«. In: *Web Information Systems and Technologies (WEBIST 2016) - Revised Selected Papers*. Herausgegeben von Valérie Monfort, Karl-Heinz Krempels, Tim A. Majchrzak und Paolo Traverso. Band 292. Lecture Notes in Business Information Processing (LNBIP). Springer International Publishing, 2017, Seiten 58–82.

[RDC14] Soudip Roy Chowdhury, Florian Daniel und Fabio Casati. »Recommendation and Weaving of Reusable Mashup Model Patterns for Assisted Development«. In: *ACM Trans. Internet Technol.* 14.2-3 (Okt. 2014), 21:1–21:23.

[Ria+08] Anton V. Riabov, Eric Boillet, Mark D. Feblowitz, Zhen Liu und Anand Ranganathan. »Wishful Search: Interactive Composition of Data Mashups«. In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. New York, NY, USA: ACM, 2008, Seiten 775–784.

[RM17a] Carsten Radeck und Klaus Meißner. »A Customizable Recommender System for Mashup Platforms«. In: *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services (iiWAS2017)*. ACM, Dez. 2017, Seiten 66–75.

[RM17b] Carsten Radeck und Klaus Meißner. »Helping Non-Programmers to Understand the Functionality of Composite Web Applications«. In: *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*. SciTePress, 2017, Seiten 109–120.

[RM18] Carsten Radeck und Klaus Meißner. »Assisted End User Development for Non-programmers: Awareness, Exploration and Explanation of Composite Web Application Functionality«. In: *Web Information Systems and Technologies (WEBIST 2017) - Revised Selected Papers*. Herausgegeben von Tim A. Majchrzak, Paolo Traverso, Karl-Heinz Krempels und Valérie Monfort. Band 322. Lecture Notes in Business Information Processing (LNBIP). Springer International Publishing, 2018, Seiten 249–275.

[Ro+08] Agnes Ro, Lily Xia, Hye-Young Paik und Chea Chon. »Bill Organiser Portal: A Case Study on End-User Composition«. In: *Web Information Systems Engineering WISE 2008 Workshops*. Herausgegeben von Sven Hartmann, Xiaofang Zhou und Markus Kirchberg. Band 5176. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, Seiten 152–161.

[RO14] Andy Ridge und Eamonn O'Neill. »Establishing requirements for End-user Service Composition tools«. In: *Requirements Engineering* 20.4 (2014), Seiten 435–463.

[Rod+14] Carlos Rodríguez, Soudip Roy Chowdhury, Florian Daniel, Hamid R. Motahari Nezhad und Fabio Casati. »Web Services Foundations«. In: Herausgegeben von Athman Bouguettaya, Z. Quan Sheng und Florian Daniel. Springer New York, 2014. Kapitel Assisted Mashup Development: On the Discovery and Recommendation of Mashup Composition Knowledge, Seiten 683–708.

[Roy+10] Soudip Roy Chowdhury, Carlos Rodríguez, Florian Daniel und Fabio Casati. »Wisdom-Aware Computing: On the Interactive Recommendation of Composition Knowledge«. In: *Service-Oriented Computing*. Herausgegeben von E. Maximilien, Gustavo Rossi, Soe-Tsyr Yuan, Heiko Ludwig und Marcelo Fantinato. Band 6568. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, Seiten 144–155.

[Roy+13] Soudip Roy Chowdhury, Olexiy Chudnovskyy, Matthias Niederhausen u. a. »Complementary Assistance Mechanisms for End User Mashup Composition«. In: *Proceedings of the 22nd International Conference on World Wide Web Companion*. WWW '13 Companion. Republic und Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, Seiten 269–272.

[Rüm+11] Andreas Rümpel, Carsten Radeck, Gregor Blichmann, Alexander Lorz und Klaus Meißner. »Towards Do-It-Yourself Development of Composite Web Applications«. In: *Proceedings of the International Conference on Internet Technologies & Society 2011 (ITS 2011)*. 2011, Seiten 231–235.

[Rüm+13] Andreas Rümpel, Vincent Tietz, Anika Wagner und Klaus Meißner. »Modeling and Utilizing Quality Properties in the Development of Composite Web Mashups«. In: *Current Trends in Web Engineering*. Herausgegeben von QuanZ. Sheng und Jesper Kjeldskov. Band 8295. Lecture Notes in Computer Science. Springer International Publishing, Juli 2013, Seiten 54–65.

