

Fakultät für Informatik und Mathematik
der Hochschule für angewandte Wissenschaften München

Abschlussarbeit

im Studiengang Informatik
zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

SMART WORKSPACES ZUR PROJEKT-KOLLABORATION

vorgelegt von

Dennis Schock

geboren am 06.08.1987 in Heilbronn

Matrikelnummer: 20426316

vorgelegt am

15.10.2018

Erstprüfer:

Prof. Dr. Gudrun Socher

Zweitprüfer:

Prof. Dr. Veronika Thurner

Hochschule für angewandte
Wissenschaften München
Lothstraße 64
80335 München



Betrieblicher Betreuer:

Dipl.-Inf. Wilhelm Haas

QAware GmbH
Aschauer Straße 32
81549 München



(Diese Seite wurde absichtlich leer gelassen)

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Abschlussarbeit selbstständig und nur unter Verwendung der von mir angegebenen Quellen und Hilfsmittel verfasst zu haben. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit hat in dieser oder vergleichbarer Form noch keinem anderem Prüfungsgremium vorgelegen.

Datum, Ort

Unterschrift

Danksagung

Ich möchte an dieser Stelle all jenen danken, die mich im Rahmen dieser Abschlussarbeit fachlich und persönlich unterstützt und somit zu ihrem Gelingen beigetragen haben.

Insbesondere möchte ich Herrn Dipl.-Inf. Wilhelm Haas meinen Dank dafür aussprechen, dass er mir mit seiner langjährigen Erfahrung stets beratend zur Seite stand. Des Weiteren gilt mein Dank Frau Prof. Dr. Gudrun Socher, welche die Erstbetreuung meiner Abschlussarbeit übernommen hat und deren Anregungen und Ratschläge eine große Hilfe für mich waren.

Zutiefst zu Dank verpflichtet bin ich auch meinen Eltern Emil und Lilli, die mir mein Studium überhaupt erst ermöglicht und mich in all meinen Entscheidungen unterstützt haben. Besonderer Dank gebührt außerdem meiner Lebensgefährtin Sarah für ihre Geduld, Ausdauer und den Rückhalt, den sie mir während meiner gesamten Studienzeit gegeben hat und weiterhin gibt. Schließlich möchte ich mich auch bei meinen Kommilitonen und Kollegen bedanken, die mich während dieser Zeit unterstützt und begleitet haben.

Fürstenfeldbruck, im Herbst 2018

Dennis Schock

Zusammenfassung

Die vorliegende Abschlussarbeit untersucht wie Arbeitsumfelder durch den Einsatz moderner Soft- und Hardware-Technologien smarter gestaltet werden können. Im Zuge dessen wurde eine Reihe von Systemen und aktuellen Entwicklungen aus diesem Bereich untersucht und vorgestellt. Der Hauptteil der Arbeit beschäftigt sich mit der Ausarbeitung von Konzept sowie Anwendungsfällen für ein neues derartiges System und dessen prototypischer Implementierung. Dieses System, welches als *Smart Lab* bezeichnet wird, dient zur Bereitstellung von automatisierten Assistenz-Funktionen, z.B. während Meetings. Die Schwerpunkte des Systems, etwa die Bedienung nach dem Prinzip von *Fire-and-Forget* und der quelloffene Code, wurden dabei so gesetzt, dass es eine Nische zwischen den zuvor vorgestellten kommerziellen Entwicklungen ausfüllt. Für die Bewertung der Nützlichkeit und Benutzbarkeit von Smart Lab wurde abschließend eine qualitative Evaluierung durchgeführt. Die dabei ermittelten Resultate lassen darauf schließen, dass die erste Iteration des Systems als Basis für Weiterentwicklungen geeignet ist. Für den Produktiveinsatz sind jedoch zunächst Erweiterungen in Bereichen wie der Systemsicherheit und der Benutzbarkeit erforderlich.

Inhaltsverzeichnis

Eidesstattliche Erklärung	iii
Danksagung	iv
Zusammenfassung	v
1. Einleitung	1
1.1. Zielsetzung und Fragestellung	1
1.2. Gliederung der Arbeit	2
2. Aktuelle Entwicklungen	3
2.1. Alexa for Business	3
2.2. Cortana	5
2.3. Interaktive Whiteboards	7
3. Konzept	9
3.1. Produktvision	10
3.2. Arten der Konfiguration	11
3.3. Definition von <i>smart</i>	12
3.4. Schwerpunkte des Projekts	13
3.5. Der Assistance-Kontext	19
3.6. Trigger	22
3.7. Assurances	24
3.8. Actions	35
3.9. Aktoen	38
3.10. Interaktion mit dem System	39
3.11. Ablauf der Systemlogik	41
3.12. Architektur	42
3.13. API	45
4. Implementierung	48
4.1. Umsetzung der Services	48
4.2. Modulstruktur	50

4.3. Konfigurationsverwaltung	53
4.4. Kommunikation zwischen Services	53
4.5. Datenmanagement-Services	56
4.6. Konfigurationssprache für Events	58
4.7. Assurances	61
4.8. Actions	63
4.9. Akteure	67
4.10. Grafische Benutzeroberfläche	70
4.11. Manipulation von Programmfenstern	71
4.12. Erweiterung von Smart Lab	74
5. Ergebnisse	77
5.1. Inbetriebnahme von Smart Lab	77
5.2. Evaluierung	81
6. Fazit und Ausblick	86
A. HTTP-Endpunkte der verschiedenen Services von Smart Lab	91
B. Klassenhierarchien der verschiedenen Kategorien von Aktor-Adaptoren	98
C. Konfiguration über ECL	104
Quellenverzeichnis	105
Abbildungsverzeichnis	110
Tabellenverzeichnis	114
Listingverzeichnis	115
Abkürzungsverzeichnis	116

1. Einleitung

Systeme zur Heimautomation und digitale Assistenten sind heutzutage fest im Verbrauchermarkt etabliert. Während erstere bereits seit einigen Jahren verfügbar sind, haben letztere durch den Fortschritt der vergangenen Jahre in den Bereichen der KI (Künstliche Intelligenz) und Computerlinguistik Entwicklungssprünge vollführt. Das Versprechen der Hersteller solcher Systeme ist es, dass ein Umfeld mit ihnen smarter gestaltet werden kann. Das heißt in der Regel, dass entsprechende Aufgaben automatisiert werden können oder Tätigkeit, bei denen dies nicht vollständig möglich ist, zumindest so umfunktioniert werden können, dass sie effizienter, leichter oder bequemer sind.

Entsprechende Systeme für Arbeitsumfelder, d.h. Umgebungen, in denen Menschen kollaborativ arbeiten wie z.B. Unternehmen oder Hochschulen, waren jedoch in der Vergangenheit in ihrer Funktionalität limitiert. Umfassende Lösungen, die alle funktionellen Aspekte solcher Umfelder integrieren und sich in ihre Geschäftsprozesse einfügen können, existierten bis vor kurzem nicht. Einerseits waren digitale Assistenten oft einer bestimmten Software zugeordnet und in ihrem Nutzen, welcher häufig lediglich virtueller Natur sein konnte, auf diese beschränkt. Andererseits waren Systeme zur Automatisierung und Manipulation der physischen Welt nicht genug in die Geschäftsprozesse eines Arbeitsumfelds integriert und damit in ihrer *Smartness* eingeschränkt. Zusätzlich war gerade die kollaborative Arbeit, die in der heutigen Arbeitswelt wichtiger denn je ist, ein vernachlässigter Fokus solcher Systeme.

Dieser Umstand beginnt sich jedoch zu ändern. Mittlerweile haben Systeme, die teilweise eigentlich aus dem Endverbrauchermarkt stammen und sich dort bewährt haben, Einzug in die Arbeitswelt gehalten, um diese smarter zu gestalten.

1.1. Zielsetzung und Fragestellung

Die vorliegende Abschlussarbeit verfolgt zwei Ziele. Sie soll einerseits einen Überblick über eine Auswahl von Assistenzsystemen für den Einsatz in Arbeitsumfeldern liefern, welche sich aktuell in der Entwicklung befinden oder unlängst erschienen sind. Andererseits beschreibt der Hauptteil der Arbeit die Konzipierung und Implementierung eines eigenen Systems dieser Art. Das System füllt dabei eine Nische aus, welche die zuvor vorgestellten Lösungen nicht abdecken und fokussiert sich auf zeitlich beschränkte Ereignisse wie Meetings als Einsatzszenario. Daneben wurde noch eine Reihe von weiteren Schwerpunkten

verfolgt, die in Abschnitt 3.4 aufgeführt sind.

Das implementierte System dient einerseits als eigenständiger funktionstüchtiger Prototyp, über den das ausgearbeitete Konzept demonstriert werden kann. Andererseits ist es möglich, das System als Basis für Erweiterungen zu verwenden, welche bestehende Technologien wie z.B. Sprachassistenten und Gebäudeautomationssysteme funktionell einbinden. Eine weitere berücksichtigte Zielsetzung ist außerdem die leichte Erweiterbarkeit des Systems um neue Funktionalität, was in Abschnitt 4.12 beschrieben ist.

Die zentrale Fragestellung der Arbeit lautet somit, welche Anforderungen die Projekt-Kollaboration der heutigen Arbeitswelt an ein solches System in Bezug auf die erforderliche Systemarchitektur sowie die umzusetzenden Anwendungsfälle stellt. Zudem sollen die identifizierten Anwendungsfälle als Proof of Concept durch eine prototypische Implementierung abgedeckt werden, welche moderne Technologien (sei es Software oder Hardware) einsetzt und orchestriert.

1.2. Gliederung der Arbeit

Im Anschluss an diese Einleitung folgt in Kapitel 2 die Vorstellung einer Auswahl von aktuellen Entwicklungen aus dem Bereich der smarten Systeme für Arbeitsumfelder. Der darauf folgende Hauptteil dieses Dokuments ist in zwei Teile gegliedert. Der erste Part ist in Kapitel 3 enthalten und erläutert das Konzept des im Rahmen dieser Abschlussarbeit implementierten Systems im Detail. Nach der Klärung aller Begriffe und Prinzipien folgt in Kapitel 4 der zweite Teil, welcher die Umsetzung des Systems schildert. Neben den verwendeten Technologien wird dort auch detailliert die Systemstruktur beleuchtet. Kapitel 5 ist dem Endresultat der Arbeit gewidmet und umfasst eine Beschreibung der Inbetriebnahme des implementierten Systems und dessen Evaluierung. Dieses Dokument schließt letztendlich mit Kapitel 6, in dem ein Fazit über die gesamte Abschlussarbeit und ein Ausblick auf mögliche zukünftige Erweiterungen des realisierten Systems gegeben wird.

2. Aktuelle Entwicklungen

In den folgenden Abschnitten wird eine Auswahl an aktuellen Entwicklungen und Technologien vorgestellt, welche den Zweck haben, Arbeitsumfelder und die Projekt-Kollaboration smarter zu gestalten. Die Liste ist dabei jedoch keinesfalls vollständig, da dies weder Anspruch noch Aufgabe dieser Abschlussarbeit war.

2.1. Alexa for Business

Der Sprachassistent *Alexa* der Firma Amazon hat sich in den letzten Jahren in vielen Heimen von Endverbrauchern etabliert. Dort wird Alexa etwa zum Ansteuern von anderen smarten Heimgeräten, Erledigen von webbasierten Aufgaben, als Informationsquelle oder für andere Zwecke eingesetzt. Der Funktionsumfang von Alexa kann über sogenannte *Skills* erweitert werden. Diese sind im Grunde sprachgesteuerte Anwendungen, welche es dem Assistenten ermöglichen, jeweils bestimmte Aufgaben zu erledigen. Die Interaktion mit Alexa findet hauptsächlich über auditive natürliche Sprache statt, die Seitens Amazon entsprechend verarbeitet wird. Für die Aufnahme der Sprachbefehle können z.B. die Geräte der Echo-Reihe von Amazon selbst verwendet werden, welche eine Kombination aus Lautsprecher und mehreren Mikrofonen darstellen. (vgl. Hoy 2018)

Das eben beschriebene Nutzungsprinzip wird von Amazon auch als *Alexa for Business* (für den Rest dieses Kapitels mit AfB abgekürzt) in die Arbeitswelt getragen. AfB ist eine Erweiterung des bestehenden Systems, welche eine Vernetzung von Echo-Geräten eines Arbeitsumfelds sowie eine umfangreichere Konfiguration und Verwaltung mancher Aspekte zulässt. Insbesondere sind folgende Punkte wesentlich für AfB: (vgl. Amazon o.J.[a]; Davis und Srivastava 2017; Oostergo 2017)

- Verschiedene Rollen für Echo-Geräte
- Zusätzliche private Skills
- Erweitertes Management von Skills
- Verwaltung von und Anbindung an Kontextinformationen
- Bereitstellen von Funktionalität, welche auf den Arbeitsalltag ausgerichtet ist

Echo-Geräte können im Verbund mit AfB auf zwei verschiedene Arten betrieben werden: als persönliche oder als öffentliche Geräte. Erstere sind einem bestimmten Mitarbeiter zugeordnet und sind in der Regel für dessen persönlichen Arbeitsplatz gedacht. Derartige Geräte können auf die persönlichen Daten wie z.B. den Kalender des zugehörigen Benutzers zugreifen und dabei helfen, Termine, To-do-Listen usw. zu verwalten. Öffentliche Geräte hingegen sind an keinen eigenen Benutzer gebunden und für den Einsatz an Orten wie Meetingräumen gedacht, zu denen mehrere Personen Zugang haben (siehe Abb. 2.1). Ihr Funktionsumfang kann eingeschränkt werden, damit Daten oder Funktionalitäten nicht an Unberechtigte angeboten werden. (vgl. Amazon o.J.[a]; Davis und Srivastava 2017; Oostergo 2017)

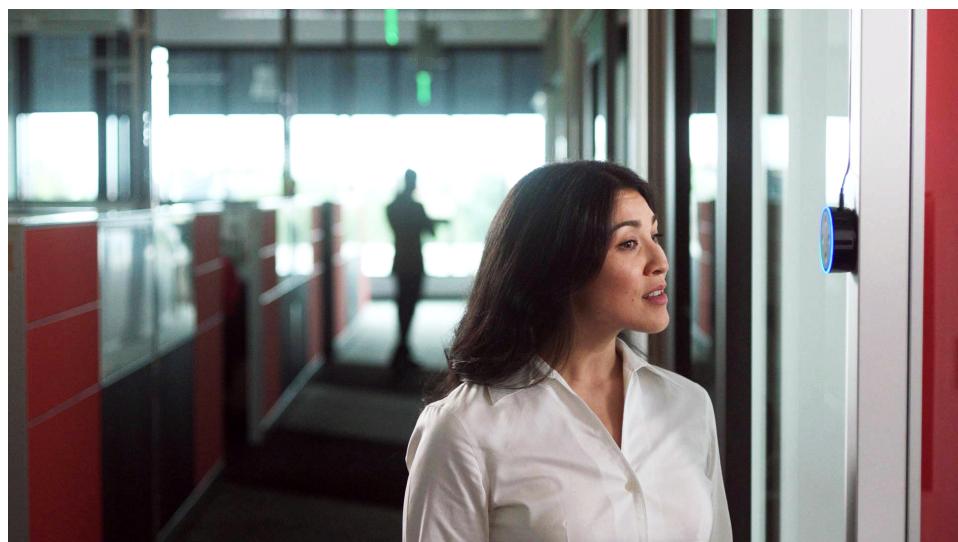


Abbildung 2.1: Öffentliche Echo-Geräte können z.B. an den Wänden von Räumen oder Gängen platziert werden, um für alle Mitarbeiter verfügbar zu sein. (Bild: Amazon o.J.[b])

Das Konzept der Skills wird bei AfB um private Skills erweitert. Diese sind Anwendungen, die speziell für ein Arbeitsumfeld entwickelt wurden und auch nur dort existieren sollen, weil sie etwa mit sensiblen Daten umgehen. Im Gegensatz zu normalen Skills können ihre privaten Pendants nicht einfach über Amazon bezogen werden, sondern sind dem zugehörigen AfB-Benutzerkonto vorbehalten. Zusätzlich können Skills gruppiert werden und nur über bestimmte Echo-Geräte verfügbar gemacht werden. (vgl. Amazon o.J.[a]; Davis und Srivastava 2017; Oostergo 2017)

Im Hintergrund von AfB existiert eine Reihe von Werkzeugen für die Datenverwaltung. Über diese können kontextuelle Informationen hinterlegt und verknüpft werden, auf die Alexa zurückgreifen kann. So ist es möglich, Skills zu erstellen, die Kontextinformationen miteinbeziehen wie z.B. den Ort, an dem sich ein Echo-Gerät befindet oder den Kalender der Person, welcher ein Echo-Gerät zugeordnet ist. (vgl. Amazon o.J.[a]; Davis und Srivastava 2017; Oostergo 2017)

AfB ist interoperabel mit einer breiten Palette an Software, die sich in Arbeitsumfeldern etabliert hat. Dazu zählen unter anderem diverse Kalenderdienste, Office-Programme und Videokonferenz-Lösungen (darunter auch Amazons eigenes Produkt *Chime*). Zusätzlich bietet AfB bereits von Haus aus eine Reihe von Skills an, die im Arbeitsalltag und der Projekt-Kollaboration hilfreich sein können. Nachfolgend ist eine kleine Auswahl der verfügbaren Funktionalitäten aufgeführt: (vgl. Amazon o.J.[a]; Davis und Srivastava 2017; Oostergo 2017)

- Verwalten von Kalendern und To-do-Listen
- Finden eines freien Meetingraums
- Liefern von Wegbeschreibungen zu bestimmten Räumlichkeiten
- Eröffnen, Betreten und Beenden von Videokonferenzen
- Steuern von kompatibler Meetingraum-Ausrüstung
- Abfragen von aktuellen projektbezogenen Daten und Berichten

Zum Zeitpunkt der Entstehung dieses Dokuments war AfB lediglich in Nordamerika verfügbar. Das System ist kostenpflichtig, wobei sich die Höhe der regelmäßig anfallenden Kosten nach der Zahl der eingesetzten persönlichen und öffentlichen Echo-Geräte richtet. (vgl. Amazon o.J.[a])

2.2. Cortana

Microsofts 2013 erschienener Assistent *Cortana* verfolgt bei der Bereitstellung eine andere Philosophie als das in Abschnitt 2.1 vorgestellte System von Amazon. Cortana wurde als installierbares Anwendungsprogramm realisiert und primär dadurch verbreitet, dass der Assistent in den Desktop- und Smartphone-Betriebssystemen von Microsoft bereits enthalten ist. Hierdurch hat Cortana automatisch sowohl bei Endverbrauchern als auch in Unternehmen einen hohen Verbreitungsgrad erreicht. Das System kann aber auch auf weiteren Plattformen wie Android und iOS nachträglich hinzugefügt werden. Zudem existiert ebenfalls dedizierte Hardware, über welche mit dem Assistenten interagiert werden kann, ähnlich wie bei Amazons Alexa. Die Kommunikation mit Cortana erfolgt über natürliche Sprache und kann dabei auditiver oder textueller Natur sein. (vgl. Hoy 2018)

Der Funktionsumfang des Systems ist wie bei Alexa über Skills strukturiert, wobei jeder Skill einem bestimmten Anwendungsfall zugeordnet ist. Einige der standardmäßig vorhandenen Skills sind im Folgenden aufgeführt: (vgl. Ballew 2018; Microsoft o.J.[b])

- Erinnern an Ereignisse (zeit- oder standortbasiert)

- Verwalten von Kalendern und To-do-Listen
- Steuern von Anwendungen
- Strukturierte Ablage und selektives Finden sowie Öffnen von Dateien

Gerade die letzten beiden Punkte der Aufzählung profitieren von der Realisierung des Assistenten als installierbare Anwendung und sind mit einem System wie Alexa nur über Umwege möglich. Dabei ist Cortana sowohl in die Betriebssysteme von Microsoft als auch in deren verbreitete Anwendungsprogramme wie das Softwarepaket *Office* und *Skype* tief integriert, um eine möglichst gute Nutzererfahrung zu bieten. (vgl. Ballew 2018; Kapko 2018)

Das System von Cortana besitzt eine KI-Komponente und ist damit fähig, durch wiederholte Verwendung zu lernen, sich auf den Benutzer einzustellen. Dadurch kann Cortana auch proaktiv auftreten und ausgehend von bisher Gelerntem dem Benutzer situativ passende Vorschläge unterbreiten. (vgl. Ballew 2018)

Die ursprünglich angepeilte Zielgruppe von Cortana waren Endverbraucher, die Windows nutzen. Das System kann zwar durchaus sowohl im privaten als auch beruflichen Alltag nutzbringend eingesetzt werden, besitzt jedoch keinen eindeutigen Fokus auf die Unterstützung von kollaborativer Projektarbeit. Diesen Umstand möchte Microsoft in absehbarer Zeit ändern. Auf ihrer hauseigenen Konferenz *Microsoft Build* wurde im Jahr 2018 eine prototypische Zukunftsvision eines KI-gestützten Meetingraums präsentiert. In diesen Raum war eine erweiterte Version von Cortana integriert, welche über eine noch in der Entwicklung befindliche Hardware Audio- und Videodaten aufnehmen konnte (siehe Abb. 2.2). Die gezeigte Funktionalität umfasste dabei folgendes: (vgl. Hachman 2018; Microsoft 2018b)

- Finden eines freien Meetingraums
- Echtzeit-Erstellung eines textuellen Transkripts von Unterhaltungen
- Echtzeit-Übersetzung von erstellten Transkripten in andere Sprachen
- Zusammenfassen von Unterhaltungen über das Identifizieren von Kerninhalten
- Video-Identifikation von und kontextspezifisches Reagieren auf bestimmte Personen
- Erweiterte Funktionalitäten für das kollaborative Bearbeiten von Dokumenten (z.B. das Hinterlegen von To-do-Nachrichten für bestimmte Personen, die dann benachrichtigt werden)

Zum Zeitpunkt der Entstehung dieses Dokuments sind neben der Präsentation keine weiteren Informationen über die entsprechenden Weiterentwicklungen von Cortana verfügbar.

