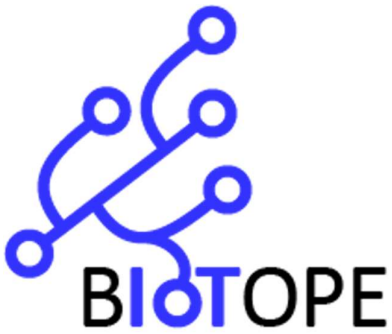


BIoTope – Basistechnologie und Engineering-Methodik für die emergente Genese und semantische Komposition von IoT-Ökosystemen

Schlussbericht BIoTope

Förderkennzeichen		01IS18079
Förderkennzeichen der Projektpartner	STO	01IS18079A
	WAG	01IS18079B
	UMA	01IS18079C
	TUC	01IS18079D
Laufzeit des Vorhabens		01.06.2019 bis 31.12.2022
Berichtszeitraum		01.06.2019 bis 31.12.2022

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Autoren (A-Z)

Burggraf, Annelore	Wolfsburg AG
Gotthardt, Roger	StoneOne AG
Knieke, Christoph	Technische Universität Clausthal, Institut für Software and Systems Engineering
Nyakam Chiadjeu, Eric Douglas	Technische Universität Clausthal, Institut für Software and Systems Engineering
Schindler, Christian	Technische Universität Clausthal, Institut für Software and Systems Engineering
Wilken, Nils	Universität Mannheim, Institut für Enterprise Systeme
Ziebura, Nikolaus	StoneOne AG

Ergebnisverbreitung

Öffentlich
X BMBF
X Projektträger
X Projektpartner

Dateiname BIoTope_Abschlussbericht.docx

Seitenzahl 33

Inhalt

1 Zusammenfassung

Ziel des Projektes ist die Entwicklung einer Basistechnologie und Engineering-Methodik, um die Genese von emergenten Services auf selbstadaptiven Systemplattformen zu ermöglichen. Dabei sollen die Kompositionsregeln nicht zentral und statisch durch die Plattform vorgegeben werden, sondern dynamisch und nachfrageorientiert konfiguriert werden können. Zudem sollen insbesondere offene Systeme, deren Schnittstellen nicht durch ein einheitliches semantisches Modell standardisiert sind, durch den zu entwickelnden FuE-Ansatz unterstützt werden, damit auch IoT-Systeme ohne Branchenstandard integriert werden können.

Um zur Systemlaufzeit eine emergente, bedarfs-orientierte Komposition von vorhandenen Softwareservices zu ermöglichen, ist es notwendig, dass die Bedarfe und Anforderungen der potenziellen Nutzer dieser emergent komponierten Services automatisiert erfasst werden können. Hierzu soll im Projekt eine Methode entwickelt werden, die auf der Basis von observierten Nutzeraktionen in einem IoT-Öko-system die Nutzerbedarfe und -anforderungen automatisiert erkennen kann.

Weiterhin bedarf es der Entwicklung einer emergenten Plattformtechnologie. Diese Technologie ist für selbstadaptive emergente IoT-Systeme besonders wichtig, wenn keine einheitliche semantische Standardisierung vorhanden ist, aber die Selbstadaptions-mechanismen auf eine maschinelle semantische Interpretation von Daten, Service und Prozess-beschreibungen angewiesen sind. Dafür soll das DevOps-Paradigma von der Software-entwicklung auf die semantische Standardisierung übertragen werden, um eine bedarfs-orientierte interaktive Integration semantischer Modelle zur Systemlaufzeit zu ermöglichen.

Zur semantischen Anbindung von IoT-Komponenten an die Plattform wird eine softwaregestützte Engineering-Methodik im Projekt entwickelt. Als möglichen Lösungsansatz zu den zuvor beschriebenen Herausforderungen schlagen wir eine Architektur für eine emergente Softwareplattform vor. Eine Softwareplattform, die diese Architektur mit der im Projekt entwickelten Engineering-Methodik implementiert, ist in der Lage, die zuvor definierten Voraussetzungen für eine emergente Softwareplattform zu erfüllen.

Im Projekt wurde die entwickelte IoT-Plattform in mehreren Bereichen angewendet. In einem Use Case wurde ein smartes Parkhaus betrachtet. Im Rahmen dieses Demonstrationsszenarios wurden Services beschrieben, die ein Nutzer eines Parkhauses einzeln oder in Kombination in Anspruch nehmen kann. Außerdem wurde ein Evaluationsszenario im Bereich Smart Office direkt an der Universität Mannheim aufgebaut. In diesem Evaluationsszenario konnte vor allem die implizite Erkennung von Nutzeranforderungen demonstriert und evaluiert werden.

2 Herausforderungen und Lösungsansatz

Die Laufzeitumgebung aktueller Softwaresysteme, wie moderne eingebettete Systeme oder Informationssysteme, besteht aus einer Vielzahl von komplexen Softwarekomponenten, die in einer verteilten Architektur auf verschiedenen Hardwarekomponenten ausgeführt werden. In einer verteilten Systemarchitektur kann jede dieser Komponenten verschiedene Anwendungen oder Services über eine Netzwerkverbindung bereitstellen. Im Kontext dieser Laufzeitumgebung, welche auch „Internet of Things“ (IoT) genannt wird, sind neuartige Softwareplattformen entstanden, die verschiedene Anbieter von Software-Services und deren Anwender an einem zentralen Ort zusammenbringen und das Anbieten und Nutzen von Services vereinfachen.

Eine wichtige Funktionalität dieser Plattformen ist die Software-Service-Komposition, bei der verfügbare Software-Services zu einer neuen, höherwertigen Anwendung verknüpft werden. Allerdings ermöglichen aktuelle Plattformen nur eine manuelle Komposition, was eine Hürde für Nutzer darstellen kann, die nicht über die nötige technische Expertise verfügen.

Im Rahmen des BIoTope Projekts wurde untersucht, wie eine vollautomatische Software-Service-Komposition auf der Basis von explizit oder implizit erkannten Nutzeranforderungen durchgeführt werden kann. Außerdem wurde die anschließende vollautomatische Ausführung des komponierten Services untersucht. Zum Beschreiben einer solchen Softwareplattform wurde im Rahmen des Projekts der Begriff „Emergente Softwareplattform“ definiert.

Die folgenden Unterkapitel beschreiben die wesentlichen technischen Herausforderungen, mit denen sich die Projektpartner bei der Umsetzung auseinandergesetzt haben und skizzieren die Architektur einer Softwareplattform, die in der Lage ist, diese Herausforderungen zu bewältigen.

2.1 Herausforderungen

Eine emergente Softwareplattform muss zur Laufzeit stetig auf Veränderungen in der Laufzeitumgebung, wie das Hinzukommen/Wegfallen von Software-Services oder veränderte Nutzeranforderungen, reagieren können. Daher entstehen alle Software-Service-Kompositionen erst zur Laufzeit der Plattform auf der Basis der automatisch erkannten Nutzeranforderungen. Im Projekt wurde genau diese Eigenschaft als „Emergenz“ im Kontext einer Softwareplattform definiert. Formaler definieren wir Emergenz wie folgt: *Eine Softwareplattform wird emergent genannt, wenn die Plattform in der Lage ist, als Antwort auf ein auslösendes Ereignis die verfügbaren Software-Services automatisch und dynamisch zu einem höherwertigen Software-Service zu komponieren. Das so entstehende Verhalten der Plattform wird nicht zur Design Zeit vordefiniert und kann nicht durch einzelne Software-Services vorhergesehen werden.*

Eine emergente Softwareplattform muss verschiedene Herausforderungen erfolgreich bewältigen können, um die erforderlichen Funktionalitäten im IoT Kontext bereitstellen zu können:

- **Dynamischer Kontext von IT-Ökosystemen**: Die möglichen Interaktionen sind nicht im Voraus planbar. Services unterliegen unterschiedlichen Lebenszyklen. Dazukommen bzw. Wegfallen ist denkbar und beeinflusst die möglichen Kombinationen.

- **Entstehung emergenter Funktionen und Services durch Aggregationen**: Zuverlässigkeit und Akzeptanz sind Herausforderungen im Vergleich zu klassischen Systemen (Systemgarantien sind hier meist querschnittlich).

- **Dynamisch Adaptive Systemeigenschaften**: Laufzeitveränderungen von IoT Services sind nicht vorhersehbar. Anforderungen und Wünsche der Nutzer können sich ändern.

- **Analyse von Nutzeranforderungen**: Wie können Nutzeranforderungen automatisch von unstrukturierten Daten abgeleitet werden? Wie können Nutzeranforderungen formalisiert werden?

- **Kompositionsmechanismus**: Wie können vorhandene Software-Services komponiert werden? Wie können Software-Services formal beschrieben werden, um eine automatische Komposition zu ermöglichen?

- **Integration in Service Registry**: Wie werden neue Software-Services in die Plattform integriert? Wie kann die semantische Interoperabilität zwischen vorhandenen Software-Services automatisch sichergestellt werden?

2.2 Lösungsansatz

Als möglichen Lösungsansatz zu den zuvor beschriebenen Herausforderungen schlagen wir die in Abbildung 1 abgebildete Architektur einer emergenten Softwareplattform vor. Die Architektur umfasst im Wesentlichen vier logische Komponenten, welche im Folgenden jeweils kurz beschrieben werden. Weitere Details zu den spezifischen Herausforderungen, der technischen Umsetzung und Forschungsergebnissen folgen in Kapitel 4.

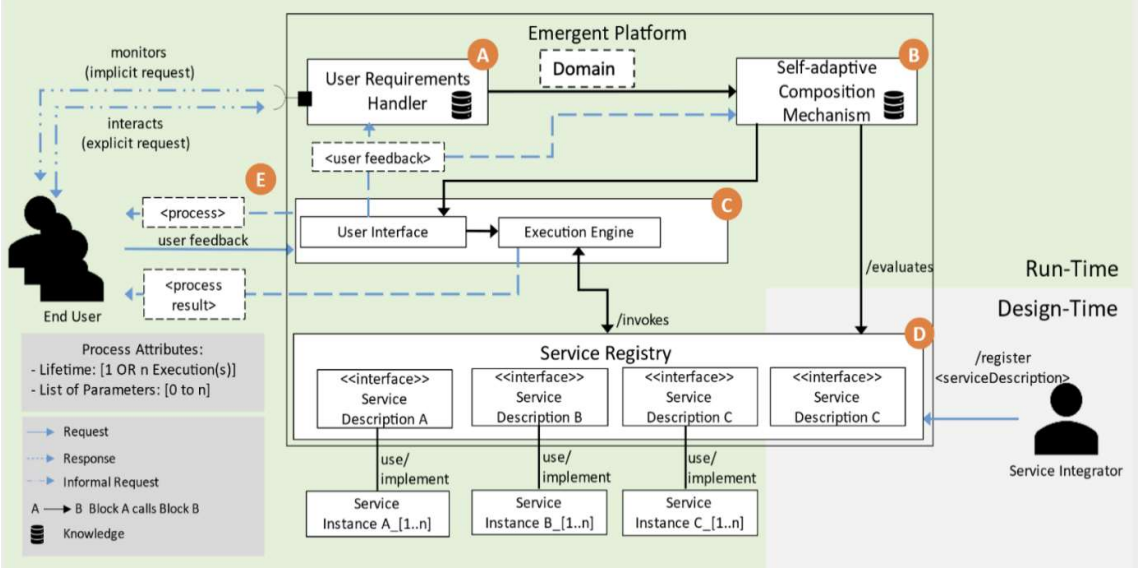


Abb. 1: Gesamtarchitektur

User Requirements Handler. Die User Requirements Handler (URH) Komponente ist dafür zuständig, die aktuellen Nutzeranforderungen zur Laufzeit der emergenten Softwareplattform automatisch zu erkennen. Dies kann dabei entweder implizit oder explizit geschehen. Im expliziten Fall stellt der Nutzer aktiv eine Anfrage, welche seine aktuellen Anforderungen ausdrückt, in einem tendenziell unstrukturierten Format (z.B. natürliche Sprache). Im impliziten Fall werden die Nutzer der Plattform durchgehend über vorhandene Sensorinfrastruktur überwacht. Aus den so entstehenden Sensormessdaten werden dann die darin implizit enthaltenen Nutzeranforderungen abgeleitet.

Self-adaptive Composition Mechanism. Die Self-adaptive Composition Mechanism (SCM) Komponente ist zuständig für die automatische Komposition eines höherwertigen Software-Service, welcher die aktuellen Nutzeranforderungen, die von der URH Komponente erkannt wurden, erfüllt. Dabei kann die SCM Komponente auf alle momentan auf der emergenten Softwareplattform verfügbaren Software-Services zurückgreifen.

Execution Engine. Die Execution Engine (EE) Komponente ist für die automatische Ausführung eines von der SCM Komponente automatisch komponierten Software-Services zuständig. Eine wesentliche Herausforderung dabei ist, die tatsächlich verwendeten Software-Service Instanzen, welche zur Ausführung des komponierten Software-Service verwendet werden sollen, auszuwählen.

Service Registry. Die Aufgabe der Service Registry (SR) Komponente ist es, eine strukturierte Wissensbasis über die aktuell in der emergenten Softwareplattform verfügbaren Software-Services für die anderen Komponenten der Plattform zur Verfügung zu stellen.

Durch diese vier Komponenten ist eine Softwareplattform, welche die oben gezeigte Architektur implementiert, in der Lage, die zuvor definierten Voraussetzungen für eine emergente Softwareplattform zu erfüllen. Weiterhin sieht die beschriebene Architektur die Interaktion mit zwei verschiedenen Rollen vor. Diese sind zum einen ein Endnutzer (End User) und zum anderen ein Serviceintegrator (Service Integrator). Mit dem Endnutzer hat die Plattform zwei mögliche Formen der Interaktion, welche allerdings beide über ein zentrales Interface abgebildet werden können. Die erste Form der Interaktion findet im Kontext der Erkennung der Nutzeranforderungen durch die URH Komponente wie zuvor beschrieben statt. Die zweite Interaktionsform findet im Kontext einer Feedbackschleife bezüglich des Ergebnisses einer Ausführung eines automatisch komponierten Software-Service zwischen Endnutzer und EE statt. Dadurch hat der Endnutzer die Möglichkeit, der Plattform Feedback bezüglich der Qualität des automatisch komponierten Software-Service zu geben. Die Rolle des Serviceintegrator umfasst die Aufgaben, die nötig sind, um neue Software-Services semantisch in die emergente Softwareplattform zu integrieren. Da bisher keine vollautomatische semantische Integration von neuen Software-Services in eine Softwareplattform zur Verfügung steht, ist für diesen Vorgang zur Zeit noch manuelle Unterstützung durch einen Serviceintegrator notwendig.