[Rüm+14] Andreas Rümpel, Carsten Radeck, Juri Tichomirow und Klaus Meissner. »Fuzzy Mashup Quality Requirements Specification for Web Users«. In: *Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the*. Sep. 2014, Seiten 262–267.

[SB05] François Scharffe und Jos de Bruijn. »A language to specify mappings between ontologies«. In: *SITIS*. 2005, Seiten 267–271.

[SE05] Pavel Shvaiko und Jérôme Euzenat. »A Survey of Schema-Based Matching Approaches«. In: *Journal on Data Semantics IV*. Band 3730. LNCS. Springer Berlin Heidelberg, 2005. Kapitel 5, Seiten 146–171.

[SE13] P. Shvaiko und J. Euzenat. »Ontology Matching: State of the Art and Future Challenges«. In: *IEEE Transactions on Knowledge and Data Engineering* 25.1 (Jan. 2013), Seiten 158–176.

[SE98] Alistair Sutcliffe und Mark Ennis. »Towards a cognitive theory of information retrieval«. In: *Interacting with Computers* 10.3 (1998), Seiten 321–351.

[Sir+09] Stéphane Sire, Micael Paquier, Alain Vagner und Jérôme Bogaerts. »A Messaging API for Inter-widgets Communication«. In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. New York, NY, USA: ACM, 2009, Seiten 1115–1116.

[Sko+09] Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, Verena Kantere und Timos Sellis. »Top-k Dominant Web Services Under Multi-criteria Matching«. In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. EDBT '09. ACM, 2009, Seiten 898–909.

[SPM06] M. Szomszor, T.R. Payne und L. Moreau. »Automated Syntactic Medation for Web Service Integration«. In: *Proceedings of the International Conference on Web Services*. Sep. 2006, Seiten 127–136.

[SPS10] Eduardo Silva, Luís Pires und Marten van Sinderen. »On the Support of Dynamic Service Composition at Runtime«. In: *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*. Herausgegeben von Asit Dan, Frédéric Gittler und Farouk Toumani. Band 6275. LNCS. Springer Berlin / Heidelberg, 2010, Seiten 530–539.

[SS11] E. Isaac Sparling und Shilad Sen. »Rating: How Difficult is It?« In: *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. New York, NY, USA: ACM, 2011, Seiten 149–156.

[Sto+06] Michael Stollberg, Emilia Cimpian, Adrian Mocan und Dieter Fensel. »A Semantic Web Mediation Architecture«. In: *Canadian Semantic Web*. Herausgegeben von Mamadou Tadiou Koné und Daniel Lemire. Boston, MA: Springer US, 2006, Seiten 3–22.

[Str08] Markus Strohmaier. »Purpose Tagging: Capturing User Intent to Assist Goal-oriented Social Search«. In: *Proceedings of the 2008 ACM Workshop on Search in Social Media*. SSM '08. New York, NY, USA: ACM, 2008, Seiten 35–42.

[Sut05] Alistair Sutcliffe. »Evaluating the Costs and Benefits of End-user Development«. In: *SIGSOFT Softw. Eng. Notes* 30.4 (Mai 2005), Seiten 1–4.

[Tan+16] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain und S. Dustdar. »From the Service-Oriented Architecture to the Web API Economy«. In: *IEEE Internet Computing* 20.4 (Juli 2016), Seiten 64–68.

[Tie+12] Vincent Tietz, Gregor Blichmann, Stefan Pietschmann und Klaus Meißner. »Current Trends in Web Engineering: Workshops, Doctoral Symposium, and Tutorials held at ICWE 2011«. In: Herausgegeben von Andreas Harth und Nora Koch. Springer Berlin Heidelberg, 2012. Kapitel Task-Based Recommendation of Mashup Components, Seiten 25–36.

[Tie+13] Vincent Tietz, Oliver Mroß, Andreas Rümpel, Carsten Radeck und Klaus Meißner. »A Requirements Model for Composite and Distributed Web Mashups«. In: *Proceedings of the Eighth International Conference on Internet and Wen Applications and Services (ICIW 2013)*. XPS, Juni 2013, Seiten 75–82.

[Tie15] Vincent Tietz. »Aufgabenbasierte Komposition von User-Interface-Mashups«. Dissertation. Technische Universität Dresden, 2015.

[Ton+13] Alberto Tonon, Michele Catasta, Gianluca Demartini, Philippe Cudré-Mauroux und Karl Aberer. »TRank: Ranking Entity Types Using the Web of Data«. In: *Proceedings of the 12th International Semantic Web Conference - Part I*. ISWC '13. Springer-Verlag New York, Inc., 2013, Seiten 640–656.