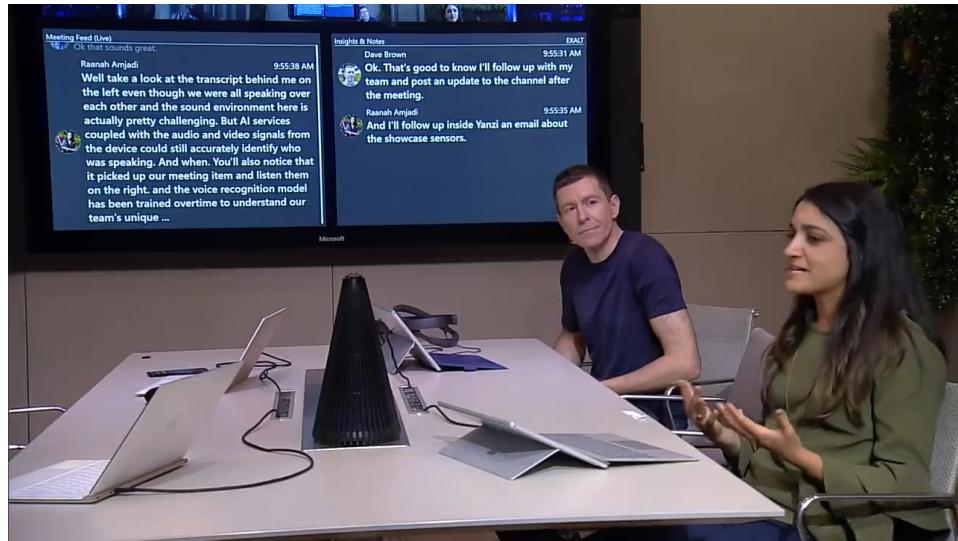


Abbildung 2.2: Die in der Mitte des Tisches befindliche Hardware ist für die Aufnahme von Audio- und Videodaten zuständig. Im Hintergrund ist zu sehen, wie ein textuelles Transkript des Meetings in Echtzeit erstellt wird. (Bild: Microsoft 2018a)

2.3. Interaktive Whiteboards

Die in den vorigen beiden Abschnitten beschriebenen Assistenzsysteme können umfassend in ein Arbeitsumfeld integriert werden. Somit sind sie flächendeckend verfügbar und können verschiedenste Aspekte einer solchen Umgebung ansteuern. Interaktive Whiteboards verfolgen einen anderen Ansatz, indem sie als gekapselte Arbeitsstationen für die kollaborative Projektarbeit fungieren. Als Beispiel für derlei Geräte soll in diesem Abschnitt exemplarisch das *Jamboard* von Google beschrieben werden. Es gibt durchaus auch Alternativen wie das *Surface Hub* von Microsoft oder das *Spark Board* der Firma Cisco. Diese dienen aber grundsätzlich dem gleichen Zweck und besitzen auch einen vergleichbaren Funktionsumfang.

Das Jamboard wirkt nach außen wie ein wahlweise freistehender oder an der Wand angebrachter Bildschirm von der Größe eines Fernsehers, in dem zusätzlich eine Kamera und ein Mikrofon verbaut sind (siehe Abb. 2.3). Die Steuerung erfolgt dabei über Toucheingaben. Primärer Zweck des Geräts ist es, mit ihm sogenannte *Jams* abhalten zu können. Jams sind Sitzungen, welche Videotelefonie, das gemeinsame Erstellen von Skizzen und die Funktionalität von Googles *G Suite*¹ miteinander kombinieren. Während Jams können einerseits bis zu 16 Personen gleichzeitig an einem einzelnen Jamboard miteinander arbeiten. Andererseits ist es auch möglich, mehrere Jamboards miteinander zu synchronisieren. Auf diese Weise können Personen an unterschiedlichen Standorten in Echtzeit gemeinsam an

¹Googles G Suite ist eine Sammlung von Cloud-Diensten für den kollaborativen Einsatz im Arbeitsumfeld. Sie umfasst z.B. einen Cloud-Speicher, E-Mail-Client, Kalender und Programme zur Textverarbeitung sowie Tabellenkalkulation.

Inhalten arbeiten und währenddessen visuell und auditiv miteinander kommunizieren. Zusätzlich zu anderen Jamboards können auch mobile Endgeräte über eine entsprechende App synchronisiert werden. Somit können auch Nutzer von Android- und iOS-Systemen an Jams teilnehmen. Eine Teilnahme über den Webbrower ist ebenfalls möglich, jedoch entfällt dann die Möglichkeit zur Mitarbeit und man ist lediglich auf den kommunikativen Aspekt des Systems beschränkt. (vgl. Google o.J.[c],[d])

Während Jams können unterschiedliche Zusatzfunktionen genutzt werden. Diese reichen von einer Handschrifterkennung, welche Geschriebenes automatisch in Druckbuchstaben umsetzt, bis hin zum Zugriff auf die G Suite. Letzteres ermöglicht es etwa, Daten wie Bilder und Tabellen aus anderen Diensten in Jams miteinzubeziehen. Um die Transition zwischen Jams reibungslos zu gestalten, werden Inhalte automatisch in Googles Cloud-Speicher gesichert. Von dort können sie mit weiteren Personen geteilt oder in zukünftigen Jams zur weiteren Bearbeitung wieder geöffnet werden. Das klassische Abfotografieren eines Whiteboards und Verschicken des Bilds entfällt somit. (vgl. Google o.J.[c],[d])

Zum Zeitpunkt der Entstehung dieses Dokuments ist Googles Jamboard bereits seit einem Jahr erhältlich. Für die Nutzung des Geräts fallen neben den einmaligen Kosten für die Hardware auch regelmäßige Kosten für die Nutzung der G Suite an (vgl. Google o.J.[c]). Die zu Beginn dieses Abschnitts erwähnten Alternativen zum Jamboard besitzen einen ähnlich geschnittenen Funktionsumfang. Natürlich wurden die Geräte nicht primär in Googles Cloud-Plattform, sondern vorzugsweise in die verfügbare Software von Microsoft bzw. Cisco integriert. Hervorzuheben ist die Kompatibilität des Surface Hub mit der ebenfalls von Microsoft stammenden AR-Brille (Augmented Reality) *HoloLens*. Zudem ist die zweite Generation des Surface Hub Teil des in Abschnitt 2.2 erwähnten Zukunftskonzepts für Meetingräume.



Abbildung 2.3: Googles Jamboard folgt dem aktuellen Trend der Bedienung über Toucheingaben. Das Gerät ist eine mobile Arbeitsstation für kollaborative Arbeit. (Bild: Google o.J.[b])

3. Konzept

Dieses Kapitel beschreibt die Konzipierung eines Systems, das in Arbeitsumfeldern eingesetzt werden kann, um sie smarter zu gestalten (was das im Detail bedeutet, wird in Abschnitt 3.3 näher erläutert). Im weiteren Verlauf des Dokuments wird das System als *Smart Lab* bzw. SL bezeichnet. SL soll dabei eine Nische ausfüllen, welche die in Kapitel 2 vorgestellten Systeme nicht abdecken. Neben der Klärung der fachlichen Prozesse werden in diesem Kapitel ebenso alle Begriffe definiert, die für das Verständnis des restlichen Dokuments relevant sind.

In Abb. 3.1 ist eine Übersicht all jener Konzepte und deren Zusammenhänge dargestellt, welche in den folgenden Abschnitten erläutert werden. Die Abbildung soll als eine Art Landkarte dienen, zu der man während der Lektüre dieses Kapitels zurückkehren kann, um gerade Erfahrenes in einem größeren Kontext einordnen zu können.

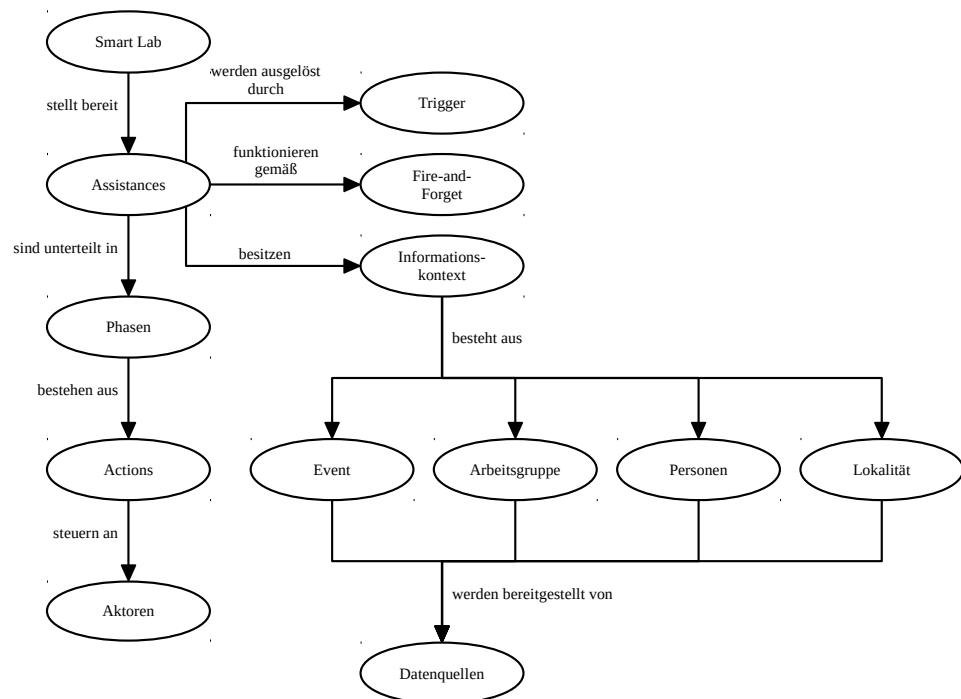


Abbildung 3.1: Die hier dargestellten Konzepte liegen SL zugrunde und sind auf bestimmte Art und Weise miteinander verknüpft.

3.1. Produktvision

SL ist ein System, das in bestehende Arbeitsumfelder integriert werden kann, um den Personen, welche sich in jenen Umgebungen bewegen, Assistenz-Funktionen bereitzustellen. Die Assistenz-Funktionen werden in diesem Zusammenhang als *Assistances* bezeichnet (für Details hierzu siehe Abschnitt 3.7) und stellen nutzbringende Automatismen für die Projekt-Kollaboration dar wie z.B. das automatische Protokollieren eines Meetings. Unter Arbeitsumfeldern sind dabei Orte zu verstehen, an denen Menschen kollaborativen Tätigkeiten nachgehen wie Unternehmen oder Hochschulen.

Damit *Assistances* einen auf die Benutzer ausgerichteten Nutzen erfüllen können, benötigen sie Kontextinformationen. Hierzu kann SL an bestehende Datenquellen (wie z.B. einen Kalenderdienst, ein System für das Identitätsmanagement etc.) angebunden werden, aber auch eigene Datenquellen in eine Systemlandschaft einführen. Aus den Informationen dieser Quellen kann dann ein entsprechender *Assistance-Kontext* zusammengesetzt werden (was in Abschnitt 3.5 näher beschrieben ist). Die Ausführung von *Assistances* wird durch sogenannte *Trigger* angestoßen (die Details hierfür können in Abschnitt 3.6 nachgelesen werden). Trigger-Signale können von innerhalb oder außerhalb des Systems kommen und liefern so eine Möglichkeit SL auf Benutzer und andere Systeme reagieren zu lassen.

Assistances bewirken, dass an SL angebundene *Aktoren* (auf welche in Abschnitt 3.9 näher eingegangen wird) so zum Einsatz kommen, dass sich deren Funktionalitäten zu einem konkreten Nutzen ergänzen. Aktoren können z.B. physisch vorhandene Geräte sein (Mikrofone, Beamer etc.), aber auch lokal installierte Programme oder Webservices. Die Orte, an denen *Assistances* bereitgestellt werden können und Aktoren (egal ob physischer oder virtueller Natur) vorhanden sind, werden im Zusammenhang mit SL als *Lokalitäten* bezeichnet. Dies ist ein in diesem Dokument wiederkehrender Begriff, der so generisch gehalten ist, dass mit ihm ganze Gebäude, Stockwerke, Räume oder aber auch nur einzelne Arbeitsplätze innerhalb eines Raums bezeichnet werden können.

SL besitzt in der ersten Iteration einen Fokus auf das Bereitstellen von *Assistances* während zeitlich beschränkter Ereignisse, die im Rahmen dieses Dokuments als *Events* bezeichnet werden (siehe Abschnitt 3.4.3). Das gängigste Beispiel für Events in einem Arbeitsumfeld sind wahrscheinlich Meetings. Jedoch soll das System so aufgebaut sein, dass es prinzipiell in Zukunft auch für die Anwendungen abseits von Events eingesetzt werden kann. Welche *Assistances* während eines Events zum Einsatz kommen, ist durch die Benutzer konfigurierbar. Weil sich SL in den Kalenderdienst eines Arbeitsumfelds integrieren lässt, ist die Konfiguration der *Assistances* direkt während des Anlegens eines Termins im Kalender möglich.

Im Kern ist SL ein System, das Aktoren eines Arbeitsumfelds orchestriert, um die Benutzer entsprechend ihrer Wünsche während Events zu unterstützen (siehe Abb. 3.2).

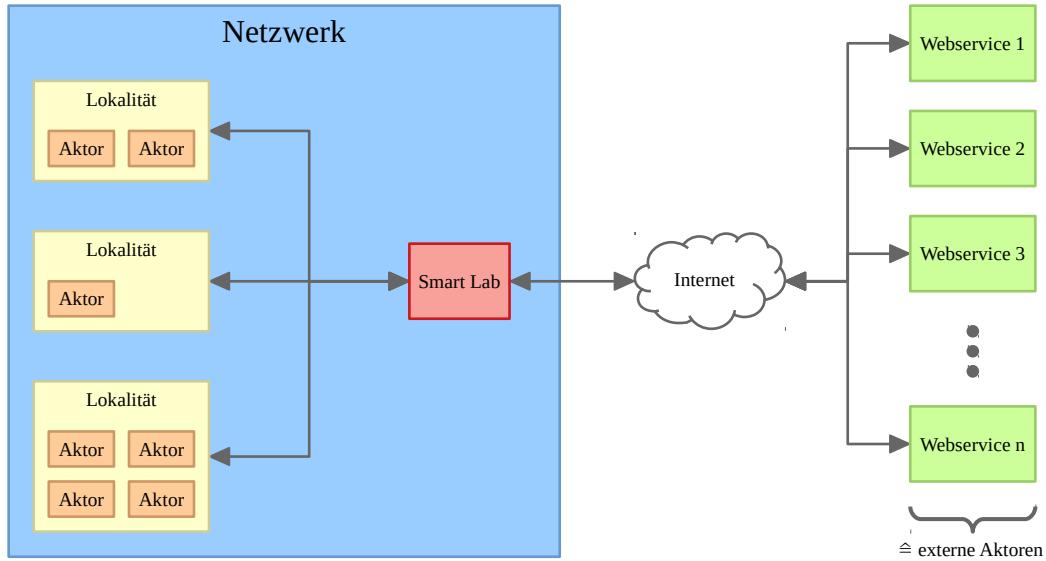


Abbildung 3.2: SL orchestriert an einer Lokalität (z.B. in einem Meetingraum) vorhandene Akteure, um nutzbringende Assistenzfunktionen bereitzustellen. Dabei werden Daten und Signale von den Lokalitäten empfangen und gegebenenfalls mit Hilfe von Webservices weiterverarbeitet. Im Gegenzug empfangen die lokalen Akteure Instruktionen von SL.

3.2. Arten der Konfiguration

Im Laufe dieses Dokuments fällt der Begriff *Konfiguration* regelmäßig. Dabei besitzt er nicht immer nur eine ganz allgemeine Bedeutung, sondern wird auch in speziellen Zusammenhängen verwendet. Diese sollen im folgenden Abschnitt erklärt werden, bevor mit der Erläuterung der eigentlichen Konzepte von SL fortgefahren wird.

Konfiguration des Systems Hierbei handelt es sich um die Festlegung globaler Einstellungen von SL oder Teilen davon. Diese Konfiguration ist grundlegend und erforderlich, damit das System überhaupt erst gestartet werden kann. Ein Beispiel hierfür ist das Bestimmen der für die Netzwerkkommunikation zu verwendenden Ports.

Konfiguration von Events Hierüber werden bestimmte Eigenschaften von Events festgelegt wie deren zeitlicher Rahmen, die Agenda, die teilnehmenden Personen, die gewünschten Assistanzen und weitere.

Konfiguration von Assistanzen Hier vorgenommene Einstellungen legen fest, welche Assistanzen für ein Event bereitgestellt werden sollen. Zusätzlich wird hier die Parametrierung ebendieser Assistanzen vorgenommen. Ihre Konfiguration ist in die des dazugehörigen Events (siehe oben) eingebettet. Da Assistanzen aber ein grundsätz-

licher Bestandteil von SL sind, soll ihre Konfiguration als eigenständiger Punkt hervorgehoben werden.

Im Gegensatz zu ihrer Bedeutung spielt die technische Umsetzung jener drei Arten der Konfiguration für das Verständnis dieses Kapitels keine Rolle. Daher soll hier einfach davon ausgegangen werden, dass entsprechende Mittel und Wege für ihre Durchführung existieren. Die letztendliche Implementierung der dazu notwendigen Mechanismen ist an verschiedenen Stellen in Kapitel 4 beschrieben und entsprechend gekennzeichnet.

3.3. Definition von *smart*

Der Begriff *smart* wird heutzutage in vielen verschiedenen Zusammenhängen verwendet und besitzt keine eindeutige Definition. Deshalb soll dessen Bedeutung im Rahmen dieser Abschlussarbeit hier geklärt werden. Damit keine völlig eigene Definition entsteht, werden die gebräuchlichsten Bedeutungen betrachtet und anschließend nach ihrer Relevanz für SL beurteilt.

Smartness by functionality Häufig werden Geräte als smart bezeichnet, wenn sie eine Vielzahl von Funktionen besitzen, auf welche ihre „normalen“ Pendants nicht zurückgreifen können. Das bekannteste Beispiel hierfür sind Smartphones.

Smartness by connectivity Geräte, die hochgradig interoperabel mit ihrer Umgebung oder anderen Geräten sind, tragen ebenfalls meist den Beinamen *smart*. Diese Definition deckt viele IoT-Geräte (Internet of Things) ab, auf die mit einem Rechner zugegriffen werden kann (z.B. smarte Haushaltsgeräte wie Kühlschränke und Waschmaschinen).

Smartness by configuration Smarte Systeme können sich gemäß einer Konfiguration verhalten und so einen bestimmten Nutzen erfüllen. Ein Beispiel hierfür ist ein System zur Heimautomation, das so konfiguriert wurde, dass es während des Urlaubs der Hausbesitzer abends das Licht einschaltet, um Einbrecher abzuschrecken.

Smartness by intelligence Systeme, deren Funktion eine KI zugrunde liegt, kommen der eigentlichen wortwörtlichen Bedeutung von *smart* am nächsten. Ein Beispiel hierfür ist die KI-Plattform *Watson* der Firma IBM. Über diese werden im Internet verschiedene Dienste für die Verarbeitung von natürlicher Sprache bereitgestellt.

Auf SL treffen die ersten drei dieser Definitionen zu. Assurances können eine breite Palette von Funktionalitäten abdecken (*smartness by functionality*). Da SL ein generisches Framework für deren Ausführung darstellt, sind sie gleichzeitig leicht in ihrer Zahl und ihrem Umfang erweiterbar.

Einen hohen Grad an Interoperabilität muss SL gezwungenermaßen besitzen, damit die Kommunikation mit verschiedenen Aktoren möglich ist (*smartness by connectivity*). Zusätzlich ist eine (möglichst plattformunabhängige) Schnittstelle nach außen erforderlich, um z.B. Trigger-Signale empfangen zu können.

SL setzt vom Benutzer voraus, dass Kontextinformationen über ein Arbeitsumfeld hinterlegt und die gewünschten Assurances vor der Nutzung konfiguriert werden (*smartness by configuration*). Ersteres stellt in der Regel einen einmaligen Aufwand dar. Der Aufwand von Letzterem kann je nach Anzahl und Umfang der gewünschten Assurances variieren und hängt auch von der Art der Events ab, in denen sie eingesetzt werden. Gleichartige und wiederkehrende Events können z.B. auch immer die gleiche Konfiguration für ihre Assurances verwenden. In jedem Fall ist eine solche Konfiguration jedoch obligatorisch, damit SL unterstützend in Aktion treten kann.

In SL ist keinerlei KI integriert (*smartness by intelligence*). Damit ist das System auch nicht fähig, von sich aus Entscheidungen zu treffen (z.B. über zeitliche Parameter oder auszuwählende Optionen) oder für die Entscheidungsfindung aktiv auf die Benutzer zuzugehen. Jegliche Auswahlen sind im Vornherein über Konfigurationen zu treffen. Dennoch ist die Architektur von SL so gestaltet, dass es prinzipiell von KI-Systemen angesprochen werden kann und es auf diese Weise möglich ist, dem Gesamtsystem eine KI-Komponente hinzuzufügen (siehe Kapitel 6).

Damit sind drei der vier hier vorgestellten Definitionen des Begriffs *smart* auf SL anwendbar. Für die letzte (und wahrscheinlich auch interessanteste) kann das System unter Vorbehalt zukünftig erweitert werden.

3.4. Schwerpunkte des Projekts

Um SL von den in Kapitel 2 vorgestellten Systemen abzugrenzen, werden bei der Konzipierung und Implementierung bestimmte Aspekte in den Vordergrund gerückt. Diese werden im Folgenden näher beschrieben.

3.4.1. Datenschutz und Abhängigkeit

Daten, welche in Arbeitsumfeldern anfallen und potenziell verarbeitet werden, sind in der Regel vertraulich. Daher spielt der Datenschutz bei Systemen wie SL eine zentrale Rolle. Die in Kapitel 2 vorgestellten Systeme involvieren meist die Aufnahme von Daten (z.B. Audio- oder Videodaten) sowie deren Weiterverarbeitung über externe Rechenzentren (z.B. die Extraktion von Sprachbefehlen aus Audiodaten oder das Erkennen von Gesichtsmustern in Videodaten) in ihrer Grundfunktionalität. Somit müssen sich Benutzer solcher Systeme automatisch einer Vertrauensfrage stellen, welche zwei Facetten besitzt: (vgl. Hoy 2018; Lang und Benessere 2018)

- Verwendet der Anbieter der Lösung die erfassten Daten für seine eigenen Zwecke weiter?
- Besitzt der Anbieter der Lösung die Fähigkeiten und Mittel, um die erfassten Daten gegen den Zugriff Dritter abzusichern?