2.3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die Arbeiten bezüglich Plattform und der Komponenten, sowie die Integrationsaufwände waren notwendig, um die Plattform zu testen und dann auch demonstrieren zu können. Die verschiedenen Komponenten waren sowohl zum Test als auch der Demonstration der Usecases ebenfalls notwendig. Insgesamt waren die Arbeiten angemessen, um die Ziele sowohl für die Plattform, die Komponenten (Workflow Designer, Execution Engine, Usecases) zu erreichen und auch die Zusammenführung der verschiedenen Komponenten der Partner in ihrem jeweiligen Entwicklungsstand zu evaluieren und bzgl. der Schnittstellen auch jeweils anzupassen.

2.4 Projektpartner und Arbeitsteilung

Die nachfolgenden Unterabschnitte enthalten eine Kurzvorstellung und Verortung in der Architektur für jeden der beteiligten Projektpartner.

2.4.1. Universität Mannheim

Die Universität Mannheim war im Projekt BIoTope hauptsächlich an der Erforschung und Entwicklung der User Requirements Handler Komponente sowie, zusammen mit der TU Clausthal, einer Methode zur iterativen semantischen Integration von Software Services verantwortlich. Dabei wurde im Bereich der User Requirements Handler Komponente der wissenschaftliche Fokus vor allem auf die automatische implizite Erkennung von Nutzeranforderungen gelegt. Weiterhin wurde durch die Universität Mannheim ein Evaluationsszenario für die in BIoTope entstandenen Ansätze und Forschungsprototypen im Bereich Smart Office aufgebaut.

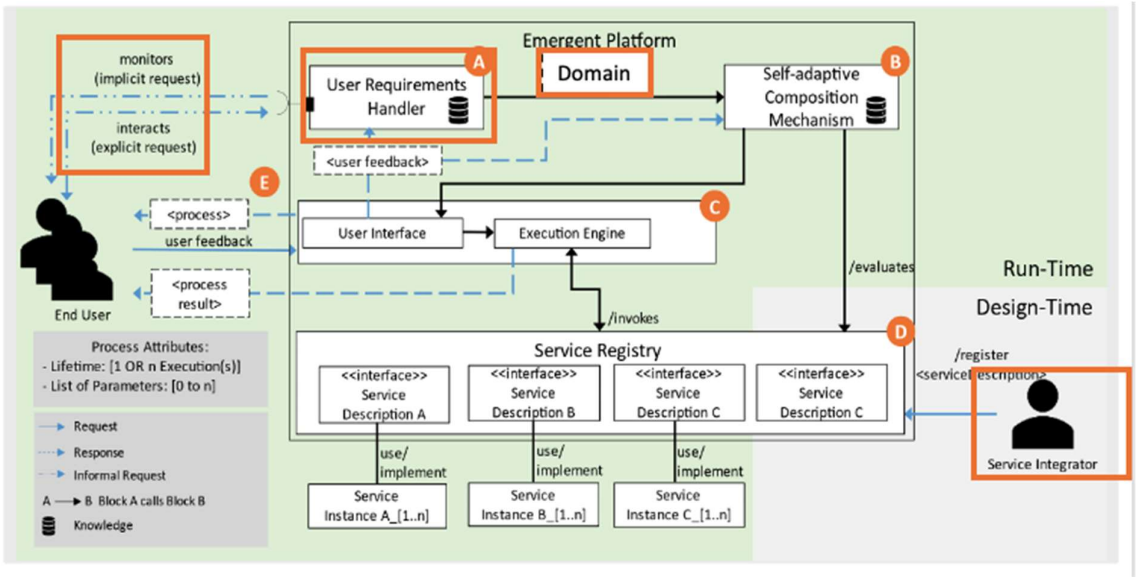


Abb. 2: User Requirements Handler

2.4.2. Wolfsburg AG

Die Wolfsburg AG vertritt im Projekt die Aufgabe des End Users und arbeitet dabei an der Gesamtsystemarchitektur, der Planung der späteren Integration der IoT Services mit. Zudem hat die Wolfsburg AG eine beratende Rolle bei der Integration aus Anwendersicht inne.

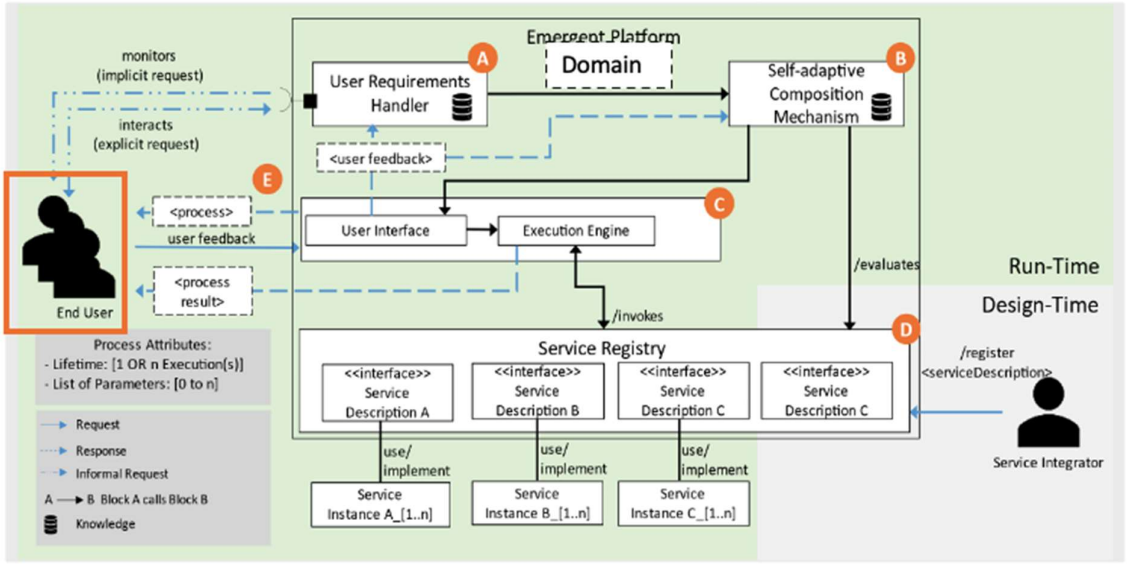


Abb. 3: Endnutzer

2.4.3. Technische Universität Clausthal

Die TU Clausthal hat in dem Projekt hauptsächlich den selbstadaptiven Kompositionsmechanismus der Plattform entwickelt. Zusätzlich dazu wurden zusammen mit der Universität Mannheim die Domain und die entsprechenden Schnittstellen vom User-Requirement Handler entwickelt. Außerdem war die TU Clausthal für die Entwicklung der Service Registry und der semantischen Integration von Service

Descriptions anhand der Domain-Konzepte verantwortlich. Die Schnittstelle zur Evaluation der Service Registry durch die Kompositionskomponente wurde ebenfalls von der TU Clausthal spezifiziert.

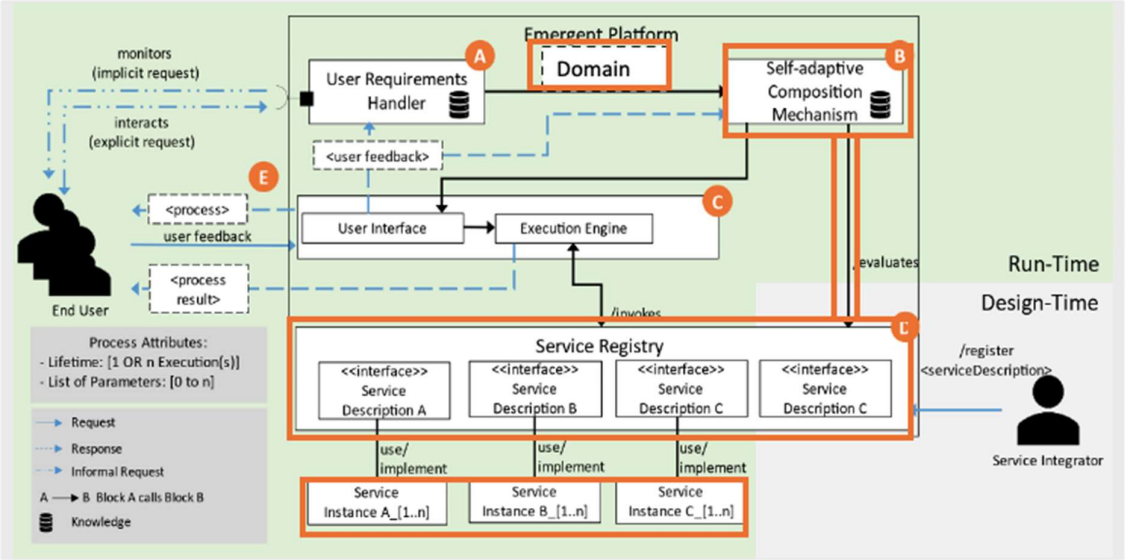


Abb. 4: Kompositionsmechanismus

2.4.4. Stone One

StoneOne war im Projekt für die Entwicklung der Execution Engine zur wirtschaftlichen Verwertung der Arbeitsergebnisse verantwortlich.

Damit Services auf der Plattform kompatibel zu weiteren Services zusammengebaut werden können, benötigt man eine einheitliche Beschreibungssprache und Ausführungsumgebung.

Die Execution Engine definiert eine Reihe von Basisoperationen, die zu einem ausführbaren Flow zusammengebaut werden können, welcher dann von der Engine interpretiert und ausgeführt werden kann.

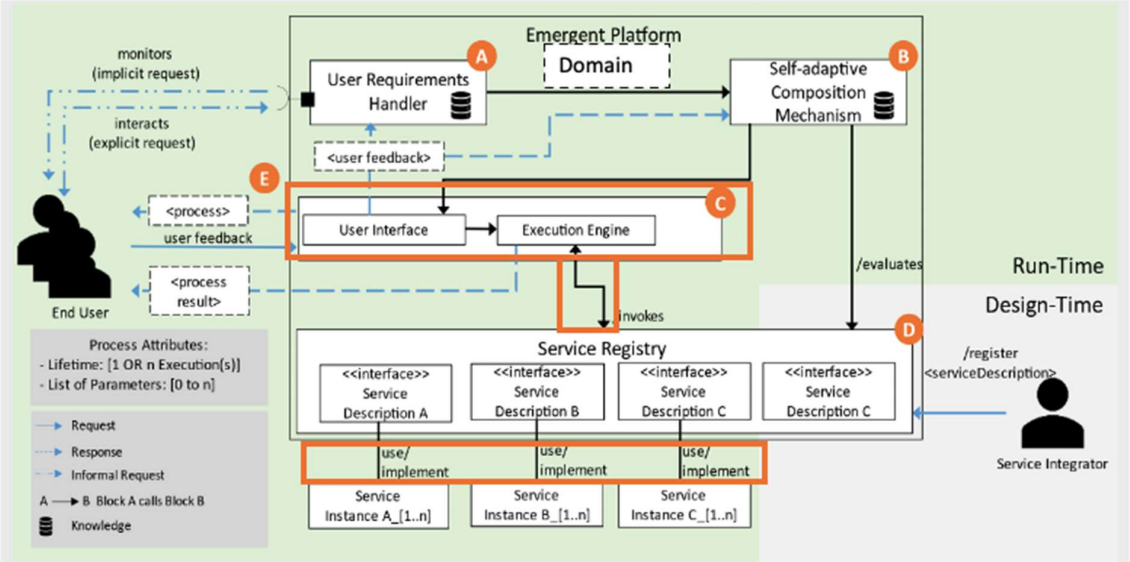


Abb. 5: Execution Engine

2.4.5. Darstellung der wichtigsten Positionen des zahlenmäßigen Nachweises

Zentrale Positionen für StoneOne / Anaqor

StoneOne war im Projekt für die Entwicklung der Plattform auf Basis der Smart Orchestra Technologie und die Integration der Partnermodule aus den Arbeitspaketen der universitären Partner verantwortlich.

Die Plattform bot damit die Schnittstellen und Services, um vom Nutzer kommende Anfragen unter Nutzung der universitären Komponenten automatisiert (emergent) zu höherwertigen User-kompatiblen Services zusammenzubauen. Dazu war eine einheitliche Beschreibungssprache und Ausführungsumgebung erforderlich.

Die Execution Engine der Plattform definiert eine Reihe von Basisoperationen, die zu einem ausführbaren Flow zusammengebaut werden können, welcher dann von der Engine interpretiert und ausgeführt werden kann.

Zudem wurde eine Komponente mit Design-Oberfläche zur Komposition der Basiselemente (IoT-Services) bezüglich der emergenten Services, sowie der technisch notwendigen Informationen und Abläufe zwischen den verschiedenen Services erstellt.

Die Integration der Partner-Komponenten war aufwändig, da die verschiedenen Module deutlich komplexer in der Entwicklung waren als in der Planung angenommen. Dies führte mehrfachen Schnittstellenänderungen und Anpassungen in der Plattform und erforderte die kostenneutrale Verlängerung der Projektlaufzeit.

Die Ausgaben wurden im Wesentlichen für die Personalkosten verwendet. Reisekosten wurden für Reisen zu projektinternen Konsortialtreffen verwendet.

Zentrale Positionen der Universität Mannheim

Bei der Universität Mannheim hat es zur Durchführung des Projekts Ausgaben in den Bereichen Personalkosten, Reisekosten und Gegenstände (Hardware) gegeben.
Die Ausgaben im Bereich der Personalkosten wurden verwendet um die im Verbundprojekt beschäftigten wissenschaftlichen Mitarbeitenden sowie die wissenschaftlichen Hilfsmitarbeitenden zu finanzieren. Die Mitarbeitenden der Universität Mannheim waren im Projekt für die Erforschung und Entwicklung der User Requirements Handler Komponente sowie einer Methode zur iterativen semantischen Integration von Software Services verantwortlich. Weiterhin, lag die wissenschaftliche Publikation der erreichten Ergebnisse im Aufgabenbereich der Mitarbeitenden der Universität Mannheim.
Die Ausgaben im Bereich der Reisekosten wurden verwendet um notwendige Reisen zu wissenschaftlichen Konferenzen im Rahmen der Publikation der erreichten Projektergebnisse sowie Reisen zu Projektinternen Konsortialtreffen zu finanzieren.
Die Ausgaben im Bereich Gegenstände wurde verwendet um die Hardware, welche für den Aufbau des Evaluationsszenarios an der Universität Mannheim benötigt wurde, zu beschaffen.

