Carl von Ossietzky Universität Oldenburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Architekturprozess und Bewertungsmethoden für Testfelder Intelligenter Verkehrssysteme

Von Fakultät für Informatik, Wirtschaft und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Gerades und Titels eines Doktors der Naturwissenschaften (Dr. rer. nat.)

angenommene Dissertation von

Herrn Dirk Beckmann,
geboren am 30. Mai 1974 in Iserlohn

6. Oktober 2023

Gutachter:

Prof. Dr.-Ing. Frank Köster

Prof. Dr.-Ing. Axel Hahn

Tag der Disputation: 21. August 2023

Zusammenfassung

Intelligente Transport- und Mobilitätsdienste ermöglichen nicht nur wesentliche Gewinne an Sicherheit, Effizienz und Komfort im Verkehr, sondern adressieren auch wichtige gesellschaftliche Herausforderungen wie Stau, Luftverschmutzung und Verkehrslärm. Die zunehmende Komplexität dieser Verkehrstelematikanwendungen erfordert besondere Sorgfalt hinsichtlich Erforschung, Entwicklung und Erprobung. Testfelder unterstützen dies als wesentliches Werkzeug. Aufgrund der breit gefächerten Anforderungen, changierender Testsysteme und technologischer sowie fachlicher Weiterentwicklungen ist ein nachhaltiger Entwurf dieser Testfelder nicht trivial. Dennoch folgten Planung, Aufbau und Erweiterung von Testfeldsysteminfrastruktur bislang oft keiner strukturierten Methode, die den Entwurf als frühestes Artefakt adäquat absichert.

Die vorliegende Arbeit zeigt für diese Aufgabenstellung einen Entwurfs- und Bewertungsprozess für Testfelder, der auf Grundlage einer umfassenden Anforderungsdefinition mit einer strukturierten Herangehensweise zu einer Testfeldarchitektur führt, die sowohl funktionalen als auch nicht-funktionalen Anforderungen wie Flexibilität, Austauschbarkeit und Koexistenz entspricht. Damit lässt sich eine robuste und nachhaltige Architektur erzielen, die vor der Realisierung hinsichtlich Risiken und empfindlicher Aspekte bewertet werden kann.

Der Entwurfsprozess nutzt ein Zerlegungsverfahren, um die funktionalen Anforderungen in kleinste atomare Diensteinheiten zu zerlegen. Diese können dann zu größeren Prozessen im Testfeld choreographiert werden. Zudem lehnt sich dieser systematische Vorgang an das *4+1-Sichten*-Softwarearchitekturmodell an und stellt so sicher, dass der resultierende Entwurf hinreichend beschrieben ist. Dieser Lösungsansatz wird nicht nur anhand konkretisierender Aspekte dargestellt, sondern auch in der Praxis unter realistischen Bedingungen und vergleichbarem Kontext angewandt werden. Die daraus gewonnenen Ergebnisse werden evaluiert und kritisch bewertet.

Darüber hinaus wird ein Bewertungsverfahren entwickelt, mit dem u. a. nicht-funktionale Anforderungen sensitiv bewertet werden können. Mit diesem kann ein Testfeldentwurf nicht nur früh hinsichtlich Qualitätsmerkmale und kritische Aspekte bewertet, sondern auch mit alternativen Entwürfen verglichen werden.

Die dargestellte systematische Architekturentwicklung sowie die Bewertung dessen ermöglichen die Entwicklung einer nachhaltigen Testfeldarchitektur. Dies erlaubt eine wirtschaftliche Erprobung von komplexen Anwendungsfällen in der ITS-Domäne.

Abstract

Intelligent transportation systems (ITS) promise significant gains in safety, efficiency and comfort in the transport domain by addressing major societal challenges, such as traffic congestion and air pollution. Due to their wide-ranging requirements and the constant evolution of technology, testing ITS solutions prior to their construction and commissioning is one of their major challenges. Planning and designing a test site in a versatile and future-proof way is therefore not a trivial task. To date, the planning, construction and evolution of a test site infrastructure has typically lacked a structured method to adequately accommodate a dedicated software development process. This document presents a tailored design and evaluation process for ITS test sites that considers both functional and non-functional requirements.

The key objective is to create a method that achieves a robust and sustainable architecture enabling the evaluation of risks and special sensitivities before undertaking the final construction of the site. The design approach uses a decomposition process to break down functional requirements into the smallest atomic service units. These units can then be choreographed to form larger processes that correspond to use cases of the test site. This systematic process adopts the '4+1 views' software architecture model to ensure that the resulting design is sufficiently described. The approach is demonstrated by applying it to representative aspects of a typical test site design. Development is conducted under realistic conditions and in a similar context. The outcome is analysed and critically evaluated to ensure feasibility and suitability. The design process is flanked by an architecture evaluation method that focusses on the sensitive assessment of non-functional requirements, such as flexibility, interchangeability and coexistence. This allows a thorough evaluation of quality characteristics and critical aspects early in the design process and also allows the comparison of the outcome with alternative designs.

Inhalt

1 Einleitung

Bei der Entwicklung und Erprobung von Fahrassistenzsystemen und automatisierten vernetzten Fahrzeugen ist im Allgemeinen ein hoher Aufwand zu leisten.
Während einfache infrastrukturgebundene Dienste wie ARI[1] mit vergleichsweise einfachen Mitteln erprobt und aufsetzend auf etablierte Rundfunktechnologien leicht eingeführt werden konnten, waren bereits erste komplexere sicherheitskritische Systeme wie bspw. ACC[2] nur mit deutlich höherem Aufwand zu erproben [WDS12].
In den letzten Jahren hat die rapide Weiterentwicklung im Bereich der Elektronik und insbesondere der Kommunikationstechnologie neue Grundlagen für immer komplexere Assistenz- und Automationssysteme geschaffen und beeinflusst das junge Forschungsfeld um Intelligent Transportation Systems (ITS). Die mit der Metapher ITS verknüpften intelligenten Verkehrssysteme zielen auf die breit gefächerte Unterstützung von Verkehrsteilnehmern und Reisenden mittels Kommunikations- und Informationstechnologie in Fahrzeugen und Verkehrsinfrastruktur unabhängig von der von ihnen genutzten Transportmode. Dabei steht nicht nur die Unterstützung des individuellen Fahrers, bspw. hinsichtlich Komfort und Sicherheit, im Vordergrund, sondern es werden auch gesamtverkehrliche Fragestellungen adressiert, wie bspw. Stauvermeidung und Umweltaspekte [FJM+01, S. 1206].
Durch die wachsende Verkehrsdichte in Industriestaaten[3] und den zunehmenden Individualverkehr in Entwicklungs- und Schwellenländern, wie bspw. Nigeria [JA05, S. 47], sind insbesondere Ballungsräume bestrebt, die negativen Aspekte wie Staus, Umweltverschmutzung, Lärm und Parkraummangel nachhaltig zu mildern [MA08, S. 26]. ITS werden als ein wichtiges Werkzeug gesehen, dieses Ziel zu erreichen. Für ländlich geprägte Regionen bieten ITS Lösungsoptionen, die auch die dort beobachteten Verkehrsprobleme lösen helfen – z. B. die Problematik nicht kostendeckender ÖPNV-Angebote in klassischen Bedienformen. Dieses technische und wirtschaftliche Potenzial wird auch auf europäischer

[1] Autofahrer-Rundfunk-Information (ARI); Kennzeichnung von Radioverkehrsdurchsagen; zwischen 1972 u. 1974 entwickelt, zwischen 1974 und 2005 in Deutschland betrieben
[2] Adaptive Cruise Control (ACC), Abstandsregeltempomat; ab ca. 1995 serienreife [VE03]
[3] vgl. hierzu [Kut05, S. 28ff]

und globaler Ebene erkannt.[4] So beschreibt bspw. die Europäische Kommission (EC) in der Richtlinie 2010/40/EU ([Eur10]) einen Maßnahmenkatalog zur Förderung u. a. von multimodalen Reiseinformationsdiensten und der Bereitstellung EU-weiter Verkehrsinformationsdienste in Echtzeit ([Eur10, S. 4]). Konkret wird seitens der Kommission der in dieser Arbeit relevante Aufbau kooperativer Dienste (s. [Kom08a, S. 12]) und kooperativer Systeme sowie deren Evaluation und Bewertung (s. [Kom08a, S. 13]) mittelfristig geplant. Längerfristig werden ITS als wichtiges Thema adressiert ([Kom11, S. 15ff] u. [Kom08b]). Diese Relevanz wird durch die momentan viel beachteten Entwicklungen zu hohen Automationen im Verkehr unterstrichen [DMM15, S. 3]. Durch deren flächendeckenden Einsatz werden nicht nur deutliche Erfolge beispielsweise hinsichtlich Verkehrssicherheit und CO_2-Reduktion erwartet, sondern auch enorme wirtschaftliche Gewinne bspw. durch Stauvermeidung und reduzierte Treibstoffkosten vorhergesagt [RWG16, S. 4f]. Die Entwicklung hoher Automationsstufen folgt einer evolutionären Entwicklung und baut somit auf einfacheren Assistenzen auf [DMM15, S. 9f]. In der praktischen Umsetzung werden diese Entwicklungen und deren Erprobung durch Testfelder unterstützt [DMM15, S. 26]. Analoge Handlungsfelder auf nationaler Ebene werden in der Strategie des Bundesministeriums für Verkehr und digitale Infrastruktur (BMVI) zum automatisierten und vernetzten Fahren dargestellt [Bun15]. Dabei ist u. a. die Verifikation und Validierung hochautomatisierter Fahrfunktionen [PEG20, S. 160] eine wesentliche Herausforderung.

Aus Aufbau, Evaluation und Bewertung intelligenter Systeme sind zunehmend anspruchsvollere Anforderungen an die zugrunde liegende Testinfrastruktur abzuleiten. Dies gilt insbesondere für kooperative Systeme, die auf einer verteilten Rechnerinfrastruktur (bspw. in unterschiedlichen Fahrzeugen, stationären Einheiten der Infrastruktur, Cloud und Mobile-Edge basierten Lösungen) sowie heterogenen Kommunikationstechnologien wie mobilem Internet und IEEE[5]-Standard IEEE 802.11p aufbauen [FSK12, S. 3]. So tauschen zum Beispiel Assistenzsysteme, die vor plötzlich auftretenden Gefahrenstellen wie Unfallstellen oder Hindernissen auf der Strecke warnen, Daten auf Grundlage solcher Kommunikationskonzepte aus und sind im Feld nur mit gravierendem Aufwand zu validieren [GNH+10, S. 491]. Dabei wird nicht nur die technische Korrektheit des Systems bewertet, vielmehr werden in der Erprobungsphase einer solchen Applikation auch nicht-funktionale Aspekte untersucht. So sind bspw. die Bedienbarkeit des Systems, die Akzeptanz durch den Fahrer

[4] Bspw. [Mar15] prognostiziert etwa eine Verdoppelung des Marktvolumens von $ 36,10 Mrd. in 2015 auf $ 63,66 Mrd. in 2022.

[5] Institute of Electrical and Electronics Engineers (IEEE), bildet u. a. weithin anerkannte Gremien zur Standardisierung

und die Wirkung auf die gesamtverkehrliche Situation ebenfalls ein möglicher Untersuchungsgegenstand, der von einem umfassenden Testfeld unterstützt werden muss. Daher sind Verifikation und Validierung unter diesen Gesichtspunkten wesentliche Aufgaben, die von einem Testfeld unterstützt werden müssen.

Eine Testlandschaft sollte zudem möglichst für eine große Vielzahl von Anwendungsfällen dienlich sein, um einen hohen Grad an Wiederverwendbarkeit darzustellen. Darüber hinaus sind die Testsysteme selbst Forschungsgegenstand und werden kontinuierlich weiterentwickelt. Daher müssen sowohl weitsichtig zukünftige Technologien integrierbar sein, als auch bestehende, heterogene und proprietäre Infrastruktur sinnvoll eingebettet werden. Es müssen generische Konzepte realisiert werden, um beste Flexibilität und Anpassbarkeit zu gewährleisten [QPIJ10, S. 2]. Einen guten Einblick in plausible Szenarien im Bereich der automatisierten Mobilität bietet beispielsweise die acatech-studie zum automatisierten Verkehrssystem [Lem15].

Hieraus leiten sich hohe Anforderungen im Hinblick auf adäquate Erprobungsmethodiken in unterschiedlichen Reifegraden der benötigten Testsysteme ab. Hier reicht die Bandbreite von formaler Verifikation von (Teil-)Funktionen über die Einbettung von ausgereiften Funktionseinheiten in eine virtuelle Testlandschaft bis hin zur Demonstration vollständiger bzw. serienreifer Systeme im Testfeld [GSV15, S. 29].

Obwohl der Einsatz eines Testfelds eine kostengünstige Systementwicklung verfolgt, sind solche Testfelder typischerweise sowohl im Aufbau als auch im Unterhalt mit gravierenden Kosten verbunden.Für kleine und mittelständische Unternehmen ist eine solche Investition oft kaum aufzubringen. Selbst für größere Unternehmen ist eine Erprobung neuer Assistenzen/Automation kostspielig und stellt ein erhebliches wirtschaftliches Risiko dar. Entsprechend der Kosten-Nutzen-Rechnung erreichen so viele tragfähige Ideen nicht die Realisierungsphase und werden nicht weiter verfolgt [LL02]. Daraus kann auch die Idee eines konföderierten Testfelds entwachsen, in dem sich mehrere größere Einrichtungen oder Parteien zusammenschließen, um aus Kostengründen ein gemeinsames generisches Testfeld mit Plattformcharakter zu errichten bzw. zu betreiben. Ebenso ist es sinnvoll, dass Versuchskampagnen über mehrere Testfelder verteilt durchführbar sind, um bspw. eine neue Assistenz unter realen Bedingungen zu erproben (bspw. Klima, regionalem Fahrerverhalten, spezieller verkehrlicher Situationen) [VBK10, S. 2]. Dadurch ergeben sich für den Aufbau eines Testfelds zahlreiche funktionale und insbesondere nicht-funktionale Anforderungen [SFT+11].

1.1 Motivation

Um komplexe Assistenz- und Automationssysteme gleichermaßen wirtschaftlich wie sicher zu entwickeln, bedarf es einer umfassenden Testlandschaft, die den gesamten Entwicklungsprozess unterstützt. Diese basiert auf einer gleichermaßen ausgedehnten wie komplexen Systeminfrastruktur. Testfelder sind in einigen Forschungsprojekten bereits aufgebaut worden (s. [FOT11]). Historisch betrachtet sind viele jedoch auf einen spezifischen Anwendungsfall[6], eine bestimmte Phase oder eine eng abgesteckte Klasse von Testfällen[7] ausgerichtet. Oft sind diese Feldtests von geringer Nachhaltigkeit, da sie an kurzlebige Projekte gebunden sind, und nur in wenigen Fällen wird ein längerfristiger Aufbau angestrebt, wie bspw. in Test Site Norway (TSN), einem norwegischen Testfeld mit projektübergreifender Laufzeit. Diese Beobachtung wird auch durch eine Auflistung 15 aktueller Testfelder für das automatisierte und vernetzte Fahren [Bun19] gestützt. In dieser sind wenige Installationen, die auf eine lange Historie zurückblicken und vermutlich eine hohe Wiederverwendung von Kernfunktionalität erreichen.
Unabhängig von der Laufzeit sind die aufgeführten Testfelder jedoch relativ isolierte Installationen mit teils proprietärem und hochgradig provisorischem Charakter. Dadurch ist ein sinnvoller Austausch von Komponenten, Software oder auch Daten zwischen Testfeldern durch deren Aufbau erschwert. Somit ist die Erprobung von Testkomponenten und Softwaremodulen in fremden Umgebungen kaum möglich und die Vergleichbarkeit der erfassten Daten praktisch nicht gewährleistet.
Die mangelnde Nachhaltigkeit dieser Installationen wurde ebenfalls in Forschungsprojekten adressiert. Insbesondere relevant sind die generellen Architekturüberlegungen aus dem Projekt ITS Test Beds [VBK10]. Die dort entwickelten Architekturkonzepte können besonders im Bereich der Integration von bestehenden Systemen [BVJ+11] und der Einbindung eines Testfelds in den Entwicklungs- und Erprobungsprozess [JvdB11] eine wichtige Ausgangsbasis bilden. Ein Manko ist allerdings, dass auch hier abgesicherte Entwurfsprozesse keine Rolle spielen [BVJ+11].
Relevante Testfeldinitiativen mit langfristiger Ausrichtung sind die Anwendungsplattform Intelligente Mobilität (AIM)[8] des Deutsches Zentrum für Luft- und Raumfahrts (DLRs)

[6] bspw. *LAVIA* (Limiteur s'Adaptant à la VItesse Autorisée), 2001 bis 2006 [ESLR06]; *Pay As You Speed*, 2004 bis 2009, [AWT+08]

[7] bspw. *euroFOT* [BFB10], acht Fahrerassistenzsysteme

[8] Großforschungsanlage zur vernetzten Forschung, Entwicklung und Anwendung für intelligente Transport- und Mobilitätsdienste des Deutschen Zentrums für Luft- und Raumfahrt (DLR) in Braunschweig [LBJK11] u. [KHFL11]

in Braunschweig, sowie in Teilaspekten sim^{TD} [FB10], die Testsite Helmond [vNHP10] bzw. Dutch Integrated Testsite Cooperative Mobility (DITCM) [PvvP16] und die Testsite Göteborg [Ryl08b], [Ryl08a]. Die dort realisierten Ansätze können wichtige Grundlagen für die Ableitung von Generalisierungskonzepten zur Definition entsprechender Infrastrukturen liefern.

Im softwaretechnischen Aufbau der Testfelder ist nur in wenigen Fällen (z. B. sim^{TD}) ein klarer methodischer Entwurfsprozess dokumentiert, der nachvollziehbar die Designentscheidungen des Entwurfs offenlegt. In keinem der aufgeführten Forschungsprojekte ist eine fundierte Güteabschätzung des eigenen Aufbaus dokumentiert.

Der Entwurf einer adäquaten Softwarearchitektur wird im Allgemeinen als ein wichtiger ([GS93, S. 4f]) oder sogar entscheidender Faktor für den Erfolg eines softwareintensiven Projektes gesehen ([HNS99, S. 3]). Deshalb sollte bei der Planung und Ausgestaltung eines Testfelds ein besonderes Augenmerk auf die Güte des Architekturentwurfs gelegt werden. Zudem ist ein Testfeldaufbau typischerweise ein komplexes verteiltes System, für welches insbesondere ein gründlicher Architekturentwurf notwendig ist.

Mit dem vorwiegenden Ad-hoc-Vorgehen[9] zur Gestaltung von Testfeldaufbauten lassen sich einige der zuvor genannten Ziele nur schlecht erreichen. Die wesentlichen Nachteile zeigen sich in mangelnder Flexibilität und Wartbarkeit des resultierenden Systems:

Dadurch, dass Testfeldaufbauten oft ad hoc entworfen werden, sind Entwurfsentscheidungen im Nachhinein kaum nachvollziehbar. Daher ist die Beziehung von bestimmten Ausprägungen des Systemdesigns zu konkreten Anforderungen nicht immer rückwirkend bestimmbar. Zudem wird der Architekturentwurf für Dritte schwer interpretierbar, mit entsprechend negativen Auswirkungen auf ein mögliches Reengineering (z. B. Veränderbarkeit, Flexibilität) [QXZ08, S. 26f], [GR08, S. 20]. Zudem sind Entwurfsentscheidungen gleichermaßen von nicht-funktionalen Anforderungen geprägt (wie heute an vielen Stellen noch von Sicherheitsanforderungen). Diese Anforderungen führen zu komplexen Verknüpfungen im Entwurf und tangieren oft das ganze System [GJKF10, S. 207f]. Änderungen und Anpassungen sind hier besonders kritisch ([Bos04, S. 196], [BB98]). Daher ist eine sorgfältige Berücksichtigung von entsprechenden Eigenschaften wie bspw. Offenheit, Flexibilität und Migrationsfähigkeit in der Entwurfsphase unabdingbar.

Liegt ein Architekturentwurf vor, sind durch dessen Bewertung vor einer Implementierung relevante Qualitätsmerkmale wie Verfügbarkeit, Robustheit (Fehlertoleranz), Veränderbarkeit und Flexibilität sowie Gesamtkosten der Lösung abschätzbar [S. 2][LRv99], [GR08,

[9] vgl. Architekturentwurf als *ad hoc affair* in [Gar00, S. 94f]

S. 7f]. Durch eine Gegenüberstellung kann der eigene Entwurf über Bewertungen auch in Relation zu bestehenden Ansätzen betrachtet werden, um Teilaspekte mit entliehenen Konzepten zu stärken oder auch grundsätzlich heterogene Konzepte vergleichend zu bewerten. Wird auf eine systematische Bewertung verzichtet, kann der Erfolg eines Entwurfs praktisch nur nach der Ausimplementierung mit evtl. längerer Betriebszeit in der Breite bewertet werden [GR08, S. 18f]. Zu diesem Zeitpunkt ist jedoch u. U. bereits erheblicher Aufwand geleistet worden. Spätere Änderungen an der Architektur können daher massive und kostenintensive Neuimplementierungen bedingen [Bos04, S. 195]. Dies wiegt besonders schwer, da es sich bei Testfeldern im Allgemeinen um großräumig aufgestellte Strukturen handelt.
Durch einen sorgfältigen methodischen Ansatz ist ein nachhaltiger Entwurf zu erzielen, der auch schon vor der ersten Implementierung des Testfeldes qualitativ bewertet werden kann.

1.2 Ziele dieser Arbeit

Aus den zuvor genannten Problemstellungen lassen sich drei zentrale Forschungsfragen für diese Arbeit ableiten:

1. Wie lässt sich eine Architektur im speziellen Kontext des Testfeldentwurfs identifizieren und beschreiben?

2. Wie kann ein methodischer Entwurfsprozess für Testfeldarchitekturen gestaltet werden?

3. Wie lassen sich Testfeldarchitekturen vor der ersten Implementierung qualitativ bewerten und mit anderen Ansätzen vergleichen?

Diese Forschungsfragen formen die konkreten Ziele für diese Arbeit, die im Folgenden ausgeführt werden.
Das Kernziel dieser Arbeit ist die Definition eines geeigneten Entwurfsprozesses für eine umfassende Systemlandschaft, welche die zuvor umrissene Problemstellung im Entwurf komplexer Testfelder berücksichtigt. Dazu wird ein ingenieurmäßiges Vorgehen entwickelt, welches den Architekturprozess zwischen Anforderungsdefinition und erster Implementierung strukturiert und dabei das Entstehen bewertbarer Artefakte fördert. Dafür werden folgende vier Kernthemen vertieft:

- **Architekturmodelle:** Es bestehen einige Paradigmen in der Softwaretechnik wie bspw. dienstorientierte Architekturen, Mediator-basierte Informationssysteme oder auch Client-Server-Modelle, die bei der Definition einer geeigneten Architektur dienlich zu sein scheinen. Dazu werden entsprechende Entwurfsmuster identifiziert und beschrieben. Besonderes Augenmerk liegt hierbei auf der Applizierbarkeit der dargestellten Architekturen auf die Herausforderungen der beschriebenen Fragestellung.

- **Generischer Architekturprozess:** Die angestrebte Architektur dieser Arbeit umfasst typische Kernkomponenten, deren fachlicher Aufbau und Einsatz klar umrissen werden. Eine weitere Herausforderung besteht in der Einbettung vorhandener Infrastrukturmerkmale in mögliche Systemlandschaften. Erst so ist eine erfolgreiche Umsetzung der Konzepte auf Grundlage vorhandener Ressourcen denkbar, da der Migrationsaufwand ohne dedizierte Berücksichtigung von Wiederverwertbarkeit und Adaptierbarkeit nicht zu rechtfertigen wäre. Schwerpunktartig werden dabei Prozesse betrachtet, die es erlauben, vorhandene und neue Entitäten in die umfassende Systemlandschaft einzufügen. Teile dieser Aspekte werden bereits im Architekturentwurf berücksichtigt und schließen die Betrachtung dieser Abläufe im Systementwurf ab.

- **Bewertung von Architekturen:** Die Anwendung verschiedener Entwurfsmuster im Design einer Architektur einer umfassenden Testlandschaft hat immer Konsequenzen für die Realisierung der Konzepte und für die Nutzung als Testinfrastruktur. Somit ist es sinnvoll, a priori Vor- und Nachteile unterschiedlicher Architekturentwürfe zu identifizieren. Auf der Basis einer strukturierten Bewertungsgrundlage lassen sich verschiedene Entwurfskonzepte vergleichen und unterschiedliche Ansätze abwägen. Dazu wird ein auf die Problemstellung zugeschnittener Ansatz zur qualitativen Bewertung von Testfeldarchitekturen aufgebaut, der zudem einen Vergleich unterschiedlicher Entwürfe ermöglicht.

- **Umsetzung und Evaluation:** Es bietet sich an, die methodischen Inhalte in der praktischen Umsetzung kritisch zu analysieren. Dabei kann bewertet werden, ob die in dieser Arbeit vorgeschlagenen Verfahren geeignet sind, eine adäquate Testfeldarchitektur zu erzeugen. Unter dem Gesichtspunkt, dass ein umfassender Testfeldentwurf auf hypothetischen Anforderungen unter entkoppelten Laborbedin-

gungen kein belastbares Ergebnis darstellt, wird dazu auf die Erfahrungen aus der praktischen Umsetzung in inhaltlich vergleichbarer Aufgabenstellung in einer komplexen Arbeitsumgebung mit heterogenen, verteilten Arbeitsteams zurückgegriffen.

1.3 Gliederung der Arbeit

Aufbauend auf Motivation und Zielsetzung in diesem Kapitel werden die fachlichen Grundlagen sowie der Stand der Technik im Kapitel 2 genauer dargestellt. Daraus wird ein methodischer Ansatz zur Spezifikation einer Testfeldarchitektur entwickelt. Dessen Entwicklungsschritte werden anhand typischer Aspekte eines Testfeldentwurfs diskutiert und in Kapitel 3 sukzessiv dargestellt. Kapitel 4 zeigt einen strukturierten Ansatz zur Bewertung von Qualitätsmerkmalen einer Testfeldarchitektur. Zudem wird ein Konzept zur vergleichenden Architekturbewertung entwickelt und so eine Relativierung zwischen verschiedenen Testfeldarchitekturen ermöglicht. Um die methodischen Überlegungen zur Spezifikation einer Testfeldarchitektur u. a. hinsichtlich Durchführbarkeit und Ergebnisqualität zu beurteilen, wird in Kapitel 5 die praktische Umsetzung der Methode aus Kapitel 3 in einem vergleichbaren Kontext diskutiert und bewertet. Das letzte Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick zu möglichen weiterführenden Arbeiten. Abbildung 1.1 illustriert die Gliederung dieser Arbeit und stellt wesentliche Ergebnisse der entsprechenden Kapitel dar.

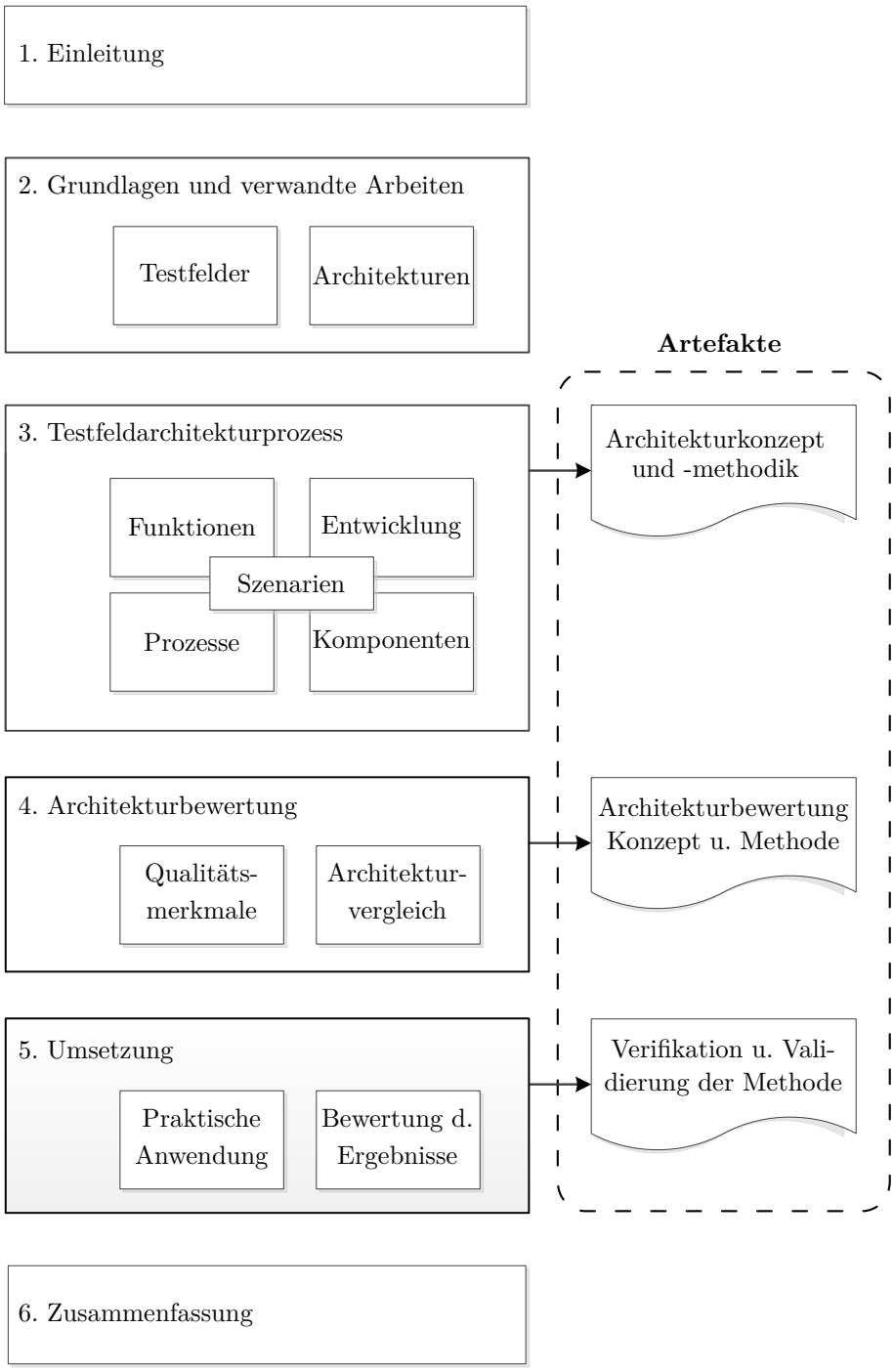

Abbildung 1.1: Gliederung dieses Vorhabens

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel werden die Grundlagen für die Ergebnisse dieser Arbeit ausgeführt und die entsprechende Reflexion in der Literatur dargestellt.
Zentrale Ziele dieser Arbeit sind die Definition eines geeigneten Architekturprozesses für Testfelder sowie die Bewertung von Testfeldarchitekturen. Dazu ist es notwendig, die domänenspezifischen Aspekte, wie den typischen technischen Aufbau eines Testfeldes und typische Anwendungsfälle einer Testlandschaft zu untersuchen, um diese für die Architektur und Bewertung zu berücksichtigen. Entsprechend betrachten die folgenden Abschnitte 2.2 und 2.3 die domänenspezifischen Komponenten in repräsentativen Anwendungsfällen und entsprechende Bausteine einer Testlandschaft. Das Zusammenspiel dieser Einheiten hat maßgebliche Relevanz für die Definition einer Architektur und wird daher in Abschnitt 2.4 vertiefend betrachtet. Die Testfeldarchitekturen aus den Projekten AIM, sim$^{\mathrm{TD}}$ und ITS Test Beds werden als Beispiele im Abschnitt 2.5 diskutiert.
Die Grundlagen für das Testfeldarchitekturkonzept aus Softwaresystemsicht werden in den folgenden Abschnitten beleuchtet. Dazu wird Softwarearchitektur als notwendige und determinierende Voraussetzung für eine komplexe Systemlandschaft in Abschnitt 2.7 erläutert und adäquate Architekturkonzepte für verteilte Systeme werden detaillierter betrachtet (Abschnitte 2.8 u. 2.9). Um eine Systemarchitektur umfassend zu beschreiben, sind eine Charakterisierung dieser aus verschiedenen Sichten in Abschnitt 2.10 und die dazu notwendigen Architekturbeschreibungssprachen in Abschnitt 2.11 dargestellt. Der folgende Abschnitt 2.12 bereitet die technischen Grundlagen zur funktionalen Zerlegung des Systems und späteren Synthese vor. Die Grundlagen zur Bewertung von Architekturen in Kapitel 4 werden im Abschnitt 2.13 ausgeführt.

2.1 Einleitung

Es ist zwingend notwendig, klar zwischen dem zu testenden System und der eigentlichen Testinfrastruktur zu unterscheiden. Dies ist insbesondere wichtig, da das zu testende System

die Anforderungen an die Testumgebung maßgeblich bestimmt. Besondere Relevanz hat hierbei sowohl die Funktion des Systems, wodurch für den Testprozess die notwendige Datenerfassung bestimmt wird, als auch der technische Aufbau des zu testenden Systems welcher die korrespondierenden Schnittstellen determiniert.

ITS bildet den Überbegriff für eine ganze Reihe technischer Konzepte, die sich auf modernen Kommunikations- und Telematikmöglichkeiten in der Verkehrsdomäne stützen. Ein weitreichendes Verständnis hat dabei das Europäische Institut für Telekommunikationsnormen (ETSI) geprägt. Intelligente Verkehrssysteme nach ETSI sind Konzepte zur Verbesserung von Sicherheit, Effizienz und Nachhaltigkeit auf Grundlage der Vernetzung von beliebigen Verkehrsteilnehmern untereinander als auch mit stationärer Infrastruktur [ETS10a, S. 77]. Dabei sind sowohl straßengebundene Fahrzeuge als auch Schienenfahrzeuge, Flugzeuge und Schiffe in dieser Betrachtung berücksichtigt.[1]

Die Bandbreite der möglichen Anwendungen reicht von Mautsystemen bis zur multimodalen Reiseplanung, die verschiedene Verkehrsträger überspannt. Abbildung 2.1 verdeutlicht das ITS-Verständnis von ETSI und zeigt mögliche Vernetzungen von Verkehrsteilnehmern und daraus abgeleitete repräsentative Anwendungsbereiche.

Die weitere Betrachtung in dieser Arbeit beschränkt sich auf straßengebundene Verkehrssysteme, welche die Kommunikation von Fahrzeugen untereinander und die Kommunikation von Fahrzeugen mit stationärer Infrastruktur nutzen.

Die Palette der möglichen Funktionalität in diesem Bereich ist breit gefächert. Sie reicht von unterschwelligen Assistenzen, die vom Fahrer kaum wahrgenommen werden, z. B. ein automatisches Softwareupdate für das Motormanagement, das ohne Eingriff vom Fahrer per Fernwartung auf die Steuerungseinheit im Fahrzeug eingespielt wird, bis zu komplexen und sicherheitskritischen Automationen, die aktiv in die Quer- oder Längsführung des Fahrzeugs eingreifen können. Beispielsweise kann ein Lichtsignalassistent in Verbindung mit einem ACC-System die Fahrtgeschwindigkeit vor einer Rotphase ohne Fahrereingriff absenken, um dann das Lichtsignal möglichst erst bei einem Phasenwechsel zu erreichen, der eine Einfahrt in die Kreuzung erlaubt.

Um möglichst anspruchsvolle Funktionen im Testfeld zu unterstützen, werden insbesondere

[1] s. u.a. in [Arn09, S. 2], [ETS09, S. 11, 17ff] u. [ETS10b, S. 12]

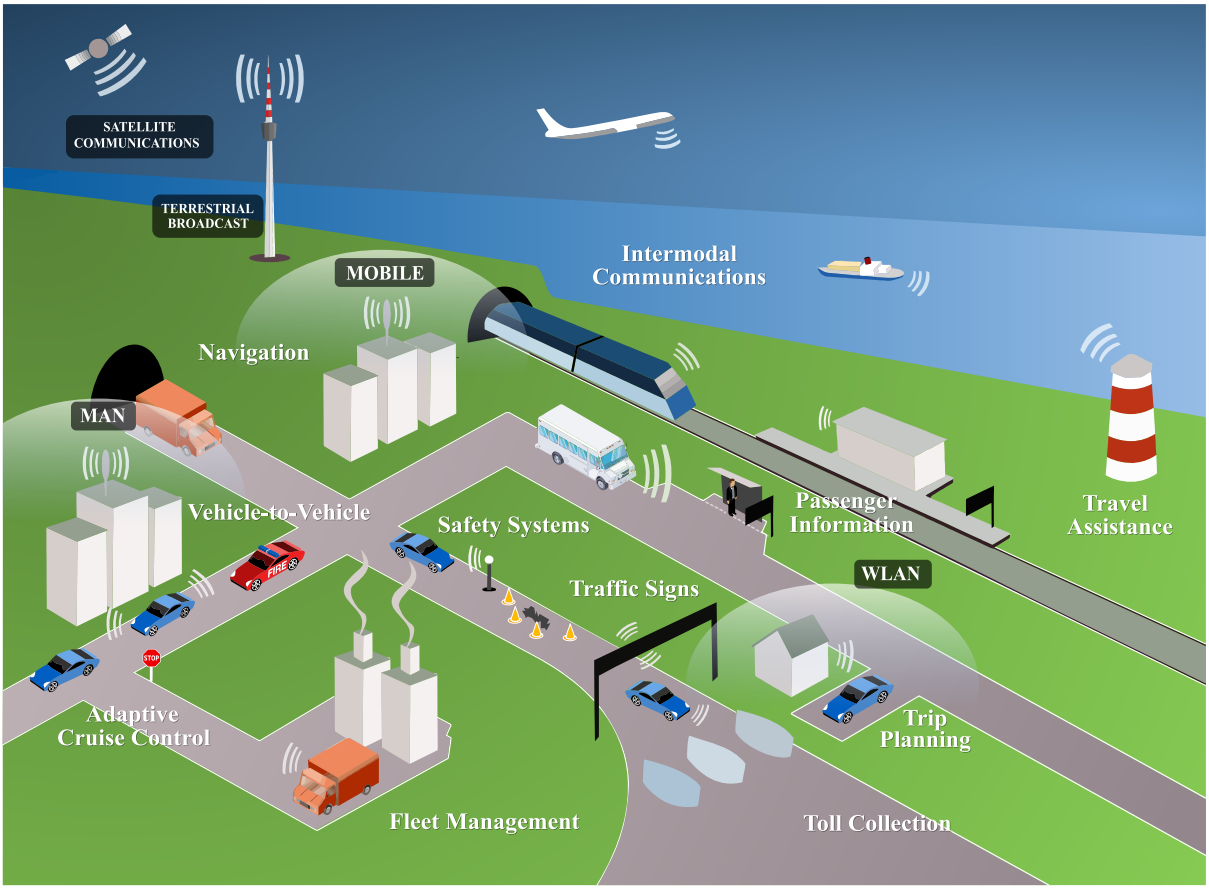

Abbildung 2.1: ITS-Landschaft nach ETSI, aus [Arn09]

kooperative Testsysteme betrachtet. Diese Systeme verwenden unterschiedliche Technologien (wie bspw. 5G[2], 3GPP Long Term Evolution (LTE)[3], Wi-Fi[4] oder Bluetooth[5]), um eine Kommunikation mit anderen Fahrzeugen und ortsgebundener Infrastruktur herzustellen.

Die Normungsorganisation ETSI identifiziert in ETSI TR 102 638 [ETS09] wesentliche Anwendungsklassen, wie bspw. Sicherheit und Effizienz. Diese werden weiter gegliedert in Applikationen (z.B. *Driving assistance – Co-operative awareness*) und präzisieren einige typische relevante Anwendungsfälle. Tabelle A.3 (im Anhang, S. 177) fasst die Einteilung zusammen.

Ein Beispiel für eine kooperative Assistenz kann die in [ETS09] beschriebene *Emergency*

[2]Mobilfunkstandard der fünften Generation, ermöglicht eine vergleichsweise schnelle Datenverbindung
[3]Mobilfunkstandard der vierten Generation
[4]Sammelbegriff für eine Reihe drahtloser Netzwerkkommunikationsverfahren, die auf IEEE 802.11-Standards basieren.
[5]drahtloses Nahbereichskommunikationsverfahren

vehicle warning-Funktion[6] sein. Hierbei sendet ein entsprechendes Einsatzfahrzeug mit Vorrang gegenüber dem restlichen Verkehr seine Position und Fahrtrichtung in den umgebenden Verkehr. Fahrzeuge, die sich in Fahrtrichtung des Einsatzfahrzeugs befinden, können dies empfangen und aufgrund der eigenen Position den Fahrer über das sich nähernde Einsatzfahrzeug informieren. Darüber hinaus kann die vom Einsatzfahrzeug ausgesendete Position von Lichtsignalanlagen ausgewertet werden. Die ermittelte Fahrtrichtung, aus der sich das bevorrechtigte Fahrzeug annähert, kann verwendet werden, um entsprechende Lichtsignale auf *Grün* (Einfahrt erlaubt) und alle andere Fahrtrichtungen auf *Rot* (Einfahrt verboten) umzustellen. So wäre gewährleistet, dass Kreuzungen mit Lichtsignalanlagen auf der Route des bevorrechtigten Fahrzeugs möglichst zügig und mit vermindertem Risiko passiert werden können. Abbildung 2.2 illustriert die Kommunikationswege zur ortsgebundenen Empfangseinheit (RSU, ❶) und zu den Einrüstungen in den Fahrzeugen (OBU, ❸) sowie die Verknüpfung mit der Lichtsignalanlagensteuerung (❷). Damit dieser Anwendungsfall verlässlich in der Praxis funktioniert, bedarf es weiterer

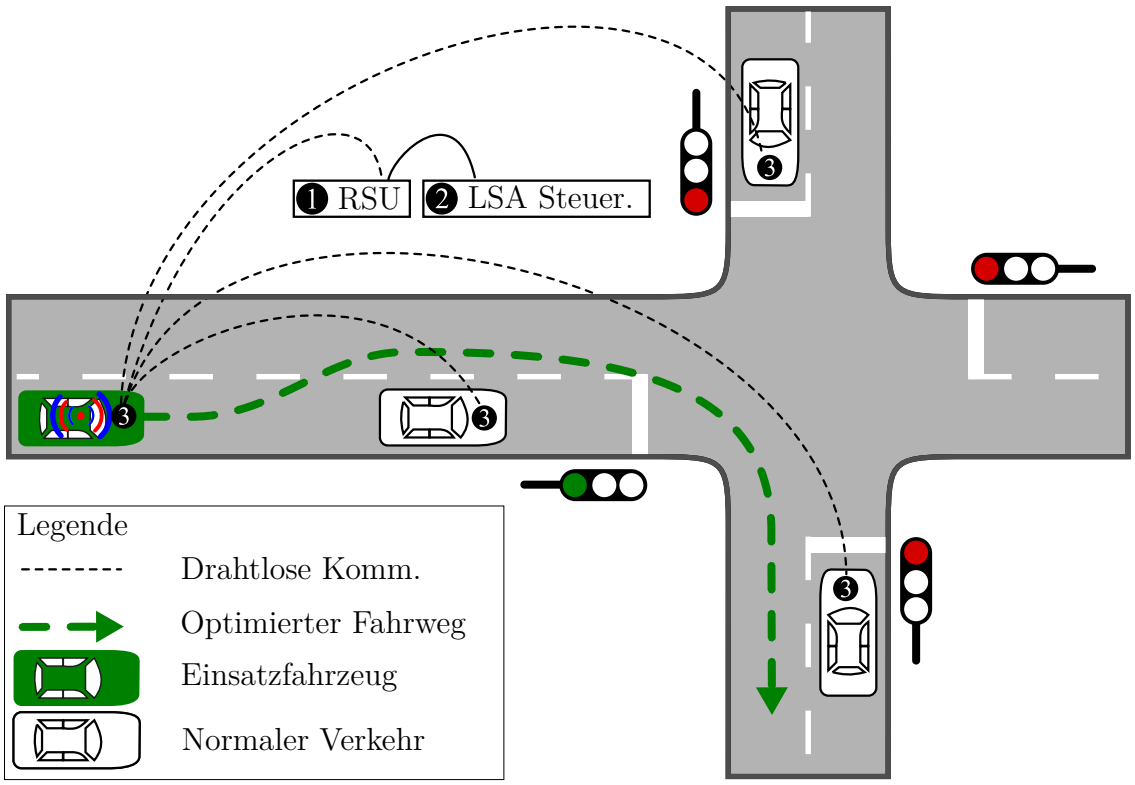

Abbildung 2.2: *Emergency vehicle warning*, angelehnt an [ETS09, S. 28]

Voraussetzungen. So sind bspw. folgende charakterisierende Anforderungen denkbar:

[6] siehe *C.1.2.1 Emergency vehicle warning* in Tab. A.3 u. [ETS09, S. 28]

- Die Fahrzeuge müssen stets Position und Fahrtrichtung hinreichend genau bestimmen können.
- Fahrzeuge und Infrastruktur müssen auf einem gemeinsamen Kommunikationskanal senden (Einsatzfahrzeug) und empfangen (umgebender Verkehr, Lichtsignalanlage).
- Es muss im Fahrzeug eine geeignete Instrumentation für eine Warnung geben.
- Die Frequenz (Anzahl der abgesetzten Positions- und Richtungsinformationen) sowie die Latenzen der Kommunikationsnachrichten sind ggf. sicherheitskritisch.
- Die Lichtsignalanlage muss auf entsprechende Fälle vorbereitet sein und das laufende Wechselzeichenprogramm möglichst ad hoc für die Sonderschaltung verlassen können.
- Die Authentizität der gesendeten Positions- und Richtungsinformationen eines Einsatzfahrzeugs ist sicherzustellen, um Missbrauch durch Dritte auszuschließen.

An diesem stark vereinfachten Beispiel[7] zeigen sich typische funktionale Kernkomponenten und Prozesse, die in einem Testfeld unter realistischen Bedingungen verifiziert werden können.
Im Folgenden werden diese Aspekte genauer untersucht. Dazu werden im nächsten Abschnitt generelle Komponenten eines solchen ITS-Anwendungsfalls tiefergehend untersucht und typische elementare Bausteine herausgearbeitet. Der folgende Abschnitt befasst sich mit den korrespondierenden Aufgaben, die ein Testfeld in diesem Kontext übernehmen kann.

2.2 Testsystem

Wie zuvor ausgeführt, ist eine klare Trennung zwischen dem zu testenden System und der eigentlichen Testinfrastruktur notwendig. Die folgende Ausführung fokussiert das Testsystem als zentralen Untersuchungsgegenstand im Testfeld und stellt dessen wesentliche Merkmale dar.
Im Rahmen dieser Arbeit wird das Testsystem als ein Anwendungsfall der in Abb. 2.1 exemplarisch dargestellten ITS-Landschaft verstanden. Dies sind insbesondere solche

[7] Eine umfassendere Betrachtung solcher Funktionen ist bspw. im Projekt SIRENE [RNT18] (Laufzeit 2017-2020) durchgeführt worden.

ITS-Anwendungen, die auf Kommunikation mit straßenseitiger Infrastruktur oder mobilen Einheiten basieren. Die besondere Relevanz solcher ist in Kapitel 1 dargestellt. Zudem ist diese Klasse der Anwendungen aus Testfeldsicht anspruchsvoll genug, um ebenfalls Anwendungen mit weniger herausfordernder Kommunikation bzw. Funktion mit abzudecken.

Aus technischer Sicht ist das Testsystem die Gesamtheit aller technischen Komponenten, die zur Darstellung einer bestimmten Assistenz- bzw. Automationsfunktion zum Zweck des Tests notwendig sind. Typischerweise sind dies jene technischen Bausteine, die in Produktionsreife die gewünschte Funktionalität im Verkehrssystem darstellen. Dazu gehören auch die genutzten Kommunikationskanäle.

Um die technischen Komponenten als typische Bausteine eines Testsystems zu generalisieren, kann auf Initiativen wie bspw. KAREN[8], CVIS[9] und DITCM[10] zurückgegriffen werden, die umfassend die technischen Grundlagen kooperativer Systeme analysieren und beschreiben. Darüber hinaus bestehen einige Standards, die als Leitfaden für das technische Grundgerüst von ITS-Applikationen herangezogen werden können. Besondere Sichtbarkeit auf europäischer Ebene hat die ETSI-Architektur ([Arn09], [ETS10b]), die in einigen Projekten[11] aufgegriffen wurde. Die Plattform des CVIS-Projektes und auch die ETSI-Architektur finden in relevanten Fachgremien Beachtung, bspw. in [B+09].

Aus der Betrachtung von Gemeinsamkeiten der Systementwürfe dieser Quellen lassen sich drei wesentliche technische Kernkomponenten isolieren:

OBU Die On-Board Unit (OBU) besteht aus einem im Fahrzeug verbauten Rechner [vOB+15, S. 6f]. Oft sind ein Bedien- und Anzeigeinstrument, eine drahtlose Kommunikationsschnittstelle und ein Interface zur Fahrzeugsensorik Teil der OBU-Peripherie. In speziellen Anwendungen sind auch aktive Eingriffe über die Fahrzeugaktorik realisiert. Bei Prototypen sind dazu oft noch Mechanismen vorgesehen, die auch nicht funktionsrelevante Daten erfassen und ggf. zur späteren Auswertung lokal vorhalten [PNW+13]. So wird eine spätere Fehlersuche erleichtert und die korrekte Funktion des Systems wird ebenfalls dokumentiert [KLGFK16, S. 2203f].

[8] KAREN (Keystone Architecture Required for European Networks), EU-Projekt von 1998 bis 2000, definierte eine anwendungsfallunabhängige ITS-Architektur, s. [BAG+00].

[9] Cooperative Vehicle Infrastructure Systems (CVIS), EU-Projekt von 2006 bis 2010, definiert und entwickelt u.a. technische Grundlagen der Kommunikation und Systemlandschaft kooperativer Systeme, s. [FOF+10].

[10] niederländische ITS-Referenzarchitektur in [PvvP16] und Testsite mit längerfristiger Ausrichtung [PNW+13], vgl. Abschnitt 2.5 und 2.6

[11] bspw. Pre-Drive C2X [BP09] und iTETRIS [RMK+13]

Mobile Endgeräte, wie bspw. Smartphones, können in entsprechenden Testszenarien, bspw. im Rahmen einer Reiseassistenz, eine besondere Rolle spielen. Dann sind sie, vergleichbar mit einer OBU, auf Basis der integrierten Anzeige, Sensorik und Kommunikationskanäle in einer ähnlichen Rolle [THLK12].

RSU Die Road-Side Unit (RSU) besteht aus einem Rechnersystem, das, vor Ort installiert, in die aktive Beeinflussung des Verkehrs eingebunden werden kann (z. B. durch die Verknüpfung mit Steuergeräten der Lichtsignalanlage), die Sensorik ausliest oder auch einen Zugriffspunkt für drahtlose Kommunikation darstellt. Daher wird die RSU typischerweise an Lichtsignalanlagen, Wechseltextanzeigen oder Verkehrserfassungssensorik ausgerichtet. Die eigentliche Steuerung bzw. Aufnahme der erfassten Daten wird oft zentral in einem Rechenzentrum vorgenommen. Daher ist eine RSU üblicherweise mit diesem vernetzt. In der Praxis ist dies zumeist eine Internetverbindung, bspw. [FSK12, S. 2], [HMF+11, S. 51]. Siehe auch ❶ in Abb. 2.2.

Zentraleinheit Typischerweise gibt es ein (zentrales) Rechnersystem mit mehreren Aufgaben. Die Kernaufgabe ist die funktionale Unterstützung der RSU und OBU sowie die Orchestrierung der involvierten Komponenten. Ferner stellt die Zentraleinheit eine Vernetzung zu externen Datenquellen und -senken dar. So kann bspw. ein bestehendes Verkehrserfassungssystem mit aktuellen Verkehrsflussdaten bedient werden. Zusätzlich ist die Anbindung externer Funktionalität (bspw. verfügbarer Parkraum) realisierbar. Oft sind administrative Aufgaben im Netzwerk der RSU und OBU von der Zentraleinheit aus initiiert.

Für einige spezielle Funktion aus dem Katalog ETSI TR 102 638 sind Komponenten optional. Für die *Slow vehicle warning* kann bspw. auf die ortsgebundene Infrastruktur verzichtet werden. Der oben beschriebene Anwendungsfall (*Emergency vehicle warning*) benötigt jedoch mobile und ortsgebundene Infrastruktur.

In einem Testsystem, insbesondere unter Betrachtung kooperativer Aspekte, ist Kommunikation ein elementarer Aspekt. Daher widmen sowohl CVIS[12] als auch der ETSI-Standard 102 638[13] anwendungsfallunabhängig eine spezielle kommunikationsorientierte Architektur mit versatilen Kommunikationskanälen und Routing. In diese ist die Laufzeitumgebung

[12] Communication Access for Land Mobiles (CALM), Unterprojekt von CVIS, vgl. [FOF+10]
[13] *Intelligent Transport Systems (ITS); Communications Architecture*, [ETS09]

der eigentlichen Anwendungen eingebettet. Im Testfeld können solche und ähnliche Architekturen eingesetzt werden, daher müssen deren Schnittstellen, Komponenten und Kommunikationskanäle hinreichend durch das Testfeld unterstützt werden.

2.3 Testfeld

Im Gegensatz zum oben umrissenen Testsystem ist das Testfeld eine Infrastruktur, die ein Testsystem und seine Komponenten zu einem speziellen Zweck unterstützt und betreibt. Um diese Differenzierung genauer auszuführen, werden in diesem Abschnitt die Einsatzziele eines Testfeldes genauer dargestellt.

Entwicklung und Erprobung der Testsysteme können durch Verifizierung (*in vitro*), durch Simulation von gewünschten Szenarien (*in vivo*) oder unter realistischen Bedingungen in einem Testfeld (*in situ*) unterstützt werden. In der Praxis werden über die Laufzeit der Entwicklung des Testsystems verschiedene Methoden komplementierend verwendet. Hybride Erprobungen mit realen und virtuellen Komponenten machen den Übergang der Erprobungsarten fließend. Insbesondere bei der Entwicklung von technischen Grundkomponenten (bspw. Kommunikationshardware) wird das Testfeld zuweilen als Simulationsumgebung verstanden.

Im Rahmen dieser Arbeit werden Testfelder mit einem klaren In-situ-Bezug fokussiert, um eine einschränkende Sichtweise auf die Laborerprobung zu vermeiden. Ein Testfeld kann dabei sowohl abgegrenzte Areale, mit ausschließlichem Experimentalverkehr, als auch offene Gebiete, die vom öffentlichen Straßenverkehr durchdrungen sind, umfassen.

Testfelder dienen typischen Einsatzzwecken, diese werden im Folgenden kurz erläutert:

Entwicklung Kernaufgabe eines Testfeldes ist die Unterstützung der Entwicklung. Anschließend an erste Versuche unter Laborbedingungen können Teilaspekte der gewünschten Funktionalität im Feld erprobt werden. Typisch ist bspw. die Untersuchung von drahtlosen Kommunikationsmedien im Feld. Das Verhalten bspw. von WiFi nach IEEE 802.11p [IEE03] in verkehrlichen Situationen lässt sich modellieren und auch simulieren [BKRC10]. Dabei werden auch spezielle Fehlermodelle betrachtet [Rap96]. Dennoch sind in situ oft relevante Abweichungen beobachtbar [ZSO+05]. Darauf aufbauend können Teilfunktionen, ggf. auch mit emulierter Funktionalität, ins Feld geführt werden. Abschließend können sukzessiv Produktivkomponenten eingeführt werden, um die Gesamtfunktionalität darzustellen.

Erprobung Ist die gewünschte Funktionalität entwickelt und bis zu einem gewissen Reifegrad stabilisiert, kann die Anzahl der zu testenden Einheiten zunehmen. Dies dient der Erprobung von Skalierungseffekten und zeigt eine entsprechende Auslastung der erprobten Einheiten (Kommunikationskanäle, Prozessorlast, Verfügbarkeit). Zudem steigt so auch die Durchdringung des Verkehrs (Penetrationsrate). Insbesondere bei kooperativen Testsystemen ist dies notwendig, damit bspw. arbiträre Begegnungen zweier ausgerüsteter Fahrzeuge beobachtbar werden. Obschon eine möglichst hohe Dichte an ausgerüsteten Fahrzeugen angestrebt wird, bleibt dies oft ein geringer Anteil. So sind die in der Erprobung beobachteten Phänomene wichtige Indikatoren, um eine darauf aufbauende Verkehrssimulation zu modellieren. So können beispielsweise die gesamtverkehrlichen Auswirkungen mit unterschiedlichen Penetrationsraten auf dieser Grundlage simuliert werden [Bv11, S. 2]. Ein Field Operational Test (FOT) ist eine zeitlich und inhaltliche begrenzte Erprobungskampagne einer abgeschlossenen Klasse von Anwendungen unter In-situ-Bedingungen.

Referenzplattform Wie zuvor beschrieben, können im Testfeld provisorische Komponenten durch solche mit stabilerem Aufbau getauscht werden. Die Modularität ist auch hilfreich, um Komponenten mit ähnlicher Funktion zu vergleichen. Beispielsweise können verschiedene WiFi-Antennen unterschiedlicher Hersteller in Fahrzeugen verbaut werden. Sofern sich keine weiteren Abweichungen im Aufbau ergeben, lassen sich aus vergleichenden Beobachtungen Rückschlüsse auf die Nutzbarkeit der unterschiedlichen Antennen ziehen. So kann das Testfeld auch als Referenz für unterschiedliche Komponenten fungieren. Ebenso können Komponenten (bspw. Fahrzeuge) aus einem Testfeld in einem anderen eingesetzt werden. So kann untersucht werden, ob die verbauten Komponenten testfeldübergreifend harmonisieren und ein Mindestmaß an Interoperabilität dargestellt wird. Referenz ist im Testfeld auch durch Redundanz gegeben. Um bei Entwicklung und Erprobung die Güte der eingesetzten Komponenten bewerten zu können, sind diese im gleichzeitigen Einsatz mit höherwertigen. Typisch ist dies bspw. bei Kommunikationskanälen: Soll eine RSU in der Serienproduktion aus Kostengründen nur drahtlos kommunizieren, kann im Testfeld eine zusätzliche kabelgebundene Vernetzung installiert sein. So kann die potenziell störanfälligere Drahtlosverbindung über die zuverlässige Kabelverbindung kontinuierlich überwacht werden.

Durch die Vielzahl der unterschiedlichen Komponenten ist selbst auf Grundlage generalisierter Konzepte die Festlegung eigener Schnittstellen, Prozesse und Proto-

kolle notwendig. Damit die einfache Verknüpfung von Komponenten auch außerhalb des Testfeldes besteht, ist eine Standardisierung solcher Eigenschaften evident. Die Bemühungen, bspw. von ETSI [ETS09], zielen auf eine Standardisierung der Systemlandschaft.

Zertifizierung Soll mit der Entwicklung von neuen Assistenz- und Automationsfunktionen Serienreife erzielt werden, so ist im Rahmen der Konformitätsbewertung oft auch eine gewisse Zertifizierung solcher Funktionen zur Zulassung im Fahrzeug sinnvoll [BVJ+11]. Dazu muss ein Testfeld in der Lage sein, bei der Prüfung der verschiedenen funktionalen und nicht-funktionalen Anforderungen zu unterstützen. Erprobung in situ weist ggf. nur Konformität in den erprobten und naturgemäß beschränkten Szenarien nach. Daher ist dies i. d. R. nicht für eine Zertifizierung hinreichend.[14]

NDS Im Gegensatz zur Entwicklung und Erprobung von Testsystemen zielen Naturalistic driving studies (NDS) auf die Beobachtung des Fahrers prinzipiell ohne Beeinflussung durch zusätzliche Systeme, um bspw. normatives Fahrerverhalten zu gewinnen oder kognitive Prozesse zu untersuchen. In einem Testfeld ist auch prinzipiell eine NDS durchführbar, da oft die Instrumentation bzw. Infrastruktur der Versuchsträger für die gewünschte Beobachtung bzw. Datenerfassung vergleichbar genutzt werden kann. Dies wird u. a. genutzt, um das Verhalten eines Verkehrsteilnehmers mit und ohne Erprobungssystem vergleichen zu können.

Darüber hinaus können NDS das normale bzw. erwartete Verhalten in einer Fahrsituation erheben. Dies kann wichtige Hinweise innerhalb der Anforderungsanalyse von Anwendungen und Assistenzen liefern, sowie helfen, deren Verhalten auf Normalwerte zu parametrisieren. So lassen sich Unterstützungssysteme entwerfen und justieren, die für einen normalen Fahrer subjektiv nachvollziehbar sind und natürlich agieren.

Für weitergehende Fragestellungen, wie bspw. Human-Machine Interaction (HMI), benötigen NDS typischerweise dedizierte Instrumentation, um die Handlungen des Fahrers genauer zu dokumentieren. So genügt es vielleicht, im Rahmen einer Erprobung zu ermitteln, ob ein Fahrer ein Assistenzsystem verwendet oder nicht. Im Kontext von HMI-Untersuchungen könnte hingegen versucht werden, mittels Eye-

[14] Da Zertifizierung in der In-situ-Erprobung nur eine untergeordnete Rolle spielt, sind weiterführende Überlegungen zu Zertifizierung und Akkreditierung von Testinfrastruktur nicht Teil dieser Arbeit.

Tracker, einem Instrument zur Bestimmung des Blickwinkels, die Zeitspanne abzumessen, die der Fahrer für die Bedienung des Systems benötigt. Die Instrumentation ist oft aufwendig und abgesehen von speziellen Versuchsaufbauten[15] nicht in der benötigten Breite vorhanden. Die Herausforderungen von NDS mit entsprechender Einrüstung zur genauen Beobachtung des Fahrerverhaltens werden im Rahmen dieser Arbeit nur partiell abgedeckt. Die illustrierten Konzepte lassen sich für den Datenerfassungsaspekt in solchen Studien übertragen, wie später in Abschnitt 3.3 ausgeführt.