Zusätzlich erzeugt die Verwendung von Lösungen externer Anbieter eine Abhängigkeit, die insbesondere beim Einsatz in staatlichen Einrichtungen unerwünscht oder von vornherein ausgeschlossen sein kann.

Um diese Problematik zu umgehen, kann SL komplett selbst im eigenen Netzwerk bereitgestellt werden. Somit werden erfasste Daten grundsätzlich erst einmal innerhalb eines Arbeitsumfelds behalten, anstatt in die Cloud eines externen Anbieters geschickt zu werden. Die Benutzung von Assurances, die ihrerseits auf externe Services zurückgreifen, ist in SL rein optional (d.h. die rechte Hälfte von Abb. 3.2 kann auch weggelassen werden). Ihre Implementierung ist zudem so flexibel, dass die verwendeten Dienste austauschbar sind. Damit wird ermöglicht, dass bei Bedarf auch eigene Lösungen statt externer Services eingehängt werden können.

3.4.2. Kompatibilität

SL kann grundsätzlich im Verbund mit einer breiten Palette von Aktoren betrieben werden, ohne dabei spezielle Hardware- oder Software-Schnittstellen vorauszusetzen. In der Praxis heißt das, dass Aktoren, welche über den USB-Anschluss (Universal Serial Bus) eines PC ansteuerbar sind, ebenso eingebunden werden können wie Aktoren, die eine beliebige Netzwerkschnittstelle besitzen oder über ein Gebäudeautomationssystem verfügbar gemacht werden.

Das heißt jedoch nicht, dass SL bereits Implementierungen für das Ansprechen verschiedenster Aktoren enthält. Vielmehr bedeutet diese Ausrichtung, dass die Architektur von SL so geschnitten ist, dass dessen Kernsystem keine bestimmte Art und Weise der Kommunikation mit Aktoren spezifiziert. Stattdessen kann das System flexibel um neue Software-Adapter erweitert werden, welche die Details zur Ansteuerung der entsprechenden Aktoren kapseln. Auf diese Weise kann auch eine Kompatibilität zu Systemen für die Gebäudeautomation geschaffen werden.

Zusätzlich sind die von SL angebotenen Schnittstellen plattform- und programmiersprachenunabhängig. Damit ist prinzipiell eine Ansteuerung über andere Systeme möglich wie z.b. einen Chatbot oder Sprachassistenten.

3.4.3. Events

Wie bereits in Abschnitt 3.1 erwähnt, liegt der Fokus von SL in der ersten Iteration darauf, im Rahmen von zeitlich beschränkten Ereignissen, sogenannten Events, nutzbrin-

gende Funktionen bereitzustellen. Diese Einschränkung dient lediglich zur realistischen Begrenzung des Umfangs dieser Abschlussarbeit und kann in Zukunft durchaus aufgehoben werden (siehe Kapitel 6). Daher werden Konzept und Architektur von SL bereits unter Berücksichtigung einer Anwendung abseits von Events aufgestellt.

Der Begriff des Events ist innerhalb dieses Dokuments meist mit *Meeting* austauschbar, da diese die häufigste Art von Events in einem Arbeitsumfeld darstellen. Dennoch stehen Events repräsentativ auch für alle anderen denkbaren Ereignisse, welche einen zeitlichen Rahmen besitzen.

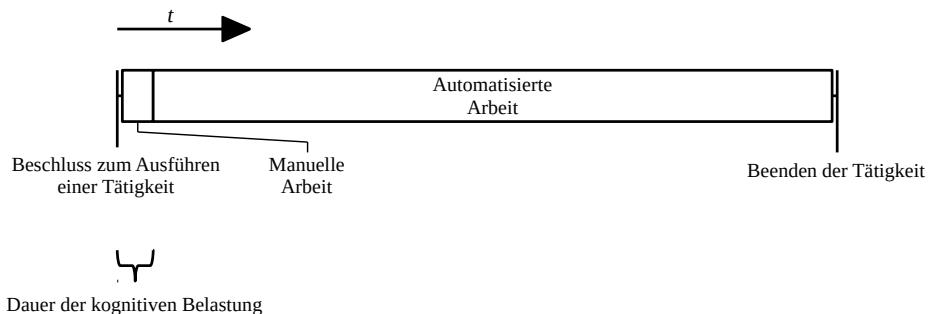
Events können für SL konfiguriert werden, indem in ihnen gewünschte Assurances und Kontextinformationen (Arbeitsgruppe, Agenda etc.) hinterlegt werden. Der genaue Ablauf der Konfiguration ist abhängig von der verwendeten Event-Datenquelle (z.B. ein Kalenderdienst) und spielt für dieses Kapitel keine Rolle. Eine Möglichkeit zur Konfiguration von Events wird hier schlicht als gegeben angenommen. Die Abschnitte 4.5 und 4.6 beschreiben den letztendlich implementierten Konfigurationsmechanismus näher.

Im Zusammenhang mit dem Beobachter-Entwurfsmuster, werden auslösende Signale manchmal ebenfalls als Events bezeichnet. Dies ist jedoch eine gänzlich andere Bedeutung des Begriffs als diejenige, welche ihm im Zusammenhang mit SL zukommt. Daher soll an dieser Stelle darauf hingewiesen werden, um Verwechslungen vorzubeugen.

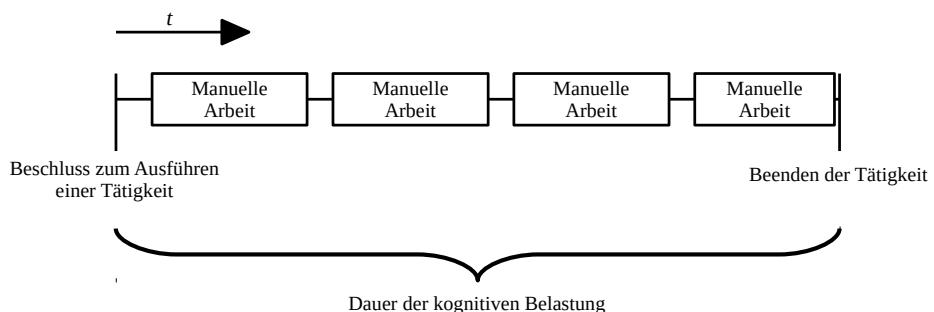
3.4.4. Automatisierungsgrad

Der Automatisierungsgrad von SL ist so gewählt, dass er einen Mittelweg zwischen einfacher Benutzung und Systemkomplexität darstellt. Zur weiteren Erläuterung soll an dieser Stelle der Ausdruck *kognitive Belastung* definiert werden. Dieser stellt einen Sammelbegriff für alles dar, was an der Aufmerksamkeit eines Menschen zehrt oder geistige Kapazität in Anspruch nimmt. Eine Tätigkeit, deren Durchführung (so simpel sie auch sein mag) ein Mindestmaß an Konzentration erfordert ist z.B. eine Art von kognitive Belastung. Aber auch Dinge, deren Durchführung man auf einen späteren Zeitpunkt vertagt hat (z.B. weil gerade schlicht keine Zeit dafür ist oder weil noch auf Informationen anderer Beteiligter gewartet werden muss), stellen eine kognitive Belastung dar. Denn selbst, wenn derartige Tätigkeiten auf einer To-do-Liste oder in einem Terminkalender vermerkt werden, muss ihre Durchführung dennoch mit anderen Aktivitäten abgestimmt und in Einklang gebracht werden. Damit kommt zur Belastung durch die Ausführung der eigentlichen Tätigkeit zusätzlich noch die durch deren Orchestrierung entstandene hinzu. Das Ziel von SL ist es, die kognitive Belastung durch Aufgaben geringer und mittlerer Komplexität, welche prinzipiell automatisiert werden können, möglichst zu minimieren. Durch die Automatisierung kann einerseits Zeit eingespart werden und andererseits werden mentale Ressourcen der Benutzer frei, die auf andere und wichtigere Aufgaben fokussiert werden können.

Das Prinzip von *Fire-and-Forget*¹ (für den Rest des Dokuments mit FaF abgekürzt) dient dabei als Leitbild für den Ablauf der bereitgestellten Assurances. Die Bedeutung von FaF ist in Abb. 3.3a dargestellt. Demnach ist für die Durchführung einer Tätigkeit nur eine einmalige Benutzerinteraktion zum Zeitpunkt des Beschlusses der Ausführung notwendig (d.h. *Fire*). Danach übernehmen Automatismen die restliche Ausführung und der Benutzer braucht sich nicht weiter darum zu kümmern (d.h. *Forget*). Somit wird das Ausführen der Tätigkeit aus Sicht des Initiators annähernd punktuell und es besteht über keinen nennenswerten Zeitraum eine kognitive Belastung.



(a) Die Ausführung der Tätigkeit nach dem Prinzip von FaF.



(b) Die manuelle Ausführung der Tätigkeit.

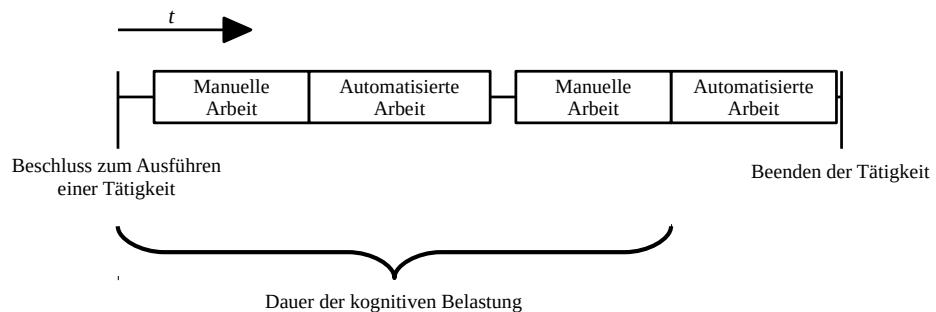
Abbildung 3.3: Verschiedene Arten der Ausführung einer Tätigkeit.

Dem gegenüber steht die manuelle Durchführung dieser Tätigkeit, welche sich über einen gewissen Zeitraum erstreckt und mehrere Male explizite Benutzerinteraktionen erfordert (dargestellt in Abb. 3.3b). Bis zum Beenden der Tätigkeit muss ihr der Initiator einen Teil seiner Aufmerksamkeit widmen und sie stellt (aus den zu Beginn dieses Abschnitts

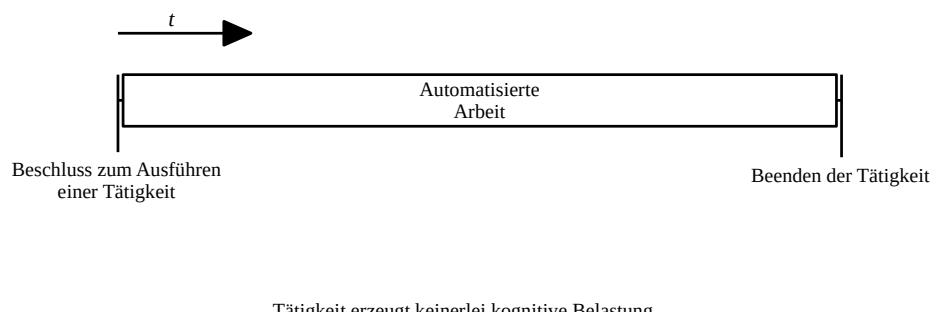
¹Ein Begriff, der aus dem Militärjargon stammt, sich jedoch heutzutage auch in der Informatik etabliert hat. Ursprünglich wurde damit die Verwendung von zielsuchenden Waffen bezeichnet.

genannten Gründen) auch dann eine kognitive Belastung dar, wenn zeitweise gar keine manuelle Arbeit geleistet wird.

Das teilautomatisierte Durchführen einer Tätigkeit (siehe Abb. 3.3c) hat Ähnlichkeit mit der Durchführung gemäß FaF. Jedoch ist hier die Strukturierung nach manueller und automatisierter Arbeit nicht strikt vorgegeben. Wo FaF lediglich zu Beginn eine Benutzerinteraktion vorsieht, können bei der teilautomatisierten Durchführung beliebige Sequenzen von manueller und automatisierter Arbeit aufeinanderfolgen. Eine Entlastung des Benutzers ist zwar gegeben, dennoch stellt die Tätigkeit bis zur Beendigung der manuellen Arbeit eine kognitive Belastung dar, was je nach Ausmaß der Teilautomatisierung eine mehr oder weniger lange Zeit sein kann.



(c) Die Ausführung einer teilautomatisierten Tätigkeit.



(d) Die Ausführung einer vollautomatisierten Tätigkeit.

Abbildung 3.3: Verschiedene Arten der Ausführung einer Tätigkeit.

Die vollautomatisierte Durchführung einer Tätigkeit (siehe Abb. 3.3d) stellt logischerweise die größte Entlastung für den Benutzer dar. Durch den Wegfall der anfänglichen Benutzerinteraktion gegenüber der Durchführung gemäß FaF, ergeben sich jedoch auch größere technische Herausforderungen. Das vollautomatisierte System muss nun selbst-

ständig die Entscheidung treffen, wann bei Benutzern der Beschluss zum Ausführen einer Tätigkeit fällt. Dies kann z.B. durch die Auswertung von deren Verhalten erfolgen (passiv) oder durch selbstständiges Nachfragen (aktiv).

Assistances von SL legen ihren Fokus primär (aber nicht zwangsläufig) auf die Durchführung gemäß FaF. Durch sie werden eigentlich manuelle Tätigkeiten mit Hilfe von Automatisierung nach diesem Prinzip umfunktioniert.

Ein Beispiel für eine solche manuelle Tätigkeit ist das Protokollieren eines Meetings und das anschließende Verteilen des Protokolls an alle Meeting-Teilnehmer. Der Protokollant muss über das gesamte Meeting hinweg einen Teil seiner Aufmerksamkeit dem Aufzeichnen des Gesagten widmen. Anschließend muss er das Protokoll eventuell aufbereiten und z.B. per E-Mail an die Teilnehmer schicken. Ist direkt im Anschluss an das Meeting keine Zeit dafür, so muss der Protokollant diese Aufgabe konsequenterweise auf einen späteren Zeitpunkt verschieben und Zeit dafür einplanen. Entsprechend Abb. 3.3b besteht damit bis zum vollständigen Abschließen der Aufgabe eine kognitive Belastung, welche im Optimalfall vermieden werden kann. Die Abschnitte 3.7.1 und 3.7.2 führen weitere Beispiele für derartige Tätigkeiten aus dem Arbeitsumfeld auf und zeigen wie deren Ablauf durch FaF verbessert werden kann.

Vom Bereitstellen vollautomatisierter Funktionalitäten wird in SL zunächst abgesehen, um den Umfang dieser Abschlussarbeit in realistischen Grenzen zu halten. Zwar würde ein vollautomatisiertes System den größten Nutzen bieten, jedoch auch eine zusätzliche Schicht an Komplexität einführen. Eine Umsetzung wäre voraussichtlich nur mit Hilfe von maschinellem Lernen möglich, was der vorerst angestrebten Definition von *smart* in diesem Projekt widersprechen würde (siehe Abschnitt 3.3).

3.4.5. Kosten-Nutzen-Verhältnis

Der Funktions- und Systemumfang von SL ist so geschnitten, dass es auf einem heutzutage handelsüblichen Desktop-PC oder Notebook betrieben werden kann. Systemteile können bei Bedarf sogar auf kostengünstigen Einplatinencomputern betrieben werden. Insgesamt fallen für den Betrieb von SL lediglich einmalige Kosten in Form von Hardware-Anschaffungen an. Dem gegenüber stehen regelmäßige Nutzungs-Beiträge für die Mehrzahl der Systeme, welche in Kapitel 2 vorgestellt wurden, zuzüglich ebenfalls anfallender Kosten für benötigte Hardware (vgl. Amazon o.J.[a]; Google o.J.[c]).

3.4.6. Open Source

Der Code von SL ist quelloffen und zusammen mit den dazugehörigen Ressourcen wie beispielhaften Konfigurationsdateien und Skripten zum Starten des Systems über das Repository <https://github.com/Kroetz/smarts-lab> des Webservice *GitHub* verfügbar.

Alle in Kapitel 2 vorgestellten Systeme sind hingegen proprietär. Damit ist SL in hohem Maße an die speziellen Anforderungen verschiedener Arbeitsumfelder anpassbar. Zusätzlich können Aspekte wie Vertrauenswürdigkeit und Sicherheit direkt im Quellcode überprüft und beurteilt werden.

3.5. Der Assistance-Kontext

Systeme für die Gebäudeautomation existieren bereits seit einigen Jahren und besitzen in der Regel eine ähnliche Definition des Begriffs *smart* wie SL. Funktionalitäten solcher Systeme sind jedoch meist kontextfrei und können damit nicht auf die Benutzer ausgerichtet werden. Ein Beispiel hierfür ist das automatische Abdunkeln von Fenstern ab einer bestimmten Uhrzeit. Unter Umständen können wünschenswerte Parameter dieses Automatismus von kontextuellen Informationen abhängen, etwa der Jahreszeit oder den Personen, welche sich im Raum befinden.

Dieser Umstand tritt in Arbeitsumfeldern noch stärker in Kraft. In einer solchen Umgebung bewegen sich viele verschiedene Menschen, die unterschiedliche Aufgaben und damit auch unterschiedliche Ansprüche an unterstützende Funktionalitäten besitzen. Ein zielgerichtetes Bereitstellen von Assurances ist somit nur mit entsprechenden Kontextinformationen möglich. Daher besitzt eine Assistance in SL immer einen Kontext, aus dem Informationen geschöpft werden können. Eine Assistance, welche z.B. das Verschicken von E-Mails involviert, kann so dem Kontext entnehmen, wer an einem Event teilnimmt und somit zu den gewünschten Empfängern zählt.

Die Bereitstellung des Kontexts ist eine der Kern-Aufgaben von SL und sorgt erst dafür, dass Assurances komplexe Aufgaben (zumindest in Teilen) automatisieren können. Abbildung 3.4 zeigt den Inhalt eines Assistance-Kontexts.

Jeder Kontext muss zwingend die Konfiguration der dazugehörigen Assistance beinhalten. Diese enthält Parameter, welche die generelle Art und Weise der Ausführung einer Assistance beeinflussen (z.B. welche Akteure einer Lokalität benutzt werden sollen) und kann sich von Assistance zu Assistance stark unterscheiden.

SL besitzt in der ersten Iteration einen Fokus auf das Bereitstellen von Assurances während Events (siehe Abschnitt 3.4.3). Daher enthält ein Kontext konsequenterweise auch Information über das dazugehörige Event wie z.B. dessen Titel und zeitlichen Rahmen. Daneben sind auch Informationen über die Arbeitsgruppe, für die eine Assistance bereitgestellt wird, enthalten. Dies umfasst z.B. deren Namen und Projektablage². Ebenfalls Teil des Kontexts sind Informationen über die Personen, für die eine Assistance bereitgestellt wird. Dies umfasst deren Namen, E-Mail-Adressen und Rollen. Zuletzt beinhaltet der Kon-

²Unter einer Projektablage wird z.B. ein Wiki, Quellcode-Repository o.Ä. verstanden, was von einer Arbeitsgruppe zur Archivierung und Strukturierung ihrer Projektdokumente verwendet wird.

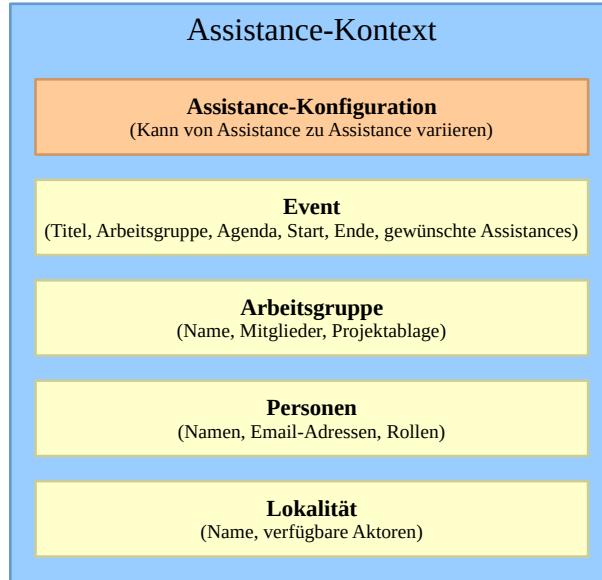


Abbildung 3.4: SL stellt jeder Assistance Kontextinformationen bereit, welche diese bei ihrer Ausführung nutzen kann. Orange dargestellte Teile des Kontexts sind obligatorisch, wohingegen gelb dargestellte Teile optional sind.

text auch Informationen über den Ort, an dem eine Assistance bereitgestellt wird. Darin enthalten sind der Name der Lokalität und die dort verfügbaren Akteure.

Der aufmerksame Leser mag an dieser Stelle eventuell den Eindruck haben, dass manche Informationen im Kontext redundant vorhanden sind. Etwa enthält ein Event eine Arbeitsgruppe, jedoch ist die Arbeitsgruppe auch ein separater Teil des Kontexts. Auch wenn SL zunächst einen Fokus auf das Assistieren während Events besitzt, ist der Assistance-Kontext dennoch bereits für Fälle abseits dieses Szenarios konzipiert. Es sind durchaus auch Assurances denkbar, die keinem Event zugeordnet sind und deshalb auch keines in ihrem Kontext besitzen. Trotzdem kann eine solche Assistance auf eine Arbeitsgruppe ausgerichtet sein. Gleichermaßen gilt auch, wenn eine Assistance nicht für eine dedizierte Arbeitsgruppe (welche eine Gruppe von Personen, die Mitglieder der Arbeitsgruppe, beinhaltet) konfiguriert ist: Sie kann dennoch auf einen gewissen Kreis an Personen ausgerichtet sein, der z.B. auch externe Personen enthält, die nicht regulär zu einem Arbeitsumfeld gehören. Im Extremfall könnte ein Kontext neben der Assistance-Konfiguration keinerlei weitere Informationen enthalten. In einem solchen Fall könnte die dazugehörige Assistance dann aber lediglich einen ungerichteten und globalen Nutzen erfüllen, der relativ eingeschränkt wäre.