Zentrale Positionen der Wolfsburg AG

Die Wolfsburg AG hat aufgrund unternehmensinterner Veränderungen ihre Beteiligung am Projekt BIOTOPE reduziert. Beantragte Materialkosten wurden nicht in Anspruch genommen. Die Mittel wurden zur Verschiebung an die Technische Universität Clausthal freigegeben.

Zentrale Positionen der Technischen Universität Clausthal

Die Technische Universität Clausthal erarbeitete die Verfeinerung der Nutzungsszenarien und der Gesamtsystemarchitektur und die Erfassung der Anforderungen an die Schnittstellen. In Zusammenarbeit mit der Universität Mannheim war sie maßgeblich für die Inkrementelle semantische Modellintegration und Inferenzmechanismen zur emergenten Serviceaggregation und dem Entwurf einer Konzeptsprache für die semantischen Annotation von Web-Services verantwortlich. In enger Abstimmung mit den Konsortialpartner StoneOne/ Anaqor wurde die Integrationsplanung abgestimmt. Die prototypischen Systemkomponenten werden als Teil der BIoTope-Plattform eingesetzt. Die finanziellen Zuwendungen wurden für die Forschungsaufgaben und Publikationen im

Rahmen des Projekts verwendet. Reisekosten wurden verwendet um Reisen zu wissenschaftlichen Konferenzen und Reisen zu projektinternen Konsortialtreffen zu finanzieren.

3 Änderungen in der Zielsetzung

Zu Beginn des Projekts war eine Evaluation und Erprobung der im Projekt zu erforschenden emergenten Softwareplattform in einem Smart City orientierten Use Case im Geschäftsfeld der Wolfsburg AG und einem Smart Event Use Case in Kooperation mit einer Event- und Veranstaltungsagentur in Mannheim geplant. Als am 22. März 2020 der erste deutschlandweite Corona Lockdown begann, befand sich das Projekt BIoTope allerdings im ersten Projektjahr und das Konsortium war noch damit beschäftigt, einen grundsätzlichen Lösungsansatz für die Herausforderungen an eine emergente Softwareplattform zu erarbeiten. Daher wurde zu diesem Zeitpunkt noch nicht mit der Planung einer konkreten Umsetzung der Evaluationsszenarien begonnen, allerdings war man zunächst optimistisch, dass die Erprobung und Evaluation der zu erforschenden Plattform weiterhin wie geplant möglich ist.

Die Corona bedingten Einschränkungen des öffentlichen Lebens dauerten jedoch wesentlich länger an, als ursprünglich absehbar war. Dies hatte zum einen eine erhebliche Verzögerung in der Personalakquise, insbesondere der universitären Partner, zur Folge. Zum anderen waren durch weitreichende Corona bedingte Einschränkungen, welche insbesondere für Unternehmen im Veranstaltungs- und Entertainment Bereich eine Einstellung des gesamten Betriebs bedeuteten, der Aufbau und die Planung von realen Evaluationsszenarien und Testinfrastrukturen nahezu unmöglich. Durch diese Einschränkungen ist es vor allem bei den Arbeiten zur Gesamtsystemintegration und der Erprobung und Evaluation zu Verzögerungen gekommen. Als Reaktion musste das Projektkonsortium einige inhaltliche Änderungen vornehmen, welche von der ursprünglichen Projektplanung abweichen. Dies betrifft vor allem das Evaluationsszenario in Kooperation mit einer Event- und Veranstaltungsagentur in Mannheim. Als Alternative wurde ein Smart-Office Evaluations Szenarios direkt an der Universität Mannheim aufgebaut und erfolgreich evaluiert. Weiterhin haben die Verzögerungen beim Aufbau des Demonstrationsszenarios und die geltenden Einschränkungen für große Veranstaltungen eine sinnvolle Organisation und Ausrichtung eines ursprünglich im Kontext des Evaluationsszenarios in Mannheim geplanten Hackathons unmöglich gemacht. Dennoch ist es den Projektpartnern gelungen, diese – vor allem auch durch die zuwendungsneutrale Verlängerung der Projektlaufzeit um sieben Monate – wieder zu kompensieren. Das ursprünglich anvisierte Ziel der Erforschung, Erprobung und Evaluation einer emergenten Softwareplattform konnte vollumfänglich umgesetzt werden. Allerdings wurden, wie zuvor erläutert, aufgrund von weitgehenden Corona bedingten Einschränkungen einige inhaltliche Änderungen gegenüber den ursprünglich geplanten Evaluationsszenarien notwendig.

4 Untersuchte Methoden und Ergebnisse

In diesem Kapitel werden die Funktionen und Verantwortlichkeiten der einzelnen Komponenten in der vorgestellten Plattformarchitektur erläutert.

4.1 Domain

Innerhalb der Plattformarchitektur der in Abbildung 1 gezeigten Architektur wird das Konzept der “Domain” eingeführt. Diese Domain ist ein formales Modell der Domäne, in der der Nutzer der

Plattform agiert und in der sich die auf der Plattform vorhandenen Services befinden. Die Domäne hat hauptsächlich die Aufgabe, eine einheitliche formale Kommunikation zwischen den verschiedenen Komponenten der Plattform (User Requirements, Handler, Composition Mechanism und Execution Engine) zu ermöglichen. Dazu modelliert das Domänenmodell alle Konzepte, die benötigt werden, um Nutzeranforderungen und vorhandene Software-Services formal zu beschreiben. Konkret haben wir für den in BIoTope entwickelten Prototyp die Planning Domain Definition Language (PDDL) verwendet, um das Domänenmodell aufzubauen. PDDL ist ein Modellierungsstandard, der im Forschungsgebiet der automatischen Planung entwickelt wurde. Da sowohl die User Requirements Handler Komponente als auch die Composition Mechanism Komponente auf Techniken des automatischen Planens zurückgreifen, war diese Wahl sehr naheliegend.

4.2 User Requirements Handler

Die Aufgabe der User Requirements Handler (URH) Komponente ist es zur Laufzeit der emergenten Softwareplattform automatisiert die aktuellen Nutzeranforderungen eines Nutzers der Plattform zu erkennen. Dies kann entweder explizit oder implizit erfolgen. Bei der expliziten Erkennung, formuliert der Nutzer seine aktuellen Anforderungen aktiv, jedoch in einem unstrukturierten Format wie zum Beispiel natürlicher Sprache. Diese unstrukturierte Formulierung der Anforderungen kann üblicherweise nicht direkt verarbeitet werden, sondern muss vorher in ein formalisiertes, strukturiertes Format übertragen werden. Diese Aufgabe übernimmt im Fall der expliziten Erkennung der Nutzeranforderungen die URH Komponente. Die zweite Möglichkeit ist die implizite Erkennung der Nutzeranforderungen. In diesem Fall treffen wir die Annahme, dass in der Umgebung des Plattformnutzers eine gewissen Anzahl an Sensoren vorhanden ist, die in der Lage sind das momentane Verhalten des Nutzers aufzuzeichnen. Dies kann beispielsweise in einem digitalen Raum der Fall sein, wenn über verschiedene digitale Sensoren aufgezeichnet werden kann wo beispielsweise ein Computer Nutzer auf dem Bildschirm klickt. Ein anderes mögliches Szenario ist ein Smart-Office Szenario in dem durch verschiedene Sensoren (Thermometer, Bluetooth-Beacons, Bewegungsmelder, Drucksensoren, etc.) das momentane Verhalten des Nutzers aufgezeichnet werden kann. Diese Sensormessdaten werden dann von der URH Komponente ausgewertet, um die implizit darin enthaltenen Nutzeranforderungen zu extrahieren. Die Forschungsarbeiten im Rahmen von BIoTope haben sich vor allem auf die implizite Erkennung fokussiert und zu dieser Art der Erkennung wurden einige neuartige Methoden erforscht und evaluiert. Weiterhin wurde im Rahmen des Evaluationsszenarios im Mobility Bereich auch die explizite Anforderungserkennung erprobt und evaluiert.

Damit die URH Komponente in der Lage ist zur Laufzeit der Plattform automatisiert eine formalisierte Darstellung der Nutzeranforderungen aus einer expliziten oder impliziten Interaktion abzuleiten, ist es notwendig, dass die möglichen Modellierungsbestandteile dieser automatisch erstellten Formalisierung bereits zur Designzeit der Plattform durch einen Plattformadministrator modelliert werden. Im Projekt BIoTope nutzen wir hierfür die Formalisierungs- und Modellierungskonzepte aus der klassischen KI Planung. In diesem Bereich wird hauptsächlich PDDL verwendet um eine Planungsdomäne und ein darauf aufbauendes Planungsproblem zu modellieren. Wir haben uns dazu entschieden, PDDL auch für die Modellierung von sowohl den Nutzeranforderungen als auch der Kompositionsumgebung der SCM Komponente zu verwenden, da beide Probleme als Planungsproblem (Komposition) bzw. Teil eines Planungsproblems (automatische Zielerkennung) betrachtet werden können. Im Fall der automatischen Nutzeranforderungserkennung nutzen wir PDDL, um die Nutzeranforderungen als sogenannte Ziele, welche als logische Konjunktion mehrerer modellierter Eigenschaften der Umgebung definiert sind, zu formulieren. Das hat sowohl den Vorteil, dass wir diese formalisierte Zielrepräsentation der Nutzeranforderungen anschließend in exakt dieser

Form an die SCM Komponente weiterleiten können, welche auf der Basis dieser formalisierten Repräsentation automatisch eine Software-Service-Komposition berechnet, als auch den Vorteil, dass wir auf bestehende Ansätze zur Zielerkennung aus der wissenschaftlichen Literatur aufbauen können. Diese Ansätze sind vor allem im Bereich der impliziten Erkennung relevant.

Grundsätzlich kann das Problem der automatischen Nutzeranforderungserkennung als ein Klassifizierungsproblem gesehen werden. Dabei ist die Menge aller distinkten Konjugationen die aus den in der verwendeten Planungsdomäne modellierten Eigenschaften gebildet werden können gleich der Menge der Klassen. Die Eingabedaten unterscheiden sich je nach Art der Erkennung (explizit vs. implizit). Im expliziten Fall bilden die explizit durch den Nutzer formulierten Anforderungen die Eingabedaten. Im impliziten Fall entsprechen die Eingabedaten der Sequenz der bisher über ein gewisses Zeitintervall aufgezeichneten Sensordaten. Um dieses Klassifizierungsproblem zu lösen können grundsätzlich drei verschiedene Arten von Ansätzen verwendet werden: Modell-basierte Ansätze, Modell-freie Ansätze und hybride Ansätze. Unter Modell-basierter Erkennung von Nutzeranforderungen verstehen wir in BIoTope, dass für die Erkennung ein Ansatz verwendet wird der ein formales Modell der Umgebung (in der BIoTope Architektur repräsentiert durch die Domain) benötigt. Ein Vorteil dieser Ansätze ist, dass sie neben dem formalen Modell der Umgebung keine Trainingsdaten benötigen, um eingesetzt zu werden. Daher eignen sie sich besonders für Szenarien in denen wenige Trainingsdaten vorhanden sind, und mit sinnvollem Aufwand ein formales Modell der Umgebung erstellt werden kann. Im Projekt BIoTope haben wir uns in dieser Kategorie vor allem mit Ansätzen beschäftigt, die in den Forschungsbereich „Plan Recognition as Planning“ fallen. Eine zentrale Annahme dieser Art von Ansätzen ist außerdem, dass ein Teil des formalen Umgebungsmodells auch eine Menge von potenziell möglichen Nutzeranforderungen ist. Unter Modell-freier Erkennung von Nutzeranforderungen verstehen wir in BIoTope, dass für die Erkennung ein Ansatz verwendet wird, der kein formales Modell der Umgebung benötigt. Stattdessen, werden Ansätze die unter diese Kategorie fallen auf der Basis von historischen Trainingsdaten darauf trainiert, auf der Basis von aktuell aufgenommenen Sensordaten die aktuellen Nutzeranforderungen zu erkennen. Daher sind diese Art von Ansätzen besonders gut für Szenarien geeignet in denen es nicht möglich oder sinnvoll ist manuell ein formales Modell der Laufzeitumgebung zu erstellen, gleichzeitig aber eine große Menge von Trainingsdaten verfügbar ist. In diesem Fall lernen solche Ansätze quasi eigenständig ein implizites Modell der Laufzeitumgebung aus den verwendeten Trainingsdaten. Ansätze die unter die dritte Kategorie, hybride Erkennung von Nutzeranforderungen, fallen zeichnen sich dadurch aus, dass diese Ansätze sowohl Methoden aus dem Bereich der Modell-basierten als auch aus dem Bereich der Modell-freien Erkennung verwenden. Hierdurch benötigen diese Ansätze in vielen Fällen auch sowohl ein formales Modell der Laufzeitumgebung als auch Trainingsdaten. Allerdings zeichnen sich hybride Ansätze dabei dadurch aus, dass auf der einen Seite das formale Modell nicht so umfangreich sein muss, da die potenziell fehlenden Teile aus den Trainingsdaten hinzugefügt werden können. Auf der anderen Seite werden auch nicht so viele Trainingsdaten benötigt, da das Modell der Laufzeitumgebung nicht aus dem Nichts neu gelernt werden muss, sondern das vorhandene formale Modell als Startpunkt fungieren kann.

Im Rahmen von BIoTope wurden vor allem Modell-freie und hybride Ansätze erforscht und evaluiert. Da hybride Ansätze allerdings auf Modell-basierten Ansätzen aufbauen, wurde diese Art von möglichen Methoden auch indirekt evaluiert. Weiterhin wurde im Rahmen des Smart-Office Use Cases auch ein Modell-basierter Ansatz in der Praxis erprobt. Im Folgenden werden die einzelnen Ansätze die im Rahmen von BIoTope erforscht und evaluiert wurden kurz erläutert. Bei Interesse an weiteren Details verweisen wir die Leser auf die in den jeweiligen Abschnitten genannten wissenschaftlichen Publikationen.