Analyseplattform Neben der in-situ-Erprobung von Testsystemen bietet sich auch eine wissenschaftliche Auswertung der erfassten Daten an. So sind als Untersuchungsgegenstand nicht nur Einflüsse durch das zu testende System denkbar, sondern bspw. auch artifizielle Störungen darstellbar. Im Brückenschlag zu Untersuchungen über NDS lassen sich so Handlungsmuster und typisches Verhalten in solchen Situationen identifizieren. Besondere Relevanz hat dieser Aspekt u.a. bei der Betrachtung von Transitionen bei automatisierter Quer- und Längsführung. Hier ist die Übernahme der Kontrolle durch den Fahrer besonders kritisch [SF08]. Letztendlich lassen sich aus den so gewonnenen Erkenntnissen wertvolle Hinweise für innovative Weiterentwicklungen ableiten [FGS+15, S. 190].

Demonstration Die im Testfeld aufgebauten Einheiten können auch zur Demonstration der Funktionalität gegenüber Interessennehmern, Kunden und Auftraggebern verwendet werden. Die Nähe zur Produktionsreife ist hier besonders vorteilhaft zur Illustration der zugrunde liegenden Funktion im Feld.

Die hier beschriebenen Verwendungszecke sind in der Praxis keine trennscharfen Einzelaspekte. Üblicherweise verschmelzen mehrere Aspekte in einer Kampagne. So kann bspw. der Erprobung eines Assistenzsystems eine gewisse Entwicklungsphase vorausgehen, welche die gewünschte Funktionalität bereitstellt. Sind im Feldversuch genügend Daten erhoben worden, um die eigentliche Auswertung durchzuführen, liegt es nahe, diese Daten nicht nur zur Beantwortung der ursprünglichen Untersuchungsfrage heranzuziehen, sondern auch im Sinne einer NDS entkoppelte Fragestellung zu adressieren. Werden bspw. in einer Erprobungskampagne einer Kreuzungsassistenz mit einer frei fahrenden Flotte Daten eingefahren, können diese ggf. ebenso zur Betrachtung kritischer Fahrsituationen außerhalb von Kreuzungen dienen.

[15] vgl. Instrumentation 100-Car-Study in [NDK+05]

Bemerkenswert ist zudem der Aspekt, dass Teilfunktionen des Testsystems durch Konzepte wie Simulation, Emulation vom Testfeld bereitgestellt werden können. Durch die Nachbildung von entsprechendem Verhalten fallen solche System in die Klasse von *Simulator Problemen*[16]. Diese Klasse ist gekennzeichnet durch eine Reihe von Herausforderungen, wie u. a. Simulationstreue[17], als Maß der Nachbildung eines inhärentes Verhalten eines dynamischen (realistischen) Originals und Simulationsvalidität, als Homomorphismus zwischen Anforderungen und dem Simulationsverhalten[18]. Dies hat klare Implikationen für den Aufbau des Testfelds und daher werden entsprechende Überlegungen in den entsprechenden Abschnitten (Kapitel 3 und 4) ausgeführt.
Um eine gute Unterstützung durch ein Testfeld in diesen Aspekten zu erzielen, ist eine genauere Betrachtung der Prozesse notwendig. Der folgende Abschnitt beleuchtet die Prozesse insbesondere zur Entwicklung und Erprobung von ITS-Applikationen genauer.

2.4 Testprozess

Die Durchführung von Kampagnen in einem Testfeld ist ein nicht trivialer Vorgang. Daher gibt es in dieser Domäne einige strukturierende und organisierende Ansätze, die auf zwei Betrachtungsebenen diese Problemstellung adressieren. Zum einen ist der Ablauf des Gesamtprozesses umfassend von der initiierenden Forschungsfrage bis zur abschließenden Auswertung sozioökonomischer Effekte zu organisieren, zum anderen bedarf es feinschrittiger Prozesse für die Koordination der technischen Einheiten auf der Ebene von isolierten Testläufen.
Besondere Relevanz hat hier das FESTA-Handbuch [FES17], das einigen Projekten als Leitfaden für die Organisation und praktische Durchführung dient. Beispielsweise nutzt das EU-Projekt DRIVE C2X einen auf FESTA aufbauenden Leitfaden [MBM+11] mit erweiterter Ausrichtung auf kooperative Systeme und testfeldübergreifende Erprobung, s. [SFT+11] u. [GNH+10]. Fokus des FESTA-Handbuchs ist die wissenschaftliche Nutzung eines Testfeldes als Werkzeug im Rahmen der Verifikation bzw. Falsifikation einer abgesteckten Forschungsfrage. Daher sind technische Rollen des Testfelds wie Erprobung und Entwicklung nur nebenläufig berücksichtigt. Dennoch kann das in FESTA definierte V-Modell (s. Abb. 2.3) durch eine Umgewichtung der Prozessschritte für technisch

[16] vgl. *simulator problem* in [BC03]
[17] vgl. *Simulation fidelity* in [LMV09, S. 62]
[18] vgl. [Sta86, S. 177ff]

ausgerichtete Kampagnen in einem Testfeld nützlich sein. So beschreibt u. a. [MBM+11] eine solche Adaption und die notwendigen Ergänzungen für die testfeldübergreifende Erprobung kooperativer Funktionen.

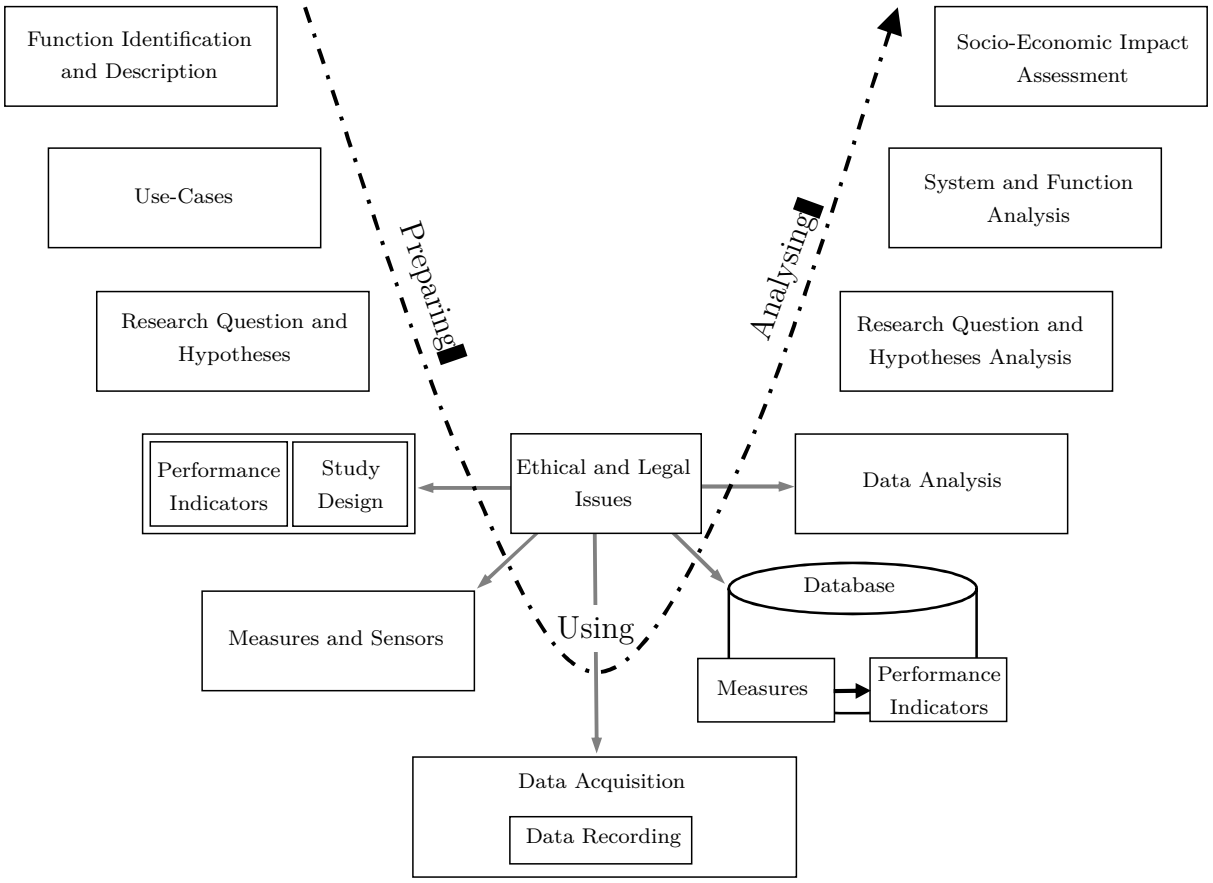

Abbildung 2.3: V-Modell eines Testprozesses nach [FES17]

Die Organisation solcher Kampagnen ist in [FES17] abstrahiert beschrieben[19]. Der Gesamtprozess des V-Modells aus [FES17] unterscheidet diese Schritte:

Function Identification and Description Zielsetzung eines FOT ist die In-situ-Auswertung von Assistenz- und Automationsfunktionen. Diese zielt sowohl auf die technische Verifikation als auch auf weiterführenden Untersuchungen bspw. hinsichtlich Sicherheit, Effizienz und Akzeptanz des Testsystems. Beide Fragestellungen müssen vor der Untersuchung präzisiert und die zu testenden Funktionen genau festgelegt werden.

Use Cases Auf Grundlage der ausgewählten Funktionen können nun bestimmte An-

[19] vgl. *Experimental procedures* in [FES17, S. 72]

wendungsfälle (*Use Cases*) gewählt werden, unter denen bspw. besonders kritische Funktionen Verwendung finden oder untersuchte Effekte besonders deutlich beobachtbar zu sein scheinen.

Research Questions and Hypotheses Vor der eigentlichen Untersuchung sollen die technischen oder wissenschaftlichen Fragestellungen festgelegt werden. So können entsprechende Hypothesen über die Auswirkungen des Systems auf Fahrer, Verkehr und Umwelt formuliert werden.

Performance Indicators Die Verifikation bzw. Bewertung muss an messbaren Kriterien wie bspw. Kennzahlen erfolgen. In diesem Prozessschritt werden adäquate Maßzahlen und Erhebungsverfahren für den Feldtest festgelegt.

Study Design Die Methodik der Versuchsdurchführung, passende Anforderungen an Probanden und den eigentlichen Versuchsaufbau müssen definiert werden.

Measures and Sensors Die Kennzahlen (*Performance Indicators*) müssen mit den im Versuchsaufbau vorhandenen Sensoren erfassbar sein. In diesem Schritt wird die Gewinnung der Kennzahlen aus den erhobenen Daten beschrieben und die notwendige Instrumentation im Versuchsaufbau sichergestellt.

Data Acquisition Die eigentliche Versuchsdurchführung als einzelner Versuchslauf oder auch als Kampagne wird in diesem Schritt durchgeführt. Dabei werden die erhobenen Daten aufgezeichnet (*recording*) und vorverarbeitet.

Database Nach Versuchsdurchführung werden die Daten für die Analyse vorbereitet. Dazu werden insbesondere aus den erfassten Daten (*Measures*) die vorab definierten Kennzahlen (*Performance Indicators*) bestimmt. Ziel dieses Schrittes ist, den analysierenden Zugriff auf die so vorbereiteten Daten zu gewährleisten.

Data Analysis Die Datenanalyse bereitet die nachfolgenden drei Schritte vor, indem auf Grundlage der erhobenen Daten Kennzahlen ermittelt werden.

Research Question and Hypotheses Analysis Auf Grundlage der durch Datenanalyse gewonnenen Kennzahlen kann die ursprüngliche Forschungsfrage beantwortet werden.

System and Function Analysis In diesem Schritt wird die Verifikation des Systems bzw. dessen getesteter Funktionen vorgenommen.

Socio-Economic Impact Assessment Auf Grundlage der Systembeobachtung im Feldtest kann abgeschätzt werden, wie sich das Systems mit unterschiedlichen Penetrationsraten auf die Umwelt bzw. auf den Gesamtverkehr auswirkt. Ebenso kann bspw. im Rahmen einer Kosten-Nutzen-Rechnung betrachtet werden, ob ein solches System wirtschaftlich erfolgreich zu sein scheint.

Feinschrittiger ist das Vorgehen in [BVJ+11] ausgeführt. Das dort skizzierte Vorgehen beschreibt grundsätzliche Prozessschritte iterativer Erprobung von Testsystemen. Ein Testfeld muss als Erprobungswerkzeug nahtlos in den globalen Erprobungsprozess eingebettet werden und die detaillierten Iterationsschritte technisch und organisatorisch unterstützen. In diesem Kontext zeigt [Bv11, S. 3f] einen typischer Prozess zur Bewertung von ITS im Testfeld. Als Grundlage werden die Überlegungen von FESTA aufgegriffen und mit einem Unterprozess zur verkehrlichen Modellierung flankiert. Dies verfeinert den oben dargestellten Analyseschritt (Data Analysis).
FESTA liefert ebenfalls einen FOT Implementation Plan (FOTIP) in [FES17, S. 179]. In diesem sind u.a. die Aktivitäten beschrieben, die typischerweise zur Durchführung eines FOT koordiniert ablaufen müssen. Die dort formulierten Aktivitäten sind nicht als allgemeingültige und standardisierte Referenz zu verstehen. Dennoch ist bietet FOTIP eine maßgebliche Richtschnur für die Planung der Aktivitäten. Abbildung 2.4 zeigt 22 Hauptaktivitäten des FOTIP und ihre chronologische Verzahnung im Gesamtablauf. Darüber hinaus werden in [FES17, S. 179ff] die oben dargestellten Aktivitäten in detailliertere Tätigkeiten aufgegliedert und ihnen verantwortliche Teams/Rollen zugewiesen.

2.5 Bestehende Aktivitäten zu Testfeldern

In den vorangehenden Abschnitten wurden Testsystem und Testfeld genauer beschrieben. So beruht bspw. die in Abschnitt 2.2 dargestellte Komponententeilung (RSU, OBU etc.) eines Testsystems auf der Beobachtung entsprechender Aufbauten (bspw. [Pv16]). Aus bestehenden Aktivitäten kann der Stand der Technik abgeleitet werden. Insbesondere die Gestaltung der Architektur in diesen Konzepten und Aufbauten ist im Rahmen dieser Arbeit von Interesse. Dazu werden in diesem Abschnitt bestehende Aktivitäten im Bereich von Testfeldern auf Relevanz untersucht und so eine detailliertere Betrachtung der Architektur und entsprechender Designentscheidungen ausgewählter Aktivitäten in Abschnitt 2.6 vorbereitet.
Insbesondere sind hier versatile Testfelder interessant, die mehrfach zu Forschungs- und

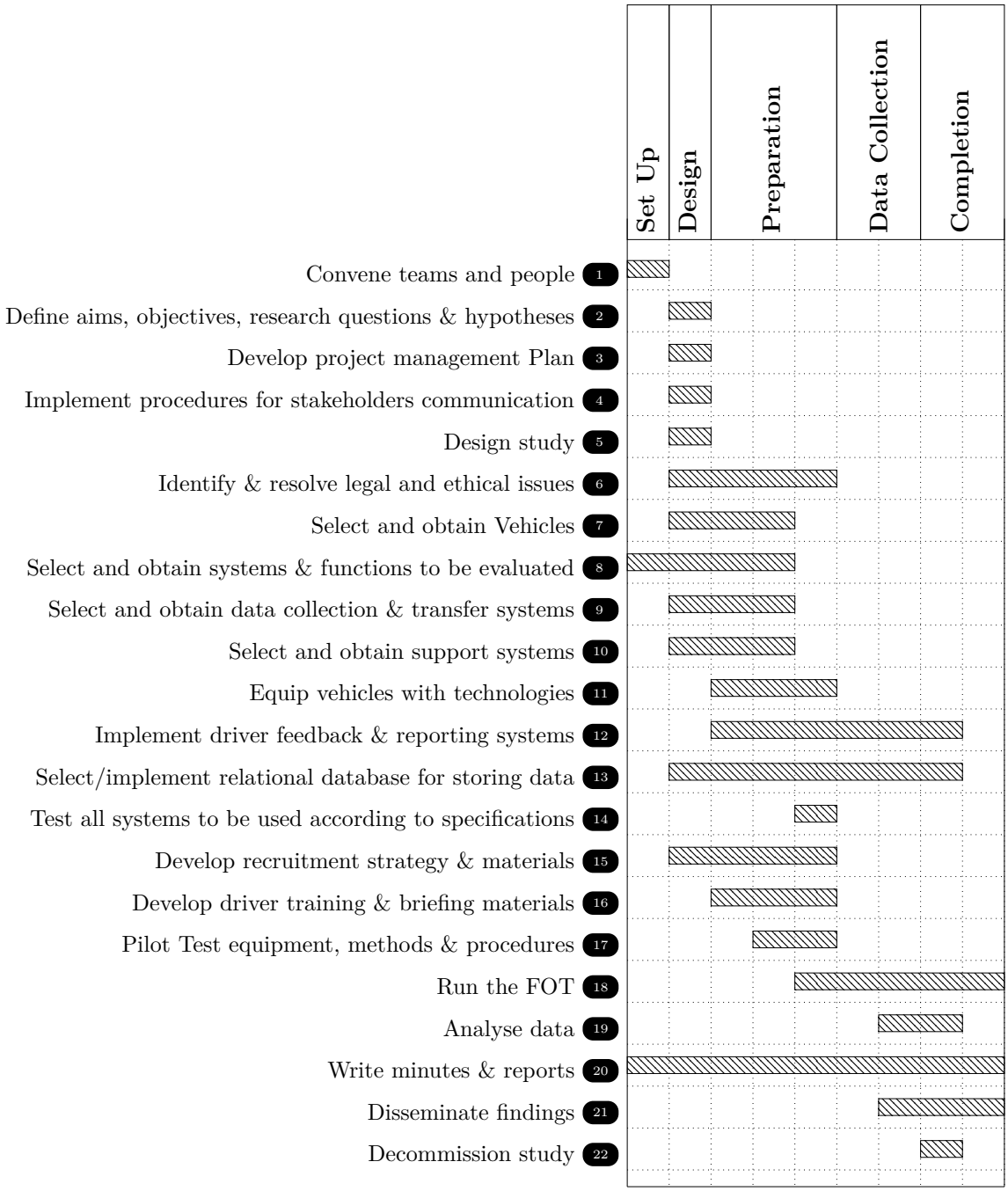

Abbildung 2.4: 22 Aktivitäten im *FOT Implementation Plan* aus [FES17, S. 175ff]

Erprobungszwecken eingesetzt werden. Eine gute Übersicht zu den relevanten Aufbauten liefert bspw. das FOT-Net Wiki[20]. Einen groben Überblick über die dort aufgeführten Aktivitäten zeigt die Auflistung A.2 (im Anhang, S. 174). Zur besseren Bewertung der zeitlichen Abfolge ist in Abb. A.1 (im Anhang, S. 176) die zeitliche Verschränkung der dort genannten Aktivitäten dargestellt.

Hinderlich bei der eingrenzenden Auswahl ist die teils schlechte Verfügbarkeit von Informationen zu den einzelnen Aufbauten. Somit ist nicht in allen Fällen eine klare Ableitung der unterliegenden Konzepte möglich. Bspw. verspricht [Tes09, S. 3] für die TSN eine große Anzahl von RSU und gibt keine weitere Konkretisierung über deren Aufbau an. Aus den mit TSN assoziierten Projekten, wie CVIS und CALM, können nur Indizien abgeleitet werden, dass entsprechende CVIS-Konzepte adaptiert wurden. Etwas genauer schlüsselt [Tes11b, S. 22] die Infrastrukturelemente an den Strecken der Test Site Sweden (TSS)[21] auf. Die stationären Einheiten der TSS sind offenbar mit dem CVIS-Logo beschriftet [Tes11a], genauer wird auch hier die zugrunde liegende Architekturadaption nicht dargelegt. Insofern lässt sich aus einigen Testfeldern wenig Erkenntnis über deren Architektur gewinnen. Insbesondere der Aufbau der stationären Infrastruktur sowie Konzepte der koordinierten Versuchsdurchführung sind daraus nicht zu gewinnen.

Über das Testfeld AIM, das sim$^{\mathrm{TD}}$-Projekt und die Vorhaben aus DITCM sind mehr Informationen verfügbar. Daher wird sich die weitere Betrachtung in Abschnitt 2.6 auf die Architekturüberlegungen dieser Felder stützen.

Die Dokumentation von Feldtests und Pilotversuchen mit kooperativen Anwendungen ist wenig zur Ableitung von Architekturentscheidungen geeignet. Grundsätzlich werden hier Anwendungen ins Feld geführt, die zur Projektlaufzeit in einem eng gesteckten Rahmen betrieben wurden. Durch den oft provisorischen Charakter dieser Aufbauten ist eine direkte Adaption von Konzepten für ein Testfeld nur bedingt sinnvoll. Das Projekt *ITS Test Beds* hat einige relevante Feldtests und Pilotversuche in genau diesem Kontext untersucht. [BVJ+11, S. 15] untersucht u.a. die Projekte COMeSafety, COOPERS, CVIS, GST, PRE-DRIVE C2X, PReVENT und SAFESPOT. Daraus wird die in [BVJ+11] dargestellte Architektur abgeleitet. Daher kann angenommen werden, dass die in der Auflistung A.2 aufgeführten Feldtests und Pilotversuche adäquat im Architekturkonzept von ITS Test Beds repräsentiert sind. Die Ergebnisse von ITS Test Beds sind u. a. in AIM aufgegriffen worden [VBK10, S. 9ff]. In anderen Fällen sind solche Beziehungen i. d. R.

[20] online unter [FOT11]

[21] Schwedisches Testfeld, seit 2006 in Betrieb, mit langfristiger Ausrichtung und verschiedenartigen Anwendungsfällen, vgl. ‚Swedish ecosystem' in [APHH16, S. 7ff]

nicht explizit dokumentiert. Aus der zeitlichen Verschränkung von ITS Test Beds, sim^{TD}, AIM und DITCM, wie in Abb. 2.5 dargestellt, kann dennoch grob die latente inhaltliche Beziehung der Konzepte taxiert werden.

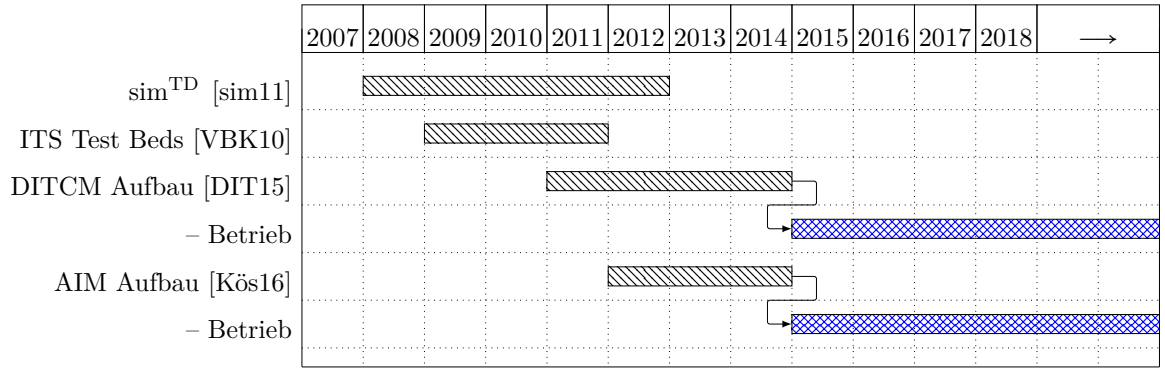

Abbildung 2.5: Projekt- bzw. Entwicklungszeitraum vom ITS Test Beds, sim^{TD}, AIM und DITCM

Bedingt durch den teils flüchtigen Charakter dieser Projekte und wandelnde Forschungsfragen besteht eine gewisse Dynamik, die hier nur als Schnappschuss eingefangen werden kann. Die im folgenden Abschnitt berücksichtigten Aktivitäten beruhen ebenfalls auf der Expertise aus vorangehenden Projekten. So kann diese begrenzte Betrachtung bestehender Aktivitäten dennoch hinreichend sein. Darüber hinaus ist eine Schlüsselanforderung an ein Testfeld eine gewisse Adaptierbarkeit bspw. an künftige Technologien. Diese Flexibilität sollte auch eine Adaption an bestehende Konzepte erlauben.

2.6 Bestehende Testfeldarchitekturen

In diesem Abschnitt werden bestehende Architekturkonzepte aus den Projekten ITS Test Beds, sim^{TD}, AIM sowie DITCM knapp dargestellt. Diese können beispielhaft für die Architekturüberlegungen in den folgenden Kapiteln sein. Dabei sind nicht nur die ausgeformten Architekturen mustergebend, sondern auch die detailliert analysierten Anforderungen an die jeweilige Testinfrastruktur der Projekte verfestigen die praktische Relevanz der angestrebten Architektur dieser Arbeit.

ITS Test Beds Das EU-Projekt[22] ITS Test Beds definiert in [BVJ+11] eine grobe Referenzarchitektur für Testfelder und beschreibt eine prototypische Umsetzung des

[22]finanziert durch die Europäische Union (EU), Laufzeit 2009 bis 2011

Konzepts in [Ver10]. Die Architektur aus Abb. 2.6 identifiziert wesentliche Architekturbausteine in vier Klassen: Die Handhabung von Infrastruktur (*Test Bed*), die zur Durchführung relevanten Komponenten und Prozesse (*Test*), die Handhabung der eigentlichen Technologien und die Datenaufzeichnung (*Technologies*) sowie die Auswertungsmechanismen (*Analysis*). In diesen Klassen sind Architekturbausteine zusammengefasst, die bestimmte Teilaspekte lösen.

Die Architektur gibt keine konkrete Vernetzung der Bausteine an. Diese, bzw. die darin enthaltenen Unterbausteine, sollen im Vorfeld der Versuchsdurchführung zu einem Testaufbau verschaltet werden. So lassen sich Soft- und Hardwarekomponenten zur Durchführung einer komplexen Kampagne in einem Feldversuch oder für eine NDS kombinieren und in Rekombination für weitere Versuchsaufbauten wiederverwenden. Um die lose definierten Komponenten in einem Testaufbau zu einem Testfeld zu

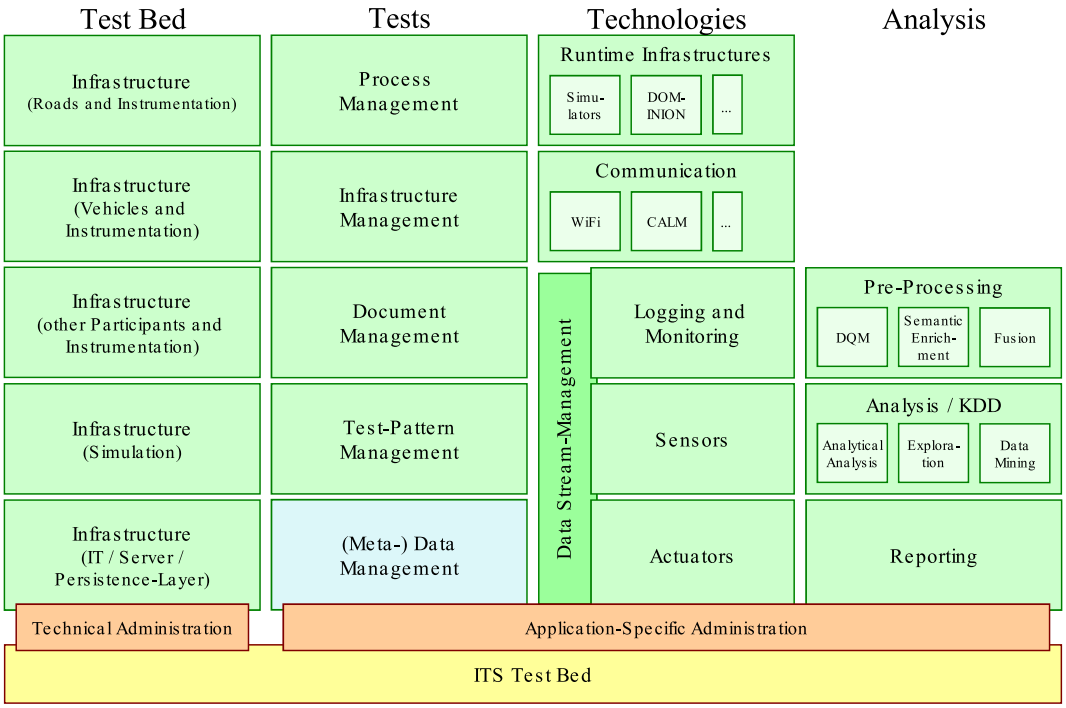

Abbildung 2.6: ITS Test Beds Referenzarchitektur aus [BVJ+11]

verschalten und als konkrete Instanz zu betreiben, bedarf es einer koordinierenden Komponente, die im Architekturbild (Abb. 2.6) selbst nicht direkt dargestellt ist. Diese komponiert die Bausteine bzw. deren Teilfunktionen und übernimmt deren Orchestrierung zur Laufzeit. ITS Test Beds nennt diese Funktionalität Test Service

Aggregation Center (TASC). Diese verbindende und koordinierende Komponente wurde im Projekt prototypisch implementiert und dient gleichzeitig als zentrale Datenhaltung im Testfeld.

Die Schwerpunkte der ITS Test Beds Architektur reflektieren einige unübliche Merkmale. Die dedizierte Handhabung von Infrastruktur in entsprechenden Datenbanken in der Kolumne *Test Bed* wird in kurzlebigeren Versuchsaufbauten und Feldtests oft nicht so berücksichtigt, da es sich um fixe Aufbauten handelt und kaum wechselnder Komponenten eingesetzt werden.

Die Datenaufzeichnung wird als zentrales Problem identifiziert und bspw. durch eine Meta-Datenbank, Datenstrommanagement und Datenvorverarbeitung flankiert. Methodisch weicht dies von üblichen Aufbauten ab, die aus den genannten Gründen eher einfache Mechanismen nutzen. Gängig ist ein einfaches Aufzeichnen der Daten wie in [FES17] vorgeschlagen. Weiterverarbeitung und Auswertung fallen dann in die Verantwortung der Analysierenden. Die Architektur sieht hier eigene Bausteine vor, welche die Datenhandhabung und -weiterverarbeitung vereinfachen sollen.

Im Detail verfeinert [BVJ+11] die Konzeption von Komponenten. Beispielsweise ist die hier besonders relevante Ausgestaltung der Kommunikation in Form eines Busses in Abb. 2.7 dargestellt. So wird ein Kommunikationsbus eingeführt, an den

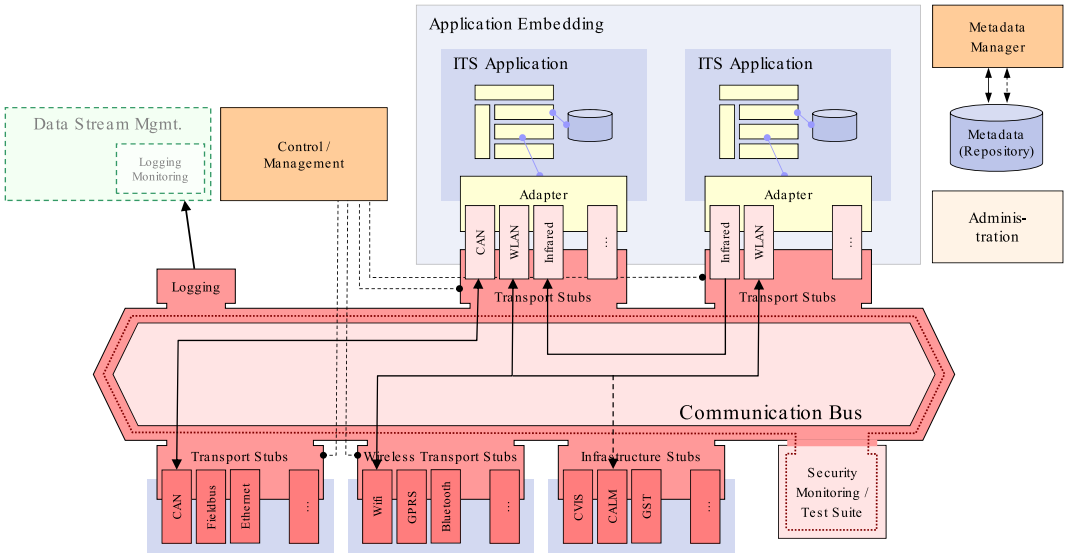

Abbildung 2.7: Konzeption des Kommunikationsbusses aus [BVJ+11]

prinzipiell alle Komponenten angeschlossen sind. Dabei spielt es keine Rolle, ob die Komponenten mobil oder stationär sind. Durch den Bus wird also eine ubiquitäre

Kommunikationsmöglichkeit aufrechterhalten. Die angeschlossenen Komponenten erhalten einen mittelbaren Zugriff auf den Kommunikationsbus. Wie in Abb. 2.7 skizziert, werden typische Kommunikationsmedien emuliert, auf die dann zugegriffen werden kann. Im Detail ist eine Emulation auf verschiedenen Schichten des ISO/ OSI-Schichtenmodells[23] vorgesehen. Diese Präzision ist in einer Referenzarchitektur vorzusehen, stellt aber keine typische Kommunikationsemulation bei einer ITS in-situ-Erprobung dar. Hier reicht eine abstrahierte Konnektivität bspw. über eine API[24]. Bemerkenswert ist hier auch das Kontrollkonzept des Busses. Es kann definiert werden, welche Komponenten miteinander kommunizieren können und auch welche Qualität die Kommunikation hat. So lassen sich leicht Szenarien erproben, in denen bspw. eine schlechte Verbindung oder eine Reichweitenbeschränkung simuliert werden soll. Ebenso werden die Sicherheitsaspekte diskutiert. Der Kommunikationsbus ist zum einen selbst gegen Fremdzugriff geschützt, erlaubt intern jedoch eine Bewertung der Sicherheit. So können sicherheitsrelevante Kommunikationsaspekte erprobt werden.

Die längerfristige Ausrichtung des resultierenden Testfelds zeigt sich auch in dem Baustein *Test-Pattern Management*. Mit diesem soll es möglich sein, die Versuchsaufbauten in einer Kampagne flexibel zu variieren und später nachzuvollziehen, in welchem Aufbau welche Daten erfasst wurden. Dies steht im Gegensatz zu stark fokussierten Testfeldaufbauten für eine oder wenige Kampagnen.

Die prototypische Implementierung der Architektur ist im Projektrahmen auf zentrale Kernkomponenten beschränkt.

Einige der in Abb. 2.6 identifizierten Bausteine sind in AIM aufgegriffen und werden sukzessive (insbesondere bedarfsorientiert) eingefügt.

sim$^{\text{TD}}$ Das Projekt *Sichere Intelligente Mobilität Testfeld Deutschland* (sim$^{\text{TD}}$) verfolgt die Erprobung von Assistenzsystemen in einem umfangreichen Feldtest. Dabei werden u. a. Anwendungen im Bereich der Fahrerinformation (z. B. Gefahrenwarnung, Verkehrslageinformation) betrachtet, die auf der Kommunikation zwischen Fahrzeugen und auch mit der Infrastruktur aufsetzen. Dazu entwickelte sim$^{\text{TD}}$ ebenfalls eine umfangreiche Testfeldarchitektur aus den in [sim09b] dokumentierten Anforderungen.

[23] ein Referenzmodell der Kommunikation in Rechnernetzen, das sieben Schichten definiert, vgl. [ISO94, S. 28], s. auch Abschnitt 2.8

[24] Application Programming Interface (API), ist eine Schnittstelle auf Quellcodeebene zur Interaktion von zwei Softwaresystemen

Diese umfassen zahlreiche Funktionen, u. a. auch kooperative Funktionen [sim09b, S. 9-11]. Da die Testfunktionen breit gefächert sind, die Datenerfassung vergleichsweise universal und die Laufzeit der Kampagnen längerfristig angelegt ist, kann auch die sim^{TD}-Architektur als generalisierbare Testfeldarchitektur angesehen werden.

Die in Abb. 2.8 dargestellte Gesamtarchitektur konkretisiert das Zusammenspiel der vier wesentlichen Komponententypen, der Fahrzeugeinheiten, der stationären Einheiten (RSU), der Verkehrsmanagementzentralen und der Versuchszentrale. Analog zu

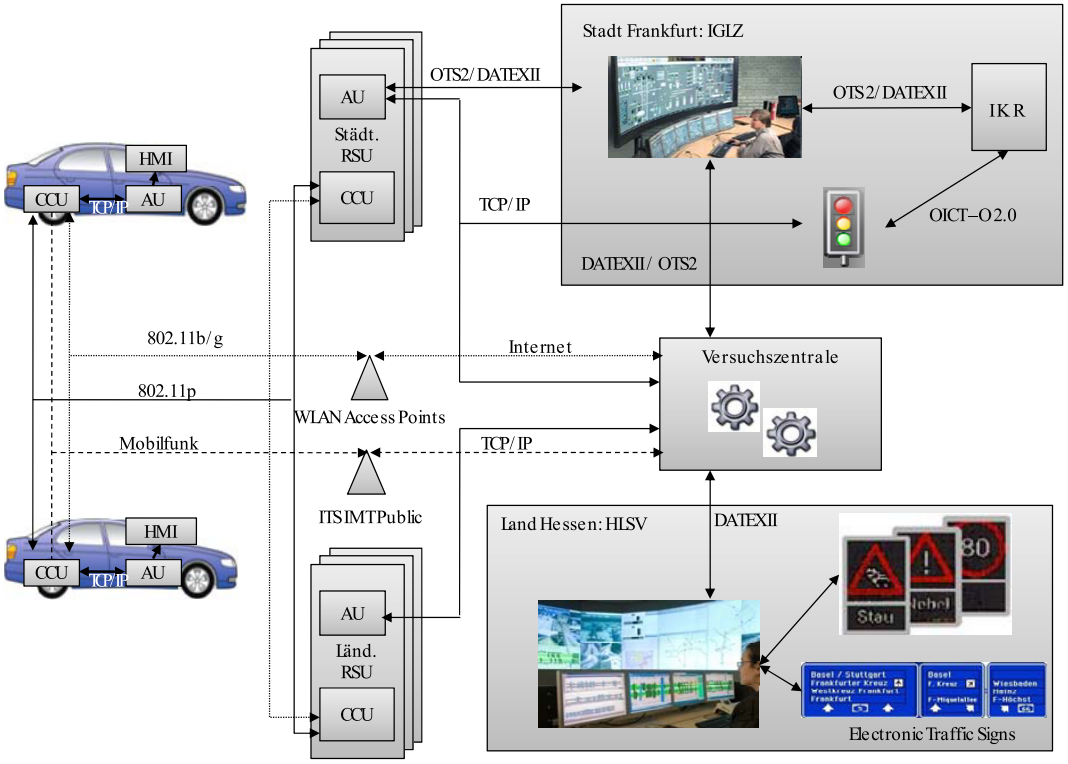

Abbildung 2.8: Gesamtarchitektur von sim^{TD} aus [sim09b, S. 4]

der CVIS-Architektur besitzen die Fahrzeuge eigene, gewidmete Hardware, die aus Communication & Control Unit (CCU), der Application Unit (nach sim^{TD}) (AU) und der Benutzungsschnittstelle (HMI) besteht. Dabei stellt die CCU wesentlich die Kommunikation sowie fahrzeugeigene Telemetriedaten bereit, während die AU die eigentliche Applikationslogik ausführt. Die beschriebenen stationären Einheiten nutzen ebenfalls diese Aufteilung mit entsprechender angepasster Funktionalität. Dabei liefern die AUs relevante Daten an die Versuchszentrale und teils auch an die Verkehrsmanagementzentren.

Konkret sind IEEE 802.11p bzw. IEEE 802.11b/g als Nahbereichskommunikations-

medium sowie mobilfunkbasierte Daten (per General Packet Radio Service (GPRS), Enhanced Data Rates for GSM Evolution (EDGE), Universal Mobile Telecommunications System (UMTS) oder High Speed Packet Access (HSPA)) vorgesehen.

Im sim^{TD} hat die Versuchszentrale eine Schlüsselrolle. In ihr laufen die versuchsrelevanten Daten auf und die Testsysteme können in einem Leitstand überwacht werden. Dabei werden Messdaten in ein Dateisystem geschrieben und können im Nachhinein ausgewertet werden. Zudem ist ein *Trace Player* vorgesehen, mit dem sich Teile der Messdaten explorativ sichten lassen. Ergänzt wird die in-situ-Erprobung durch ein Laborsystem, in dem wesentliche Komponenten unter Laborbedingungen erprobt werden können [sim09b, S. 51].

Zusammenfassend sind die sim^{TD}-Architekturkonzepte sowohl in vielen technischen Aspekten sehr konkret (bspw. Kommunikationsmedien) und zudem eng angelehnt an die oben vorgestellten Konzepte aus CVIS [FOF+10] und ETSI [ETS09].

AIM Ein weiteres Testfeldkonzept ist in AIM umgesetzt [KLGFK16]. Als im Wesentlichen für Forschung und Entwicklung von ITS-Anwendungen ausgelegtes Testfeld zielt der Aufbau des Testfelds auf eine langfristige und nachhaltige Nutzung von ca. 15 Jahren [FKM14, S. 1]. Dazu ist eine flexible und offene Systemarchitektur umgesetzt worden, die Aspekte wie Wiederverwendbarkeit und Adaptierbarkeit berücksichtigt [FSK12, S. 1f]. Die AIM-Infrastruktur umfasst eine Referenzstrecke in der Stadt Braunschweig, die mit insgesamt 25 RSU ausgerüstet ist [KLGFK16, S. 2206], welche teils mit Lichtsignalanlagen gekoppelt sind [FHL+11, S. 2]. Zudem ist eine Kreuzung mit spezieller Sensorik, wie Kameras, Radarsensoren oder Laserscanner, ausgestattet [KLGFK16, S. 2199]. Als mobile Einheiten sind bspw. OBU in Versuchsfahrzeuge integriert [FSK12, S. 5]. Sowohl OBU als auch RSU sind mit den einschlägigen Kommunikationsstandards, wie 5G und IEEE 802.11p, ausgestattet [FKM14, S. 2]. So können in AIM u. a. neuartige Fahrerassistenzsysteme entwickelt und im Feld getestet werden [FHL+11, S. 2]. Damit entspricht AIM gut den zuvor umrissenen Testfeldcharakteristiken.

In [THLK12] und [FSK12] sind die hier relevanten Architekturaspekte von AIM dargestellt. Die in AIM verbauten RSU entsprechen im Wesentlichen dem aus sim^{TD} bekannten Aufbau. Die zentrale Steuerung wird über die CCU erzielt. Diese wird von der AU flankiert, welche die für die ITS-Dienste notwendige Geschäftslogik anbietet. Wesentliche Schnittstellen sind drahtgebunden an Lichtsignalanlage und Verkehrs-

zentrale sowie drahtlos via Wireless Local Area Network (WLAN) und 802.11p an mobile Einheiten im Testfeld realisiert [FSK12, S. 2f]. Zudem können an der RSU zusätzliche Sensoren vorhanden sein, wie bspw. Radarsensoren. Abbildung 2.9 illustriert den Aufbau einer solchen RSU in AIM.

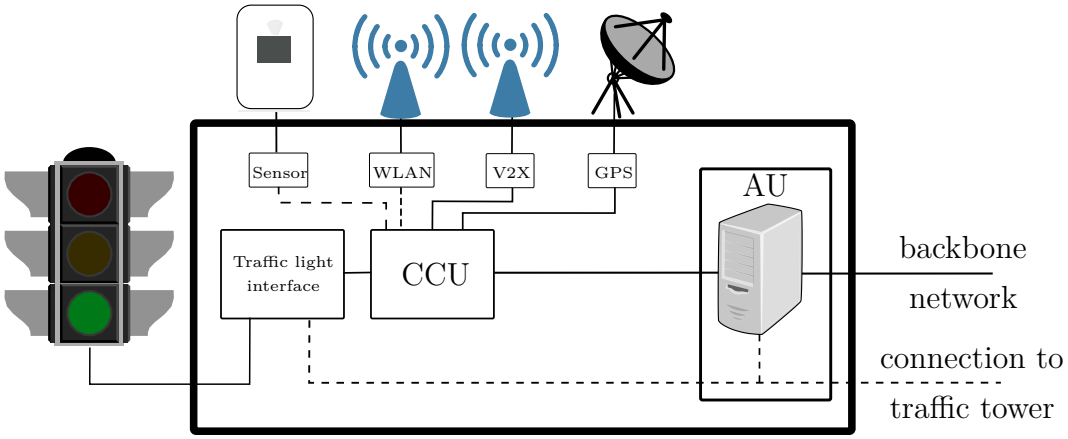

Abbildung 2.9: Hardwareaufbau einer RSU in AIM aus [FSK12, S. 3]

Das Zusammenspiel der fahrzeugseitigen AIM-Komponenten ist in [THLK12] genauer dargestellt. Den Kern bildet die Geschäftslogik des Testsystems, bspw. Assistenz- und Automationsfunktionen, die Zugriff auf Fahrzeugsensorik und -aktuatorik haben. Fahrer bzw. Passagiere können dabei die bordeigenen Anzeigen und Armaturen nutzen, um mit dem System zu interagieren. Analog zu den Schnittstellen der oben beschriebenen RSU verwendet das System im Fahrzeug den 802.11p-Standard zur Kommunikation mit straßengebundener Infrastruktur, wie bspw. Lichtsignalanlagen. Zudem wird WLAN als Kommunikationsmedium angeboten, um bspw. mit Anwendungen auf Mobiltelefonen oder anderen Mobilgeräten zu interagieren. Das Zusammenspiel der Komponenten ist in Abb. 2.10 dargestellt. Die fahrzeugseitigen Systeme von AIM verwenden eine Serviceorientierte Architektur (SOA) insbesondere zur Kommunikation mit mobilen Endgeräten [THLK12, S. 751] sowie Business Process Execution Language (BPEL) zur Modellierung von Prozessen [THLK12, S. 753].

AIM verfügt über eine Kreuzung mit spezieller Sensorik u. a. zur Erforschung von typischem Verhalten der Verkehrsteilnehmer im Sinn von NDS [KLG16, S. 1]. Dazu werden motorisierte bzw. nicht-motorisierte Verkehrsteilnehmer von unterschiedlicher Sensorik in einem zentralen System (DISCUs) erfasst. Dies hat Zugriff auf den Status

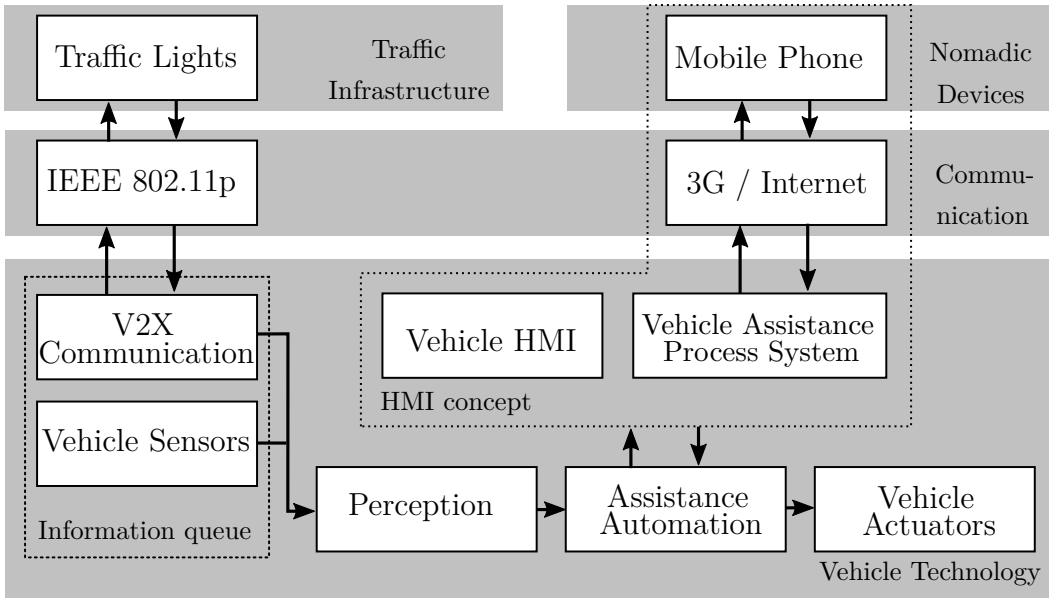

Abbildung 2.10: AIM-Architekturdarstellung aus [THLK12, S. 5]

der Lichtsignalanlage an der Kreuzung und kann die Bewegungen der Verkehrsteilnehmer sowie Videoaufzeichnungen nachhalten. Abbildung A.2 (im Anhang, S. 180) zeigt ein entsprechendes Architekturbild dieser Installation. Die Wiederverwendung dieser Installation in unterschiedlichen Projekten ist in [KLG16, S. 4] ausgeführt.

Zusammenfassend sind die AIM-Architekturkonzepte in vielen technischen Aspekten ebenfalls konkret (bspw. Kommunikationsmedien, SOA, BPEL). Die technische Umsetzung orientiert sich an den Konzepten aus CVIS und ETSI und ähnelt sim^{TD} bspw. hinsichtlich RSU. Die Aufzeichnung von Bewegungsprofilen kann als vergleichsweise randwertiger Anwendungsfall eines Testfalls gesehen werden. Dennoch zeigt dieser Fall die Intention einer nachhaltigen und längerfristigen Nutzung auf.

DITCM Das Akronym bezeichnet eine Arbeitsgruppe in den Niederlanden, die über den Zusammenschluss von mehreren Einrichtungen im Raum Helmond (Niederlande) u. a. eine umfassende Testlandschaft für kooperative ITS-Anwendungen darstellen will [DIT15, S. 7], [PvvP16, S. 2]. Mit Stand 2016 hat die Arbeit an der Umsetzung des Testfelds begonnen [DIT15, S. 5]. Im Kontext dieser Arbeit ist bemerkenswert, dass der zugrunde liegende Architekturprozess sowohl einsehbar als auch in Ergebnissen dokumentiert ist[25]. Als methodischen Ansatz zur Synthese einer Test-

[25]s. DITCM Library, online `http://www.ditcm.eu/library`, zuletzt aufgerufen 12.09.2016

feldarchitektur betrachtet DITCM zunächst den Stand der Technik ([BVSv16]), legt eine Referenzarchitektur für Testsysteme ([PvvP16]) fest und leitet daraus einen konkreten Architekturentwurf ([Pv16]) ab [vOB+15, S. 13f].

Die Referenzarchitektur für Testsysteme ist in den drei Sichten ([PvvP16, S. 5]) dargestellt: *Physical View* beschreibt die Gestaltung der Teilsysteme und deren Schnittstellen[26], *Functional View* definiert die Verteilung von Funktionalität über die Bausteine der Systemlandschaft[27] und *Communications View* legt die Schnittstellen zwischen den Architekturelementen fest.

[NPW+13] liefert einen ersten Eindruck über die DITCM-Testinfrastruktur. Der technische Aufbau ähnelt der in sim^{TD} und AIM beobachteten Systemlandschaft mit RSU und OBU. Zudem wird in [NPW+13, S. 3] ein Test Management Center angedeutet, welches wesentliche Testfeldaufgaben funktional abdeckt.

Auf den ersten Blick lässt sich feststellen, dass ITS Test Beds die weitreichendsten Konzepte zur flexiblen Wiederverwendung umfasst. Die Rekombination der Testfeldkomponenten verschiedener Versuchsaufbauten ist in sim^{TD} nicht vorgesehen. AIM folgt ebenfalls einer konkreten Verschaltung (bspw. RSU), zeigt jedoch in gewissen Teilen eine Adaptierbarkeit durch Wiederverwendung in unterschiedlichen Projekten. Die technischen Konzepte in sim^{TD} und AIM sind konkreter formuliert, die Verwendung von Medien auf Basis von IEEE 802.11p als lokales Funknetz oder UMTS bzw. 5G als großflächiges Funknetz ist für eine Vielzahl an Anwendungen hinreichend, aber wird bei preisgünstigen, hochmobilen Anwendungen auf Basis von Kommunikation mittels Infrarot oder Bluetooth kein passendes Testfeld sein. Ebenso ist eine feste Verknüpfung der möglichen Kommunikationswege der in Abb. 2.8 dargestellten Gesamtarchitektur wenig förderlich für innovative Anwendungen, die bspw. auf einer Kommunikation zwischen Fahrzeug und Smartphone aufsetzen. Hier bietet AIM einen entsprechenden Ansatz. ITS Test Beds und sim^{TD} identifizieren einen zentralen Leitstand und eine Datenerfassungsinstanz, wobei ITS Test Beds weitreichender die Analyse unterstützt und sim^{TD} explorative Werkzeuge wie den *Trace Player* bereitstellt. In AIM ist eine Anbindung an ein Verkehrsmanagementzentrum in [FHL+11, S. 4] und [FSK12, S. 3] angedeutet, deren Funktionsweise analog zu Werkzeugen wie dem *Trace Player* jedoch nicht weiter ausgeführt. Die Testfeldarchitektur von ITS Test Beds zeigt im Vergleich zu den konkreten Umsetzungen von sim^{TD} und AIM eine komplementäre Herangehensweise. Der sehr flexible, umfassende und unkonkrete Ansatz von ITS Test Beds ergänzt die

[26] exemplarisch in Abbildung A.3 im Anhang dargestellt, S. 181
[27] exemplarisch in Abbildung A.4 im Anhang dargestellt, S. 182

konkrete und teils starre Ausprägung von sim^{TD} und AIM. Für DITCM liegt noch kein bewertbarer Testfeldarchitekturentwurf vor. Aus [NPW+13, S. 3] geht hervor, dass ebenfalls eine Architektur angestrebt wird, die wesentliche Komponenten wie bspw. RSU und OBU von sim^{TD} und AIM aufgreift. Vor dem Hintergrund der unterschiedlichen Entwicklungs- bzw. Projektzeiträume (vgl. Abb. 2.5) scheinen dies evolutionär stabile Komponenten.
Kapitel 4 führt eine ausführliche Architekturbewertung dieser Architekturen aus und vergleicht die ITS Test Beds-, sim^{TD}- und AIM-Konzepte über deren Qualitätsmerkmale. Die Betrachtung der beiden Testfeldarchitekturen schließt die domänenspezifischen Grundlagen ab. Die folgenden Abschnitte greifen die softwaretechnischen Grundlagen auf, die für Architekturgestaltung und vergleichende Bewertung benötigt werden.

2.7 Architekturgestaltung

Nachdem in vorhergehenden Abschnitten die domänenspezifischen Grundlagen dargestellt wurden, ist in diesem Abschnitt eine Betrachtung der fachlichen Grundlagen ausgeführt. Architekturüberlegungen für ein Testfeld stehen im Zentrum dieser Arbeit. Deshalb wird nun Architektur als fundamentales Konzept der Softwareentwicklung ausgeleuchtet. Zudem wird beschrieben, wie eine Architektur aufgebaut und dargestellt werden kann. So werden adäquate Architekturparadigmen identifiziert und im folgenden Schritt deren Anwendbarkeit auf die Problemstellung bewertet.

Bereits in den 1960er Jahren zeigte sich, dass Planung und Entwicklung von Software anderen Regeln unterliegen, als dies beispielsweise in anderen kollaborativen Disziplinen (bspw. Schiffbau) der Fall war. Die fachliche Herausforderung trat in den Hintergrund, die Größe und Komplexität der Softwareentwicklungsprozesse wurden zunehmend als problematisch verstanden[28]. Traditionelle Planungs- und Kontrollmechanismen waren nicht einfach auf neue Problemstellungen übertragbar[29]. Zum Ende der 1960er Jahre wurden Softwaresysteme so umfangreich, dass deren Software kaum mehr von einer Person oder einem kleinen Team überblickt werden konnte. Dies führte zur Erkenntnis, dass angepasste strukturgebende Methoden und Werkzeuge notwendig sind, um Umfang und Komplexität moderner Softwaresysteme zu handhaben und die kollaborative Bearbeitung sicherzustellen [McC97, S. 142f]. Die resultierende Struktur wird gemeinhin als Softwarearchitektur aufgefasst [SSS81]. Spätestens mit dem IEEE 1471-2000 Standard ([IEE00]) hat der Architekturbegriff weite Verbreitung erfahren.

In der Literatur wird Softwarearchitektur oft als strukturierte Darstellung von Softwarekomponenten, deren Schnittstellen und Beziehungen aufgefasst:

> „The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.“ [BCB97, S. 3]

Deutlich wird bei [BCB97] nicht nur der strukturierende Teil, sondern auch der Aspekt,

[28] „As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem [...] This is the software architecture level of design.“ aus [GS93, S. 1]

[29] Ein Vergrößern der Entwicklerzahl bei einem verspäteten Projekt führt zu einer noch späteren Fertigstellung, vgl. *The mythical man-month* in [Bro95].

dass die Komponenten Schnittstellen bereitstellen, um ihre Funktionalität untereinander referenzieren zu können. Dieser Auffassung des Architekturbegriffes wird in dieser Arbeit gefolgt.
Weiterhin ist die Struktur eine Abstraktion auf hoher Ebene:

> „The architecture of a system is concerned with the top-level decomposition of the system into its main components.“ [Bos00, S. 10]

So betont [Bos00] die Zerlegung des (geplanten) Systems in Kernkomponenten. Die Zerlegung ist der bedeutsamste methodische Ansatz bei der Definition einer Architektur. Er wird später eine wichtige Rolle bei der Ausgestaltung einer geeigneten Architektur für Testfelder spielen.
Der Hauptzweck einer Architektur liegt in der abstrakten Beschreibung des Gesamtsystems. Dadurch lassen sich in einer frühen Entwicklungsphase gewisse nicht-funktionale Aspekte des intendierten Systems bewerten. Neben der Leistung und Zuverlässigkeit können so bspw. auch Wartbarkeit und Weiterentwicklungsaufwand besser abgeschätzt werden.
Eine Architektur kann allen an der Entwicklung Beteiligten als Grundlage dienen, kleinere, isolierte Teile des Gesamtvorhabens zu betrachten und ohne tiefe Kenntnis über die Interna anderer Bereiche weiter zu detaillieren. Die Architektur kann als plastische Voransicht fungieren. So kann bspw. mit einer szenarienbasierten Architekturbewertung[30] bestimmt werden, wie gut der gewählte Architekturentwurf qualitativ zu dem Anwendungsfall passt.
Die Architektur hat auch über die Grenzen des anvisierten Zielsystems hinaus Auswirkungen, denn es werden wichtige technische und organisatorische Vorgaben für die angrenzende Soft- und Hardwarelandschaft festgelegt. Bei Produktlinien betrachtet man z. B. nicht nur isoliert eine konkrete Anwendungsimplementierung, sondern sucht möglichst umfassend eine Gesamtbetrachtung, die sowohl funktional als auch zeitlich (Weiterentwicklung) nahestehende Anwendungen umfasst. Durch produktübergreifende Konsistenz lässt sich bspw. ein Paradigmenbruch vermeiden und es lassen sich hohe Integrations- und Weiterentwicklungskosten umgehen. Daher werden auch die bestehenden Architekturdefinitionen von Testsystemen in diesem Zusammenhang betrachtet (vgl. Testsystem, S. 15).
In der Literatur werden Architekturen mit einem domänenspezifischem Zuschnitt als Domain-Specific Software Architecture (DSSA) diskutiert. Diese werden u.a. von [HR94] als Verbund von Softwarekomponenten verstanden, die auf eine bestimmte Domäne spezialisiert sind, die für eine effektive Nutzung in dieser Domäne generalisiert sind und in einer

[30] vgl. Software Architecture Analysis Method (SAAM) in [KABC96, S. 5ff]

standardisierten Struktur (Topologie) zusammengefasst sind. Wesentliche Zielsetzung ist dabei die effiziente Anwendung (*mapping*) von generalisierten Lösungen (*solution elements*) für spezifische Aufgabenstellungen (*solution elements*) innerhalb der entsprechenden Domäne [TCY95, S. 4]. Eine in der Praxis umfasst eine DSSA typischerweise eine Referenzarchitektur, eine Komponentenbibliothek sowie ein Verfahren zur Auswahl bzw. Konfiguration von Komponenten für die Zielarchitektur [HRPL+95, S. 288].
Im Gegensatz zu dem oben ausgeführten Architekturbegriff ist eine Referenzarchitektur als Menge grundlegender Designentscheidungen aufzufassen, die auf mehrere artverwandte Systeme einer Domäne anwendbar sind [CTK19, S. 112]. Sie kann dazu u. a. Vorschläge zur Kombination von Funktionsmodulen und zum Datenfluss ([BCB97]) sowie *best practices* ([NBM12, S. 1197]) umfassen. Diese Strukturen sowie das in der Gestaltung eingeflossene Domänenwissen ([NBM12, S. 1197ff]) können dann als abstraktes Muster zur Ableitung einer konkreten Architektur eines entsprechenden Zielsystems dienen [CMV+10, S. 19ff].
Neben der Spezialisierung in Sinn einer DSSA bestehen auch generalisierte Architekturmuster und -stile, die im diesem Zusammenhang relevant sind. Diese werden im Folgenden betrachtet.

2.8 Architekturmuster und -stile

In der Literatur sind Architekturmuster gegenüber Architekturstilen begrifflich nicht strikt unterschieden. Oft werden diese synonymisch aufgefasst. Sind einzelne Funktionen im Zielsystem identifiziert, können Architekturmuster deren wesentliche Aspekte strukturieren. Beispielsweise kann so die Kontrollstruktur ausgeformt und definiert werden, wie Komponenten untereinander Daten austauschen. Architekturmuster sind ein etabliertes Werkzeug, um Software mit überschaubarem Umfang zu strukturieren. Architekturmuster sind jedoch nicht gleichzusetzen mit Entwurfsmustern (*design pattern*). Letztere adressieren implementierungsnahe Lösungen immer wiederkehrender Problemstellungen auf niedriger Abstraktionsebene, wie bspw. *Dekorierer*- und *Beobachter*-Muster[31].
Architekturstile liefern wichtige konzeptionelle Muster, um die im Abschnitt zuvor ausgeführten strukturellen Probleme zu adressieren. Typischerweise durchziehen sie die technische Planung einer Entwicklung in allen Abstraktionsebenen und bilden so eine grobe Leitlinie für die Zerlegung in funktionale Bausteine sowie deren Verknüpfungen.