SL ist dafür zuständig, den Kontext einer Assistance zusammenzusetzen und bereitzustellen. Das Fehlen einer KI-Komponente in SL (siehe Abschnitt 3.3) verhindert jedoch, dass

das System Kontextinformationen eigenständig aus aufgenommenen Daten (z.B. Audio- oder Videodaten) ermitteln oder Benutzer aktiv danach fragen kann. Daher muss SL die nötigen Informationen selbst aus bestimmten Quellen beziehen. Diese Datenquellen, die für das Zusammenbauen der Assistance-Kontexte genutzt werden, müssen a priori mit entsprechenden Informationen befüllt werden. Der Verzicht auf *smartness by intelligence* geht also zu Lasten eines höheren (aber größtenteils einmaligen) Aufwands bei der Einrichtung von SL.

Die Datenquellen können entweder bereits in der Systemlandschaft eines Arbeitsumfelds vorhanden sein oder zusammen mit SL eingeführt werden. Ein vorhandener Kalenderdienst oder ein System für das Identitätsmanagement können so bei der Konstruktion eines Kontexts als Datenquellen für den Event- oder Personen-Teil dienen. Oft verwalteten Kalenderdienste auch noch die vorhandenen gemeinsam genutzten Lokalitäten eines Arbeitsumfelds, um für sie eine geordnete Möglichkeit der Buchung zu bieten. Auf der anderen Seite können auch neue Datenquellen speziell für SL eingeführt werden wie z.B. eine für die vorhandenen Arbeitsgruppen. Die Nutzung dieser Informationen ist in der Regel auf SL beschränkt und es ist dementsprechend unwahrscheinlich, dass bereits eine solche Datenquelle in einer Systemlandschaft vorhanden ist. Eine mögliche Verknüpfung von SL und Datenquellen ist in Abb. 3.5 zu sehen. Neben den Datenquellen für Events, Lokalitäten, Arbeitsgruppen und Personen gibt es noch eine, welche die verfügbaren Akteure enthält. Diese ist nicht direkt beim Zusammenbau des Assistance-Kontexts involviert, wurde jedoch der Vollständigkeit halber mit in Abb. 3.5 aufgenommen. Näheres zu Akteuren ist in Abschnitt 3.9 zu finden.

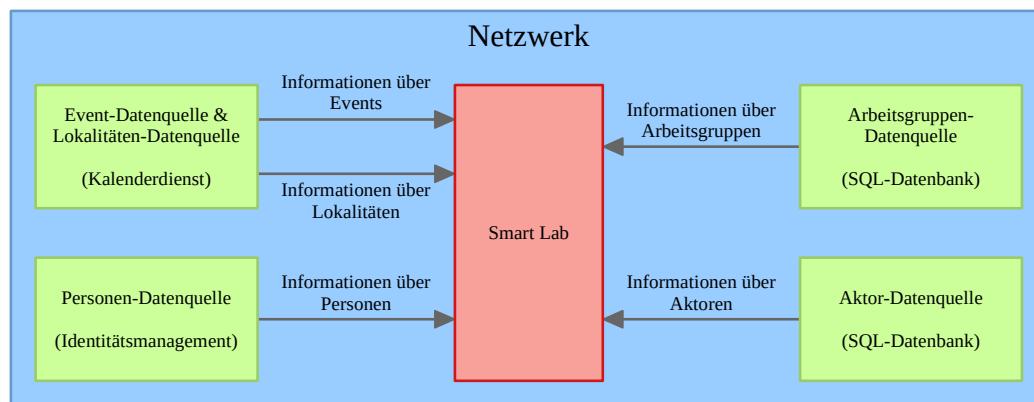


Abbildung 3.5: SL kann an verschiedene Datenquellen angebunden werden, deren Informationen für den Zusammenbau eines Assistance-Kontexts verwendet werden. Die hier gezeigten Services und Komponenten, welche als Datenquellen verwendet werden, sind lediglich ein Beispiel für eine mögliche Konfiguration des Systems.

Ob vorhandene Datenquellen angebunden werden oder neue in eine Systemlandschaft eingeführt werden, ist in SL frei konfigurierbar. Auf diese Weise ist SL in verschiedenste Systemlandschaften integrierbar, unabhängig davon, ob sie einen Kalenderdienst, ein Identitätsmanagement o.Ä. besitzen.

Die Art der Datenquellen ist nicht weiter spezifiziert. Denkbar sind Quellen in Form von simplen Datenbanken, aber auch das Einbinden eines externen Cloud-Service (z.B. eines Kalenderdiensts). In jedem Fall müssen die Datenquellen im Vornhinein mit entsprechenden Informationen befüllt werden, damit SL sie für die Kontext-Bereitstellung verwenden kann.

3.6. Trigger

Trigger-Signale sind das, was in SL die Ausführung einer Assistance loslässt. Dabei sind die Trigger selbst vollkommen von ihren Konsequenzen losgelöst. Vielmehr beschreiben die Signale ein Ereignis, welches eingetreten ist. Auf dieses kann dann vom System eine entsprechende Reaktion erfolgen (oder auch nicht, falls keine Reaktion notwendig ist). Der Fokus von SL auf Events (siehe Abschnitt 3.4.3) macht entsprechende Trigger notwendig, die den zeitlichen Rahmen von Events markieren. Zu diesem Zweck werden vier verschiedene Trigger-Signale definiert, welche in Abb. 3.6 zu sehen sind.

set up event Dieser Trigger signalisiert den formalen zeitlichen Anfang eines Events wie z.B. den Start eines Meetings in einem Kalender. Das Trigger Signal ist rein zeitabhängig.

start event Dieses Signal resultiert aus einer Benutzerinteraktion mit SL. Die Benutzer zeigen damit, dass ein Event aus ihrer Sicht beginnen kann.

stop event Dieser Trigger wird ebenfalls über die Benutzer generiert. Sie signalisieren damit, dass ein Event aus ihrer Sicht beendet ist.

clean up event Dieses Signal markiert das formale zeitliche Ende eines Events wie etwa das Ende eines Meetings in einem Kalender. Wie sein Gegenstück *set up event* ist dieser Trigger rein zeitabhängig.

Erreicht ein Event sein formales zeitliches Ende, wird neben dem Trigger *clean up event* auch automatisch *stop event* ausgelöst, falls das erforderlich sein sollte. Dies ist genau dann nötig, falls zuvor das Signal *start event*, jedoch nie manuell das dazugehörige *stop event* gesendet wurde³.

³Das in diesem Absatz beschriebene Verhalten ist in der ersten Iteration von SL lediglich Teil des Konzepts. Im Gegensatz zu allen anderen erläuterten Prinzipien und Mechanismen wurde es jedoch noch nicht implementiert.

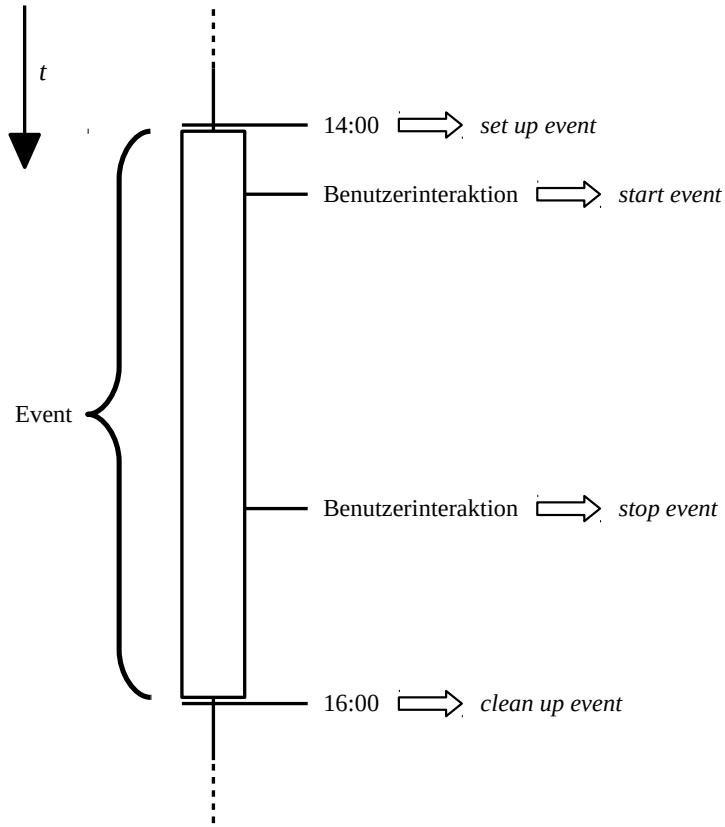


Abbildung 3.6: SL kennt vier verschiedene Trigger-Signale, die den zeitlichen Rahmen von Events markieren.

Die Entscheidung den Start und das Ende eines Events in jeweils einen zeit- und einen benutzergesteuerten Trigger aufzuspalten, mag unter Umständen zunächst verwunderlich sein. Es gibt jedoch einerseits Assurances, die bereits wirken sollen, wenn Benutzer eine Lokalität betreten (z.B. die Vorbereitung von Geräten, Programmen und sonstigen Umgebungsparametern). Andererseits sind auch Assurances denkbar, die während eines Events erst in Kraft treten sollen, wenn alle betreffenden Benutzer versammelt sind und kollektiv dessen Start beschlossen haben. Ein Beispiel hierfür sind Assurances, welche das Aufnehmen von Video- oder Audiodaten involvieren. Wenn die Aufzeichnung von Daten bereits laufen würde, während zwei Benutzer noch auf ihre verbleibenden Kollegen warten und sich solange über private Dinge unterhalten, hätte dies eher den Charakter einer Überwachung. Darüber hinaus können Arbeitsgruppen unter Umständen auch gar nicht zu einem Event erscheinen, was die Datenaufnahme komplett obsolet machen würde. Auf welche Art und Weise die benutzergesteuerten Trigger-Signale gesendet werden, ist an dieser Stelle nicht

spezifiziert und auch nicht weiter relevant. Denkbar sind z.B. das Drücken eines Knopfs, ein Sprachbefehl oder weitere ähnliche Methoden.

Wie Assistancess auf Trigger reagieren ist nicht durch die Trigger selbst spezifiziert. Stattdessen kapseln Assistancess dieses Verhalten, was ihnen auch die Flexibilität gibt, manche Signale einfach zu ignorieren. Weiterhin sind Trigger-Signale nicht eindeutig bestimmten Assistancess zugeordnet. Es können durchaus mehrere Assistancess auf ein und dasselbe Signal reagieren. Ein Trigger kann von Event zu Event verschiedene Auswirkungen haben. Welche dies sind, hängt ganz davon ab, welche Assistance für ein Event konfiguriert wurden.

SL kann Trigger-Signale von außerhalb des Systems empfangen oder in manchen Fällen auch selbst generieren. Die vorgestellten zeitgesteuerten Trigger können etwa durch ein Subsystem bereitgestellt werden, das deren korrespondierenden Zeitpunkt durch das Abfragen einer Event-Datenquelle, z.B. eines Kalenderdiensts, ermittelt. Für das Empfangen von Trigger-Signalen besitzt SL ein plattformunabhängiges API (Application Programming Interface), damit gemäß Abschnitt 3.3 eine möglichst breite Interoperabilität zu anderen Systemen gewährleistet ist. Auf die Schnittstelle wird in Abschnitt 3.13 im Detail eingegangen.

In der ersten Iteration sind für SL lediglich die in diesem Kapitel vorgestellten vier Trigger spezifiziert. Es sind jedoch durchaus weitere Signale denkbar, welche nicht unbedingt an den zeitlichen Rahmen von Events geknüpft sind. Dazu können SL leicht weitere Signale hinzugefügt werden, ohne dass dies dabei Auswirkungen auf bereits bestehende Trigger oder Assistancess hat.

3.7. Assistancess

Im Folgenden wird eines der wichtigsten Konzepte von SL vorgestellt: Assistancess. Zusätzlich werden in Abschnitt 3.7.1 diejenigen Assistancess beschrieben, welche im Rahmen dieser Abschlussarbeit ausgearbeitet und implementiert wurden. In Abschnitt 3.7.2 sind weitere Anwendungsfälle für Assistancess aufgeführt, die jedoch nicht so detailliert wie in Abschnitt 3.7.1 beschrieben sind und auch nicht umgesetzt wurden.

Assistancess stellen den fachlichen Nutzen dar, den SL erbringen kann. Ihre Bereitstellung ist der primäre Zweck des gesamten Systems. Die Menge der denkbaren Assistancess schließt sämtliche zumindest teilweise automatisierbaren Tätigkeiten ein, die für ein Arbeitsumfeld typisch sind. Die Ausführung einer Assistance wird durch Trigger-Signale angestoßen, welche für die entsprechende Assistance relevant sind. Mit den Assistancess werden zwei Ziele verfolgt:

- Sie sollen Aufgaben automatisieren, die bislang manuell erledigt werden müssen (z.B. das Protokollieren eines Meetings).

- Sie sollen Funktionalitäten ermöglichen, die manuell entweder gar nicht oder nur mit großem Aufwand umsetzbar sind (z.B. das Entsperren eines Raums für die Dauer der Raum-Buchung exklusiv für Mitglieder der Arbeitsgruppe, welche ihn gebucht hat).

Die ultimativ angestrebte Folge davon ist es, einerseits Zeit der Benutzer einzusparen und andererseits ihre kognitive Belastung zu verringern (siehe Abschnitt 3.4.4). Daher liegt den Assistanzen von SL nach Möglichkeit das Prinzip von FaF zugrunde. Dies bedeutet, dass für die Ausführung einer Assistance lediglich zu Beginn eine kurze Benutzerinteraktion nötig sein sollte, sodass der Benutzer seine Aufmerksamkeit danach anderen Dingen widmen kann. Diese Interaktion kann z.B. das Liefern eines Trigger-Signals sein oder das Anlegen bzw. Erweitern einer Assistance-Konfiguration für ein Event.

Die Konfiguration von gewünschten Assistanzen muss von den Benutzern explizit in einem Event hinterlegt werden. Dort werden jeweils allgemeine Parameter, welche für die Ausführung einer Assistance relevant sind, spezifiziert (z.B. welche Aktoren verwendet werden sollen). Neben diesen in ihrer Konfiguration enthaltenen Parametern kann eine Assistance je nach Komplexität und Umfang ihrer Funktionalität eventuell auch weitere Kontextinformationen benötigen. Dazu können z.B. Informationen über die Arbeitsgruppe zählen, für die eine Assistance bereitgestellt wird. Mit Hilfe des Kontexts können Assistanzen Funktionen erfüllen, welche auf die Benutzer eines Arbeitsumfelds ausgerichtet sind.

Die Ausführung von Assistanzen unterteilt sich in die drei Phasen *begin*, *end* und *during*. Die ersten beiden dieser Phasen werden jeweils einmal zu Beginn und am Ende des Zeitraums ausgeführt, für den eine Assistance aktiv ist. Die Phase *during* hingegen kann beliebig oft und jederzeit zwischen *begin* und *end* vorkommen (siehe Abb. 3.7).

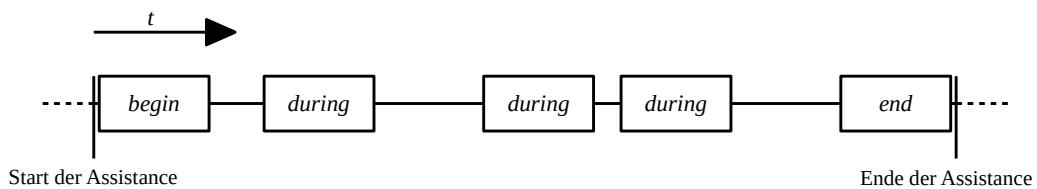


Abbildung 3.7: Ein Beispiel für die Ausführungsreihenfolge der drei Phasen einer Assistance.

Jede der drei Phasen kann prinzipiell unterschiedliche Ausführungslogik enthalten. Die Ausführungslogik zweier Assistanzen oder auch zweier Phasen von ein und derselben Assistance kann aber auch teilweise fachlich identische Teile beinhalten. Zu diesem Zweck setzen sich die Phasen von Assistanzen aus Bausteinen eines gemeinsamen Pools zusammen.

Jene Bausteine werden für den Rest dieses Dokuments als *Actions* bezeichnet. Für das Verständnis des Aufbaus von Assurances reicht es zu wissen, dass Actions wiederverwendbare Elemente sind, welche in den Phasen einer Assurance hintereinandergeschaltet werden können. Der Name einer Action beschreibt eindeutig den Nutzen, den sie erfüllt, wodurch deren Rolle in einer Assurance klar ersichtlich ist. Actions können (je nachdem was sie tun) auch Daten zurückliefern, welche wiederum als Input für andere Actions dienen können. Das Konzept von Actions wird in Abschnitt 3.8 ausführlich behandelt.

Abbildung 3.8 zeigt beispielhaft die vollständige Definition des Ausführungsverhaltens einer Assurance namens *minute taking*, deren Zweck es ist, während eines Events automatisch Protokoll zu führen.

Assurance	Trigger	Phase	Action
<i>minute taking</i>	<i>start event</i>	<i>begin</i>	<i>audio recording start</i>
	<i>stop event</i>	<i>end</i>	<i>audio recording stop</i>
			<i>speech to text</i>
			<i>data upload (Audiodaten)</i>
			<i>data upload (Transkript)</i>
	-	<i>during</i>	-

Abbildung 3.8: Die vollständige Definition des Ausführungsverhaltens der Assurance *minute taking*.

Die Kopplung von Trigger-Signalen zu den Phasen einer Assurance, deren Ausführung sie anstoßen, ist in der entsprechenden Assurance selbst gekapselt. Im Fall von *minute taking* löst das Trigger-Signal *start event* die Phase *begin* aus und das Signal *stop event* stößt die Phase *end* an. Der Phase *during* ist kein Trigger-Signal zugeordnet und ihre Ausführung kann dementsprechend auch nicht ausgelöst werden.

Die Phasen von *minute taking* zeigen deutlich, dass Art und Anzahl ihrer enthaltenen Actions stark variieren können. Die Phase *begin* enthält lediglich eine Action, welche die Aufnahme von Audiodaten startet. Bei Beendigung der Assurance (d.h. wenn die Phase *end* ausgeführt wird) geschehen mehrere Dinge:

1. Die Aufnahme von Audiodaten wird gestoppt.
2. Die aufgenommenen Audiodaten werden in ein textuelles Transkript umgewandelt.

3. Die aufgenommenen Audiodaten werden an ein bestimmtes Ziel übertragen.
4. Das erstellte Transkript wird ebenfalls an ein bestimmtes Ziel übertragen.

Natürlich müssen auch Actions parametriert werden, um korrekt arbeiten zu können (Welches Gerät soll für die Aufnahme von Audiodaten verwendet werden? Welcher Speech-to-Text-Service soll verwendet werden? An welches Ziel, z.B. eine Projektablage, sollen Daten übertragen werden?). Auf diesen Umstand wird in Abschnitt 3.8 näher eingegangen. Die dritte und letzte Phase *during* schließlich enthält keinerlei Actions und erfüllt damit auch keinen praktischen Nutzen in *minute taking*. Das ist jedoch wenig verwunderlich, da die Ausführung dieser Phase gar nicht erst ausgelöst werden kann.

Diagramme wie Abb. 3.8 beschreiben das komplette Ausführungsverhalten von Assurances. Die Abschnitte 3.7.1 und 3.7.2 enthalten ebenfalls Diagramme dieser Art, um weitere Assurances zu charakterisieren.

3.7.1. Ausgearbeitete Assurances

In diesem Abschnitt werden die fünf Assurances und deren Anwendungsfälle vorgestellt, welche im Rahmen dieser Abschlussarbeit für die erste Iteration von SL vorgesehen sind. Ihre letztendliche Implementierung ist in Abschnitt 4.7 beschrieben.

Automatisches Protokollieren eines Events (alias *minute taking*)

Meetings o.Ä. haben oft einen engen Zeitrahmen und beinhalten gleichzeitig eine hohe Dichte an Informationen. Nicht immer wird alles Gesagte aufgenommen oder bleibt die nötige Zeit, um Dinge zu wiederholen. Daher ist die Möglichkeit, Besprochenes über eine Aufzeichnung rekapitulieren zu können, ein gutes Mittel, um die Konsistenz von Wissen und Vorstellungen innerhalb von Arbeitsgruppen sicherzustellen. Manuelles Protokollieren ist dabei suboptimal, weil der Protokollant einen nicht unwesentlichen Teil seiner Aufmerksamkeit dieser Aufgabe widmen muss. Gleichzeitig muss er Informationen filtern, da ein wortwörtliches Festhalten des Gesagten in der Regel aus zeitlichen Gründen nicht möglich ist. Dieses Filtern ist jedoch stark subjektiv und manchmal sind es gerade auf ganz bestimmte Weise formulierte Aussagen, die einen Sachverhalt am besten beschreiben, aber durch das Filtern zerstört werden.

Die Assistance *minute taking* hat den Zweck das Protokollieren von Events von jeglicher kognitiver Arbeit zu befreien und nach dem Prinzip von FaF umzugestalten. Gleichzeitig soll ein exaktes und ungefiltertes Protokoll sowohl in auditiver als auch in textueller Form erstellt werden, sodass nachträglich über eine einfache Textsuche gezielt Begriffe wiedergefunden werden können.

Die inneren Mechanismen von *minute taking* wurden bereits größtenteils in Abschnitt 3.7 als allgemeines Beispiel für den Ablauf einer Assistance beschrieben. Daher wird an dieser

Stelle auf eine Wiederholung verzichtet. Gleiches gilt für die Darstellung des Ausführungsverhaltens, welches bereits in Abb. 3.8 zu sehen ist. Es sei jedoch noch zusätzlich angemerkt, dass als Ziel für das Hochladen der Audiodaten und des Transkripts die Projektablage einer Arbeitsgruppe verwendet wird. Auf diese Weise wird auch die Bereitstellung der Protokolle von SL übernommen und sie stehen jedem Mitglied der Gruppe zur Verfügung.