Hybride Zielerkennung auf der Basis von Planungsalgorithmen und Bayeschen Netzen. Im Rahmen von BIoTope wurde ein hybrider Ansatz zur automatischen Nutzerzielerkennung erforscht und prototypisch implementiert. Wir betrachten diesen Ansatz als „hybrid“, da dieser einen bestehenden Ansatz zur Zielerkennung, der auf dem Einsatz von automatischen Planungssystemen und manuell modelliertem Domänenwissen basiert, mit einem daten-basierten Ansatz, welcher auf der Basis von historischen Trainingsdaten lernt um die Wahrscheinlichkeiten möglicher Nutzerziele zu bestimmen, kombiniert. Die Intuition hinter der Kombination eines Ansatzes der auf manuell modelliertem Domänenwissen basiert und eines Ansatzes der Domänenwissen automatisch auf der Basis von historischen Daten lernt, ist, dass solch ein hybrider Ansatz die Stärken dieser beiden grundlegenden Techniken nutzen kann und somit in der Lage sein sollte, bessere Ergebnisse zu erzielen. Um dies zu evaluieren, wurde der hybride Ansatz in ersten Experimenten auf der Basis eines in der wissenschaftlichen Community bekannten und öffentlich verfügbaren Datensatzes, welcher beobachtete Nutzerdaten aus einem Smart-Home Szenario enthält, evaluiert. Die Ergebnisse dieser Evaluation haben gezeigt, dass alle drei Ansätze gute Ergebnisse für die automatisierte Erkennung von Nutzerzielen erreichen. Zusätzlich zeigen die Ergebnisse, dass der hybride Ansatz tatsächlich bessere Ergebnisse erzielt als die beiden einzelnen Ansätze.

Die Ergebnisse wurden in zwei wissenschaftlichen Publikationen mit den Titeln „Combining Symbolic and Statistical Methods for Goal Recognition in Smart Home Environments“ und „Combining Symbolic and Data-Driven Methods for Goal Recognition“ im Rahmen der „19th IEEE International Conference on Pervasive Computing and Communications“ (PerCom 2021) veröffentlicht.

Hybrider Ansatz auf der Basis von Planungslandmarken und Bayeschen Netzen. Der zuvor beschriebene Ansatz wurde außerdem in verschiedene Richtungen weiterentwickelt. Zum einen wurde der Einsatz einer kürzlich publizierten Weiterentwicklung des Planungssystem basierten Ansatzes, welcher auf sogenannten Planungslandmarken basiert, untersucht. Experimente mit diesem Ansatz haben gezeigt, dass der Einsatz dieser Weiterentwicklung vor allem Vorteile bezüglich der Rechenzeit, die benötigt wird, um ein Zielerkennungsproblem zu lösen, bietet. Die Ergebnisse zu diesem Ansatz wurden in Form einer wissenschaftlichen Publikation mit dem Titel „Leveraging planning landmarks for hybrid online goal recognition“ auf dem Workshop on Scheduling and Planning Applications im Rahmen der International Conference on Automated Planning and Scheduling 2022 präsentiert.

Anforderungserkennung auf der Basis Vektorgleichheit.

Weiterhin wurde in BIoTope ein Ansatz zur automatischen Nutzeranforderungserkennung entwickelt, der auf dem Vergleich von Vektorgleichheit basiert. Grundsätzlich ist dieser Ansatz verwandt mit dem zuvor schon erwähnten Planungslandmarken basierten Ansatz, allerdings kann der Vektorbasierte Ansatz mehr Fakten, die über die Umgebung des Nutzers bekannt sind, verwenden. Erste Experimente haben gezeigt, dass der Vektorbasierte Ansatz eine wesentlich höhere Erkennungsgenauigkeit aufweist, als der Planungslandmarken basierte Ansatz. Die Veröffentlichung dieser Ergebnisse in Form einer wissenschaftlichen Publikation sind momentan zur Begutachtung eingereicht.

Anforderungserkennung auf der Basis eines Naive Bayes Klassifizierers.

Im Rahmen des Smart-Office Use Cases wurde im Projekt BIoTope außerdem der Einsatz eines Naive Bayes Klassifizierers zur impliziten Erkennung von Nutzeranforderungen evaluiert. Grundsätzlich trifft dieser Ansatz relativ starke Annahmen, die in der Realität fast nie vollständig gelten. Allerdings wurde in der Vergangenheit bereits gezeigt, dass Naive Bayes Klassifizierer trotz dieses Umstands sehr gute Genauigkeit erreichen können. Die Experimente im Rahmen von BIoTope haben dies bestätigt und

zeigen, dass ein Naive Bayes Klassifizierer auf der Basis der im Kontext des Smart-Office Use Cases gesammelten Daten gute bis sehr gute Ergebnisse erzielt.

4.3 Self-adaptive Composition Mechanism

Der Self-adaptive Composition Mechanism spielt eine zentrale Rolle in der vom Projektkonsortium erarbeiteten Softwareplattform. Die Hauptaufgabe dieser Komponente ist die Komposition von im IoT-Ökosystem bereitgestellten Services, um ein gegebenes höherwertiges Ziel zu erreichen. Dazu gehört die Ermittlung, welche Services benötigt werden und wie diese Services nacheinander aufgerufen werden können, um das Ziel zu erreichen.

Das Berechnen einer geeigneten Komposition wird als Planungsproblem formuliert, bei dem das Requirement des Users das Ziel ist und die Service Descriptions des IoT-Ökosystems mögliche Aktionen sind, die unternommen werden können. Die Service Descriptions sind hierfür semantisch durch Vorbedingungen (Preconditions) und Effekte aus der Domaine beschrieben.

Es gibt bestehende Ansätze und Tools, mit denen entsprechende Domänen formal beschrieben werden können und die eine Berechnung von Plänen ermöglichen. Auf ein bestehendes Planungstool wurde bei der Implementierung der Kompositionskomponente aufgebaut.

In der Gesamtarchitektur hat der Self-Adaptive Composition Mechanism die folgenden Schnittstellen zu anderen Komponenten: Sie hat eine Schnittstelle vom User Requirements Handler, von dem das Ziel eines End Users bereitgestellt wird. Von der Service Registry kann die Komponente alle aktuell verfügbaren Service Descriptions abfragen und auf dieser Grundlage die Komposition errechnen. Eine weitere Schnittstelle ist für die Übergabe der errechneten Komposition zur Execution Engine zuständig.

4.4 Execution Engine

- **Ziel und Zweck der Execution Engine:**

Damit Services auf der Plattform kompatibel zu weiteren Services zusammengebaut werden können, benötigt man eine einheitliche Beschreibungssprache und Ausführungsumgebung. Die Execution Engine definiert eine Reihe von Basisoperationen, die zu einem ausführbaren Flow zusammengebaut werden können, welcher dann von der Engine interpretiert und ausgeführt werden kann. Ein solcher Flow kann automatisch auf Basis eines Composition-Results erstellt werden, oder auch manuell über ein Service-Composer-UI. Mittels eines integrierten Editors kann ein solcher Flow erstellt bzw. visualisiert werden. Die (derzeit noch manuelle) Erstellung eines Flows ist Voraussetzung für die Integration externer Service-Instanzen auf der Plattform. Jeder Flow kann auch wieder als Baustein in anderen Flows verwendet werden. Damit ist die Basis geschaffen, um aus einem Compostion-Result automatisch höherwertige und direkt ausführbare Flows zu erzeugen.

- **Basisoperationen der Plattform:**

Jeder Flow besteht aus einer Reihe von Basisoperationen, die von der Execution Engine vorgegeben werden. Diese sind:

Start	Der Einstiegspunkt. Jeder Service hat genau einen Startknoten, an dem die Ausführung beginnt. Am Startknoten werden Parameter definiert, die bei der späteren Ausführung übergeben werden müssen.

Ende	Hier endet die Ausführung. Es kann aus Gründen der Übersichtlichkeit mehrere Ende-Knoten geben. Am Ende-Knoten werden die Rückgabewerte definiert, die bei einer Verwendung des Services in einem weiteren Service ausgewertet werden können.
Rule	Eine Regelauswertung entsprechend der If-Then-ElseIf-Else Struktur. Eine Regel kann beliebig viele ElseIf Blöcke enthalten. Mit diesem Operator können Verzweigungen im Ablauf realisiert werden, basierend auf Auswertungen der bisherigen Datenwerte.
Rest	Mit diesem Knotentyp kann eine externe Funktion/Service über einen HTTP-REST Aufruf angebunden werden.
ExtractFromJson	Liegt ein Eingabedatum vom Typ String vor, bei dem es sich eigentlich um ein serialisiertes JSON-Objekt handelt, so kann mithilfe dieses Knotens auf die Attribute im JSON-Objekt zugegriffen werden. Dies kommt vor allem dann vor, wenn Rückgabewerte von externen Funktionen, die zuvor per REST aufgerufen wurden, evaluiert werden müssen.
ComposeToJson	Der Gegenpart zu ExtractFromJson. Hier kann ein JSON-Objekt aus mehreren Datenwerten erzeugt und dann in einen String serialisiert werden.
StringBuilder	Dieser Knotentyp erlaubt dem Anwender, beliebige Strings aus einzelnen Datenwerten zusammenzustellen.
Debug	Nur für Entwicklungszwecke einsetzbar. Es werden die aktuellen Zustände des Executionflows (bezogen auf Ausführung und Daten) in das Logfile der Plattform geschrieben.
Service	Hiermit wird ein weiterer Service, der bereits als Service-Flow vorliegt, in die Ausführung integriert. Die Datenkanäle für die Eingabe und Ausgabedaten sind hierbei abhängig von dem zu integrierenden Service.
Confirmation	Mit diesem Knotentyp kann eine Rückfrage an den Anwender getriggert werden. Dies ist dann sinnvoll, wenn z.B. eine Buchung bestätigt werden soll, oder um eine Entscheidung zwischen mehreren Alternativen abzufragen.

Weitere Basisoperationen sind in Planung.

- **Beispiel eines ausführbaren Flows eines integrierten externen Service:**

Die Integration eines externen Service besteht im Wesentlichen aus dessen Aufruf via HTTP. Hierzu wird ein REST-Knoten verwendet, der nun passend konfiguriert werden muss. Eingangsparameter können als Pfad-/Request-/oder Content-Parameter übergeben werden, müssen dazu jedoch ggf. noch konvertiert werden. Handelt es sich bei dem Ergebnis des Aufrufs um einen strukturierten Datentyp, so muss auch dieser aufbereitet werden, damit die einzelnen Datenwerte nach Beendigung des Flows weiterverwendet werden können.

Das folgende Beispiel zeigt einen externen Service aus dem Demonstrationsszenario des Projektes bei dem mittels eines POST-Aufrufes eine Dienstleitung (Autowäsche) gebucht werden kann.

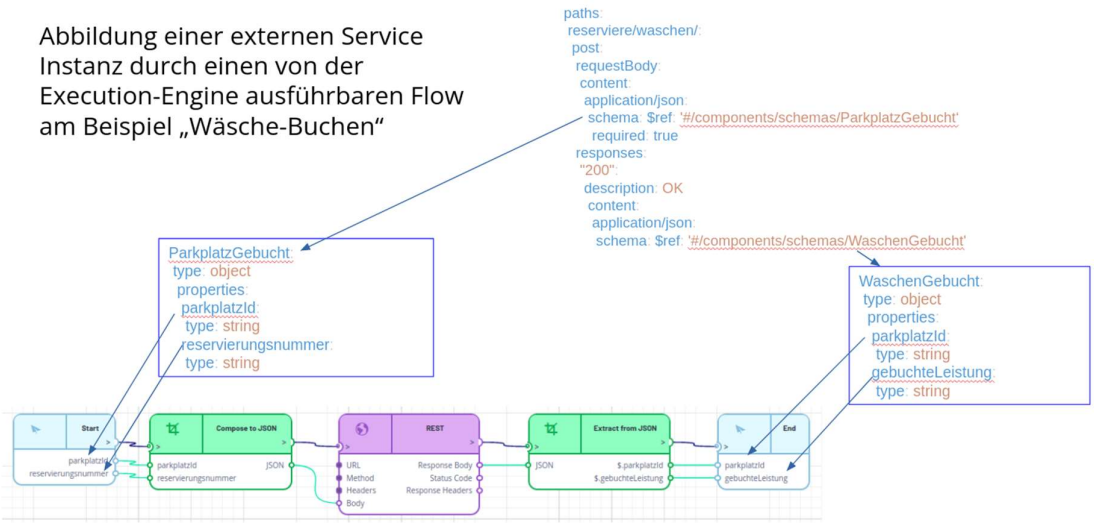


Abb. 6: Service Instanz „Wäsche Buchen“

- **Beispiel eines dynamisch erzeugten Flows aus einem Composition-Result:**

  Ein Composition-Result besteht aus einer Reihe von Action-Aufrufen, sowie einem Environment, welches unter anderem die Typdefinitionen enthält.

  Da zu jeder Action mindestens ein passender Service-Flow in der Service-Registry der Execution-Engine enthalten ist, kann ein passender Meta-Flow (bestehend aus Aufrufen weiterer Flows) automatisch erzeugt werden, indem die Service-Flows entsprechend der Reihenfolge der Actions im Meta-Flow nacheinander aufgerufen und die benötigten Eingangsdaten jedes Service-Flows mit den Werten der Ausgangsdaten der zuvor ausgeführten Service-Flows belegt werden.

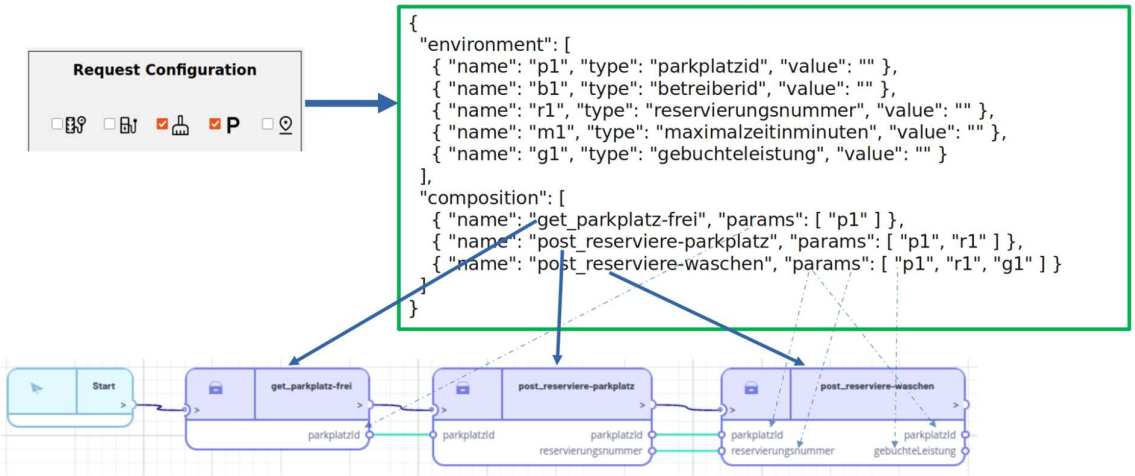


Abb. 7: Dynamischer Flow

- **Schematische Darstellung der Ausführung des dynamisch erzeugten Flows:**

Bei der Ausführung eines Meta-Flows werden die darin über Service-Nodes enthaltenen Sub-Flows wie eigenständige Services abgearbeitet. Genau genommen geschieht dies rekursiv, da auch Sub-Flows wiederum über Service-Nodes integrierte Sub-Flows enthalten können. Jeder Meta-Flow, egal ob dynamisch oder manuell erzeugt, kann somit wieder Teil eines neuen Meta-Flows werden. Für eine Verwendung in einem dynamisch erzeugten Flow benötigt es dann lediglich einer Action-Referenz auf eine passende Servicebeschreibung in der Service-Registry der Composition Komponente.