[Tri+14] Tuan-Dat Trinh, Peter Wetz, Ba-Lam Do u. a. »A Web-based Platform for Dynamic Integration of Heterogeneous Data«. In: *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2014)*. iiWAS '14. New York, NY, USA: ACM, 2014, Seiten 253–261.

[Tri+15] Tuan-Dat Trinh, Peter Wetz, Ba-Lam Do, Elmar Kiesling und A. Min Tjoa. »Semantic Mashup Composition from Natural Language Expressions: Preliminary Results«. In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015)*. iiWAS '15. ACM, 2015, 44:1–44:9.

[Tsc+14] Alexey Tschudnowsky, Stefan Pietschmann, Matthias Niederhausen, Michael Hertel und Martin Gaedke. »From Choreographed to Hybrid User Interface Mashups: A Generic Transformation Approach«. In: *Web Engineering: 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*. Herausgegeben von Sven Casteleyn, Gustavo Rossi und Marco Winckler. Springer International Publishing, 2014, Seiten 145–162.

[Tsc14] Sergej Tschigraj. »Aggregierte Nutzung von Mashup-Qualitätseigenschaften auf Anwendungsebene«. Bachelorarbeit. Technische Universität Dresden, 2014.

[TTA11] Boris Tapia, Romina Torres und Hernán Astudillo. »Simplifying mashup component selection with a combined similarity- and social-based technique«. In: *Proceedings of the 5th International Workshop on Web APIs and Service Mashups (Mashups 2011)*. 2011.

[VFM13] Martin Voigt, Martin Franke und Klaus Meißner. »Capturing and Reusing Empirical Visualization Knowledge«. In: *Proceedings of the 1st International Workshop on User-Adaptive Visualization (WUAV)*. 2013.

[VGG13] A. Vozniuk, S. Govaerts und D. Gillet. »Towards Portable Learning Analytics Dashboards«. In: *2013 IEEE 13th International Conference on Advanced Learning Technologies*. Juli 2013, Seiten 412–416.

[Voi+12] Martin Voigt, Stefan Pietschmann, Lars Grammel und Klaus Meißner. »Context-aware Recommendation of Visualization Components«. In: *Proceedings of the 4th International Conference on Information, Process, and Knowledge Management (eKNOW 2012)*. XPS, Feb. 2012, Seiten 101–109.

[Voi14] Martin Voigt. »Kontextsensitive Informationsvisualisierung mit kompositen Rich Internet Applications für Endnutzer«. Dissertation. Technische Universität Dresden, 2014.

[Wan+15] G. Wang, Y. Han, Z. Zhang und S. Zhang. »A Dataflow-Pattern-Based Recommendation Framework for Data Service Mashup«. In: *IEEE Transactions on Services Computing* 8.6 (Nov. 2015), Seiten 889–902.

[Wec16] Stefan Weckend. »Erhöhung der Plausibilität von Empfehlungen bei der Komposition von Mashups«. Bachelorarbeit. Technische Universität Dresden, 2016.

[Wil+12] Scott Wilson, Florian Daniel, Uwe Jugel und Stefano Soi. »Orchestrated User Interface Mashups Using W3C Widgets«. In: *Proceedings of the 11th International Conference on Current Trends in Web Engineering*. ICWE'11. Berlin, Heidelberg: Springer-Verlag, 2012, Seiten 49–61.

[WJ04] Volker Wulf und Matthias Jarke. »The economics of end-user development«. In: *Commun. ACM* 47.9 (Sep. 2004), Seiten 41–42.

[WLZ13] Bifan Wei, Jun Liu und Qinghua Zheng. »A Survey of Faceted Search«. In: *Journal of Web Engineering* 12.1&2 (2013), Seiten 41–64.

[Yan+12] Jianyu Yang, Jun Han, Xu Wang und Hailong Sun. »MashStudio: An On-the-fly Environment for Rapid Mashup Development«. In: *Internet and Distributed Computing Systems*. Herausgegeben von Yang Xiang, Mukaddim Pathan, Xiaohui Tao und Hua Wang. Band 7646. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, Seiten 160–173.

[Zha+10] Chenting Zhao, Chun'e Ma, Jing Zhang u. a. »HyperService: Linking and Exploring Services on the Web«. In: *Intl. Conf. on Web Services (ICWS 2010)*. IEEE, Juli 2010, Seiten 17–24.