[31] vgl. *Decorator* in [GHJV01, S. 199] und *Observer* in [GHJV01, S. 287]

Dadurch ist die Gesamtstruktur auf allen Abstraktionsebenen einheitlich und transparent und die implizierten nicht-funktionalen Eigenschaften des Systems auf den entsprechenden Ebenen bewertbar.
Bei der Erforschung und Entwicklung von Architekturen haben sich einige generische Architekturmuster bewährt. Diese sind u. a.:

Layers Das *Layers*- oder Schichtenmuster zerlegt die Gesamtfunktionalität eines kontrollflussorientierten Systems in verschiedene Schichten. Dabei werden die Schichten aus Gruppen ähnlicher Teilfunktionen gebildet. Mit jeder Schicht nimmt die Abstraktion zu. Ein typisches Beispiel für ein solches Muster ist das ISO/OSI-Schichtenmodell[32].

Pipes&Filter Das *Pipes&Filter*-Muster sieht in datenflussorientierten Systemen eine Kombination von Teilfunktionen durch sequenzielle Abarbeitung vor. Dabei werden die Funktionen über Kanäle (*Pipe*) verknüpft und arbeiten auf Grundlage der Ausgabe des vorangehenden Prozesses. Die manipulierten Daten (*Filter*) werden dann entsprechend an den nachfolgenden Prozess weitergegeben. Ein anschauliches Beispiel für eine entsprechende Verarbeitung sind Verknüpfungen von Linux/Unix-Befehlen in einer Kommandozeile.

Blackboard Das *Blackboard*-Muster zielt auf datenzentrierte Anwendungen, die kein fest definiertes Lösungskonzept haben. Dies ist bspw. bei Sprach- und Texterkennung typisch. Das Blackboard fungiert dabei als dynamischer Datenpool, auf dem verschiedene Funktionen arbeiten können. Das Ergebnis wird nach jeder Bearbeitung bewertet, um einen möglichen Gütegewinn der erreichten Lösung bewerten zu können. So können auch Teilergebnisse weiterverwendet oder aber auch verworfen werden.

Model-View-Controller Das Architekturmuster Model-View-Controller (MVC) schlägt gemäß *separations of concern* eine Strukturierung der Software und Datenhaltung in drei wesentliche Segmente vor: Das *Model* umfasst die eigentliche Datenhaltung und Funktionen, die zum Zugriff auf diese benötigt werden. Im *View* werden Funktionen gekapselt, die zur Darstellung (bzw. Interaktion) benötigt werden. *Controller* umfasst die eigentliche Funktionslogik (bspw. Geschäftsprozesse und Fachfunktionen).

Ursprünglich von [MHR79] 1979 zur Modellierung von Benutzeroberflächen vorgeschlagen, ist MVC inzwischen fundamentales Architekturmuster für komplexe

[32] Open Systems Interconnection (Reference Model) (OSI) der Internationale Organisation für Normungs (ISOs), vgl. [ISO94, S. 28]

Systeme [KP88]. Insbesondere in heterogenen Systemlandschaften, wie bspw. Internetdiensten, sind MVC-strukturierte Softwaresysteme de facto Standard. In diesem Kontext sind MVC-basierte Architekturen von besonderer Relevanz, da dieses Konzept in vielen ITS-Systemen verwendet wird (bspw. [BST11, S. 11]). Zudem wurden MVC-basierte Systeme von verschiedenen Gremien als Standard vorgeschlagen, wie bspw. in der *European ITS Communication Architecture* (vgl. [B+09, S. 189]).

Objektorientierung Kernidee der Objektorientierung (OO) ist die Verknüpfung von Daten und Methoden in einer Einheit, dem Objekt. Durch Konzepte wie Vererbung, Modularisierung, Kapselung und Abstraktion kann die OO ebenfalls als Muster zur Architekturgestaltung dienen.

In den letzten Jahren haben sich weitere Muster zum Aufbau größerer Architekturen bewährt. Für den Aufbau einer Testfeldarchitektur sind eben solche interessant, die insbesondere für eine verteilte, heterogene Systemlandschaft gut geeignet sind. Dies grenzt eine mögliche Auswahl ein, nennenswert sind hier u. a.:

Client-Server-Modell Beim *Client-Server-Modell* wird im Wesentlichen zwischen zwei komplementären Rollen unterschieden. Zum einen stellt der Server gewisse Dienste zur Verfügung, zum anderen können diese von einem Client über ein definiertes Protokoll abgerufen werden. Dieses Architekturmuster liegt sehr vielen internetbasierten Diensten zugrunde. Bspw. erlaubt ein FTP-Server[33] mit entsprechenden Anfragen per FTP von Clients den Dateiaustausch zwischen zwei entfernten Rechnern. Vergleichbare Mechanismen lassen auch eine entfernte Funktionsausführung zu. So lassen sich mit diesem Modell ebenfalls weitreichende Architekturen für verteilte Systeme aufstellen.

Dienstorientierung Eine SOA beruht auf der Aufteilung der Gesamtfunktionalität in kleine Einheiten, die Dienste. In diesen ist jeweils ein wohldefinierter Teil der Geschäftslogik auf einer bestimmten Abstraktionsebene gekapselt. Durch eine hierarchische Orchestrierung von Diensten und deren geschickte Wiederverwertung können ebenfalls umfassende Architekturen für sehr komplexe Anwendungslandschaften (bspw. über verteilte Systeme) definiert werden. Dabei setzt die SOA auf dem zuvor beschriebenen Client-Server-Modell auf. Ein Dienst beantwortet Anfragen anderer

[33] File Transfer Protocol (FTP) ist ein typisches Netzwerkprotokoll zum Austausch von Dateien über Netzwerke (bspw. Internet).

Dienste (Server-Rolle, auch Serviceanbieter) und stellt selbst Anfragen an andere Dienste (Client-Rolle, auch Servicekonsument). So agierende Dienste werden auch Webservices genannt.

Praktisch umgesetzt wird eine SOA oft mit Webservices, die über XML-Nachrichten, z. B. Simple Object Access Protocol (SOAP) oder Advanced Message Queuing Protocol (AMQP), kommunizieren. Eine detaillierte Formatbeschreibung solcher Nachrichten kann in einer Web Services Description Language (WSDL), einer Beschreibungssprache für die von einem Webservice angebotenen Dienste, definiert sein, die Dienste leicht reorganisier- und wiederverwendbar machen.

Abschnitt 2.9 steigt tiefer in das Zusammenspiel von Diensten in verteilten Systemen ein und greift dazu die hier vorgestellten Konzepte auf.

Aus der Betrachtung der bestehenden Architekturüberlegungen zeichnet sich ab, dass die Systemlandschaft eines Testfeldes über viele heterogene Komponenten verteilt sein wird. Somit liegt es nahe, den Architekturentwurf auf die Anforderungen für ein verteiltes Testfeld auszurichten. Die beschriebene SOA ist gut geeignet, als Architekturparadigma für diesen Einsatzzweck zu dienen. Daher ist es an dieser Stelle nützlich, genauer die damit verbundenen Konzepte zu beleuchten.

Das Paradigma einer SOA ist in der rudimentären Auffassung als Zerteilung der Gesamtfunktionalität in atomare Dienste kein hinreichendes Entwurfsmuster, um ein umfassendes Architekturkonzept zu stellen. Viele Aspekte bleiben unberücksichtigt, bspw. ist ein Dienstorchestrierungsmechanismus nicht inhärent. So bedarf es komplementierender Konzepte, die folgend kurz umrissen werden.

Sind einfache Funktionseinheiten in Diensten isoliert, gibt es verschiedene Mechanismen, diese zu kombinieren. Eine gängige Methode ist der explizite Aufruf (*request*) eines Dienstes durch einen anderen. So wird für jeden Aufruf eine entsprechende Antwort vom aufgerufenen Dienst zurückgeliefert. Eine weitere Möglichkeit, Dienste zu verknüpfen, beruht auf ereignisgetriebenen Konzepte. Dazu registrieren sich bspw. Dienste A und B bei Dienst C. Dienst C produziert über die Laufzeit verschiedene Ereignisse, die von den Diensten A oder B mitverfolgt werden. Potenziell kann so ein entsprechendes Ereignis von Dienst A empfangen werden, das eine definierte Funktion des Dienstes anspricht. Eine bekannte Realisierung ist z. B. im Simple Network Management Protocol (SNMP) gegeben. Dies ist ein verbreitetes Kommunikationsprotokoll zur Überwachung von Netzwerkkompo-

nenten wie Router, Switches oder Server. Bspw. können solche Komponenten via SNMP unaufgeforderte *Traps*-Ereignisnachrichten an eine Kontrolleinheit senden. Diese fungiert dann als Beobachter für solche Ereignisse.
Um dieses Konzept in die Architekturüberlegungen einfließen zu lassen, wird dessen Charakteristik im folgenden Abschnitt beschrieben.

2.9 Architekturen für verteilte Systeme

Aus den Betrachtungen der verschiedenen Infrastrukturelemente eines typischen Testsystems wird schnell sichtbar, dass es sich bei den Unterstützungssystemen der Testfeldsystemlandschaft um kein abgeschlossenes und monolithisches System handeln wird. Einen solcher Zusammenschluss vieler Komponenten (oder Prozesse) zu einem Gesamtsystem wird auch *Verteiltes System* genannt. Die Betrachtung der Architekturen für solche Systeme ist Inhalt dieses Abschnitts.
[Cou05, S. 25-26] charakterisiert verteilte Systeme u. a. als:

Offen & Heterogen Verteilte Systeme sind nicht abgeschlossen, sondern sind offen für die Einbettung unterschiedlicher Komponenten von verschiedenen Herstellern, die intern (Programmierung) individuell gelöst sind.

Nebenläufig Die Abarbeitung der Prozesse kann echt nebenläufig erfolgen, indem komplexe Algorithmik auf viele Recheneinheiten verteilt wird, um eine schnelle Abarbeitung zu erzielen. Daher müssen die Abhängigkeiten der Prozesse genau betrachtet werden, um bei der Aggregation der Teilergebnisse auf ggf. fehlende Berechnungen zu reagieren.

Fehlertolerant Durch die Abhängigkeiten der verteilten Einheiten kann nicht immer sichergestellt werden, dass die Abarbeitung wie geplant funktioniert. So müssen alternative Konzepte bestehen, die bei entsprechenden Fehlern auf anderen Funktionen zurückgreifen oder einen leistungsverminderten Dienst anbieten.

Skalierbar Durch die offene Gestaltung verteilter Systeme ist es ein typisches Szenario, dass solche Systeme durch Hinzufügen weiterer Komponenten und Funktionen wachsen. Dies kann vorhandene und bereits gut ausgelastete Kernkomponenten an die Leistungsgrenzen führen. Somit müssen Konzepte bestehen, die es diesen Komponenten erlaubt, die zusätzliche Last abzufangen.

Das Konzept des verteilten Systems passt sowohl gut zu den Überlegungen hinsichtlich der Konzepte des Client-Servers und der Diensteorientierung im vorigen Abschnitt als auch zu bestehenden Architekturüberlegungen von Testsystemen, wie bspw. in CVIS. Bestehende Testfeldarchitekturen wie sim^{TD} und ITS Test Beds beschreiben ebenfalls verteilte Systeme.
In der Charakterisierung klingen schon einige offensichtliche Risiken verteilter System an. [Som16] fasst vier wesentliche Herausforderungen zusammen:

Komplexität Durch die Verteilung der Funktionalität auf verschiedene Einheiten ist ein Komplexitätszuwachs offensichtlich. Es müssen besondere Mechanismen zum Test solcher Systeme bestehen. Bspw. kann die zu erwartende Laufzeit einer Funktion nur a priori bestimmt werden, wenn zuvor alle an der Bearbeitung beteiligten Funktionsuntereinheiten bekannt und die zugrunde liegenden zeitkritischen Ressourcen identifiziert sind. So sind Design und Erprobung solcher Systeme nicht trivial.

Sicherheit Die Verteilung der Systeme bedingt einen Datenaustausch der Funktionen untereinander. Somit ist Kommunikation ein wichtiger Aspekt bei der Gestaltung verteilter Systeme. Kommunikationswege sind jedoch auch aus sicherheitskritischer Perspektive zu betrachten. So sind deren Verfügbarkeit und Zuverlässigkeit nicht per se gegeben und die Kommunikationswege zwischen verteilten Diensten potenziell abhör- und manipulierbar.

Handhabbarkeit Wie zuvor beschrieben, können Funktionen ggf. nicht verfügbar sein oder es besteht keine stabile Kontrolle über Funktionen. Daraus können sich im Fehlerfall schwer vorausberechenbare Situationen ergeben, in denen bspw. wechselwirkende Prozesse auftreten und die Funktion des Gesamtsystems gefährden.

Unvorhersagbarkeit Typischerweise kann vor dem Aufruf nicht gewährleistet werden, dass die adressierte Funktion überhaupt zur Verfügung steht. So kann für eine Prozesskette in einem echten verteilten System keine Prognose über die Abarbeitung (in Zeit / Qualität) gegeben werden. Um dem gegenzusteuern, müssen Mechanismen etabliert werden, die eine gewisse Verfügbarkeit des Systems bspw. durch QoS[34] garantieren und entsprechend eine gewisse Fehlertoleranz notwendig.

[34]Quality of Service (QoS), Begriff aus der Telekommunikation, beschreibt herkömmlich die Güte einer Telefongesprächsverbindung durch gewisse Kennzahlen wie Rauschen und Verzerrung. Hier als Qualitätsmetrik funktionaler Komponenten in verteilten Systemen gebraucht.

Es bestehen verschiedene Konzepte zur Gestaltung einer Architektur für ein solches System. Ein prominentes Beispiel ist das im Internet ubiquitäre Client-Server-Konzept, welches in Abschnitt 2.8 bereits beschrieben wurde. Ebenfalls bekannt wurden P2P[35]-Architekturen zur Verteilung von Daten im Internet. Dies zeigt, dass einige Architekturmuster nur für einen eng abgesteckten Anwendungsfall konzipiert sind. Dies lässt spezialisierte Muster, wie bspw. P2P- und Multimedia-Architekturen[36], weniger geeignet erscheinen.

In verteilten Systemen wird oft eine Middleware[37] eingesetzt, um die Kommunikation gekapselter Komponenten zu ermöglichen. Neben JEE[38] und .NET[39] basieren Middlewareumsetzungen oft auf Common Object Request Broker Architecture (CORBA).

Die Funktionsweise einer Middleware kann am Beispiel CORBA-basierter Architekturen dargestellt werden. Herzstück einer solchen Architektur ist der Object Request Broker (ORB), der lokationstransparent Anfragen von einem Klienten zu angebotenen Objekten bzw. Diensten vermittelt. Im Detail wird für eine geplante Schnittstelle eine formale Spezifikation in einer Interface Definition Language (IDL) erstellt. Diese wird von einem IDL-Compiler in entsprechende Interface-Objekte gewandelt, die, eingebunden in den jeweiligen Funktionsteil, das Interfacing abdecken. So entsteht ein *stub* für die Klient- und ein *tie* (o. *skeleton*) für die Serverseite. Aus Sicht des Klienten kann dann auf ein entferntes Objekt des Servers wie auf ein lokales zugegriffen werden. Abbildung 2.11 illustriert dies.

Das CORBA zugrunde liegende Architekturparadigma für verteilte Systeme ist somit gut geeignet eine verteilte Funktionalität darzustellen. Weniger einfach ist eine Realisierung in CORBA, wenn verschiedene Hersteller der ORB und unterschiedliche Programmiersprachen eingesetzt werden. So kann mit Objektserialisierung und Marshalling[40] auch dieses Problem adressiert werden, stellt jedoch einen signifikanten Mehraufwand hinsichtlich Komplexität, Erweiterbarkeit und Wartbarkeit dar.

[35] Peer-to-Peer (P2P); diese Architekturen gehen von einem Netzwerk mit Knoten mit gleichem bzw. ähnlichem Funktionsumfang aus. So kann ein Peer als ein Funktionsklon betrachtet werden und so die gewünschte Funktionalität über verschiedene Peers im P2P-Netz verteilt werden. Vgl. [RH08, S. 433ff].

[36] vgl. [RH08, S. 409f]

[37] Softwareschicht, die verschiedene Softwarekomponenten integrieren soll sowie gemeinsam genutzte Funktionen im System anbietet.

[38] Java Platform, Enterprise Edition

[39] Microsoft .NET Platform

[40] alg. die Umwandlung von Daten in ein Transportformat, bspw. Extensible Markup Language (XML) für den strukturierten Datenaustausch

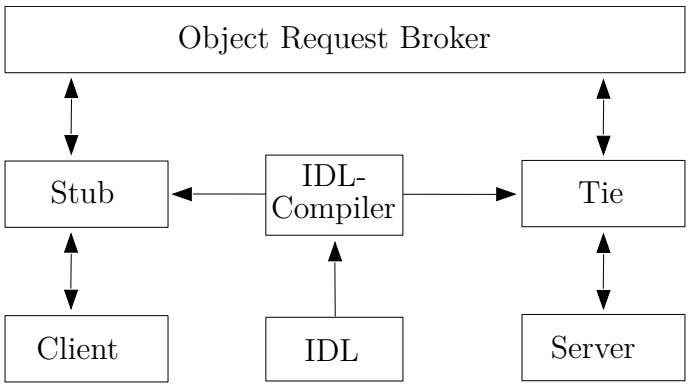

Abbildung 2.11: Typische Verknüpfung von Funktionalität über Stub & Tie, angelehnt an [Obj11b, S. 15]

Eine Möglichkeit zu Modellierung eines verteilten Systems ist die Diensteorientierung. Dieses Konzept wurde in Abschnitt 2.8 schon angesprochen. Aufgrund der Relevanz wird an dieser Stelle eine Vertiefung durchgeführt.

Als kleinste Einheit liegen üblicherweise Dienste bzw. Services vor, die eine einfache Funktion erfüllen. Diese können bspw. über den Zerlegungsvorgang (s. Abschnitt 2.12) gewonnen werden. Die Dienste können sich gegenseitig aufrufen und so Daten und Funktionen des jeweils aufgerufenen Dienstes nutzen. Der aufrufende Dienst wird dann *Konsument* und der aufgerufene *Anbieter* genannt. Im Detail findet dabei typischerweise eine Kommunikation nach dem Client-Server-Prinzip[41] statt, siehe Abb. 2.12. Dabei fungieren die Schnittstellen in den Diensten jeweils als *SOAP-Server* bzw. *-Client*. In der Praxis ist dabei eine Beschreibung notwendig, welche die nach außen freigegebenen Schnittstellen des anbietenden Dienstes genauer beschreibt und auch die Ausgabe entsprechend definiert. Dies kann mit verschiedenen Methoden umgesetzt werden. Eine weitverbreitete nutzt dazu eine Beschreibungssprache (WSDL) sowie ein Diensteverzeichnis. Im Prinzip registriert jeder Dienst seine Funktionalität und das Format, in dem diese Funktionen adressiert werden können per WSDL-Dokument bei einem UDDI-Repository[42]. Dienste können so nach benötigten Funktionen über das Repository geeignete Dienste identifizieren und

[41] In einer diensteorientierten Architektur wird grundsätzlich nicht zwischen Client und Server unterschieden, da Diensten i. d. R. beide Rollen inhärent sind. In der beschriebenen konkreten Kommunikationssituation liegt dieses Muster jedoch zugrunde.

[42] Universal Description, Discovery and Integration (UDDI) ist ein Verzeichnisdienst, der insbesondere in diensteorientierten Architekturen die funktionale Beschreibung von Diensten ermöglicht und auf der Grundlage eine Suche nach Diensten erlaubt.

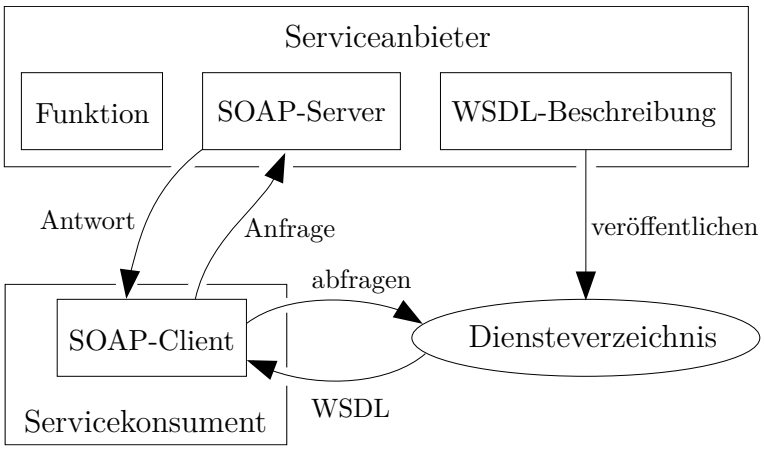

Abbildung 2.12: Typisches Zusammenspiel von Diensten mittels SOAP, WSDL und UDDI

erhalten zudem eine genaue Beschreibung der Dienste. Dies ermöglicht eine wohldefinierte Kommunikation der Dienste. Registrierung bei einem Diensteverzeichnis und dessen Abfrage ist ebenfalls in Abb. 2.12 dargestellt.

Im Vergleich zu CORBA lassen sich viele Gemeinsamkeiten identifizieren, wie bspw. die Spezifikation von Schnittstellen in einem zentralen Format (IDL u. WSDL) und die automatisierte Ableitung passender Schnittstellen. Im Gegensatz zu CORBA ist SOA wesentlich leichter in heterogene Softwarelandschaften zu etablieren, da das Datenaustauschformat von den unterliegenden Technologien (bspw. Programmiersprachen o. UDDI) abstrahiert. Daher korrespondiert das SOA-Paradigma gut mit nicht-funktionalen Anforderungen wie bspw. Flexibilität [RWJ+11, S. 86f].

Diese Überlegungen werden später in der Konkretisierung des Architekturentwurfs (Kap. 3) als essentielles Paradigma und auch in der praktischen Umsetzung der Methode aufgegriffen.

2.10 Architektursichten

Zur umfassenden Beschreibung von Architekturen ist es i.d.R. nicht hinreichend, nur eine Beschreibung aus einer gewissen Blickrichtung vorzunehmen [CGB+10, S. 22]. So ist auch eine technische Konstruktionszeichnung, die nur aus einer Schnittebene besteht, nicht hinreichend, um Konturen und Strukturen komplexer Körper darzustellen. Analog lässt die Ausführung eines Klassendiagramms als isolierte Architekturbeschreibung bspw.

technischen Aufbau und Datenbankschemata unklar. Daher bedarf es mehrerer Sichtweisen auf den Architekturentwurf, um unterschiedliche Aspekte einer komplexen Software umfassend darzustellen.
In der Literatur werden solche Architektursichten (*views* [ISO11]) diskutiert, die aus Blickwinkeln (*viewpoints* [ISO11]) die Interessen der Stakeholder am Zielsystem widerspiegeln. Die Summe solcher Perspektiven kann hinreichend sein, eine Architektur hinreichend zu beschreiben. [ISO94] schlägt Betrachtungen aus der Perspektive von *Functional*, *Code*, *Development*, *Concurrency*, *Physical*, *User action* und *Data view* vor. Oft wird eine etwas reduzierte Perspektivensicht vorgezogen. Entsprechend schlägt [HNS99] mit *Conceptual-*, *Module-*, *Code-* und *Execution View* vier Sichten vor.[43]
Darauf aufbauend beschreibt [Kru95] ein *4+1 View Model*, das folgende Perspektiven identifiziert:

Logische Sicht Die Logische Sicht reflektiert im Wesentlichen die funktionalen Anforderungen an das System. Dazu werden abstrakte Sichten auf die Funktionsweise des Zielsystems erstellt. Abhängig von dem gewählten Architekturmuster kann bspw. die in Abschnitt 2.12 beschriebene Zerlegung in kleine funktionale Einheiten eine solche Sicht darstellen.

Prozesssicht Die Verschaltung der in der Logischen Sicht identifizierten Komponenten wird in dieser Sicht betrachtet. Beispielhaft dafür sind die lang- und kurzlebigen Prozesse, die im Abschnitt 2.12 ausgeführt werden.

Entwicklungssicht Die Entwicklungssicht beschreibt das Zielsystem aus der Perspektive eines Softwareentwicklers. Im Fokus steht dabei die Softwarestrukturierung bspw. in Subsysteme, Module und Pakete sowie deren Interfaces.

Physikalische Sicht Diese Sicht beschreibt die involvierten physikalischen Einheiten (Hardware) und die auf ihnen abgebildeten Aspekte der Gesamtfunktionalität.

Szenarien Die Sicht auf Szenarien ist im von [Kru95] beschriebenen *4+1 View Model* weniger eine eigenständige Sichtweise, sondern eher eine sichtenübergreifende Betrachtung zur Identifikation von Kernkomponenten und zur Bewertung bzw. Verfeinerung der gewählten Architekturdefinition.

Abbildung 2.13 skizziert die unterschiedlichen Sichten der Auflistung und die vorgeschlagenen Architekturentwicklungsschritte nach [Kru95].

[43] vgl. [SNH95, S. 198ff], Architektursichten mit teils geschichtetem Abstraktionskonzept.

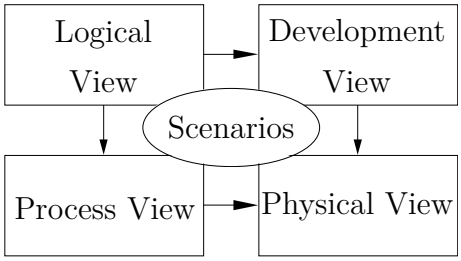

Abbildung 2.13: *4+1 View Model* nach [Kru95]

Die Auflistung weicht von den in [ISO94] propagierten Ansichten auf niedrigster Abstraktionsstufe ab (*Code-Sicht*). Diese Ebene ist in verteilten Systemen aufgrund der angestrebten Kapselung oft weniger strikt betrachtet als in der Entwicklung von monolithischer Software.

2.11 Architekturbeschreibungssprachen

Eine Architekturbeschreibungssprache bzw. Architecture Description Language (ADL) ist ein wichtiges Werkzeug zur Beschreibung von Architekturen mit Hilfe von formalen und semi-formalen Beschreibungen oder Diagrammen. Zielsetzung ist die Vermeidung von Defiziten hinsichtlich Konsistenz, Korrektheit und Vollständigkeit ([Gar00, S. 95]), die bei der Verwendung informeller Beschreibungen auftreten kann. Dabei sollen die Notationen einer ADL sowohl menschenlesbar als auch maschineninterpretierbar sein, um so die von dem Softwareentwickler definierte Beschreibung, bspw. in der maschinelle Verifikation oder Quellcodeerstellung, wiederzuverwenden. Formale Architekturbeschreibungen mit einer ADL sind typischerweise geeignet, bestimmte Aspekte einer Architektur analytisch zu bewerten.[Cle96, S. 21f] Durch deren formale Natur sind dies bspw. Konsistenz und Vollständigkeit.[44]

Eine bekannte ADL ist *WRIGHT*, die in [All97, S. 9ff] detailliert beschrieben und mit anderen Beschreibungssprachen verglichen wird.

In der Literatur wird kein De-facto-Standard einer formalen ADL reflektiert. Die resultierenden formalen Beschreibungen sind teils komplex und dadurch schlecht menschenlesbar. So ist die Transition der planerischen Intuition in die Beschreibung potenziell fehlerbehaftet. Zudem sind die Sprachen oft auf eine eng umrissene Klasse von Analysen ausgerichtet. Unter anderem fokussiert WRIGHT den Nachweis von Deadlock-Freiheit. Als Deadlock wird ein oft unerwünschter Zustand bezeichnet, in dem zwei Prozesse wechselseitig auf die

[44] vgl. Bewertung von *consistency* und *completeness* in [All97, S. 10] u. [Cle96, S. 21]

Freigabe von Ressourcen warten. Prozess 1 wartet auf die Freigabe von Ressource A und allokiert Ressource B. Prozess 2 wartet auf die Freigabe von Ressource B und allokiert gleichzeitig Ressource A. Die Prozesse sind gegenseitig „verklemmt“.[45]

Als weniger formale ADL wird oft die Unified Modeling Language (UML) eingesetzt. Per UML lassen sich wesentliche Struktur- und Verhaltensaspekte in der Softwaremodellierung darstellen. Insbesondere sind Benutzeraktivitäten, Rollen (u. a. Akteure) und logische Komponenten abbildbar. Darüber hinaus sind auch Datenhaltungsschemata sowie Prozesse modellierbar. Es bestehen weitere Konzepte wie bspw. Systems Modeling Language (SysML), die auf UML basieren und dessen Umfang erweitern.

Diese haben besonders in verteilten Systemen eine besondere Relevanz. Daher sind weitere Beschreibungsmechanismen entstanden, die versuchen, bestehende Geschäftsprozesse möglichst direkt zu modellieren. Eine bekannte Darstellungsart ist die Business Process Model and Notation (BPMN), die, vergleichbar mit den Aktivitätsdiagrammen in UML, spezielle Flussdiagramme für diesen Zweck verwendet.

BPMN stellt ebenfalls eine grafische Spezifikationsnotation dar. Sie basiert auf einer Reihe von Symbolen und Elementen zur Modellierung von (Geschäfts-)Prozessen, geprägt in [Obj11a]. Zielsetzung bei der Entwicklung von BPMN war die Schaffung einer leicht interpretierbaren Notation für Analysten von Geschäftsprozessen, um so die bestehende Lücke zwischen der Geschäftsprozessmodellierung und der Implementierung von Prozessen zu schließen. So lassen sich ereignisgesteuerte Prozessketten und Ablaufdiagramme darstellen. Die BPMN-modellierten Geschäftsprozesse sind gut menschenlesbar.

Auf Ausführungsebene wird oft die formalere Beschreibung per BPEL eingesetzt. BPEL nutzt XML, eine Notation zur strukturierten Datenbeschreibung, und lässt sich oft direkt aus der Modellierung in BPMN gewinnen. BPEL ist gängiges Werkzeug zur Orchestrierung von Diensten in einer SOA. Die Transition von der menschenlesbaren BPMN zur maschinenlesbaren BPEL ist ebenfalls im BPMN-Standard beschrieben.

2.12 Zerlegungskonzepte

Wie oben ausgeführt, gibt es keinen allgemeingültigen und evidenten Algorithmus, um funktionale Anforderungen in eine hinreichende Architektur zu überführen [Cle94, S. 404]. Ein adäquates Werkzeug zur Lösung dieser Problemstellung ist die Zerlegung der Gesamt-

[45] vgl. Deadlock-Freedom in [All97, S. 73ff]

funktionalität in Teilfunktionen. Die grundlegende Idee hierbei ist, im Sinne eines *divide and conquer*-Ansatzes, die Funktionalität in immer feinere Teilfunktionen aufzugliedern. So ist jeder Zerlegungsschritt mit einer Abstraktionsreduktion verbunden.

Gleichermaßen ist zu beschreiben, wie die identifizierten Funktionen untereinander verzahnt sind. Dabei hängt die Zerlegungsmethodik signifikant von dem angestrebten Architekturkonzept ab. Bei einer Objektorientierung wird versucht, die Gesamtfunktion in immer feinere Klassen bzw. Objekte zu zerteilen. Bei einer Architektur, die dem Paradigma *Pipes&Filter* folgt, wird eine Zerlegung in kleinste Modifikationseinheiten (*Filter*) angestrebt und deren Verschaltung (*Pipes*) spezifiziert.

Für die Zerlegung gibt es verschiedene strukturierte Ansätze. Beispielsweise schlägt [Par72] eine Zerlegung in Module vor und [BCB97] beschreibt *Attribute-Driven Design* als einen Zerlegungsansatz. [HKN+07, S. 113] beschreibt eine mit *analysis* eine Zerlegung, deren Ergbnisse durch *synthesis* eine konsolidierte Architekturlösung geformt werden.

Unter der Annahme, dass die zu modellierende Architektur eine dienstorientierte sein wird, in der komplexe Prozesse ablaufen und an deren Umsetzung verteilte Entwicklungsteams arbeiten, lassen sich adäquate Zerlegungsverfahren eingrenzen.

Zudem finden momentan Zerlegungsansätze Beachtung, die möglichst früh Feedback potenzieller Nutzer einbeziehen. Dies verspricht den Vorteil, dass die funktionale Zerlegung nah an der fachlichen Expertise der Endnutzer ausgerichtet ist. So besteht die Chance, dass die technische Architektur praktisch vorhandene Konzepte bzw. Geschäftsprozesse der Anwendungsdomäne gut reflektiert sowie das Risiko einer fundamentalen Abweichung und eine entsprechend späte Identifizierung vermieden werden.

Durch die direkte Einbindung von Anwendern besteht die Möglichkeit, dass diese aufgrund von Erfahrungen ggf. mit einem bestehenden System einen sehr ähnlichen Entwurf beschreiben. So kann die Festlegung auf eine grundsätzlich andere und möglicherweise in Aspekten bessere Lösung erschwert werden.

Im Folgenden wird daher ein von [Sch08] vorgeschlagener Ansatz zur Identifikation atomarer Dienste und deren verknüpfender Prozesse für eine dienstorientierte Architektur näher ausgeführt. Als Grundlage dient dazu ein fünfstufiges Schichtenmodell, s. Abb. 2.14.

Im Zerlegungsprozess werden die Schichten von oben nach unten durchlaufen und die entsprechend verfeinernden Teilaspekte beschrieben. Im letzten Schritt ist dann eine Übersicht über die atomaren Dienste erreicht, die die Funktionalität der überliegenden Schichten widerspiegelt.

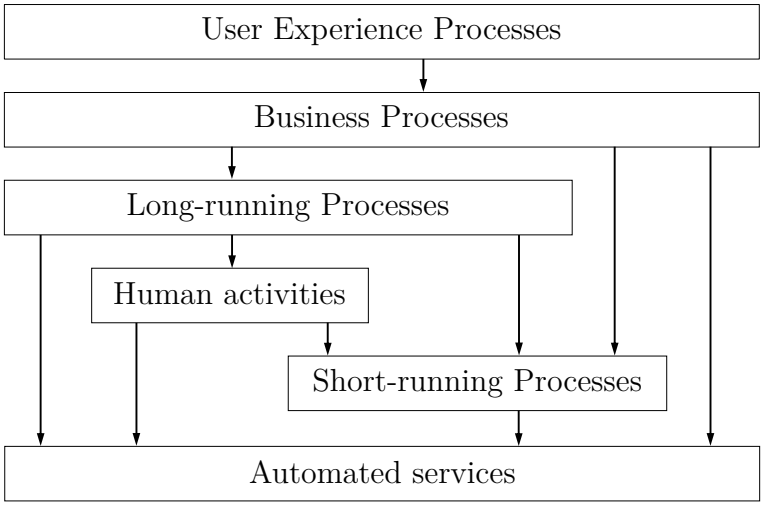

Abbildung 2.14: Schichtenmodell der funktionalen Zerlegung nach [Sch08]

Die Schichten fokussieren jeweils eine spezifische Abstraktion bzw. einen Teilaspekt der Funktionalität. Die folgende Liste beschreibt die Schichten kurz:

User Experience Processes Das Zielsystem Die User Experience Project (UEP)-Schicht repräsentiert die Benutzersicht auf das spätere System. Hier sind die Dialoge und Interaktionen eines Nutzers während einer Sitzung beschrieben. Interfaces in das System können ebenfalls in dieser Schicht beschrieben werden.

Business processes In dieser Schicht sind all die (Geschäfts-) Prozesse beschrieben, die von den Dialogen in der UEP Schicht gesteuert werden können.

Long-running processes In der Long Running Processes (LRP)-Schicht sind eben jene langlaufenden Prozesse beschrieben, die nicht sofort abgearbeitet werden können (vgl. SRP). Typischerweise sind dies Prozesse, in denen die Reaktion eines weiteren Systems abgewartet werden muss oder eine größere externe Operation durch einen Menschen im Sinne der *Human activities* durchgeführt wird. Oft wird ein LRP vom Benutzer initiiert, aber das Ablaufen des Prozesses wird nicht in der Benutzersitzung erwartet.

Human activities Diese Schicht beschreibt alle Prozesse, die von Personen vorgenommen werden. Dabei wird zwischen dem Benutzer der Applikation und dritten Personen unterschieden. Mit *Human activities* sind dritte Personen gemeint, bspw. wenn ein Techniker zu einer RSU gesandt wird, um vor Ort einen Reset durchzuführen.

Short-running processes Die Short Running Processes (SRP)-Schicht umfasst jene Prozesse, die praktisch sofort und ohne weitere (manuelle/externe) Abhängigkeiten abgearbeitet werden können. Die Ergebnisse des angestoßenen SRP sollten praktisch instantan vorliegen.

Automated services Diese Schicht umfasst atomare Prozessschritte, die funktional die Prozesse der überliegenden Schichten vollständig abbilden. Dies sind auch eben jene atomaren Dienste, die zum Aufbau der dienstorientierten Architektur notwendig sind.

Bei der Ausprägung der Prozessschichten ist zu beachten, dass nur Funktionen unterliegender Prozessschichten referenziert werden. Somit ist eine klare Hierarchie im Schichtenmodell gegeben. So bleibt der Zerlegungsprozess transparent und die unbeabsichtigten Effekte wie Rekursion werden vermieden.
Wird der in Abb. 2.14 beschriebene top-down-Zerlegungsprozess ausgeführt, sind als Ergebnis atomare Dienste (*Automated services*) identifiziert. Diese bilden eine gute Ausgangsbasis, um beim Implementationsschritt die Module oder Dienste anhand der Funktionalität aufzugliedern. Die darüber identifizierten Schichten der kurz- und langlaufenden Prozesse sowie die Interaktion dritter Personen in den Prozessen können herangezogen werden, um das Zusammenspiel der so identifizierten funktionalen Einheiten zu orchestrieren.

2.13 Bewertung von Architekturen

Vor der oft ressourcenintensiven Realisierung eines Systems ist die Qualität der gewählten Architektur zu beantworten, denn es ist kaum anzunehmen, dass bei einem ungeeigneten Architekturentwurf dennoch ein adäquates Zielsystem entstehen kann. Dabei sind in erster Linie funktionale Leistungsbewertungen[46] gängig[47], aber auch Zuverlässigkeit, Verfügbarkeit, Effizienz, Erweiterbarkeit und Wartbarkeit sind typische Qualitätsmerkmale, die bei dieser Betrachtung Beachtung finden. Sind die gewünschten qualitativen Eigenschaften in der Architektur abgesichert, ist das Risiko einer Abweichung von Anforderungen und Implementierung vermindert. Eine Nachbesserung der Architektur ist in diesem Entwicklungsschritt oft erheblich einfacher und kostengünstiger als im fertig ausimplementierten Zielsystem [GR08, S. 7].

[46] bspw. Abschätzungen, ob das Zielsystem eine entsprechend hohe Auslastung hinreichend handhabt, vgl. [SW01]
[47] z. B. in sim$^{\mathrm{TD}}$, s. [sim09a, S. 22, 67f]

Um die Bewertung von Architekturen durchzuführen, ist ein strukturiertes und nachvollziehbares Vorgehen vorteilhaft. In diesem Abschnitt werden verschiedene Möglichkeiten dargestellt, um den Architekturentwurf für ein Testfeld qualitativ zu bewerten.

Zunächst ist dazu der hier zentrale Qualitätsbegriff genau zu definieren. Aus verschiedenen Perspektiven betrachtet, kann Qualität bspw. in fünf Sichtweisen bewertet werden. So beschreibt [Gar84] eine transzendentale Betrachtung (*transcendent*), die zwar die Qualität eines Produktes erkennt, aber keine konkrete Quantifizierung definiert; eine Produktsicht (*product-based*), die Qualität als inhärente, quantifizierbare und bewertbare Eigenschaft eines Produktes versteht; eine Nutzersicht (*user-based*), die subjektiv Qualität als Eignung eines Produktes bzw. seiner Eigenschaften für den individuellen Einsatzzweck auffasst; eine Herstellersicht (*manufacturing-based*), die wesentlich auf die Erfüllung gegebener Spezifikationen ausgerichtet ist, und eine wertebasierte Sicht (*value-based*) auf die Korrelation zwischen individuellen Produkteigenschaften und dem dafür erzielbaren Erlös.

Im Rahmen dieser Arbeit sind insbesondere Produktsicht (*product-based*) und Nutzersicht (*user-based*) relevant, da diese eine quantifizierte Architekturbewertung ermöglichen und die Konformität mit den Bedürfnissen der Anwender fokussieren. Weniger relevant hingegen ist die Qualitätsauffassung der Herstellersicht und der wertbasierten Sicht, da diese Ansätze stark auf betriebswirtschaftliche Aspekte ausgerichtet sind, die hier nicht im Vordergrund stehen.

So sind aus der Produktsicht Qualitätsunterschiede zwischen zwei Produkten durch deren unterschiedliche Merkmale bestimmbar und auch messbar. Genauer definiert ISO/IEC 9126 in [DIN94] (Software-)Qualität als „*Gesamtheit der Merkmale eines Software-Produktes, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen*“ und versteht Qualitätsmerkmale als „*[...] eine Eigenschaft, die zur Unterscheidung von Produkten, Bausteinen oder Herstellungsprozessen in qualitativer (subjektiver) oder quantitativer (messbarer) Hinsicht herangezogen werden kann*“. Qualitätseigenschaften sind eine detailliertere Konkretisierung eines Qualitätsmerkmales.

[ISO01] definiert umfassend Qualitätsmerkmale aus den Bereichen Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit und Portierbarkeit. Abbildung 2.15 zeigt eine Übersicht der in ISO/IEC 9126 beschriebenen Qualitätsmerkmale.

Unabhängig von der gewählten Bewertungsmethodik sind daraus entsprechende Qualitätsmerkmale als Bewertungsgrundlage zu wählen und zu konkretisieren. Im Rahmen dieser Arbeit ist eine Objektivierung der Qualitätsmerkmale auf Grundlage der in Abschnitt 2.3 beschriebenen Verwendungszwecke für ein Testfeld gut durchführbar. So kann bspw. das

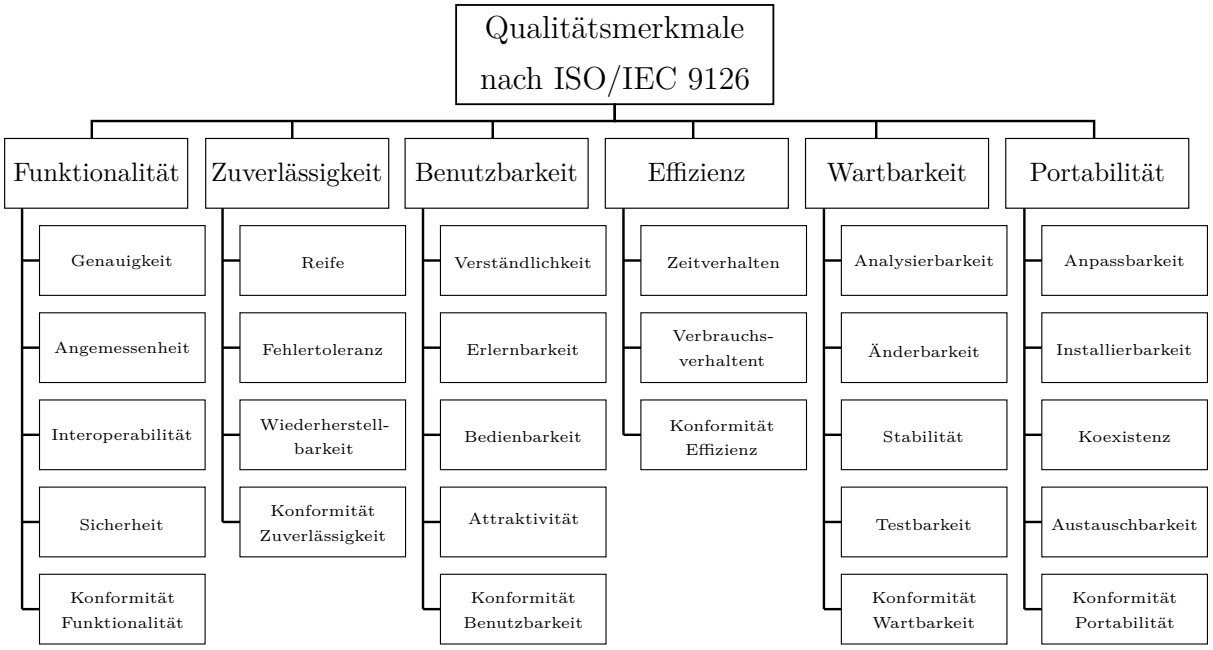

Abbildung 2.15: Qualitätsmerkmale nach ISO/IEC 9126

Qualitätsmerkmal Performanz mit einer konkreten Reaktionszeit des Systems oder einer maximalen Latenz beschrieben werden. Dazu bedarf es einer strukturierten Methode, die auf bestehende Bewertungsverfahren aufbaut. Diese werden im Folgenden skizziert.

[KABC96] schlägt Software Architecture Analysis Method (SAAM) als eine szenarienbasierte Architekturbewertung vor. SAAM ist eine gegenüber formaleren Bewertungsverfahren intuitive Herangehensweise, die zu untersuchenden Qualitätseigenschaften anhand von Nutzungsszenarien zu validieren. Dazu trifft der Softwarearchitekt auf ein Team von potenziellen Nutzern des geplanten Systems (bspw. Endbenutzer oder Kunden). In diesen Sitzungen sollen offene Fragen des Systemdesigns in den Nutzungsszenarien durch den Softwarearchitekten aufgedeckt und von den Nutzern beantwortet werden.

Ein Weiterentwicklung dieser Methode ist die Architecture Tradeoff Analysis Method (ATAM) ([KKC00]). Zielsetzung dieses Verfahrens ist ebenfalls die Analyse und Bewertung von Architekturentscheidungen auf Basis der gewählten Qualitätsmerkmale. Dazu wird mit dem *Quality Attribute Tree* eine Baumstruktur aufgebaut, die alle Konkretisierungen und deren initiale Bewertung (Priorität u. Realisierungsaufwand) umfasst. Auf dieser Grundlage sind die Risiken des Architekturentwurfs bewertbar. Darüber hinaus lassen sich kritische Punkte (*sensitive points*, [KKC00, S. 22f]) ausmachen, an denen spätere Änderungen weitreichende Folgen haben können. Zudem wird so herausgestellt, welche

Kompromisse (*trade-offs*, [KKC00, S. 23]) zur Realisierung gegenläufiger Anforderungen beim Architekturentwurf eingegangen wurden. Auch hier ist eine intensive Interaktion zwischen Architekt und fachlichen Experten als wichtiger Bestandteil vorgesehen.
Die von [LBvVB02] vorgeschlagene Methode Architecture-Level Modifiability Analysis (ALMA) betrachtet das isolierte Qualitätsmerkmal Modifizierbarkeit (*modifiability*) in unterschiedlichen Szenarien. Ziel von ALMA sind Abschätzungen zum Änderungs- und Wartungsaufwand sowie die Identifikation von Wirkfaktoren (*impact factors*), von denen diese Aufwände abhängen. So lassen sich Risiken des gewählten Architekturansatzes hinsichtlich der Modifizierbarkeit bewerten und vergleichen. Im Gegensatz zu SAAM und ALMA sieht ATAM keine direkte Interaktion zwischen Systemnutzern vor. Die fünf Prozessschritte werden nur von einem Auswertungsteam absolviert.
Aus SAAM entstanden neben ATAM und ALMA eine Reihe Derivate mit leicht abweichenden Schwerpunkten wie bspw. SAAMCS, SAAMER, FAAM und PASA.[48]
Es gibt eine Reihe weiterer Architekturbewertungsmethoden, wie bspw. ALPSM[49], die wie ALMA ein bestimmtes Qualitätsmerkmal bzw. eine Merkmalklasse (bspw. AQA[50]) fokussieren oder aber auf ein spezielles Zielsystem zugeschnitten sind, wie bspw. QSEM[51] zur Analyse skalierbarer Webanwendungen.
Die vorgestellten Methoden versuchen gleichermaßen die Architekturbeschreibung hinsichtlich gewählter Qualitätsmerkmale zu verbessern. Dabei setzen SAAM und ALMA auf die Interaktion mit Experten (bspw. Anwendern), um möglichst genaue qualitative Anforderungen an das Zielsystem zu erheben, wobei ALMA ein Evaluationsteam mit Experten in der Domäne einsetzt. In beiden Fällen ist der Erfolg des Verfahrens an die Güte der Interaktion von Architekten und fachlichen Experten gebunden.
ATAM scheint zwar im Gegensatz zu SAAM empfindlicher gegenüber ungünstig gewählten Szenarien, liefert aber transparente Risiko- und Kompromissbewertungen und führt auch zu einem besseren Verständnis der architekturrelevanten Anforderungen. Steht eine Untersuchung der Modifizierbarkeit einer Architektur im Vordergrund, so ist insbesondere das ALMA-Verfahren relevant. In der Praxis ist ebenfalls eine Kombination verschiedener Methoden denkbar, um sukzessiv die Güte des Architekturentwurfs abzusichern. Um

[48] Scenario-based Architecture Analysis Method of Complex Scenarios (SAAMCS), s. [LRv99]; Software Architecture Analysis for Evolution and Reusability (SAAMER), s. [LBKK97]; Family Architecture Assessment Method (FAAM), s. [Dol01]; Performance Assessment of Software Architectures (PASA), s. [Thi05]
[49] Architecture Level Prediction of Software Maintenance (ALPSM), s. [BB99]
[50] Architecture Quality Assessment (AQA), s. [HKL97]
[51] Quantitative Scalability Evaluation Method (QSEM), s. [WS05]

Tabelle 2.1: Auflistung bekannter Architekturbewertungsverfahren

Methode	Auswertungs-grundlage	Untersuchungs-gegenstand	Untersucht Beziehungen	Auswertungs-team
ALMA	Szenarien	Änderbarkeit	nein	ja
ALPSM	Szenarien	Wartbarkeit	nein	nein
AQA	Prototypen	Verständlichkeit, Attraktivität, Änderbarkeit	nein	nein
ATAM	Szenarien	mehrere QM	ja	nein
FAAM	Szenarien	Interoperabilität, Änderbarkeit	nein	ja
PASA	Szenarien	Zeitverhalten	nein	nein
SAAM	Szenarien	u. a. Modifizierbarkeit	nein	ja
SAAMCS	Szenarien	Flexibilität	nein	ja
SAAMER	Szenarien	Modifizierbarkeit	nein	ja
SBAR	Prototypen, math. Modelle, Simulation	mehrere Qualitätsmerkmale	ja	nein

möglichst adäquate Verfahren für die Fragestellungen zu wählen, bietet bspw. [DN02, S. 648ff] eine Übersicht und [EHM07, S. 14ff], [GR08, S. 18f] eine Taxonomie zur Bewertung von verschiedenen Methoden und einen Auswahlprozess an.

Ein weiterer Aspekt ist die wechselseitige Abhängigkeit von Qualitätsmerkmalen. So soll bspw. eine Flugzeughülle aus Sicherheitsgründen möglichst stabil sein, um die Insassen am besten zu schützen, gleichermaßen ist aus Effizienzgründen eine besonders leichte Hülle vorteilhaft. Solche Überlegungen werden insbesondere in ATAM berücksichtigt. Dadurch ist dieses Verfahren für solche Fragestellungen vorzuziehen.

Die geschickte Auswahl eines geeigneten Bewertungsverfahrens ist nicht trivial [EHM07, S. 1]. In Kapitel 4 werden die hier aufgeführten Bewertungsverfahren aufgegriffen und am konkreten Beispiel wird ein Verfahren in einem strukturierten Entscheidungsprozess ausgewählt.

In der Praxis ist nicht nur eine isolierte Architekturbewertung relevant, sondern es kann auch ein bewertender Vergleich zweier oder mehrerer Architekturen untereinander sinnvoll sein. Dies kann bspw. zur iterativen Verbesserung von Architekturaspekten oder für Kaufentscheidungen bestehender Produkte hilfreich sein [ABC+97, S. 6]. Für diese

besondere Art der Architekturbewertung gibt es zwei gängige Ansätze: The Software Architecture Comparison Analysis Method (SACAM)[52] und Domain-Specific Software Architecture Comparison Model (DoSAM)[53]. SACAM versucht, zwei unterschiedliche Architekturkonzepte auf einer gemeinsamen Abstraktionsebene zu beschreiben und führt dazu eine neue Architektursicht ein. In dieser Sicht wird aus ausgewählten Architektureigenschaften (bspw. Abhängigkeiten von Modulen) eine Bewertungsmetrik aufgebaut. Die jeweilige Metrik wird dann auf Eignung (*fitness*) bewertet. So gibt SACAM Aufschluss über die Tauglichkeit (*suitability*) der verglichenen Architekturentwürfe. Bemerkenswert ist, dass SACAM Architekturen mit sehr unterschiedlichen Anwendungsdomänen im Vergleich zulässt. Dies ist bspw. für integrative Architekturen für domänenüberspannende Systemlandschaften bedeutsam. Im Gegensatz zu SACAM beschränkt sich DoSAM auf Vergleiche von Architekturen aus einer gemeinsamen Domäne. Dazu baut DoSAM ein eigenes Domain Architecture Comparison Framework (DACF) aus fünf Komponenten auf. Kern des Frameworks ist die Modellierung der konzeptionellen Architektur, welche die Gegebenheiten und gewünschten Funktionen der Domäne abstrakt beschreibt. Für dieses Referenzarchitekturmodell werden Qualitätskriterien erfasst und priorisiert. Die eigentliche vergleichende Bewertung erfolgt über eine Abbildung (*mapping*) des konkreten Architekturentwurfs auf das Referenzmodell. In einem zweiphasigen Prozess wird dann über verschiedene Kriterien wie bspw. die Funktionsabdeckung und einen Einzelvergleich der zuvor identifizierten Qualitätsmerkmale ein Maß für die Eignung (*fitness*) ermittelt.

Die Bewertung von Architekturen ist eng verwandt mit Softwaremetriken. Letztere bewerten eine bestehende Software, i. d. R. nach der Implementierung, hinsichtlich genau definierter Aspekte. So sind bspw. die Metriken von Halstead[54] und McCabe[55] geeignet, um Komplexitätsbewertungen von Software durchzuführen und somit Rückschlüsse auf Wartbarkeit und Wiederverwendbarkeit zu ziehen. Zur Bewertung von Architekturen vor deren Implementierung sind solche Metriken nur bedingt geeignet, da sie nur auf bestehende Software anwendbar sind oder über eine Simulation einer Architekturmodellierung gewonnen werden können. Letztere ist nicht unproblematisch [MJB+12, S. 10]. Architekturbewertungen hingegen sind besonders gewinnend, wenn sie vor der Implementierung zur Optimierung der Architektur genutzt werden. Metriken können genutzt werden, die in der Anforderungsphase festgelegten Qualitätseigenschaften zu verifizieren,

[52] s. [SBV03]
[53] s. [BRST05]
[54] vgl. [Hal77]
[55] vgl. [McC76]

und bieten insbesondere in späteren Phasen des Software-Lebenszyklus eine gute Bewertungsgrundlage bspw. für Ausimplementierung und Weiterentwicklungen [GR08, S. 18f].[56] Das SBAR-Verfahren[57] ist ein Architekturbewertungsansatz, der erste Prototypen u. a. mit mathematischen Modellen und Simulationen auf Validität untersucht. SBAR sowie das oben beschriebene AQA sind somit für eine A-priori-Validierung von Architekturkonzepten weniger geeignet. Weiterhin bestehen hybride Ansätze, wie bspw. Palladio [RBH+16], die auf Grundlage ein Modellierung der Zielarchitektur werkzeuggetrieben Softwaremetriken aufbauen. Aufgrund des A-posteriori-Charakters werden im Rahmen dieser Arbeit Softwaremetriken nicht weiter verfolgt.

2.14 Architekturprozesse

Die zuvor beschriebenen Aspekte wie z. B. die Zerlegung in Funktionen und die Bewertung von Architekturen als Teilschritte der Architekturentwicklung sind eingebettet in einen Gesamtprozess. Oft ist dieser ein intuitiver Vorgang, der eher von bestehenden organisatorischen Strukturen geprägt wird[58] als von einer fachlich-methodischen Herangehensweise.

Die einzelnen Prozessschritte und deren Abfolge innerhalb eines solchen Gesamtprozesses hängen vom gewählten Vorgehensmodell ab.

Ein typisches Vorgehensmodell ist die Iterative Softwareentwicklung. Ihr Merkmal ist ein zyklisches Durchlaufen der vier Hauptprozessschritte Entwerfen, Umsetzen, Testen, Bewerten. Abb. 2.16 zeigt eine typische iterative Entwicklung mit entsprechenden Prozessschritten. Die Entwicklung startet mit einem Konzeptionsschritt, der die adressierte Problematik absteckt. Die darauffolgende Anforderungsanalyse sammelt und analysiert die Anforderungen, die an das zukünftige System gestellt werden. Sind die funktionalen und nicht-funktionalen Anforderungen bekannt, kann im nächsten Schritt der erste Entwurf einer Architektur durchgeführt werden. Dieser dient zur Entwicklung einer (ersten) Testversion. Diese hat ggf. noch nicht alle Funktionen, aber wesentliche Merkmale werden sichtbar und Schlüsseltechnologien eingesetzt. Hier beginnt der iterative Zyklus: Die Testversion wird versuchsweise ins Feld geführt, dort erprobt und die dabei identifizierten Mängel und Probleme zur Verbesserung einer neuen Testversion herangezogen. Im Prinzip

[56] z. B. COCOMO-Modell in [BAB+00]
[57] Scenario-based Software Architecture Reengineering (SBAR), s. [BB98]
[58] vgl. *Conway's Law* und Beziehung von Architektur und Organisationsstruktur in [BHS07, S. 202]

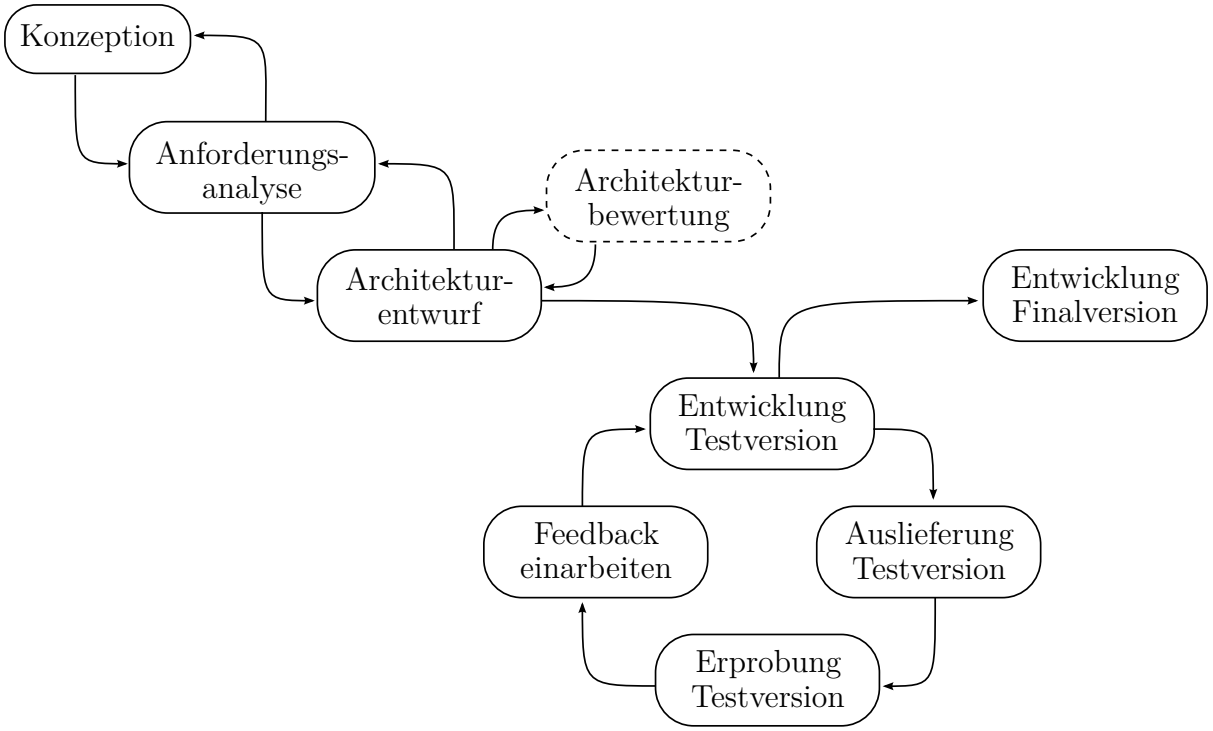

Abbildung 2.16: Architektur im iterativen Entwicklungszyklus angelehnt an [BCB97]

wiederholt sich dieser Zyklus, bis die Auswertung ergibt, dass eine befriedigende Reife des Prototypen erreicht ist. Es kann dann noch eine Reprogrammierung notwendig sein, um die gewünschte Finalversion zu gewinnen.

Der Großteil der Architekturdefinition findet im Prozessschritt zwischen der Anforderungsanalyse und der ersten Testversion statt. Dort wird aufbauend auf den funktionalen und nicht-funktionalen Anforderungen die technische Grundkonzeption erarbeitet und ein adäquater Architekturstil gewählt. Anschließend kann eine funktionale Zerlegung durchgeführt werden. Das Produkt sollte eine umfassende Architektur passend zu den zuvor analysierten Anforderungen ergeben. Die Qualität des Entwurfs kann auch durch eine anschließende Architekturbewertung abgeschlossen werden. In der erreichten Detailschärfe können die identifizierten funktionalen Komponenten feiner ausdefiniert und eine (Test-) Implementierung angestrebt werden.