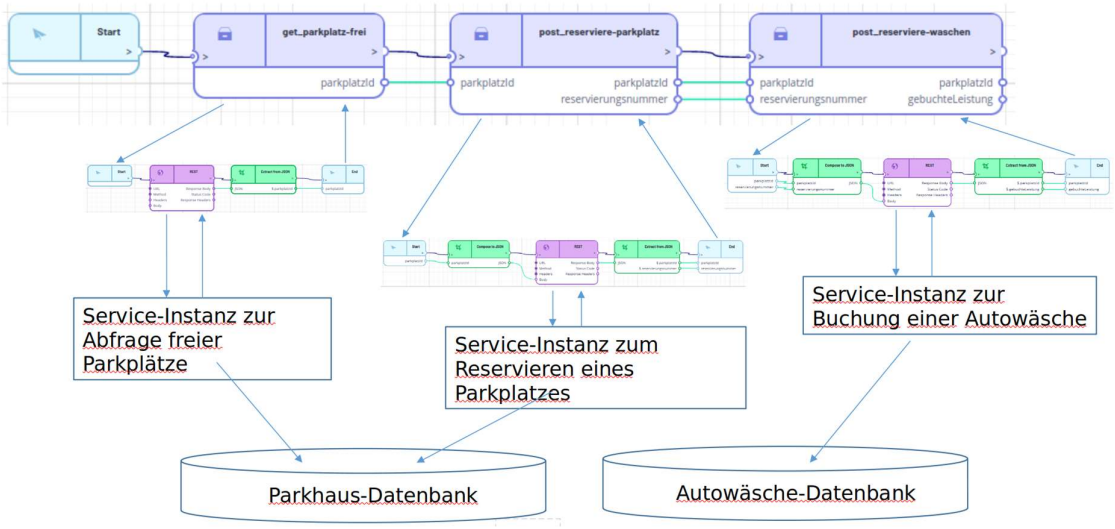


Abb. 8: Schematische Ausführung eines Flows

4.5 Service Registry

Die Service Registry ist die Komponente der Architektur, welche die anzubietenden Service Descriptions der Plattform beinhaltet. Hierbei geht es primär um ein Set von eindeutigen Service Descriptions, die mit verschiedenen Serviceinstanzen verknüpft werden können. Die Service Registry verwaltet den Standort sowie den Interaktionsstil mit den Serviceinstanzen. Dies wird durch die use/implement Beziehung zwischen der Service Registry und den Serviceinstanzen, wie im Architekturbild zu sehen ist, realisiert.

Die formale Sprache *Planning Domain Definition Language (PDDL)* wird hierbei für die Beschreibung von Services benutzt. Eine Service Description ist eine eindeutige Action in der Service Registry. Eine Action beschreibt ein Service fachlich. Darüber hinaus bestehen Actions aus Vorbedingungen und Effekten, wodurch ihre Nutzung ermöglicht wird. Des Weiteren haben Actions Parameter, welche für ihre Vorbedingung und Effekte gelten müssen. Die folgende Abbildung zeigt ein Beispiel für eine Action in der Service Registry. Der Name der Action ist hier post_reserviere-wash. ?p, r? und ?l sind mit den jeweiligen Typen parkplatzid, reservierungsnummer und gebuchteleistung die Parameter der Action. Die Konzepte (reserved_parking ?p ?r) und (reserved_washing ?p ?l) sind entsprechend precondition und effect.

Die Service Registry verfügt über zwei deploybare Komponenten (Back-End, Front-End). Die Front-End Komponente verwaltet die Verfügbarkeit der Service Descriptions und die Back-End Komponente die Gesamtlogik der Service Registry. Darüber hinaus kann eine Service Description aktiviert oder deaktiviert werden. Deaktivierte Service Descriptions werden nicht in die Abrechnungen der Service Registry miteinbezogen. Die folgende Abbildung zeigt links den Status der Service Registry. Die Schaltflächen stellen die Verfügbarkeit dar: Wenn die Farbe grün ist, ist die Action verfügbar. Ist die

Farbe grau, ist die Action nicht verfügbar. Das Beispiel in der unteren Grafik zeigt, dass die Action get_parkplatz-navigation-parkplatzid deaktiviert ist. Damit steht diese der Service Registry zur Anwendung nicht zur Verfügung.

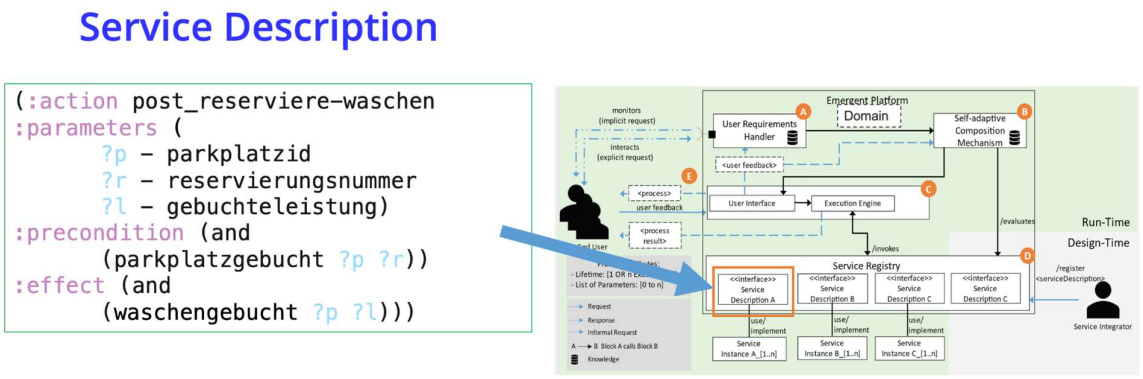


Abb. 9: Service Description

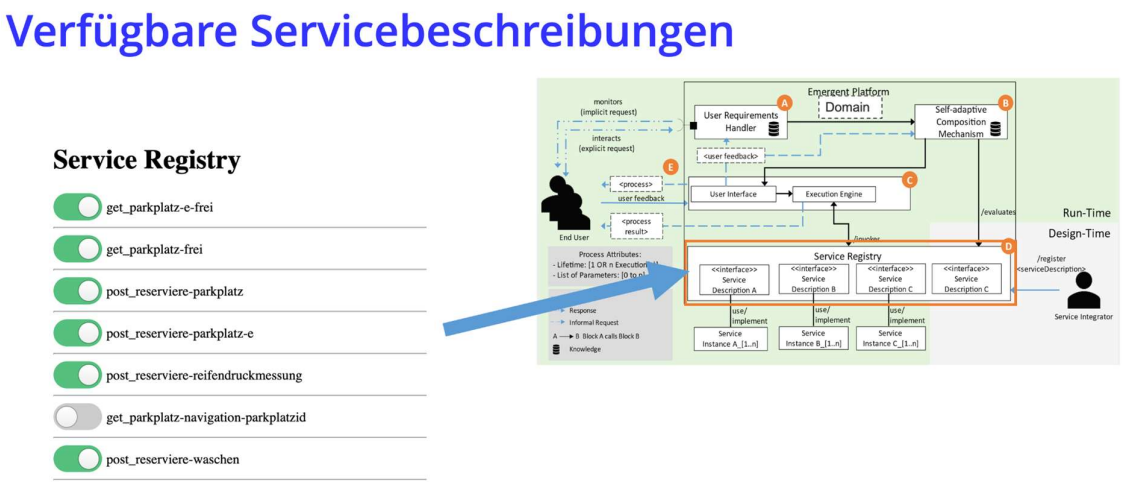
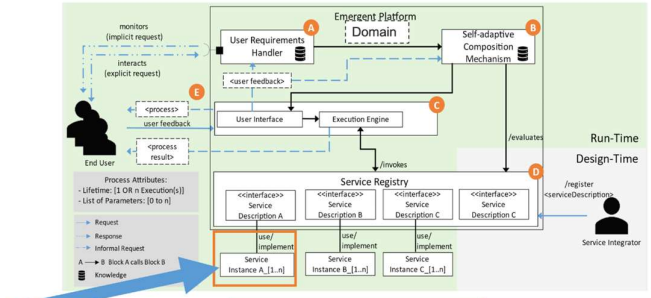


Abb. 10: Service Registry

Durch seine verschiedenen Schnittstellen lässt sich die Service Registry von anderen Komponenten abfragen oder steuern. Mit der Realisierung der /evaluates Schnittstelle aus dem Architekturbild stellt die Service Registry ihr Wissen der Komponente B zur Verfügung. Somit können bspw. verfügbare Service Descriptions abgefragt werden. Die Schnittstelle zur Execution Engine gewährleistet die sichere Erreichbarkeit der Serviceinstanzen. Sollte eine neue Service Description in der Service Registry integriert werden, wird dies durch Ihre Schnittstelle zum Service Integrator ermöglicht. Eine Beispiel Serviceinstanz und ihre formale Beschreibung sind in der folgenden Abbildung dargestellt.

Service Instance

- Formale Beschreibung eines REST-Endpunktes mit genauem Path, Method und (optional) genauen RequestBodies, ResponseBodies




```
paths:
 reserviere/waschen/:
  post:
   requestBody:
    content:
     application/json:
      schema: $ref: '#/components/schemas/ParkplatzGebucht'
       required: true
   responses:
    "200":
     description: OK
      content:
       application/json:
        schema: $ref: '#/components/schemas/WaschenGebucht'
```

```
ParkplatzGebucht:
 type: object
  properties:
   parkplatzId:
    type: string
   reservierungsnummer:
    type: string
```

```
WaschenGebucht:
 type: object
  properties:
   parkplatzId:
    type: string
   gebuchteLeistung:
    type: string
```

Abb. 11: Service Instance

Semantische Service Integration

Die semantische Integration von neuen Service Instanzen in eine bestehende Serviceplattform stellt eine signifikante Herausforderung dar. Dies liegt im Wesentlichen daran, dass sich die meisten Serviceanbieter beim Designen ihrer Service Schnittstellen nicht an Branchenstandards halten bzw. dass solche Branchenstandards, je nach Servicedomäne, bisher gar nicht oder nur teilweise definiert wurden. Aus diesem Grund muss jede Service Plattform intern ein eigenes semantisches Modell pflegen in das alle neuen Service Instanzen, die von der Plattform genutzt werden sollen, integriert werden müssen. Im Fall der in BIoTope entwickelten Plattform verwaltet die Service Registry das semantische Modell aller auf der Plattform nutzbaren Services. Das bedeutet, dass alle Service Instanzen die eine der in der Service Registry verfügbaren Service Descriptions implementiert, mit dieser Service Description semantisch integriert werden muss. Eine typische Methode, um eine solche semantische Integration durchzuführen ist eine manuelle Erstellung eines sogenannten Software Adapters. Dies erfordert allerdings viel manuellen Aufwand durch einen Service Integrator. Um dieses Problem zu lösen, wurde in BIoTope eine neuartige Methode zur teilweise automatisierten, iterativen semantischen Service Integration erforscht und evaluiert. Die grundlegende Idee hinter dieser Methode ist es, bereits vorhandenes Integrationswissen bei neuen Integrationsfällen wiederzuverwenden und somit die semantische Integration von neuen Service Instanzen zu beschleunigen. Im Optimalfall ist es, sobald genug Integrationswissen auf der Plattform vorhanden ist, möglich, eine neue Service Instanz vollautomatisch in das bestehende semantische Servicemodell zu integrieren. Die Forschungsergebnisse zu dieser Integrationsmethode wurden in mehreren wissenschaftlichen Publikationen mit den Titeln “Automated configuration in adaptive iot software ecosystems to reduce manual device integration effort: Application and evaluation of a novel engineering method”, “Gabble: Managing Integration Knowledge in IoT-Systems with Logical Reasoning”, “Knowledge-Driven Architecture Composition: Assisting the System Integrator to Reuse Integration Knowledge” und “Applying Knowledge-Driven Architecture Composition with Gabble” auf verschiedenen wissenschaftlichen Konferenzen veröffentlicht. Bei Interesse an weiterführenden

Details zu dieser Methode, verweisen wir die Lesenden auf die genannten wissenschaftlichen Publikationen.

5 Evaluation und Demonstration

5.1 Demonstrationsszenario Mobility

Im Projekt wurde die entwickelte IoT-Plattform in mehreren Bereichen angewendet. In diesem Abschnitt des Berichtes wird der Use Case „Mobilität“ näher erläutert. Da dem Projektkonsortium durch die Wolfsburg AG ein Industriepartner mit bestehender Infrastruktur in Form eines mit Sensorik und Services (zur Abfrage von Parkplatzbelegungen) ausgestatteten Parkhauses angehört, wurde in dem Use Case ein smartes Parkhaus betrachtet. Im Rahmen dieses Demonstrationsszenarios wurden Services beschrieben, die ein Nutzer eines Parkhauses einzeln oder in Kombination in Anspruch nehmen kann.