Die Parametrierung von *minute taking* umfasst die folgenden Werte:

- Sprache, welche in den Audiodaten gesprochen wird
- Ordner der Projektablage, in dem die Audiodaten und das Transkript abgelegt werden sollen
- Mikrofon-Aktor, der für die Aufnahme der Audiodaten verwendet werden soll

Für das Starten der Protokollierung müssen die Benutzer lediglich das Trigger-Signal *start event* für ihr spezifisches Event auslösen (die Mittel und Wege dazu sind in Abschnitt 3.10 beschrieben). Das Beenden der Protokollierung und die damit verbundene Weiterverarbeitung der aufgenommenen Daten kann zwar auch manuell über den Trigger *stop event* herbeigeführt werden, ist jedoch nicht zwingend notwendig. Sollte ein Event sein formales zeitliches Ende erreichen, so wird eine laufende Protokollierung beendet, indem neben dem Trigger-Signal *clean up event* auch automatisch *stop event* ausgelöst wird (siehe Abschnitt 3.6). Somit wird nur eine einzelne kurze Benutzerinteraktion zu Beginn benötigt und *minute taking* ist gemäß FaF benutzbar.

Automatisches Öffnen von Internetseiten (alias *website displaying*)

Während Meetings o.Ä. werden oft Informationen über Internetseiten geteilt. Diese können z.B. eine Präsentation enthalten oder sonstige für eine Arbeitsgruppe relevante Informationen. Statt sich solche Seiten vorzumerken und während eines Events manuell aufzurufen, kann die Assistance *website displaying* entsprechend konfiguriert werden. Anschließend übernimmt SL das Öffnen und Schließen der Internetseiten für die Dauer des dazugehörigen Events. Natürlich müssen für diese Assistance an einer Lokalität ein dedizierter Rechner mit Webbrower sowie ein Anzeigegerät vorhanden sein, welche von SL verwendet werden können.

Abbildung 3.9 zeigt das Ausführungsverhalten von *website displaying*. Die Ausführung der Assistance wird durch den formalen zeitlichen Rahmen eines Events vorgegeben (d.h. die Trigger-Signale *set up event* und *clean up event*). Während der Phase *begin* wird ein Webbrower geöffnet und die gewünschte URL (Uniform Resource Locator) angesteuert. Während der Phase *end* wird konsequenterweise eben diese Webbrower-Instanz auch wieder geschlossen. Dies passiert aber nur, falls die ursprünglich aufgerufene URL seitdem nicht verändert wurde. Damit wird verhindert, dass von den Benutzern manuell aufgerufene

Internetseiten, welche potenziell nicht gespeicherte Daten enthalten, z.B. eine unvollendete E-Mail, unerwünscht geschlossen werden. Die Phase *during* hat in *website displaying* keine Relevanz.

Assistance	Trigger	Phase	Action
<i>website displaying</i>	<i>set up event</i>	<i>begin</i>	<i>web browser opening</i>
	<i>clean up event</i>	<i>end</i>	<i>web browser closing</i>
	-	<i>during</i>	-

Abbildung 3.9: Die vollständige Definition des Ausführungsverhaltens der Assistance *website displaying*.

Die Parametrierung von *website displaying* umfasst die folgenden Werte:

- URL der Internetseite, die angezeigt werden soll
- Webbrowser-Aktor, welcher für das Öffnen der Internetseite verwendet werden soll
- Anzeigegerät-Aktor, der für das Darstellen der Internetseite verwendet werden soll

Der FaF-Charakter dieser Assistance entsteht, indem *website displaying* im Vorfeld eines Events für die gewünschte URL konfiguriert wird. Danach ist keine weitere Benutzerinteraktion mehr nötig.

Automatisches Öffnen von Dateien (alias *file displaying*)

Die Assistance *file displaying* ist das Gegenstück zu *website displaying*, um den Inhalt von Dateien (z.B. Präsentationen), welche über die Projektablage einer Arbeitsgruppe verfügbar sind, für die Dauer eines Events anzuzeigen. Auch hier werden ein dedizierter Rechner sowie ein Anzeigegerät an der Lokalität benötigt, die von SL verwendet werden können. Zudem muss auf diesem Rechner ein entsprechendes Programm zum Öffnen der Datei installiert sein, für das SL einen Software-Adapter besitzt. Dieser kapselt die Steuerung des Programms durch SL, z.B. über die Verwendung einer bestimmten Bibliothek.

Das Ausführungsverhalten von *file displaying* ist in Abb. 3.10 dargestellt. Der formale zeitliche Rahmen eines Events (d.h. die Trigger-Signale *set up event* und *clean up event*) steuert die Ausführung der Assistance. Während der Phase *begin* wird die anzuseigende Datei aus der Projektablage einer Arbeitsgruppe heruntergeladen und anschließend geöffnet.

In der Phase *end* wird die verwendete Programm-Instanz wiederum geschlossen. Die Phase *during* hat für den Ablauf keine Bedeutung.

Assistance	Trigger	Phase	Action
<i>file displaying</i>	<i>set up event</i>	<i>begin</i>	<i>data download</i>
			<i>file opening</i>
	<i>clean up event</i>	<i>end</i>	<i>file closing</i>
	-	<i>during</i>	-

Abbildung 3.10: Die vollständige Definition des Ausführungsverhaltens der Assistance *file displaying*.

Die Parametrierung von *file displaying* umfasst die folgenden Werte:

- Verzeichnis der zu öffnenden Datei in der Projektablage
- Programm-Aktor, welcher für das Öffnen der Datei verwendet werden soll
- Anzeigegerät-Aktor, der für das Darstellen der geöffneten Datei verwendet werden soll

Wie bei *website displaying* entsteht der FaF-Charakter dieser Assistance dadurch, dass sie im Vorfeld eines Events für die gewünschte Datei konfiguriert werden kann und danach keine weitere Benutzerinteraktion mehr nötig ist. Es ist dabei lediglich zu beachten, dass *file displaying* auf das Öffnen von Dateien aus der Projektablage einer Arbeitsgruppe beschränkt ist.

Anzeigen der Agenda eines Events (alias *agenda showing*)

Wie die genaue Agenda eines Events aussieht, steht nicht zwangsläufig schon weit im Voraus fest. Im Laufe der Zeit können neue Punkte für die Tagesordnung entstehen, weil Mitglieder einer Arbeitsgruppe beispielsweise neue Informationen in Erfahrung bringen, welche sie teilen wollen, oder zusätzlichen Klärungsbedarf entwickeln. In SL enthält jedes Event eine Agenda (siehe Abb. 3.4), die als eine Art verteilte To-do-Liste verwendet und im Vorfeld eines Events flexibel mit textuellen Inhalten befüllt werden kann.

Die vollständige Agenda eines Events kann über SL in Form einer Internetseite aufgerufen werden. Die Assistance *agenda showing* sorgt dafür, dass die aktuelle Version dieser Seite

automatisch während eines Events angezeigt wird. Damit existiert gleich zu Beginn ein optischer Überblick über den voraussichtlichen Umfang eines Events, welcher eine eventuell notwendige Verlängerung oder Umschichtungen auf andere Events sichtbar macht.

Abbildung 3.11 zeigt das Ausführungsverhalten von *agenda showing*. Es entspricht dem von *website displaying*, da hier ebenso eine Internetseite im Webbrowser geöffnet wird (jedoch eine, die von SL selbst bereitgestellt wird). Dies macht den Nutzen der zusätzlichen Abstraktionsschicht in Form von Actions deutlich: Sie sind generisch genug, sodass sie von verschiedenen Assurances verwendet werden können.

Assistance	Trigger	Phase	Action
<i>agenda showing</i>	<i>set up event</i>	<i>begin</i>	<i>web browser opening</i>
	<i>clean up event</i>	<i>end</i>	<i>web browser closing</i>
	-	<i>during</i>	-

Abbildung 3.11: Die vollständige Definition des Ausführungsverhaltens der Assistance *agenda showing*.

Die Parametrierung von *agenda showing* umfasst die folgenden Werte:

- Webbrowser-Aktor, welcher für das Öffnen der Agenda-Übersicht verwendet werden soll
- Anzeigegerät-Aktor, der für das Darstellen der Agenda-Übersicht verwendet werden soll

Die Assistance *agenda showing* ist gemäß FaF nutzbar, weil zusätzliche neu aufkommende Agendapunkte direkt in die Konfiguration eines Events (im Sinne von Abschnitt 3.2) eingefügt werden. Danach bedarf es keiner weiteren Benutzerinteraktion mehr. Das bedeutet, dass es nicht nötig ist, die Agendapunkte während eines Events wieder aktiv nachzuschlagen, so wie es der Fall wäre, wenn sie in einem anderen Medium wie z.B. einem Notizbuch vermerkt worden wären. Zusätzlich haben auf diese Weise alle Mitglieder einer Arbeitsgruppe Einsicht in den momentanen Status einer Event-Agenda.

Automatisches Vorbereiten von Geräten (alias *device preparation*)

An Lokalitäten können Geräte vorhanden sein, welche während eines Events zum Einsatz kommen sollen (z.B. ein Beamer in einem Meetingraum). Damit Benutzer diese Geräte

nicht einzeln manuell ein- und ausschalten müssen und Geräte mit einer langen Anlaufzeit schon zu Beginn eines Events bereitstehen, kann deren Aktivierung bzw. Deaktivierung über die Assistance *device preparation* von SL übernommen werden.

Für die Ansteuerung spezifischer Geräte wird ein entsprechender Software-Adapter vorausgesetzt, der die Kommunikation zwischen SL und dem Gerät kapselt (z.B. über Infrarot-Signale oder über eine Netzwerkschnittstelle). Von *device preparation* selbst werden ansonsten keinerlei technische Anforderung an ein solches Gerät gestellt. Es muss lediglich auf irgendeinem Wege ein- und ausschaltbar sein.

In Abb. 3.12 ist das Ausführungsverhalten von *device preparation* zu sehen. Der Ablauf der Assistance wird über den formalen zeitlichen Rahmen eines Events (d.h. die Trigger-Signale *set up event* und *clean up event*) gesteuert. Während der Phase *begin* wird das gewünschte Gerät eingeschaltet und während der Phase *end* wieder ausgeschaltet. Die Phase *during* besitzt in *device preparation* keine Relevanz.

Assistance	Trigger	Phase	Action
<i>device preparation</i>	<i>set up event</i>	<i>begin</i>	<i>device activation</i>
	<i>clean up event</i>	<i>end</i>	<i>device deactivation</i>
	-	<i>during</i>	-

Abbildung 3.12: Die vollständige Definition des Ausführungsverhaltens der Assistance *device preparation*.

Die Parametrierung von *device preparation* umfasst die folgenden Werte:

- Aktor, welcher ein- und ausgeschaltet werden soll

Der FaF-Charakter dieser Assistance entsteht dadurch, dass sie im Vorfeld eines Events für das gewünschte Gerät konfiguriert werden kann und danach keine weitere Benutzerinteraktion mehr nötig ist.

3.7.2. Weitere Anwendungsfälle für Assistanzen

Im Folgenden werden einige weitere Anwendungsfälle für Assistanzen beschrieben, welche während dieser Abschlussarbeit identifiziert wurden. Für die erste Iteration von SL wurde jedoch von ihrer detaillierten Konzeption und Umsetzung abgesehen, um den Umfang des Projekts einzuschränken. Daher sind ihre Beschreibungen teilweise wesentlich kürzer gehalten

als diejenigen der Assurances aus Abschnitt 3.7.1. Diese Anwendungsfälle können jedoch trotzdem als Grundlage für zukünftige Weiterentwicklungen von SL dienen.

Automatisches Digitalisieren und Verteilen von Whiteboard-Aufschrieben

Analoge Medien wie Whiteboards sind auch in unserer heutigen Zeit nicht aus Arbeitsumfeldern wegzudenken. Eine Einbindung in moderne digitale Prozesse ist jedoch quasi nicht vorhanden. Meist werden Aufschriebe mit dem Smartphone abfotografiert und per E-Mail oder Chat-Client verteilt.

SL hat den Anspruch hochgradig interoperabel zu sein (siehe Abschnitt 3.4.2), was auch die Anbindung analoger Medien mit einschließt. Im Fall von Whiteboards kann dies über eine Assistance erreicht werden, die auf eine Benutzerinteraktion reagiert (z.B. das Drücken eines IoT-Knopfs). Die Reaktion könnte dergestalt sein, dass eine fix angebrachte Netzwerkamera das Whiteboard fotografiert. Anschließend wird das Foto von SL in die Projektionslage einer Arbeitsgruppe hochgeladen. Das bedeutet, dass nach der Benutzerinteraktion das Whiteboard direkt gesäubert und weiterverwendet werden kann, ohne dass noch weitere manuelle Schritte unternommen werden müssen. Die Assistance läuft also gemäß FaF ab.

Abbildung 3.13 zeigt wie das Ausführungsverhalten einer solchen Assistance, die für den Rest dieses Abschnitts als *image preservation* bezeichnet wird, aussehen könnte. Das Sichern des Whiteboard-Inhalts hängt in keiner Weise vom zeitlichen Rahmen eines Events ab und kann beliebig oft sowie jederzeit während eines Events angefordert werden. Daher besitzen die Phasen *begin* und *end* keine Bedeutung und die Ausführungslogik spielt sich stattdessen während *during* ab. Um das in dieser Phase enthaltene Verhalten zu realisieren, ist einerseits eine neue Action *image acquisition* nötig, welche das Aufnehmen des Fotos steuert. Andererseits ist auch eine Erweiterung der in Abschnitt 3.6 beschriebenen Trigger um ein neues Signal notwendig. Dieses Signal (hier als *user request* bezeichnet) steht stellvertretend für eine beliebig geartete Benutzerinteraktion, mit der eine Reaktion von SL angefordert werden kann. Alternativ ist auch ein weniger generisches Signal denkbar, das mehr auf den Anwendungsfall von *image preservation* ausgerichtet ist. Die Sinnhaftigkeit beider Varianten ist stark von sonstigen zukünftigen Weiterentwicklungen von SL abhängig. Daher sollte sie auch zu gegebener Zeit im entsprechenden Kontext noch einmal neu bewertet werden.

Natürlich ist der Nutzen von *image preservation* auch abseits von Whiteboards anwendbar. Diese sollten hier nur als leicht verständliches Beispiel dienen. Prinzipiell kann die Assistance auf alles angewandt werden, dessen Aussehen von Zeit zu Zeit auf Abruf festgehalten werden soll.

Assistance	Trigger	Phase	Action
<i>image preservation</i>	-	<i>begin</i>	-
	-	<i>end</i>	-
	<i>user request</i>	<i>during</i>	<i>image acquisition</i>
			<i>data upload</i>

Abbildung 3.13: Die vollständige Definition des Ausführungsverhaltens der Assistance *image preservation*.

Automatische Zugangskontrolle von Räumen

In manchen Arbeitsumfeldern (z.B. Bildungseinrichtungen) kann es erwünscht sein, Räume nur für bestimmte Personen zugänglich zu machen. Dies kann z.B. zur Vermeidung von Störungen oder der Nachvollziehbarkeit der Raum-Nutzung in Schadensfällen dienen. SL könnte mit einer entsprechenden Assistance dafür sorgen, dass ein Raum für die Dauer eines Events nur von Personen betretbar ist, die in der Konfiguration der Assistance angegeben wurden (z.B. die Mitglieder einer Arbeitsgruppe). Voraussetzung hierfür ist, dass in einem Arbeitsumfeld ein entsprechendes System für die Steuerung der Zugangskontrolle existiert, welches zudem ein API besitzt, das von SL angesprochen werden kann.

Automatische benutzerorientierte Einrichtung von Arbeitsplätzen

Elektrische Arbeitsplatz-Ausstattung, welche sich vom Benutzer an seine Bedürfnisse anpassen lässt (wie z.B. höhenverstellbare Tische), sind heutzutage keine Seltenheit mehr. Dass sich solche Ausstattungselemente auch durch andere Systeme von außen ansteuern lassen, ist jedoch eher unüblich, aber durchaus denkbar. Derartige Elemente könnten durch eine entsprechende Assistance von SL für bestimmte Personen eingestellt werden, sodass diese z.B. während eines Meetings eine für sie vorbereitete Umgebung vorfinden.

Automatisches Lüften und Abdunkeln von Räumen

Klassische Aufgaben von Gebäudeautomationssystemen sind das Öffnen, Schließen und Abdunkeln von Fenstern. Wenn eine Kompatibilität zwischen SL und solchen Systemen hergestellt werden kann, wären Assistanzen realisierbar, die diese Steuerungsaufgaben einbeziehen können. Denkbar ist eine Assistance, welche die Fenster eine gewisse Zeit vor einem Event öffnet, um den Raum bis zum Start des Events zu lüften. Dies würde

konsequenterweise ein neues Trigger-Signal erforderlich machen, das *vor* dem formalen zeitlichen Anfang eines Events gesendet wird. Während eines Events könnte eine solche Assistance auf einen Trigger reagieren, welcher von einem CO₂-Sensor stammt und so den Raum den Bedürfnissen entsprechend lüften. Zusätzlich könnte der Raum abgedunkelt werden, falls im Event über eine andere Assistance wie *file displaying* oder *website displaying* (siehe Abschnitt 3.7.1) eine Präsentation hinterlegt wurde.

3.8. Actions

Actions sind die Bausteine, aus denen sich die Ausführungslogik von Assurances zusammensetzt. Sie sind fachlich (aber keinesfalls technisch) atomar. Das bedeutet, dass sie aus der Sicht einer Person, die sie manuell ausführen würde, nicht in weitere sinnvolle Tätigkeiten unterteilt werden können. Actions stellen generische wiederverwendbare Tätigkeiten dar und sind so weit abstrahiert, dass sie nicht an die Verwendung eines bestimmten Aktors gebunden sind. Die Action *audio recording start* dient etwa zur Aktivierung der Aufnahme von Audiodaten, wobei das zu verwendende Mikrofon in der Action selbst nicht festgelegt ist. Stattdessen muss der konkrete Aktor, über welchen eine Action ihre Funktionalität erfüllen soll, als Parameter spezifiziert werden.

Actions kommen in den verschiedenen Phasen von Assurances zum Einsatz. Genau genommen sind die Phasen nichts anderes als eine Hintereinanderschaltung von Actions, deren Inputs und Outputs gekoppelt sein können. Abbildung 3.14 zeigt wie die Actions der Phase *end* von *minute taking* (siehe Abb. 3.8) miteinander verknüpft sind.

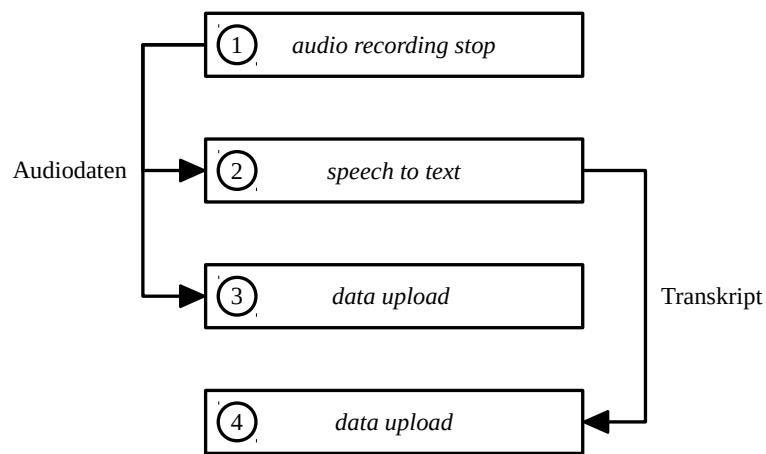


Abbildung 3.14: Die Kopplung der Actions in der Phase *end* der Assistance *minute taking*. Die Inputs und Outputs der Actions sind dabei über Pfeile dargestellt und die Reihenfolge ihrer Ausführung über Zahlen.

Im Gegensatz zu Assurances besitzen Actions keine Kontextinformationen und bestehen auch nicht aus Phasen. In den meisten Fällen stellen sie zeitlich punktuelle Vorgänge dar. Darüber hinaus wissen Actions auch nicht, in welcher Assurance sie ablaufen. Doch auch wenn Actions keinen eigenen Kontext besitzen, so müssen sie dennoch parametrisiert werden. Die Action *web browser opening* benötigt z.B. eine URL, welche beim Öffnen des Webbrowsers angesteuert werden soll. Diese Parameter werden von den ausführenden Assurances bereitgestellt, die sie wiederum selbst entweder aus ihrer eigenen Konfiguration oder ihren Kontextinformationen beziehen.

Im Folgenden ist die fachliche Seite aller Actions beschrieben, welche im Rahmen dieser Abschlussarbeit umgesetzt wurden. Details zu deren Implementierung sind in Abschnitt 4.8 zu finden.

audio recording start Startet die Aufnahme von Audiodaten.

Input:

- ID des zu verwendenden Mikrofon-Aktors

Output: -

audio recording stop Stoppt die Aufnahme von Audiodaten.

Input:

- ID des zu verwendenden Mikrofon-Aktors

Output: Audiodatei

data upload Überträgt Daten an eine bestimmte Projektabbage.

Input:

- Metainformationen zur Projektabbage
- Ordner, in dem die hochzuladende Datei abgelegt werden soll
- Name der hochzuladenden Datei
- Nachricht (wird z.B. als Commit-Nachricht verwendet, falls die Projektabbage eine Versionsverwaltung ist)
- Hochzuladende Daten

Output: -

data download Lädt Daten von einer bestimmten Projektabbage herunter.

Input:

- Metainformationen zur Projektabbage
- Verzeichnis der herunterzuladenden Datei

Output: Heruntergeladene Datei

device activation Schaltet ein bestimmtes Gerät ein.

Input:

- ID des einzuschaltenden Aktors

Output: -

device deactivation Schaltet ein bestimmtes Gerät aus.

Input:

- ID des auszuschaltenden Aktors

Output: -

file opening Öffnet eine Datei mit einem bestimmten Programm.

Input:

- ID des zu verwendenden Programm-Aktors
- ID des Anzeigegerät-Aktors, auf dem die zu öffnende Datei dargestellt werden soll
- Zu öffnende Datei

Output: ID der geöffneten Programm-Instanz

file closing Schließt eine geöffnete Datei.