In jedem Iterationsschritt kann auch die zugrunde liegende Architektur erneut bewertet werden. Obwohl bis dahin bereits viel Arbeit in die Ausimplementierung der ursprünglichen Idee geflossen ist, kann die bestehende Architektur verworfen und eine erneute

Implementierung mit adäquaterem Architekturstil eingeleitet werden. Aus wirtschaftlichen Gründen sollte dieser Schritt vermieden werden und die Architekturbewertung als Risikominimierung den ersten Entwurf absichern.
Der in diesem Abschnitt verwendete iterative Ansatz ist beispielhaft für andere Vorgehensmodelle. Es sind entsprechende Entwurfsschritte in den anderen Konzepten, wie bspw. im sequentiellen Wasserfallmodell, ebenfalls repräsentiert.

2.15 Zusammenfassung

Dieses Kapitel hat die Grundlagen beschrieben, die im Folgenden zur Testfeldarchitekturmethode und -bewertung aufgegriffen werden. Dazu wurden zunächst die domänenspezifischen Grundlagen ausgeführt, die eine Präzisierung der fachlichen Einheiten Testfeld und Testsystem in der ITS-Landschaft charakterisieren. Die Ausführung typischer Anwendungsszenarien und eine Analyse bestehender Testfeldarchitekturen bereiten den Methodenaufbau und das Bewertungsverfahren weiter vor.
Definition eines geeigneten Architekturprozesses für Testfelder sowie die Bewertung von Testfeldarchitekturen bedürfen softwaretechnischer Grundlagen. Dazu wurde Softwarearchitektur als notwendige und determinierende Voraussetzung für eine komplexe Testfeldsystemlandschaft ausgeführt. Adäquate Architekturkonzepte für verteilte Systeme, Architekturbeschreibungen und die dazu notwendigen Architekturbeschreibungssprachen wurden als notwendige Werkzeuge eines strukturierten Entwurfs erläutert. Als wesentliche Grundlage für den eigenen Ansatz wurden auch Konzepte zur funktionalen Zerlegung und Synthese dargestellt.

3 Testfeldarchitekturprozess

In den vorangehenden Abschnitten sind sowohl die domänenspezifischen als auch softwaretechnischen Grundlagen für den Aufbau und die Beschreibung einer Testfeldarchitektur ausgeführt worden. Im vorliegenden Kapitel werden diese zur Anwendung gebracht, um in einem systematischen Vorgehen eine Testfeldarchitektur aufzubauen. In diesem Abschnitt wird ein solcher Ansatz entwickelt und beschrieben. Angelehnt an die Architekturprozesse in Abschnitt 2.14 werden die relevanten anwendungsspezifischen und fachlichen Grundlagen aus dem vorangehenden Kapitel aufgegriffen und sukzessiv verfeinert, um von den zuvor grob umrissenen Anforderungen für ein solches Testfeld zu einer genaueren Sicht zu gelangen. Darauf aufbauend wird eine Zerlegung der Gesamtfunktionalität durchgeführt und im folgenden Syntheseschritt zu einer Testfeldarchitektur aufgebaut. Diese wird mit den in Kapitel 2 beschriebenen Verfahren dargestellt. Die hier angewandte Methode und die daraus abgeleitete Struktur wird in Abschnitt 3.2 erörtert.

3.1 Formalisierung

Damit der Testfeldarchitekturprozess begonnen werden kann, muss zum einen geklärt sein, welche Anforderungen an die Formalisierung gestellt werden, da dies den folgenden Entwurfsprozess in Schritten und Werkzeugen maßgeblich prägt. Zum anderen müssen funktionale und nicht-funktionale Anforderungen an das Zielsystem vorliegen. Dieser Abschnitt baut eine exemplarische Konfiguration auf.
Ob ein Architekturentwurf formal spezifiziert werden soll, oder ob eine semi-formale oder nicht-formale Spezifikation hinreichend ist, hängt vom jeweiligen Anwendungsfall ab. Bei einem Testfeld ist dies an der fachlichen Ausrichtung auszumachen: Im Rahmen dieser Arbeit wird angenommen, dass bei der In-situ-Erprobung sicherheitskritischer Testsysteme der Aspekt der formalen Verifikation bereits abgeschlossen ist, da typischerweise ein solcher Nachweis nicht unter Realbedingungen im FOT experimentell erbracht wird.
Hinsichtlich der funktionalen und nicht-funktionalen Anforderungen stützt sich der Archi-

tekturentwurf auf die in Kapitel 2 ausgeführten domänenspezifischen Grundlagen. Obwohl die Anforderungsanalyse für einen konkreten Entwurf weder umfassend noch hinreichend präzise ausgeführt ist, ist diese dennoch ausreichend, um ein methodisches Vorgehen über funktionale Zerlegung und Synthese zumindest an Teilaspekten auszuführen. Durch die Fokussierung auf den methodischen Architekturentwurf ist eine umfassende hypothetische Architekturdefinition in aller Vollständigkeit im Rahmen dieser Arbeit nicht sinnvoll.

An dieser Stelle sind auch insbesondere die nicht-funktionalen Anforderungen an ein Testfeld zu betrachten. Diese haben fundamentalen Einfluss auf die Architektur, wie in Kapitel 4 dargestellt wird.

Im Folgenden wird daher ein zentraler nicht-funktionaler Aspekt herausgegriffen, an dem die Einflüsse auf den Entwurf gezeigt werden können. Ein wesentlicher Aspekt kann die Flexibilität sein, wie in der Motivation ausgeführt. Das Testfeld soll bspw. für verschiedene Versuchsaufbauten wiederverwendet werden. Zudem müssen auch verschiedenste technische Komponenten in den Versuchsaufbau integrierbar sein. Dies schließt auch zukünftige Technologien ein. Daher ergeben sich aus praktisch allen nicht-funktionalen Anforderungen gewisse Konsequenzen für den Aufbau der Architektur.[1]

Als Beispiel kann der nicht-funktionale Aspekt Flexibilität in drei unterschiedlichen Ebenen im Architekturentwurf reflektiert werden. Es kann eine Flexibilität auf der Interaktionsebene angestrebt werden. Für ein Testfeld würde das bedeuten, dass bspw. eine offene Kommunikationsarchitektur gesucht wird, die Kommunikationsverbindungen nicht vorgibt. Bei einem Webbrowser ist dies in der frei wählbaren Anfrage (URL[2]) an Webserver realisiert. Auf Ebene der Softwarekomponenten kann Flexibilität durch Einbettung von neuen bzw. geänderten Softwarekomponenten in die bestehende Systemlandschaft erfolgen. Dies ist ein typischer Vorgang in einem Testfeld, da unterschiedliche und auch kontinuierlich wechselnde Testsysteme erprobt werden. Im Beispiel des Browsers werden diese Aspekte typischerweise neben integrierten Skriptsprachen durch ein Plug-In-System unterstützt. Auf oberster Ebene sind auch flexible Architekturen möglich. [CdlFBS01] definiert eine *dynamic architecture*. Obschon Terminologie und Definition von dynamischen Architekturkonzepten in der Literatur umstritten sind[3], folgen die Überlegungen aus ITS Test Beds, mittels frei kombinierbaren Komponenten praktisch beliebige Kompositionen darzustellen, diesem

[1] vgl. Einbindung von Altsystem in [MHC05]

[2] Uniform Resource Locator (URL) ist eine Zeichenkette zur Adressierung von Quellen, wie Webservern, im Internet.

[3] Bspw. zeigt [Per08] die Vorteile einer evolutionären Architekturanpassung, benennt aber auch die damit verbundenen Risiken z.B. für langlaufende Produktlinien.

Ansatz. Die Abwägung der Flexibilität ist ein wichtiger Entscheidungsschritt und hat kardinale Auswirkungen auf die restliche Architekturdefinition und wird ebenfalls in der Bewertung von Architekturen im folgenden Kapitel 4 aufgegriffen.
Im Folgenden werden daher wenige grundsätzliche Annahmen und (nicht-)funktionale Aspekte auf Basis der zuvor beschriebenen fachlichen Grundlagen (Kapitel 2) festgelegt, die unabhängig von den konkreten Anforderungen an ein bestimmtes Testfeld als allgemeingültig angenommen werden dürfen:

1. Das zu gestaltende Testfeld soll die in Abschnitt 2.2 ausgeführte Klasse von kooperativen Testsystemen mit in Abschnitt 2.3 umrissenen Funktionalität unterstützen.

2. Aus der Betrachtung bestehender Architekturen, wie bspw. dem Aufbau von CVIS, ist ableitbar, dass drei wesentliche Einheiten (OBU, RSU und Zentraleinheit, vgl. Abschnitt 2.2), je nach Ausrichtung der Kampagne, im Testfeld unterstützt werden müssen. Diese Komponenten müssen teils überwacht oder emuliert werden.

3. Dem entsprechend kann das Testfeld (Teil-)Funktionen des Testsystems nachbilden, so dass die Funktion virtuell zur Verfügung stehen. Beispielsweise spielt die Kommunikation der verteilten Komponenten untereinander eine wichtige Rolle. Es kann daher erforderlich sein, die Kommunikation in einem jeweils adäquaten Umfang zu simulieren.

4. Die Komponenten der Testsysteme und somit auch die angrenzenden Komponenten des Testfeldes selbst müssen austauschbar sein. Bereits vorhandene Systeme sollen migrierbar und zukünftige anbindbar sein.

5. Das Testfeld soll Testsystem im Verhalten beobachtbar machen. Im Beispiel Kommunikation führt dies zur der Anforderung einer transparenten und nachvollziehbaren Aufzeichnung, die eine entsprechende Analyse ermöglichen.

Auf Grundlage dieser Festlegung kann dann die Methodik zum Architekturentwurf konkretisiert werden. Dies wird im nächsten Abschnitt ausgeführt.

3.2 Methodik

Wie in den Grundlagen ausgeführt, beschreibt das von [Kru95] vorgeschlagene *4+1 View Model* eine Softwarearchitektur im Allgemeinen hinreichend. Um eine Testfeldarchitektur

festzulegen, ist daher eine sorgfältige Beschreibung gemäß den *4+1-Sichten* de facto Ziel und Ergebnis des Testfeldarchitekturprozesses. [Kru95] gibt jedoch keinen konkreten Vorschlag, wie diese Sichten systematisch entwickelt werden sollen.

Um diese Problemstellung zu lösen, bietet sich die in Abschnitt 2.12 dargestellte Dekomposition nach [Sch08] an. Anhand dieser kann die intendierte Gesamtfunktionalität in atomare Funktionen und deren Verschaltung (Prozesse) zerlegt werden. Die resultierende Beschreibung der Ergebnisse entspricht genau den *viewpoints* ‚Logische Sicht' und der ‚Prozesssicht' im *4+1 View Model*. Zur Durchführung der Dekomposition werden u. a. Benutzerinteraktionen mit dem Zielsystem beschrieben, aus denen Funktionen und Prozesse gewonnen werden. Diese Beschreibung entspricht den von [Kru95] vorgeschlagenen Szenarien. Über die Kombination von atomaren Funktionen mittels der isolierten Prozesse ist nun eine Synthese der gewünschten Funktionalität herleitbar. Dieser Syntheseschritt formt größere Funktionseinheiten, die softwareseitig in geeigneten Strukturen aufgebaut werden können. Die Definition dieser Strukturen bzw. deren Darstellung stellt den *viewpoint* ‚Entwicklungssicht' dar. Die Komposition dieser Strukturen auf (Hardware-) Einheiten bildet die ‚Physikalische Sicht' im Architekturentwurf. Beide Sichten werden in den korrespondierenden Abschnitten (3.7 u. 3.8) aufgebaut.

In der konkreten Umsetzung dieser Arbeit wird zunächst eine Top-down-Herangehensweise verfolgt. Über die Anwendungsszenarien eines Testfelds (Abschnitt 3.3) lassen sich konkrete Rollen im Testprozess definieren (Abschnitt 3.4). Aus dieser Betrachtung kann der *User Experience Process* nach [Sch08] als initialer Zerlegungsschritt durchgeführt werden (Abschnitt 3.5) und schließt so die sichtenübergreifende Betrachtung zur Identifikation von Kernkomponenten (Szenariensicht im *4+1 View Model*) sowie die Reflexion über deren wesentliche funktionale Anforderungen (Logische Sicht im *4+1 View Model*) ab.

Die so identifizierten atomaren funktionalen Einheiten können dann zu den zuvor definierten Prozessen (UEP) kombiniert werden. Diese Zerlegung in Testfeldprozesse ist in Abschnitt 3.6 ausgeführt und liefert als Ergebnis eine Testfeldprozessbeschreibung, die der Prozesssicht gemäß *4+1 View Model* entspricht.

Nach der vorangehenden Zerlegung in kleinste funktionale Einheiten und deren Verschaltung in UEP kann die Synthese analog zu [Sch08] erfolgen. Gemäß [Kru95] stützt sich die Herleitung der *viewpoints* Entwicklungssicht und Physikalische Sicht auf die zuvor entwickelten Sichten. Dem folgend wird im Abschnitt 3.7 zunächst die Komposition von atomaren Funktionen und den dazugehörigen Prozessen in softwaretechnischen Einheiten

beschrieben. Die Architekturentwicklung wird in Abschnitt 3.8 mit der adäquaten Verteilung der zuvor entwickelten (Software-)Komponenten und Prozessen auf physikalische Einheiten in der Testfeldlandschaft abgeschlossen.

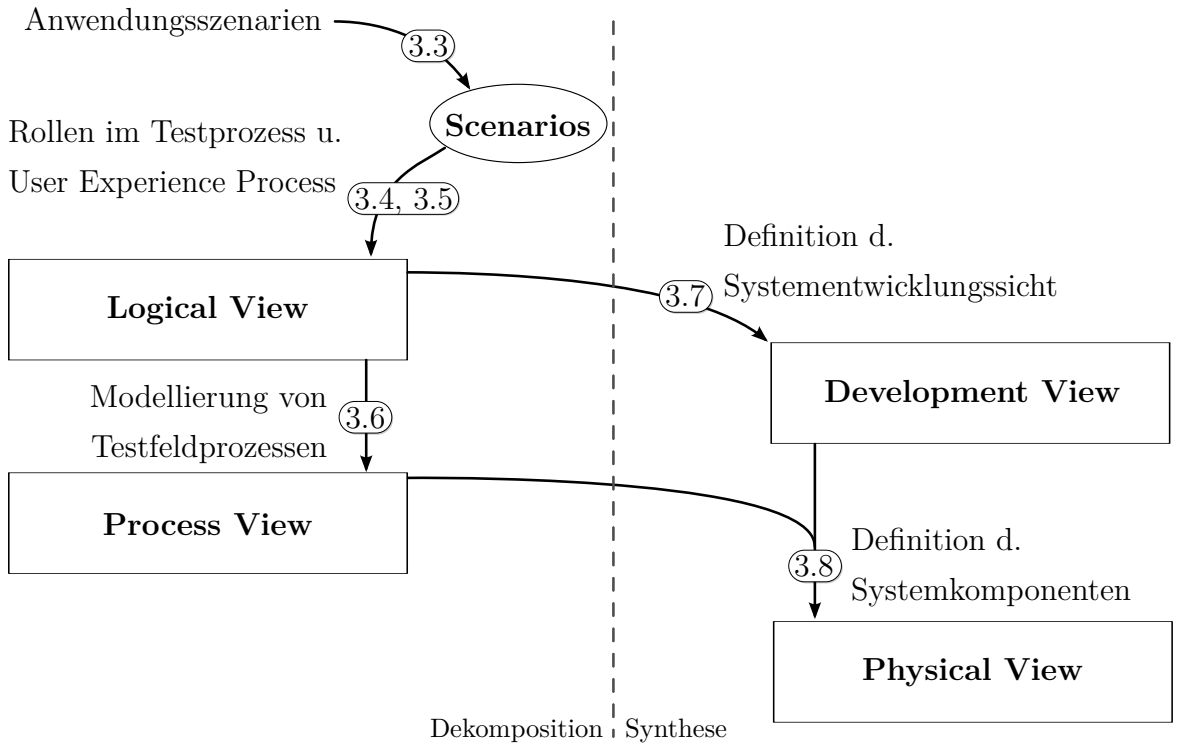

Abbildung 3.1: Architekturentwurfsschritte: mittels Dekomposition und Synthese zum *4+1 View Model*; mit Referenzen auf die entsprechenden Kapitel

In Summe fügen sich die so ermittelten *viewpoints* zu einer umfassenden Architekturbeschreibung zusammen. Das beschriebene Vorgehen im Entwurfsprozess ist mit den korrespondierenden Abschnitten dieser Arbeit in Abb. 3.1 anhand des *4+1 View Model* illustriert. Dabei folgt der Entwurfsprozess der ursprünglich von [Kru95] vorgeschlagenen Reihenfolge der Entwicklungsschritte, wie in Abb. 2.13 dargestellt.
Aus qualitativer Sicht ist durch diesen Ansatz sichergestellt, dass bei sorgfältiger Umsetzung von Dekomposition und Synthese die Architektur den zuvor abgesteckten Anforderungen genügt und die gewünschte Funktionalität umfasst. Da dieser Ansatz auf der Zerlegung bzw. Rekombination von funktionalen Aspekten aufsetzt, ist die Konformität insbesondere mit nicht-funktionalen Anforderungen wie bspw. Interoperabilität, Austauschbarkeit oder Zuverlässigkeit dadurch naturgemäß nicht oder nur bedingt garantiert. Diese Qualitätseigenschaften bedürfen einer gesonderten Validierung, die in Kapitel 4 genauer betrachtet wird.

3.3 Anwendungsszenarien

In Abschnitt 2.10 wird die Untersuchung und Spezifikation von Anwendungsszenarien als ein sinnvoller Grundstein für das weitere Vorgehen beschrieben. Dazu wird im Folgenden anhand von Szenarien die grundlegende Funktion eines Testfeldes aus top-down-Perspektive betrachtet. Analog zur klaren Trennung von Testfeld und Testsystem ist eine strikte Differenzierung von Szenarien des Testsystems und denen des Testfeldes sinnvoll. Die Testsystemszenarien sind ein wichtiger Punkt im Entwicklungsprozess von ITS-Applikationen und daher von großer Bedeutung. In diesem Abschnitt werden jedoch zunächst reine Anwendungsfälle des Testfelds betrachtet, wohingegen Testsystemszenarien Untersuchungsgegenstand im Entwicklungszyklus des Testsystems und somit als gewidmeter Prozessschritt im FESTA-V-Modell herausgestellt sind. Später in Kapitel 4 werden im Rahmen von Architekturbewertungen bspw. die flexiblen Anpassungen des Testfelds an unterschiedliche Szenarien innerhalb des Testsystems gesondert aufgegriffen.
Als gute Vorlage zur Festlegung von typischen Anwendungsfällen können die in Abschnitt 2.3 aufgeführten Funktionen des Testfeldes dienen. Um die dort aufgelisteten Funktionen in möglichst überdeckungsfreie Anwendungsfälle zu überführen, müssen die Funktionen auf Gemeinsamkeiten bzw. besondere Eigenarten untersucht werden.

Tabelle 3.1: Gegenüberstellung von Anwendungszwecken und Prozessschritten im Testfeld

Funktion \ **Prozesschritt**	Func. Ident. & Description	Use Cases	Research Quest. & H.	Performance Indicators	Study Design	Measures & Sensors	Data Acquisition	Database	Data Analysis	Research Q. & H. Analysis	System & Function Analysis	S/E. Impact Assessment
Entwicklung		■		■	■	■	■	■	■		■	■
Erprobung	■	■	■	■	■	■	■	■	■	■	■	■
Referenzplattform		■		■	■	■	■	■	■		■	
NDS	■	■	■	■	■	■	■	■	■	■		
Analyseplattform								■	■	■	■	■
Demonstration											■	■

Wie in Abschnitt 2.3 gezeigt wurde, ist es eine Aufgabe des Testfeldes, den Entwicklungs-

prozess zu unterstützen. Abgesehen von den hier nicht vertieften Zertifizierungsaspekten fungiert ein Testfeld typischerweise zur Erprobung, als Referenzplattform, NDS, Analyseplattform und zur Demonstration von Testsystemen. Um zu klären, inwieweit diese Funktionen artverwandt sind, lassen sich die Prozessschritte aus dem V-Modell aus FESTA als gekapselte Teilfunktionen auffassen. Jede Testfeldfunktion wird durch einen bzw. mehrere dieser Teilprozesse dargestellt. Diese Assoziation ist in Tabelle 3.1 genauer ausgeführt. Wie zu erwarten, hat dabei die Erprobung größte Unterstützung durch Teilfunktionen, da die Richtlinien aus FESTA genau auf diesen Testfeldzweck fokussieren. Unter der Prämisse, die dargestellten Prozessschritte sind allgemeingültig, bilden Entwicklung, Referenzplattform, NDS, Analyseplattform und Demonstration Spezialisierungen der Erprobung.

Somit sollte eine Betrachtung der Gesamtfunktionalität des Erprobungsprozesses hinreichend sein, um die abhängigen Spezialisierungen mit zu berücksichtigen. Daher wird die Erprobung, im Sinn der in FESTA definierten Prozesskette, für die weiteren Überlegungen das rahmengebende Szenario stellen.

Die Szenariobeschreibung im Architekturentwurf ist oft natürlichsprachlich. Etwas formaler lassen sich Szenarien mit UML-Diagrammen beschreiben. Dazu sind bspw. Anwendungsfalldiagramme (Use-Case-Diagramme) gut geeignet. Ein solches Diagramm wird später für die zu unterstützenden Prozesse im gewählten Szenario in Abschnitt 3.5 strukturiert aufgebaut. Nach Festlegung auf die Erprobung als universellen Anwendungsfall, der andere wesentliche Szenarien des Testfelds, wie Referenzplattform, NDS und Analyseplattform umfasst, kann dieser nun in feineren Funktionen beschrieben werden. Um so umfassende *User Experience Processes* auszuprägen, müssen jedoch die Rollen im Zielsystem klar definiert sein. Diesem Aspekt wendet sich der folgende Abschnitt zu.

3.4 Rollen im Testprozess

In diesem Abschnitt wird ausgehend von den anwendungsspezifischen Grundlagen eine Verfeinerung der Anforderungen dargestellt. Dazu werden die im Betrieb eines Testfelds bestehenden Benutzerrollen identifiziert und für diese eine funktionale Zerlegung ausgehend von *User Experience Processes* durchgeführt, wie in Abschnitt 2.12 ausgeführt. Ohne eine klare Vorstellung von den Rollen bleibt es bspw. vage, wer letztendlich von der technischen Unterstützung profitiert und an welchen Bedürfnissen gewisse Funktionen ausgerichtet werden.

In der Praxis gibt es eine Vielzahl an Rollen, die teils wesentlichen Einfluss auf die Gestaltung eines Testfelds und der dort etablierten Prozesse haben. Dies sind bspw. Personen/Rollen hinsichtlich Datenschutz, IT-Sicherheit oder Ethikfragen, aber auch öffentliche Hand oder Bürgerinteressen. Der Fokus dieser Arbeit liegt jedoch wesentlich auf der Softwaregestaltung von Rollen, die mit dem Testfeld direkt interagieren.

Klare Benutzerrollen im Testfeld sind in der Literatur kaum identifiziert. Obschon bspw. [FES17] wesentlich die Rolle der Probanden in Feldtests behandelt, sind diese jedoch Rollen im Testsystem und nicht im Testfeld. Ebenso fokussieren bspw. CVIS und sim$^{\text{TD}}$ wesentlich die Benutzerrollen innerhalb des Testsystems. Diese werden in der Konzeptionsphase eines Testsystems untersucht und stellen einen wichtigen Betrachtungsgegenstand innerhalb eines Testsystementwurfs dar. Auf primärer Betrachtungsebene der Benutzerrollen eines Testfelds sind sie von untergeordneter Relevanz. In [BVJ+11, S. 96] werden vier grob umrissene Benutzerrollen unterschieden. Dies sind *Developer*, *Researcher*, *Customer* sowie *TASC Operator*. [FES17, S. 179] identifiziert hingegen zwölf Rollen. Diese nicht genauer spezifizierten Rollen sind jedoch stark an den typischen Organisationsstrukturen für Forschungsprojekte ausgerichtet. Rollen wie *Project Steering Committee* oder *Project Management Team* scheinen in einem Testfeld keine direkte Repräsentation zu finden. Daher kann dieses Rollenmodell nicht maßgeblich sein und ermöglicht keine bijektive Projektion auf die vier Rollen aus [BVJ+11].

Weder [FES17] noch die Planungen bzw. Erfahrungen aus den Aufbauten als CVIS und sim$^{\text{TD}}$ konkretisieren Überlegungen hinsichtlich Datenschutz und Privatsphäre. Darüber hinaus sind ethische Fragestellungen und die Einbindung von Bürgern und öffentlicher Hand nicht Bestandteil des Rollenverständnisses. Obwohl dies erhebliche Auswirkungen auf die Gestaltung der Architektur haben kann ([FKL16, S. 122f]), beschränkt sich die nachfolgende Betrachtung auf die Rollen, die aktiv am Betrieb des Testfelds teilhaben.

Um die signifikanten Rollen für ein Testfeld zu verfeinern, kann auf Grundlage der in [FES17] definierten Prozessschritte (vgl. Abschnitt 2.4 u. Abb. 2.3) untersucht werden, welche Personengruppen und insbesondere welche Kompetenzen in diesen Prozessen involviert sind. Im Einklang zu den in [BVJ+11] umrissenen Benutzerrollen lassen sich vier klare Kompetenzfelder (Technik, Wissenschaft, Organisation und Anwendung) unterscheiden. Tabelle A.4 (im Anhang, S. 179) identifiziert grob wesentliche Tätigkeiten mit entsprechendem Sachverstand für die Prozessschritte aus Abb. 2.3. Darüber hinaus identifiziert die Tabelle wichtige prozessschritt- und kompetenzübergreifende Schnittstellen an benachbarten Tabellenzellen.

In Abschnitt 2.4 wurde die hinreichende Abdeckung der FESTA-Methodologie für die in Abschnitt 2.3 ausgeführten Anwendungszwecke eines Testfelds dargestellt. Somit ist die Identifikation der Kompetenzen innerhalb der Prozessschritte für abweichende Testfeldausrichtungen gültig.
Die Aufgaben lassen sich feiner aufgliedern als in Tabelle A.4 dargestellt. Dadurch kann genau zwischen zwei verschiedenen Kompetenzen unterschieden werden. So ist bspw. die Einrüstung der Instrumentation im Testfeld sowie die Modellierung der Datenhaltung beides eine technische Kompetenz. In der Praxis werden unterschiedliche Personen involviert sein. Dies kann bei der Präzisierung der Rollen berücksichtigt werden. Im Rahmen dieser Arbeit ist eine Unterscheidung der vier wesentlichen Kompetenzen hinreichend, um daraus, angelehnt an [BVJ+11] und in Kombination mit Versuchsteilnehmern, fünf klare Benutzerrollen zu unterscheiden:

Techniker Diese Benutzerrolle umfasst alle technischen Aufgaben wie die Einrüstung von spezieller Infrastruktur und die technische Handhabung des Versuchsablaufs. Ebenso zählt dazu der technische Aspekt der fachlichen Forschungsfrage. Die Technikerrolle soll bspw. in der Konzeption des Feldversuchs dessen technische Umsetzbarkeit untersuchen.

Wissenschaftler Je nach Ausrichtung können im Feldversuch verschiedene Aspekte untersucht werden. Die Rolle des Wissenschaftlers ist daher die Verknüpfung der fachlichen Fragestellungen eines Interessenten mit den technischen Mitteln eines Testfelds. Die Ausgestaltung eigener Forschungsfragen und die Versuchsauswertung sind Hauptaufgaben dieser Rolle.

Organisator Die Durchführung der Kampagne ist wesentliche Aufgabe dieser Rolle. Dazu werden die drei anderen Rollen koordiniert.

Interessent Für eine Kampagne in einem Testfeld kann es einen Interessenten geben, der, losgelöst von einer streng wissenschaftlichen Untersuchung, das Testfeld als Forschungsanlage zur Beantwortung wichtiger Fragen bspw. zur technischen Umsetzbarkeit oder Rentabilität auffasst. Gleichermaßen bringt ein Interessent ggf. spezielles Fachwissen ein und spezifiziert anfangs die zu testende Funktion und initiiert die Ausprägung der Forschungsfrage.

Nach Abschluss der Kampagne sollten die gewonnenen Erkenntnisse die ursprünglichen Fragen des Interessenten beantworten.

Proband Im gewählten Anwendungsszenario Entwicklung steht oft der Anwender, dem Systemfunktionen bereitgestellt werden, im Mittelpunkt. In diesem Szenario können Teile der Funktion vom Testfeld bereitgestellt und mittels Probanden erprobt werden. In anderen Anwendungsszenarien wie bspw. NDS interagiert ein Proband ggf. direkt mit Komponenten des Testfeldes, bspw. um Feedback über ein Systemverhalten zu geben. Daher ist die Rolle des Probanden zumindest in einigen Anwendungsfällen relevant.

Die Aufteilung der Benutzerrollen korrespondiert dabei nicht unbedingt mit realen Personen. In der Praxis können mehrere Rollen von einer Person bekleidet werden oder, in großen Versuchsaufbauten wie bspw. sim^{TD}, können mehrere Personen gewisse Teilaspekte einer Rolle abdecken. Die oben beschriebene Einteilung kann dabei als gröbste Teilung gelten, eine Verschmelzung von diesen Rollen scheint wenig plausibel. Eine feinere Unterteilung, wie in [FES17, S. 179] vorgeschlagen, ist aus methodischer Sicht möglich. Allerdings lassen sich aus einer feineren Rolleneinteilung für keine zusätzlichen Aspekte für die Generalisierung der Methodik gewinnen.
Ausgehend von der Rollenfestlegung in diesem Abschnitt können für die entsprechenden Benutzergruppen die typischen Aktivitäten (Geschäftsprozesse) identifiziert werden, die ein Testfeld unterstützen soll. Dies wird im folgenden Abschnitt zu UEP weiter ausgeführt.

3.5 User Experience Process

Als initialer Schritt zur Zerlegung der Gesamtfunktionalität eines Testfeldes wird eine Untersuchung der User Experience Processes angestrebt, wie in Abschnitt 2.12 ausgeführt. Dazu werden die bestehenden Geschäftsprozesse der Benutzerrollen genauer beschrieben. Diese sind a priori nicht evident, da Testfelder kein Standardwerkzeug der Domäne darstellen und somit keine etablierten Prozesse vorliegen. Die Literatur bietet in der Aufteilung der Aktivitäten im FOTIP (vgl. Abb. 2.4 u. [FES17, S. 180ff]) eine gute Ausgangsbasis. Aus diesen können nun jene gewählt werden, die durch das Testfeld unterstützt werden sollen und ggf. mit weiteren Prozessen angereichert werden.
Die FOTIP-Aktivitäten 1 bis 6 sind nicht technisch ausgerichtet und bedürfen so keiner gesonderten technischen Unterstützung durch ein Testfeld. Die Architektur aus ITS Test Beds berücksichtigt insbesondere die Handhabung der Infrastruktur. Daher sind die Aktivitäten 7 bis 10, die sich ebenfalls mit der Auswahl und Bereitstellung befassen, hier zu berücksichtigen. Aktivität 11 (*Equip vehicles with technologies*) und 14 (*Test all systems*

to be used according to specifications), also die Bereitstellung von korrekter Infrastruktur für einen Testlauf, korrespondiert ebenfalls mit den Mechanismen des ITS-Test-Beds-Architekturkonzepts *Tests*. Ein Feedback-Kanal für den Probanden *Implement driver feedback & reporting systems (12)* ist weder in sim$^{\text{TD}}$ noch in ITS Test Beds vorgesehen. Dennoch scheint dies Konzept nach [FES17] sinnvoll und wird daher berücksichtigt. Die Erfassung der Daten wird eine zentrale Rolle spielen, daher ist die Berücksichtigung von Aktivität 13 evident. Die Aktivitäten 15 und 16 sind entweder entkoppelt vom Testfeld oder können mit der vorhandenen testfeldeigenen Instrumentationen vorangetrieben werden, wie beispielsweise die Demonstration einer Assistenz zur Fahrerschulung. Daher haben diese Aktivitäten durchaus Relevanz, werden aber im Architekturentwurf nicht gesondert betrachtet. Die Unterstützung der verbleibenden Aktivitäten, von den Vorversuchen (17) bis zum Rückbau des Versuchs (22), gehört zu den maßgeblichen Aufgaben eines Testfelds.

Die so ermittelten Aktivitäten korrespondieren gut mit den Bausteinen der Architektur aus ITS Test Beds. Marginal abweichend werden u.a. Pre-Processing und Datenstrommanagement dort als eigene Bausteine aufgefasst, während der FOTIP keine detaillierte Aktivität diesem Aspekt widmet. Die in ITS Test Beds beschriebene Methodik, insbesondere der Testprozess [BVJ+11, S. 61], wird repräsentiert. Die Betrachtungen der sim$^{\text{TD}}$-Architekturüberlegungen und auch der in [sim10] beschriebenen Methodik decken keine zusätzlichen Aktivitäten auf.

Somit kann die Auswahl der Aktivitäten als hinreichend für die Durchführung der in Abschnitt 2.3 beschriebenen Funktionen eines Testfeldes gelten. Auf dieser Grundlage können nun verfeinerte Aktivitäten definiert werden. Dazu liefert [FES17] ebenfalls eine gute Vorlage. Die dort aufgeführten verfeinerten Aktivitäten decken zumindest grob die zu erwartenden Unterstützungsprozesse durch ein Testfeld ab. Exemplarisch sind die relevanten Aktivitäten und ihre korrespondierenden Unteraktivitäten in Tab. A.1 (im Anhang, S. 171) aufgelistet.

Auf dieser Basis können nun Use-Case-Diagramme im Sinn der in Abschnitt 2.12 beschriebenen funktionalen Zerlegung erstellt werden. Dazu ist eine Assoziation der Aktivitäten auf die zuvor definierten Rollen notwendig. Dies ist in Abb. 3.2 dargestellt. In Verbindung mit den Unteraktivitäten aus Tab. A.1 entstehen so umfassende Use-Case-Diagramme, die auf hoher Abstraktionsebene die Rollen mit den Prozessen und den entsprechenden Verfeinerungen darstellen. Exemplarisch ist dies für die Rollen *Wissenschaftler* und *Interessensnehmer* in Abb. A.5 (im Anhang, S. 183) dargestellt.

Sowohl [FES17] als auch [sim10] beschreiben eine Befragung der Probanden während

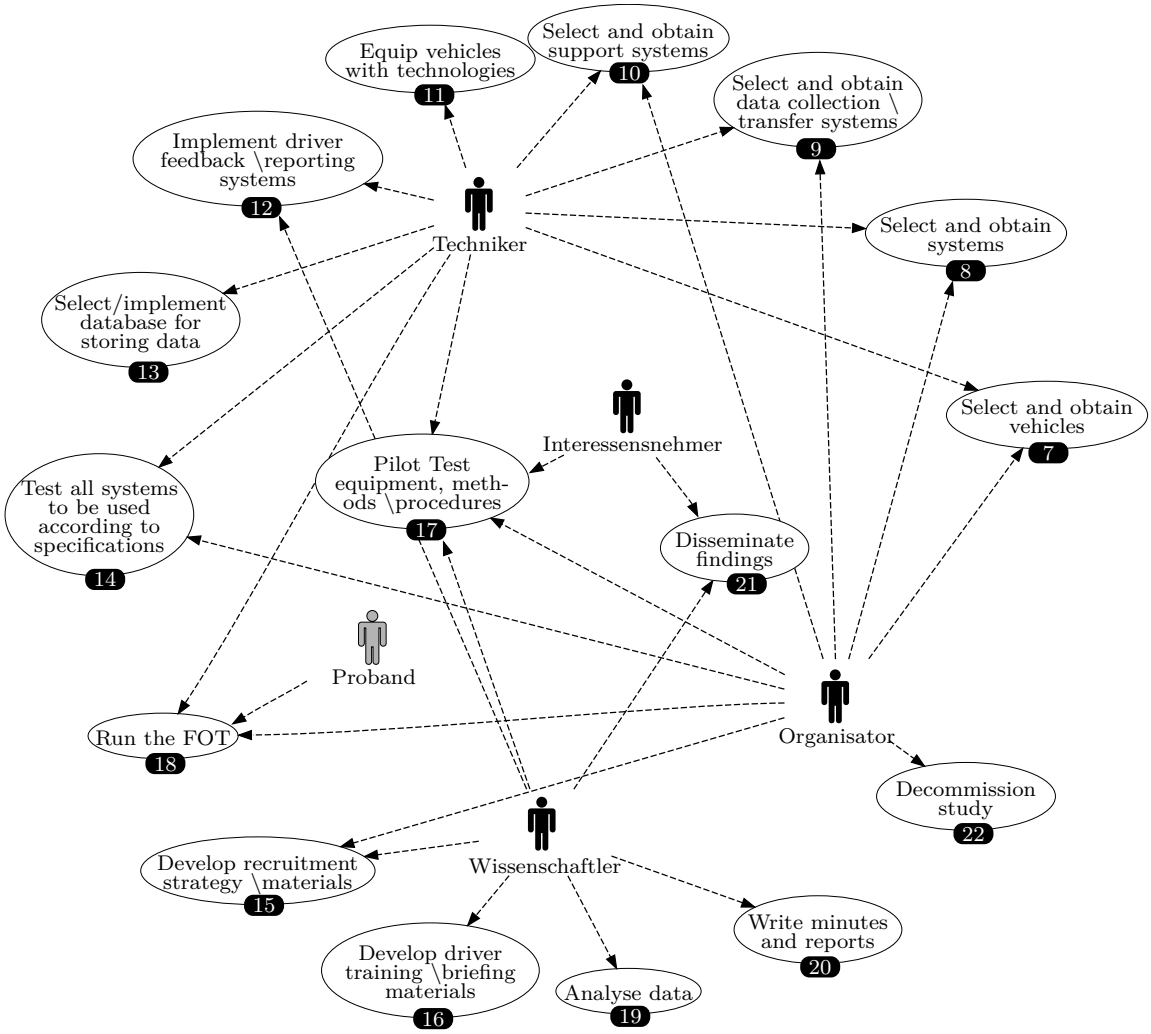

Abbildung 3.2: Use Case Modellierung von Rollen und Aktivitäten mit annotierten Referenzen auf Abb. 2.4

der Durchführung. Ein solches *Driver Log Book* dient zur Dokumentation subjektiver Eindrücke des Probanden und kann neben den versuchsrelevanten Fragen auch genutzt werden, um zeitnah Probleme oder technische Schwierigkeiten aufzudecken. In sim$^{\text{TD}}$ ist die Fahrerbefragung bspw. als Multiple-Choice-Dialog nach Stillstand des Fahrzeugs vorgesehen[4]. Typischerweise ist eine solche Befragung nicht im selbst Testsystem realisiert, da im operativen Betrieb keine Nutzerbefragung vorgesehen ist. Somit kann dies eine typische unterstützende Aufgabe eines Testfeldes sein, eben diese Mechanismen im Feldversuch anzubieten oder zumindest vergleichbare Funktionalität für das Testsystem bereitzustellen. Die Benutzerbefragung kann als Ansatz dienen, die funktionale Zerlegung auf Grundlage

[4]vgl. *Entwurfsbeispiele für Fahrerbefragung* in [sim10, S. 161]

der oben beschriebenen Benutzerprozesse weiter voranzutreiben. Dort ist die Durchführung des Feldversuchs (*Run FOT*) und als Verfeinerung die Erhebung von zusätzlichen Daten (*Collect additional (offline) Data*) modelliert. Dieser kann durch den eigentlichen Erhebungsvorgang weiter verfeinert werden, in dem Probanden ein entsprechendes Formular (*Handle Driver Log Book*) gezeigt wird. Dies ist, wie in Abschnitt 2.12 beschrieben, ein langlaufender Prozess, da die Interaktion eines Probanden bei der Bearbeitung des Formulars (*Fill Log Book*) abgewartet werden muss. Das Ausfüllen der Befragung kann zur Auslösung weiterer Prozesse führen, die entweder den Eintrag ablegen oder auf ein Problem hindeuten. Abb. 3.3 zeigt ein Use-Case-Diagramm, das die Prozesse der Benutzerbefragung illustriert.

Im gezeigten Beispiel kann zudem eine Personalisierung (*Provide personalized Form*) vorgesehen sein. Auf diesen Vorgang kann *Handle Driver Log Book* initiieren und *Fill Log Book* nutzen. Die dargestellten Prozesse bauen ihre Funktion aus atomaren Diensten auf. Diese sind ebenfalls in Abb. 3.3 beispielhaft dargestellt. Das Speichern eines Benutzerfragebogens (*Submit Log*) ist aus den Funktionen *Validate Log* und *Store Log* aufgebaut. In *Fire Event*, also der Auslösung eines bestimmten Ereignisses, ist die Wiederverwendung einer Funktion für mehrere Prozesse, hier *Submit Issue* und *Submit Repair Request*, dargestellt. Ist die Modellierung weniger fokussiert auf eine Zerlegung in atomare Dienste, sondern bspw. auf die Identifikation einer geeigneten Objektstruktur, so würden sich hier u. a. Klassendiagramme als adäquate semi-formale Darstellungsart anbieten.

3.6 Modellierung von Testfeldprozessen

Die zuvor dargestellte Zerlegung beruht im Wesentlichen auf Prozessen. Dies sind nicht nur die Geschäftsprozesse, also die Vorgänge der technischen Unterstützung durch ein Testfeld, sondern auch die daraus abgeleiteten lang- und kurzlaufenden Prozesse sowie Handlungen. Sie sind somit wichtige Einflussgrößen im Architekturentwurf und bedürfen einer genauen Beschreibung. Dazu bestehen verschiedene Möglichkeiten. Für die in diesem Kapitel dargestellte Herangehensweise bietet sich BPMN als Beschreibungsnotation gut an, da diese schon in [Sch08] als solche vorgeschlagen wurde. Zudem kann BPMN in ausführbare Geschäftsprozesse wie BPEL transformiert werden, ohne dabei unpräzise natürlichsprachliche Beschreibungen in eine maschinenausführbare Darstellung zu übertragen. Zudem ist eine Prozessbeschreibung in BPMN leichter durch entsprechende Werkzeuge bewertbar.

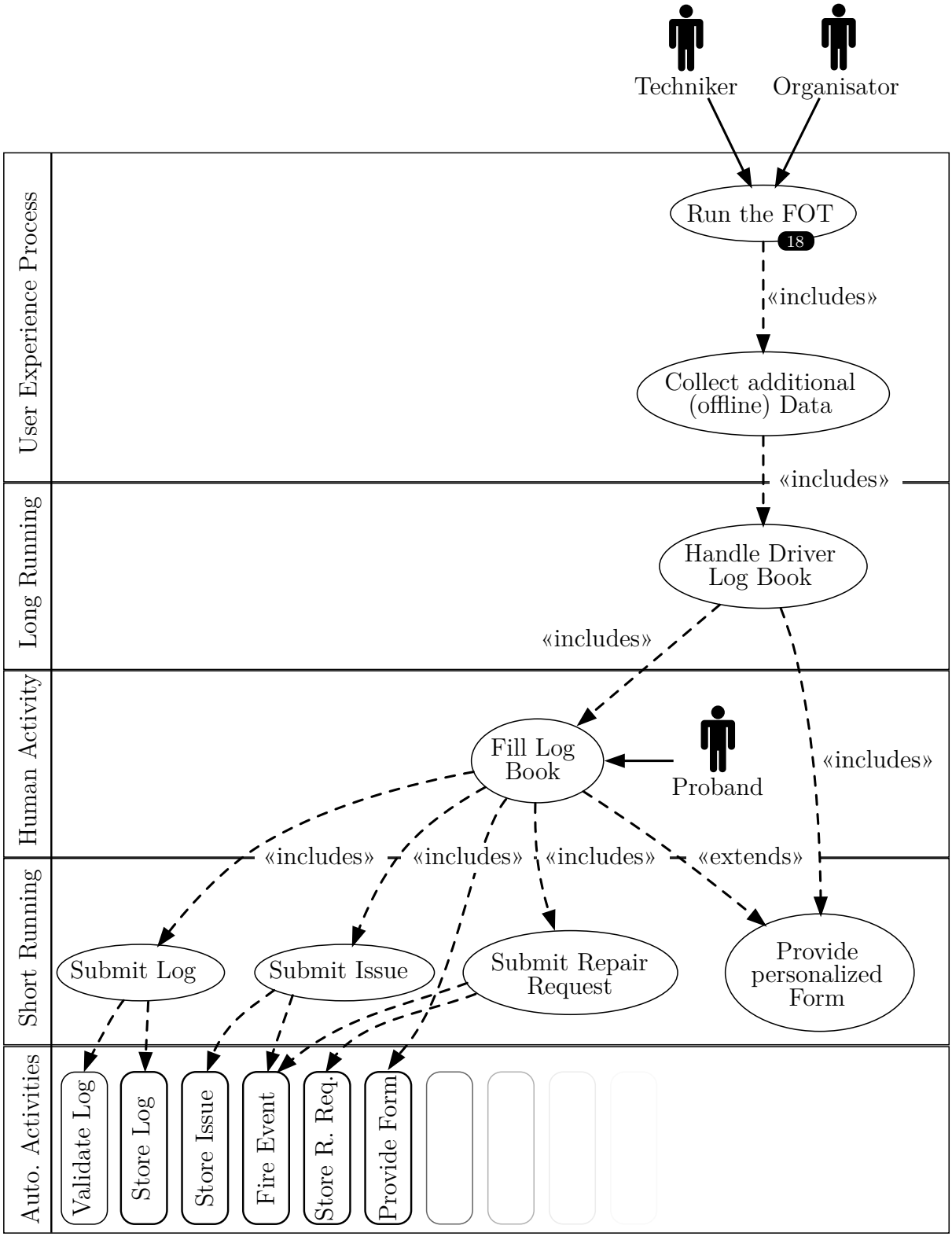

Abbildung 3.3: Exemplarische Modellierung von Handlungen, lang- und kurzlaufenden Prozessen

Analog zu der Zerlegung im vorangehenden Abschnitt können als Grundlage die von FESTA geprägten 22 Prozessschritte dienen, die in [FES17, S. 180-213] skizziert werden. Die Prozessschritte des Gantt-Diagrammes aus Abb. 2.4 sind sowohl nebenläufig als auch aufeinander aufbauend. Daher ist *Write minutes and reports* (20) ein kontinuierlicher, nebenläufiger Prozess, der die gesamte Durchführung eines FOT überspannt. Obschon die Prozesse *Equip vehicles with technologies* (11) und *Test all systems to be used according to specifications* (14) in Abb. 2.4 nebenläufig dargestellt sind, liegt dennoch eine logische sequenzielle Abfolge zugrunde. Der Probelauf (14) kann natürlich erst dann erfolgen, wenn die Instrumentation gemäß Aktivität (11) eingerüstet ist. In der Praxis werden beide Prozesse wechselwirken, denn es ist die Aufgabe eines Vorversuchs, ggf. vorhandene Problematiken im Versuchsaufbau zu identifizieren. So kann dieser dann auf Grundlage der Beobachtungen aus Aktivität (14) kontinuierlich verbessert werden. Somit überlappen die Prozesse in Abb. 2.4. Die Auswahl der zu modellierenden Prozesse folgt den Überlegungen aus dem vorangehenden Abschnitt. Modelliert man diese als BPMN in Sequenz und Nebenläufigkeit, so kann sich ein Prozessmodell wie in Abb. 3.4 ergeben.

Es zeigt die im vorangehenden Abschnitt aus dem FOTIP-Gantt-Diagramm (Abb. 2.4) herausgelösten Prozesse und stellt deren Nebenläufigkeit und Abhängigkeit dar. Jeder Prozessschritt im Diagramm kann nun wieder in kleinere Prozesseinheiten zerlegt werden. So ist für die im Diagramm modellierte Ebene, analog zu den Überlegungen aus dem vorangehenden Abschnitt, eine Präzisierung der Prozesse in Aktivitäten (vgl. Tab. A.1) sinnvoll. BPMN bietet die zur Modellierung notwendigen Konzepte sowohl auf grober Abstraktionsebene, wie in Abb. 3.4 dargestellt, als auch in feiner Detaillierung, wie in Abschnitt 2.11 ausgeführt.

Die hier beschriebene Modellierung ist stark auf die Orchestrierung von Diensten in einem diensteorientiertem Architekturkonzept zugeschnitten. Liegen andere Paradigmen zugrunde, müssten ebenfalls Prozesse definiert und beschrieben werden. Auf semi-formaler Ebene bieten sich dazu Kommunikations- und Sequenzdiagramme an. Diese beschreiben zwar ebenfalls den intendierten Ablaufplan, sind danach aber händisch in eine geeignete Implementierung zu überführen. Dies ist bei der Verwendung von BPMN und der unterstützten Transition in eine Ausführungssprache (BPEL) vermeidbar. Unabhängig von dem unterliegenden Paradigma, führt die Identifikation von Prozessen und zumindest groben Funktionen zu einem Prozess- und Funktionsverständnis des Systems. Die so identifizierten atomaren Funktionen können nun zu einer neuen Struktur synthetisiert werden,

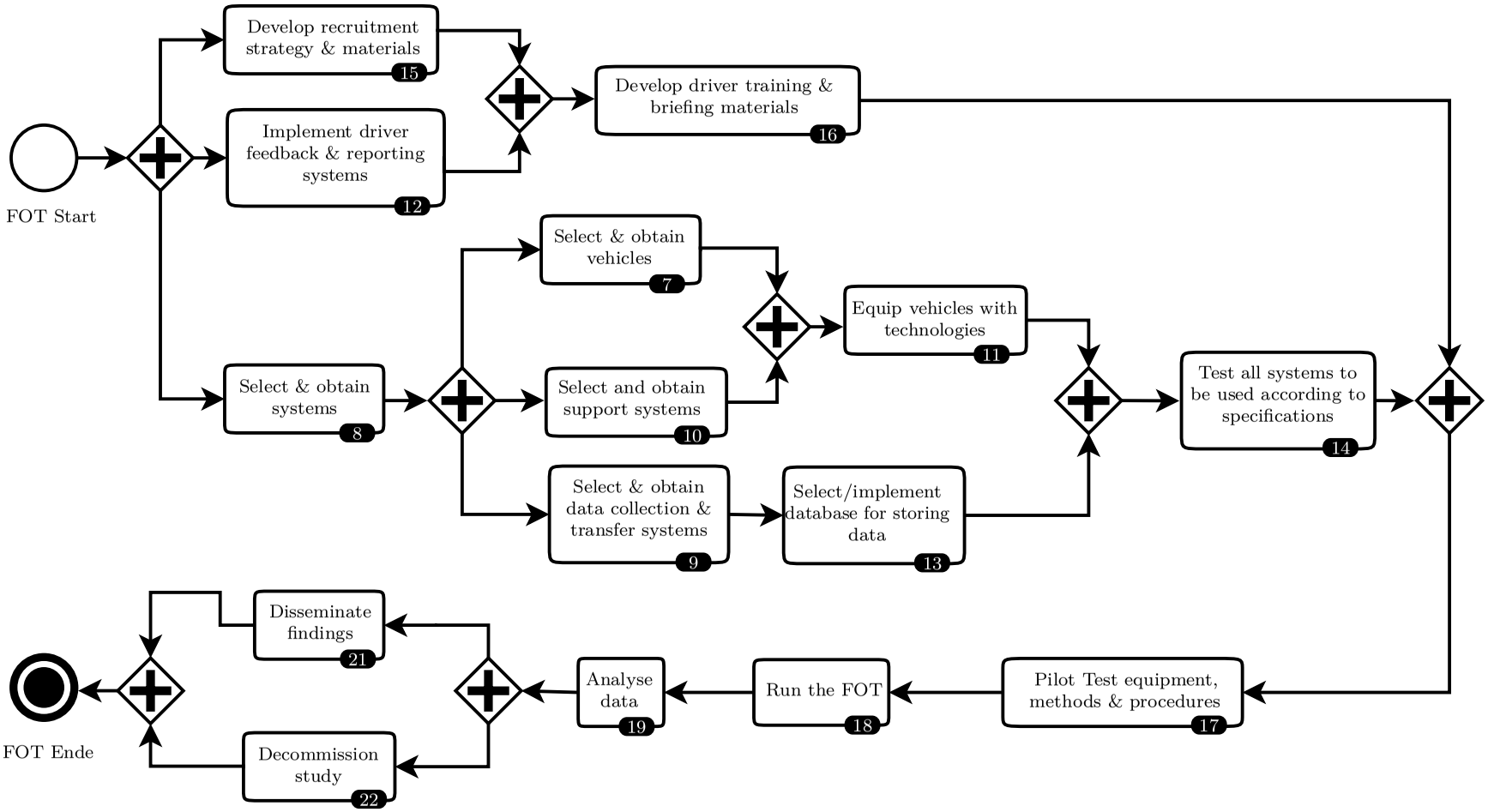

Abbildung 3.4: Exemplarische BPMN-Darstellung der Aktivitäten aus Abb. 3.2 mit annotierten Referenzen auf Abb. 2.4

die physische Gegebenheiten, wie z. B. räumliche oder logische Verteilung, reflektiert. So lassen sich Systemkomponenten aus deren Funktionalität definieren. Diesen Ansatz verfolgen die nachstehenden Abschnitte.
In den vorangehenden Abschnitten wurden grobe Funktionsszenarien eines Testfelds konkretisiert und anhand entsprechender Rollen und deren Interaktion mit dem Zielsystem in atomare Funktionen zerlegt. In diesem Abschnitt wurde gezeigt, wie diese Einheiten anhand von konkreten Prozessen orchestriert werden. Im Ergebnis liegt an dieser Stelle im Architekturentwurf nun eine detaillierte Darstellung der drei Perspektiven (Szenario, Logische Sicht und Prozesssicht, vgl. Abb. 3.1) vor. Dies ist Ausgangspunkt für die bislang noch nicht betrachtete Verteilung der Funktion auf verschiedenen Einheiten und der Softwarestrukturierung, die in den beiden folgenden Abschnitten ausgeführt wird.

3.7 Systementwicklungssicht

Nachdem in den vorangehenden Abschnitten auf Basis eines generischen Szenarios die Funktion des Systems, dessen zugrunde liegende Prozesse und Kernkomponenten identifiziert wurden, sind nun weiterführende Überlegungen zur Realisierung Inhalt dieses Abschnitts. Wie in Abschnitt 2.10 ausgeführt, beschreibt die Systementwicklungssicht das Zielsystem aus der Perspektive eines (Software-)Entwicklers.[5] Im Zusammenhang mit dem in Abschnitt 2.12 vorgeschlagenen Zerlegungsprozess wird die Systementwicklungssicht als erster Syntheseschritt aufgefasst, der die identifizierten atomaren Dienste in größere Funktionseinheiten (bspw. Pakete, Module) zusammenfasst. Im Kontrast zum Top-down-Ansatz der funktionalen Zerlegung entsteht so bottom-up eine umfassende Perspektive auf das System in größeren funktionalen Einheiten, Schnittstellen und deren Zusammenspiel. Nachdem die funktionale Zerlegung in dem vorangehenden Abschnitt ausgeführt wurde, können die Überlegungen zur Gestaltung der Systementwicklungssicht auf deren atomare Dienste (s. Abschnitt 3.5) aufsetzen. Dieser Abschnitt beschreibt das genaue Vorgehen dazu detailliert und zeigt eine typische Systementwicklungssicht.
[Kru95] beschreibt den Prozess *Subsystem decomposition*[6]. Dieser soll den angestrebten Funktionsumfang aus einer Top-down-Perspektive in der Systementwicklungssicht abdecken. Da im Zerlegungsprozess[7] schon kleinste Funktionseinheiten identifiziert sind, ist an

[5] „*[...] which describes the static organization of the software in its development environment.*“ [Kru95, S. 2]
[6] vgl. [Kru95, S. 6]
[7] s. *Automated Activities* in Abschnitt 3.5

dieser Stelle bereits eine Synthese möglich, die Bottom-up-Funktionseinheiten zu größeren Strukturen zusammenfasst. Die u. a. von [Kru95] beschriebenen Gestaltungsrichtlinien können dabei weiter bestehen. [Kru95] betont dabei, inhärente Anforderungen der Softwareentwicklung und des -managements zu berücksichtigen, sowie die Wiederverwertung, Portierbarkeit und Sicherheit von Komponenten in die Architekturentwicklung einfließen zu lassen. Diese nicht-funktionalen Aspekte werden in diesem und insbesondere im folgenden Abschnitt zur Darstellung der Systemkomponenten aufgegriffen und im folgenden Kapitel 4 qualitativ bewertet.

Zunächst ist eine Konkretisierung sinnvoll, in welcher Weise die Synthese erfolgen soll, bzw. welche Form von Softwarebausteinen komponiert werden soll. In den Grundlagen wurden in Abschnitt 2.8 einige typische Architekturmuster vorgestellt. So wäre bspw. eine Synthese der atomaren Funktionseinheiten in UEP mittels Pipes&Filters denkbar. Wie im Abschnitt 2.9 zu Architekturen für verteilte Systeme ausgeführt wurde, scheint insbesondere das Konzept der Diensteorientierung geeignet, da insbesondere die nichtfunktionalen Anforderungen wie bspw. Interoperabilität, Austauschbarkeit und Flexibilität durch eine SOA gut abgesichert scheinen [RWJ+11, S. 86f, 97f]. So folgt die hier aufgebaute Systementwicklungssicht der in [Sch08] vorgeschlagenen Diensteorientierung.

Das Beispiel aus Abb. 3.3 skizziert den Zerlegungsprozess zur Identifikation atomarer Dienste (bzw. *Automated Activities*) der Benutzerbefragung. So sind u. a. Dienste zur Überprüfung (*Validate Log*) und Speicherung (*Store Log*) von Logbucheinträgen vorgesehen. Zudem sind Dienste vorgesehen, die Probleme (*Issues*) mit der Versuchsdurchführung oder Reparaturen (*Repair*) an Versuchsfahrzeugen handhaben. Ziel der Synthese kann es sein, Dienste mit verwandter Funktion in Paketen zu bündeln. Es ist denkbar, ein Paket *Proband Offline Interface* zu definieren, das alle Funktionen umfasst, die benötigt werden, die Interaktion mit dem Probanden, bspw. im Debriefing, durchzuführen. Die Teilfunktionen Logbuch, Probleme u. Reparaturen können in separaten Paketen *Log Book*, *Issue Tracker* und *Repair Manager* gefasst werden. Diese lassen sich dann im *Proband Offline Interface* wiederverwenden (engl. *merge*).

Dieses Konzept kann auch genutzt werden, um eine Persistenzebene für Logbucheinträge zu modellieren. So wird das Paket *Log Book Persistance* mit dem Paket *Log Book* kombiniert, sodass Letzteres auf die entsprechenden Speicher- und Laderoutinen zugreifen kann.

So entsteht das in Abb. 3.5 dargestellte Paketdiagramm, welches seinerseits Teil eines übergeordneten Pakets *PKG Proband UI* sein kann, das die Interaktion mit Probanden ganzheitlich abdeckt.

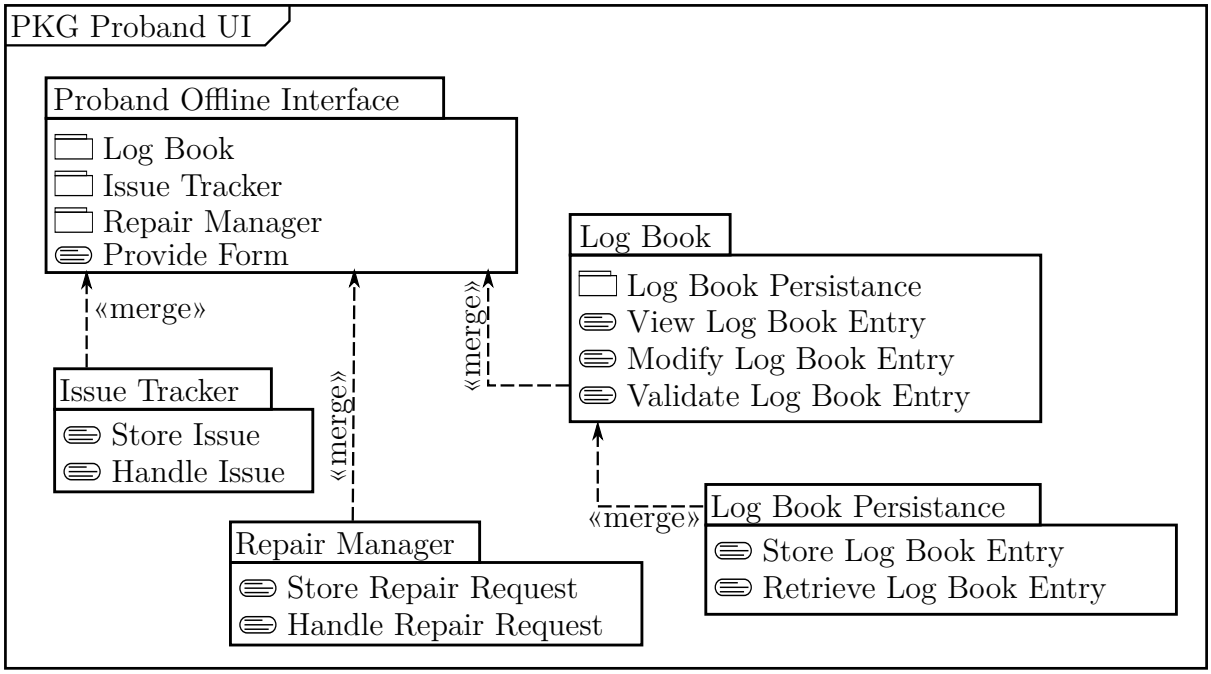

Abbildung 3.5: Exemplarisches Paketdiagrammbeispiel zur Probandeninteraktion

Im oben skizzierten Beispiel ist die enge Verzahnung zwischen Logischer Sicht und Systementwicklungssicht gut sichtbar. Die in Abb. 3.3 dargestellten Dienste sollten signifikant mit denen der UEP korrespondieren. So kann aus der Kombination der beiden Sichten die Interaktion zwischen Subsystemen, Softwaremodulen und -paketen in den Prozessen abgeleitet werden.
Die hier ermittelte Systementwicklungssicht stellt eine weitere Architekturbeschreibung gemäß des *4+1 View Model* dar. Für eine vollständige Architekturbeschreibung bedarf es jedoch noch einer genaueren Betrachtung der physikalischen Komponenten (Hardware). Dies wird als letzter Schritt im Architekturentwurf im folgenden Abschnitt ausgeführt.