Services im Szenario:

- get_parkplatz-e-frei: Kann genutzt werden, um einen freien Parkplatz mit Ladesäule für ein E-Auto zu erfragen. Der Service ermittelt einen freien Parkplatz. (Der Parkplatz wird dadurch nicht reserviert)
- get_parkplatz-e-frei: Kann genutzt werden, um einen freien Parkplatz zu erfragen. Der Service ermittelt einen freien Parkplatz. (Der Parkplatz wird dadurch nicht reserviert)
- post_reserviere-parkplatz: Der Service kann genutzt werden, um einen Parkplatz zu reservieren.
- post_reserviere-parkplatz-e: Der Service kann genutzt werden, um einen Parkplatz mit Ladesäule zu reservieren.
- post-reserviere-reifendruckmessung: Der Service kann an einem beliebigen Parkplatz als Zusatzleistung gebucht werden. Wenn eine Buchung erfolgt, ist denkbar, dass ein Mitarbeiter des Serviceanbieters die Prüfung und Anpassung des Reifendrucks für das auf dem entsprechenden Parkplatz abgestelltes Fahrzeug durchführt.
- post_reserviere-waschen: Analog der angebotenen Reifendruckmessung ist eine Fahrzeugwäsche Teil des Szenarios. Eine Einschränkung, die für die Wäsche gilt, ist, dass die Fahrzeugwäsche nur an Parkplätzen ohne Ladesäule angeboten werden kann.
- get_parkplatz-navigation-parkplatzid: Dieser Service ermöglicht eine Navigation eines Nutzers zu einem bestimmten Parkplatz. Der Service ermöglicht über Aktoren (Pfeile) eine Anzeige zu dem entsprechenden Parkplatz. Dieser Service ermöglicht eine Anzeige, die unabhängig von Nutzer-Endgeräten (Mobiltelefonen) möglich ist.

Die beschriebenen Services wurden in Bezug auf ihre Schnittstelle beschrieben. Hierzu wurde der OpenAPI-Specification (https://swagger.io/docs/specification/about/) Standard genutzt.

In der prototypischen Implementierung des Szenarios wurden zu den Spezifikationen (Service Descriptions) konforme Implementierungen entwickelt. Der Use Case wurde trotz teilweise vorhandener und angeschaffter Sensorik aufgrund von für das Projekt externen Einflüssen (Corona, Lieferengpässe von Hardware und zum Ende der Projektlaufzeit hin durchgeführten Baumaßnahmen und winterlich bedingter Witterung) software-basiert auf der entstandenen IoT-Plattform realisiert.

Durch die dadurch notwendige Abweichung vom geplanten Einsatz im Wolfsburger Parkhaus musste die Evaluierung und Erprobung auch auf einer softwarebasierten Parkhaus-Oberfläche erfolgen.

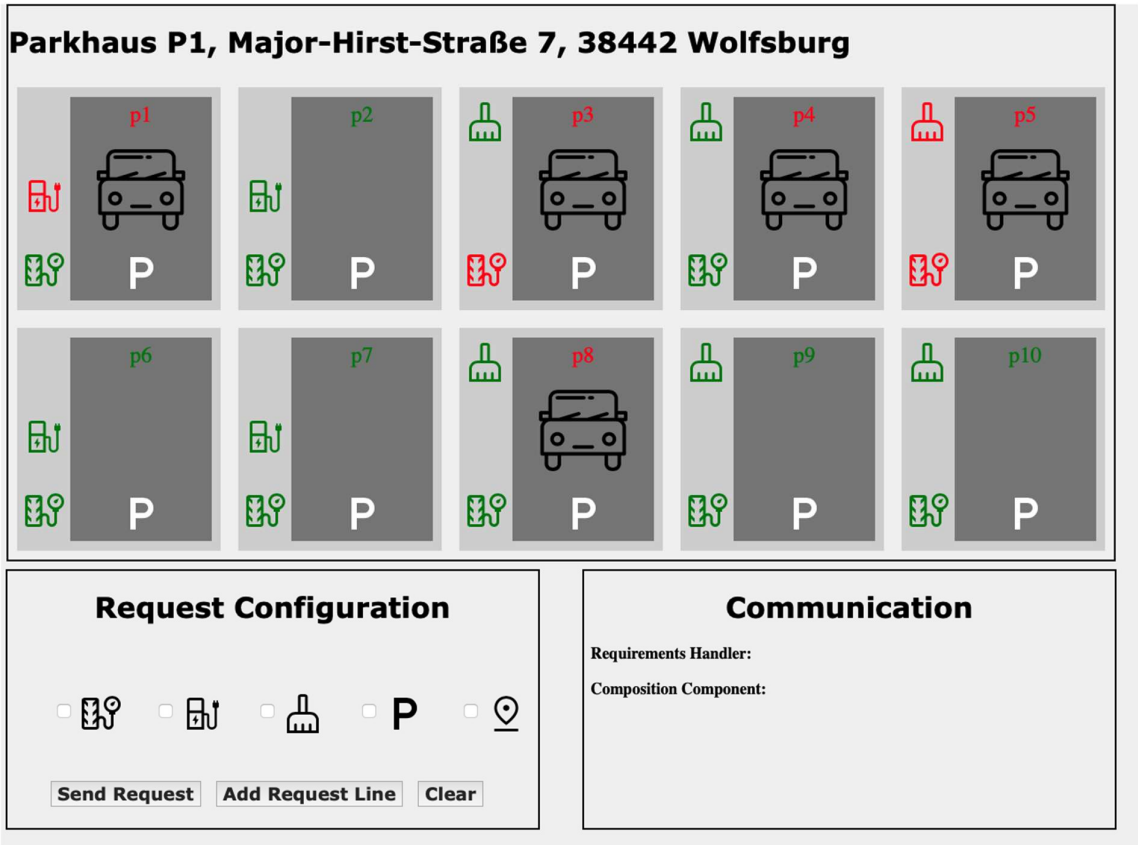


Abb. 12: Demonstrationsszenario Parkhaus

Das Demonstrationsszenario Mobilität ist ein Parkhaus, das umfassende Service-Lösungen bietet und als graphische Oberfläche in Abbildung 12 dargestellt ist. Es sind 10 Parkplätze erkennbar, die entweder belegt oder frei sind. Dass ein Parkplatz belegt ist, ist an der roten Bezeichnung (siehe z.B. p3) und dem Symbolbild eines Autos zu erkennen. An den Parkplätzen werden weitere Leistungen angeboten. Dies umfasst die Möglichkeit, ein E-Auto zu laden (an den vier Parkplätzen im linken Bereich), an den anderen Parkplätzen gibt es die Möglichkeit, eine Autowäsche durchführen zu lassen ("Besen"-Symbol). An allen Parkplätzen gibt es die Möglichkeit, den Reifendruck eines Autos prüfen (und anpassen) zu lassen.

Das Parkhaus bietet zusätzlich die Möglichkeit, eine Navigation zu einem bestimmten Parkplatz auf der Fahrbahn zwischen den Parkplätzen anzuzeigen. Dies ist durch die Pfeile in der Abbildung 13 symbolisiert.

Die Abbildung 12 zeigt unten links den Requestkonfigurator zu den implementierten Use-Cases an. Damit kann ein Nutzer eine explizite Anfrage an die Plattform stellen. Jede Checkbox steht für eine Anforderung, die der Nutzer an das Parkhaus stellen kann. Diese sind nicht eins zu eins auf die auf der Plattform verfügbaren Servicebeschreibungen abgestimmt. Es gibt Servicebeschreibungen, die ein Nutzer nicht stellen kann, die aber in der Komposition später genutzt werden (beispielsweise Bestimmung eines freien Parkplatzes).

Alle Checkboxen einer Zeile des Konfigurators beziehen sich auf einen Parkplatz. In der Abbildung ist eine Anfrage nach einem Parkplatz mit Lademöglichkeit und Luftdruckmessung gewünscht. Zusätzlich soll auch die Navigation zu dem entsprechenden Parkplatz angezeigt werden.

Wenn eine Anfrage mehrere Parkplätze umfassen soll, können mehrere Zeilen des Konfigurators gleichzeitig befüllt werden. Entsprechende Teilanfragen je Zeile beziehen sich dann auf die einzelnen Parkplätze.

Die Oberfläche des Prototyps zeigt zusätzlich auch die (Zwischen-)Ergebnisse der einzelnen Komponenten an, damit die im Requestkonfigurator zusammengestellte Anfrage ersichtlich ist. Listing 1 zeigt die zu der Anfrage in Abbildung 13 passende formale Anfrage.


```
{ "environment": [
        { "value": "", "type": "parkplatzid", "name": "p1" },
        { "value": "", "type": "betreiberid", "name": "b1" },
        { "value": "", "type": "reservierungsnummer", "name": "r1" },
        { "value": "", "type": "maximalzeitinminuten", "name": "m1" },
        { "value": "", "type": "gebuchteleistung", "name": "g1" } ],

"init": [],
"goal": "(and (reifendruckservice r1) (eparkplatzgebucht p1 r1 m1) (navigationbestaetigung p1))" }
```


Listing 1

Die Anfrage besteht aus drei wesentlichen Teilen. Diese sind environment, init und goal. Das Environment beinhaltet typisierte Variablen aus der Domain, die für die Anfrage (goal) verwendet werden können. Die Variablen können optional auch schon Werte enthalten, falls diese bekannt sind. Der init-Bereich kann einen aktuellen Zustand enthalten, der die Ausgangslage für die Komposition bildet. Es ist also möglich, ein Ziel erreichen zu wollen, das teilweise bereits erfüllt ist, darauf wird in der Komposition eingegangen und muss nicht erneut erreicht werden.

Der goal-Bereich enthält die eigentliche Anfrage des Nutzers. In dem Beispiel sieht man, wie die Variable „p1“ sowohl in der eparkplatzbuchung als auch in der navigationsbestätigung genutzt wird. Gleiches gilt für die Variable „r1“ vom Typ reservierungsnummer, die bei der Reifendurckmessung und der eparkplatzbuchung verwendet wird.

Diese Anfrage wird an die Kompositionskomponete weitergegeben, die anhand der in der Service Registry verfügbaren Service Beschreibungen eine Komposition berechnet. Die dem Beispiel entsprechende Komposition ist in Listing 2 aufgeführt.


```
{ "composition": [
        { "name": "get_parkplatz-e-frei", "params": [ "p1", "b1" ] },
        { "name": "post_reserviere-parkplatz-e", "params": [ "p1", "r1", "b1", "m1" ] },
        { "name": "post_reserviere-reifendruckmessung", "params": [ "p1", "m1", "r1" ] },
        { "name": "get_parkplatz-navigation-parkplatzid", "params": [ "p1" ] } ], "environment": [
        { "name": "p1", "type": "parkplatzid", "value": "" },
        { "name": "b1", "type": "betreiberid", "value": "" },
        { "name": "r1", "type": "reservierungsnummer", "value": "" },
        { "name": "m1", "type": "maximalzeitinminuten", "value": "" },
        { "name": "g1", "type": "gebuchteleistung", "value": "" }
] }
```


Listing 2

Die wesentlichen Bestandteile dieser Nachricht sind composition und das environment. Das environment der initialen Anfrage (Listing 1) bleibt hier unverändert und wird zur Execution-Engine weitergegeben. Die composition ist das Ergebnis der Kompositionskomponente und enthält eine geordnete Liste aller Service Descriptions, die aufgerufen werden müssen, um das Ziel zu erreichen. Enthalten sind hier auch die entsprechenden Parameter, die den Descriptions übergeben werden müssen und auch von den Service Descriptions bereitgestellt werden.

Die so erstellte Komposition wird entsprechend der Beschreibung unter 4.4.5 in einen von der Execution Engine ausführbaren Meta-Flow transformiert und ausgeführt, wobei dann die externen Service-Instanzen für Parkplatz-Suche, Reserviere-Parkplatz, Buche-Wäsche und Navigation aufgerufen werden.

Die Forschungsergebnisse zu diesem Demonstratrationsszenario wurden in wissenschaftlichen Publikationen mit dem Titeln „Emergent Software Service Platform and its Application in a Smart Mobility Setting“ und „Towards Transforming OpenAPI Specified Web Services into Planning Domain Definition Language Actions for Automatic Web Service Composition“ veröffentlicht.

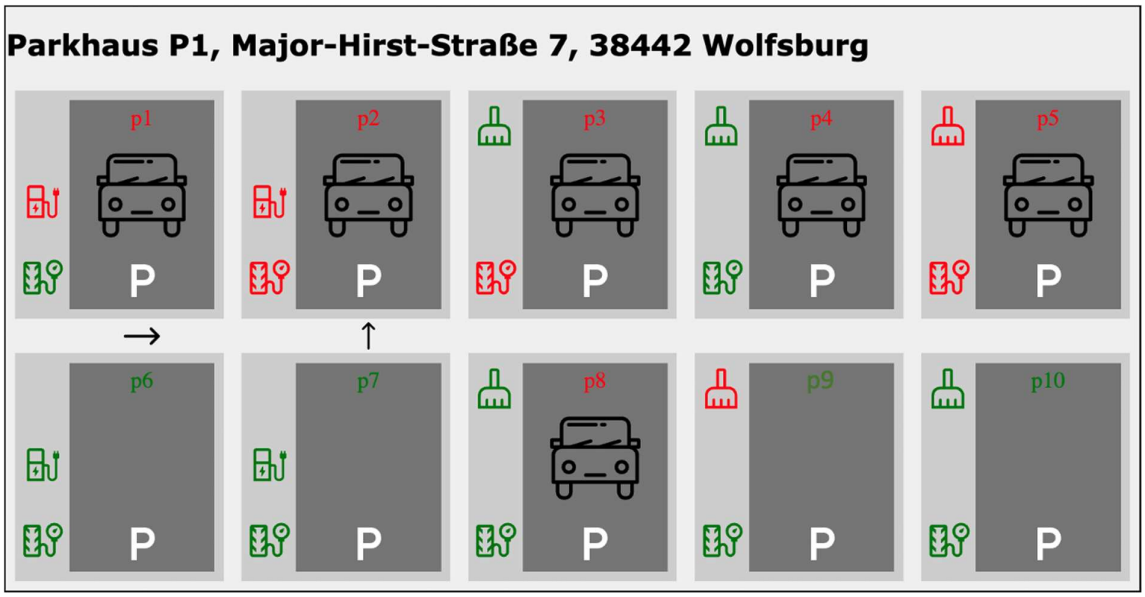


Abb. 13: resultierender Parkhauszustand

5.2 Demonstrationsszenario Smart Office

Wie bereits zuvor schon erläutert, war es aufgrund der Corona bedingten Einschränkungen nicht möglich ein Evaluationsszenario im Bereich Event- und Veranstaltungsmanagement mit einem Anwendungspartner aufzubauen. Daher wurde alternativ ein Evaluationsszenario im Bereich Smart Office direkt an der Universität Mannheim aufgebaut. In diesem Evaluationsszenario soll vor allem die implizite Erkennung von Nutzeranforderungen demonstriert und evaluiert werden.