Input:

- ID des zu verwendenden Programm-Aktors
- ID der zu schließenden Programm-Instanz

Output: -

speech to text Wandelt Audiodaten in ein textuelles Transkript um.

Input:

- Umzuwendende Audiodatei
- In der Audiodatei gesprochene Sprache

Output: Textuelles Transkript

web browser opening Öffnet einen Webbrowser und steuert eine bestimmte URL an.

Input:

- ID des zu verwendenden Webbrowser-Aktors
- ID des Anzeigegerät-Aktors, auf dem der zu öffnende Webbrowser dargestellt werden soll
- Zu öffnende URL

Output: ID der geöffneten Webbrowser-Instanz

web browser closing Schließt einen geöffneten Webbrowser⁴.

Input:

- ID des zu verwendenden Webbrowser-Aktors
- ID der zu schließenden Webbrowser-Instanz

Output: -

3.9. Aktoren

Aktoren repräsentieren jegliche Elemente, die von SL angesprochen werden können, um einen konkreten Effekt zu erzielen. Dies können einerseits physische Geräte sein (Mikrofone, Beamer etc.), andererseits können sie jedoch auch virtueller Natur sein (lokal installierte Programme, Webservices etc.). Damit SL Aktoren ansteuern kann, werden entsprechende Software-Adapter benötigt, welche die dazu notwendige Art der Kommunikation kapseln (z.B. über eine Netzwerkschnittstelle, Infrarot-Signale, eine Programmbibliothek etc.). Actions verwenden je nach Parametrierung entsprechende Adapter, um ihre jeweilige Aufgabe zu erfüllen.

Die Benutzung von Aktoren ist limitiert auf solche, deren Verfügbarkeit in einer entsprechenden Datenquelle hinterlegt wurde (siehe Abb. 3.5). Auf diese Weise liegt die Entscheidungsgewalt über die Nutzung von Aktoren beim Systemadministrator und nicht bei den Benutzern. So können z.B. in der Assistance *file opening* (siehe Abschnitt 3.7.1) lediglich Programme zum Öffnen verwendet werden, welche auch als Aktor registriert worden sind. Die folgenden Informationen müssen beim Anlegen eines Aktors angegeben werden.

Typ Der Typ eines Aktors bezeichnet seine spezifische Ausführung. Bei physischen Aktoren wie Mikrofonen ist dies die Modellbezeichnung des Geräts, bei virtuellen Aktoren wie

⁴Dabei werden jedoch nur Webbrowsers-Tabs geschlossen, welche zuvor mit *web browser opening* geöffnet und seitdem nicht verändert worden sind. Für andere Zwecke weiterverwendete oder manuell geöffnete Tabs werden nicht geschlossen, damit eventuell nicht gespeicherte Arbeit nicht verloren geht.

Webbrowsern der Name des Programms (z.B. *firefox*). Der Typ entschiedet darüber, welcher Software-Adapter für die Ansteuerung eines Aktors zum Einsatz kommt.

Name Der Name eines Aktors hat keinerlei Relevanz für die Ausführungslogik von SL. Über ihn kann jedoch eine von Menschen lesbare Bezeichnung hinterlegt werden (z.B. *Rechter Beamer in Konferenzraum 3*). Ein solcher Name erleichtert die Identifikation von Aktoren in GUIs o.Ä. erheblich.

Verantwortlicher Delegat Das Konzept von *Delegaten* ermöglicht es SL, auch Aktoren über das Netzwerk anzusteuern, die eigentlich nur eine lokale Schnittstelle besitzen. Auf das Prinzip von Delegaten wird in Abschnitt 3.12 näher eingegangen. An dieser Stelle reicht es zu wissen, dass die Information über den verantwortlichen Delegaten eines ausschließlich lokal ansteuerbaren Aktors nötig ist, damit SL diesen Aktor auch verwenden kann.

SL kann prinzipiell an eine bestehende Datenquelle für die Verwaltung der verfügbaren Aktoren angebunden werden (wie bei allen anderen Datenquellen auch). Da diese Informationen jedoch SL-spezifisch sind, ist es wahrscheinlicher, dass eine solche Datenquelle zusammen mit SL neu in eine Systemlandschaft eingebracht werden muss.

3.10. Interaktion mit dem System

Wie in Abschnitt 3.4.4 beschrieben wird in SL eine Bedienung nach dem Prinzip von FaF angestrebt. Dies bedeutet, dass Benutzerinteraktionen zwar möglichst selten und kurz gehalten, aber dennoch vorhanden sind. Daher benötigt das System auch Schnittstellen, über die Benutzer mit ihm interagieren können. In der ersten Iteration von SL umfassen diese Interaktionen folgendes:

- Anlegen und Konfigurieren von Events und den gewünschten Assurances (im Sinne von Abschnitt 3.2)
- Generieren der Trigger-Signale *start event* und *stop event* (siehe Abschnitt 3.6)
- Verlängern von laufenden Events

Die Durchführung von Ersterem kann variieren und ist abhängig davon, welche Art von Datenquelle für Events verwendet wird (siehe Abb. 3.5). Kommt beispielsweise ein Kalenderdienst zum Einsatz, so können Benutzer dessen GUI nutzen, um Events anzulegen und Assurances einzurichten.

Für das Generieren der Trigger und das Verlängern von Events bietet SL eine spezielle Weboberfläche an. Diese dient daneben auch als Statusübersicht für das aktuell laufende

Event einer bestimmten Lokalität. Das Konzept von SL sieht vor, dass jede an SL angebundene Lokalität ein Anzeigegerät besitzt, welches *permanent* den Event-Status ebendieser Lokalität anzeigt. Die Anzeige sollte für keinen weiteren Zweck verwendet werden, damit Benutzer zu jeder Zeit Zugriff auf die Metainformationen eines eventuell gerade laufenden Events haben und dieses kontrollieren können. Als Gerät für die Anzeige würde sich z.B. ein über Toucheingaben gesteuerter Tablet-PC anbieten, der neben der Eingangstür angebracht ist oder an einer zentralen Stelle der Lokalität griffbereit liegt.

Abbildung 3.15 zeigt einen Entwurf der Status-Oberfläche. Sie enthält grundlegende Metainformationen wie den Titel eines Events und dessen verbleibende Dauer. Daneben bietet sie Schaltflächen zum Starten und Beenden eines Events, welche die Trigger-Signale *start event* bzw. *stop event* (siehe Abschnitt 3.6) für die entsprechende Lokalität senden. Weil Meetings o.Ä. oft länger dauern als ursprünglich angenommen, ist auch eine Schaltfläche für das einfache und flexible Verlängern eines gerade laufenden Events vorhanden. Sollte gerade kein Event an einer Lokalität stattfinden, spiegelt sich dies auch in der Status-Oberfläche wider und es werden entsprechend keine Informationen und Schaltflächen eingeblendet.

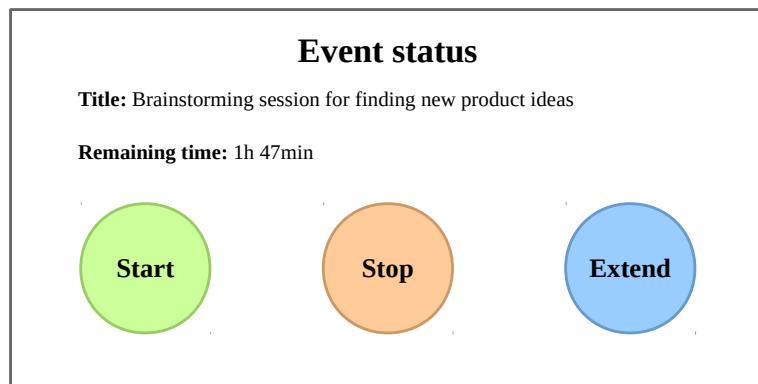


Abbildung 3.15: Ein Entwurf der Weboberfläche für den Event-Status einer Lokalität.

Für alle weiteren Interaktionen mit dem System existieren in der ersten Iteration von SL keine dedizierten Oberflächen. Stattdessen muss direkt das vom System angebotene API verwendet werden, welches in Abschnitt 3.13 näher beschrieben ist. Dies betrifft insbesondere auch die Interaktion mit Datenquellen, wobei natürlich deren GUI verwendet werden kann, falls sie bereits über eine solche verfügen (z.B. bei Verwendung eines webbasierten Kalenderdiensts als Event-Datenquelle).

3.11. Ablauf der Systemlogik

Dieses Kapitel vereint die Konzepte der vorigen Abschnitte zu einem ganzheitlichen Bild der fachlichen Ausführungslogik von SL. Abbildung 3.16 zeigt den Ablauf von der Konfiguration eines Events durch den Benutzer bis hin zur Unterstützung, welche der Benutzer durch SL erfährt.

Der Prozess beginnt mit dem Anlegen und Konfigurieren eines Events durch den Benutzer inklusive dem Festlegen der gewünschten Assurances (im Sinne von Abschnitt 3.2). Der genaue Vorgang ist dabei abhängig von der verwendeten Event-Datenquelle. Bei einem Kalenderdienst kann z.B. dessen Weboberfläche dafür verwendet werden. Die Konfiguration der Assurances erfolgt über eine speziell hierfür definierte Sprache und ist den Möglichkeiten der Datenquelle entsprechend in sie eingebettet (was in Abschnitt 4.6 näher beschrieben ist).

Sobald das Event begonnen hat, werden für dieses entsprechende Trigger gesendet. In der ersten Iteration von SL können das entweder die zeitgesteuerten Signale *set up event* und *clean up event* sein (welche automatisch von SL bereitgestellt werden) oder die benutzergesteuerten Signale *start event* und *stop event* (deren Senden eine Benutzerinteraktion erfordert). Von SL empfangene Trigger-Signale werden an alle konfigurierten Assurances eines Events weitergeleitet. Zusätzlich wird für jede dieser Assurances ein entsprechendes Paket aus Kontextinformationen zusammengebaut, auf welches sie beim Bereitstellen ihrer Funktionalität zurückgreifen kann.

Sollte eine Assurance auf ein Trigger-Signal reagieren (was nicht unbedingt der Fall sein muss), wird die Ausführung einer ihrer Phasen angestoßen. Weil die Ausführung dieser Phasen unter Umständen sehr lange dauern kann (z.B. wenn sie die Umwandlung von Audiodaten in ein textuelles Transkript involvieren), werden sie asynchron abgearbeitet, damit sie sich nicht gegenseitig blockieren. Während der Durchführung einer solchen Phase werden die in ihr enthaltenen Actions nacheinander abgearbeitet. Die synchrone Ausführung ist notwendig, da die Actions prinzipiell miteinander gekoppelt sein können (d.h. der Output einer Action ist gleichzeitig der Input für eine andere). Actions sind abhängig von Aktoren, welche ihre Funktionalität letztlich bereitstellen. Daher müssen für die spezifischen Aktoren, deren Verwendung in einer Assurance konfiguriert wurde, die entsprechenden Software-Adapter aufgelöst werden. Mit diesen können die Aktoren dann schließlich von SL angesteuert und zum Einsatz gebracht werden.

Während die in den vorangehenden Absätzen beschriebenen Vorgänge asynchron ablaufen, profitiert der Benutzer für die Dauer seines Events (und je nach Assurance auch darüber hinaus) von ihren Effekten. Der gesamte in Abb. 3.16 dargestellte Ablauf kann natürlich beliebig wiederholt werden und ist auf weitere Events anwendbar.

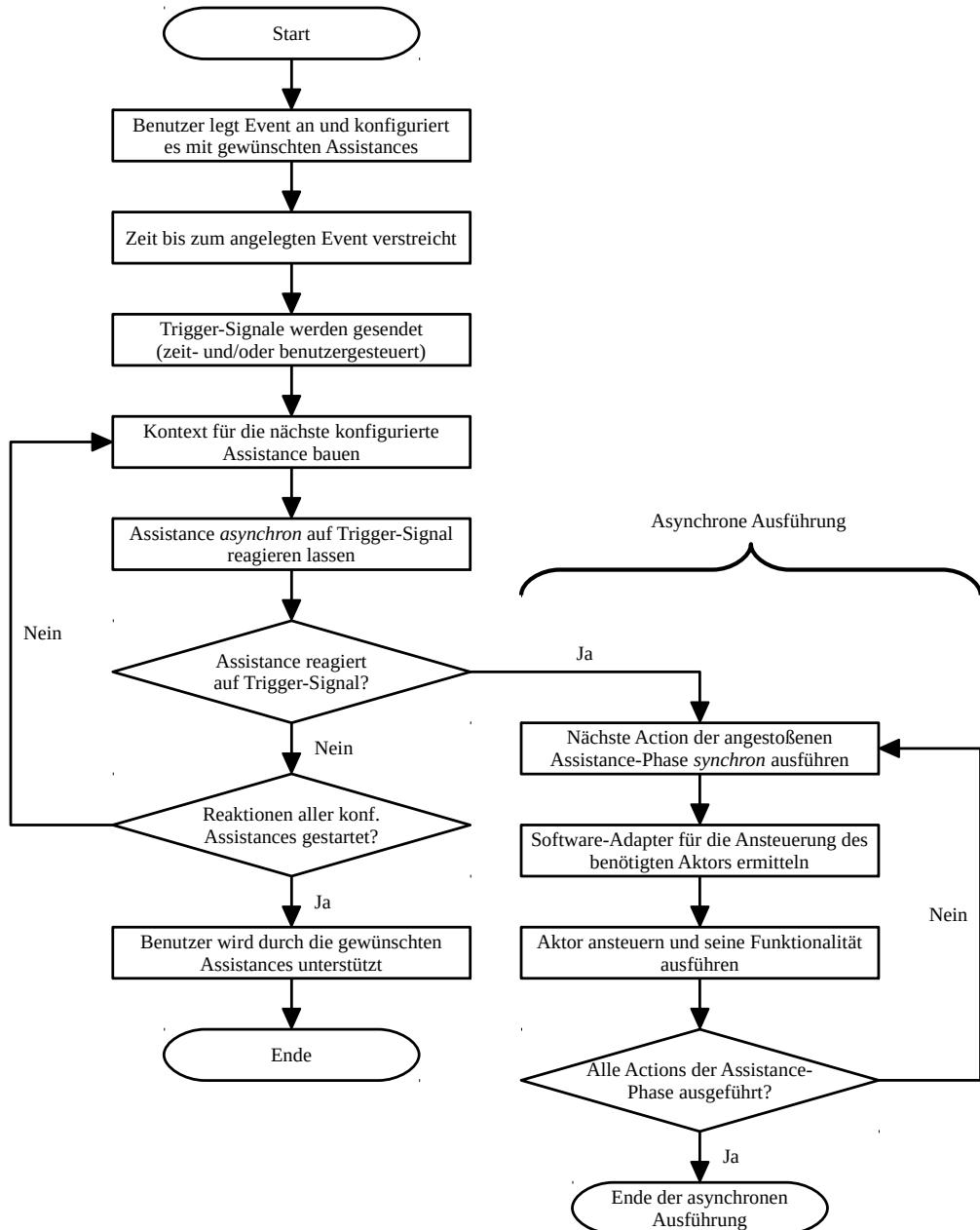


Abbildung 3.16: Der Ablauf der fachlichen Ausführungslogik von SL. Der rechte Zweig des Diagramms stellt dabei eine asynchrone Ausführung dar, die unabhängig vom Hauptast abläuft.

3.12. Architektur

SL baut auf einer serviceorientierten Architektur auf, die in Abb. 3.17 zu sehen ist. Um eine Orientierung in dem zu dieser Abschlussarbeit gehörenden Quellcode zu erleichtern, wurden

die Komponenten in Abb. 3.17 mit ihren englischen Namen betitelt. Die Zuordnung zu den bisher verwendeten deutschen Begriffen ist jedoch leicht ersichtlich (z.B. steht *actuator* für Aktor, *location* für Lokalität etc.). In den folgenden Absätzen werden die Rollen, welche die einzelnen Services in dem Vorgang aus Abb. 3.16 spielen, näher erläutert. Dabei soll dem logischen Ablauf vom Senden der Trigger-Signale bis zum letztendlichen Ansteuern der Akteure gefolgt werden.

Das Bereitstellen der Trigger wird von zwei Services bewerkstelligt. Zum einen werden automatisch generierte Signale (d.h. die in Abschnitt 3.6 beschriebenen zeitgesteuerten Trigger) vom *trigger provider service* versendet. Dieser erfordert keinerlei Zutun von außen und besitzt auch kein eigenes API. Zum anderen können Benutzer die in Abschnitt 3.10 beschriebene Status-Oberfläche eines Events nutzen, um Signale händisch zu versenden (d.h. die in Abschnitt 3.6 beschriebenen benutzergesteuerten Trigger). Diese Oberfläche wird in Form einer Internetseite vom *GUI service* ausgeliefert, genau wie alle anderen von SL bereitgestellten Seiten. Dazu gehört in der ersten Iteration von SL lediglich noch die in Abschnitt 3.7.1 beschriebene Agenda-Übersicht von Events. An dieser Stelle soll noch hervorgehoben werden, dass der *GUI service* nicht selbst Trigger-Signale versendet, sondern lediglich eine Internetseite ausliefert, die dies kann. Daher besitzt er in Abb. 3.17 auch keine entsprechende Verbindungsleitung wie der *trigger provider service*.

Zielpunkt für die gesendeten Trigger-Signale ist der *trigger service*. Er nimmt die Signale entgegen und ermittelt welche Assurances auf sie reagieren. Für jene Assurances baut er anschließend einen Assistance-Kontext zusammen, wofür er auf fünf verschiedene Datenmanagement-Services zugreift. Diese verwalten den Zugriff auf Informationen über Akteure, Lokalitäten, Arbeitsgruppen, Personen und Events. Jeder der fünf Services ist jeweils an eine Datenquelle angebunden, was z.B. ein Kalenderdienst, ein System für das Identitätsmanagement, eine simple Datenbank etc. sein kann (siehe Abschnitt 3.5). Die Ausführung jeder Assurance, die auf ein Trigger-Signal reagiert, wird vom *trigger service* angestoßen, indem er den *assurance service* aufruft. Da der *trigger service* asynchron arbeitet, vermerkt er die parallele Ausführung der Assurances und deren Status im *job service*. Dieser dient als Nachschlagewerk für den Fortschritt von asynchronen Tätigkeiten im System, was in der ersten Iteration von SL aber lediglich vom *trigger service* genutzt wird. Mit seiner Hilfe können z.B. potentielle externe Systeme, welche an SL angebunden sind und Trigger senden, den Status der Abarbeitung der Reaktionen auf die Signale mitverfolgen.

Der *assurance service* ist verantwortlich für die Durchführung einzelner Assurance-Phasen. Erhält er einen Aufruf, so führt er die gewünschte Phase mit den gelieferten Kontextinformationen aus. Dabei löst er auf, welche Actions benötigt werden und fordert seinerseits deren Ausführung vom *action service* an. Dieser ist für die Durchführung einzelner Actions zuständig. Dazu ermittelt er den genauen Typ eines Aktors (siehe Ab-

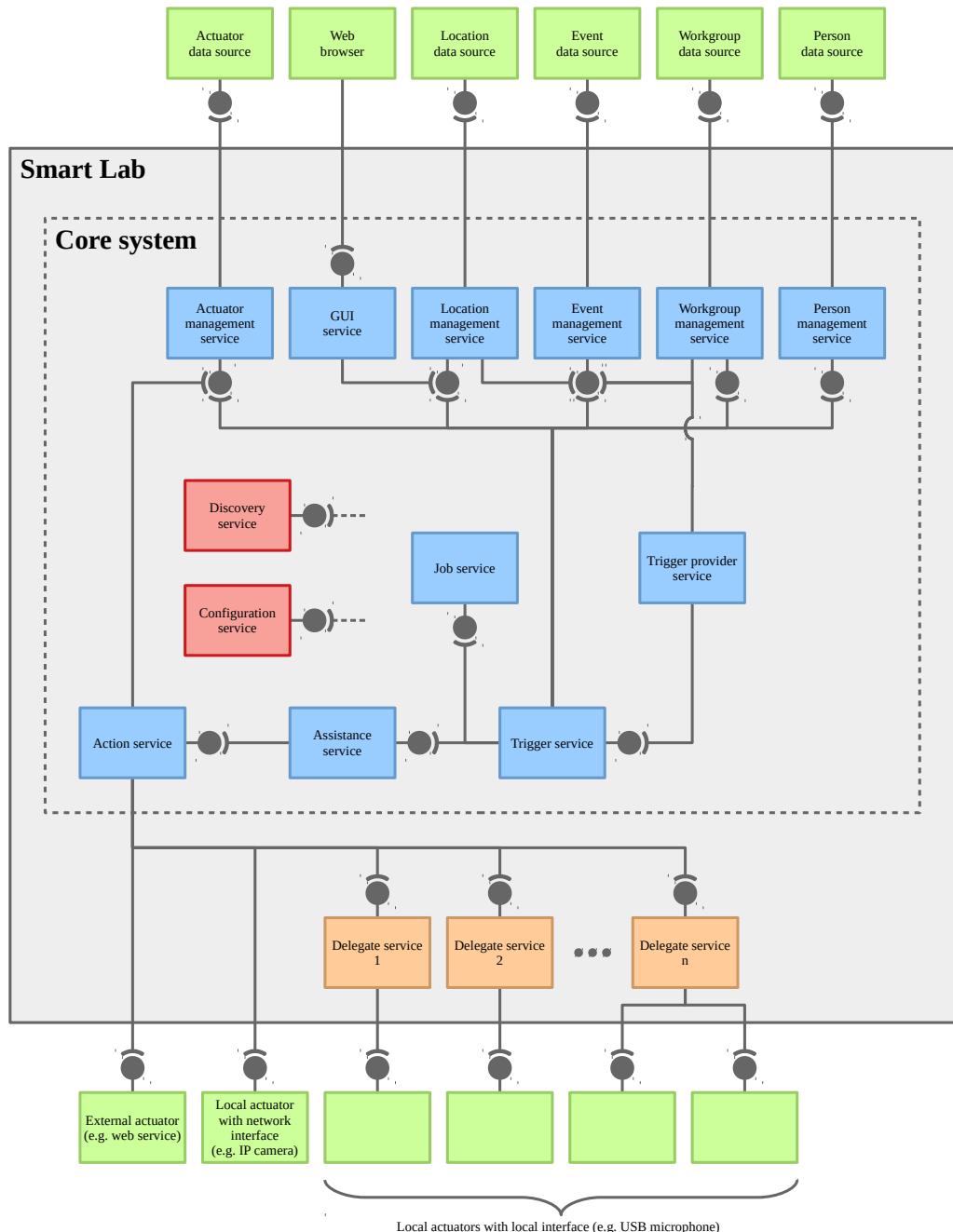


Abbildung 3.17: SL basiert auf einer serviceorientierten Architektur. Blaue Komponenten realisieren die fachlichen Prozesse von SL und gehören zu dessen Kernsystem. Die roten Komponenten sind ebenfalls dem Kernsystem zugeordnet, erfüllen aber rein technische Zwecke und sind deshalb auch an alle anderen Services angebunden (ihre Verbindungslien wurden daher der Übersichtlichkeit halber weggelassen). Orangene Komponenten gehören zwar zu SL, sind jedoch nicht Teil des Kernsystems. Grüne Komponenten sind nicht Bestandteil von SL, aber an das System angebunden.

schnitt 3.9), welcher im *actuator management service* hinterlegt ist. Ist ein Aktor per Fernzugriff erreichbar (z.B. ein Webservice oder ein Gerät mit einer Netzwerkschnittstelle), so wird er direkt vom *action service* angesteuert. Weil das Kernsystem von SL für die Ausführung auf einem zentralen Server vorgesehen ist, wird für die Kommunikation mit Aktoren, die lediglich eine lokale Schnittstelle besitzen (z.B. ein USB-Mikrofon), eine zusätzliche Software-Schicht benötigt⁵. Diese Schicht besteht aus dem *delegate service*, welcher als Mittelsmann für SL dient, um derlei Aktoren anzusteuern. Dabei können beliebig viele Instanzen des *delegate service* verwendet werden, wobei wiederum beliebig viele Akto- ren von einem einzelnen Delegaten verwaltet werden können. Die einzige Voraussetzung ist, dass die Delegaten und ihre jeweiligen Akto- ren auf bzw. an demselben Rechner betrieben werden. Das heißt, dass Instanzen des *delegate service* auf Computern ausgeführt werden, welche zusammen mit ihren angeschlossenen Akto- ren fest zur Ausstattung einer Lokalität gehören und auch dort positioniert sind. Das Kernsystem von SL hingegen ist wie bereits erwähnt für die Ausführung auf einem zentralen Server gedacht.