3.8 Systemkomponenten

Die bisherigen Architekturausprägungen beruhen auf der sukzessiven Zerlegung von Prozessen und abstrakten Diensten, sowie einer ersten Synthese dieser zu Softwarekomponenten. Ein kritischer Schritt in der Gestaltung der Architektur ist die Verteilung der zuvor identifizierten Dienste auf die elektronische Ausrüstung eines Testfeldes, um ein geschlossenes Gesamtkonzept zu formen [All97, S. 1], da zunächst aus Sicht der Systemarchitektur aus der isolierten Betrachtung atomarer Dienste nicht festgelegt wird, welche Softwarekompo-

nenten auf welchen Hardwareeinheiten zum Einsatz gebracht werden. Die Kommunikation bzw. Vernetzung ist bei einer verteilten Architekturlandschaft, wie die eines Testfeldes, besonders zu betrachten. Dieser Blickwinkel auf den Architekturentwurf wird im Folgenden ausgeleuchtet.

Die Entwicklung der Sicht auf Systemkomponenten ist stark durch die enge Verknüpfung von Testfeld und Testsystem determiniert. Typischerweise ist die Architektur des Testsystems vorgegeben. Das Testfeld muss sich also i. d. R. an das Testsystem anpassen. Wegen der intendierten Virtualisierbarkeit ist es nicht hinreichend, analog zur typischen Anbindung von Altsystemen, eine triviale modulare Schnittstellenkapselung[8] vorzusehen. Daher ist kein rein synthetischer Ansatz sinnvoll, der isoliert die in den vorangehenden Abschnitten definierten Funktionseinheiten über Hardwareeinheiten verteilt. Vielmehr müssen die Vorgaben, die sich aus der (verteilten) Systemlandschaft des Testsystems ergeben, berücksichtigt werden. Daher wird in diesem Abschnitt die Sicht auf Systemkomponenten aus grober Abstraktionsebene (top-down) auf Grundlage der domänenspezifischen Vorgaben durch Testsysteme verfeinert. Daraus entsteht zunächst eine grobe Sicht auf die im Testfeld vorzusehende Infrastrukturlandschaft.

Losgelöst von der Betrachtung der identifizierten Prozesse und atomaren Dienste stehen einige physikalische Einheiten, bedingt durch den typischen Testsystemaufbau, vorab fest. Dies sind die mobilen Einheiten (OBU), die bspw. in Fahrzeugen verbaut oder von Reisenden mitgeführt werden und die stationären Einheiten (RSU), die bspw. Zugriff auf lokale Infrastruktur ermöglichen. Zudem sind in vielen Szenarien Testsysteme auf ein dahinterliegendes Rechnersystem angewiesen, das im Testbetrieb gleichzeitig testfeldrelevante Aufgaben übernimmt wie bspw. die Datenerfassung. Selbst wenn im individuellen Testfall nicht alle drei Komponenten eingesetzt werden, ist diese grobe physikalische Gliederung ein wiederkehrendes Muster der in Abschnitt 2.5 untersuchten Aktivitäten.

Im Gegensatz zu den vorangehenden Abschnitten ist zur weiteren Präzisierung einer Gestaltungsidee das FESTA-Handbuch als Hauptquelle weniger hilfreich, da [FES17], durch den eigenen Anspruch als Leitfaden, sich wesentlich auf die Definition von Prozessen und Methodiken erstreckt. Durch entsprechende Konkretisierungen bieten die Überlegungen aus Abschnitt 2.5 und insbesondere 2.6 zu ITS Test Beds, AIM, DITCM und sim^{TD} einen geeigneteren Ausgangspunkt.

Im Einklang mit Punkt 2 (S. 65) muss festgelegt werden, welche Komponenten im Testfeld betrieben werden sollen und welche Komponenten benötigt werden, um diese im Sinn

[8] vgl. *Wrapping Strategies* zur Integration von Altsystemen in [ACD10]

eines Testfeldes zu unterstützen bzw. zu überwachen. Die Architekturdefinition in ITS Test Beds präzisiert diesen Aspekt in [BVJ+11] nicht, während [Ver10] zwar Komponenten des Testfelds identifiziert, aber leider keine genauere Beschreibung von unterstützten Testsystemen bzw. deren Anforderungen enthält. Die Architektur aus sim^{TD} kann hier als Vorlage dienen. Abbildung 2.8 zeigt wesentlich drei Komponententypen: Fahrzeugeinheiten, stationäre Einheiten und die Versuchszentrale. Letztere korrespondiert mit dem TASC-Konzept von ITS Test Beds. AIM und DITCM zeigen eine identische Aufteilung.
Jede dieser Komponenten besitzt eine eigene logische Einheit, die Berechnungen durchführen kann. In ITS Test Beds ist bspw. die Einbindung von Dominion als Laufzeitumgebung vorgeschlagen. Bei sim^{TD} sind dies die *Application Units* bzw. RSU bei AIM und DITCM. Nimmt man dies als Vorlage, lässt sich eine grobe Systemaufteilung gewinnen, die in Abb. 3.6 dargestellt ist.

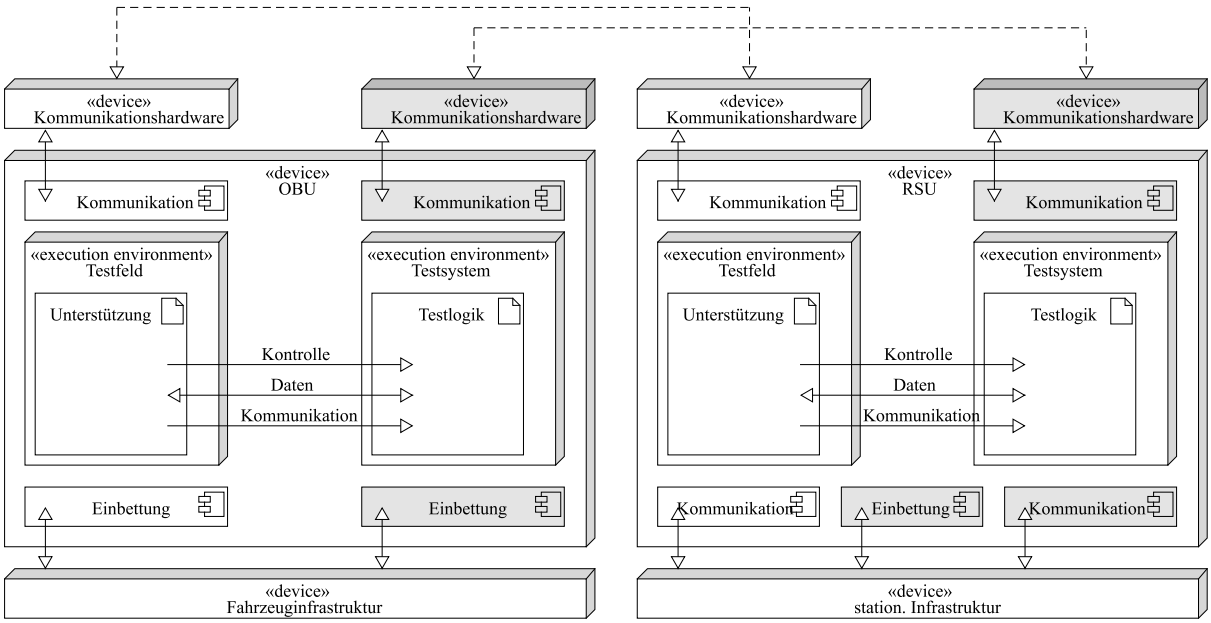

Abbildung 3.6: Systemsicht von OBU und RSU

Kernidee dieser Ausgestaltung ist zum einen der mögliche redundante Aufbau der Einheiten OBU und RSU, zum anderen die Unterstützung in drei Aspekten: Kontrolle, Daten und Kommunikation. Dies folgt dem Konzept *separation of concern*, welches im Kontext von MVC eine klare Trennung wesentlicher Funktionen im Softwaresystem vorsieht und so u. a. gute Handhabbarkeit und Isolation in komplexen heterogenen Systemen verspricht (s. Abschnitt 2.8). Abweichend zum View in MVC ist Kommunikation als wesentliches Segment dargestellt. Dies ist in der hohen Relevanz des Kommunikationsaspekts begründet.

Obwohl bspw. OBU im Testsystem durchaus über eine für die Anwendung wichtige Visualisierung und direkte Benutzerinteraktion verfügen können, ist diese auf hoher Abstraktionsebene im Testfeld jedoch von eher untergeordneter Relevanz.
Die wesentlichen Komponenten innerhalb der Einheiten können sowohl testsystem- als auch testfeldseitig vorhanden sein. Dies ermöglicht, mittels Testfeldkomponenten evtl. fehlende Bausteine im Testfeld zu ersetzen oder virtualisieren. So werden im Testfeld für OBU und RSU vergleichsweise hochwertige Komponenten eingesetzt, wie bspw. ein relativ leistungsfähiger Rechner mit ggf. mehreren Kernen, der für ein operatives System nicht ökonomisch wäre. So existiert in jedem Fall eine Referenz, von der aus zu einem parallelen und eigenständigem System migriert werden kann.
Folgt der Entwurf der hier vorgeschlagenen Dreiteilung, kann die Unterstützung der in den OBU und RSU vorhandenen Testsystemlogik prinzipiell auf drei Arten erfolgen:

Kontrolle Das Testfeld ist ein verteiltes System, in dem bspw. die eingesetzten Komponenten überwacht werden müssen, um so sicherzustellen, dass der Versuchsaufbau wie geplant funktioniert. Die zu testende Logik auf einer der Einheiten ist im Erprobungszustand ggf. instabil. Daher kann das Testfeld bspw. neue Versionen aufspielen und starten. So ist u. a. eine hindernisfreie Bereitstellung, Arbitrierung und Wartung von Komponenten im verteilten Testfeld möglich.

Daten An dieser Stelle muss zwischen Daten und Kommunikation klar unterschieden werden. Der Aspekt ‚Daten' befasst sich mit deren Informationsgehalt. Dem gegenüber fokussiert der Begriff ‚Kommunikation' die Art und Weise des Austausches von Informationen zwischen Komponenten. So ist der Aspekt ‚Daten' hier also die potenziell ubiquitäre Verfügbarkeit von Daten im Testfeld. Dies erleichtert die rasche Entwicklung von Testsystemen, da bspw. auf externe Daten aus dem Fahrzeug heraus zugegriffen werden kann, ohne die vollständige Kommunikationskette über die RSU nachzuvollziehen.

Kommunikation Die Kommunikation der Einheiten untereinander kann durch das Testfeld dargestellt werden. Ohne dass die Implementierung der Testsystemlogik umfänglich Rücksicht auf die zugrunde liegenden Konzepte nehmen muss, kann angenommen werden, dass eine Kommunikation durch das Testfeld gelöst ist. Ebenso kann diese Virtualisierung der Kommunikation genutzt werden, um technische Eigenheiten des intendierten Kommunikationsmediums zu emulieren. Sollen bspw. mehrere RSU via mobilem Internet (bspw. 5G) vernetzt werden, ist in schwierigen Gegebenheiten

vielleicht mit einer signifikanten Ausfallrate zu rechnen. Im Testfeld kann dies über eine durchaus stabile kabelgebundene Netzwerkverbindung dargestellt werden, die artifiziell und nach Vorgaben des Testenden Ausfälle zeigt.

Abbildung 3.6 illustriert diese drei Aspekte Kontrolle, Daten und Kommunikation in der Verzahnung von Unterstützungs- und Testsystem. Ebenfalls ist die Einbettung der OBU an die unterliegende Fahrzeuginfrastruktur angedeutet. Aus Sicht der Anwendung kann das Testfeld als Abstraktionsschicht dienen, um leichtgewichtig auf Fahrzeugressourcen wie bspw. einen CAN-Bus[9] zuzugreifen. Gleiches gilt für die RSU. Von hier aus kann auf die Sensorik an bzw. in der Straße zugegriffen werden. Ist die RSU mit Aktuatorik ausgerüstet, wie bspw. einer Lichtsignalanlage, dann ist eine entsprechende Ansteuerung denkbar und wird durch Virtualisierung durch das Testfeld leicht in Testsystemen realisierbar.

Aus der gleichen Perspektive kann auch die Versuchszentrale dargestellt werden. Analog zu RSU und OBU findet auch hier eine Unterstützung der Testsystemlogik statt. Letztere fungiert hier als Zentraleinheit bzw. *Central System* aus [FOF+10]. Hier ist ebenfalls eine Unterstützung des Testsystems in den drei oben beschriebenen Aspekten möglich. Zudem koordiniert die Versuchszentrale auch die Aktivitäten im Testfeld. So wird auch die für RSU und OBU beschriebene Funktionalität dargestellt. So wird bspw. das Verhalten der Kommunikationswege (sofern durch das Testsystem genutzt) beeinflusst und die zentrale Datenhaltung in dieser Einheit gelöst. Die Zentraleinheit ist die koordinierende Komponente im Netzwerk der Einheiten OBU und RSU und steuert sämtliche testfeldrelevante Aufgaben. Diese zentrale Steuerung ermöglicht leicht zwischen verschiedenen Konfigurationen des Testfelds zu schalten und begünstigt daher sowohl den problemlosen Wechsel zwischen verschiedenen Versuchsaufbauten als auch unterstützende Konzepte wie bspw. Erprobung in einer Sandbox. Der Entwurf folgt auch hier der Segmentierung in wesentlichen Aufgaben hinsichtlich Kommunikation, Daten und Kontrolle sowie der potentiellen Virtualisierung von Testsystemlogik im Testfeld.

Das in Abb. 3.6 skizzierte Testsystem wird i. d. R. ein komplexes eigenständiges Softwaresystem sein. Im Fall von OBU und RSU können dies bspw. entsprechende Komponenten aus CVIS [FOF+10] oder ETSI [ETS09] sein. Unabhängig von der konkreten Realisierung müssen Schnittstellen zwischen Testsystem und Unterstützung dargestellt werden. Im Fall einer an ETSI angelehnten RSU/OBU schlägt [BVJ+11] eine Anbindung über Service

[9] Controller Area Network (CAN) ist ein oft in Fahrzeugen zu findender Feldbus. So können ggf. wichtige Telemetriedaten wie Geschwindigkeit und Position zugreifbar werden.

Access Points (SAPs)[10] vor, über die Komponenten des Systems verbunden sind. Werden SAP entsprechend genutzt, ist es leicht möglich, die Interaktion zwischen den Komponenten zu protokollieren, in die Interaktion einzugreifen oder sogar ganze Komponenten zu emulieren. Auf Seite der Unterstützung müssen entsprechende Mechanismen realisiert werden, die an einen SAP andocken, um bspw. die Interaktion zu protokollieren (Datenaspekt), aktiv Komponenten des Testsystems zu steuern (Kontrollaspekt) oder bestimmte Kommunikationskanäle zu emulieren (Kommunikationsaspekt).

Die Ausgestaltung der Unterstützung ist üblicherweise stark an das Testsystem angelehnt. So fasst bspw. sim$^{\text{TD}}$ die Kontrolle über den Versuchsablauf nicht als eigenständige Aufgabe auf, sondern integriert entsprechende Funktionalität direkt in die Komponenten des sim$^{\text{TD}}$-Systems. Viele Feldtests wie bspw. DRIVE C2X folgen diesem Muster. Dadurch ist dieser Aufbau weniger flexibel gegenüber evolutionären Anforderungen und eine solche Vermischung aus Anwendungs- und Testfunktionalität ist nur in Testfeldern mit sehr starren Anforderungen sinnvoll.

ITS Test Beds schlägt eine eigene Laufzeitumgebung, wie bspw. Dominion Runtime Environment [BVJ+11, S. 81ff], für Komponenten des Testsystems vor. So können auch Komponenten virtualisiert werden und deren Funktion ist somit von der zugrunde liegenden Infrastruktur losgelöst. Dies ist nach [BVJ+11] insbesondere bei der frühen Entwicklung von Assistenzen und Automationen hilfreich, die von der Exploration in Simulationen zur in-situ-Erprobung wechseln. Bei einer seriennahen Erprobung steht die Erprobung aller Komponenten bei möglichst realistischen Bedingungen im Vordergrund. So ist eine Virtualisierung in einem solchen Versuchsszenario konzeptionell zu vermeiden.

Aus Testfeldsicht optimal ist daher ein Unterstützungssystem, das in der Lage ist, die Testsystemkomponente vollständig virtuell darzustellen und nach Bedarf die Unterstützung der Funktionalität auf Minimalaspekte (bspw. Datenaufzeichnung, Kontrolle) zu reduzieren. Dazu verwendet das Unterstützungssystem redundante (Referenz-)Komponenten. So ergibt sich ein hoher Freiheitsgrad an Testaufbauten. In der Praxis ist diese Ambition von nichtfunktionalen Aspekten wie Kosten oder verfügbare Baugröße begrenzt.

Im universalen Anwendungsfall ‚Erprobung' muss die Arbitrierung von verschiedenen Testsystemen ebenfalls von dem Testfeld vorgenommen werden. Im Detail muss die auszuführende Testlogik für eine Einheit (RSU/OBU) bestimmt und festgelegt werden, welche Ressourcen (bspw. Kommunikationskanäle) für diese notwendig sind. Im Fall unabhängiger sequentieller Kampagnen ist die Administration verschiedener Konfigurationen eine hand-

[10] Interaktionsschnittstelle zwischen Systemkomponenten, s. [ETS09, S. 11ff]

habbare Aufgabe. Im verteilten Testfeld sind jedoch ggf. unterschiedliche Tests simultan zu erwarten. Zudem sind Untersuchungen zur Wechselwirkung unterschiedlicher Anwendungen denkbar. In sim^{TD} wird eine begrenzte Zahl von Anwendungen ins Feld geführt, der dazu notwendige Logikteil ist statisch konstruiert. Zur Laufzeit werden nur Funktionen oder Module angesprochen, die zur Konfigurationszeit schon auf dem System befindlich waren.[11] So sieht sim^{TD} hier keine Flexibilität vor. ITS Test Beds beschreibt mit *Test Pattern Management* eine Architekturkomponente, die eine flexible (Fern-)Konfiguration aller zum Test gehörender Komponenten vorsieht. Dadurch ist zwischen Testläufen ein Konfigurationswechsel möglich. Eine noch höhere Dynamik wird hier vom Testsystem selbst geboten. Bspw. CVIS oder ETSI sehen einen gewissen Pool an möglichen Anwendungen vor, die auf den Systemkomponenten (z. B. RSU oder OBU) vorgehalten werden und arbitrieren diese eigenständig. Dadurch ist es möglich, ganz unterschiedliche ITS-Anwendungen simultan im Feld zu testen. Die Funktion der hochflexiblen Arbitrierung in das Testsystem zu verlagern ist praxisnah und widerspricht nicht dem Testfeldkonzept, da durch ggf. enge Kopplung von Testsystem und Unterstützung die Kontrolle weiterhin möglich ist.

Als zentrale Einheit in der Testlandschaft ist die Versuchszentrale nicht nur für die Bereitstellung der eigentlichen ITS-Funktion zuständig, sondern auch für die Koordination des Versuchsbetriebs. Diese beiden Aspekte werden im technischen Aufbau dieser Einheit verschmolzen. Die bestehenden Versuchszentralen wie bspw. aus sim^{TD} oder DRIVE C2X sind stark zugeschnitten auf die individuellen Forschungs- und Erprobungsaspekte. Eine Umsetzung genereller Konzepte, wie bspw. in [BVJ+11] vorgeschlagen, ist, mit Stand 2019, in den untersuchten Testfeldern nicht erkennbar.

Wie in den Grundlagen ausgeführt, ist eine singuläre Betrachtung isolierter Anwendungen im Testfeld schwer vorstellbar. CVIS beschreibt mit dem *ITS App Store*[12] Mechanismen, um unterschiedliche Anwendungen dem Anwender zugänglich zu machen. In jüngeren Projekten wird die Systemlandschaft der Zentraleinheit zunehmend komplexer und besteht u. a. aus sehr unterschiedlichen Diensten, die teils aufeinander aufbauen.[13] Daher scheint ein Aufbau analog zum vergleichsweise statischen sim^{TD}-Ansatz wenig sinnvoll. Die ITS-Test-Beds-Architektur sieht hier eine gewisse Flexibilität vor, prägt jedoch kein genaues Systemverständnis in diesem Aspekt.

[11] Es ist eine gewisse Flexibilität durch unterschiedliche Versuchsszenarien gewährleistet, vgl. [sim09b, S. 71f]. Ein echter Austausch von (Software-)Komponenten ist nicht vorgesehen.

[12] ein virtueller Marktplatz für ITS-Anwendungen, s. [Kom10, S. 39ff]

[13] bspw. die Systemarchitektur einer verteilten Reiseassistenz in [BST11]

Für einen nachhaltigen Entwurf kann erneut das Konzept der Virtualisierung aufgegriffen werden. Zwar kann die Testlogik als monolithischer Funktionskern aufgefasst werden, der in einer geeigneten Laufzeitumgebung eingebettet ausgeführt wird. Dies entspricht dem sim^{TD}-Aufbau und entsprechend eng umrissene und insbesondere bestehende Systeme können so erprobt werden. Dennoch ist auch für die Entwicklung eine gewisse Redundanz sinnvoll, bspw. können Teile des Testsystems durch (Referenz-)Komponenten der Versuchszentrale ersetzt werden. Dies kann bis zu einer vollständigen Virtualisierung der gewünschten Funktionalität reichen. Diese Entwurfsentscheidung beinhaltet auch den Freiheitsgrad, später im Testbetrieb die Kapazitäts- und Lastverteilung zwischen den Komponenten des Testsystems flexibel zu erproben. So können ebenfalls diese Effekte und deren Handhabung[14] untersucht und bewertet werden.
An dieser Stelle lässt sich gut die Funktionalität der Versuchszentrale aus Entwicklungssicht und Prozesssicht entwickeln. Es lassen sich folgende Schlüsselkomponenten induzieren, deren wesentlicher Charakter im Folgenden grob umrissen ist:

Leitstand Aus Testfeldsicht ist eine grundlegende Arbitrierung der Komponenten notwendig. Wie schon ausgeführt sind bspw. die vorhandenen Komponenten (bspw. RSU und die dort vorhandene Infrastruktur) mit der Testlogik zu verknüpfen oder es sind Mechanismen zu konfigurieren, um die analyserelevanten Daten aufzuzeichnen. Dazu ist ein Leitstand als Endbenutzerschnittstelle vorzusehen, welche die nach Abschnitt 3.5 ermittelte Interaktion mit dem Testfeld erlaubt. Solche Komponenten sind u. a. auch in sim^{TD} zu diesem Zweck realisiert worden.

Die feine Ausgestaltung dieser Komponente kann sich insbesondere auf die UEP und die aus den Szenarien abgeleiteten Prozesse stützen.

Diensteverzeichnis ITS Test Beds widmet mehrere Architekturkomponenten dem Management von Infrastruktur und Diensten. Es liegt daher nahe, dies als eigenständige Aufgabe in einer physischen Einheit aufzufassen. Im Kontext einer verteilten Architektur entspricht das Diensteverzeichnis einer eigenständigen Komponente, die das Auffinden[15] und Nutzen entsprechender Testfeldkomponenten sowohl zur Versuchsvorbereitung als auch zur Laufzeit ermöglicht. Das Diensteverzeichnis fungiert

[14] bspw. *Graceful degradation*, die Reaktion des Systems auf unvorhergesehene Störungen, hier in Rechenkapazität bzw. Lastverteilung

[15] ermöglicht u. a. *service discovery*, eine automatische Detektion von Infrastruktur und Diensten im Testfeld

also als ein Repository der verfügbaren Komponenten. Diese können über ihre funktionale und nicht-funktionale Beschreibung identifiziert werden. In Verbindung mit der Aufzeichnung durch die Datenerfassung (s. u.) ist es so möglich, vor, während und nach Versuchsdurchführung nachzuvollziehen, welche Komponenten in welcher Funktion eingesetzt wurden. Dadurch, dass nicht-funktionale Aspekte ebenfalls im Diensteverzeichnis abrufbar sind, sind auch die in [BVJ+11] ausgeführten Metadaten der Komponenten im Versuchsaufbau nachvollziehbar.

Ergänzend zu den Metadatenkonzepten in [BVJ+11] wird die Beschreibung der Entitäten mit einer Semantik erweitert, die eine klare physische Assoziation erlaubt, so ist bspw. eine Zuweisung eines bestimmten Temperaturgebers bzw. seines Temperaturwertes zu einer RSU wünschenswert. So lassen sich überhaupt erst lose funktionale Einheiten (z. B. Sensorwerte) mit Komponenten im Testfeld zu Testsystemkomponenten verschalten.

Testvorbereitung Die oben ausgeführte flexible Umgestaltung der Testinfrastruktur zwischen Versuchsläufen oder ggf. während der Versuchsdurchführung bedarf einer Arbitrierung. Die Planung des Versuchslaufs und Eingriffe in den Versuch werden dabei vom Leitstand aus vorgenommen. Die Testvorbereitung verschaltet dann die benötigten Komponenten im Testfeld. Dies ist analog zu den Überlegungen zum *Test Pattern Management* aus [BVJ+11]. Die Aufgabe dieser Komponente ist auch eine aktive Überwachung des Testfeldaufbaus im Sinne eines Monitorings. So kann bspw. auf Ausfälle bei Komponenten zur Laufzeit reagiert werden, indem automatisch alternative Bauteile verschaltet werden oder ein händisches Eingreifen über den Leitstand technisch umgesetzt wird.

Datenerfassung Die Datenerfassung ist eine zentrale Aufgabe des Testfelds und erschließt die nachträgliche Auswertung des Versuchslaufs. In bestehenden Aufbauten wird diese Aufgabe nicht in gewidmeten Komponenten reflektiert (bspw. [FOF+10]), obwohl die Wichtigkeit klar herausgestellt ist (u. a. in [FES17]). Hingegen hat [BVJ+11] die Datenerfassung, -haltung, -vorverarbeitung und -auswertung in eigenen Architekturkomponenten abgebildet. So liegt es nahe, zumindest eine eigene Komponente der Datenerfassung an zentraler Stelle innerhalb der Versuchszentrale vorzusehen. Kernaufgabe dieser Einheit ist die Aufzeichnung aller versuchsrelevanten Daten im

Testfeld und die Bereitstellung dieser für Entwickler und Analysten gleichermaßen. Dies umfasst nicht nur vorab definierte Kennzahlen (s. *Indicators*, [FES17]), sondern auch die oben ausgeführten Metadaten.

Bei der feineren Ausprägung dieser Komponente sind besonders die Prozesse aus der Betrachtung der UEP relevant. So lässt sich insbesondere ableiten, welche weiterverarbeitenden Schritte vom Testfeld übernommen werden sollen und wie externe Datenanalysewerkzeuge angebunden werden können.

Entwicklung In wenigen der untersuchten Testfelder wird ausdrücklich auf die Anforderungen zur Entwicklungsunterstützung von Testsystemen eingegangen. Aktivitäten wie bspw. DRIVE C2X erproben eine abgesteckte Klasse von Anwendungen mit einem hohen Reifegrad. Die Nutzung eines Testfeldes zur Entwicklungszeit ist auch in [BVJ+11] nicht deutlich verfolgt. Dennoch ist diese Aufgabe für ein Testfeld relevant und eine entsprechende Unterstützung als Komponente eines Testfeldes vorzusehen. Das Testfeld DITCM ist ausdrücklich zur Entwicklungsunterstützung gedacht [Aut14, S. 2], fokussiert jedoch primär auf das Testsystem und dokumentiert unterstützende Mechanismen nur am Rande.

Wie diese Unterstützung definiert wird, hängt stark von den Entwicklungszielen ab. Bei einer prototypischen Entwicklung sind bspw. ein SDK[16] und eine Sandbox[17] ein hilfreiches Werkzeug. Hingegen können zur Erprobung seriennaher Produkte mittels HiL[18]- oder SiL[19]-Konzepten Komponenten aus dem künftigen Produktivsystem in ein Testfeld integriert werden.

Basisdienste Für die Entwicklung von Testsystemen ist ein gewisser Pool an Grundfunktionalität wünschenswert, die schon vom Testfeld bereitgestellt wird. Dies sind im Wesentlichen generische Dienste, die unabhängig von der Ausrichtung des Testsystems übliche Funktionen bereitstellen. So ist bspw. die zeitsynchrone Erfassung von Daten in unterschiedlichen Komponenten des Testsystems und auch des Testfeldes eine wiederkehrende Aufgabe. Als Basisdienst kann daher eine Zeitquelle als ge-

[16] Software development kit (SDK), i. d. R. ein Paket aus Werkzeugen und Dokumenten zur Softwareerstellung

[17] *Sandbox* als Testumgebung zur isolierten Erprobung von neuen o. geänderten Testsystemen; ggf. in einer realistischen Einbettung im Testfeld

[18] Hardware in the Loop (HiL); Integration einer Komponente in das Testfeld, um dieses in einer realistischen Umgebung zu erproben

[19] Software in the Loop (SiL), analog HiL, Integration von Softwaremodulen und -funktionen

meinsame Referenz für Komponenten realisiert werden. Ein komplexerer Basisdienst könnte bspw. die Abwicklung von Finanztransaktionen sein, die als generalisierter Abrechnungsdienst, bspw. für Tickets oder PaYS[20], Testsystemen zur Verfügung steht.

Daher ist eine zentrale Komponente im Testfeld vorzusehen, um Testsystemen für diese Dienste eine generelle und nachhaltige Lösung anzubieten. So lässt sich im Testfeld ein Repository aus Basisdiensten aufbauen, die über das Diensteverzeichnis durch das Testsystem adressierbar sind.

Externe Funktionslogik Nicht alle Funktionen, die zur Versuchsdurchführung benötigt werden, können sinnvoll durch ein Testfeld bereitgestellt werden. So kann bspw. eine Verkehrsbeeinflussungsanlage angesteuert werden oder muss ein im Testfeld bezahltes Ticket beim Verkehrsanbieter registriert werden. Naturgemäß sind dies heterogene Schnittstellen, die stark von den Kommunikationspartnern abhängen. Die homogenisierte Bereitstellung solcher Funktionen in das Testfeld ist Aufgabe der Externen Funktionslogik.

Externe Datenquellen Analog zu der Externen Funktionslogik beruhen einige Testsysteme auf externen Datenquellen, die nicht vom Testfeld selbst bereitgestellt werden können. Bspw. können Sensoren in der Fahrbahn oder Wetterdaten von externen Quellen einfließen. Zudem können ggf. Planungsdaten zur Wartung bzw. Instandhaltung vom Anbieter bezogen werden, wie bspw. in AIM umgesetzt. Solche Schnittstellen müssen analog zur Externen Funktionslogik harmonisiert im Testfeld verfügbar gemacht werden.

Abbildung 3.7 zeigt ein vereinfachtes Verteilungsdiagramm der Versuchszentrale mit mehreren Testsystemen und den oben ausgeführten Elementen der Versuchszentrale. Der Kommunikationsaspekt ist in diesem Darstellung durch entsprechende Schittstellen hervorgehoben.
Nachdem die wesentlichen Strukturen aus den durch das Testsystem vorgegebenen Anforderungen top-down festgelegt sind, kann die feinere Ausprägung der Systemkomponenten vorgenommen werden. Dazu wird im Folgenden das im Abschnitt zur Prozess- und Entwicklungssicht aufgebaute Beispiel zur Probandeninteraktion aufgegriffen:

[20] ein System zur Konditionierung des Fahrers zur Risikovermeidung über finanzielle Anreize, vgl. [AWT+08]

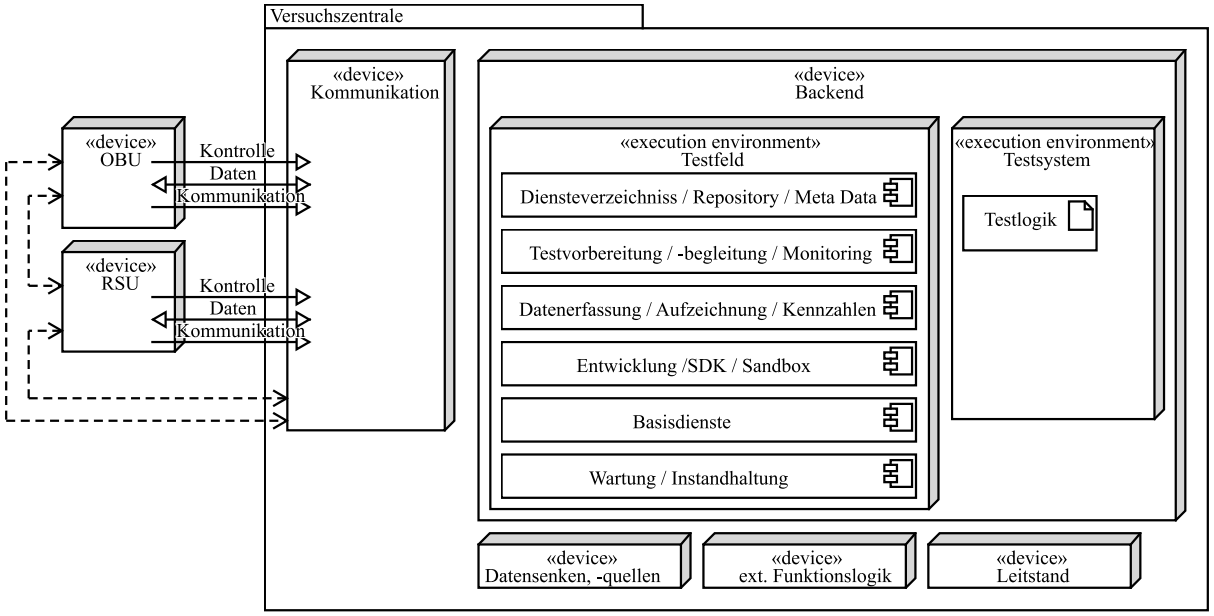

Abbildung 3.7: Komponenten der Versuchszentrale

In Abb. 3.3 wurde diese Aktivität in atomaren Diensten (bzw. *Automated Activities*) skizziert und daraus das in Abb. 3.5 gezeigte Paketdiagramm entwickelt. Unter der Annahme, dass der Proband *Issue*- und *Repair*-Requests über eine im Fahrzeug verbaute OBU oder ein Mobilgerät vornehmen wird und das technische Personal diese Meldungen gebündelt am Leitstand bearbeitet, ist eine Verteilung der Funktion über die Systemkomponenten OBU und Versuchszentrale modellierbar. Entsprechend kann jeweils die Interaktion des Probanden mit der OBU in deren Unterstützungslogik realisiert werden und die des technischen Personals nutzt die Systemkomponenten Leitstand und Basisdienste für diesen Zweck.

Dieses Beispiel zeigt, wie sich die Prozesse und Elemente der Entwicklungssicht zu einer geschlossenen Sicht auf die Verteilung von Funktionalität (Dienste, Prozesse) in der Testfeldsystemlandschaft zusammenstellen lassen. Als Ergebnis liegt dann eine Systemkomponentensicht vor, die die vier weiteren Sichten zu einer umfassenden Architekturbeschreibung im Sinne des von [Kru95] vorgeschlagenen *4+1 View Model* vervollständigt.

3.9 Zusammenfassung

Wesentliches Ziel dieser Arbeit ist die Erarbeitung einer strukturierten Methode, um eine geeignete Systemarchitektur für eine Testfeldlandschaft auszuprägen. In diesem

Abschnitt wurden einige grundsätzliche, den Entwurf bestimmende, Annahmen formuliert. Auf dieser Basis wurde eine eigene Methode vorgestellt, die auf der Kombination der Architekturbeschreibung in *4+1-Sichten* mit der funktionalen Zerlegung zur Gestaltung dienstorientierter Architekturen aufsetzt. Die so abgeleiteten Entwicklungsschritte wurden exemplarisch durchgeführt und dargestellt.
Analog zu der in Abschnitt 3.2 dargestellten Methode wurde Entwicklung als repräsentatives Szenario gewählt. Daraus konnten die wesentlichen Rollen in Abschnitt 3.3 und 3.4 identifiziert werden. Die Zerlegung im Rahmen des User Experience Process (Abschnitt 3.5) konkretisiert eine repräsentative Interaktion zwischen einem Probanden und dem Testfeld. Dies erlaubt eine Modellierung von Prozessen, wie in Abschnitt 3.6, und eine exemplarische Definition der Systementwicklungssicht passend zum UEP (Abschnitt 3.5). Die Entwicklung der Systemkomponenten in Abschnitt 3.8 vervollständigt die Sichten zu einer umfassenden *4+1* Architekturbeschreibung, die letztendlich als Grundlage für eine Realisierung dienen kann. In Anlehnung an Abb. 3.1 sind die für das Beispiel des *Log Book* relevanten Ergebnisse dieser Schritte sind in Abb. 3.8 stark vereinfacht dargestellt.
In diesem Abschnitt konnte ein umfassender Architekturprozess dargestellt werden, der über verschiedene Zerlegungs- und Syntheseschritte eine Architektur aufbaut, die aus funktionaler Sicht mit den Anforderungen adäquat korreliert.
Die Architekturmethode beachtet relevante nicht-funktionale Anforderungen an ein Testfeld. Das dargestellte Verfahren bietet keine Garantie, dass alle nicht-funktionalen Anforderungen im Entwurf adäquat berücksichtigt wurden. Die nachfolgende detailliertere A-posteriori-Betrachtung der nicht-funktionalen Aspekte des Entwurfs sichert diesen dahingehend ab und liefert eine qualitative Bewertung der entwickelten Architektur.

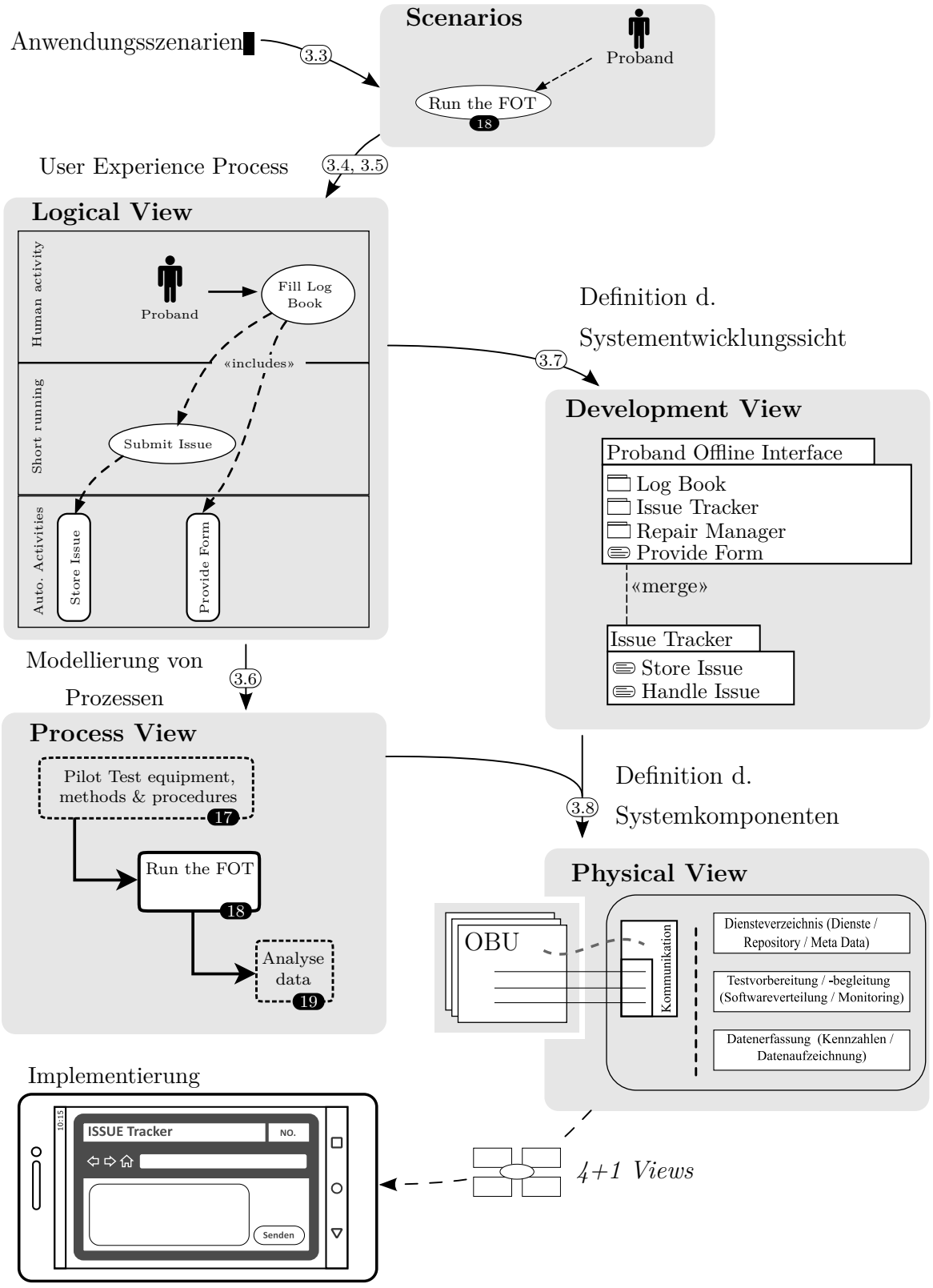

Abbildung 3.8: Zusammenhang und Artefakte der Architekturentwurfsschritte

4 Architekturbewertung

In den Grundlagen wurde die enge Kopplung des Architekturentwurfs mit den qualitativen Eigenschaften des daraus entwickelten Softwaresystems dargestellt. Gleichzeitig ist die Architektur eines der ersten Artefakte in der Entwicklung eines Softwaresystems. Daher ist es u. a. aus wirtschaftlichen Gründen wünschenswert, möglichst frühzeitig die qualitativen Eigenschaften des Zielsystems zu validieren und kritisch zu bewerten, so kann bspw. das Risiko einer Fehlentwicklung in dieser Entwurfsphase minimiert werden. In Kapitel 3 ist ein eigener, auf Testfeldarchitekturen zugeschnittener, Entwurfsprozess vorgestellt worden, dessen Ergebnis eine Architektur in *4+1-Sichten* ist. Obwohl der Entwurfsprozess sorgsam die funktionalen Aspekte berücksichtigt, sind insbesondere die emergenten nichtfunktionalen Anforderungen in der Architektur qualitativ zu bewerteten.
Dieses Kapitel verfolgt die Bewertung von Testfeldarchitekturen. Dazu werden zunächst die speziellen Rahmenbedingungen bei der Bewertung solcher Architekturen in Abschnitt 4.1 ausgeführt.
Die genaue Zielsetzung und die eigene Methode werden in Abschnitt 4.2 vorgestellt und diskutiert. Letztere wird im Abschnitt 4.3 am eigenen Testfeldentwurf zum Einsatz gebracht. Die Ergebnisse der Bewertung werden in Abschnitt 4.4 diskutiert.
Zur vergleichenden Bewertung eines Architekturentwurfs mit anderen (bestehenden) Entwürfen wird in Abschnitt 4.5 ein entsprechendes Verfahren entwickelt. Dies vergleicht das eigene Ergebnis mit den zuvor diskutierten Testfeldstrukturen aus sim^{TD} und ITS Test Beds. Mit der Interpretation dieser Bewertung in Abschnitt 4.6 und der Integration entsprechender Erkenntnisse in Architekturprozess in Abschnitt 4.7 schließen die Überlegungen zur Architekturbewertung ab.

4.1 Grundlegende Bedingungen

In Kapitel 3 wurde ein Architekturprozess für eine Testfeldsystemlandschaft vorgestellt. Dessen Ergebnis ist eine komplexe Architektur. Deren Qualität insbesondere hinsichtlich

nicht-funktionaler Anforderungen ist nicht trivial zu bewerten. Obwohl die einzelnen Architekturentscheidungen in diesem Prozess gut abgewogen werden, ist der Architekturentwurf insgesamt qualitativ zu bewerten und in Relation zu bestehenden Architekturkonzepten zu betrachten. Aus diesem Blickwinkel lassen sich neben charakteristischen Attributen auch mögliche Kompromisse, Risiken und kritische Aspekte des Entwurfs aufdecken. Prinzipiell können dabei die in Abschnitt 2.13 dargestellten Methoden zur Bewertung von Architekturen eingesetzt werden. Im Fall einer Testfeldarchitektur sind dabei einige besondere Herausforderungen in der Domäne gegeben:

- Die Testfeldarchitektur ist selbst Werkzeug zur Bewertung von komplexen Systemen, deren genaue funktionale Ausrichtungen prinzipiell nicht hart umrissen und zudem auch signifikanten Änderungen über die Lebenszeit hinweg unterworfen sind. Dies macht die Untersuchung von nicht-funktionalen Anforderungen wie Austauschbarkeit und Interoperabilität besonders wichtig.

- Durch die Einbettung in ein Testfeld und insbesondere durch die Simulation gewisser Funktionen des Zielsystems ergeben sich eine Reihe grundsätzlicher Fragestellungen, die in dieser Phase zu bewerten sind. Von besonderer Relevanz sind bspw. jene, die sich mit der Wechselwirkung des Testfeld und Testsystem ergeben: Entspricht das beobachtet Verhalten genau dem des eigenständigen Zielsystems?

- Aus der Nachbildung von Funktionalität durch das Testfeld ergeben sich Fragestellungen nach der Treue (*fidelity* [LMV09, S. 62]) gegenüber dem Verhalten entsprechender Realkomponenten. Beide Aspekte führen zu kritischen Abwägungen (*trade-offs*) in der Bewertung.

- Abschnitt 3.1 legt zwar grundlegende Anforderungen für eine Testfeldarchitektur fest, diese sind jedoch für den konzeptionellen Entwurf nicht genau quantifiziert. Daher ist eine präzise quantitative Bewertung auf Grundlage der in den Anforderungen spezifizierten Kennzahlen nicht möglich. Hier ist eine Feststellung der durch den Architekturentwurf bedingten Systemgrenzen möglich, die bei einer späteren Überführung in ein Realsystem Hinweise auf dessen Einschränkungen und Kompromisse geben kann.

- Die in Abschnitt 2.13 ausgeführten Methoden nutzen zum überwiegenden Teil Expertenteams im Bewertungsprozess. In der Einbettung in iterative oder sogar agile

Architekturprozesse (vgl. Abschnitt 2.14) ist die bspw. von SAAM und ATAM vorgeschlagene Interaktion mit einem Expertenteam nicht unproblematisch durchführbar. Es muss daher ein Verfahren ohne eine solche Interaktion Anwendung finden.

- Obwohl die in der Motivation ausgeführten Vorzüge einer nachhaltigen Architektur evident scheinen, stellen der Testfeldaufbau sowie die Integration eines Systems in eine komplexe Testfeldlandschaft einen Aufwand dar. Entsprechend muss der Testfeldentwurf hinsichtlich seiner Kosten und Nutzen untersucht werden.

Nach dieser Konkretisierung können Zielsetzung und Methode der Architekturbewertung im folgenden Abschnitt ausgeführt werden.

4.2 Zielsetzung und Methode

Vor der eigentlichen Untersuchung muss definiert werden, welche Qualitätsmerkmale der Architektur betrachtet werden sollen. Die umfassende Aufstellung solcher Merkmale in ISO/IEC 9126 (s. Abb. 2.15) kann nur als grobe Vorauswahl der relevantesten Qualitätsmerkmale dienen. Prinzipiell sind alle Merkmale dieser Norm untersuchbar. Aus pragmatischer Sicht ist dies nicht immer sinnvoll. Daher sollte eine möglichst geschickte Vorauswahl erfolgen, welche die relevantesten Merkmale identifiziert, um sicherzustellen, dass zumindest die relevantesten Anforderungen im Architekturentwurf repräsentiert sind. Vor der eigentlichen Entwicklung eines komplexen Systems, wie hier ein Testfeld, steht i. d. R. eine umfassende Anforderungsanalyse. In dieser werden die Anforderungen an das Testfeld erhoben und ausgeführt. In vielen Vorgehensmodellen zur Softwareentwicklung sind auch Gewichtungen der einzelnen Anforderungen vorgesehen. Dies ist bspw. bei dem Rapid Application Development (RAD) ein Instrument, um die Reihenfolge und Wichtigkeit der Implementierungen festzulegen. Die Gewichtung spielt auch eine Schlüsselrolle im Qualitätsmanagement der Softwareentwicklung, bspw. verwendet [FP98] auf den Anforderungen basierende Softwaremetriken u. a. zum Controlling und Management von Softwareentwicklungsprozessen.
In der Anforderungsanalyse können dazu verschiedene Techniken eingesetzt werden. Ein bekanntes Verfahren ist bspw. die von [LT64] vorgeschlagene Conjoint-Analyse. Dies ist eine multivariate Analysemethode, die darauf abzielt, Details des Zielsystems durch Expertenteams (bspw. Nutzer) in seinen Eigenschaften zu bewerten. Es kann prinzipiell zu jedem Merkmal eines Produktes und auch Merkmalsklassen eine gewisse Präferenz durch

die Experten bestimmt werden. Ein solches Analyseergebnis kann dann benutzt werden, um die Anforderungen zu priorisieren und, darauf aufbauend, Designentscheidungen zu treffen.

Im Rahmen einer Architekturbewertung ist es sinnvoll, aus den Gewichtungen der Anforderungen die wichtigsten Qualitätsmerkmale für diese Analyse abzuleiten. Sind diese bestimmt, so weisen die am höchsten priorisierten Anforderungen auf besonders kritische Qualitätsmerkmale hin, die in einer Architekturbewertung untersucht werden sollten. Daher kann auch aus Sicht der Qualitätssicherung ein fundierter Kompromiss zwischen qualitativer Teilbetrachtung und dem dafür notwendigen Aufwand getroffen werden.

Wie in der Einleitung schon ausgeführt, ist im Rahmen dieser Arbeit die Anforderungsanalyse u. a. auf Basis bestehender Systeme und der darin reflektierten Anforderungen aufgesetzt. Ebenso sind die Anforderungen in diesem Fall nicht durch ein Expertenteam gewichtbar. Deshalb kann die typische Ableitung kritischer Qualitätsmerkmale aus priorisierten Anforderungen hier nicht angewendet werden. Die Anforderungen für den Architekturentwurf sind in ITS Test Beds nicht feingranular genug dokumentiert, um daraus eine stabile Priorisierung von Anforderungen abzuleiten. Der in [BVJ+11] dargestellte Entwurfsprozess stützt sich wesentlich auf die Analyse bestehender Aktivitäten wie bspw. FESTA und CVIS[1]. Eine priorisierte Auflistung von Einzelanforderungen ist nicht dokumentiert.

Zwar beschreibt [HNS+10] die Anforderungserhebung in sim^{TD}, jedoch beschränkt sich dieses Dokument auf die Methodik der Anforderungserfassung und zeigt einen stark verdichteten Überblick auf die 664[2] Anforderungen, die in diesem Projekt diskutiert wurden. Es wird eine quantitative Priorisierung aller Anforderungen vorgenommen in den Kategorien *Normal* (0 ... 300), *Wichtig* (301 ... 1200) und *Kritisch* (1201 ... 3000). Somit wäre aus den sim^{TD}-Anforderungen eine gute Ableitung möglich. Obschon [HNS+10, S. 21] eine Auflistung der einzelnen Anforderungen referenziert, ist diese Quelle nicht öffentlich zugänglich und steht somit als Referenz nicht zur Verfügung.

Die grundlegenden Überlegungen in Abschnitt 3.1 können hier als Hinweis auf besonders gewichtige und kritische Anforderungen dienen, um daraus korrelierende Qualitätsmerkmale nach ISO/IEC 9126 für die folgende Untersuchung abzuleiten. Es sind folgende Qualitätsmerkmale maßgeblich:

Interoperabilität Zweck eines Testfeldes ist u. a. die Integration unterschiedlichster Kom-

[1] vgl. [BVJ+11, S. 11ff u. 42ff]

[2] s. [HNS+10, S. 19]

ponenten, um daraus einen Versuchsaufbau bereitzustellen. Die Fähigkeit, verschiedenartige Komponenten über Standardisierungen in komplexen Funktionsnetzwerken interagieren zu lassen, ist eine der Kernanforderungen an ein Testfeld.

Austauschbarkeit Die eingesetzten Funktionen müssen, unabhängig davon, ob dies testfeldeigene oder vom Testsystem bereitgestellte Komponenten sind, austauschbar in das Testfeld integriert sein. Folglich ist bspw. gewährleistet, dass über eine Kampagne ggf. unterschiedliche Komponenten zum Einsatz kommen (HiL o. SiL), defekte Komponenten mit Vergleichstypen ersetzbar sind oder auch ganze Funktionsgruppen im Rahmen einer Modernisierung austauschbar sind.

Analysierbarkeit Das Testfeld ist Werkzeug zur Bewertung des eingebetteten Testsystems. Ohne fundierte Annahmen über das Verhalten des Testfeldes selbst ist praktisch keine Bewertungsmöglichkeit gegeben. Praktisch relevant ist hier bspw. das Laufzeitverhalten der Testfeldkomponenten. So ist bspw. die Latenz eines virtuellen Kommunikationskanals relevant, da dies direkt in das Kommunikationsverhalten des Testsystems einfließt. Das Qualitätsmerkmal beschreibt dazu die Möglichkeit, inwieweit das Testfeldsystem in dieser Hinsicht untersuchbar ist. Eine möglichst gute Analysierbarkeit ist zudem Grundlage für weitere Bewertungen. Daher basieren bspw. Abschätzungen zur Weiterentwicklung auf möglichst genauer Kenntnis des Ursprungssystems.

Änderbarkeit und Migrationsfähigkeit Die Forschungs- und Erprobungsschwerpunkte im Testfeld können sich über die Laufzeit ändern. Daher wird am Testfeld, wie an jeder anderen Software auch, die Notwendigkeit entstehen, auf geänderte Anforderungen zu reagieren. Ebenso ist im Rahmen einer kontinuierlichen Verbesserung des Systems auch eine Weiterentwicklung und Korrektur von Fehlern vorzusehen. Die Änderbarkeit ist dabei ein Merkmal, das den Aufwand unter diesen Gesichtspunkten bemisst.

Koexistenz In den Überlegungen zur Versuchszentrale sind mehrere Testsysteme im Testfeld anzunehmen. Ebenso sind die Testsysteme selbst aus Diensten aufgebaut, die gleiche oder ähnliche Funktionalität bereitstellen. Beispielsweise ist ein Vorhalten von unterschiedlichen Routingalgorithmen denkbar, die probeweise in einem Testsystem zum Einsatz kommen. Zudem sind viele Aspekte des Testfeldes redundant ausgelegt

und es bedarf zumindest einer rudimentären Arbitrierung, um diese sinnvoll zu nutzen. Das Qualitätsmerkmal der Koexistenz betrachtet die Fähigkeit des Testfelds, mit ähnlichen oder gleichen Funktionen umzugehen.

Die hier skizzierte Auswahl aus möglichen Qualitätsmerkmalen repräsentiert prominente Anforderungen aus Abschnitt 3.1 und ist keinesfalls als umfassend zu verstehen. Für eine weitere Bewertung der Architektur hat diese Auswahl dennoch Bestand, da die so identifizierten Qualitätsmerkmale vor dem Hintergrund der in Kapitel 2 ausgeführten Anforderungen als wesentliche verstanden werden können. Es verschmelzen Aspekte aus den unterschiedlichen Merkmalen aus ISO/IEC 9126 in der Betrachtung. Änderbarkeitsbewertungen im Rahmen der Wartbarkeit sind bspw. inhaltlich eng verwandt mit der Betrachtung der Wandlungsfähigkeit im Kontext der Portabilität.
Nach Auswahl der Qualitätsmerkmale ist nun das weitere Vorgehen bestimmbar. Das in [EHM07] beschriebene Verfahren (s. Abb. 4.1) kann nun als Wegweiser zur Auswahl einer (oder mehrerer) Bewertungsmethodiken dienen.

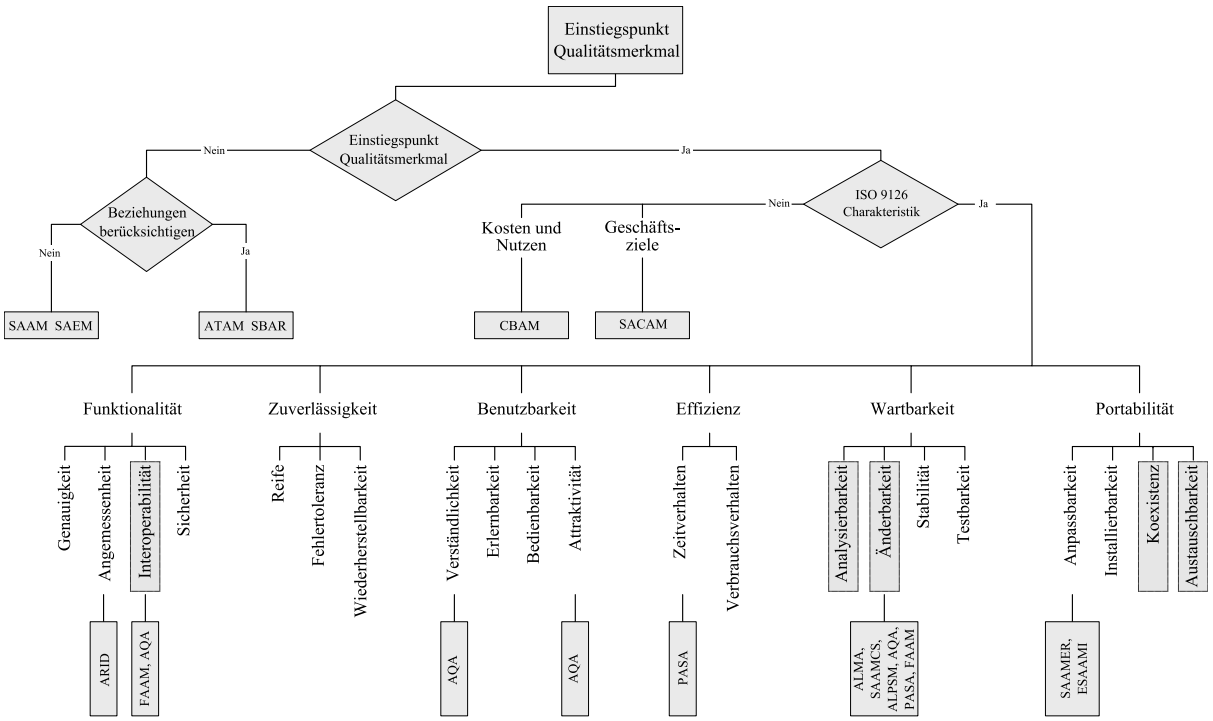

Abbildung 4.1: Auswahl von Bewertungsmethoden über gewählte Qualitätsmerkmale, angelehnt an [EHM07]

Es muss entschieden werden, ob die Wechselwirkung von Qualitätsmerkmalen betrachtet werden soll oder ob darauf verzichtet werden kann. Die Exploration reziproker Merkmals-

kombinationen ist durchaus relevant, wie in Abschnitt 4.1 dargelegt. ATAM und SBAR werden dazu von [EHM07] vorgeschlagen. Für die isolierte Bewertung der fünf gewählten Merkmale weist [EHM07] FAAM und AQA für die Bewertung von Interoperabilität sowie ALMA, SAAMCS, ALPSM, AQA, PASA und FAAM für die Bewertung von Änderbarkeit als besonders geeignet aus. Für Analysierbarkeit, Koexistenz und Austauschbarkeit gibt [EHM07] keine Empfehlung. Also wäre die Auswahl für die Betrachtung (ohne Trade-offs) auf FAAM und AQA eingrenzbar. Da jedoch die Untersuchung von Trade-offs zwischen den Qualitätsmerkmalen durchaus relevant ist, jedoch kein Reengineering verfolgt wird, ist somit ATAM vorzuziehen, das sowohl zur Bewertung von isolierten Merkmalen ([KKC00] illustriert den ATAM-Ansatz über die Attribute *modifiability*, *security*, *performance* und *availability*) als auch zur Bewertung von Trade-offs dient. Letzteres macht ATAM zu einem guten Startpunkt, um die in Abschnitt 4.1 ausgeführten Herausforderungen zu adressieren.

Da möglichst auf ein Expertengremium verzichtet werden soll, muss ein adaptiertes ATAM-Verfahren eingesetzt werden, welches diese Einschränkung berücksichtigt. In der Literatur werden einige leichtgewichtige ATAM-Derivate diskutiert, wie bspw. das pragmatische *mini-ATAM* aus [CKK01, S. 140]. Dieses aufgreifend wird ein eigener, verschlankter Ansatz in drei Phasen verfolgt. Gemäß der in Abschnitt 4.1 dargestellten Anforderungen an die Bewertungsmethode verzichtet der eigene Ansatz auf eine Interaktion mit Experten- bzw. Evaluationsteams (vgl. [KKC00, S. 25ff] und kann so den neunstufigen ATAM-Ansatz in drei Phasen verdichten:

Phase I Zu Beginn werden die wesentlichen Architekturmuster der Architektur identifiziert. Es wird eine Begründung für deren Auswahl gegeben und aufgeführt, welche Zielsetzungen des geplanten Systems dadurch adressiert werden. Im Rahmen dieser Arbeit können dazu Überlegungen aus Kapitel 3 referenziert werden. Die dort beschriebenen Designentscheidungen und vorgeschlagenen Technologien sind hinreichend, um die folgenden Phasen zu unterstützen.

Phase II In diesem Arbeitsschritt werden relevante Szenarien aus der Anforderungsdefinition gewählt und priorisiert. Auf Grundlage der Szenarien und Designentscheidungen wird der aus ATAM übernommene Utility Tree (UT) aufgebaut. In diesem sind die gewählten Qualitätsmerkmale, relevante Aspekte (*concerns*) der Szenarien und nach Wichtigkeit sowie Realisierungsaufwand priorisierte (Einzel-)Anforderungen strukturiert. Diese Phase wird detailliert in Abschnitt 4.3 ausgeführt.