Abb. 14: Evaluationsszenario Event- und Veranstaltungsmanagement

Wahrscheinlich werden viele der Leserinnen und Leser dieses Berichtes mit der Situation vertraut sein, dass man die App (Applikation), welche genau in diesem Moment benötigt wird, aufgrund der großen Menge von installierten Apps nicht auf Anhieb auf dem Bildschirm des Smartphones finden kann. Das Smart Office Evaluationsszenario betrachtet dieses Problem und zeigt eine mögliche Lösung für dieses Problem auf der Basis der im Rahmen von BIoTope entwickelten impliziten Nutzeranforderungserkennungsansätze auf. Dabei werden im Wesentlichen zwei verschiedene Szenarien betrachtet: Zum einen verschiedene Situationen (Kontexte) die ausschließlich im Büroalltag auftreten in denen allerdings verschiedene Apps auf einem Smartphone relevant sind. Zum anderen verschiedene Situationen die generell während des alltäglichen Lebens auftreten (z.B. Einkaufen gehen, sich Zuhause aufhalten, ins Fitnessstudio gehen, etc.). In diesen Szenarien ist es dann die Aufgabe der im Projekt BIoTope entwickelten Technologie zur automatischen Erkennung von Nutzeranforderungen, aufgrund der Sensordaten, welche vom Smartphone des Nutzers aufgenommen werden, implizit die Anforderungen des Nutzers an die nächste zu verwendende App (entspricht im Kontext dieser Demonstrationsszenarien einem Service) zu erkennen und dem Nutzer auf dieser Basis eine Menge von installierten Apps auf dem Startbildschirm anzuzeigen, die diese Anforderungen bestmöglich erfüllen können. In BIoTope wurden für dieses Szenario bisher nur Apps betrachtet, die lokal auf dem betrachteten Smartphone installiert sind. In diesem Szenario zeigt die Anwendung emergentes Verhalten, da sich die Oberfläche des Startbildschirm zur Laufzeit, emergent an die momentan erkannten Nutzeranforderungen anpasst. Perspektivisch stellt allerdings die Anbindung der auf dem Smartphone direkt ausgeführten Software zur Erkennung der Nutzeranforderungen an eine global gehostete Plattform (wie im Smart Mobility Use Case demonstriert) um zusätzlich eine emergente Komposition von Software Services, welche die momentan erkannten Nutzeranforderungen erfüllen können, zu ermöglichen. Der Ablauf und das Format der Anforderung an die Plattform wäre dann vergleichbar zu dem bereits beschriebenen Mobility Use Case.

Modell-basierte Erkennung von Nutzeranforderungen

Im Rahmen des Smart-Office Use Cases wurde ein im Projekt BIoTope entwickelter Modell-basierter Ansatz zur impliziten Nutzeranforderungserkennung, welcher auf Planungslandmarken basiert, in der Praxis erprobt. Hierfür wurde mithilfe der formalen Beschreibungssprache PDDL ein formales Modell der Büroräume des Instituts für Enterprise Systeme an der Universität Mannheim erstellt. Auf der Basis dieses Modells und aktuell zur Laufzeit, durch die entwickelte Smartphone-App, aufgenommenen Sensordaten konnte dieser Ansatz dann verwendet werden, um automatisch die aktuellen Nutzeranforderungen abzuleiten. Auf der Basis dieser Nutzeranforderungen wurden dann die empfohlenen Apps auf dem Screen des Smartphones angepasst. Da für das Szenario, welches direkt in den Büroräumen des InES lokalisiert war, keine historischen Daten aufgenommen wurden, kann die

Genauigkeit dieses Ansatzes in diesem Szenario nicht exakt quantifiziert werden. Allerdings zeigt das entstandene Demonstrationsvideo, dass die Erkennung der aktuellen Nutzeranforderungen auf der Basis der aktuell aufgenommenen Sensordaten sehr präzise Ergebnisse erzielt.

Modell-freie Erkennung von Nutzeranforderungen

Im Rahmen eines studentischen Projekts wurden für den Smart-Office Use Case Sensordaten von 3 Personen über einen Zeitraum von insgesamt ca. 3 Monaten gesammelt. Aufgrund von technischen Problemen in der frühen Entwicklungsphase konnte von keiner der Personen Daten für den vollen Zeitraum aufgezeichnet werden. Weiterhin ist auch die Anzahl der Tage an denen Daten für die verschiedenen Personen zur Verfügung stehen sehr unterschiedlich. Allerdings stellen die vorhandenen Daten trotzdem eine ausreichende Datengrundlage zur Bewertung des Leistungspotenzials eines Ansatzes zur automatischen Nutzeranforderungserkennung dar. Abbildung 15 zeigt eine Übersicht über die Daten, die von den 3 Testpersonen gesammelt wurden. Die Anzahl der Tage für die Daten pro Person vorliegen schwankt zwischen 23 und 63 Tagen. Dies ist, wie bereits erwähnt, auf technische Schwierigkeiten während der ersten Entwicklungszyklen der Smartphone App zurückzuführen. Weiterhin zeigen die Daten, dass jede der Testpersonen 70 oder mehr Apps während des Zeitraums, in dem die Daten gesammelt wurden verwendet, hat. Dies bestätigt noch einmal, dass ein durchschnittlicher Smartphone Nutzer über einen Zeitraum von wenigen Monaten wesentlich mehr Apps verwendet, als auf eine Seite des Smartphonebildschirms passen würden. Allerdings zeigen die Daten auch, dass es bei allen Testpersonen einige wenige Apps (hier dargestellt sind 9 Apps) gibt, die über 68% der App Nutzungen ausmachen.

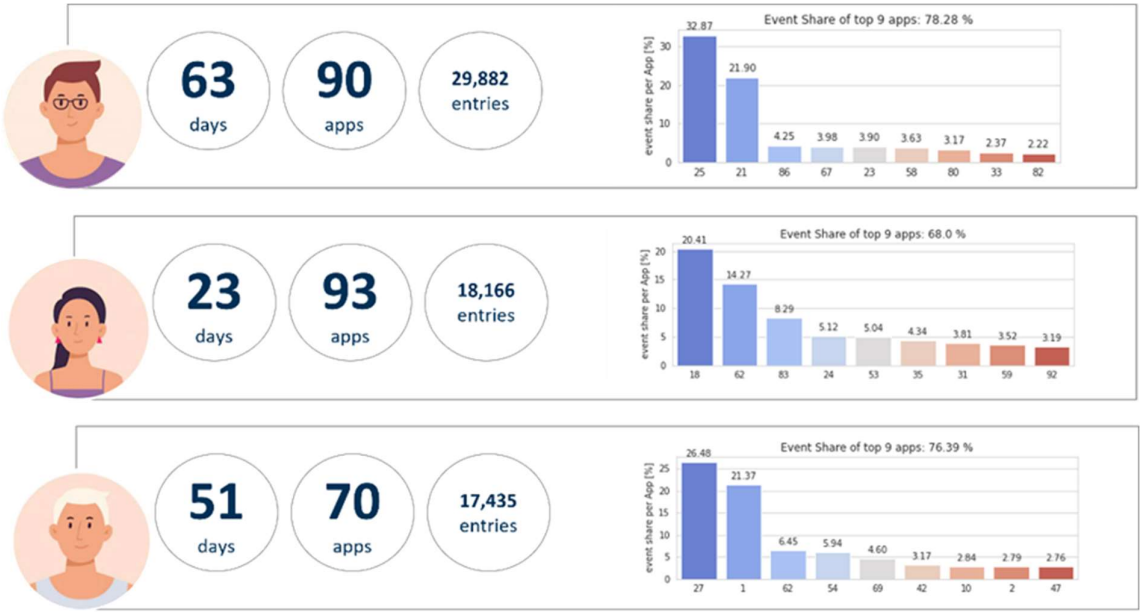


Abb. 15: Überblick App Nutzung

Abbildung 16 zeigt für die Daten jeder Testperson, wie wichtig die verschiedenen Sensorwerte für die Vorhersage des in diesem Szenario verwendeten Naive Bayes Modells sind.

Interessant ist zu sehen, dass die Wichtigkeit der verschiedenen Sensorwerte nicht bei allen Testpersonen identisch sind. Daraus lässt sich ableiten, dass es durchaus relevant ist, ein Modell zur Erkennung von Nutzeranforderungen pro Nutzer zu personalisieren, da die Verhaltensweisen, welche durch die Sensorwerte abgebildet werden, individuellen Mustern unterliegen. Eine zweite Beobachtung, die aufgrund der gesammelten Daten gemacht werden kann, ist, dass die App, welche direkt vor einer anderen App verwendet wurde, einen großen Einfluss auf die Anforderungen des Nutzers an die App, welche als nächstes verwendet wird, hat. Dies wird durch die hohen Werte durch die „packageName_*“ Einträge in der Abbildung deutlich. Diese Beobachtung lässt sich auch intuitiv erklären, da viele Vorgänge die man im Alltag mithilfe von Smartphone Apps bewältigt nicht nur eine App beinhalten, sondern teilweise auch eine Menge von Apps. Ein Beispiel hierfür ist das Beantworten von Nachrichten. Da der durchschnittliche Nutzer inzwischen mehr als eine Messaging App verwendet.

Abb. 16: Nutzung unterschiedlicher Messaging Apps

Abbildung 17 zeigt die Genauigkeit der trainierten Naive Bayes Modelle (es wurde jeweils ein Modell pro Nutzer trainiert) auf den im Projekt gesammelten Daten. Dabei wurde jeweils die Genauigkeit für komplette Kalenderwochen berechnet. Da für jede Testperson unterschiedlich viele Daten gesammelt wurden, sind hier auch die Ergebnisse für unterschiedliche Anzahlen von Kalenderwochen abgebildet. Für jede Kalenderwoche ist die Genauigkeit des Modells jeweils für mehrere Evaluationsschemata abgebildet. Insgesamt werden immer 9 Evaluationsschemata betrachtet. Diese unterscheiden sich dadurch, wie viele der Apps, die von dem Modell mit den höchsten Wahrscheinlichkeiten vorhergesagt werden, für die Genauigkeitsberechnung betrachtet werden. „Top-1“ entspricht dabei, dass die Vorhersage nur als richtig betrachtet wird, wenn die App, welche vom Modell mit der höchsten Wahrscheinlichkeit vorhergesagt wird, auch die App ist welche als nächstes vom Nutzer verwendet wird. „Top-9“ dagegen entspricht dem Schema, dass die Vorhersage als richtig betrachtet wird, wenn die App, welche tatsächlich als nächstes vom Nutzer verwendet wurde, unter den 9 Apps ist, die mit den 9 höchsten Wahrscheinlichkeiten durch das Modell vorhergesagt wurden. Die abgebildeten

Ergebnisse zeigen, dass das Modell mit zunehmender Datenmenge (für jede Woche wurden alle vorher gesammelten Daten als Trainingsdaten verwendet), wie auch zu erwarten, bessere Ergebnisse erzielt. Insgesamt kann man auch sagen, dass die trainierten Modelle gute Genauigkeitsergebnisse erzielen und für 2 von 3 Testpersonen für das Top-9 Evaluationsschema eine Genauigkeit von bis zu 90% erzielen.

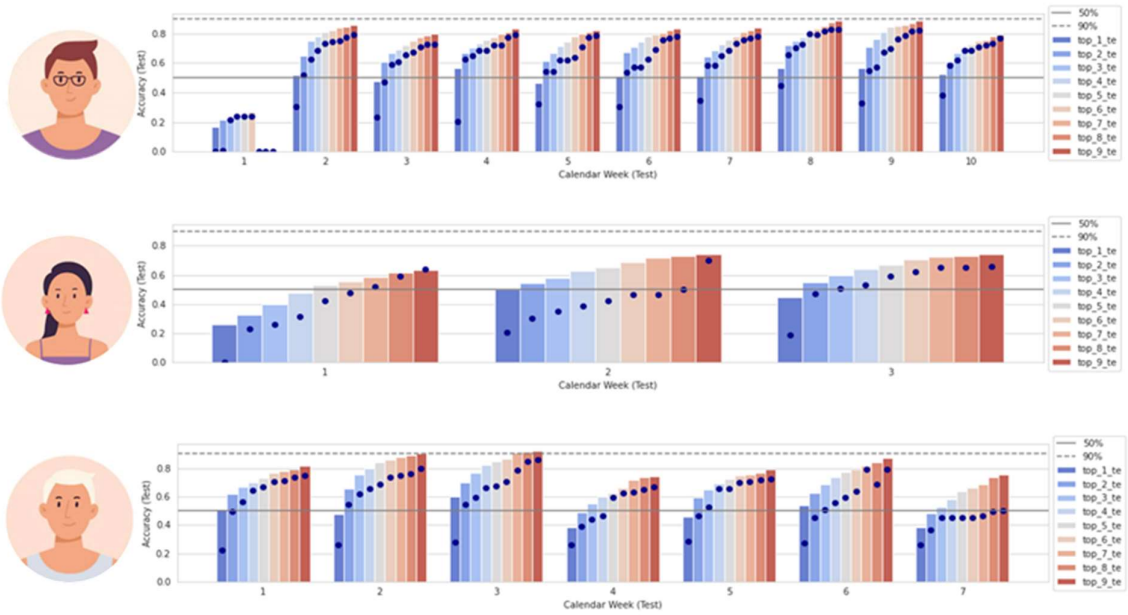


Abb. 17: Genauigkeit der trainierten Naive Bayes Modelle

6 Öffentlichkeitswirksame Maßnahmen

6.1 Abschlusspräsentation

Im Rahmen der Abschlussveranstaltung des Projektkonsortiums konnten dem Projektträger DLR die Ergebnisse in Form einer Powerpoint-Präsentation sowie eines Films gezeigt werden. Alle Projektpartner stellten ihre Ergebnisse vor, die in der vertrauensvollen Zusammenarbeit im Projekt entstanden sind. Besonders der Film konnte veranschaulichen, wie das emergente System funktioniert und die im Mobility Use Case definierten Szenarien abbilden.

6.2 Konferenzbeiträge und Publikationen

Im Berichtszeitraum wurden folgende Publikationen von den Projektpartnern erstellt, eingereicht und angenommen:

- Wilken, N., Ailane, M. T., Bartelt, C., Burzlaff, F., Knieke, C., Lawrenz, S., Rausch, A., Strasser, A. (2020, October). Dynamic adaptive system composition driven by emergence in an IoT based

environment: Architecture and challenges. In Proceedings of the 12th International Conference on Adaptive and Self-Adaptive Systems and Applications, Nice, France (pp. 25-29).