Webreferenzen

[@Apiant] *Apiant*. Abgerufen: 30.09.2016. 2016. URL: https://apiant.com/.

[@ASWG] *Activity Streams*. Abgerufen: 20.12.2016. Mai 2011. URL: http://activitystrea.ms/.

[@Axis2] *Apache Axis2*. Abgerufen: 08.01.2016. 2010. URL: http://axis.apache.org/.

[@BA] British Airways. *Picture Your Holiday*. Abgerufen: 10.10.2016. 2016. URL: http://pictureyourholiday.ba.com/.

[@DoC] *Projektwebseite*. Abgerufen: 14.01.2016. 2013. URL: http://mmt.inf.tu-dresden.de/Forschung/Projekte/DoCUMA/.

[@EDYRA] *Projektwebseite*. Abgerufen: 08.01.2016. 2014. URL: http://mmt.inf.tu-dresden.de/Forschung/Projekte/EDYRA/.

[@Goo] Google Inc. *ZDFmediathek*. Abgerufen: 14.09.2017. 2017. URL: https://www.google.de/.

[@Iberia] Iberia. *Inspirational Search Engine*. Abgerufen: 10.10.2016. 2016. URL: http://www.iberia.com/gb/inspire-me/.

[@IFTTT] *IFTTT*. Abgerufen: 08.01.2016. 2016. URL: http://ifttt.com/.

[@Jena] *Apache Jena*. Abgerufen: 08.01.2016. 2015. URL: http://jena.apache.org/.

[@JUnit] *JUnit*. Abgerufen: 08.01.2016. 2015. URL: http://junit.org/.

[@MDN] *The Places frecency algorithm*. Abgerufen: 07.04.2016. 2014. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/Places/Frecency_algorithm.

[@Mer] Duane Merrill. *Mashups: The new breed of Web app*. letzter Zugriff: 26.02.2011. Aug. 2006. URL: http://www.ibm.com/developerworks/library/x-mashups.html.

[@OWL] W3C. *OWL Web Ontology Language — Semantics and Abstract Syntax*. Abgerufen: 06.10.2016. URL: https://www.w3.org/TR/owl-semantics/.

[@QUDT] *QUDT – Quantities, Units, Dimensions and Data Types Ontologies*. 2013. URL: http://qudt.org/.

[@RDFS] W3C. *RDF Schema 1.1*. Abgerufen: 06.10.2016. URL: https://www.w3.org/TR/rdf-schema/.

[@RDFSe] W3C. *RDF 1.1 Semantics*. Abgerufen: 06.10.2016. URL: https://www.w3.org/TR/rdf11-mt/.

[@SAWSDL] W3C. *Semantic Annotations for WSDL and XML Schema*. Abgerufen: 07.10.2016. URL: https://www.w3.org/TR/sawsdl/.

[@SKOS] *SKOS Simple Knowledge Organization System Reference*. Abgerufen: 02.08.2016. 2009. URL: https://www.w3.org/TR/skos-reference/.

[@SoapUI] *SoapUI*. Abgerufen: 08.01.2016. 2016. URL: http://www.soapui.org/.

[@WA] Wolfram Alpha LLC. *Wolfram|Alpha: Computational Knowledge Engine*. Abgerufen: 14.09.2017. 2017. URL: https://www.wolframalpha.com/.

[@Wil] Scott Wilson. *Design challenges for user-interface mashups: user control and usability in inter-widget communications*. Abgerufen: 29.09.2016. März 2012. URL: https://scottbw.wordpress.com/2012/03/07/design-challenges-for-user-interface-mashups-user-control-and-usability-in-inter-widget-communications/.

[@WSMO] Adrian Mocan, Emilia Cimpian, Michael Stollberg, Francois Scharffe und James Scicluna. *WSMO Mediators*. WSMO working group. Dez. 2005. URL: http://www.wsmo.org/TR/d29/.

[@WSoc] W3C. *The WebSocket API*. Abgerufen: 08.08.2016. URL: https://www.w3.org/TR/websockets/.

[@ZDF] Zweites Deutsches Fernsehen. *ZDFmediathek*. Abgerufen: 14.09.2017. 2017. URL: https://www.zdf.de/.

[Nie00] Jakob Nielsen. *Why You Only Need to Test with 5 Users*. Last accessed 02.01.2017. 2000. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[Spe06] Donna Spencer. *Four Modes of Seeking Information and How to Design for Them*. Abgerufen: 18.08.2016. 2006. URL: http://boxesandarrows.com/four-modes-of-seeking-information-and-how-to-design-for-them/.