Phase III Die eigentliche Untersuchung der Architektur wird durchgeführt. Dazu wird im Wesentlichen der UT interpretiert.

Der UT identifiziert Anforderungen mit gleichermaßen hoher Wichtigkeit und hohem Realisierungsaufwand. Dies sind für die Implementierung des Zielsystems besonders risikoreiche Aspekte. Ein guter Architekturentwurf sollte möglichst wenig kritische Aspekte aufweisen. Deren Reduktion ist ein typisches Optimierungsziel eines Architekturprozesses. Durch die Darstellung von Attributen mit niedriger Wichtigkeit und hohem Realisierungsaufwand kann eine Verhältnismäßigkeit abgeleitet und eine Bewertung zu Wirtschaftlichkeit des Systementwurfs in diesem Aspekt angegeben werden.

Weitere Untersuchungsgegenstände können empfindliche Aspekte sein. Dies sind signifikante Relationen zwischen Bewertungen und Designentscheidungen. So wird aufgedeckt, welche Bewertungen durch welche Designentscheidungen bedingt sind. Aus gegenläufigen Wirkfaktoren lassen sich dann Kompromisse im Entwurf ausmachen. Die Analyse des UT wird in Abschnitt 4.4 detailliert ausgeführt.

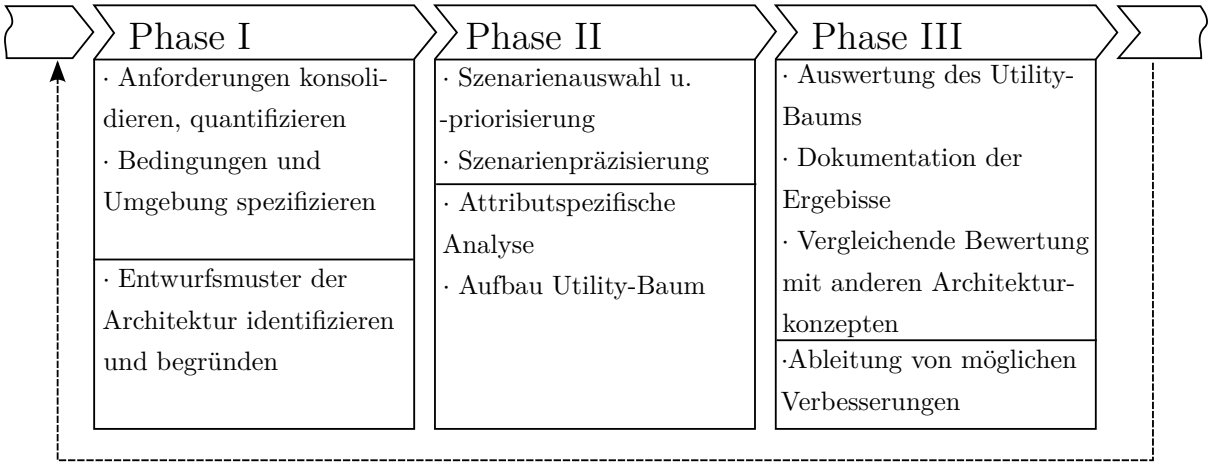

Abbildung 4.2: Angepasstes ATAM-Verfahren

Abbildung 4.2 stellt die oben beschriebenen Phasen des angepassten Verfahrens dar und illustriert eine mögliche Iteration der Phasen im Rahmen eines Entwurfsprozesses.
So ist eine umfassende Bewertung auf Grundlage der gewählten Qualitätsmerkmale und Szenarien möglich. Durch den UT wird auch ein pragmatischer Verbesserungsprozess unterstützt. Dazu können Architekturkonzepte variiert und der UT an entsprechende

Schlüsselstellen angepasst werden. So sind die Effekte der Veränderung zur Entwurfszeit und später auch im Change Management[3] gut nachvollziehbar. Zudem ist ein solcher Prozess gut in den Entwicklungsprozess einbettbar (vgl. Abb. 2.16).
Im folgenden Abschnitt wird das hier umrissene Verfahren exemplarisch ausgeführt und die abgeleiteten Ergebnisse diskutiert.

4.3 Architekturbewertung

Im vorangehenden Teil wurden die Qualitätsmerkmale Interoperabilität, Austauschbarkeit, Analysierbarkeit, Änderbarkeit und Koexistenz zur Untersuchung bestimmt. Nun wird das, im letzten Abschnitt beschriebene, dreiphasige Bewertungsverfahren angewendet. Die ersten beiden Phasen werden in diesem und der Interpretationsteil der dritten Phase im folgenden Abschnitt ausgeführt.
In der ersten Phase des Bewertungsprozesses wird eine Identifikation der Schlüsseltechnologien des Architekturentwurfs durchgeführt. Dies wird später eine Beurteilung der Vor- und Nachteile dieser Lösungen ermöglichen. Die vorgeschlagenen Technologien sind in Kapitel 3 ausführlich dargestellt.
In der zweiten Phase wird der UT aufgebaut. Dazu wird jedes gewählte Qualitätsmerkmal einer Reihe von Aspekten zugeordnet, die dann in Szenarien nach Wichtigkeit und Realisierungsaufwand bewertet werden. Dazu stützt sich diese Phase auf die in den Grundlagen ausgeführten Anforderungen an ein Testfeld und referenziert ebenso die Grundannahmen in Abschnitt 3.1. Die Bewertung erfolgt dann auf den in der ersten Phase, bzw. den in Kapitel 3 ausgeführten Architekturkonzepten. Dabei wird eine zyklische Evidenz dadurch vermieden, dass die in Kapitel 3 dargestellten funktionalen Gestaltungsaspekte im Rahmen der Architekturbewertung anhand der nicht-funktionalen Eigenschaften betrachtet wird.
Die folgende Aufzählung führt auf Basis der zuvor gewählten Qualitätsmerkmale und ausgewählter Anwendungsfälle relevante Szenarien genauer aus, gibt die Wichtigkeit sowie den geschätzten Realisierungsaufwand an und begründet diese Bewertung. Die Bewertung erfolgt anhand der im Kontext der UT bevorzugten Skala[4] mit den Skalenwerten H, M und L, siehe Tab. A.5 (im Anhang, S. 184). Es werden folgende Bewertungen gegeben:

[3] Change Management überwacht Modifikationen an bestehenden Systemen, um die Effekte der Änderungen auf das Gesamtsystem zu bewerten und potenzielle Komplikationen, wie bspw. Wechselwirkungen, Seiteneffekte und Inkompatibilität, im Vorfeld auszuräumen.
[4] vgl. *relative rankings* in [KKC00, S. 18]

Interoperabilität Die reibungslose Verschaltung von ggf. heterogenen Komponenten im verteilten Testfeld ist eine Kernanforderung an die Systemlandschaft.

1 (H,L): Interoperabilität spielt in der Erprobung von Testsystemen selbst eine große Rolle. So ist bspw. die Interoperabilität von Kommunikationsmedien (bspw. 802.11p) im Kontext kooperativer Assistenzen ein gängiger Untersuchungsgegenstand im Testfeld. Das Testfeld muss Funktionen anbieten, die diese Kommunikation überwachen und ggf. Probleme bei der Interoperabilität darstellen können. Durch Redundanz-, Monitoring- und Virtualisierungskonzepte sollte diese Anforderung gut unterstützt werden. Konkret werden die drei Aspekte über die redundante Ausgestaltung der Einheiten OBU und RSU (vgl. Abb. 3.6) adressiert. Über die Gestaltungsidee *Kontrolle, Daten und Kommunikation* (vgl. Abs. 3.8) werden Monitoring sowie Virtualisierung funktional adressiert. Die stützt die Abschätzung, dass Interoperabilität in der Erprobung mit geringem Aufwand realisiert werden kann.

2 (H,L): Heterogene Komponenten müssen sinnvoll miteinander bspw. zu Versuchsaufbauten verschaltet werden können. Dazu sind die Komponenten im Testfeld mit eigenen Adaptern in die SOA-Landschaft eingebunden. Deren Schnittstellen sind standardisiert. So ist ein einheitlicher Zugriff auf solche Komponenten, unabhängig von den zugrunde liegenden inhärenten Schnittstellen der Komponenten, möglich. Somit sind unterschiedliche Komponenten gleicher Funktionalität dennoch effizient und sinnvoll nutzbar. Die generellen Vorzüge einer Dienstorientierung in diesem Aspekt werden u. in Abs. 2.8 und 2.9 erörtert. Die Homogenisierung und Adaption dieser Komponenten in die SOA und Standards ist daher als vergleichsweise gering einzuschätzen.

3 (H,L): Eine Referenzplattform kann genutzt werden, um bspw. neue Sensoren in bestehenden Testsystemaufbauten zu testen (HiL). Zudem ist hier der in (2) beschriebene Aufwand zur Anpassung bzw. Reimplementierung der Adapter notwendig. Dieser ist jedoch nicht erheblich.

Austauschbarkeit Die Austauschbarkeit ist ein wichtiger Aspekt für nahezu alle Verwendungszwecke eines Testfeldes. Im Kontext der Architekturbewertung sind besonders Entwicklung, Erprobung und Demonstration von dieser Systemanforderung geprägt:

4 (H,L): Während der Entwicklung werden praktisch kontinuierlich Entwicklungsreleases erstellt. Diese müssen dann in das Testfeld eingebunden werden. Dies entspricht einem ständigen Austausch von Schlüsselfunktionalität und ist durch eine flexible

und anpassbare Orchestrierung leicht umsetzbar. Analog zu (2) ist dafür Aufwand zur Anpassung bzw. Reimplementierung der Adapter notwendig, dieser ist jedoch als *low* abzuschätzen.

5 (H,L): Ergänzend zu (4) ist es notwendig, dass durch einen solchen Tausch parallele Kampagnen nicht beeinflusst werden. In Abschnitt 3.8 wird bspw. das Sandbox-Konzept zur Lösung dieser Problematik vorgeschlagen. Somit sollte eine Kapselung von problematischem Funktionscode zur Entwicklungszeit möglich sein. Die Umsetzung solcher Konzepte erscheint wenig aufwendig. XXXXXX

6 (H,L): Das Testfeld besteht aus einer umfangreichen und komplexen Systemlandschaft. Wie bei jedem anderen System auch können Komponenten ausfallen oder müssen im Rahmen einer Modernisierung ausgetauscht werden. Daraus lässt sich ableiten, dass Kernkomponenten des Testfeldes selbst mit geringem Aufwand gegen ggf. modernisierte austauschbar sein müssen. In einem solchen Fall sind die Adaptionen in die SOA des Testfelds anzupassen bzw. zu implementieren. Analog zu (2) stellt dies einen vergleichsweise geringen Aufwand dar.

7 (H,L): Im Rahmen der Erprobung werden vergleichsweise reife Testsysteme bzw. Testsystemkomponenten ins Feld geführt. Daher muss das Testfeld in der Lage sein, insbesondere Fremdkomponenten in Testsystemen zu unterstützen. Dazu werden leicht adaptierbare Schnittstellen für solche Komponenten genutzt und folglich ist mit einem geringen Aufwand für die Einbettung zu rechnen.

8 (H,L): In Demonstrationen wird typischerweise ein relativ reifes Testsystem präsentiert. Diese Systeme können in sich geschlossene oder proprietäre Systeme sein. Das Testfeld muss also in der Lage sein, mit wenig Adaptionsaufwand teils geschlossene Testsysteme zu unterstützen und zu betreiben. Dazu ist eine hohe Flexibilität der Infrastruktur Voraussetzung. Typischerweise ist die Adaption komplexer Systeme eine anspruchsvolle und somit aufwendige Aufgabe. Daher ist in diesem Fall das offene Dienstekonzept hilfreich, an sinnvollen Übergangsstellen die Einbettung möglichst reibungsfrei zu gestalten. Die Adaption kann, analog zu (2), mit *low* abgeschätzt werden.

Analysierbarkeit Die Analysierbarkeit von Testfeldkomponenten und insbesondere Basisdiensten ist ein wichtiger Aspekt. Ohne tief greifende Kenntnisse über die angebotene Funktionalität ist keine Unterstützung für das Testsystem denkbar.

9 (H,L): Damit Testsysteme durch Basisfunktionen aus dem Testfeld unterstützt

werden, muss deren Funktion exakt bekannt und überprüfbar sein. Der Aufwand für eine genaue Spezifikation, Verifikation und ggf. Charakterisierung ist vergleichsweise niedrig.

10 (L,H): Aus der Nachbildung von Funktionalität über das Testfeld ergeben sich Fragestellungen nach der Verhaltenstreue im Vergleich zu entsprechenden Realkomponenten. Die kann sich nachteilig auf die Analysierbarkeit auswirken. Aus Sicht der Erprobung sicherheitskritischer oder zumindest sicherheitsrelevanter Assistenzen ist bspw. das Zeitverhalten der aus dem Testfeld entliehenen Funktionalität kritisch. Obwohl Echtzeitfähigkeit nur für eine begrenzte Menge sicherheitskritischer Testsysteme notwendig ist (niedrige Relevanz L), muss dennoch für alle Basisdienste ein Zeitverhalten angegeben werden, wie bspw. mittlere Antwortzeit oder Latenz. Zudem ist die Verfügbarkeit von Komponenten aus Testsystemsicht kritisch. Das Testsystem muss sich auf die vom Testfeld bereitgestellten Funktionen verlassen können. Dazu muss bestimmbar sein, ob eine Funktion verfügbar ist. Die genannten Punkte sind in SOA-basierten Architekturen grundsätzlich schwierig handhabbar (vgl. Abs. 2.9), daher ist mit entsprechend hohem Aufwand für eine Kompensation zu rechnen.

11 (M,M): Die in (9) und (10) formulierten Anforderungen sind insbesondere für eine Referenzplattform bedeutsam. Die als Referenz angebotenen Dienste müssen in zeitkritischen Testsystemen ein deterministisches Laufzeitverhalten zeigen. Dies ist bei der vorgesehenen Realisierung von Basisdiensten in einer SOA nicht einfach darstellbar.

12 (L,H): Als Referenz muss das Testfeld verlässlichere und leistungsfähigere Komponenten bereitstellen, als dies im Regelbetrieb des produktiven Testsystems notwendig wäre. Durch eine SOA als Grundkonzept ist jedoch eine grundsätzliche Aussage über die Verfügbarkeit eines Dienstes schwierig, da ggf. ein komplexes Dienstenetzwerk mit komplexen inhärenten Eigenschaften die eigentliche Funktion bereitstellt. Aus Sicht der Analysierbarkeit ist hier mit deutlich mehr Aufwand zur Kompensation zu rechnen, um die in Abs. 2.9 dargestellten Risiken eines verteilten Systems hinreichend zu adressieren.

Änderbarkeit Wird das Testfeld als Analyseplattform genutzt, dann ist die flexible Durchführung von sequentiellen als auch parallelisierten Kampagnen zu unterstützen. Diese können experimentellen oder provisorischen Charakter haben und stark chan-

gierende Anforderungen an den unterliegenden Versuchsaufbau stellen. Insbesondere bei der Exploration von Konzepten müssen im Testfeld quasi ad hoc entsprechende Komponenten in der Funktion erweitert bzw. im Verhalten angepasst werden.

13 (H,L): Dienste und Funktionen können im Testfeld in unterschiedlichen Versionen angeboten werden. So kann eine bestehende Funktion übernommen werden und durch Abänderung an die nun vorliegenden veränderten Anforderungen angepasst werden. Die alte Funktion bleibt dabei für andere Kampagnen erhalten und kann parallel eingesetzt werden. Im Einklang mit (2) reduziert sich der Realisierungsaufwand für Änderungen an den Funktionen auf ein Mindestmaß.

14 (H,L): Analog zur Austauschbarkeit kann die Einbindung neuer Komponenten (Hard- und Software) gefordert sein. Diese bedingen ggf. einen erweiterten Funktionsumfang. Dazu muß die Portierung dieser Elemente in die SOA des Testfelds geändert werden. Die dafür notwendigen Adaptionen stellen einen Aufwand dar, durch die Flexibilität (2) reduziert sich der Realisierungsaufwand entsprechend auf ein Mindestmaß.

Koexistenz Die Koexistenz ist in der Testfelddomäne ein ubiquitäre Anforderung in praktisch allen Verwendungsszenarien:

15 (H,L): Zur Entwicklungszeit stellt das Testfeld gewisse Funktionalität virtualisiert oder als Basisdienst bereit (vgl. Abschnitt 3.8). Das Testfeld muss während der Entwicklung in der Lage sein, zwischen Testsystem- und Testfeldfunktionalität überzublenden. Das Testsystem kann dabei entsprechende Funktionen aus testfeldeigenen Diensten entleihen oder eigene Funktionalität nutzen, wie im Kontext der redundanten Ausgestaltung der Einheiten OBU und RSU (vgl. Abb. 3.6) diskutiert. Durch die lose Kopplung der Dienste im Testfeldsystem ist hier zumindest ein Überblenden aus der Dienstebene problemlos möglich. Daher ist der entsprechende Realisierungsaufwand gering.

16 (H,L): Das Testfeld muss in der Lage sein, beliebige Komponenten eines Testsystems parallel zu Referenzkomponenten zu betreiben, während des Testbetriebs zu arbitrieren und ggf. abweichendes Verhalten (bspw. Paketverlust[5]) zu dokumentieren. Dazu sind Redundanzen, bspw. drahtgebundene Kommunikation parallel zu drahtlo-

[5]In einem paketbasierten Netzwerk kann es dazu kommen, dass ein Paket nicht den Empfänger erreicht. Besonders im Kontext drahtloser Übertragungen (bspw. 802.11p) ist dies ein gängiger Untersuchungsgegenstand.

ser Kommunikation, im Testfeld vorgesehen. Das Testsystem nutzt dabei genau einen Kanal, der verbleibende kann dann ggf. zur Kontrolle mitverwendet werden. Das Testfeld kann ggf. auch zwischen den Kommunikationskanälen umschalten. Sind die Kommunikationsschnittstellen genormt, bedarf es dazu keiner aufwendigen Adaption und es liegt ein überschaubarer Realisierungsaufwand vor, die Schnittstellen im Testfeld standardisiert zu halten. Abschnitt 3.8 erläutert die Gestaltungsidee über drei Aspekte (Kontrolle, Daten und Kommunikation), die den Parallelbetrieb von Komponenten mitbedenkt.

Auf Grundlage der hier dargestellten Überlegungen lässt sich nun der gewünschte UT aufbauen. Dessen grafische Repräsentation ist in Abb. 4.3 dargestellt. Dieser Baum ist Bewertungsgrundlage für die im folgenden Abschnitt ausgeführte Interpretation.

4.4 Interpretation

Nachdem im vorangehenden Abschnitt der Aufbau des UT dargestellt wurde, ist nun in der letzten Phase (III) die eigentliche Bewertung der Architektur möglich. Dazu erfolgt eine Untersuchung des UT. Analog zum ATAM-Verfahren werden mögliche Risiken, empfindliche Aspekte[6] und Trade-offs ermittelt.

Risiken Als risikoreich einzustufen sind all jene Blätter des UT, die mit *(H,H)* bewertet sind. Diese sind für den Erfolg der Architektur hochwichtig und gleichermaßen aufwendig in der Realisierung. Im Baum (Abb. 4.3) kommen diese nicht vor. Jedoch sind die Bewertungen 10 (L,H), 11 (M,M) und 12 (L,H) bemerkenswert. Solche Bewertungstupel im UT sind üblicherweise ungünstig und somit Ansatzpunkt von Optimierungen. Es kann in einer weiteren Iteration des Architekturprozesses nachgebessert werden, indem bspw. komplexe Komponenten in mehrere kleinere und einfacher zu realisierende Komponenten zerlegt werden.[7]

Auffällig ist im UT ein Vorherrschen von *(H,x)*-Tupeln. Dies liegt schlicht an der Auswahl der Bewertungsaspekte, die sich an möglichst wichtigen Anforderungen orientiert. Daher spiegeln sich im gezeigten UT nur wenige niedriger priorisierte Bewertungen.

[6] vgl. *sensitivities* in [KKC00, S. 33]

[7] *Divide and conquer*, geläufiges Paradigma zur Komplexitätsreduktion, bei dem das Gesamtproblem in kleinere, leichter beherrschbare (*conquer*) Teilprobleme unterteilt (*divide*) wird.

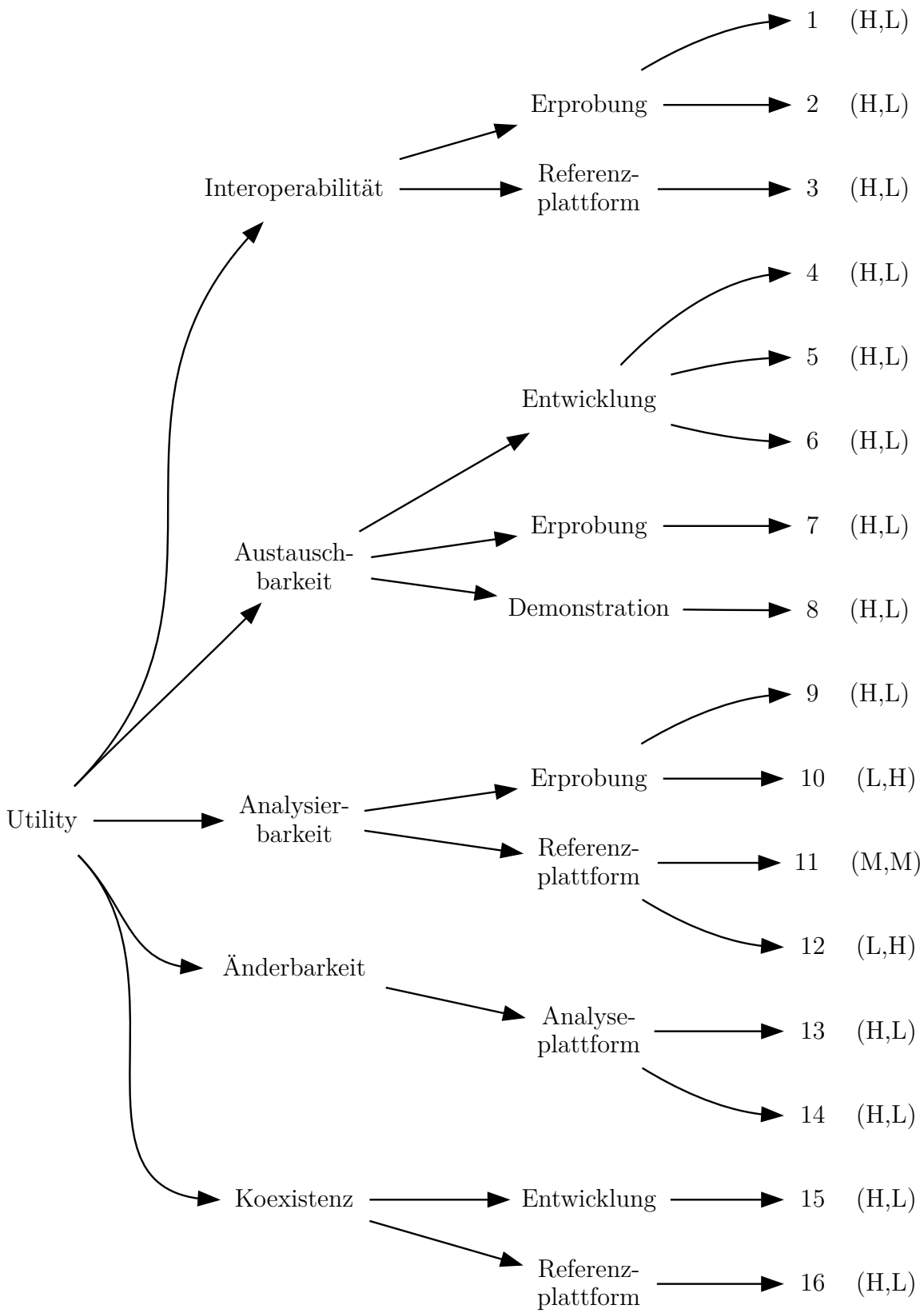

Abbildung 4.3: ATAM-Baum

Empfindliche Aspekte Dies sind Entwurfsentscheidungen, die direkte und deutliche Wirkfaktoren für die Bewertungen sind. Im hier diskutierten Beispiel ist eine solche Entscheidung der flexible Architekturentwurf auf Grundlage einer Diensteorientierung. Diese Entscheidung wirkt sich positiv auf eine Reihe von Bewertungsaspekten aus. Mit genau diesem Architekturmerkmal korrelieren jedoch auch die negativen Beurteilungen im Kontext der Analysierbarkeit hinsichtlich Zeitverhalten, Verfügbarkeit und Determinismus der so orchestrierten Testfeldkomponenten. Somit ist die Architektur empfindlich gegenüber veränderlichen Anforderungen, die vorsehen, das Testfeld bspw. mehr im Kontext sicherheitskritischer Assistenzen einzusetzen. Solche Testfeldziele werden also nur schwach unterstützt.

Die Untersuchung der empfindlichen Aspekte offenbart also Abhängigkeiten zwischen Zielsystemeigenschaften und Technologiekonzepten. Daraus ableitbar sind potentielle Schwachpunkte des Architekturentwurfs im Kontext zukünftiger Adaptionen. Korreliert man empfindliche Aspekte mit der Wahrscheinlichkeit des Auftretens respektiver Anforderungen, kann es ein Optimierungsziel sein, in den wahrscheinlicheren Szenarien eine möglichst gute Zukunftssicherheit über geschickte Entwurfsentscheidungen darzustellen. Dies ist durch eine Reduktion empfindlicher Aspekte im vorliegenden UT oder aber durch explizite Einbeziehung solcher Szenarien in die Anforderungen möglich. Letzteres verschiebt problematische Zukunftsszenarien in die Risikobetrachtung und kann mit den oben ausgeführten Maßnahmen gehandhabt werden.

Trade-offs Aus den empfindlichen Aspekten lassen sich mittelbar auch die Trade-offs des Entwurfs ableiten. Der negative Einfluss der Flexibilität einer SOA ist als empfindlicher Aspekt identifiziert. Die positiven Effekte der Flexibilität sind hinreichend diskutiert worden. Grundsätzlich führt eine solche Wechselwirkung im Entwurfsprozess zu einer im Allgemeinen fairen Kompromissentscheidung. In diesem Fall ist zu entscheiden, ob die konkrete Anforderung hinsichtlich hoher Flexibilität wichtiger ist als die Berücksichtigung möglicher zukünftiger Unterstützung von sicherheitskritischen Assistenzen.

Hier bietet sich eine Revision an, die entweder eine klare Entwurfsentscheidung fällt bzw. bestätigt, oder aber eine oder mehrere technische Alternativen diskutiert.

Im Fall des diskutierten UT ist die Anzahl der Blätter gering und somit eine gewisse Überschaubarkeit gegeben. Bei umfangreichen Bäumen kann auch ein quantitativer Ansatz

gewählt werden, der Einzelbewertungen zur besseren Beurteilung verdichtet. Dies ermöglicht nicht nur eine sinnvolle Aggregation zahlreicher Einzelbewertungen, sondern auch eine bessere Vergleichbarkeit über Architekturen hinweg und eine grafische Darstellung.
Als erster Schritt werden dazu die einzelnen Bewertungen über ihren Realisierungsaufwand quantifiziert. Ein möglicher Ansatz ist dabei, schlicht die Realisierungskosten über Gewichtungen aufzuaddieren. In Tabelle A.5 (im Anhang, S. 184) ist eine Quantifizierung der ATAM-Skala anhand einer einfachen Bewertungsfunktion b vorgeschlagen, die für jede Bewertung $n \in \{H,M,L\}$ einen Zahlwert $b(n) \in \{5, 2, 1\}$ liefert. Daraus lassen sich Verdichtungen in Form von Summen ($\sum b(n_i)$) erzielen. Im gezeigten Beispiel (Abb. 4.3) könnte so eine Maßzahl M für jedes Qualitätsmerkmal angegeben werden. So lassen sich auch plausible Abschätzungen angeben, z. B. als Maßzahl $M_{1...3}$ für das Qualitätsmerkmal Interoperabilität mit:

$$M_{1...3} = \sum_{i=1}^{3} b(n_i) = b(n_1) + \ldots + b(n_3) = 3$$

Etwas geschickter ist eine Kombination aus Priorität und Realisierungsaufwand zu diesem Zweck. Diese bewirkt, dass die für das System höher priorisierten Anforderungen deutlicher in der Verdichtung repräsentiert sind und entsprechend weniger wichtige Anforderungen mit kritischer Aufwandsabschätzung schwächer in die Bewertung einfließen. Dazu kann bspw. ein Produkt aus beiden Faktoren gebildet werden $b'(k, n) = b(k) \cdot b(n)$. Die Funktionswerte von $b'(k, n)$ sind im Diagramm in Abb. A.6 (s. Anhang S. 184) dargestellt. Abbildung A.6 zeigt, dass besonders (H,H), (M,H) und (H,M) zu hohen Werten führen. Das unterstützt bei der Identifikation von Risiken. Gleichermaßen lassen sich Bewertungsfunktionen konstruieren, die bspw. bei (L,H), (L,M) oder (M,H) hohen Werten entsprechen. Dies könnte eine Untersuchung auf ein ungünstiges Verhältnis aus Aufwand und Nutzen unterstützen.
Zusätzlich, um die Maßzahlen von dem Einfluss der Kardinalität zu umgehen, bietet sich eine Vereinheitlichung über arithmetische Mittel an:

$$M_{1...n} = \frac{1}{n} \sum_{i=1}^{n} b'(k_i, n_i)$$

Unabhängig von der bevorzugten Bewertungsverdichtung entstehen so für die Äste des Baums auf Ebene der Anwendungsfelder oder Qualitätsmerkmale entsprechende Maßzahlen. Für den hier aufgebauten UT ist eine numerische Darstellung in der Tabelle A.6 (im Anhang, S. 185) gegeben. Die hier diskutierten Verdichtungen $\overline{b}$ und $\overline{b'}$ sind ebenfalls dargestellt.

Es können geläufige Diagramme, wie bspw. Netzdiagramme, eingesetzt werden, um die Ergebnisse visuell aufzubereiten und grafisch in Relation zu setzen. Das Netzdiagramm in Abb. 4.4 zeigt die Architekturbewertung auf Qualitätsmerkmalebene verdichtet. Als Verdichtungsfunktion wurde die oben diskutierte gemittelte Summe $\overline{\mathtt{b}}$ bzw. $\overline{\mathtt{b}'}$ verwendet.

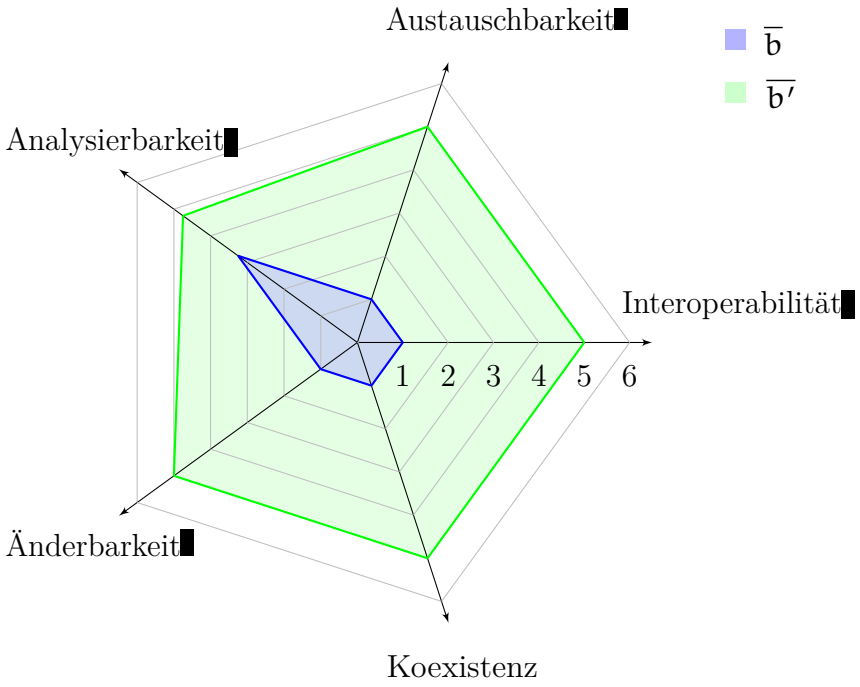

Abbildung 4.4: Darstellung der Architekturbewertung anhand $\overline{\mathtt{b}}$ bzw. $\overline{\mathtt{b}'}$ verdichteter Qualitätsmerkmale

Das Diagramm zeigt einen deutlichen Peak von $\overline{\mathtt{b}}$ bei der Analysierbarkeit. Hier spiegeln sich die oben schon beschriebenen problematischen Bewertungen 10 (L,H), 11 (M,M) und 12 (L,H) im Diagramm wider. Ohne tiefere Kenntnis der Einzelbewertungen weisen hohe Werte von $\overline{\mathtt{b}}$ auf ggf. problematische Aufwandsabschätzungen hin. Hier sollte im Rahmen einer Entwurfsrevision besonderes Augenmerk auf der Kosten-Nutzen-Abschätzung liegen. In der Darstellung ist $\frac{1}{4}\overline{\mathtt{b}'}$ vergleichsweise ausgewogen und zeigt an der Achse Analysierbarkeit einen kleinen Einbruch $(4,75)$ im Vergleich zu den anderen Achsen (5). Die Bewertungsfunktion $\mathtt{b}'(\mathtt{k},\mathtt{n})$ berücksichtigt die Priorisierung der Anforderungen. So bewirken die Bewertungen 10 (L,H), 11 (M,M) und 12 (L,H) mit mittelmäßiger und niedriger Priorität keine signifikante Abweichung.
In diesem Abschnitt wurde gezeigt, wie kritische Aspekte des Architekturentwurfs, wie Risiken, empfindliche Aspekte und Trade-Offs, ermittelt werden können. Es wurde eine numerische Repräsentation des UT entwickelt und verschiedene Bewertungsfunktionen

dargestellt, die solche Untersuchungen unterstützen können. Dies ist ein wesentlicher Beitrag zur Güteabschätzung eines Architekturentwurfs. In der Praxis ist es sinnvoll, zwei oder mehrere Entwürfe zu vergleichen. Auf diese Fragestellung geht der folgende Abschnitt genauer ein.

4.5 Vergleichende Architekturbewertung

Aus der isolierten Bewertung des in Kapitel 3 ausgeführten Architekturkonzepts erschließt sich zwar eine Einschätzung, inwieweit das gewählte Konzept die konkreten Anforderungen erfüllt. Es ist jedoch kein Hinweis gegeben, ob der Architekturentwurf besser oder schlechter abschneidet als bereits existierende Ansätze (s. ITS Test Beds und sim^{TD} in Abschnitt 2.5). Dies ist insbesondere interessant, um den eigenen Ansatz in Relation zu diesen zu setzen und im direkten Vergleich weitere, im Bewertungsverfahren ggf. noch nicht identifizierte Trade-offs zu identifizieren sowie bessere (Teil-)Lösungen zu erkennen. Grundsätzlich kann dazu eine vergleichende Architekturbewertung durchgeführt werden. Ein mögliches Konzept zur vergleichenden Architekturbewertung sowie mögliche Optimierungen werden im Folgenden diskutiert.

Ähnlich wie bei den Architekturbewertungsverfahren, werden in der Literatur einige Vorschläge zur vergleichenden Bewertung von Architekturen gemacht. Zwei bekannte sind bspw. die in Abschnitt 2.13 ausgeführten SACAM und DoSAM. Diese Verfahren berücksichtigen i. d. R. die starke Heterogenität unterschiedlicher Ansätze und entwickeln daher homogenisierende Konzepte wie bspw. das DACF[8] zur belastbaren Bewertung von gewählten Kriterien.

Im Rahmen dieser Arbeit wird jedoch eine andere Zielsetzung verfolgt: Hier stellt sich die Frage, ob der eigene Entwurf, in Relation zu anderen, ein guter ist und ob Entwurfsmuster fremder Entwürfe gewinnend für den eigenen Ansatz übernommen werden können. Unter diesem eingegrenzten Blickwinkel wird in diesem Abschnitt ein eigener Ansatz zur vergleichenden Architekturbewertung ausgeführt, der eben diese Zielsetzung verfolgt und auf unnötige Schritte, wie bspw. Homogenisierungsbestrebungen (in SACAM) und Referenzmodellbildung (in DoSAM), verzichtet.

Als Ausgangsbasis kann angenommen werden, dass eine eigene Anforderungsdefinition

[8] eigenes Framework von DoSAM zur vergleichenden Bewertung von Architekturen in einer Domäne; s. [BRST05, S. 4ff]

besteht, die ausführlich quantifizierte und priorisierte Anforderungen umfasst. Zudem ist ein eigener Ansatz für ein Architekturkonzept skizziert und dessen Designentscheidungen nachvollziehbar dokumentiert.

Auf dieser Grundlage kann nun eine Architekturbewertung durchgeführt werden. Diese reflektiert die Eignung, Risiken, Trade-Offs und Schwachstellen des eigenen Ansatzes. Zudem können anhand der eigenen Anforderungen weitere Architekturkonzepte gleichermaßen untersucht werden. So lassen sich die gleichen Aspekte wie zuvor herausstellen.

Besonders vorteilhaft ist bei dem zuvor skizzierten Bewertungsverfahren die Korrelation zwischen Qualitätsmerkmalen, Anforderungen, Szenarien und der resultierenden Baumstruktur. Dies erlaubt einen stark symmetrischen Aufbau der UT für jede Architekturbewertung. Da die Anforderungen identisch beibehalten werden, bleiben die gewählten Qualitätsmerkmale, relevante Aspekte der Szenarien und (Einzel-)Anforderungen übereinstimmend für jede zusätzliche Untersuchung gleich. So ist ein direkter Vergleich der nach Wichtigkeit sowie Realisierungsaufwand priorisierten Einzelanforderungen gut möglich und erschließt eine strukturierte Bewertung über mehrere Architekturkonzepte hinweg.

Auf dieser Grundlage können nun Schlüsselaspekte der Architekturen verglichen werden. Beispielsweise ist ein Risikovergleich über Anzahl und Art der als riskant[9] identifizierten Einzelanforderungen möglich. Ergebnis einer solchen Untersuchung ist dann ein fundierter Vergleich zwischen der eigenen und anderen Architekturen. Es können qualitative Aussagen getroffen werden wie z. B.: „*Die eigene Architektur ist mit den Komponenten* $\mathtt{a}, \mathtt{b}, \mathtt{c}$ *aufwendiger zu realisieren, jedoch werden Qualitätsmerkmale* $\mathtt{x}, \mathtt{y}, \mathtt{z}$ *besser unterstützt.*“

Sind schwache Teillösungen des eigenen Entwurfs aufgedeckt, die in anderen Architekturen besser dargestellt wurden, kann überlegt werden, deren Konzepte in den eigenen Entwurf aufzunehmen. Diese Adaption kann dann wieder in den UT aufgenommen und dort erneut bewertet werden. Auf diese Weise ist eine aspektweise Verbesserung des eigenen Entwurfs auf Grundlage des Vergleichs möglich.

Aus den Betrachtungen in Abschnitt 2.5 und 4.1 wird deutlich, dass eine vergleichende Architekturbewertung nicht immer ideale Ausgangsbedingungen vorfindet. Insbesondere können die Rahmenbedingungen sehr verschieden sein, sodass konkrete Systementwürfe in bestimmte Architekturansätze gedrängt werden. Zudem ist bspw. die Begründung der Architekturdesignentscheidungen i. d. R. für den eigenen Entwurf noch gut nachvollziehbar. In den drei untersuchten Beispielen von ITS Test Beds, sim$^{\mathrm{TD}}$ und AIM ist die Dokumen-

[9] ebensolche Anforderungen, die als besonders wichtig und gleichermaßen aufwendig klassifiziert werden, vgl. Abschnitt 4.3

tation teils weniger belastbar und somit müssen aus den unbegründeten Definitionen die zugrunde liegenden Motive gemutmaßt werden. Dies schwächt ggf. die Bewertung des Designs im Ergebnis. Die Bewertung fokussiert im Folgenden ITS Test Beds und sim^{TD} als vergleichsweise komplementäre Entwürfe, da sim^{TD} und AIM zumindest im technischen Aufbau von OBU und RSU weitgehend ähnlich sind, wie in Abschnitt 2.5 ausgeführt.

Im Fall der sim^{TD}-Architektur liegt eine Realisierung vor. Aus dieser sollten sich a posteriori klar die jeweiligen Realisierungsaufwände im UT bestimmen lassen. Eine solche Dokumentation lag jedoch für keine der hier untersuchten Testfeldarchitekturen vor. Zudem bedarf es einer ausführlichen Architekturbeschreibung. Beispielsweise stellt ITS Test Beds zwar eine Architekturbeschreibung ([BVJ+11]), diese ist jedoch insbesondere in Teilaspekten nicht tief greifend beschrieben und tangiert kritische Teilaspekte wie bspw. das *Test Pattern Management*[10] nur auf konzeptioneller Ebene. Dadurch ist eine fundierte Bewertung in diesen Entwurfsaspekten deutlich erschwert. Sind im Einzelfall keine klaren Architekturmuster ausgeführt, sind zudem kaum gewinnende Ideen für den eigenen Entwurf ableitbar und eine vergleichende Bewertung scheint zumindest in diesem Aspekt wenig hilfreich. Aus [Ver10] können jedoch einige Erkenntnisse über die prototypische Entwicklung gezogen werden. Diese lassen Rückschlüsse auf mögliche Umsetzungen eines Produktivsystems zu. Auf dieser Grundlage kann der in Abb. 4.3 gezeigte Baum mit korrespondierenden Bewertungen in den Blättern entsprechend der technischen Konzepte von ITS Test Beds und sim^{TD} neu aufgebaut werden. Die Bewertungen werden detailliert in Tab. A.7 und A.8 (im Anhang, S. 186f) gezeigt.

Obwohl Vergleiche zwischen Bäumen prinzipiell keine triviale Aufgabe sind, können in diesem speziellen Fall aufgrund der hohen Symmetrie die Bewertungsbäume dennoch gut verglichen werden. Dazu bieten sich verschiedene Methoden an.

Zum einen können die Bewertungen wechselseitig verglichen werden. Gemäß den zuvor formulierten Überlegungen sind bspw. *(H,H)*-Bewertungen potenziell ungünstiger. Liegen hier Unterschiede zwischen den Einzelbewertungen, kann dies Anlass dazu geben, die damit verbundenen Technologien genauer zu untersuchen. Dazu ist im Einzelfall eine Abhängigkeitsuntersuchung möglich, die aufdeckt, welche Wirkfaktoren die Bewertung beeinflussen. Umgekehrt ist so bestimmbar, welche Architekturkonzepte ggf. verbessert werden können.

Zum anderen kann wie zuvor ein Diagramm eingesetzt werden. Das Netzdiagramm in Abb. 4.5 zeigt die Architekturbewertungen der drei Architekturen auf Qualitätsmerkmal-

[10] vgl. [BVJ+11, S. 79]

ebene verdichtet. Als Verdichtungsfunktion wurde eine gemittelte Summe aus $\mathtt{b}'(\mathtt{k},\mathtt{n})$ verwendet. Die weitere Deutung und Interpretation dieser Graphen wird im nächsten

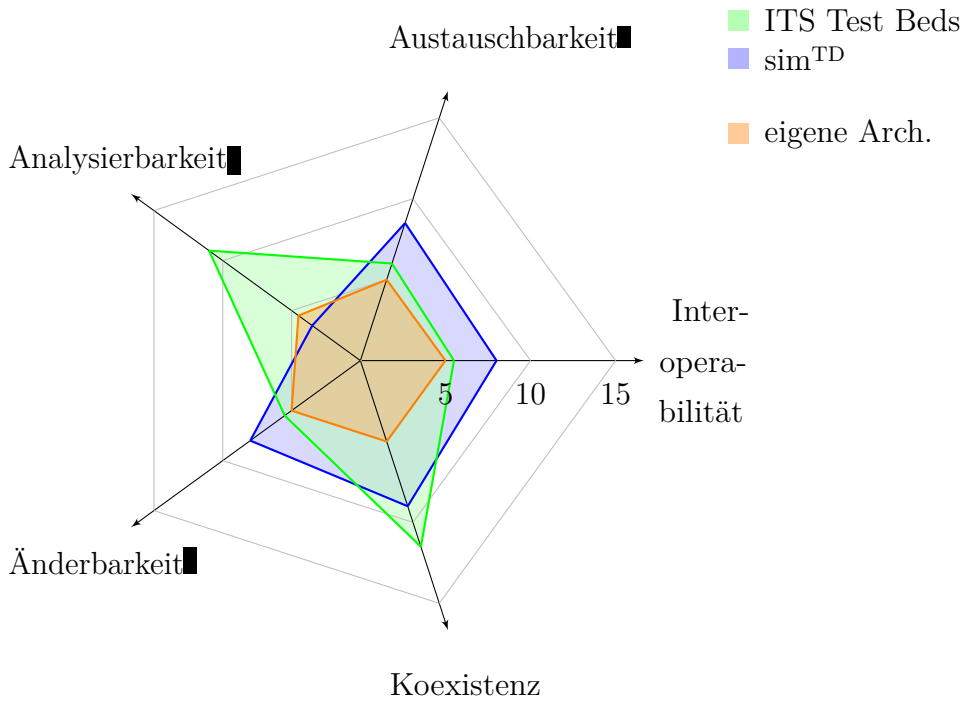

Abbildung 4.5: Gegenüberstellung von ITS Test Beds, sim^{TD} und eigener Architekturbewertung über Qualitätsmerkmale verdichtet

Abschnitt ausgeführt.

4.6 Interpretation der Metriken

Das Diagramm in Abb. 4.5 zeigt die verdichteten Bewertungen der drei untersuchten Architekturen in einem Netzdiagramm. Die eigene Architektur ist dabei ganz analog zu dem vorangehenden Diagramm (Abb. 4.4) und mit der in Abschnitt 4.4 diskutierten Fläche dargestellt. Die beiden verbleibenden Kurven repräsentieren die kumulierten Bewertungen von sim^{TD} und ITS Test Beds. Aus der Lage der Kurven zueinander, der gegenseitigen Überdeckung sowie der (relativen) Dimension können nun folgende Schlüsse gezogen werden:
Die sim^{TD}-Kurve ist leicht aus dem Zentrum nach rechts verschoben. Dadurch ist ableitbar, dass im Bereich Interoperabilität, Austauschbarkeit, Änderbarkeit und Koexistenz höhere Aufwände geleistet werden müssen, um die Anforderungen zu erfüllen. Im Gegensatz dazu

ist das Qualitätsmerkmal Analysierbarkeit nur schwach ausgeprägt. Hier wird offenbar mit geringem Realisierungaufwand gerechnet. Insgesamt ist im Vergleich zum eigenen Entwurf die überdeckte Fläche sichtlich größer. Unter Berücksichtigung der Bewertungsfunktion, bei der höhere Werte i. d. R. schlechter sind, schneidet der Konzeptentwurf von sim$^{\text{TD}}$ hinsichtlich der Konformität auf die dargestellten Anforderungen insgesamt schlechter ab.

Eine solche Bewertung passt auch zu dem in Abschnitt 2.5 beschriebenen ersten Eindruck: Die sim$^{\text{TD}}$-Architektur ist durchaus auf eine abgesteckte Menge an Anwendungen ausgerichtet. Somit sind wenig Mechanismen vorgesehen, die auf besonders gute Austauschbarkeit oder Änderbarkeit abzielen. Positiv ist die gute Analysierbarkeit. Dadurch, dass eben keine besonders flexible Diensteverschaltung gewählt wurde, ist die Analysierbarkeit in Teilen besser gegeben.

Die ITS-Test-Beds-Kurve zeigt zwei wesentliche Ausreißer bei Analysierbarkeit und Koexistenz. Um diese Aspekte gemäß den Anforderungen umzusetzen, wird mit höherem Aufwand gerechnet. Ansonsten liegt die Kurve pareto-optimal gegenüber der Bewertung des eigenen Ansatzes und die überdeckte Fläche ist ebenfalls größer. Somit schneidet auch der Konzeptentwurf von ITS Test Beds hinsichtlich der Konformität auf die dargestellten Anforderungen insgesamt schlechter ab.

In der Bewertung von ITS Test Beds schlagen sich insbesondere Analysierbarkeit und Koexistenz negativ nieder. Diese werden bspw. in [BVJ+11] kaum diskutiert und sind daher in den Beschreibungen der Komponenten praktisch nicht wiederzufinden. So ist mehr Aufwand nötig, diese Aspekte anforderungskonform umzusetzen. Die verbleibenden Qualitätsmerkmale Interoperabilität, Austauschbarkeit und Änderbarkeit sind nur marginal schwächer als der eigene Entwurf. Hier fehlen ITS Test Beds nur wenige Bausteine in der Architektur, um entsprechend nachzubessern. Ein Beispiel dafür ist Diensteorchestrierung mittels TASK. In diesem Werkzeug findet sich bspw. kein Mechanismus, um Komponenten zu versionieren bzw. versionierte Komponenten zu handhaben. Dies führt zu schwach skeptischeren Aufwandsabschätzungen.

Diese vergleichende Betrachtung zeigt, wie im vorliegenden Fall der eigene Entwurf mit bestehenden Entwürfen von sim$^{\text{TD}}$ und ITS Test Beds verglichen werden kann. Sofern bei den verglichenen Entwürfen strukturgleiche Bäume (UT) entwickelt werden, ist eine gute Vergleichbarkeit gegeben. Die oben ausgeführten Schlussfolgerungen beschränken sich daher nicht nur auf die konkreten Beispiele, sondern können prinzipiell für alle homogen geformten Bewertungsbäume analog durchgeführt werden. Sind so kritische

Aspekte identifiziert, kann hier auch entschieden werden, ob sich eine Umgestaltung oder auch eine Entleihung von entsprechenden Konzepten aus anderen Ansätzen gewinnend einsetzen lässt. So lässt sich ggf. schrittweise eine Verbesserung der Architektur erzielen; dies bettet sich gut in einen iterativen Entwicklungsprozess ein (s. Abschnitt 2.14), wie im Folgenden genauer ausgeführt wird.

4.7 Methodische Einbettung in den Architekturprozess

Kapitel 3 betrachtet den Testfeldarchitekturprozess, also den maßgeblichen Prozess, um eine Testfeldarchitektur zu entwickeln. In diesem Kapitel wurde der Prozess der Architekturbewertung diskutiert, welcher eine Abschätzung von Risiken, empfindlichen Aspekten und Trade-Offs ermöglicht und anhand einer vergleichenden Architekturbewertung Stärken und Schwächen des Entwurfs im Vergleich zu alternativen Ansätzen ermittelt. Es bleibt offen, wie die beiden wesentlichen Aspekte der Architekturentwicklung und der -bewertung zusammenspielen und in den überspannenden Entwicklungsprozess einer Testfeldarchitektur eingebunden sind. Dieser Abschnitt stellt daher den Gesamtprozess aus Architekturentwicklung und -bewertung dar und erörtert dessen Einbettung in den Architekturprozess.

Für die Integration der Ergebnisse aus der Architekturbewertung in einen iterativen Entwurfsprozess, wie in Abschnitt 2.14 beschrieben, bieten sich neben einem vollständigen Neuentwurf auch Verbesserungen im dekompositorischen und synthetischen Teil des Entwurfsprozesses an. Dies kann beispielsweise durch die Rekomposition der kleinsten Diensteinheiten und die Neuzuordnung auf physikalische Komponenten geschehen, was zu grundlegend abweichenden Architekturentscheidungen führen kann. Die Auswirkungen davon fließen letztendlich in Phase I und II in das Bewertungsverfahren ein. Daher ist eine Überarbeitung der beiden Syntheseschritte zunächst empfehlenswert. In diesem Zusammenhang können auch die Ergebnisse einer vergleichenden Architekturbewertung genutzt werden, indem besser bewertete Ansätze aus dieser in den eigenen Entwurf integriert werden.

Um dies genauer zu beschreiben, zeigt Abb. 4.6 den Gesamtprozess aus Architekturentwicklung und -bewertung als Flussdiagramm.

Gezeigt sind in Abb. 4.6 die Prozessschritte *Anforderungsanalyse* und *Entwicklung Testversion*, die sich auf den Ablauf im iterativen Entwicklungszyklus beziehen und über die gleichlautenden Schritte in diesem verankern (dargestellt in Abb. 2.16 auf Seite 61). Die

Architekturentwicklung und die *-bewertung* sind als Container dargestellt. Sie umfassen die wesentlichen Prozesse der *Dekomposition*, der *Synthese* (wie in Kapitel 3 beschrieben) und die drei Phasen des angepassten Architekturbewertungsverfahrens (siehe Abschnitt 4.2).

Das Flussdiagramm zeigt auch die zentralen Artefakte, die in den jeweiligen Prozessschritten gewonnen werden. Im Fall der *Dekomposition* sind das die drei abgeleiteten Sichten *Scenarios*, *Logical View* sowie *Process View*, während aus der *Synthese* die Sichten *Development View* und *Physical View* hervorgehen. Ebenfalls ist gezeigt, dass die Synthese auf den Ergebnissen der Dekomposition aufbaut, wie in Abschnitt 3.2 genauer erläutert.

Im Flussdiagramm folgen dann unmittelbar die drei Phasen der Architekturbewertung. Diese sind direkt mit dem Syntheseprozess verknüpft. Da die Architekturentwicklung zwar eine gute Übereinstimmung mit den funktionalen Anforderungen erreicht, aber noch keine Bewertung hinsichtlich nicht-funktionaler Anforderungen erfolgt ist, ist eine Bewertung notwendig und rechtfertigt diese direkte Verbindung.

Aus *Phase II* resultiert der Utility Tree als maßgebliche Grundlage für die Bewertung. Die Bilanzierung dieses Baums in *Phase III* wird in der *Auswertung* ebenfalls als Artefakt im Flussdiagramm dargestellt.

Bei der Interpretation des Utility Trees (siehe Überlegungen dazu in Abschnitt 4.4) kann die Bewertung individueller Merkmale zu einer überarbeiteten Entwurfsentscheidung führen. Im einfachsten Fall könnte dies aufgrund eines ungünstigen Verhältnisses von Realisierungskosten zu -nutzen geschehen und zu einer Anpassung der Systemanforderungen führen.

Im Flussdiagramm ist dieser Fall als Entscheidungspunkt *Entwurfsgüte ausreichend* dargestellt, der den Ablauf im positiven Fall zum Schritt *Entwicklung Testversion* führt. Liegt eine Überarbeitungsentscheidung vor, führt der Fluss zurück zur *Anforderungsanalyse*, in der die entsprechenden Änderungen umgesetzt werden. Dies führt zwangsläufig zu einer angepassten Architektur und folglich zu einer neuen Bewertung. Dies könnte beispielsweise eine Spezialisierung des Entwurfs auf weniger Szenarien bedeuten. Infolgedessen könnte das Szenario *Referenzplattform* aus den Anforderungen gestrichen werden. Dies hätte zur Folge, dass der ATAM-Baum in Abb. 4.3 die kritischen Blätter hinsichtlich Analysevermögen verlieren würde und zu einer verbesserten Bewertung führen würde. Diese Kompromittierung der Anforderungen kann durch die Phase der Anforderungsdefinition im Rahmen des iterativen Entwurfs (siehe Abb. 2.16) abgesichert werden, um sicherzustellen, dass die funktionalen Anforderungen weiterhin der erwarteten Systemleistung entsprechen. Bei

einer Anpassung grundlegender Anforderungen kann ein kompletter Neuentwurf notwendig sein, der mit einem höheren Aufwand verbunden ist als eine Änderung innerhalb des Architekturprozesses.

Über diesen Zyklus wird eine iterative Verbesserung des Architekturentwurfs und seiner Artefakte erreicht, was eines der Hauptziele dieser Arbeit ist. Daher bietet das Flussdiagramm in Abb. 4.6 einen guten Überblick über den Ablauf der in dieser Arbeit entwickelten Methodik.

Der im Fluss vorgesehene Rücksprung ist aus Entwicklungssicht in der Praxis mit Kosten verbunden, da das entsprechende Team geänderte Anforderungen prüfen und alle wesentlichen Entwurfsschritte neu durchlaufen muss. Es ist evident, dass diese Revision der Anforderungen günstiger ist, als während oder sogar nach der Implementierung des Testsystems. Dennoch ist eine weitere Optimierung wünschenswert.

Abhängig vom Analyseergebnis kann ein Rücksprung auf den Prozess der *Synthese* versucht werden. Dies scheint vielversprechend, wenn die Analyse des Utility Tree auf Probleme bei der Entwicklungssicht bzw. der physischen Verteilung der Komponenten im System hinweist. Ein Beispiel dafür könnte das zuvor diskutierte Verteilungsverhältnis von Funktionalität zwischen potenziell schwacher Hardware in mobilen Endgeräten und leistungsstarker straßenseitiger Hardware sein. Ist hier eine Anpassung möglich, so sind die folgenden Bewertungsschritte (*Phase I*...*Phase III*) vergleichsweise wenig aufwändig zu durchlaufen.

Ist eine solche Umgestaltung noch nicht ausreichend, kann versucht werden, die Entwicklung der Prozesse (UEP und unterliegende Prozesse) im Einklang mit den Bewertungsergebnissen zu verbessern. Die Entwicklung der Szenarien zu einem Rollenbild und deren Festlegung auf die Aufgaben des Zielsystems sind dafür wesentliche Ansatzpunkte. Durch eine Anpassung auf dieser Ebene ergibt sich eine Neudefinition der abgeleiteten Sichten. Im Flussdiagramm entspricht dies einem Rücksprung auf den Prozess *Dekomposition*. Zwar bewirkt dies einen entsprechend höheren Aufwand als das Re-Engineering innerhalb des Syntheseprozesses, spart aber dennoch eine umfassende Reflexion der Anforderungen ein.

In der Praxis arbeiten oft unterschiedliche Teams an den Anforderungen und der Architekturentwicklung. Daher ist der Arbeitsablauf auch aus dieser Perspektive zu betrachten. Entsprechend sind die oben diskutierten Rücksprünge auf *Dekomposition* und *Synthese* eher als technische Optimierung zu verstehen, die im harten Umriss einer bestehenden Anforderungsdefinition mögliche Umsetzungen abwägen. Ein Rücksprung auf die *Anforde-*

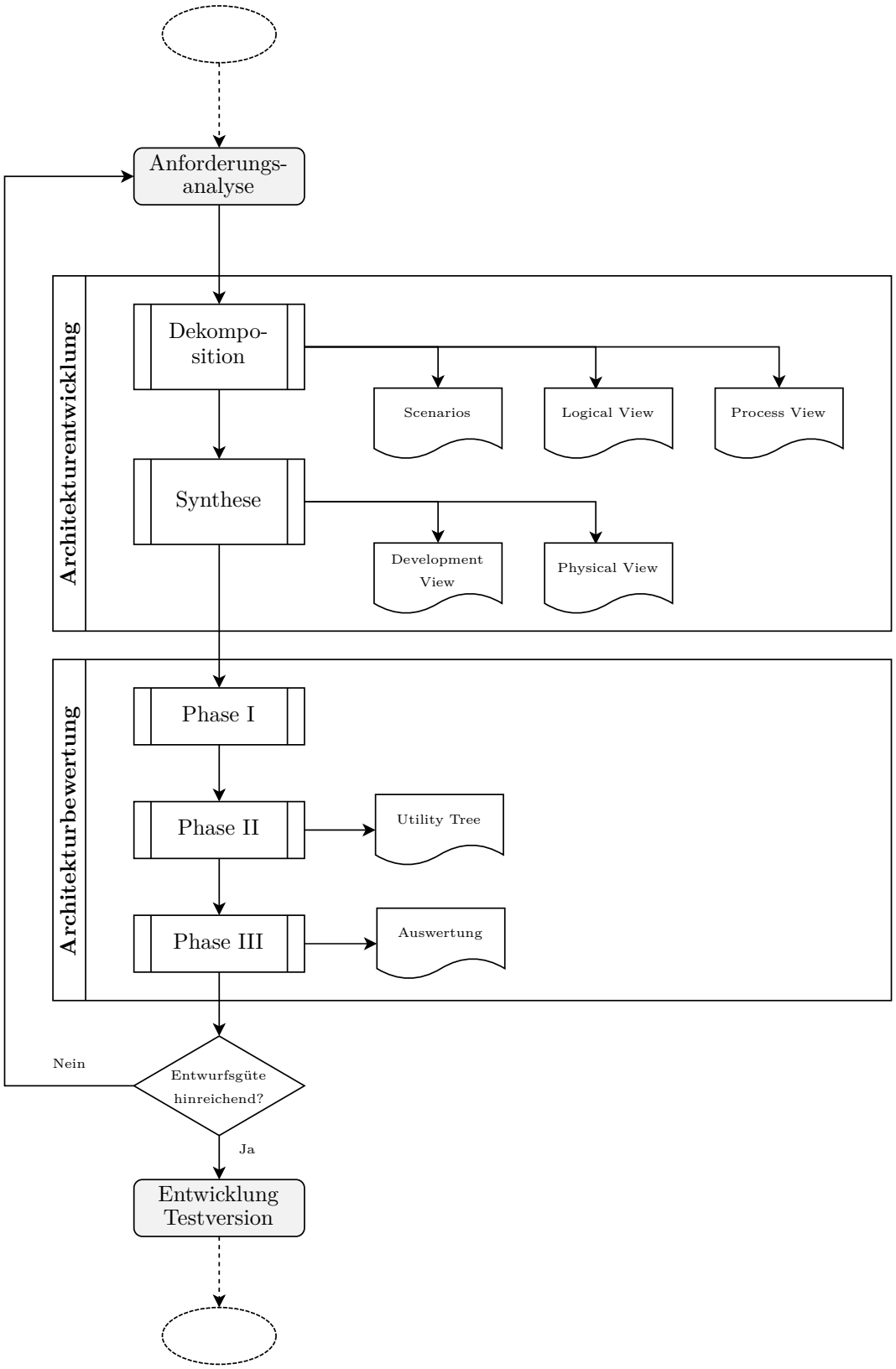

Abbildung 4.6: Gesamtdarstellung Testfeldarchitekturprozess; Flussdiagramm

rungsanalyse bewirkt in verteilten Teams eine Einbindung derer, die den Funktionsumfang des Testfelds fokussieren, jedoch nicht zwangsläufig die technischen Implikationen der Umsetzung betrachten. Aus dieser gemeinsamen Abwägung können dann grundlegende Entscheidungen zur Anpassung der Anforderungsdefinition getroffen werden, die wiederum andere Entwurfsentscheidungen maßgeblich beeinflussen können. Daher bleibt der Rücksprung auf die *Anforderungsanalyse* ein essentielles Element dieser Methode, um alle Änderungen zu ermöglichen, die über die technische Adaption mittels Rücksprüngen auf *Dekomposition* und *Synthese* hinausgehen.
Aus dieser Überlegung heraus stellt der in Abb. 4.6 dargestellte Fluss einen generischen Verbesserungsprozess bei der Entwicklung einer Testfeldarchitektur dar.