- Burzlaff, F., Jacobs, S., & Bartelt, C. (2020). Automated configuration in adaptive iot software ecosystems to reduce manual device integration effort: Application and evaluation of a novel engineering method. In Proc. 12th Int. Conf. on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE2020).
- Wilken, N., Stuckenschmidt, H., & Bartelt, C. (2021, March). Combining symbolic and data-driven methods for goal recognition. In 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) (pp. 428-429). IEEE.
- Wilken, N., & Stuckenschmidt, H. (2021, March). Combining symbolic and statistical knowledge for goal recognition in smart home environments. In 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) (pp. 26-31). IEEE.
- Burzlaff, F., & Bartelt, C. (2021, May). Knowledge-driven architecture composition: Assisting the system integrator to reuse integration knowledge. In Web Engineering: 21st International Conference, ICWE 2021, Biarritz, France, May 18–21, 2021, Proceedings (pp. 305-319). Cham: Springer International Publishing.
- Burzlaff, F., Ackel, M., & Bartelt, C. (2021). Gabble: Managing Integration Knowledge in IoT-Systems with Logical Reasoning.
- Burzlaff, F. (2021). Knowledge-driven architecture composition.
- Burzlaff, F., Ackel, M., & Bartelt, C. (2022). Applying Knowledge-Driven Architecture Composition with Gabble. In European Conference on Software Architecture (pp. 283-305). Springer, Cham.
- Wilken, N., Cohausz, L., Schaum, J., Lüdtke, S., Bartelt, C., & Stuckenschmidt, H. (2022). Leveraging planning landmarks for hybrid online goal recognition. CEUR Workshop Proceedings.
- Schindler, C., Knieke, C., Rausch, A., & Nyakam Chiadjeu, E. D. (2023). Towards Transforming OpenAPI Specified Web Services into Planning Domain Definition Language Actions for Automatic Web Service Composition. In Proceedings of the 15th International Conference on Adaptive and Self-Adaptive Systems and Applications, Nice, France.
- Knieke, C., Nyakam Chiadjeu, E. D., Rausch, A., Schindler, C., Bartelt, C., Wilken, N., & Ziebura, N. (2023). Emergent Software Service Platform and its Application in a Smart Mobility Setting. In Proceedings of the 15th International Conference on Adaptive and Self-Adaptive Systems and Applications, Nice, France.

6.3 Dissertationen

Zu Themen des BIoTope Projekts werden bzw. wurden diverse, zum Zeitpunkt der Verfassung dieses Berichts bereits weitgehend abgeschlossene Doktorarbeiten an den beiden am Projekt teilnehmenden Universitäten durchgeführt. Diese Arbeiten konstituieren einen ansehnlichen Beitrag zur Förderung des technisch-wissenschaftlichen Nachwuchses; darüber hinaus werden mit den zugeordneten, öffentlich verfügbaren Dissertationen, die in BIoTope entwickelten Ergebnisse und Methoden in besonderer Tiefe und Ausführlichkeit dargestellt.

6.4 Projektvideos

Die im Projekt entstandenen Videos werden auf Youtube veröffentlicht.

7 Ergebnisverbreitung und Verwertungsplan

7.1 Verwertungen im Unternehmensbereich

Ausgehend von den Arbeitsergebnissen der Konsortialpartner und den Entwicklungen der StoneOne AG (Anaqor AG) wird eine konkrete Verwertung der Ergebnisse angestrebt, der die Erfolgschancen und Schritte für die Umsetzung der Projektergebnisse bis zur Marktreife darstellt sowie die weitere Verwertung und Anschlussfähigkeit eruiert. Neben Marketinginitiativen mit der Zielgruppe Messegesellschaften sollen insbesondere auch Mobilitätsanbieter mit den Projektergebnissen angesprochen werden und für die entwickelten Technologien begeistert werden.

Projektbegleitend wurden bereits erste Initiativen zur Vermarktung gestartet. Zum einen nutzt StoneOne verschiedene Marketingformate, um BIoTOPE potenziellen Vertriebspartnern und Endkunden vorzustellen. Hierzu zählen Roadshows, Messen und Fachveranstaltungen sowie die Präsenz im Forum Digitale Technologien. Die Mitwirkung bei entsprechenden Veranstaltungen wird in 2023 ff weiter verstärkt, so plant StoneOne etwa auf der Hannover Messe 2024 mit einem eigenen Exponat vertreten zu sein.

7.1.1 Zielgruppe Messegesellschaften

StoneOne (Anaqor) wird sich auf 25 Messestandorte fokussieren und mit Verbänden zur Organisation zusammenarbeiten (AUMA e.V., mBDKV e.V., …). Begleitend hierzu werden Direct Marketing Kampagnen geplant und Key Accounts angesprochen.

7.1.2 Zielgruppe Mobilitätsanbieter

Eine wesentliche Zielgruppe sind große Parkhausbetreiber und Parkhäuser. StoneOne (Anaqor) wird ca. 20 Anbieter mit > 4.000 Stellplätzen ansprechen und in den Rahmen des Direct Marketing + Key Account Managements integrieren.

Begleitend analysieren wir den Markt, um potenzielle Branchen und Segmenten mit IoT-Anforderungen mit designten/generierten Workflows aus einfachen (elementaren) Bestandteilen zu identifizieren.

7.1.3 Präsentation im Rahmen der Initiative #Wolfsburg Digital

Die bewährte Zusammenarbeit mit der Initiative #Wolfsburg Digital wird im Rahmen der weiteren Entwicklung Smart City mit den Stadtwerken und weiteren Stakeholdern fortgesetzt.

7.2 WAG

In der Verwertung der Projektergebnisse aus Sicht des Partners Wolfsburg AG werden die Ergebnisse im Rahmen der Initiative #WolfsburgDigital präsentiert. Dabei werden die Smart City Anwendungen vor Stadt Wolfsburg, Volkswagen, den Stadtwerken Wolfsburg AG und weiteren Stakeholdern präsentiert. Weitere Verwertungsmöglichkeiten werden geprüft.

7.3 TUC und UMA

Die wissenschaftlichen Projektpartner TUC und UMA konnten im Rahmen des Projekts den Transfer von Forschungsergebnissen in einen praxis- und industrienahen Use Case erfolgreich vollziehen. Die dabei gewonnenen Erfahrungen konnten darüber hinaus über wissenschaftliche Publikationen bereits mit der wissenschaftlichen Community geteilt werden. Hierdurch konnten auch die Promotionsvorhaben, von im Projekt arbeitenden Doktoranden, weiter vorangetrieben werden. Des Weiteren wurden die im Projekt gewonnenen Erfahrungen und Erkenntnisse in diversen Lehrveranstaltungen der wissenschaftlichen Partner eingebracht. Zusätzlich wurde den Studierenden der verschiedenen Partner die Mitarbeit an den innovativen Themengebieten im Projekt in Form von Abschlussarbeiten oder hilfswissenschaftlichen Tätigkeiten ermöglicht.

Neben den bereits erfolgten Maßnahmen zur Ergebnisverbreitung und -verwertung stellen die im Projekt gewonnenen Erfahrungen und erzielten Ergebnisse eine wichtige Rückmeldung für die Planung zukünftiger Forschungsvorhaben dar. Hierbei ist insbesondere die experimentelle Forschung im Bereich Data Science von der umfangreichen Verfügbarkeit realer Daten abhängig.

Zuwendungsempfänger: StoneOne AG (Anaqor)	Förderkennzeichen: 01IS18079A
Titel des Teilvorhabens: **BIoTope**	
Projektleiter: Andreas Liebing (Stoneone / Anaqor)	Tel.: +49 030 46999 4176 Email: andreas.liebing@anaqor.io
Laufzeit des Vorhabens: von 01.06.2019 bis 31.12.2022	
Berichtszeitraum: von: 01.06.2019 bis 31.12.2022	Datum: **11.03.2023**

Einleitung

Ziel des Projektes war die Entwicklung einer Basistechnologie und Engineering-Methodik, um die Genese von emergenten Services auf selbstadaptiven Systemplattformen zu ermöglichen. Dabei sollen die Kompositionsregeln nicht zentral und statisch durch die Plattform vorgegeben werden, sondern dynamisch und nachfrageorientiert konfiguriert werden können. Zudem sollen insbesondere offene Systeme, deren Schnittstellen nicht durch ein einheitliches semantisches Modell standardisiert sind, durch den zu entwickelnden FuE-Ansatz unterstützt werden, damit auch IoT-Systeme ohne Branchenstandard integriert werden können.

Überblick

Um Services auf der Anaqor-Plattform miteinander zu verbinden, benötigt man eine einheitliche Beschreibungssprache und Ausführungsumgebung. Die Execution Engine stellt eine Reihe von Basisoperationen zur Verfügung, die zu einem ausführbaren Flow kombiniert werden können. Dieser Flow wird von der Engine interpretiert und ausgeführt. Der Flow kann automatisch basierend auf einem Composition-Result erstellt oder manuell über ein Service-Composer-UI erstellt werden. Ein integrierter Editor ermöglicht die Erstellung und Visualisierung des Flows. Die manuelle Erstellung eines Flows ist erforderlich, um externe Service-Instanzen in die Plattform zu integrieren. Jeder Flow kann auch als Baustein in anderen Flows wiederverwendet werden. Dies legt den Grundstein dafür, automatisch höherwertige und direkt ausführbare Flows aus einem Composition-Result zu generieren.

Ein Composition-Result besteht aus einer Reihe von Action-Aufrufen, sowie einem Environment, welches unter anderem die Typdefinitionen enthält.
Da zu jeder Action mindestens ein passender Service-Flow in der Service-Registry der Execution-Engine enthalten ist, kann ein passender Meta-Flow (bestehend aus Aufrufen weiterer Flows) automatisch erzeugt werden, indem die Service-Flows entsprechend der Reihenfolge der Actions im Meta-Flow nacheinander aufgerufen und die benötigten Eingangsdaten jedes Service-Flows mit den Werten der Ausgangsdaten der zuvor ausgeführten Service-Flows belegt werden

Bei der Ausführung eines Meta-Flows werden die enthaltenen Sub-Flows, die als Service-Nodes fungieren, als eigenständige Services abgearbeitet. Dies geschieht rekursiv, da auch die Sub-Flows wiederum integrierte Sub-Flows enthalten können. Jeder Meta-Flow, ob dynamisch oder manuell erstellt, kann somit Teil eines neuen Meta-Flows werden. Um einen dynamisch erzeugten Flow zu verwenden, wird lediglich eine Action-Referenz auf eine geeignete Servicebeschreibung in der Service-Registry der Composition-Komponente benötigt.

Anaqor war im Projekt für die Entwicklung der Execution Engine zur wirtschaftlichen Verwertung der Arbeitsergebnisse verantwortlich.

Damit Services auf der Plattform kompatibel zu weiteren Services zusammengebaut werden können, benötigt man eine einheitliche Beschreibungssprache und Ausführungsumgebung.

Die Execution Engine definiert eine Reihe von Basisoperationen, die zu einem ausführbaren Flow zusammengebaut werden können, welcher dann von der Engine interpretiert und ausgeführt werden kann.

Sie wurde in enger Abstimmung mit den Konsortialpartnern der Technischen Universität Clausthal, Der Universität Mannheim und der Wolfsburg AG entwickelt, um den Anforderungen an Kompatibilität der entwickelten Partnermodule gerecht zu werden und die wirtschaftliche Nutzbarkeit zu gewährleisten.

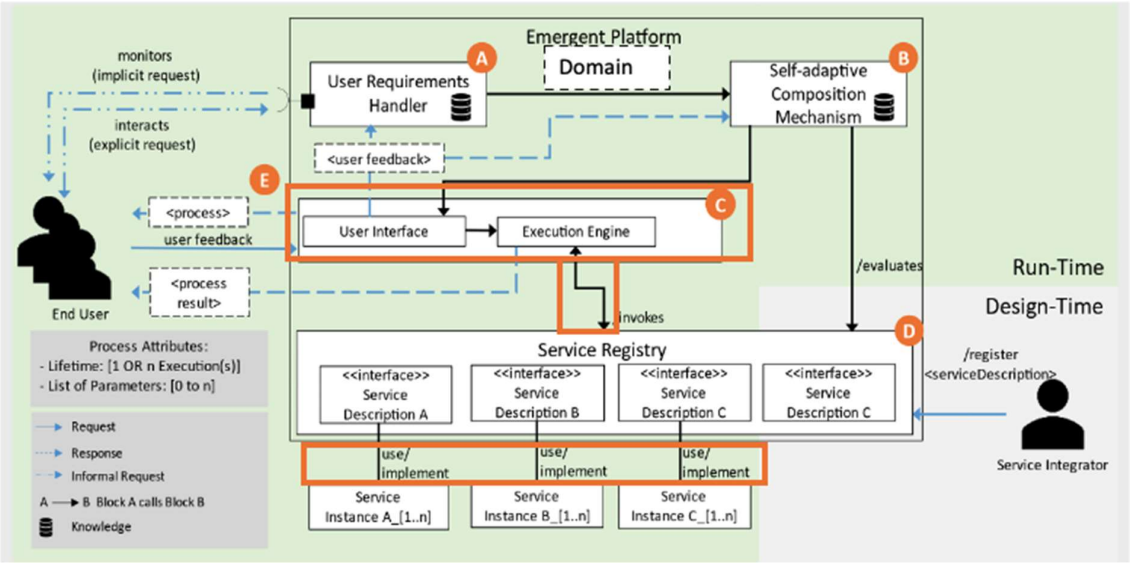


Abb.: Execution Engine

Die beschriebene Architektur einer emergenten Softwareplattform besteht aus vier Komponenten: dem User Requirements Handler (URH), dem Self-adaptive Composition Mechanism (SCM), der Execution Engine (EE) und der Service Registry (SR). Der URH erkennt Nutzeranforderungen entweder explizit durch Anfragen des Nutzers oder implizit durch Überwachung der Nutzer über Sensorinfrastruktur. Der SCM komponiert automatisch einen Software-Service, der die erkannten Nutzeranforderungen erfüllt. Die EE führt den komponierten Software-Service aus und wählt die entsprechenden Software-Service-Instanzen aus. Die SR stellt eine Wissensbasis über verfügbare Software-Services in der Plattform bereit.