Der *discovery service* und der *configuration service* erfüllen einen rein technischen Nutzen. Ersterer sorgt dafür, dass alle anderen Dienste miteinander kommunizieren können, indem er als Nachschlagewerk für deren Netzwerkadressen fungiert. Letzterer ist eine zentrale Anlaufstelle, von der alle anderen Dienste ihre jeweiligen Konfigurationen beziehen können. Prinzipiell sind beide Services obligatorisch für den Betrieb von SL. Das System kann jedoch auch als Monolith betrieben werden (Details hierzu sind in Abschnitt 4.2 zu finden). Dadurch verschmelzen die in blau dargestellten Komponenten des Kernsystems (siehe Abb. 3.17) zu einer einzelnen Anwendung. In diesem Fall ist der *discovery service* dann nicht mehr notwendig.

In der ersten Iteration besitzt SL kein Gateway, das als zentraler Zugriffspunkt auf die Services des Systems dient. Daher kann jeder der Services von außen gleichermaßen ohne Beschränkung verwendet werden, selbst solche mit Funktionalität, welche eigentlich nur systeminternen Zwecken dient.

3.13. API

Wie in Abschnitt 3.12 beschrieben basiert SL auf einer serviceorientierten Architektur und bietet damit auch eine Reihe von APIs nach außen an. Ein Ziel von SL ist es, möglichst interoperabel zu anderen Systemen zu sein (siehe Abschnitt 3.4.2). Daher sind die Schnittstellen der Services als plattform- und programmiersprachenunabhängige HTTP-APIs (Hypertext Transfer Protocol) definiert, die in weiten Teilen dem REST-Paradigma (Representational State Transfer) folgen. Anhang A liefert eine Übersicht davon, welcher

⁵Ansonsten müsste das Kernsystem auf dem Rechner, an den ein solcher Aktor angeschlossen ist, betrieben werden. Dies wäre spätestens bei der Verwendung mehrerer Akto- ren mit lokaler Schnittstelle an verschiedenen Lokalitäten nicht mehr umsetzbar.

Dienst welche Funktionalitäten anbietet. Die aufgeführten Informationen beschränken sich dabei lediglich auf die wesentlichen Eigenschaften der definierten HTTP-Endpunkte, weil sie ansonsten den Rahmen dieses Dokument sprengen würden. Details, welche darüber hinaus gehen (benötigte Parameter, zurückgegebene HTTP-Statuscodes etc.), können in der mit *SpringFox* bzw. dem darin enthaltenen Framework *Swagger* generierten API-Dokumentation nachgelesen werden. Diese kann für jeden hochgefahrenen Service separat unter der URL `http://<Service-Hostname>:<Service-Port>/swagger-ui.html` aufgerufen werden. Darüber hinaus enthält das zu dieser Abschlussarbeit gehörende GitHub-Repository (siehe Abschnitt 3.4.6) auch eine Sammlung von Beispiel-Aufrufen für jeden einzelnen HTTP-Endpunkt, den SL anbietet. Diese können in das API-Entwicklungswerkzeug *Postman* importiert und von dort aus einfach abgeändert sowie versendet werden.

Anhang A umfasst sämtliche für SL definierten Schnittstellen. Das schließt auch diejenigen ein, die eigentlich nur für systeminterne Zwecke gedacht sind (z.B. die des *assistance service*, *action service* und *delegate service* aus Abb. 3.17). Außerdem sind die APIs des *configuration service* und *discovery service* in keiner Tabelle aufgeführt. Sie basieren komplett auf externen Programmbibliotheken, sodass eine eigene Schnittstellen-Definition nicht nötig war. Die Abschnitte 4.3 und 4.4 beschreiben die Implementierung der beiden Dienste und die verwendeten Bibliotheken im Detail.

Die Tabellen in Anhang A enthalten aus Platzgründen nicht die vollständigen Ressourcen-URLs, sondern sind um den Protokoll-, Host- und Port-Teil gekürzt. Das bedeutet, dass z.B. die Ressource `http://<Hostname>:<Port>/smart-lab/api/event/<ID>` in Anhang A nur als `smart-lab/api/event/<ID>` aufgeführt ist. Etwaige URL-Parameter sind dabei von spitzen Klammern umgeben.

Auf eine detaillierte Erläuterung aller Schnittstellen soll an dieser Stelle verzichtet und erneut auf die Tabellen in Anhang A verwiesen werden. Allerdings sind trotzdem einige Design-Entscheidungen, welche beim Entwurf der APIs getroffen wurden, hervorzuheben. So enthält das API des *assistance service* nur jeweils einen HTTP-Endpunkt für die drei Phasen einer Assistance (siehe Tabelle A.9). Es wurde bewusst darauf verzichtet, die Aufrufe expliziter Assurances in die Schnittstelle einzubauen. Da prinzipiell von einer Erweiterung von SL um weitere Assurances auszugehen ist, hätte dies ansonsten ein stetiges und nach oben unbeschränktes Erweitern der Schnittstelle zur Folge gehabt. Stattdessen kann über das API des *assistance service* eine bestimmte Phase einer Assistance, die im übergebenen Assistance-Kontext festgelegt ist, ausgeführt werden.

Aus demselben Grund besitzen die APIs des *action service* und *delegate service* lediglich einen einzelnen HTTP-Endpunkt zur Ausführung einer Action (siehe Tabellen A.10 und A.11). Auch hier ist prinzipiell von einer Erweiterung von SL um weitere Actions auszugehen, was keinen Einfluss auf die Schnittstellen der Dienste haben sollte. Zusätzlich ist das API des *delegate service* leicht anders als dasjenige des *action service* gehalten

und benötigt mehr Parameter. Dieser Umstand röhrt daher, dass der *delegate service* so konzipiert ist, dass er alle Informationen, welche er benötigt, bereits in einem Aufruf erhält. Somit ist keine Rückkopplung zum Kernsystem notwendig. Alle notwendigen Interaktionen (z.B. das Auflösen des Typs eines Aktors) mit Teilen des Kernsystems werden bereits vom *action service* abgehandelt, bevor er die Ausführung einer Action an eine Instanz des *delegate service* weiterleitet. Dies spiegelt sich auch in den Verknüpfungen der Services in Abb. 3.17 wider.

4. Implementierung

Dieses Kapitel beschreibt wie SL gemäß dem Konzept aus Kapitel 3 umgesetzt wurde. Dabei wurden weitestgehend alle beschriebenen Prinzipien und Funktionsweisen implementiert (mit Ausnahme der in Abschnitt 3.7.2 aufgeführten möglichen Anwendungsfälle für Assurances). Die erste Iteration von SL ist somit funktionsfähig und die Inbetriebnahme ist in Abschnitt 5.1 beschrieben. SL wurde zum überwiegenden Teil in der Programmiersprache *Java* (Version 8) implementiert. Daneben kam die Sprache *C#* (.NET Version 4.7) für einige Windows-spezifische Parts (siehe Abschnitt 4.11) zum Einsatz sowie die Skriptsprache *Groovy* (Version 2.4) für die Integrationstests.

4.1. Umsetzung der Services

Die in Abb. 3.17 gezeigten Services wurden mit dem Java-Framework *Spring Boot* realisiert. Diese Entscheidung wurde im Hinblick auf die in Abschnitt 3.4 beschriebenen Schwerpunkte des Projekts getroffen. Mit dem quelloffenen Spring Boot umgesetzte Services können im eigenen Netzwerk bereitgestellt werden, was bei der Nutzung eines externen Cloud-Diensts wie z.B. *AWS* (Amazon Web Services) nicht gegeben ist. Das Framework stellt zudem bei der Anwendungsentwicklung eine Reihe von zusätzlichen Funktionalitäten bereit, deren Verwendung hauptsächlich über Java-Annotationen gesteuert wird. Die folgenden Funktionalitäten von Spring Boot wurden bei der Entwicklung der Services von SL umfassend genutzt und ziehen sich durch das gesamte Projekt.

Dependency Injection Spring Boot verwaltet die Erzeugung und Bereitstellung von Komponenten einer Anwendung, den sogenannten *Beans*. Beans sind Klassen, die beim Start einer Applikation von Spring Boot in einer konfliktfreien Reihenfolge erzeugt und allen anderen Programmteilen (d.h. auch anderen Beans) zur Verfügung gestellt werden. Die meisten Klassen von SL werden als Beans von Spring Boot verwaltet. (vgl. Webb et al. o.J.)

REST-Controller Mit Spring Boot können REST-Endpunkte über Java-Annotationen definiert werden, welche bei ihrer Aktivierung benutzerdefinierte Logik ausführen. Die Controller sind dabei Beans, die Aufrufe an die Endpunkte verarbeiten. Jeder der Services von SL besitzt einen eigenen REST-Controller. In Listing 4.1 ist beispielhaft einer der Endpunkte eines solchen Controllers von SL dargestellt. Erfolgt ein

passender HTTP-Aufruf an die spezifizierte URL, so wird die annotierte Methode ausgeführt. (vgl. Webb et al. o.J.)

Konfiguration einer Anwendung Über Spring Boot können Konfigurationsdaten in Form von Java-Properties-Dateien geladen werden. Über solche Properties können einerseits Eigenschaften von Spring Boot selbst bzw. zum Framework kompatiblen Bibliotheken beeinflusst werden. Andererseits kann aber auch die eigene Anwendung durch benutzerdefinierte Properties konfiguriert werden. Das System von SL, d.h. jeder der zu SL gehörenden Services, wird durch von Spring Boot verwaltete Properties-Dateien konfiguriert (im Sinne von Abschnitt 3.2). (vgl. Webb et al. o.J.)

Listing 4.1: Spring Boot erlaubt eine einfache Realisierung von REST-Endpunkten mit Java. Der hier dargestellte Endpunkt stammt aus dem Controller des *assistance service* (siehe Abb. 3.17) und dient zum Starten der Phase *begin* einer Assistance. Zur besseren Lesbarkeit wurden alle Konstanten des eigentlichen Quellcodes durch ihre entsprechenden String-Literale ersetzt.

```
@PostMapping(
    value = "/smart-lab/api/assistance	begin",
    consumes = "application/json")
public ResponseEntity<Void> beginAssistance(
    @RequestBody IAssistanceContext context) {
    ...
}
```

Die Nutzung von Spring Boot in SL ist jedoch nicht nur auf die eben genannten Funktionalitäten beschränkt. Darauf hinaus bietet das Framework noch zahlreiche weitere Vorteile, die auch über Bibliotheken erweitert werden können. Daher soll an dieser Stelle nicht auf alle einzelnen Verwendungszwecke eingegangen werden. Falls weitere Aspekte von Spring Boot bei einzelnen Teilen der Implementierung von SL zum Einsatz kamen, ist das in den entsprechenden Abschnitten dieses Kapitels beschrieben.

Anwendungen, welche Spring Boot verwenden, werden in einer JAR-Datei (Java Archive) verpackt. Zusätzlich ist in der Datei auch ein eingebetteter Applikationsserver enthalten, der es ermöglicht, die Anwendung durch ein einfaches Ausführen der JAR-Datei zu starten. Im Fall von SL kann eine solche Anwendung einen oder mehrere Services umfassen. (vgl. Webb et al. o.J.)

Alle Services von SL sind gemäß einer Schichtenarchitektur aufgebaut, um die Annahme von HTTP-Anfragen, die Geschäftslogik und die Datenhaltung sauber voneinander zu trennen. Diese Architektur ist in Abb. 4.1 dargestellt. Jede der Schichten ist durch eine eigene von Spring Boot verwaltete Bean realisiert.

Anfragen an einen Service werden von der Controller-Schicht angenommen und letztendlich auch wieder beantwortet. Diese Schicht kapselt die technischen Details der Kommunikation mit dem Service und ist in SL immer als REST-Controller umgesetzt. Von dort werden Anfragen an die Business-Schicht weitergeleitet, welche sich um die fachliche Bearbeitung einer Anfrage kümmert. Hier laufen z.B. Vorgänge ab wie das Auflösen, ob eine Assistance auf ein Trigger-Signal reagiert. Müssen Daten im Rahmen der Bearbeitung durch die Business-Schicht persistiert oder abgefragt werden, kommt die dritte und letzte Ebene, die Repository-Schicht, zum Tragen. Diese Schicht kapselt die Art und Weise der Datenmanipulation und stellt z.B. den Adapter für verwendete Datenquellen dar (siehe Abb. 3.5). Konsequenterweise ist die Repository-Schicht nur in solchen Services vorhanden, welche auch die Persistierung von Daten involvieren.

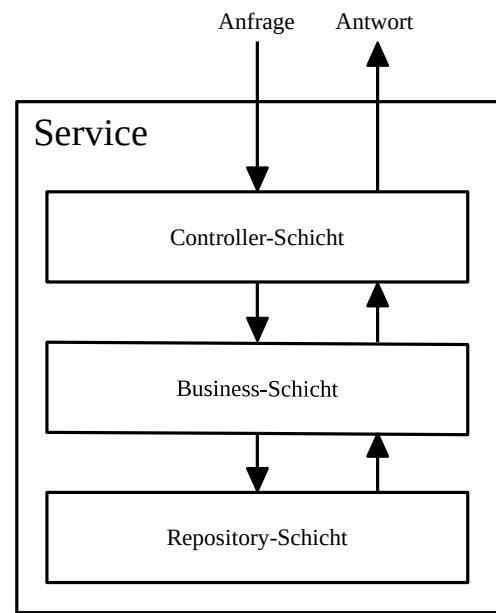


Abbildung 4.1: Die Services von SL sind in Schichten aufgeteilt, welche miteinander kommunizieren.

4.2. Modulstruktur

Bei der Entwicklung von SL kam das Build-Management-Werkzeug *Apache Maven* (Version 3.5.2) zum Einsatz. Neben dessen Funktionalitäten zur Automatisierung des Build-Prozesses und Verwaltung der Abhängigkeiten wurde auch die Möglichkeit genutzt Quellcode in Maven-Module zu unterteilen. Der überwiegende Teil des Quellcodes von SL (d.h. der in Java und Groovy geschriebene Code) ist in eine Modul-Hierarchie eingeteilt, die in Abb. 4.2 abgebildet ist.

Die beiden Modul-Beziehungen, welche durch die Pfeile in Abb. 4.2 dargestellt sind, mögen auf den ersten Blick ähnlich erscheinen. Sie haben jedoch gänzlich unterschiedliche Bedeutungen und sind völlig unabhängig voneinander. Untermodule auf der einen Seite werden automatisch in den Build-Prozess ihrer übergeordneten Module miteinbezogen. Wenn ein Modul auf der anderen Seite von einem weiteren erbt, heißt dies, dass es verschiedene Eigenschaften von ihm übernimmt (dessen Abhängigkeiten etc.). Die in Abb. 4.2 dargestellten Module binden sich in hohem Maße auch gegenseitig als Abhängigkeiten ein.

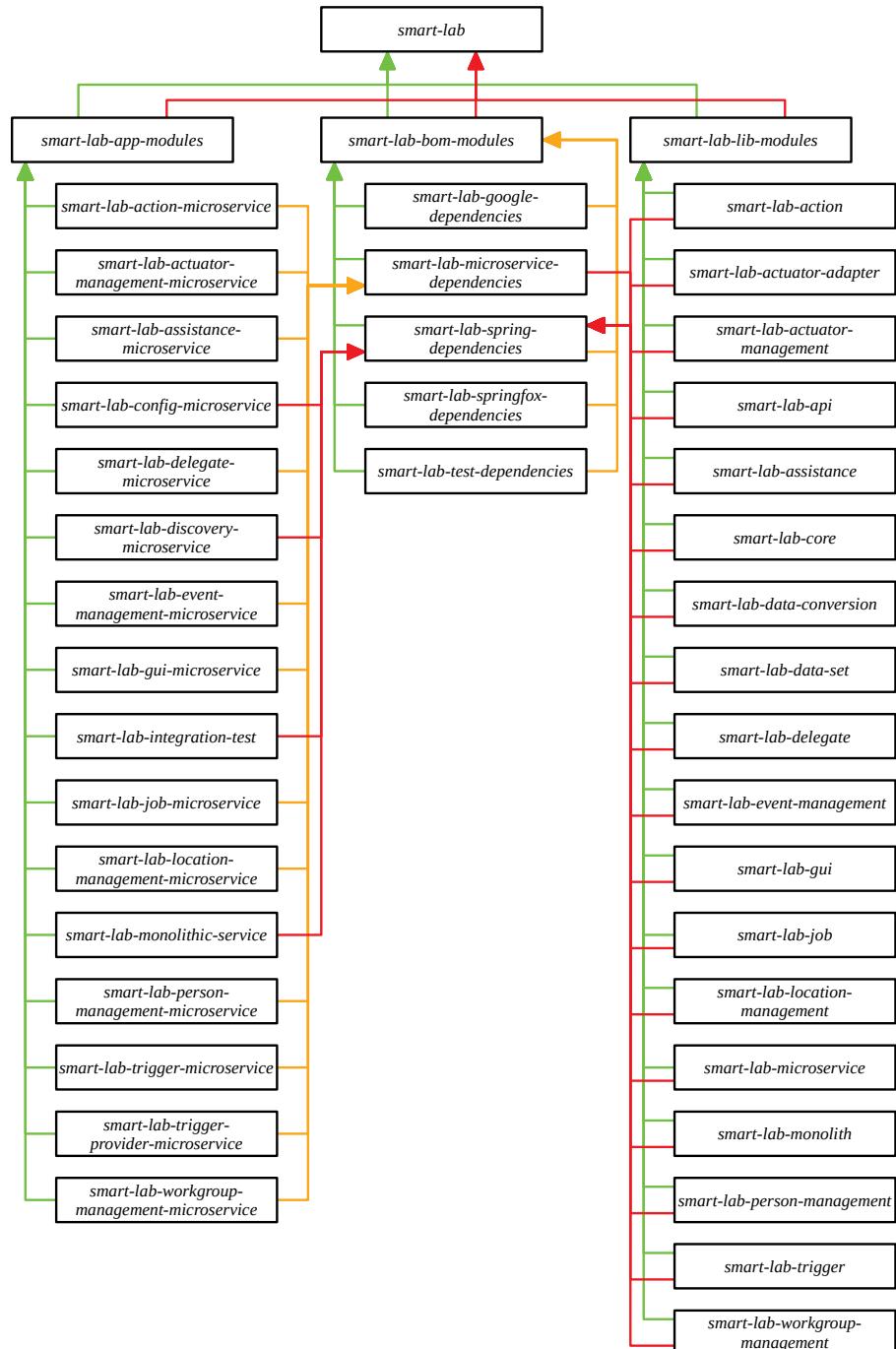


Abbildung 4.2: Der Quellcode von SL besteht aus einer Hierarchie aus Maven-Modulen. Grüne Pfeile zeigen die Beziehung „ist ein Untermodul von“ an. Rote und gelbe Pfeile symbolisieren die Beziehung „erbt von“ (zur Vermeidung von mehrdeutigen Pfeilverläufen wurden zwei verschiedene Farben verwendet).

Diese Beziehung wurde jedoch aus Gründen der Übersichtlichkeit nicht dargestellt und muss dem zu dieser Abschlussarbeit gehörenden Quellcode entnommen werden.

Das Wurzel-Modul von SL enthält Metadaten und Definitionen für die restlichen Module und dient als Einstiegspunkt, um den Build-Prozess für das gesamte Projekt zu starten. Es unterteilt sich in einer ersten Stufe in jeweils ein Modul für Applikationen, BOMs (Bill of Materials) und Bibliotheken. Um nicht den Rahmen dieses Dokuments zu sprengen, werden sich die Erläuterungen dieses Abschnitts auf den grundlegenden Inhalt jener drei Untermodule beschränken. Die Module der untersten Ebene werden weitestgehend ausgelassen, jedoch ist ihre jeweilige Bedeutung in den zugehörigen POM-Dateien¹ (Project Object Model) beschrieben und ergibt sich auch meist direkt aus ihren Namen.