4.8 Zusammenfassung

In diesem Kapitel wurde die Bewertung von Testfeldarchitekturen dargestellt. Die speziellen Rahmenbedingungen bei der Bewertung solcher Architekturen wurden in Abschnitt 4.1 präzisiert, die Zielsetzung und die eigene Methode zur Architekturbewertung wurden in Abschnitt 4.2 ausgeführt. Abschnitt 4.3 führte eine entsprechende Bewertung am eigenen Testfeldentwurf (aus Kapitel 3) durch. Die Erkenntnisse aus dieser Bewertung wurden in Abschnitt 4.4 dargestellt und diskutiert.
Um einen Architekturentwurf mit anderen (bestehenden) Entwürfen zu bewerten, wurde in Abschnitt 4.5 ein entsprechendes Verfahren gezeigt. So wurde der eigene mit den zuvor diskutierten Testfeldstrukturen aus sim^{TD} und ITS Test Beds verglichen und in Abschnitt 4.6 diskutiert.
Als Ergebnis liefert dieses Kapitel ein gut angepasstes Verfahren, wie in Abschnitt 4.7 als Flussdiagramm dargestellt, um typische nicht-funktionale Aspekte im Testfeldentwurf sowohl in isolierter als auch in vergleichender Betrachtung mit bestehenden Ansätzen zu bewerten. Dies liefert signifikante Hinweise auf die Güte des eigenen Architekturentwurfs und hilft, Schwachstellen und Risiken in den frühsten Artefakten einer Implementierung aufzudecken. Ergänzend zum methodischen Verfahren zur Entwicklung einer Testfeldarchitektur, welches (wesentlich) funktionale Aspekte berücksichtigt, trägt die Architekturbewertung zum zukunftssicheren und wirtschaftlichen Entwurf bei.

5 Umsetzung

In den vorangehenden Kapiteln wird ein methodischer Ansatz zur Gestaltung und Bewertung von Architekturen diskutiert. Die dort dargestellten Verfahren illustrieren zwar Teilaspekte des Architekturentwurfs bzw. dessen Bewertung, jedoch wird kein vollständiges Architekturergebnis aufgebaut und bewertet. So lässt sich aus den isolierten Überlegungen noch nicht ableiten, ob die Verfahren in praktischer Umsetzung tatsächlich geeignet sind, eine adäquate Testfeldarchitektur zu gewinnen. Daher fokussiert dieses Kapitel darauf, die Umsetzbarkeit, die Qualität des resultierenden Entwurfs, im Sinn einer Verifikation, sowie die Relevanz der Methoden in praktischer Umsetzung zu validieren.
Zur Verifikation der Methoden bieten sich einige Möglichkeiten an. So könnte bspw. der Aufbau einer Referenzarchitektur auf Basis von typischen Anforderungen und mit den Verfahren aus Kapitel 3 gezeigt werden. Jedoch bleibt dies ein isolierter Entwurf auf hypothetischen Grundlagen, der ohne praktische Umsetzung und Reflexion in der Domäne ein schwer zu bewertendes Ergebnis liefert.
Aufgrund des hypothetischen Charakters solcher Evaluationen wird in diesem Abschnitt ein konkreterer Bewertungsansatz verfolgt. Im Projekt Instant Mobility[1] konnten die in Kapitel 3 diskutierten Verfahren praktisch zum Einsatz gebracht werden. Daher wird in diesem Abschnitt zunächst der Kontext der praktischen Umsetzung erläutert. Darauf aufbauend wird die Umsetzung der Methoden selbst dargestellt und kritisch bewertet.

5.1 Einleitung

Um im praktischen Einsatz Umsetzbarkeit, Qualität des resultierenden Entwurfs sowie Relevanz der in Kapitel 3 beschriebenen Verfahren sinnvoll zu diskutieren, muss eine möglichst gute Deckung zwischen den jeweiligen korrespondierenden Zielsetzungen von Entwurfsverfahren gemäß Kapitel 3 und Architekturentwicklung im Projekt gegeben sein. So ist sichergestellt, dass die Verfahren in Kontext des Testfeldentwurfs Bestand haben.

[1] Laufzeit von 2011 bis 2013; s. [Ins12]

Dazu werden in Abschnitt 5.2 zunächst die Randbedingungen des Entwurfsprozesses in Instant Mobility ausgeleuchtet. Darauffolgend werden einige Kernanforderungen im Architekturprozess des Projekts dargestellt, um eine fachliche Nähe zum Testfeldkontext darzustellen. Im Abschnitt 5.3 wird die konkrete Realisierung am praktischen Beispiel im Projekt dargestellt und qualitativ bewertet.

5.2 Entwurfsprozess und inhaltliche Zielsetzung in Instant Mobility

Das Instant Mobility Projekt war ein durch die EU gefördertes, zweijähriges Forschungsprojekt zur Anforderungserhebung und Spezifikation eines umfassenden Feldversuchs. In dem zu spezifizierenden Versuchsaufbau sollten Future Internet Technologies (FIT), also moderne Konzepte der Informations- und Kommunikationstechnik, wie bspw. Cloud Computing, Big Data handling und Internet of Things, bestehende und innovative Anwendungsfälle der Transport- und Mobilitätsdomäne unterstützen. So war Instant Mobility Teil der Spezifikationsphase des EU-Förderungsprogramms Future Internet Public-Private Partnership (FI PPP), dessen Zielsetzung über die Spezifikation hinaus die tatsächliche Realisierung dieser FIT in breit angelegten Feldversuchen mit mehr als 100.000 Probanden anstrebte. Dazu wurden Anwendungsfälle gewählt, bei denen möglichst die Vorteile der FIT gezeigt werden konnten. Unter anderem wurden eine multi-modale Reiseassistenz, ein virtualisiertes Verkehrsmanagement und Assistenzen aus dem Logistikbereich spezifiziert. Diese ITS-Anwendungen wurden auch zur Absicherung des Entwurfs in Teilaspekten realisiert und die Umsetzung demonstriert. Deren geplante Evaluation in Feldversuchen entspricht einem FOT.

Das Instant Mobility Konsortium bestand aus 21 Partnern, darunter sind neun Industriepartner vertreten [Ins12]. Von dem Projektbudget von 7,9 Mio.€ wurden ca. 36 % für die Spezifikation und ca. 30 % für die prototypische Umsetzung verwendet. Somit stellen die Arbeiten in diesen Schwerpunkten wesentliche Ergebnisse des Projekts dar. Durch die Einbettung des Projektes in den Kontext von FI PPP wurden darüber hinaus wesentliche Projektressourcen zur Adaption der außerhalb des Projektes bereitgestellten FIT aufgewendet.

Durch die Einbettung in das FI PPP unterlag der Spezifikationsprozess einigen besonderen Auflagen, die ein typischer Entwurfsansatz nicht berücksichtigen muss (s. [BSvV13, S. 9]). Wesentlich prägend war die enge Anbindung an weitere Projekte wie bspw. FI-WARE. Diese

lieferten in einem eigenen, abgekoppelten Spezifikationsprozess wesentliche FIT basierte Komponenten (Module), die durch Instant Mobility in der Spezifikation aufgegriffen und integriert werden sollten.

Die Projektstruktur folgt den wesentlichen Arbeitsinhalten mit korrespondierenden Arbeitspaketen, u. a. Anforderungsanalyse, Spezifikation und Implementierung. Die für diese Arbeite wichtigen Architekturüberlegungen sind im Spezifikationsprozess von Instant Mobility dargestellt. Die Arbeiten im Projekt folgen einem iterativ agilen Ansatz.[2]

Dadurch werden die konventionell sequenziellen Softwareentwicklungsschritte über Anforderungsdefinition, Entwurf und Implementierung mehrfach durchlaufen, um so bspw. geänderten Anforderungen durch neue Erkenntnisse gerecht zu werden.

Daraus ist der in Abb. 5.1 illustrierte dreistufige Ansatz abgeleitet worden, der in zwei Iterationen zur gewünschten Spezifikation führt. Die erste Stufe greift die Ergebnisse der Anforderungsanalyse auf und leitet eine initiale Transition in eine funktionale Spezifikation ein. Die zweite Stufe konsolidiert die Ergebnisse externer Projekte und liefert so funktionale Bausteine, die im Entwurf aufgegriffen werden können. In der letzten Stufe wurde der bisherige Entwurf an den Anforderungen validiert. Dadurch ist auf hoher Entwurfsebene ein Prozesskorsett vorgegeben, in den sich der eigentliche Architekturprozess auf Arbeitsebene einfügen muss. Dies entspricht dem in Abschnitt 2.14 dargestellten Vorgehensmodell nach [BCB97].

Diese drei Stufen umfassen die funktionale Zerlegung und Synthese (vgl. [BST11, S. 10]) analog zu dem in Abschnitt 2.12 eingeführten und im eigenen Konzept in Kapitel 3 aufgegriffenen Zerlegungs- und Syntheseprozess. Dabei orientieren sich die Architekturentwicklungsschritte an den *4+1 Views* aus Abschnitt 2.10, s. [BKZS12, S. 4ff]. Somit folgt der Architekturprozess der grundsätzlichen Idee, anhand der *4+1 Views* die Entwicklungsschritte zu gestalten. Dies entspricht somit gut dem in Abschnitt 3.2 ausgeführten methodischen Ansatz.

Zur Qualitätssicherung des Architekturentwurfs wurden sowohl innerhalb als auch außerhalb des Projektes Gremien und Rollen[3] eingesetzt, deren Aufgabe es war, die fachliche Güte des Entwurfs zu gewährleisten. Zudem war durch die Einbettung in FI PPP ein konzeptioneller Abgleich mit ähnlichen Projekten aus anderen Domänen und der zentralen

[2] *Requirement driven; Iteratively, incrementally and collaboratively; Multi-view approach; Transparency*, s. [BSvV13, S. 19]

[3] u. a. *Chief Technology Officer* (CTO) und *Concertation Board*, vgl. [Nag12]

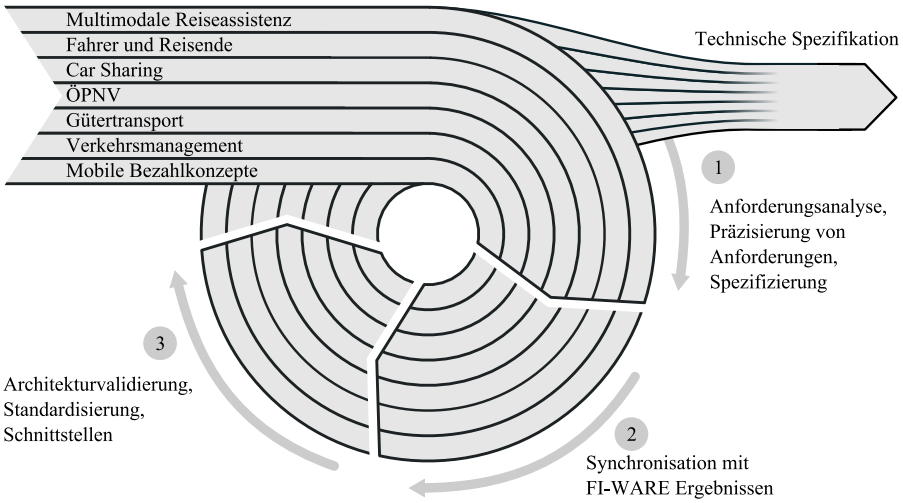

Abbildung 5.1: Architekturprozess in Instant Mobility aus [BST11]

Entwicklung von domänenübergreifenden Modulen notwendig. Dazu wurde ein projektübergreifendes *Architectureboard* vorgesehen, in dem der Architekturentwurf von Instant Mobility reflektiert und bewertet wurde [Nag12, S. 7f].

Zu Beginn des Projekts wurden fünf fachliche Schwerpunkte für die tiefere Anforderungsanalyse herausgearbeitet: „*Multimodal travel made easy, the sustainable car, collective transport, trucks and the city, online traffic and infrastructure management.*“[KLV+11, S. 14] Diese wurden in konkretere Anwendungsfälle verfeinert, die spezielle Teilaspekte des jeweiligen Schwerpunktes abdecken. Die Darstellung dieser in [KLV+11, S. 14ff] lässt den Schluss zu, dass nahezu alle Anwendungsfälle auf Kommunikationsverfahren wie UMTS, LTE oder 5G aufsetzen, um Reisende oder mobile Endgeräte mit einem Rechenzentrum zu verbinden. Dies macht sie zu typischen ITS-Anwendungsfällen, wie in Kapitel 2 ausgeführt. Dabei wird keine unmittelbare Kommunikation über Funknetzwerke (wie bspw. IEEE 802.11p) von Einheiten im Feld vorgesehen. Dadurch sind die in Instant Mobility gewählten Anwendungen keine gute Repräsentation von Anwendungsfällen mit direkter Kommunikation von Infrastrukturelementen wie bspw. zwei OBU untereinander. Aus diesem Satz an Anwendungsfällen sind konkretere Anforderungen an ausgewählte Anwendungsfälle in [KLV+11] dokumentiert. Insbesondere die in [LAPB12, S. 9f] dargestellten nicht-funktionalen Anforderungen, wie *Scalability*, *Interoperability* und *Expandability*, korrespondieren gut mit den in Kapitel 4 untersuchten Qualitätsmerkmalen.

Um eine Spezifikation für einen umfassenden Feldtest zu erzielen, muss eine technische

Basis für die darauf aufsetzenden Anwendungen entworfen werden. Durch die besondere Einbettung des Projekts in den FI PPP-Kontext unterliegt die technische Umsetzung dieser Basis ähnlichen Anforderungen wie im Testfeld, wie bspw.:

- Interoperabilität und Austauschbarkeit sind notwendig, um externe Softwareprodukte von FI-WARE flexibel in den Feldtest einzubinden.
- Aus der Durchführung des Feldversuchs sollen u. a. auch die positiven Effekte von FIT in der Domäne nachgewiesen werden. Somit muss die zugrunde liegende Infrastruktur über die reine Funktion hinausgehend analytische Daten handhaben.
- Über die Laufzeit des FI PPP werden Komponenten des Feldversuchs sukzessiv in Richtung Marktreife weiterentwickelt. Dadurch werden Aspekte wie Koexistenz und Änderbarkeit gefordert.

Im Kontrast zur Nachhaltigkeit von Testfeldern ist im Rahmen des FI PPP und auch außerhalb dessen keine Wiederverwendung der Installation mit ähnlichen oder andersartigen Anwendungen direkt geplant. Daher ist das Zielsystem keine reine Testfeldarchitektur. Dennoch sind zumindest über die Einbettung im FI PPP eine projektübergreifende Verwendung und ein entsprechender Ausbau zu berücksichtigen.
Obwohl der Instant-Mobility-Entwurf somit kein vollwertiges Testfeld umfasst, sind die Zielsetzungen im Kontext von FI PPP und Einsatzzweck durchaus mit den Verwendungszwecken aus Abschnitt 2.3 vergleichbar. Insbesondere die Rahmenbedingungen auf Prozessebene und der Einsatz in komplexen, heterogenen Arbeitsteams verlassen kontrollierte Laborbedingungen und stellen so einen guten Untersuchungsgegenstand dar.

5.3 Praktische Umsetzung und Bewertung

Die im Kontext dieser Arbeit relevanten Arbeitsergebnisse sind in den Projektberichten [BST11, S. 10ff] und [BSvV13, S. 19ff] ausführlich dokumentiert. Dabei beschreibt [BST11] den methodischen Ansatz sowie den Stand der Architekturüberlegungen relativ früh in der Projektlaufzeit. Der abschließende Bericht [BSvV13] nimmt dies als Ausgangspunkt und bietet eine umfassendere und aktualisierte Darstellung zum Projektende. Durch den Vergleich der beiden Dokumente können zudem wesentliche Abweichungen vom Projektplan identifiziert und so Rückschlüsse auf mögliche Hindernisse während der Umsetzung gezogen werden.

Der ursprüngliche iterative, inkrementelle und kollaborative Ansatz wurde über die Projektlaufzeit beibehalten [BSvV13, S. 20]. Dabei wurde die im vorigen Abschnitt ausgeführte Zerlegung in kleinste atomare Einheiten anhand der *4+1-Sichten* umgesetzt und der resultierende Entwurf iterativ verfeinert [BSvV13, S. 22]. Über diesen Prozess wurde eine umfassende Architektur für das Projekt gestaltet und verfeinert, welche die gewünschten Eigenschaften bietet. Die Darstellung der Architektur in *4+1-Sichten* nutzt vergleichbare Diagramme zur Beschreibung der Ergebnisse. Beispielsweise verwendet die Modellierung des *end-to-end itinery planning*[4] UML-Diagramme, die inhaltlich den in Abschnitt 3.5 diskutierten Diagrammen zur Modellierung der UEP entsprechen (vgl. Abb. 3.2). Analog verwendet die Prozesssicht BPMN zur Orchestrierung der kleinsten Funktionseinheiten. Der Prozess *Pre-trip journey planning* ist exemplarisch in Abb. A.8 (im Anhang, S. 189) dargestellt. Dies entspricht der in Abschnitt 3.6 ausgeführten Prozessbeschreibung (vgl. Abb. 3.4).

Bis zum Ende des Projektes wurden sechs Teilsysteme der Architektur detailliert spezifiziert [BSvV13, S. 58]. Auf dieser Grundlage entstand ein Prototyp, der Teile der Spezifikation nutzt, um daraus drei wesentliche Funktionen[5] zu demonstrieren.

Die ausführliche Architekturbeschreibung in [BSvV13] steht im Wesentlichen im Einklang mit den ursprünglichen, in [BST11] ausgeführten Überlegungen. Es gibt nur marginale Abweichungen, die im Kontext des Architekturentwurfs relevant sind. Beispielsweise plant [BST11] ursprünglich eine SOA. In der Spezifikation wird dieses Paradigma jedoch zugunsten einer Resource Oriented Architecture (ROA) aufgegeben, um so die extern entwickelten Komponenten besser einzubinden [BSvV13, S. 32ff]. Die Dokumentation der Prototypenentwicklung berichtet von drei wesentlichen Abweichungen zur ursprünglichen Planung [BAN+12, S. 10f]. Deren Ursache liegt wesentlich an der komplexen Beziehung zwischen dem Projekt und externen Entwicklungen im FI PPP. Beispielsweise liegen zwar Spezifikationen für spezielle FIT-Funktionen vor, jedoch liegen diese zur Realisierungsphase der Prototypen nicht vollständig in Software vor. So war die Implementierung teils gezwungen, fehlende Module improvisierend zu ersetzen.

Aus den hier ausgeführten Beobachtungen des Architekturprozesses im Projekt Instant Mobility kann geschlussfolgert werden, dass der dabei verfolgte methodische Ansatz grundsätzlich erfolgreich war, um eine umfassende Architektur für das Projekt darzustellen

[4] s. Abb. A.7, im Anhang, S. 188

[5] Reiseassistenz, innerstädtische Logistik und Verkehrsmanagement, vgl. [BAN+12, S. 11]

bzw. fachlich kritische Aspekte prototypisch zu realisieren. Die ausgeführten Abweichungen waren für die Umsetzung der Methode unkritisch und externen Faktoren geschuldet. Der resultierende Entwurf wurde in internen und externen Gremien reflektiert.
Analog zu dem in Kapitel 3 dargestelltem Testfeldarchitekturprozess folgte der Instant Mobility Ansatz der grundsätzlichen Idee, anhand der *4+1 Views* die Architekturentwicklungsschritte zu gestalten. Dabei wurden im Projekt ebenfalls Architektursichten in vergleichbaren Diagrammen zur Ergebnisbeschreibung dargestellt. Dadurch ist eine gewisse Vergleichbarkeit zwischen der in Instant Mobility beobachteten Methode und den in Kapitel 3 dargestellten Verfahren gegeben.
Inhaltlich ist ein direkter Vergleich zwischen Instant Mobility und dem Entwurfsverfahren für Testfelder nicht direkt gegeben, da bspw. Instant Mobility kein Testsystem integrieren oder emulieren musste. Dennoch war Instant Mobility angehalten, die Entwicklungen bspw. von FI-WARE aufzugreifen und in den eigenen Entwurf zu integrieren. Aus der damit verbundenen komplexen Verzahnung der Projekte mit projektübergreifenden Zielsetzungen im Kontext von FI-PPP und dem Einsatzzweck in der Transportdomäne sind diese dennoch vergleichbar.

5.4 Zusammenfassung

Die in Kapitel 3 vorgeschlagene Entwurfsmethode für Testfelder wurde zuvor nur hypothetisch diskutiert. Daher zeigt dieses Kapitel eine praktische Umsetzung in einem inhaltlich vergleichbaren Kontext, um so Rückschlüsse auf die Umsetzbarkeit und Ergebnisgüte des Verfahrens zu erzielen. Dazu wurden das in weiten Teilen analoge methodische Vorgehen und korrespondierende Zielsetzungen dargestellt, um die Relevanz der Projektergebnisse für die Bewertung der eigenen Methode sicherzustellen.
Es wurde gezeigt, dass im Projekt ähnlich komplexe Rahmenbedingungen vorliegen. Maßgeblich führt die Adaption von externen Modulen und verteilten Entwicklungen in prinzipiell isolierten Teams zu vergleichbar verteilten und heterogenen Strukturen, wie dies auch bei Testfeld und Testsystem der Fall ist. Obwohl der Architekturentwurf im Projekt kein vollwertiges Testfeld umfasst, stellt die Umsetzung im Projekt für die Bewertung der eigenen Methode einen guten Untersuchungsgegenstand dar.
Der vorangehende Abschnitt zeigt, dass die eigene Methode in praktischer Anwendung zu den gewünschten Ergebnissen führt: Die im Projekt entwickelten Architektursichten entsprechen den in Kapitel 3 diskutierten *4+1 Views*. Auf deren Grundlage konnten

entsprechende Prototypen entwickelt werden. Die qualitativen Aspekte wurden in Gremien dargestellt und diskutiert. Daher kann geschlussfolgert werden, dass der Architekturprozess im Projekt Instant Mobility und der dabei verfolgte methodische Ansatz grundsätzlich erfolgreich waren, um eine umfassende Architektur für das Projekt darzustellen bzw. fachlich kritische Aspekte prototypisch zu realisieren.
Da dieser inhaltlich der in Kapitel 3 vorgeschlagenen Methode und die komplexen Rahmenbedingungen denen eines Testfeldentwurfs entsprechen, kann im Sinn einer Methodenvalidierung abgeleitet werden, dass der eigene Ansatz in der konkreten Anwendung sowohl umsetzbar ist als auch zu einer adäquaten Testfeldarchitektur führt.

6 Zusammenfassung

Das erste Kapitel lieferte einen Einstieg in den Kontext von Testfeldern sowie der damit verbundenen komplexen Systemlandschaft und motiviert so die Relevanz der konkreten Fragestellung nach einem strukturieren Prozess zur Entwicklung einer adäquaten Testfeldarchitektur.
Diese wird in drei konkreten Forschungsfragen adressiert:

1. Wie lässt sich eine Architektur im speziellen Kontext des Testfeldentwurfs identifizieren und beschreiben?

2. Wie kann ein methodischer Entwurfsprozess für Testfeldarchitekturen gestaltet werden?

3. Wie lassen sich Testfeldarchitekturen vor der ersten Implementierung qualitativ bewerten und mit anderen Ansätzen vergleichen?

Daraus werden Zielsetzung und Gliederung der Arbeit abgeleitet, wie in Abb. 1.1 dargestellt.
Das folgende Kapitel 2 vertieft die testfeldrelevanten fachlichen Grundlagen sowie den Stand der Technik mit besonderem Augenmerk auf Technologien, die im Architekturkontext relevant sind.
Das Kapitel 3 baute auf dieser Grundlage einen methodischen Ansatz zur Spezifikation einer Testfeldarchitektur auf und durchlief dessen Entwicklungsschritte anhand typischer Aspekte des Testfeldentwurfs. Dazu wird eine funktionale Zerlegung der Anforderungen in atomare Funktionseinheiten vorgenommen und diese zu Testfeldprozessen choreographiert. Der Entwurf entwickelt so die Sichten des *4+1 View Model*, um eine umfassende Architekturbeschreibung zu erreichen.
Als Ergebnis dieses Kapitels liegt ein systematischer Testfeldarchitekturprozess vor, der die zweite Forschungsfrage beantwortet. Das Ergebnis des Prozesses ist eine adäquate Testfeldarchitekturbeschreibung in *4+1-Sichten*, welche die erste Forschungsfrage abdeckt.

Im Kapitel 4 wurde ein eigener strukturierter Ansatz zur Bewertung von Qualitätsmerkmalen einer Testfeldarchitektur aufgebaut. Mit diesem können insbesondere nicht-funktionale Anforderungen a priori qualitativ bewertet werden. Ergänzend wurde ein Konzept zur vergleichenden Architekturbewertung dargestellt und so eine Gegenüberstellung zwischen einer oder mehreren Testfeldarchitekturen ermöglicht. Beide Bewertungsverfahren beantworten die dritte Forschungsfrage. Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick zu möglichen weiterführenden Arbeiten.

6.1 Diskussion

Die wesentlichen Vorteile einer Softwarearchitektur und eines strukturierten Entwurfsprozesses sind in der Literatur de facto unbestritten, die Vorzüge der darauf aufbauenden Architekturbewertung bspw. hinsichtlich potenzieller Risiken, Austauschbarkeit und Analysierbarkeit offensichtlich. Dennoch sind bislang strukturierte Architekturprozesse und -bewertungen in der Testfelddomäne selten dokumentiert.

Die in dieser Arbeit dargestellten Verfahren zum methodischen Aufbau und zur Bewertung einer Testfeldarchitektur können daher einen wertvollen Leitfaden zur Entwicklung eigener komplexer Testfeldlandschaften und deren qualitativer Betrachtung liefern. So ist es möglich, den typischerweise durch Rahmenbedingungen geformten Entwurfsprozess methodischer und nachvollziehbarer zu gestalten. Auf Basis einer guten Anforderungsdefinition sollte die so entwickelte Systeminfrastruktur bspw. intuitiven Entwürfen deutlich in funktionalen und nicht-funktionalen Aspekten sowie mittel- und langfristig in Kosten und Nutzen überlegen sein. Insbesondere die Betrachtung der nicht-funktionalen Aspekte wie bspw. Interoperabilität, Wartbarkeit und Änderbarkeit führen sicher zu besseren Entwürfen, die über die Lebenszeit hinweg effizientere Anpassungen des Testfelds an die Evolution der Testsysteme und geänderte Anforderungen verschiedener Untersuchungszielsetzungen ermöglichen. Dies ermöglicht die wirtschaftlichere und sicherere Erprobung von ITS-Anwendungsfällen und ermöglicht bspw. über nachhaltige Testfeldaufbauten auch eine Entwicklung bzw. Erprobung für finanzschwächere kleine und mittelständische Unternehmen. Durch eine langfristige Ausrichtung und Wiederverwendbarkeit der Testfeldarchitektur bspw. im Kontext von NDS oder Analyseplattform können Testfelder über Kampagnen hinweg wirtschaftlicher betrieben werden, da deren individuelle Versuchsad-

aption mit geringerem Aufwand durchgeführt und der technische Aufbau nicht durch die Reimplementierung vormals entwickelter Komponenten aus vorhergehenden Kampagnen unnötig belastet wird.
Insbesondere der in Kapitel 3 dargestellte Zerlegungs- und Syntheseprozess bedarf substantieller Domänenkenntnisse und ist somit kein agnostischer, von der Domäne losgelöster Algorithmus, der es erlaubt, ohne weitere Grundlagen einen Entwurf über generische Konzepte zu synthetisieren. Wie auch bei der Adaption einer DSSA zu einer konkreten Architekturinstanz ist dies ein komplexer und aufwändiger Schritt, der weiterhin den Entwicklungsprozess eines Testfeldes belastet.

6.2 Ausblick

Die in Kapitel 5 dargestellte Anwendung der Methode in vergleichbarem Kontext kann als erster Hinweis auf deren Validität gesehen werden. Dennoch ist eine Validierung der hier vorgeschlagenen Entwurfs- und Bewertungsmethoden anhand einer konkreten Umsetzung auf Basis realer Anforderungen wünschenswert.
Die Bemühungen insbesondere in Kapitel 3 zur Identifikation generischer Eigenschaften von Testfeldern in die dieser Domäne und der Ableitung kleinster (Dienst-)Einheiten kann eine guter Startpunkt für eine DSSA bieten. Dazu kann auf Basis einer erschöpfenden Anforderungsanalyse ein entsprechendes umfassendes Domänenmodell[1] erstellt werden. Dies kann mit den vorgestellten Methoden zu einer Referenzarchitektur bzw. einer Komponentenbibliothek weiter entwickelt werden. Dadurch würde sich eine besser Adaption des impliziten Domänenwissens in daraus abgeleitete Architekturen realisieren lassen und der Entwurfsprozess weiter verkürzt.
Obwohl die Vorteile der über die Lebenszeit hinweg effizienteren Anpassungen des Testfelds an veränderliche Testsystembedingungen evident scheinen, kann in einer aufbauenden Studie untersucht werden, ob und in welchem Grad ein strukturierter Architekturprozess anderen Entwurfsparadigmen überlegen ist. Hier bietet sich auch eine detailliertere Kosten-Nutzen-Analyse an, die Aufwände zwischen dieser Methode und bspw. intuitiv getriebenen Entwurfsprozessen in Relation setzt.
Die wechselseitige Abhängigkeit zwischen Testsystem und Testfeld kann ein weiterer Untersuchungsgegenstand sein, beispielsweise können im Testsystem Methoden implementiert sein, die Funktionen des Testfeldes bedienen, wie Datenaufzeichnung bzw. Logging;

[1] vgl. Entwicklungsprozess in [TCY95]

bei einer NDS könnte die sichtbare Einrüstung mit Kameras zu einem abweichenden Verkehrsverhalten führen. Diese Effekte können im ungünstigsten Fall die Bewertung im Testfeld verzerren oder zur fehlerhaften Abschätzung beim Zielsystem führen, da dies kein reiner technischer Architekturaspekt ist, der bspw. auf Ebene von Trade-offs berücksichtigt wurde. Analog zu Übertragbarkeit von Erkenntnissen aus Simulatorstudien auf Realverhalten biete sich eine genauer Untersuchung dieses Aspektes an.
Wie in der Einleitung dargestellt, sind klare und methodische Architekturprozesse mit gut dokumentierten Entwürfen in der Domäne eher selten. Ebenso sind Testfelder, insbesondere im Rahmen von Projekten, oft von kurzer Lebensdauer und mangelnder Wiederverwendbarkeit. Diese Arbeit leistet einen Beitrag, qualitativ hochwertige Entwürfe für nachhaltige Testfelder zu entwickeln. Es ist zu hoffen, dass dies die Forschung und Entwicklung im Bereich von ITS beflügelt und so die gesellschaftlichen Herausforderungen adressiert.

Glossar

Zum erleichterten Verständnis der in dieser Arbeit verwendeten Begriffe aus der Verkehrsdomäne werden diese unten kurz zusammengefasst und erläutert. Dabei bezieht sich das Symbol ~ auf den zu erklärenden Begriff und das Symbol → referenziert einen weiteren, in diesem Glossar erläuterten Begriff.

Abstandsregeltempomat

Der ~ ist ein → *Assistenzsystem*, das den Fahrer bei der → *Längsführung* seines Fahrzeugs unterstützen kann. Dazu wird mittels fahrzeugseitiger Sensorik die Geschwindigkeit des Fahrzeugs automatisch an die des vorausfahrenden Fahrzeugs angepasst, um einen adäquaten Abstand zu wahren.

ACC

Adaptive Cruise Control, siehe → *Abstandsregeltempomat*

Assistenzsystem

Ein ~ unterstützt den Fahrer bei der Fahraufgabe. Die wesentliche Verantwortung liegt beim Fahrer, während das ~ ihn in Teilen entlastet [DMM15, S. 7f], wie bspw. bei Navigation oder → *Längsführung* (s. → *Abstandsregeltempomat*).

Dominion

~ ist eine Entwicklungs- und Laufzeitplattform, die eine strukturierte Entwicklung und auch den (verteilten) Betrieb prototypischer Systemimplementierungen in der gesamten → *Testfeldsystemlandschaft* erlaubt. Vgl. [SHG+10].

Einsatzfahrzeug

~ ist ein Fahrzeug von Behörden (bspw. Polizei) oder Hilfsorganisationen (bspw. Technisches Hilfswerk), welches einsatzbedingt über spezielle Sonder- und Wegerechte verfügen kann.

Erprobungskampagne

~ ist eine → *Kampagnen* im → *Testfeld* zur Erprobung eines → *Testsystems*.

Fahrassistenzsystem

siehe → *Assistenzsystem*

Feldtest

~ ist eine In-situ-Erprobung einer typischerweise eng abgesteckten Klasse von Anwendungsfällen. Ein ~ wird oft in Projekten mit begrenzter Laufzeit, insbesondere zur Validierung, eingesetzt. Siehe Abschnitt 2.4.

FESTA-Handbuch

Das ~ ist ein Leitfaden für die Organisation und praktische Durchführung von → *FOT* und → *NDS*.

FOT

siehe → *Feldtest*

IEEE 802.11p

~ ist ein Kommunikationsstandard zur drahtlosen Kommunikation, der einen vergleichsweise schnellen Verbindungsaufbau ermöglicht. Daher wird ~ im ITS vornehmlich zum Datenaustausch zwischen Fahrzeugen untereinander und mit Infrastruktur eingesetzt.

Intelligente Verkehrssysteme

~ bildet den Überbegriff für eine ganze Reihe verkehrstechnischer Konzepte zur Verbesserung von Sicherheit, Effizienz und Nachhaltigkeit auf Grundlage der Vernetzung von Verkehrsteilnehmern untereinander sowie mit stationärer Infrastruktur mittels → *Telematik* [Arn09, S. 2]. Siehe Abschnitt 1

ITS

Intelligent Transportation Systems, siehe → *Intelligente Verkehrssysteme*

Kampagne

Eine ~ beinhaltet oft zeitlich befristete Aktivitäten in einem → *Testfeld*, die bspw. die Erprobung einer bestimmten Anwendung in mehreren Versuchsdurchführungen umfassen können.

Kreuzungsassistenz

Umfasst → *Assistenzsysteme*, die Verkehrsteilnehmer in Kreuzungssituationen unterstützen. Bspw. durch Warnung vor einem bevorrechtigten → *Einsatzfahrzeug*, s. Abb. 2.2.

Längsführung

Die ~ bezeichnet den Aspekt der Geschwindigkeitskontrolle mittels Brems- und Gaspedal innerhalb der Fahraufgabe, wie bspw. Abstand halten oder vor Kreuzungen abbremsen.

Leitstand

Zur Versuchsdurchführung im → *Testfeld* müssen dessen Komponenten bspw. bereitgestellt, verschaltet und zur Laufzeit überwacht werden. Ein ~ bezeichnet das System, mit dem diese Aufgabe im → *Testfeld* gelöst wird.

Lichtsignal

Umgangssprachlich: Ampel. Das ~ kann unterschiedliche Leuchtsignalfarben anzeigen und so den Verkehrsfluss bspw. an einer Straßenkreuzung oder -einmündung regeln.

Lichtsignalanlage

~ bezeichnet den Zusammenschluss aller Komponenten, bspw. an einer Straßenkreuzung oder -einmündung, die mittels → *Lichtsignalen* den Verkehrsfluss an dieser Stelle steuern können.

Logistik

~ bezeichent alle Prozesse zum Transport, zur Lagerung und Kommissionierung von Waren.

Mautsystem

Das ~ umfasst alle Komponenten, die notwendig sind, Maut als Wegzoll für die Nutzung spezieller Infrastruktur, wie Brücken oder Autobahnstrecken, zu erheben.

Motormanagement

Das ~ ist typischerweise ein eingebettetes System im Fahrzeug, das die Motorsteuerung eines Verbrennungsmotors mittels Aktuatoren und Sensoren zur Aufgabe hat.

multimodal

Als ~ wird ein Gütertransport oder eine Reise beschrieben, wenn mindestens zwei unterschiedliche → *Verkehrsträger* dazu genutzt werden.

NDS

Eine *Naturalistic Driving Study* bezeichnet eine Untersuchungskampagne mit dem Ziel, unbeeinflusstes Fahrerverhalten zu beobachten. Dies erfolgt oft, um bspw. später in einem → *FOT* ein möglicherweise abweichendes Verhalten durch die Interaktion mit einem neuen System zu beobachten. Siehe auch ~ in Abschnitt 2.3.

OBU

~ bezeichnet eine mobile Rechnereinheit, die, typischerweise im Fahrzeug verbaut, ITS-Anwendungen unterstützt. Siehe Abschnitt 2.2.

Penetrationsrate

Die ~ , bspw. einer ITS-Anwendung oder Fahrzeugassistenz, bezeichnet das Verhältnis zwischen Verkehrsteilnehmern, die entsprechende Anwendung oder Ausrüstung einsetzen können, und denen, die diese nicht zur Verfügung haben.

Phasenwechsel

~ bezeichnet einen Schritt gemäß des → *Wechselzeichenprogramms* bei → *Lichtsignalen*, bspw. Übergang grün → gelb.

Pilotversuch

~ ist eine → *Kampagnen* im → *Testfeld* zur Demonstration oder Erprobung eines → *Testsystems*, um die möglicherweise technisch risikobehaftete Entwicklung und Fragen wie Akzeptanz und Wirtschaftlichkeit im → *Testfeld* vor Markteinführung zu bewerten.

Querführung

Die ~ bezeichnet den Lenkaspekt innerhalb der Fahraufgabe, wie bspw. Spur halten, abbiegen oder ausweichen.

Reiseassistenz

~ ist eine Dienstleistung, die einen Reisenden während seiner Reise leitet und unterstützt. Im Kontext dieser Arbeit ist ~ eine Software auf einem mobilen Endgerät, welche diese Aufgabe übernimmt. Dadurch kann flexibel auf die Bedürfnisse des Reisenden eingegangen werden und viele Vorgänge, wie bspw. Ticketumbuchung bei Abweichungen vom geplanten Reiseverlauf, im Hintergrund schnell und ohne wesentlichen Eingriff des Reisenden ausgeführt werden.

Rotphase

Ein Phase im → *Wechselzeichenprogramm* eines → *Lichtsignals*. Aus Sicht eines Fahrers darf der durch das → *Lichtsignal* geschützte Bereich während der ~ nicht befahren werden.

RSU

~ bezeichnet eine ortsfeste Rechnereinheit, die u. a. Zugriff auf Verkehrsinfrastruktur wie Lichtsignalanlagen ermöglicht. Siehe Abschnitt 2.2.

Telematik

Portmanteauwort zusammengesetzt aus Telekommunikation und Informatik; ~ ist die Verbindung von zwei oder mehr rechnergestützten Informationssystemen mittels Telekommunikation.

Telemetrie

Als ~ wird die ortsferne Erfassung von Messwerten bezeichnet, wie bspw. Sensorwerte eines Fahrzeugs, die via Mobilfunk an eine stationäre Einheit übermittelt werden.

Telemetriedaten

Die ~ sind die durch ein → *Telemetriesystem* erfassten (Mess-)Daten.

Telemetriesystem

Bezeichnet den Verbund aus Komponenten und Verfahren um → *Telemetrie* zu betreiben.

Testfeld

Das ~ ist eine Infrastruktur, die bspw. eine Entwicklung und Erprobung von → *Testsystemen* und seinen Komponenten ermöglicht. Siehe Abschnitt 2.3.

Testfeldarchitektur

Eine ~ bezeichnet eine Softwarearchitektur, die ein → *Testfeld* umfassend beschreibt. Siehe auch Softwarearchitektur in Abschnitt 2.7.

Testfeldarchitekturprozess

Ein ~ bezeichnet den Entwurfsvorgang zur Definition einer geeigneten → *Testfeldarchitektur*. Siehe Abschnitt 2.14.

Testfeldsystemlandschaft

Die ~ umfasst alle im → *Testfeld* eingesetzten Komponenten, wie Hard- und Software, sowie deren Kommunikation bzw. Schnittstellen sowie ggf. das eingebettete → *Testsystem*.

Testsite

siehe → *Testfeld*

Testsystem

Das ~ ist Untersuchungs-, Entwicklungs-, Erprobungs- oder Demonstrationsgegenstand in einem → *Testfeld*. Im Rahmen dieser Arbeit ist dies eine ITS-Anwendung. Siehe Abschnitt 2.2.

Validierung

Die ~ ist die Bewertung einer ITS-Anwendung, um abzusichern, dass die Anwendung die Zielsetzung adäquat adressiert. Vgl. → *Verifikation*.

Verifikation

Die ~ ist die Bewertung einer ITS-Anwendung, um abzusichern, dass die Anwendung gemäß Spezifikation umgesetzt wurde. Vgl. → *Validierung*.

Verkehrsmanagement

Ziel von ~ ist die Verbesserung der Verkehrsabläufe innerhalb eines → *Verkehrssystems* durch Beeinflussung der → *Verkehrsteilnehmer*. Beispielsweise können von einem städtischen ~ → *Wechseltextanzeigen* genutzt werden, um → *Verkehrsteilnehmer* auf mögliche Staus im Stadtgebiet hinzuweisen und alternative Routen vorzuschlagen.

Verkehrssystem

Das ~ bezeichnet alle räumlichen, zeitlichen, technischen, organisatorischen, politischen, juristischen oder betrieblichen Gesichtspunkte [AH06, S. 37f], die zur Erbringung einer speziellen Transportleistung notwendig sind. Im Rahmen dieser Arbeit sind dies u. a. Fahrzeuge, → *Lichtsignalanlage*, Straße, Kreuzung, Fahrer als Teil des straßengebundenen Verkehrssystems.

Verkehrsteilnehmer

Ein ~ ist eine Person, die aktiv am Verkehr teilnimmt. Bspw. als Fahrer oder Fußgänger.

Verkehrsträger

Der ~ umfasst alle Anbieter von Transportleistungen und lässt sich u.a. über das genutzte → *Verkehrssystem* klassifizieren. [AH06, S. 44]

Versuchszentrale

~ bezeichnet eine ortsfeste Rechnereinheit, die u. a. in der Rolle des → *Leitstands* die zur Versuchsdurchführung relevanten Komponenten im → *Testfeld* bereitstellt, handhabt und Daten aufzeichnet. Siehe Abschnitt 3.8.

Wechseltextanzeige

Die ~ ist üblicherweise eine fahrbahnnahe Anzeigetafel. Diese kann Verkehrsteilnehmern kurze und leicht lesbare Informationen im Textformat zeigen. Dadurch können diese bspw. vor besonderen Situationen wie Umleitungen oder Stau informiert werden. Siehe auch *Electronic traffic signs* in Abb. 2.8, S. 32.

Wechselzeichenprogramm

~ bezeichnet den Ablauf verschiedener Phasen bei → *Lichtsignalen*. Eine übliche, wiederholte Reihenfolge ist bspw. rot → gelb-rot → grün → gelb → rot ...

Abkürzungen

ACC	Adaptive Cruise Control
ADL	Architecture Description Language
AIM	Anwendungsplattform Intelligente Mobilität
ALMA	Architecture-Level Modifiability Analysis
ALPSM	Architecture Level Prediction of Software Maintenance
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AQA	Architecture Quality Assessment
ARI	Autofahrer-Rundfunk-Information
ATAM	Architecture Tradeoff Analysis Method
AU	Application Unit (nach sim^{TD})
BMVI	Bundesministerium für Verkehr und digitale Infrastruktur
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
CALM	Communication Access for Land Mobiles
CAN	Controller Area Network
CCU	Communication & Control Unit
CORBA	Common Object Request Broker Architecture
CVIS	Cooperative Vehicle Infrastructure Systems
DACF	Domain Architecture Comparison Framework
DITCM	Dutch Integrated Testsite Cooperative Mobility
DLR	Deutsches Zentrum für Luft- und Raumfahrt

DoSAM	Domain-Specific Software Architecture Comparison Model
DSSA	Domain-Specific Software Architecture
EC	Europäische Kommission
EDGE	Enhanced Data Rates for GSM Evolution
ETSI	Europäische Institut für Telekommunikationsnormen
EU	Europäische Union
FAAM	Family Architecture Assessment Method
FIT	Future Internet Technologies
FI PPP	Future Internet Public-Private Partnership
FOT	Field Operational Test
FOTIP	FOT Implementation Plan
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
HiL	Hardware in the Loop
HMI	Human-Machine Interaction
HSPA	High Speed Packet Access
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	Internationale Organisation für Normung
ITS	Intelligent Transportation Systems
LRP	Long Running Processes
LTE	3GPP Long Term Evolution
MVC	Model-View-Controller
NDS	Naturalistic driving studies

OBU	On-Board Unit
OO	Objektorientierung
ORB	Object Request Broker
OSI	Open Systems Interconnection (Reference Model)
P2P	Peer-to-Peer
PASA	Performance Assessment of Software Architectures
QoS	Quality of Service
QSEM	Quantitative Scalability Evaluation Method
RAD	Rapid Application Development
ROA	Resource Oriented Architecture
RSU	Road-Side Unit
SAAM	Software Architecture Analysis Method
SAAMCS	Scenario-based Architecture Analysis Method of Complex Scenarios
SAAMER	Software Architecture Analysis for Evolution and Reusability
SACAM	The Software Architecture Comparison Analysis Method
SAP	Service Access Point
SBAR	Scenario-based Software Architecture Reengineering
SDK	Software development kit
SiL	Software in the Loop
SNMP	Simple Network Management Protocol
SOA	Serviceorientierte Architektur
SOAP	Simple Object Access Protocol
SRP	Short Running Processes
SysML	Systems Modeling Language
TASC	Test Service Aggregation Center
TSN	Test Site Norway

TSS	Test Site Sweden
UDDI	Universal Description, Discovery and Integration
UEP	User Experience Project
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
UT	Utility Tree
WLAN	Wireless Local Area Network
WSDL	Web Services Description Language
XML	Extensible Markup Language

Abbildungen

Tabellen

Literatur

[ABC+97] Abowd, Gregory, Len Bass, Paul Clements, Rick Kazman, Linda Northrop und Amy Zaremski: *Recommended Best Industrial Practice for Software Architecture Evaluation.* Technischer Bericht CMU/SEI-96-TR-025, Software Engineering Institute, Carnegie Mellon University, Jan. 1997.

[ACD10] Almonaies, Asil A., James R. Cordy und Thomas R. Dean: *Legacy system evolution towards service-oriented architecture.* In: *International Workshop on SOA Migration and Evolution*, Seiten 53–62, März 2010.

[AH06] Ammoser, Hendrik und Mirko Hoppe: *Glossar Verkehrswesen und Verkehrswissenschaften: Definitionen und Erläuterungen zu Begriffen des Transport- und Nachrichtenwesens.* Nummer 2 in *Diskussionsbeiträge aus dem Institut für Wirtschaft und Verkehr.* Technische Universität Dresden; Inst. für Wirtschaft und Verkehr, Feb. 2006.

[All97] Allen, Robert: *A Formal Approach to Software Architecture.* Dissertation, Carnegie Mellon, School of Computer Science, Jan. 1997.

[APHH16] Aapaoja, Aki, Eetu Pilli-Sihvola, Ville Hinkka und Raine Hautala: *Preconditions for establishing and maintaining test sites for cooperative mobility.* In: *Proceedings of the 23rd World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2016.

[Arn09] Arndt, Martin: *Towards a pan European architecture for cooperative systems.* Presentation of the 16th World Congress and Exhibition on Intelligent Transport Systems and Services, Sep. 2009.

[Aut14] Automotive Campus, Helmond: *DITCM Testsite & Facilities.* Broschüre, Dez. 2014.

[AWT+08] Agerholm, Niels, Rasmus Waagepetersen, Nerius Tradisauskas, L. Harms und Harry Lahrmann: *Preliminary results from the Danish Intelligent Speed Adaptation project Pay as You Speed*. IET Intelligent Transport Systems, 2(2), Juni 2008.

[B+09] Bossom, Richard et al.: *European ITS Communication Architecture – Overall Framework – Proof of Concept Implementation*. Technischer Bericht D31, COMeSafety (Communication for eSafety) Consortium, 2009.

[BAB+00] Boehm, Barry W., Chris Abts, A. Winsor Brown, Sunita Chulani und Bradford K. Clark: *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR, Aug. 2000.

[BAG+00] Bossom, Richard, Victor Avontuur, Jean-François Gaillet, Gino Franco und Peter Jesty: *European ITS Framework Architecture – Overview*. Technischer Bericht D3.6, KAREN (Keystone Architecture Required for European Networks) Project, Aug. 2000.

[BAN+12] Beckmann, Dirk, Cristina Pena Alcega, Thierry Nagellen, Rafael Salinas und Jean Marie Dautelle: *WP5.1 - Instant Mobility prototype description*. Technischer Bericht D5.1, Instant Mobility Project, Sep. 2012.

[BB98] Bengtsson, PerOlof und Jan Bosch: *Scenario-based software architecture reengineering*. In: *Proceedings of Fifth International Conference on Software Reuse*, Juni 1998.

[BB99] Bengtsson, PerOlof und Jan Bosch: *Architecture Level Prediction of Software Maintenance*. In: *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, CSMR 99. IEEE Computer Society, März 1999.

[BC03] Bray, Ian K. und Karl Cox: *The Simulator; Another, Elementary Problem Frame?* In: Salinesi, C., B. Regnell und E. Kamsties (Herausgeber): *Proceedings of the Ninth International Workshop on Requirements Engineering: Foundation for Software Quality-REFSQ 2003*, Juni 2003.

[BCB97] Bass, Len, Paul Clements und Ken Bass: *Software Architecture in Practice*. Addison-Wesley Professional, Dez. 1997.

[BFB10] Benmimoun, Mohamed, Felix Fahrenkrog und Ahmed Benmimoun: *Automatisierte Situationserkennung zur Bewertung des Potentials von Fahrerassistenzsystemen im Rahmen des Feldversuchs euroFOT.* In: *Fahrerassistenz und Integrierte Sicherheit. 26. VDI/VW-Gemeinschaftstagung*, Okt. 2010.

[BHS07] Buschmann, Frank, Kevlin Henney und Douglas C. Schmidt: *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages.* Wiley, Juni 2007.

[BKRC10] Bieker, Laura, Daniel Krajzewicz, Matthias Röckl und Hans Capelle: *Derivation of a fast, approximating 802.11p simulation model.* In: *Proceedings of the Intelligent Transport Systems Telecommunications (ITST2010)*, Nov. 2010.

[BKZS12] Beckmann, Dirk, Frank Köster, Mahdi Zargayouna und Gérard Scemama: *Architecture methodology in the Instant Mobility project.* In: *Proceedings of the 19th World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2012.

[Bos00] Bosch, Jan: *Design and Use of Software Architecture: Adopting and Evolving a Product-Line Approach.* Addison-Wesley, 2000.

[Bos04] Bosch, Jan: *Software Architecture: The Next Step.* In: Oquendo, Flavio, Brian Warboys und Ron Morrison (Herausgeber): *Software Architecture*, Band 3047 der Reihe *Lecture Notes in Computer Science*, Seiten 194–199. Springer Berlin / Heidelberg, 2004.

[BP09] Bechler, Marc und PRE DRIVE C2X Project Consortium: *Refined Architecture.* Technischer Bericht D1.4, PRE DRIVE C2X Project Consortium, 2009.

[Bro95] Brooks, Frederick P.: *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition).* Addison-Wesley Professional, Aug. 1995.

[BRST05] Bergner, Klaus, Andreas Rausch, Marc Sihling und Thomas Ternité: *DoSAM – Domain-Specific Software Architecture Comparison Model.* In:

Proceedings of the First international conference on Quality of Software Architectures and Software Quality, and Proceedings of the Second International conference on Software Quality, Seiten 4–20, Sep. 2005.

[BST11] Beckmann, Dirk, Tobias Schlauch und Daniel Thielen: *WP4.1 - Global architecture definition & requirements.* Technischer Bericht D4.1, Instant Mobility Project, Dez. 2011.

[BSvV13] Beckmann, Dirk, Tobias Schlauch und Hans van Vliet: *WP4.1 - Instant Mobility functional & technical specifications.* Technischer Bericht D4.16, Instant Mobility Project, März 2013.

[Bun15] Bundesministerium für Verkehr und digitale Infrastruktur (BMVI): *Strategie automatisiertes und vernetztes Fahren*, Sep. 2015.

[Bun19] Bundesministerium für Verkehr und digitale Infrastruktur: *Digitale Testfelder für das automatisierte und vernetzte Fahren im Realverkehr in Deutschland (Stand: Januar 2019)*, Jan. 2019. Online erhältlich unter `https://www.bmvi.de/SharedDocs/DE/Anlage/DG/Digitales/uebersicht-digitale-testfelder-avf-bmvi.pdf?__blob=publicationFile`; zuletzt abgerufen am 18.12.2019.

[Bv11] Beckmann, Dirk und Martijn van Noort: *Integrated architecture for the assessment of ITS in a test bed.* In: *Proceedings of the 8th European Congress and Exhibition on Intelligent transport Systems and Services*, Juni 2011.

[BVJ+11] Beckmann, Dirk, Diana Vonk Noordegraaf, Eline Jonkers, Erwin Vermassen, Jürgen Rataj und Frank Köster: *Definition of the reference architecture.* Technischer Bericht D2.1, ITS Test Beds Consortium, 2011.

[BVSv16] Broeders, Wim, Jaap Vreeswijk, Paul Spaanderman und Borgert van der Kluit: *Dutch ITS profile.* Technischer Bericht Version 0.2, DITCM Innovations, Sep. 2016.

[CdlFBS01] Cuesta, Carlos E., Pablo de la Fuente und Manuel Barrio-Solárzano: *Dynamic coordination architecture through the use of reflection.* In: *Proceedings of the 2001 ACM symposium on Applied computing*, SAC '01. ACM, 2001.

[CGB+10] Clements, Paul, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers und Reed Little: *Documenting Software Architectures: Views and Beyond*. SEI Series in Software Engineering. Addison-Wesley Professional, 2 Auflage, Okt. 2010.

[CKK01] Clements, Paul, Rick Kazman und Mark Klein: *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, Nov. 2001.

[Cle94] Clements, Paul C.: *From Domain Model to Architectures*. In: *Focused Workshop on Sofinr-e Architemwe*, Seiten 404–420, 1994.

[Cle96] Clements, Paul C.: *A Survey of Architecture Description Languages*. In: *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD '96, Washington, DC, USA, 1996. IEEE Computer Society.

[CMV+10] Cloutier, Robert, Gerrit Muller, Dinesh Verma, Roshanak Nilchiani, Eirik Hole und Mary Bone: *The Concept of Reference Architectures*. Systems Engineering, 13(1):14–27, Feb. 2010.

[Cou05] Coulouris, George: *Distributed Systems: Concepts and Design*. Addison Wesley, 4. Auflage, Mai 2005.

[CTK19] Cha, Sungdeok, Richard N. Taylor und Kyochul Kang: *Handbook of Software Engineering*. Springer, 2019.

[DIN94] DIN Deutsches Institut für Normung: *Bewerten von Softwareprodukten: Qualitätsmerkmale und Leitfaden zu ihrer Verwendung; identisch mit ISO/IEC 9126: 1991*. Deutsche Norm. Beuth, Okt. 1994.

[DIT15] DITCM Innovation services: *DITCM Programme 2015-2019; The Netherlands as a Living Lab for Cooperative Driving*. Broschüre, Jan. 2015.

[DMM15] Dokic, Jadranka, Beate Müller und Gereon Meyer: *European roadmap smart systems for automated driving*. European Technology Platform on Smart Systems Integration, Apr. 2015.

[DN02] Dobrica, Liliana und Eila Niemelä: *A survey on software architecture analysis methods*. IEEE Trans. Softw. Eng., 28(7):638–653, Juli 2002.

[Dol01] Dolan, Thomas J.: *Architecture Assessment of Information-system Families: A Practical Perspective.* Dissertation, Technische Universiteit Eindhoven, Jan. 2001.

[EHM07] Eicker, Stefan, Christian Hegmanns und Stefan Malich: *Auswahl von Bewertungsmethoden für Softwarearchitekturen.* ICB Research Reports 14, University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB), 2007.

[ESLR06] Ehrlich, Jacques, Farida Saad, Sylvain Lassarre und Sébastien Romon: *Assessment of LAVIA systems: experimental design and first results on system use and speed behavior.* In: *Proceedings of the 13th World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2006.

[ETS09] ETSI: *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions.* Technischer Bericht ETSI TR 102 638 V1.1.1, European Telecommunications Standards Institute, Juni 2009.

[ETS10a] ETSI: *Human Factors (HF); Intelligent Transport Systems (ITS); ICT in cars.* Technischer Bericht ETSI TR 102 762 V1.1.1, European Telecommunications Standards Institute, Apr. 2010.

[ETS10b] ETSI: *Intelligent Transport Systems (ITS); Communications Architecture.* Technischer Bericht ETSI EN 302 665 V1.1.1, European Telecommunications Standards Institute, Sep. 2010.

[Eur10] Europäisches Parlament und Rat der Europäischen Union: *Richtlinie 2010/40/EU des Europäischen Parlaments und des Rates zum Rahmen für die Einführung intelligenter Verkehrssysteme im Straßenverkehr und für deren Schnittstellen zu anderen Verkehrsträgern*, Jul. 2010.

[FB10] Fahrenkrog, Felix und Ahmed Benmimoun: $\mathsf{sim}^{\mathsf{TD}}$ – *A Large Scale Field Trial with Co-operative Systems*, 2010.

[FES17] FESTA Consortium: *FESTA Handbook (Version 7).* Field opErational teSt supporT Action (FESTA), Jan. 2017.

[FGS+15] Fricke, Nicola, Stefan Griesche, Anna Schieben, Tobias Hesse und Martin Baumann: *Driver behavior following an automatic steering intervention.* Accident Analysis & Prevention, 83, Okt. 2015.

[FHL+11] Frankiewicz, Tobias, Arno Hinsberger, Tobias Lorenz, Hans-Josef Hilt, Sebastian Weber, Horst Wieker und Frank Köster: *Standortbestimmung und Integration von ITS Roadside Stations für die Anwendungsplattform Intelligente Mobilität.* In: *AAET - Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, Feb. 2011.

[FJM+01] Figueiredo, Lino, Isabel Jesus, J. A. Tenreiro Machado, Jose Rui Ferreira und J. L. Martins de Carvalho: *Towards the development of intelligent transportation systems.* In: *Intelligent Transportation Systems.* IEEE, Aug. 2001.

[FKL16] Förster, David, Frank Kargl und Hans Löhr: *PUCA: A pseudonym scheme with strong privacy guarantees for vehicular ad-hoc networks.* Ad Hoc Networks, 37, Feb. 2016.

[FKM14] Frankiewicz, Tobias, Frank Köster und Meike Möckel: *Measurement and Evaluation of Communication parameters on a Vehicle-to-Infrastructure Communication Test Site.* In: *International Conference on Connected Vehicles & Expo (ICCVE 2014)*, Nov. 2014.

[FOF+10] Fisher, Hans-Joachim, Erik Olsen, Imre Fazekas, Jean-François Gaillet und Matthias Mann: *Final Architecture and System Specifications.* Technischer Bericht D3.4, CVIS integrated project, Juni 2010.

[FOT11] FOT-NET Consortium: *FOT-Net Wiki, the free encyclopaedia of Field Operational Tests*, 2011. Online erhältlich unter `http://wiki.fot-net.eu`; zuletzt abgerufen am 12.09.2016.

[FP98] Fenton, Norman E. und Shari Lawrence Pfleeger: *Software Metrics: A Rigorous and Practical Approach.* PWS Publishing Co., Boston, MA, USA, 2. Auflage, 1998.

[FSK12] Frankiewicz, Tobias, Lars Schnieder und Frank Köster: *Application platform for Intelligent Mobility - Test site architecture and Vehicle2X communication setup.* In: *Proceedings of the 19th World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2012.

[Gar84] Garvin, David: *What does product quality really mean?* MIT Sloan Management Review, 26(1):Seiten 25–45, Okt. 1984.

[Gar00] Garlan, David: *Software architecture: a roadmap.* Proc. of the 22nd International Conference on Software Engineering, Future of Software Engineering Track, Seiten 91–101, Feb. 2000.

[GHJV01] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, Bonn, März 2001.

[GJKF10] Gacnik, Jan, Henning Jost, Frank Köster und Martin Fränzle: *The DeSCAS Methodology and Lessons Learned on Applying Formal Reasoning to Safety Domain Knowledge.* In: *FORMS/FORMAT 2010 – 8th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems*, Dez. 2010.

[GNH+10] Grossmann, Jürgen, Carsten Neumann, Andreas Hinnerichs, Horst Rechner, Thomas Hecker und Ilja Radusch: *Test von verteilten C2X-Applikationen*, Band 2, Seiten 491–496. Gesellschaft für Informatik (GI), 2010.

[GR08] Graham, T. C. Nicholas und Banani Roy: *Methods for Evaluating Software Architecture: A Survey.* Technischer Bericht, School of Computing 2008-545 – Queen's University, Apr. 2008.

[GS93] Garlan, David und Mary Shaw: *An Introduction to Software Architecture*, Band 2 der Reihe *Series on Software Engineering and Knowledge Engineering*, Seiten 1–39. World Scientific Publishing Company, 1993.

[GSV15] Gesellschaft für Informatik e. V., SafeTRANS e. V. und Verband der Automobilindustrie e. V. (Herausgeber): *Automotive Roadmap Embedded Systems 2015.* SafeTRANS e. V., Sep. 2015.

[Hal77] Halstead, Maurice Howard: *Elements of software science.* Operating and programming systems series. Elsevier, 1977.

[HKL97] Hilliard II, Richard F., Michael J. Kurland und Steven D. Litvintchouk: *MITRE's Architecture Quality Assessment.* In: *Proceedings of 1997 Software Engineering & Economics Conference*, 1997.

[HKN+07] Hofmeister, Christine, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran und Pierre America: *A General Model of Software Architecture Design Derived from Five Industrial Approaches.* Journal of Systems and Software, 80(1):106–126, Jan. 2007.

[HMF+11] Haeusler, Florian, Henning Mosebach, Peter Follin, Dieter Heussner, Harri Koskinen, Sander Maas, Filippo Visintainer, Marga Sáez, Francisco Sanchez, Robert Protzmann und Thierry Ernst: *Internal report IR33.3 – Consolidated description of test sites.* Technischer Bericht, DRIVE C2X Project Consortium, Sep. 2011.

[HNS99] Hofmeister, Christine, Robert Nord und Dilip Soni: *Applied Software Architecture.* Addison-Wesley Professional, Nov. 1999.

[HNS+10] Hinnerichs, Andreas, Carsten Neumann, Christian Schlotter, Guido Hoffmann, Özgür Ünalan und Torsten Ronneberger: *Anforderungen an das Gesamttestsystem.* Technischer Bericht D13.3, sim$^{\mathrm{TD}}$ Konsortium, Apr. 2010.

[HR94] Hayes-Roth, Frederick: *Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program.* Technischer Bericht, Teknowledge Federal Systems, Feb. 1994.

[HRPL+95] Hayes-Roth, Barbara, Karl Pfleger, Philippe Lalanda, Philippe Morignot und Marko Balabanovic: *A Domain-Specific Software Architecture for Adaptive Intelligent Systems.* IEEE Trans. Softw. Eng., 21(4):288–301, Apr. 1995.

[IEE00] IEEE Architecture Working Group: *IEEE Std 1471-2000, Recommended practice for architectural description of software-intensive systems.* Technischer Bericht, IEEE, Okt. 2000.

[IEE03] IEEE: *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (ANSI/IEEE Std 802.11, 1999 Edition (R2003)).* Institute of Electrical and Electronics Engineers, Inc., Juni 2003.

[Ins12] Instant Mobility Consortium: *Instant Mobility – Partners*, 2012. Online erhältlich unter `http://instant-mobility.com/index.php/partners.html`; zuletzt abgerufen am 10.01.2014.

[ISO94] ISO/IEC 7498-1:1994: *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. Standard, ISO/IEC, 1994.

[ISO01] ISO/IEC 9126: *Software engineering – Product quality*. Standard, ISO/IEC, Jun. 2001.

[ISO11] ISO/IEC 42010:2011: *Systems and Software engineering – architecture descriptions*. Standard, ISO/IEC, Dez. 2011.

[JA05] Jimoh, Y.A. und O.O. Adeleke: *Potential Benefits of Intelligent Transport System (ITS) in Nigeria*. USEP: Journal Of Research In Civil Engineering, 2(1):46–56, 2005.

[JvdB11] Jonkers, Eline, Martijn van Noort, Martijn de Kievit und Frank Berkers: *Integrated Assessment of societal impacts of intelligent Transport Systems in the ITS Test Beds project*. In: *Proceedings of the 8th European Congress and Exhibition on Intelligent transport Systems and Services*, Juni 2011.

[KABC96] Kazman, Rick, Gregory Abowd, Len Bass und Paul Clements: *Scenario-Based Analysis of Software Architecture*. IEEE Software, 13(6), 1996.

[KHFL11] Köster, Frank, Marco Hannibal, Tobias Frankiewicz und Karsten Lemmer: *Anwendungsplattform für Intelligente Mobilität – Dienstespektrum und Architektur*. AAET – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel, Feb. 2011.

[KKC00] Kazman, Rick, Mark Klein und Paul Clements: *ATAM: Method for Architecture Evaluation*, Aug. 2000.

[KLG16] Knake-Langhorst, Sascha und Kay Gimm: *AIM Research Intersection: Instrument for traffic detection and behavior assessment for a complex urban intersection*. Journal of large-scale research facilities, 2(A65):1–5, Apr. 2016.

[KLGFK16] Knake-Langhorst, Sascha, Kay Gimm, Tobias Frankiewicz und Frank Köster: *Test Site AIM - Toolbox and Enabler for Applied Research and Development in Traffic and Mobility.* Transportation Research Procedia, 14:2197–2206, Juni 2016.

[KLV+11] Kompfner, Paul, Yanying Li, Fillipo Visitainer, Gilles Betis, Annika Strömdahl und Cristina Torres: *WP3.1 - Instant Mobility Use Case Scenarios definition & analysis, preliminary report.* Technischer Bericht D3.1, Instant Mobility Project, Sep. 2011.

[Kom08a] Kommission der Europäischen Gemeinschaften: *Mitteilung der Kommission zum Aktionsplan zur Einführung intelligenter Verkehrssysteme in Europa*, Dez. 2008.

[Kom08b] Kommission der Europäischen Gemeinschaften: *Vorschlag für eine Richtlinie des Europäischen Parlaments und des Rates zur Festlegung eines Rahmens für die Einführung intelligenter Verkehrssysteme im Straßenverkehr und für deren Schnittstellen zu anderen Verkehrsträgern*, Dez. 2008.

[Kom10] Kompfner, Paul: *The ITS App Store*, 2010.

[Kom11] Kommission der Europäischen Gemeinschaften: *Weissbuch: Fahrplan zu einem einheitlichen europäischen Verkehrsraum – Hin zu einem wettbewerbsorientierten und ressourcenschonenden Verkehrssystem*, März 2011.

[KP88] Krasner, Glenn E. und Stephen T. Pope: *A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80.* Journal of Object-Oriented Programming, Aug. 1988.

[Kru95] Kruchten, Philippe: *Architectural Blueprints: The „4+1“ View Model of Software Architecture.* IEEE Software, 12(6), Nov. 1995.

[Kös16] Köster, Frank: *Anwendungsplattform für Intelligente Mobilität – Dienstespektrum und Architektur.* ZEVrail, Jahrgang 140 (Ausgabe 8), Aug. 2016.

[Kut05] Kutter, Eckhard: *Entwicklung innovativer Verkehrsstrategien für die mobile Gesellschaft: Aufgaben, Maßnahmenspektrum, Problemlösungen.* Erich Schmidt, Apr. 2005.

[LAPB12] Li, Yanying, Cédric Andry, Cristina Peña und Marco Bottero: *Instant Mobility Use Case Scenarios Functional and Non-Functional Requirements.* Technischer Bericht D3.5, Instant Mobility Project, Apr. 2012.

[LBJK11] Lorenz, Tobias, Martin Baumann, Klaus Jaschke und Frank Köster: *A Modular and Scalable Application Platform for Testing and Evaluating ITS Components.* In: *20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.* IEEE Computer Society Conference Publishing Services (CPS), 2011.

[LBKK97] Lung, Chung-Horng, Sonia Bot, Kalai Kalaichelvan und Rick Kazman: *An Approach to Software Architecture Analysis for Evolution and Reusability.* In: *Proceedings of CASCON (Centre for Advanced Studies Conference)*, Nov. 1997.

[LBvVB02] Lassing, Nico, PerOlof Bengtsson, Hans van Vliet und Jan Bosch: *Experiences with ALMA: Architecture-Level Modifiability Analysis – Architecture Analysis Experiences.* Journal of Systems and Software, 61:47–57, 2002.

[Lem15] Lemmer, Karsten: *Neue autoMobilität: Automatisierter Straßenverkehr der Zukunft.* acatech POSITION. Utz, 2015.

[LL02] Leviäkangas, Pekka und Jukka Lähesmaa: *Profitability evaluation of intelligent transport system investments.* Journal of transportation engineering, 128(3):11, 2002.

[LMV09] Liu, Dahai, Nikolas D. Macchiarella und Dennis A. Vincenzi: *Simulation fidelity.* Human factors in simulation and training, Seiten 61–73, 2009.

[LRv99] Lassing, Nico, Daan Rijsenbrij und Hans van Vliet: *On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn't Everything.* In: *Proceedings of the Second Nordic Software Architecture Workshop (NOSA '99)*, 1999.

[LT64] Luce, Duncan R. und John W. Tukey: *Simultaneous conjoint measurement: A new type of fundamental measurement.* Journal of Mathematical Psychology, 1(1):1–27, 1964.

[MA08] Molnar, Eva und Constantinos Alexopoulos: *ITS in urban transport: the challenges for the UNECE Transport Division.* Eurotransport Magazine, 5:26–29, Nov. 2008.

[Mar15] Markets and Markets: *Intelligent Transportation System Market by Roadway (Hardware, Software, & Services), Aviation Tool (Kiosk, Multi-User Flight Information Display, and Smart Gate System), Railway, Maritime, Protocol, Application, and Geography – Global Forecast to 2022*, Juli 2015.

[MBM+11] Mosebach, Henning, Dirk Beckmann, Mohamed Mahmod, Florian Häusler, Andras Csepinszky, Francois Fischer und Ion Giosan: *Operational and Technical Guidelines (Deliverable).* Technischer Bericht D27.1, DRIVE C2X Consortium, Dez. 2011.

[McC76] McCabe, Thomas J.: *A Complexity Measure.* IEEE transactions on software engineering, 2(4):308–320, 1976.

[McC97] McConnell, Steve: *Software's Ten Essentials.* IEEE Software, 14(2):144, 143, 1997.

[MHC05] Meier, René, Anthony Harrington und Vinny Cahill: *A framework for integrating existing and novel intelligent transportation systems.* In: *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, Seiten 154–159, Sep. 2005.

[MHR79] M. H. Reenskaug, Trygve: *Models-Views-Controllers.* Technischer Bericht, Xerox PARC, Dez. 1979.

[MJB+12] Mårtensson, Frans, Per Jönsson, Perolof Bengtsson, Håkan Grahn und Michael Mattsson: *A Case Against Continuous Simulation for Software Architecture Evaluation.* Applied Simulation and Modelling (ASM2003), Apr. 2012.

[Nag12] Nagellen, Thierry: *WP1 - First Technical Quality Report.* Technischer Bericht D1.5, Instant Mobility Project, Apr. 2012.

[NBM12] Nakagawa, Elisa Yumi, Martin Becker und José Carlos Maldonado: *A knowledge-based framework for reference architectures.* In: Ossowski, Sascha und Paola Lecca (Herausgeber): *Proceedings of the 27th Symposium on Applied Computing*, Seiten 1197–1202. ACM, 2012.

[NDK+05] Neale, Vicki L., Thomas A. Dingus, Sheila G. Klauer, Jeremy Sudweeks und Michael Goodman: *An Overview of the 100-Car Naturalistic Study and Findings*, 2005.

[NPW+13] Netten, Bart, Igor Passchier, Harry Wedemeijer, Sander Maas und Coen van Leeuwen: *Technical Evaluation of Cooperative Systems – Experience from the DITCM Test Site.* In: *Proceedings of the 9th European Congress and Exhibition on Intelligent transport Systems and Services*, Sep. 2013.

[Obj11a] Object Management Group (OMG): *Business Process Model and Notation (Version 2.0)*, Jan. 2011.

[Obj11b] Object Management Group (OMG): *The Common Object Request Broker: Architecture and Specification (Version 3.3)*, Nov. 2011.

[Par72] Parnas, D. L.: *On the Criteria To Be Used in Decomposing Systems into Modules.* Communications of the ACM, 15:1053–1058, 1972.

[PEG20] PEGASUS Konsortium: *PEGASUS Schlussbericht für das Gesamtprojekt*, Feb. 2020. Online erhältlich unter `https://www.pegasusprojekt.de/files/tmpl/pdf/PEGASUS_Abschlussbericht_Gesamtprojekt.PDF`; zuletzt abgerufen am 09.05.2020.

[Per08] Perry, Dewayne E.: *Issues in Architecture Evolution: Using Design Intent in Maintenance and Controlling Dynamic Evolution.* In: Morrison, Ron, Dharini Balasubramaniam und Katrina Falkner (Herausgeber): *Software Architecture: Second European Conference.* Springer Berlin / Heidelberg, Sep. 2008.

[PNW+13] Passchier, Igor, Bart Netten, Harry Wedemeijer, Sander Maas, Coen Van Leeuwen und Peter Paul Schackmann: *DITCM roadside facilities for cooperative systems testing and evaluation.* In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, Seiten 936–942. IEEE, 2013.

[Pv16] Passchier, Igor und Marcel van Sambeek: *Architecture for C-ITS applications in the Netherlands.* Technischer Bericht, DITCM Consortium, März 2016.

[PvvP16] Passchier, Igor, Marcel van Sambeek, Joëlle van den Broek und Paul Potters: *The Dutch C-ITS Reference architecture.* In: *Proceedings of the 11th European Congress and Exhibition on Intelligent transport Systems and Services*, Juni 2016.

[QPIJ10] Quintero, Luis Felipe Herrera, Francisco Maciá Pérez, Virgilio Gilart Iglesias und Diego Marcos Jorquera: *Roadside Unit as a Service Based on a Model for the IT Integration into the ITS.* In: *Proceedings of the 17th World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2010.

[QXZ08] Qin, Zheng, Jian-Kuan Xing und Xiang Zheng: *Software Architecture.* Advanced Topics in Science and Technology in China. Springer Berlin / Heidelberg, Apr. 2008.

[Rap96] Rappaport, Theodore S.: *Wireless Communications: Principles and Practice.* IEEE Press, 1996.

[RBH+16] Reussner, Ralf H., Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer und Klaus Krogmann: *Modeling and simulating software architectures : the Palladio approach.* MIT Press, Okt. 2016.

[RH08] Reussner, Ralf und Wilhelm Hasselbring: *Handbuch der Software-Architektur.* dpunkt, 2008.

[RMK+13] Rondinone, Michele, Julen Maneros, Daniel Krajzewicz, Ramon Bauza, Pasquale Cataldi, Fatma Hrizi, Javier Gozalvez, Vineet Kumar, Matthias Röckl, Lan Lin, Oscar Lazaro, Jérémie Leguay, Jérôme Haerri, Sendoa Vaz, Yoann Lopez, Miguel Sepulcre, Michelle Wetterwald, Robbin Blokpoel und Fabio Cartolano: *ITETRIS: a modular simulation platform for the large scale evaluation of cooperative ITS applications.* Simulation Modelling Practice and Theory, Feb. 2013.

[RNT18] Ruppe, Sten, Ronald Nippold und Jan Trumpold: *SIRENE - Secure and Intelligent Road Emergency Network.* In: *SUMO User Conference 2018*, Mai 2018.

[RWG16] Römer, Michael, Christian Weiss und Steffen Gaenzle: *How Automakers Can Survive the Self-Driving Era.* Technischer Bericht, A. T. Kearney, Juli 2016.

[RWJ+11] Richling, Jan, Matthias Werner, Michael C. Jaeger, Gero Mühl und Hans-Ulrich Heiss: *Autonomie in verteilten IT-Architekturen.* Oldenbourg, 2011.

[Ryl08a] Rylander, David: *Cooperating Systems in the Swedish test site.* Presented at the 8th European Congress and Exhibition on Intelligent transport Systems and Services, Juni 2008.

[Ryl08b] Rylander, David: *Preparing a Cooperative Systems Test Site.* In: *Proceedings of the 8th European Congress and Exhibition on Intelligent transport Systems and Services*, Juni 2008.

[SBV03] Stoermer, Christoph, Felix Bachmann und Chris Verhoef: *SACAM: The Software Architecture Comparison Analysis Method.* Technischer Bericht, Carnegie Mellon University, 2003.

[Sch08] Schoonderwoerd, Ruud: *Process-oriented modeling for SOA: A technique for process decomposition.* IBM Corporation Developer Works, Nov. 2008.

[SF08] Schieben, Anna Maria und Frank Flemisch: *Who is in control? Exploration of transitions of control between driver and an eLane vehicle automation.* In: *Integrierte Sicherheit und Fahrerassistenzsysteme*, Band 2048. VDI, Okt. 2008.

[SFT+11] Stahlmann, Rainer, Andreas Festag, Andrea Tomatis, Ilja Radusch und Francois Fischer: *Starting European Field Tests for CAR-2-X Communication: The DRIVE C2X Framework.* In: *Proceedings of the 18th World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2011.

[SHG+10] Schröder, Mark, Marco Hannibal, Jan Gacnik, Frank Köster, Christian Harms und Tobias Knostmann: *Ein Labor zur modellbasierten Gestaltung interaktiver Assistenz und Automation im Automotive-Umfeld.* In: *AAET 2010 - Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, Feb. 2010.

[sim09a] SIM^{TD}: *Anforderungen der Funktionen an die Gesamtarchitektur.* Technischer Bericht D11.4, sim^{TD} Konsortium, Okt. 2009.

[sim09b] SIM^{TD}: *Konsolidierter Systemarchitekturentwurf.* Technischer Bericht D21.2, sim^{TD} Konsortium, Okt. 2009.

[sim10] SIM^{TD}: *Versuchsplan 1.0.* Technischer Bericht D41.1, sim^{TD} Konsortium, Juni 2010.

[sim11] SIM^{TD} KONSORTIUM: *sim^{TD} / projektprofil.* Broschüre, Mai 2011.

[SNH95] SONI, DILIP, ROBERT L. NORD und CHRISTINE HOFMEISTER: *Software Architecture in Industrial Applications.* In: *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, Seiten 196–207. ACM, Apr. 1995.

[Som16] SOMMERVILLE, IAN: *Software Engineering.* Addison Wesley, 10. Auflage, 2016.

[SSS81] SANDEWALL, E., C. STROMBERG und H. SORENSEN: *Software Architecture Based on Communicating Residential Environments.* In: *Proceedings of 5th International Conference on Software Engineering.* IEEE Computer Society Press, 1981.

[Sta86] STANISLAW, HAROLD: *Tests of computer simulation validity: what do they measure?* Simulation & Games, 17(2):173–191, Juni 1986.

[SW01] SMITH, CONNIE U. und LLOYD G. WILLIAMS: *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison-Wesley Professional, 1. Auflage, Sep. 2001.

[TCY95] TRACZ, WILL, LOU COGLIANESE und PATRICK YOUNG: *Domain-Specific Software Architecture Engineering Process Guidelines ADAGE-IBM-92-02 Version 2.0*, Feb. 1995.

[Tes09] TEST SITE NORWAY: *Test Site Norway*, 2009. Online erhältlich unter `http://www.item.ntnu.no/its-tsn/TSN_rev6.pdf`; zuletzt abgerufen am 02.10.2016.

[Tes11a] TEST SITE SWEDEN: *Test Site Sweden / About TSS*, 2011. Online erhältlich unter `http://www.testsitesweden.com/about-tss`; zuletzt abgerufen am 12.10.2011.

[Tes11b] Test Site Sweden: *Test Site Sweden / R&DDD*, 2011. Online erhältlich unter `http://www.testsitesweden.com/sites/default/files/tss_presentation.ppt`; zuletzt abgerufen am 17.10.2013.

[Thi05] Thiel, Steffen: *A Framework to Improve the Architecture Quality of Software-Intensive Systems*. Dissertation, Universität Duisburg-Essen, Dez. 2005.

[THLK12] Thielen, Daniel, Christian Harms, Tobias Lorenz und Frank Köster: *Communication between Vehicle and Nomadic Device Used for Assistance and Automation*, Jan. 2012.

[VBK10] Vermassen, Erwin, Dirk Beckmann und Frank Köster: *The ITS Test Beds Project*. In: *Proceedings of the 17th World Congress and Exhibition on Intelligent Transport Systems and Services*, Okt. 2010.

[VE03] Vahidi, A. und A. Eskandarian: *Research advances in intelligent collision avoidance and adaptive cruise control*. Intelligent Transportation Systems, IEEE Transactions on, 4(3):143 – 153, Sept. 2003.

[Ver10] Vermassen, Erwin: *Assembling and validating a first version of the ITS test bed*. Technischer Bericht D2.2, ITS Test Beds Consortium, 2010.

[vNHP10] van den Broek, Thijs Hendrikus Adrianus, Bart Netten, Marika Hoedemaeker und Jeroen Ploeg: *The experimental setup of a large field operational test for cooperative driving vehicles at the A270*. In: *Intelligent Transportation Systems (ITSC), 13th International IEEE Annual Conference on Intelligent Transportation Systems*, Sep. 2010.

[vOB+15] van Sambeek, Marcel, Frank Ophelders, Tjerk Bijlsma, Borgert van der Kluit, Oktay Türetken, Rik Eshuis, Kostas Traganos und Paul Grefen: *Towards an Architecture for Cooperative ITS Applications in the Netherlands*, Apr. 2015.

[WDS12] Winner, Hermann, Bernd Danner und Joachim Steinle: *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Vieweg+Teubner, 2 Auflage, 2012.

[WS05] Williams, Lloyd G. und Connie U. Smith: *QSEM: Quantitative Scalability Evaluation Method*. In: *31st International Computer Measurement Group Conference*. Computer Measurement Group, Dez. 2005.

[ZSO+05] Zang, Yunpeng, Lothar Stibor, Georgios Orfanos, Shumin Guo und Hans-Juergen Reumerman: *An error model for inter-vehicle communications in highway scenarios at 5.9GHz.* In: *Proceedings of the 2nd ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks.* ACM, Okt. 2005.

Anhang

Tabelle A.1: Testfeldrelevante Aktivitäten im Testprozess, verdichtet aus [FES17]

Aktivität	**Unteraktivität**
Select and obtain Vehicles	Procure Vehicles
	Sign off Vehicles
Select and obtain systems	Procure Data handling systems
	Sign off handling systems
Select and obtain data collection & transfer systems	Procure Data handling systems
	Sign off Data handling systems
Select and obtain support systems	Procure Support Systems
	Sign off Support Systems
Equip vehicles with technologies	Prepare a system installation & integration specification
	Equip vehicles w. FOT technologies to be evaluated (if not already in vehicles)
	Equip vehicles w. data collection & transfer systems
	Equip vehicles w. FOT support systems
	Sign off on system integration activities
Implement driver feedback & reporting systems	Install systems & procedures to report current technical problems
	Install systems & procedures to report technical problems offline
	Install systems & procedures to monitor driver
	Sign off feedback systems
Select / implement database for storing data	Adopt data storage mechanisms

(Fortsetzung Tabelle A.1)

Aktivität	**Unteraktivität**
	Prepare Data storage
	Archive Data
Test all systems to be used according to specifications	Test acceptance
	Conduct usability testing
	Deploy least amount of vehicle
	Fine tune vehicles and technologies
	Manage participants
Develop recruitment strategy & materials	Manage participants
Develop driver training & briefing materials	Manage participants
Pilot Test equipment, methods & procedures	Deploy a small sample of vehicles
	Fine tune vehicles and technologies
Run the FOT	Deploy vehicles fleet
	Manage Data Collection
	Deploy vehicles fleet
	Collect additional (offline) data
	Handle user feedback
	React on problems
	Commence preliminary evaluation
	Maintain fleet
	Maintain instrumentation
	Detect obstructive behaviour
	Conduct interviews
Analyse data	Analyse objective data
	Analyse subjective data
Write minutes and reports	Report test setup
	Report test pattern
	Report test documentation
Disseminate findings	Report findings
	Report conclusion

(Fortsetzung Tabelle A.1)

Aktivität	**Unteraktivität**
	Showcase FOT
Decommission study	Debrief team
	Debrief participants

Tabelle A.2: Betrachtete Testfelder

Forschung und Methodik

AIM	*Anwendungsplattform Intelligente Mobilität*, Testfeld in Braunschweig
DITCM	*Dutch Integrated Testsite Cooperative Mobility*, Testfeld im Raum Helmond
simTD	*Sichere Intelligente Mobilität – Testfeld Deutschland*, Testfeld im Rhein-Main-Gebiet
TSN	*Test Site Norway*, Testfeld in Trondheim, Norwegen
TSS	*Test Site Sweden*, Testfeld mit NDS-Ausrichtung, Raum Göteborg, Schweden

Pilotversuche

Co-Gistics	Erprobt kooperative ITS-Technologien im Logistiksektor.
Compass4D	Erprobt Kreuzungsassistenzen und Gefahrenwarnung auf Basis von kooperativen ITS-Technologien
COSMO	*Cooperative Systems for Sustainable Mobility and Energy Efficiency* Erprobt Assistenzen zur Reduktion von Emissionen im Feld.
FREILOT	Erprobt Systeme zur Emissionsreduktion von Nutzfahrzeugen in Stadtgebieten.
HeERO	*Harmonized eCall European Pilot* Felderprobung der eCall-Assistenz (automatischer Notruf bei Unfall).
Roadwise	Feldversuch zur Erprobung von Verkehrsinformationsanzeigen im Fahrzeug sowie Infotainmentaspekten.

Forschungsprojekte und Aktivitäten um Testfelder

Aktiv	*Adaptive und kooperative Technologien für den intelligenten Verkehr* Entwickelt u. a. neue Fahrerassistenzsysteme und ein effizientes Verkehrsmanagement zu CO_2-Reduktion.
AutoNet2030	Entwickelt hohe Automatisierung auf Basis kooperativer ITS-Technologien
COOPERS	*COOPERative Systems for intelligent road safety* Entwickelt intelligente Verkehrsmanagmentsysteme über variable Verkehrszeichen und Anzeigen im Fahrzeug.
FESTA	*Field opErational teSt supporT Action* Entwickelt Methoden und Richtlinien zur Durchführung von Feldtests.
ITS Test Beds	Entwickelt eine Referenzarchitektur für Testfelder und testfeldüberspannende Versuchsaufbauten.
PRE-DRIVE C2X	Entwickelt grundlegende Konzepte und technische Grundlagen für das Projekt DRIVE C2X.
PRESERVE	Fokussiert Sicherheitsaspekte der V2X-Kommunikation.
Safespot	Entwickelt kooperative Assistenzsysteme, die auf eine Erhöhung der Sicherheit abzielen.
UDrive	groß angelegte Naturalistic Driving Study zum normativen Verhalten von Verkehrsteilnehmern
UR:BAN	Entwickelt Fahrerassistenz- und Verkehrsmanagementsysteme im Stadtraum.

(Fortsetzung Tabelle A.2)

Feldtests

ARTRAC	Felderprobung von Radartechnologien zum Schutz von besonders gefährdeten Verkehrsteilnehmern wie bspw. Radfahrern oder Fußgängern
AOS	niederländische Initiative zur Kollisionsvermeidung bei Nutzfahrzeugen.
Belonitor	Konzept zu positiven Konditionierung des Fahrers zu einer effizienterer Fahrweise.
COSAL	Fahrerassistenz zur Einhaltung von Geschwindigkeitsbegrenzungen, vgl. LAVIA
DIAMANT	*Demonstration of ITS Applications through Multimedia Over Networks using DAB* Felderprobung von Benachrichtigungssystemen für Fahrer
DRIVE C2X	Felderprobung ausgewählter kooperativer Assistenzsysteme in sechs Testfeldern
ecoDriver	Felderprobung von Fahrerassistenz zur Reduktion des Kraftstoffverbrauchs insbesondere bei LKW
eCoMove	Felderprobung von Fahrerassistenz zur Reduktion des Kraftstoffverbrauchs um bis zu 20 %
euroFOT	Felderprobung ausgewählter ITS Assistenzsysteme in mehreren Testfeldern
FOTsis	*Field Operational Test on Safe, Intelligent and Sustainable Road Operation* Erprobt marktreife Systeme zum Management von stationärer Infrastruktur (z. B. Verkehrszeichen, Maut) im Feld.
INFATI	Erprobt ein Assistenzsystem zur Einhaltung der Geschwindigkeitsbegrenzung im Feldtest.
ISA Trials	Feldtests in verschiedenen Europäischen Ländern; ähnlicher Anwendungsfall wie INFATI
LAVIA	*Limiteur s'Adaptant à la VItesse Autorisée* Erprobung von intelligenten Geschwindigkeitsanzeigen im Fahrzeug; ähnlich INFATI
Pay As You Speed	Erprobt die Wirkung von konditionierenden Assistenzsystemen im Feldtest, ähnlich Belonitor
SCORE@F	*Système Coopératif Routier Expérimental Français* Erprobt ausgewählte kooperative Systeme im Feldtest.
TeleFOT	Fokussiert die Effekte auf den Fahrer durch (zusätzliche) Assistenzsysteme im Feldtest.

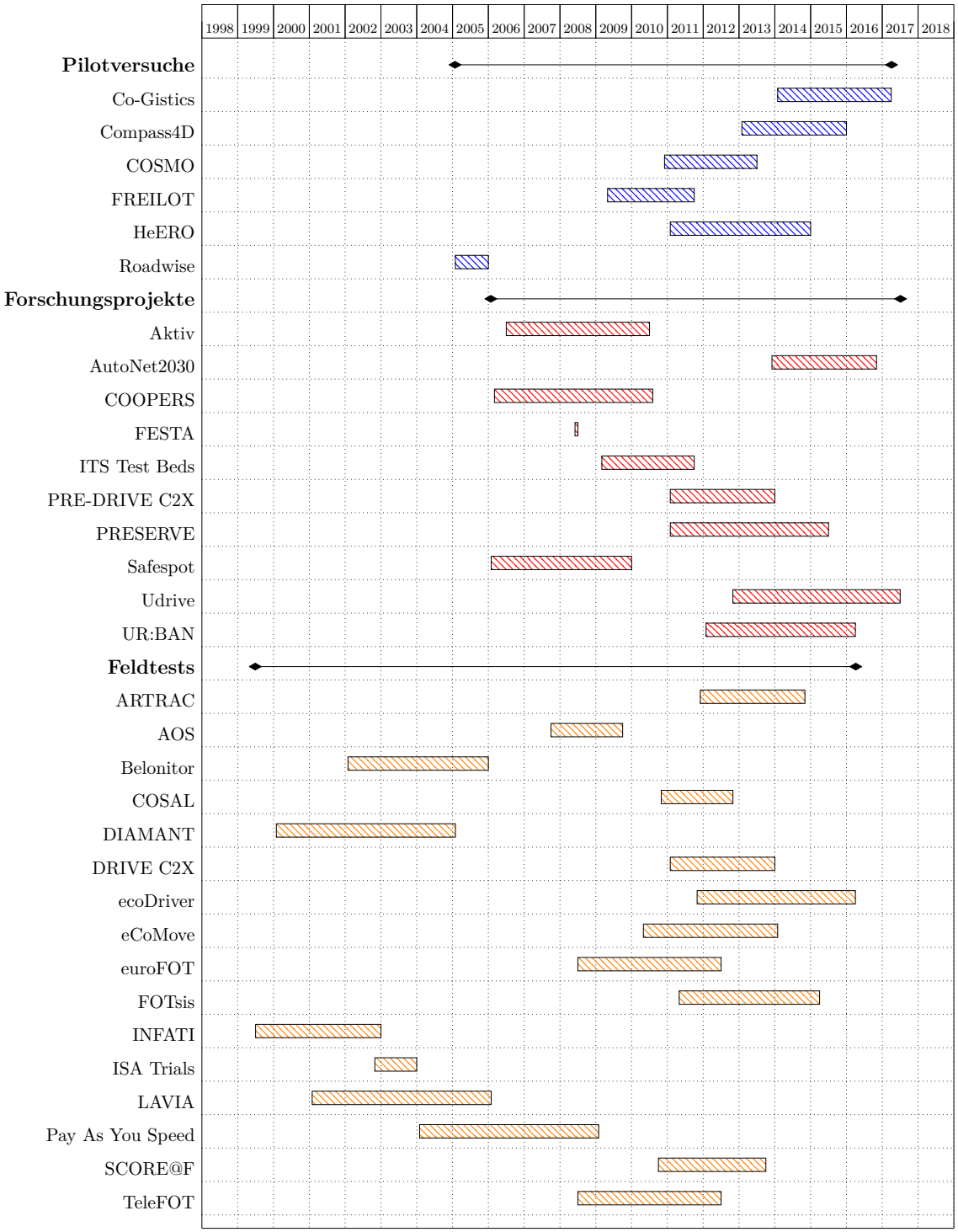

Abbildung A.1: Zeitliche Darstellung der Pilotversuche, Forschungsprojekte und Feldtests aus Tab. A.2

Tabelle A.3: Anwendungsbeispiele von ITS nach ETSI (aus [ETS09])

Anwendungsklasse	**Applikation**	**Anwendungsfall**
Active road safety	**Driving assistance – Co-operative awareness**	Emergency vehicle warning
		Slow vehicle indication
		Intersection collision warning
		Motorcycle approaching indication
	Driving assistance – Road Hazard Warning	Emergency electronic brake lights
		Wrong way driving warning
		Stationary vehicle - accident
		Stationary vehicle - vehicle problem
		Traffic condition warning
		Signal violation warning
		Roadwork warning
		Collision risk warning
		Decentralized floating car data – Hazardous location
		Decentralized floating car data – Precipitations
		Decentralized floating car data – Road adhesion
		Decentralized floating car data – Visibility
		Decentralized floating car data – Wind
Cooperative traffic efficiency	**Speed management**	Regulatory / contextual speed limits notification
		Traffic light optimal speed advisory
	Co-operative navigation	Traffic information and recommended itinerary
		Enhanced route guidance and navigation
		Limited access warning and detour notification
		In-vehicle signage

(Fortsetzung Tabelle A.3)

Anwendungsklasse	Applikation	Anwendungsfall
Co-operative local services	**Location based services**	Point of Interest notification
		Automatic access control and parking management
		ITS local electronic commerce
		Media downloading
Global internet services	**Communities services**	Insurance and financial services
		Fleet management
		Loading zone management
	ITS station life cycle management	Vehicle software / data provisioning and update
		Vehicle and RSU data calibration

Tabelle A.4: Kernkompetenzen in der Planung und Durchführung von Feldversuchen

Prozessschritt	Technik	Wissenschaft	Organisation	Anwendungsdomäne
Function Identification and description	—	Spezifikation des Zielsystems	—	Spezifikation des Zielsystems
Use Cases	Identifikation kritische Systemkomponenten	Identifikation aussagekräftiger Szenarien	—	—
Research Questions and Hypotheses	Technische Machbarkeit	Definition der Forschungsfrage	—	—
Performance Indicators	Technische Machbarkeit	Definition von Kennzahlen / Methodik	—	—
Study design	Aufbau der Versuchslandschaft	Definition der Versuchsdurchführung	Planung des Versuchs	—
Measures and Sensors	Einrüstung der Sensorik	—	Planung des Versuchs	—
Data Acquisition	Durchführung des Versuchs	—	Durchführung des Versuchs	—
Database	Datenverarbeitung	—	Bereitstellung der Daten, Ermittlung der Kennzahlen	—
Data Analysis	—	Analyse	—	—
Research Question and Hypotheses Analysis	—	Auswertung	—	Reflexion der Ergebnisse
System and Function Analysis	—	Auswertung	—	Reflexion der Ergebnisse
Socio-Economic Impact Assessment	—	Analyse, Auswertung	—	Reflexion der Ergebnisse

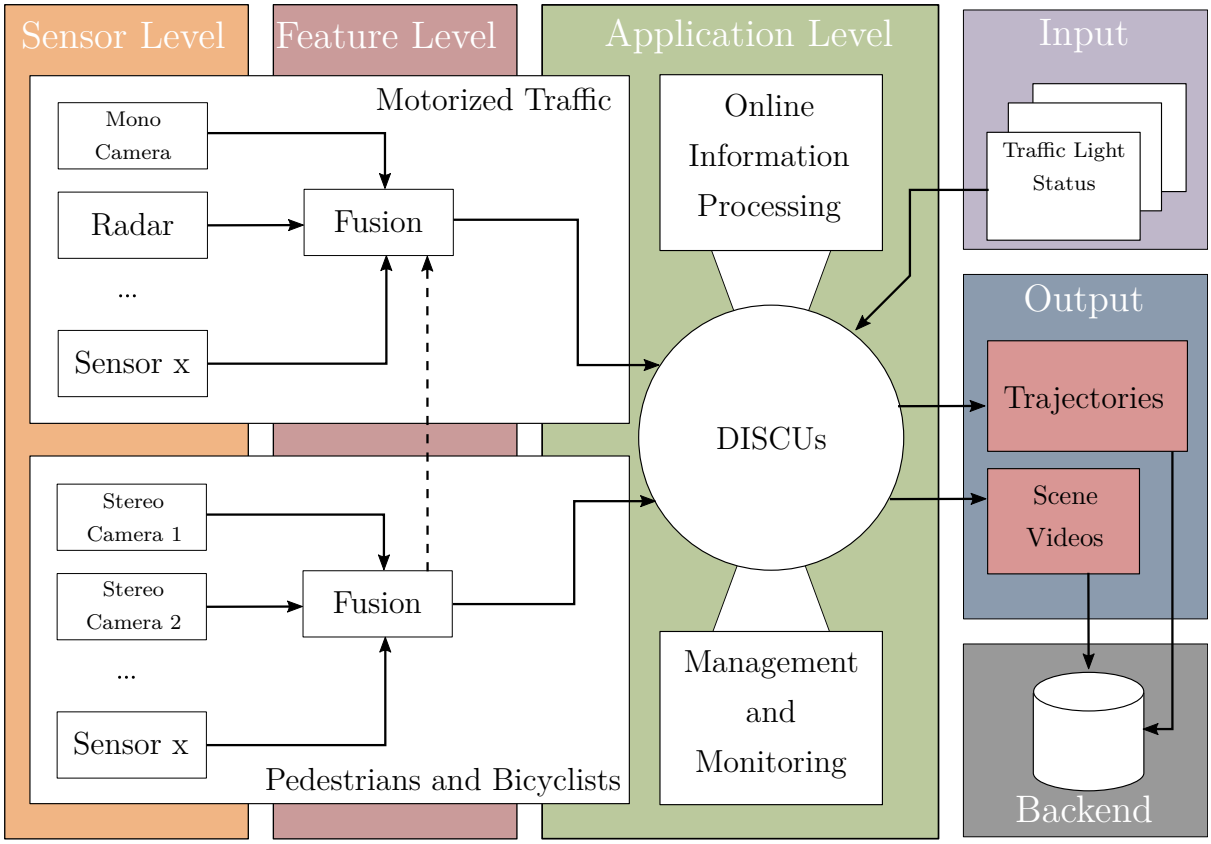

Abbildung A.2: Architekturdarstellung der AIM-Forschungskreuzung aus [KLG16, S. 2]

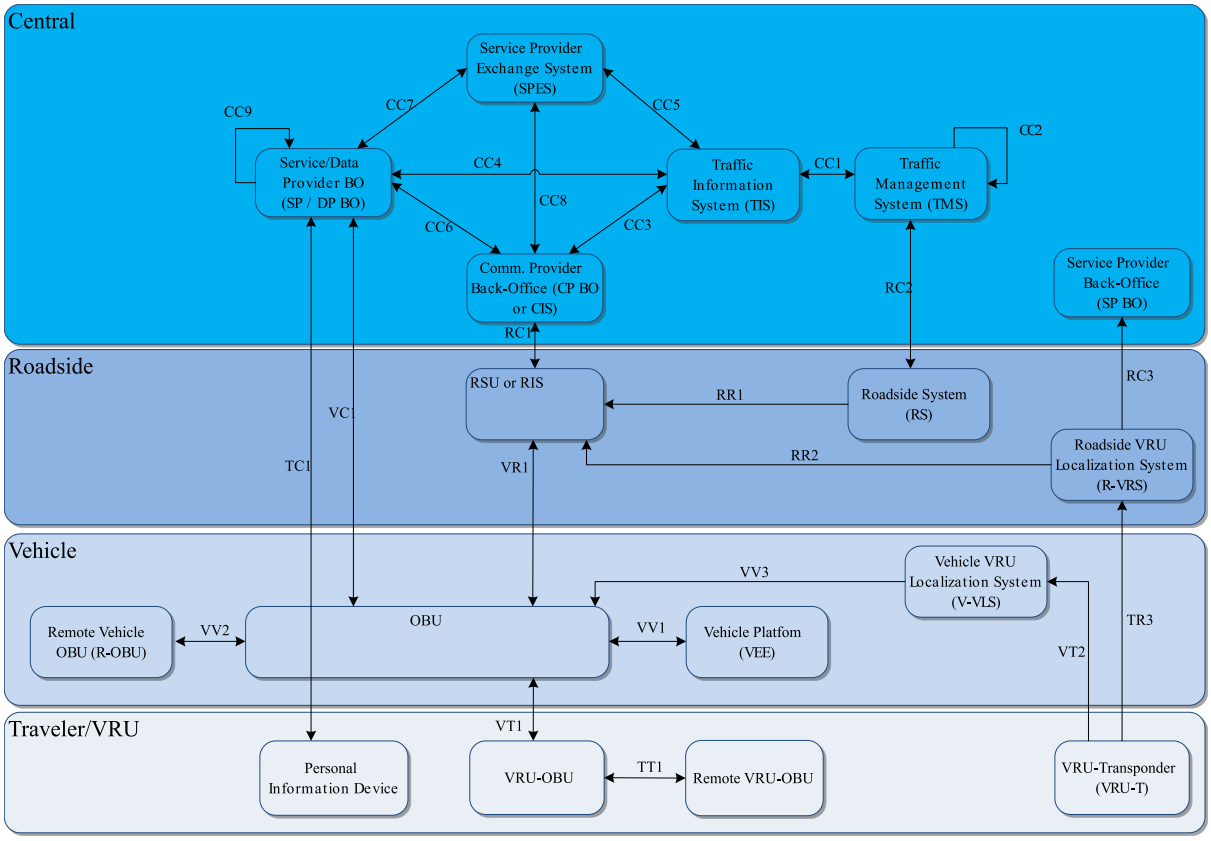

Abbildung A.3: DITCM-Architekturdarstellung, physikalische Sicht mit Subsystemen und Schnittstellen, aus [Pv16, S. 12]

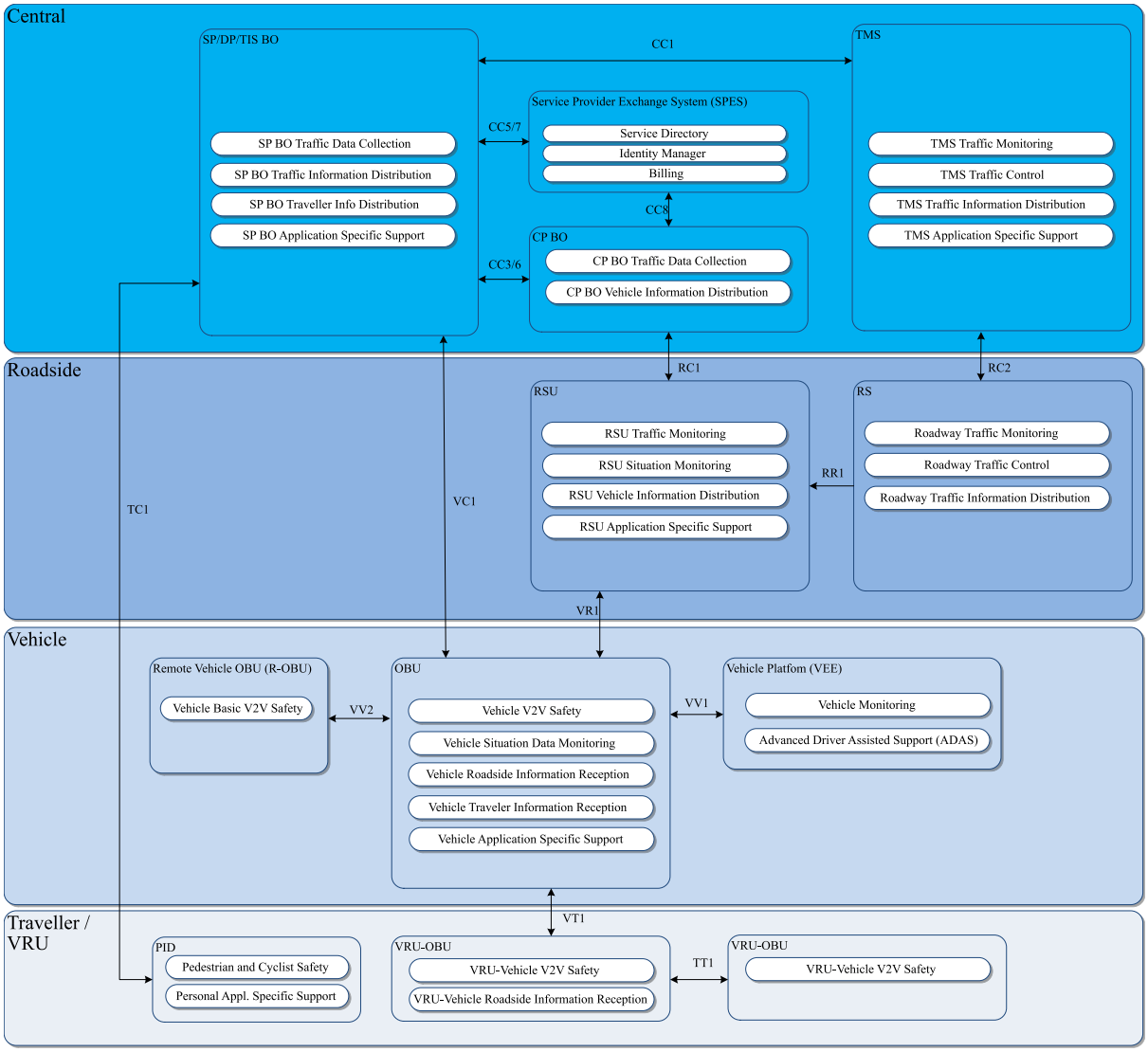

Abbildung A.4: DITCM-Architekturdarstellung, funktionale Sicht mit Subsystem, funktionalen Komponenten und Schnittstellen aus [Pv16, S. 13]

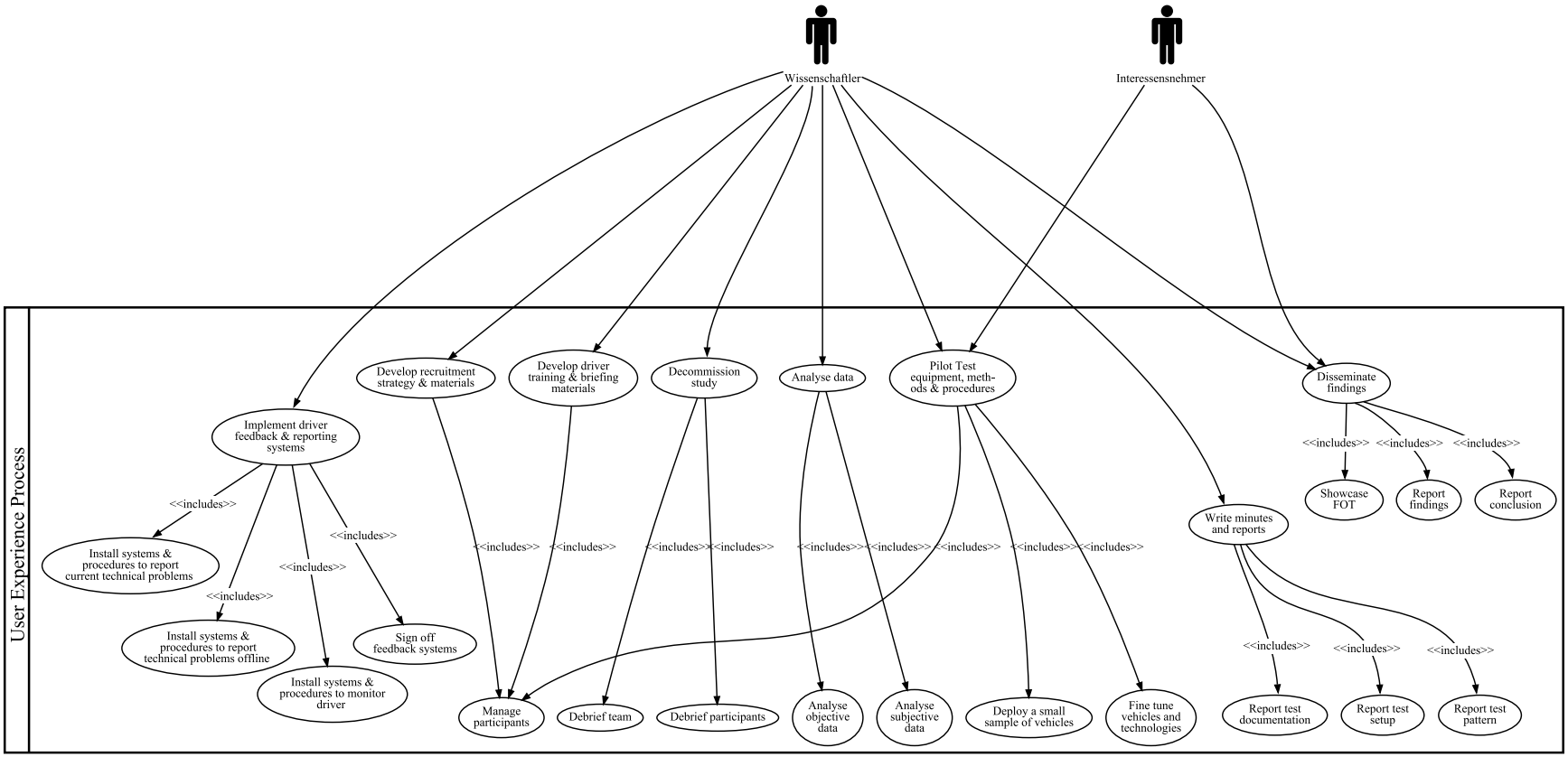

Abbildung A.5: Auszug aus der Use Case Modellierung abstrakter Aktivitäten

Tabelle A.5: Verwendete Skala in der Bewertung von Wichtigkeit und Realisierungsaufwand

Symbol	Wichtigkeit	Realisierungsaufwand	Gewichtungsfunktion ($\mathfrak{b}(\mathfrak{n})$)
H	hochwichtig	hoch	$\mathfrak{b}(H) = 5$
M	mittelmäßig wichtig	mittel	$\mathfrak{b}(M) = 2$
L	wenig wichtig	niedrig	$\mathfrak{b}(L) = 1$

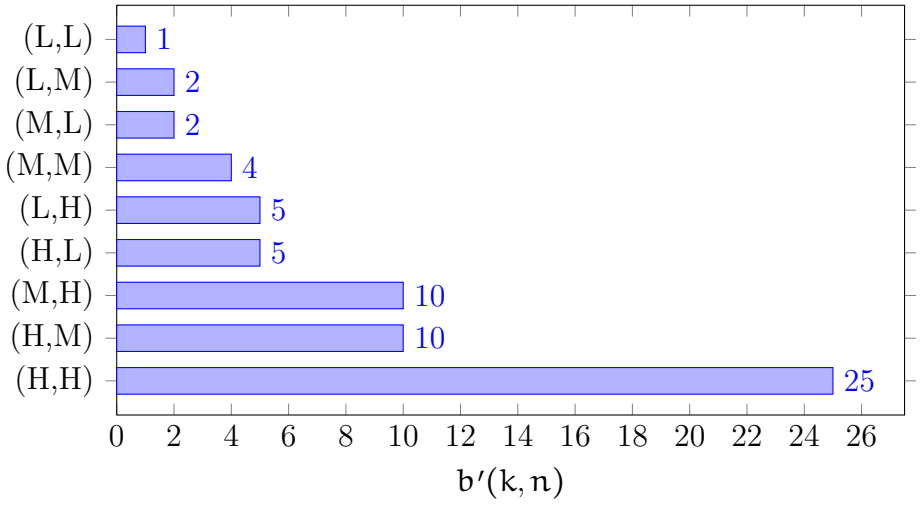

Abbildung A.6: Funktionswerte von $\mathfrak{b}'$ für verschiedene Bewertungstupel

Tabelle A.6: Numerische Darstellung der Architekturbewertung und Verdichtung über arithmetische Mittel

Qualitätsmerkmal	**Nr.**	**Priorität** k	**Realisierungs-aufwand** n	$b(n)$	$b'(k,n)$	$\overline{b}$, $\overline{b'}$
Interoperabilität	1	H	L	1	5	$\overline{b}_{1...3} = 1$ $\overline{b'}_{1...3} = 5$
	2	H	L	1	5	
	3	H	L	1	5	
Austauschbarkeit	4	H	L	1	5	$\overline{b}_{4...8} = 1$ $\overline{b'}_{4...8} = 5$
	5	H	L	1	5	
	6	H	L	1	5	
	7	H	L	1	5	
	8	H	L	1	5	
Analysierbarkeit	9	H	L	1	5	$\overline{b}_{9...12} = 3,25$ $\overline{b'}_{9...12} = 4,75$
	10	L	H	5	5	
	11	M	M	2	4	
	12	L	H	5	5	
Änderbarkeit	13	H	L	1	5	$\overline{b}_{13...14} = 1$ $\overline{b'}_{13...14} = 5$
	14	H	L	1	5	
Koexistenz	15	H	L	1	5	$\overline{b}_{15...16} = 1$ $\overline{b'}_{15...16} = 5$
	16	H	L	1	5	

Tabelle A.7: Numerische Darstellung der Architekturbewertung von sim^{TD}, analog zu Tab. A.6

Qualitätsmerkmal	**Nr.**	**Priorität** k	**Realisierungs-aufwand** n	$b(n)$	$b'(k,n)$	$\overline{b}$, $\overline{b'}$
Interoperabilität	1	H	L	1	5	$\overline{b}_{1...3} = 1,\overline{6}$ $\overline{b'}_{1...3} = 8,\overline{3}$
	2	H	M	2	10	
	3	H	M	2	10	
Austauschbarkeit	4	H	L	1	5	$\overline{b}_{4...8} = 1,8$ $\overline{b'}_{4...8} = 9$
	5	H	M	2	10	
	6	H	M	2	10	
	7	H	M	2	10	
	8	H	M	2	10	
Analysierbarkeit	9	H	L	1	5	$\overline{b}_{9...12} = 2,5$ $\overline{b'}_{9...12} = 4$
	10	L	M	2	2	
	11	M	M	2	4	
	12	L	H	5	5	
Änderbarkeit	13	H	M	2	10	$\overline{b}_{13...14} = 1,5$ $\overline{b'}_{13...14} = 7,5$
	14	H	L	1	5	
Koexistenz	15	H	M	2	10	$\overline{b}_{15...16} = 1,5$ $\overline{b'}_{15...16} = 7,5$
	16	H	L	1	5	

Tabelle A.8: Numerische Darstellung der Architekturbewertung von ITS Test Beds, analog zu Tab. A.6

Qualitätsmerkmal	**Nr.**	**Priorität** k	**Realisierungs-aufwand** n	$b(n)$	$b'(k,n)$	$\overline{b}$, $\overline{b'}$
Interoperabilität	1	H	L	1	5	$\overline{b}_{1...3} = 1,\overline{3}$ $\overline{b'}_{1...3} = 6,\overline{6}$
	2	H	L	1	5	
	3	H	M	2	10	
Austauschbarkeit	4	H	L	1	5	$\overline{b}_{4...8} = 1,4$ $\overline{b'}_{4...8} = 7$
	5	H	L	1	5	
	6	H	M	1	5	
	7	H	M	1	5	
	8	H	L	1	5	
Analysierbarkeit	9	H	H	5	25	$\overline{b}_{9...12} = 5$ $\overline{b'}_{9...12} = 11,25$
	10	L	H	5	5	
	11	M	H	5	10	
	12	L	H	5	5	
Änderbarkeit	13	H	L	1	5	$\overline{b}_{13...14} = 1,5$ $\overline{b'}_{13...14} = 7,5$
	14	H	M	2	10	
Koexistenz	15	H	M	2	10	$\overline{b}_{15...16} = 2$ $\overline{b'}_{15...16} = 10$
	16	H	M	2	10	

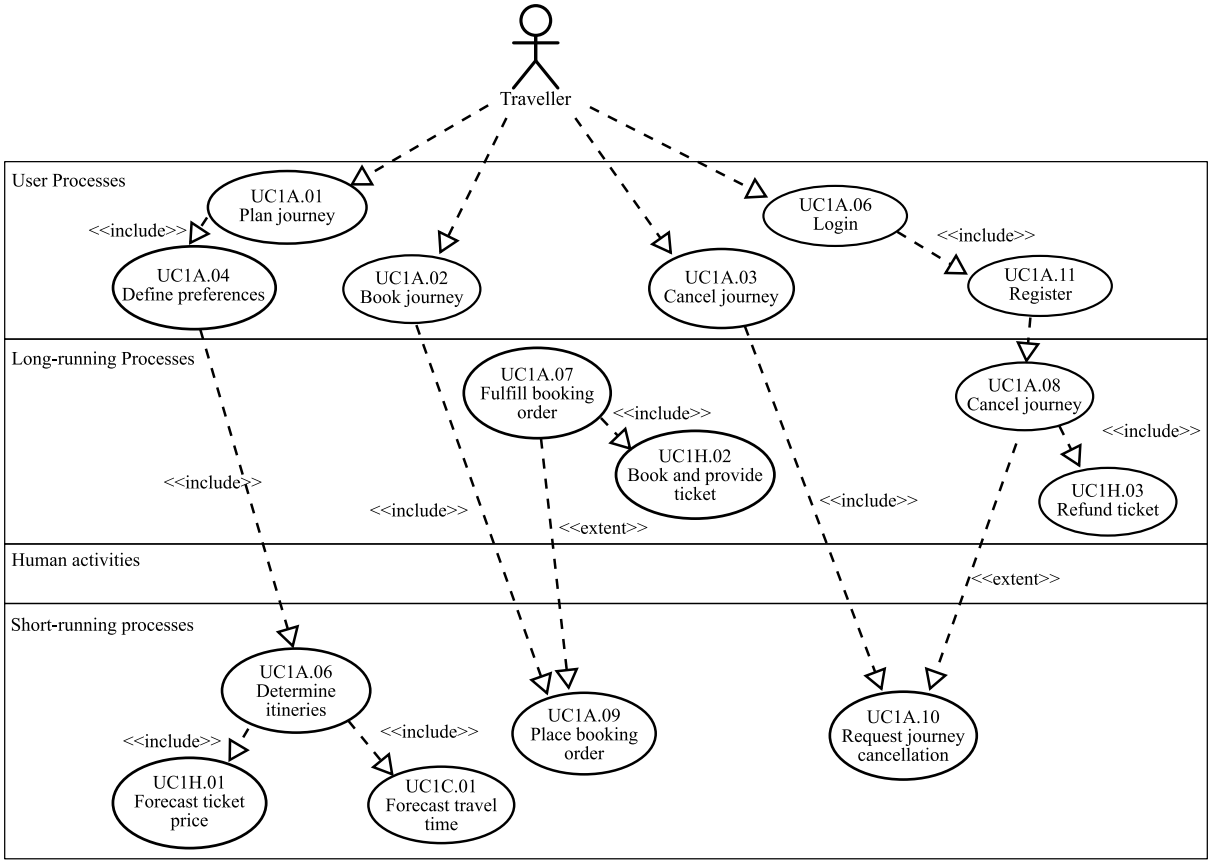

Abbildung A.7: UML-Beispiel *end-to-end itinery planning* aus [BST11, S. 20]

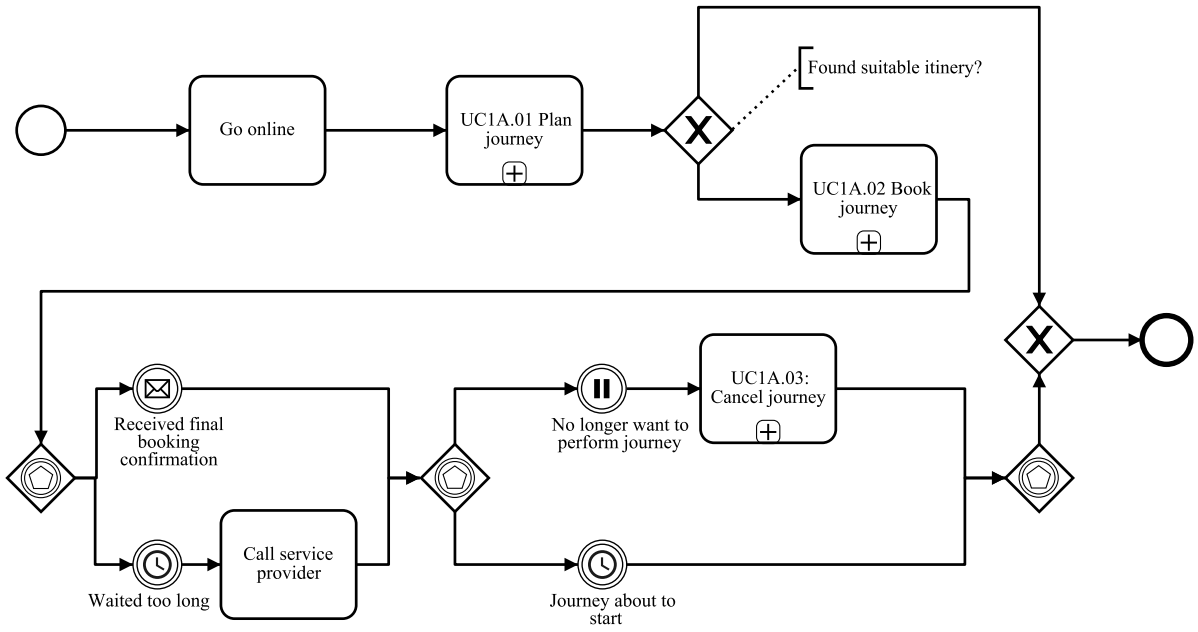

Abbildung A.8: BPMN-Beispiel *Pre-trip journey planning* aus [BST11, S. 18]

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichungen, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 6. Oktober 2023.

_____

Dirk Beckmann