BOM-Module Diese Module enthalten keinen Quellcode und dienen lediglich zum gebündelten Einbinden mehrerer zusammengehöriger Abhängigkeiten. BOM-Module können dazu entweder ihrerseits als Abhängigkeiten eingebunden werden oder als Eltern-Module verwendet werden, von denen geerbt wird.

Bibliotheks-Module Die Module dieser Kategorie enthalten die gesamte Logik von SL (Services, Assurances etc.). Sie verwenden zwar in großem Maße Spring Boot, werden jedoch nicht in ausführbaren, sondern normalen JAR-Dateien verpackt. Auf diese Weise können sie in andere Module eingebunden und die Logik von SL somit flexibel auf beliebig viele Anwendungen verteilt werden.

Applikations-Module Hier enthaltene Module werden zu Spring Boot Anwendungen in Form von ausführbaren JAR-Dateien verpackt und dienen zum Starten der verschiedenen Services von SL. Die Module bestehen in der Regel nur aus einer einzigen Klasse, welche den Einsprungpunkt der Applikation enthält. Alle Dienste aus Abb. 3.17 können hier als separate Anwendungen gefunden werden (d.h. als sogenannte *Microservices*). Dadurch dass sie sämtliche Logik von den Bibliotheks-Modulen importieren, kann der Umfang einer Anwendung beliebig skaliert werden. Als Beispiel hierfür dient das Modul *smart-lab-monolithic-service*, welches sämtliche in blau dargestellten Dienste aus Abb. 3.17 in einer einzelnen Applikation vereint.

Die Ausführung des Kernsystems von SL ist auf zwei verschiedene Arten möglich. Entweder werden alle Services separat über ihre jeweiligen Applikations-Module gestartet oder SL wird als Monolith über das im vorigen Absatz erwähnte Modul hochgefahren. Erstere Methode wird für den Rest dieses Dokuments als *Ausführung als Microservices* und letztere als *Ausführung als Monolith* bezeichnet.

Die Ausführung als Monolith stellt dabei die Standard-Methode dar und ist auch der Weg, welcher in Abschnitt 5.1.2 für die Inbetriebnahme von SL beschrieben ist. Die Ausfüh-

¹POM-Dateien dienen zur Konfiguration von Maven. Jedes Modul verfügt über eine solche Datei.

rung als Microservices ist in der ersten Iteration von SL noch experimenteller Natur und nicht vollständig lauffähig (in dem zu dieser Abschlussarbeit gehörenden Quellcode sind entsprechende To-do-Markierungen hinterlegt). An dieser Stelle sei noch angemerkt, dass der *discovery service* ausschließlich bei der Ausführung als Microservices benötigt wird. Wenn das System als Monolith gestartet wird, sind die Dienste von SL im Vornhinein miteinander verknüpft.

4.3. Konfigurationsverwaltung

In Abschnitt 4.1 wurde bereits beschrieben, dass Spring Boot Funktionalitäten für die Konfiguration von Anwendungen über Java-Properties-Dateien anbietet. Da SL aus mehreren Anwendungen besteht, die alle individuell konfiguriert werden müssen, stellt eine manuelle Verwaltung der Konfigurationsdateien für verschiedene Umgebungen einen gewissen Aufwand dar. Deshalb wurde in SL die Bibliothek *Spring Cloud Config* verwendet, um einen Service für die Konfigurationsverwaltung zu realisieren (in Abb. 3.17 als *configuration service* dargestellt). Über diesen Dienst wird die Konfiguration des gesamten Systems im Sinne von Abschnitt 3.2 zentral vorgenommen (vgl. Spring Cloud Config o.J.[a],[b]).

Dem Service wird das Verzeichnis aller zu verwendenden Konfigurationsdateien übergeben. Diese können entweder lokal oder über ein GitHub-Repository verfügbar sein. Alle anderen Dienste von SL beziehen ihre Konfigurationen dann vom *configuration service*, anstatt sie bei ihrem Start manuell zugewiesen zu bekommen. Falls also unterschiedliche Konfigurationsdatensätze für das System verwendet werden sollen, so muss lediglich die Startprozedur des *configuration service* angepasst werden (indem ihm ein entsprechender Pfad-Parameter übergeben wird). (vgl. Spring Cloud Config o.J.[a],[b])

Auf der einen Seite bedeutet das, dass die übrigen Services lediglich Informationen darüber benötigen, wie sie den *configuration service* erreichen können. Auf der anderen Seite heißt es auch, dass der *configuration service* vor allen anderen Diensten gestartet werden muss, weil deren Initialisierung sich auf ihn stützt. Dieser Umstand ist unabhängig davon, ob das System als Monolith oder als Microservices gestartet wird. (vgl. Spring Cloud Config o.J.[a],[b])

4.4. Kommunikation zwischen Services

Prinzipiell kommunizieren die Services von SL netzwerkbasiert. Dazu wurden mit *Feign*, welches Teil des Netflix-Stacks für Spring Boot ist, deklarative HTTP-Clients für die einzelnen Services geschrieben. Ein solcher Feign-Client besteht aus einem Java-Interface, das mit speziellen Annotationen versehen ist. Das Interface kann dann im übrigen Quellcode verwendet werden, um HTTP-Aufrufe an bestimmte URLs abzusetzen und diese entspre-

chend zu parametrieren. Zur Laufzeit wird dann von Feign eine Implementierung für alle Clients generiert. (vgl. Feign o.J. Spring Cloud Netflix o.J.[a],[c])

Listing 4.2 zeigt beispielhaft einen Auszug aus dem Feign-Client des *action service* (siehe Abb. 3.17). Die dargestellte Methodendeklaration beschreibt einen HTTP-POST-Aufruf an die spezifizierte URL. Einer der Parameter des Aufrufs wird dabei als Teil der URL übergeben, der andere im Nachrichtenrumpf.

Listing 4.2: Die netzwerkbasierte Kommunikation mit dem *action service* wird programmatisch über einen Feign-Client ermöglicht. Zur besseren Lesbarkeit wurden alle Konstanten des eigentlichen Quellcodes durch ihre entsprechenden String-Literale ersetzt.

```
@FeignClient(
    name = "smart-lab-action-microservice",
    path = "/smart-lab/api/action")
public interface IActionApiClient {

    @PostMapping(
        value = "/{actionId}/execute",
        consumes = "application/json")
    ResponseEntity<IActionResult> executeAction(
        @PathVariable("actionId") String actionId,
        @RequestBody IActionArgs actionArgs);
}
```

Zur vollständigen Adressierung können in Feign-Clients auch Protokoll, Hostname und Port angegeben werden. Listing 4.2 zeigt jedoch die Adressierung über den Service-Namen. Hierfür ist ein zusätzlicher Dienst als Vermittler notwendig (in Abb. 3.17 als *discovery service* dargestellt). Verfügbare Dienste melden sich bei ihm mit ihrem Namen an und können von anderen Services dann über ihren jeweiligen Namen adressiert werden. Zu diesem Zweck dient der *discovery service* als ein Verzeichnis, das Service-Namen zu Netzwerkadressen auflöst. Er stellt damit eine zentrale Anlaufstelle für die Kommunikation zwischen Diensten dar. Für die Realisierung dieses Service wurde *Eureka* verwendet, welches ebenfalls Teil des Netflix-Stacks für Spring Boot und damit auch kompatibel zu Feign ist. Eureka bietet daneben noch die Möglichkeit zur automatischen Lastverteilung von Anfragen, falls mehrere Instanzen eines Service existieren. Dies hat jedoch bei der Implementierung von SL keine übergeordnete Rolle gespielt. (vgl. Sarkar 2017; Spring Cloud Netflix o.J.[b],[c])

Wird SL als Monolith gestartet, entfällt die Notwendigkeit des *discovery service* komplett. Die Kommunikation der Services des Kernsystems (was den Großteil aller Services von SL einschließt) läuft dann nicht mehr netzwerkbasiert ab. Stattdessen werden die HTTP-

Endpunkte der verschiedenen Controller-Schichten direkt als Java-Methoden innerhalb der monolithischen Anwendung aufgerufen. Ausgenommen hiervon ist die Kommunikation zwischen dem Kernsystem und den Delegaten. Diese werden weiterhin über Feign-Clients angesprochen, jedoch müssen ihre genauen Netzwerkadressen zuvor in den Konfigurationsdateien von SL hinterlegt werden.

Die Kapselung der Kommunikationsmechanismen ist in Abb. 4.3 beispielhaft für den *action service* dargestellt (existiert aber analog auch für die anderen Dienste). Abhängig davon, ob SL als Monolith oder als Microservices gestartet wurde, werden unterschiedliche Beans verwendet, welche die Service-Kommunikation implementieren (sogenannte *Connectors*). Der Microservice-Connector verwendet den Feign-Client des *action service*, der wiederum HTTP-Anfragen an den entsprechenden Controller generiert. Der Monolith-Connector ruft hingegen direkt auf Java-Ebene die Methoden des Controllers auf.

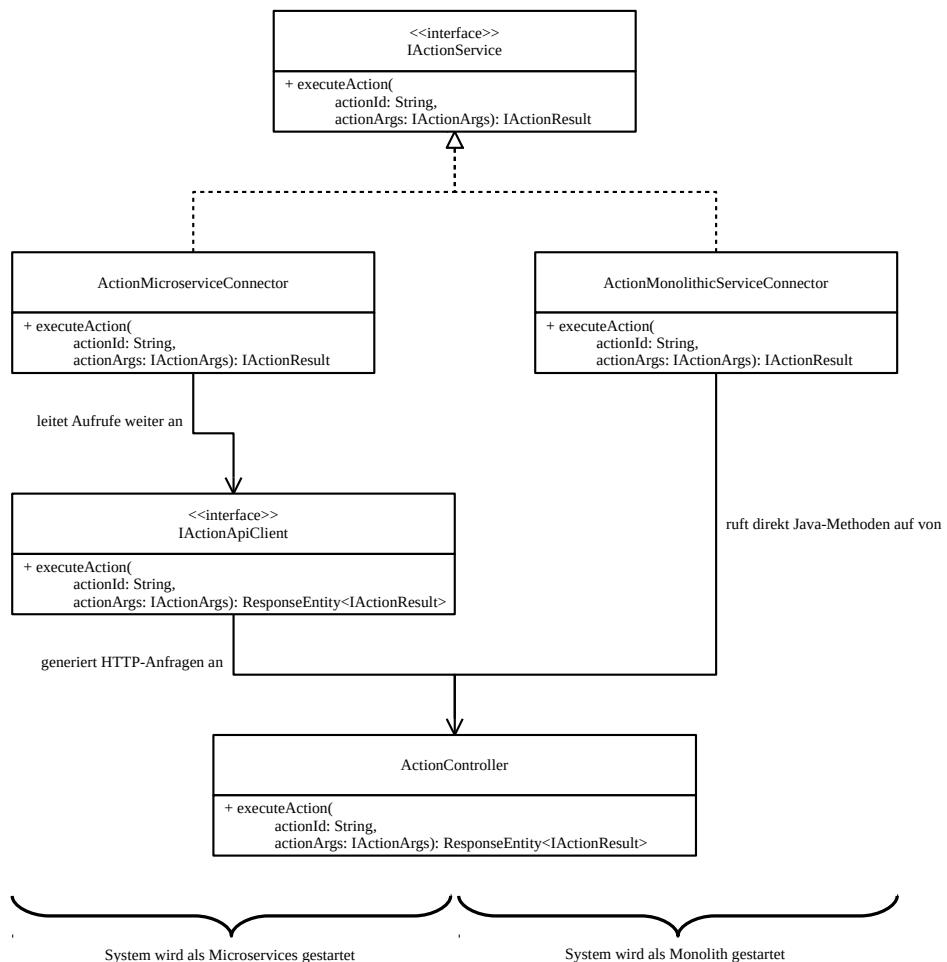


Abbildung 4.3: Die Mechanismen der Kommunikation zwischen den Diensten von SL sind abhängig davon, ob das System als Monolith oder als Microservices gestartet wurde.

4.5. Datenmanagement-Services

Der folgende Abschnitt beschreibt die Spezifika mehrerer Services von SL, welche alle einen grundsätzlich ähnlichen Aufbau besitzen: die Services für das Datenmanagement. Diese Dienste verwalten die Informationen über Akteure, Lokalitäten, Arbeitsgruppen, Personen und Events, auf die SL zugreifen kann (siehe den oberen Bereich von Abb. 3.17). Dabei ist jeder der Services an eine Datenquelle angebunden, deren Eigenheiten von SL zunächst nicht weiter spezifiziert werden. Prinzipiell ist eine Anbindung an jegliche Art von Datenquelle möglich. Diese Flexibilität röhrt von den Repository-Schichten (siehe Abb. 4.1) der fünf Datenmanagement-Services. Die Schichten dienen dabei nicht zwingend selbst zur Datenhaltung, sondern können auch Adapter für entsprechende Datenquellen darstellen, welche diese Aufgabe übernehmen.

Für jeden der fünf Services wurde jeweils eine Mock-Implementierung der Repository-Schicht geschrieben. Jene speichert Daten zwar selbst ab, persistiert sie jedoch nicht, sondern hält sie lediglich im Arbeitsspeicher. Diese prototypischen Implementierungen haben den Zweck, SL unkompliziert mit Datenquellen zu versehen, die in jeder Systemlandschaft funktionieren. Die Interaktion mit ihnen (d.h. ihr Befüllen und das Manipulieren von enthaltenen Daten) ist aber auf die Verwendung der angebotenen HTTP-Endpunkte beschränkt. Weboberflächen o.Ä. existieren hierfür in der ersten Iteration von SL noch nicht.

Die Mock-Implementierungen sind von außen nicht sichtbar und das System verhält sich mit ihnen genauso, als ob „echte“ persistente Datenquellen angebunden wären. Damit ist ihre Verwendung also geeignet, um die Funktionalität von SL zu demonstrieren, sie sind jedoch nicht für den Produktiveinsatz gedacht. Auf lange Sicht sollten sie alle durch entsprechende Adapter für geeignete Datenquellen ersetzt werden. Ein solcher Adapter wurde zusätzlich zu den Mock-Implementierungen für SL umgesetzt. Er ermöglicht es, den Kalenderdienst der Firma Google als Datenquelle für den *event management service* anzubinden. So werden alle Events, mit denen SL arbeitet, auf Termine dieses Kalenderdiensts abgebildet. Zusätzlich kann so auch dessen Weboberfläche neben den HTTP-Endpunkten von SL zur Datenmanipulation genutzt werden. Auf diese Weise wird auch die Konfiguration von Events (im Sinne von Abschnitt 3.2) vorgenommen. Die Konfiguration der Assurances eines Events und das Festlegen SL-spezifischer Daten ist in das Beschreibungsfeld von Terminen in Googles Kalenderdienst eingebettet. Dort können Assurances gemäß der Syntax einer speziellen Konfigurationssprache eingerichtet werden, was in Abschnitt 4.6 näher beschrieben ist. Für die programmatische Kommunikation mit dem Kalenderdienst wurde die von Google selbst bereitgestellte Java-Implementierung ihres API genutzt (vgl. Google 2018a).

Natürlich erfordert die Anbindung an den Kalenderdienst eine entsprechende Konfiguration von SL. Dies geschieht über die in Abschnitt 4.3 erwähnten Java-Properties-Dateien.

Dort muss neben den Zugangsdaten für den Dienst auch eine Zuordnung von Lokalitäten zu einzelnen Kalendern hinterlegt werden, welche die dort stattfindenden Events als Termine beinhalten. SL ist so weit abstrahiert, dass es das Konzept eines Kalenders nicht kennt. Daher ist eine solche Zuordnung erforderlich, damit der Adapter seinen Dienst verrichten kann.

Abbildung 4.4 zeigt die Klassenhierarchie der Repository-Schicht des *event management service*. Eine analoge Hierarchie existiert auch für die anderen Datenmanagement-Services. Dabei sind die grau hinterlegten Event-spezifischen Komponenten durch ihre entsprechenden Pendants für Lokalitäten, Aktoren, Arbeitsgruppen und Personen ausgetauscht. Lediglich der Adapter für den Kalenderdienst von Google besitzt kein Gegenstück, da er in der ersten Iteration von SL den einzigen implementierten Datenquellen-Adapter darstellt.

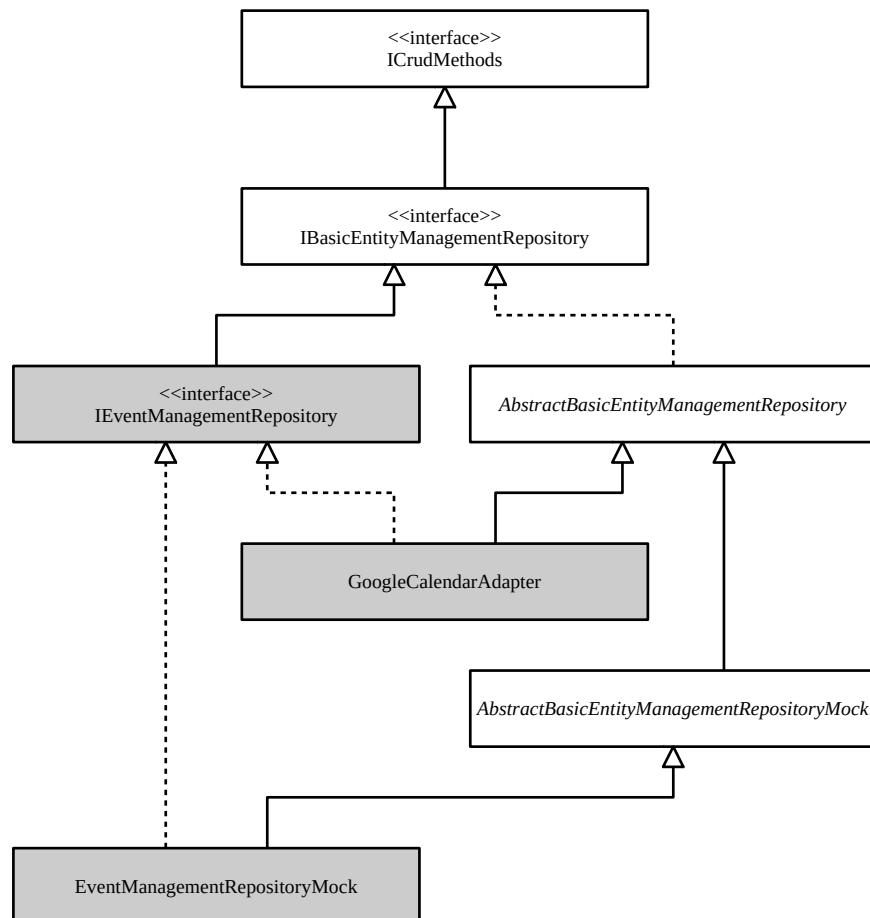


Abbildung 4.4: Die Repository-Schicht des *event management service* ist in einer Klassenhierarchie organisiert. Spezifika der Komponenten wie Instanz- und Klassenvariablen sowie Methoden wurden aus Gründen der Übersichtlichkeit weggelassen. Grau hinterlegte Klassen sind spezifisch für die Datenhaltung von Events.

Die Business-Schicht der Datenmanagement-Services (siehe Abb. 4.1) enthält keine spezielle Logik. Dort findet hauptsächlich die Datenvälidierung statt und Aufrufe werden anschließend an die Repository-Schicht weitergeleitet.

4.6. Konfigurationssprache für Events

Aus der Nutzung eines Kalenderdiensts als Event-Datenquelle ergeben sich zusätzliche Herausforderungen. Die Integration des entsprechenden Service von Google (siehe Abschnitt 4.5) lässt zwar eine Abbildung von SL-Events auf Termine eines Kalenders zu. Jedoch sind weder das Datenmodell noch die Weboberfläche von Googles Kalenderdienst an die Erfordernisse von SL anpassbar (z.B. über Plugins). Dennoch ist es notwendig, gemeinsam mit einem Event bzw. Termin gewisse für SL spezifische Daten zu speichern wie z.B. die Arbeitsgruppe, welcher ein Event zugeordnet ist.

Daher wurde eine Konfigurationssprache implementiert, über die potenziell beliebige Eigenschaften eines Events festgelegt werden können. Diese wird für den Rest des Dokuments mit ECL (Event Configuration Language) abgekürzt und stellt ein für die Konfiguration von Events und Assurances (im Sinne von Abschnitt 3.2) verwendetes Werkzeug dar. Für die Speicherung eines Konfigurationsblocks in ECL ist lediglich ein dafür nutzbares Textfeld erforderlich, was bei Googles Kalenderdienst gegeben ist (so wie bei der überwiegenden Mehrheit der sonstigen denkbaren Event-Datenquellen). Damit wurde eine Möglichkeit geschaffen, mit der Eigenschaften von Events in rein textueller Form abgespeichert werden können. Diese Methode ist einerseits skalierbar, da sie der Mächtigkeit einer Event-Datenquelle anpassbar ist (somit müssen nur diejenigen Eigenschaften über ECL codiert werden, welche nicht nativ in der Datenquelle gespeichert werden können). Und andererseits ist sie zukunftssicher, weil davon auszugehen ist, dass annähernd jede Event-Datenquelle über ein Textfeld verfügt, das für die Konfiguration über ECL genutzt werden kann. Der in Abschnitt 4.5 beschriebene Adapter für Googles Kalenderdienst verwendet hierfür das Beschreibungsfeld von Terminen, was in Abb. 4.5 dargestellt ist. Dort sind die drei Event-Eigenschaften zu sehen, die in der ersten Iteration von SL über ECL codiert werden können:

- Arbeitsgruppe, der ein Event zugeordnet ist
- Agenda eines Events
- Assurances, welche für ein Event gewünscht sind, sowie deren Konfiguration

Die Strukturierung von ECL erfolgt dabei über Schlüsselwörter, denen ein At-Zeichen vorangestellt ist. Die Schlüsselwörter können öffnende und schließende Tags darstellen, aber auch einzelne Eigenschaften eines Events auszeichnen wie z.B. beim Festlegen der

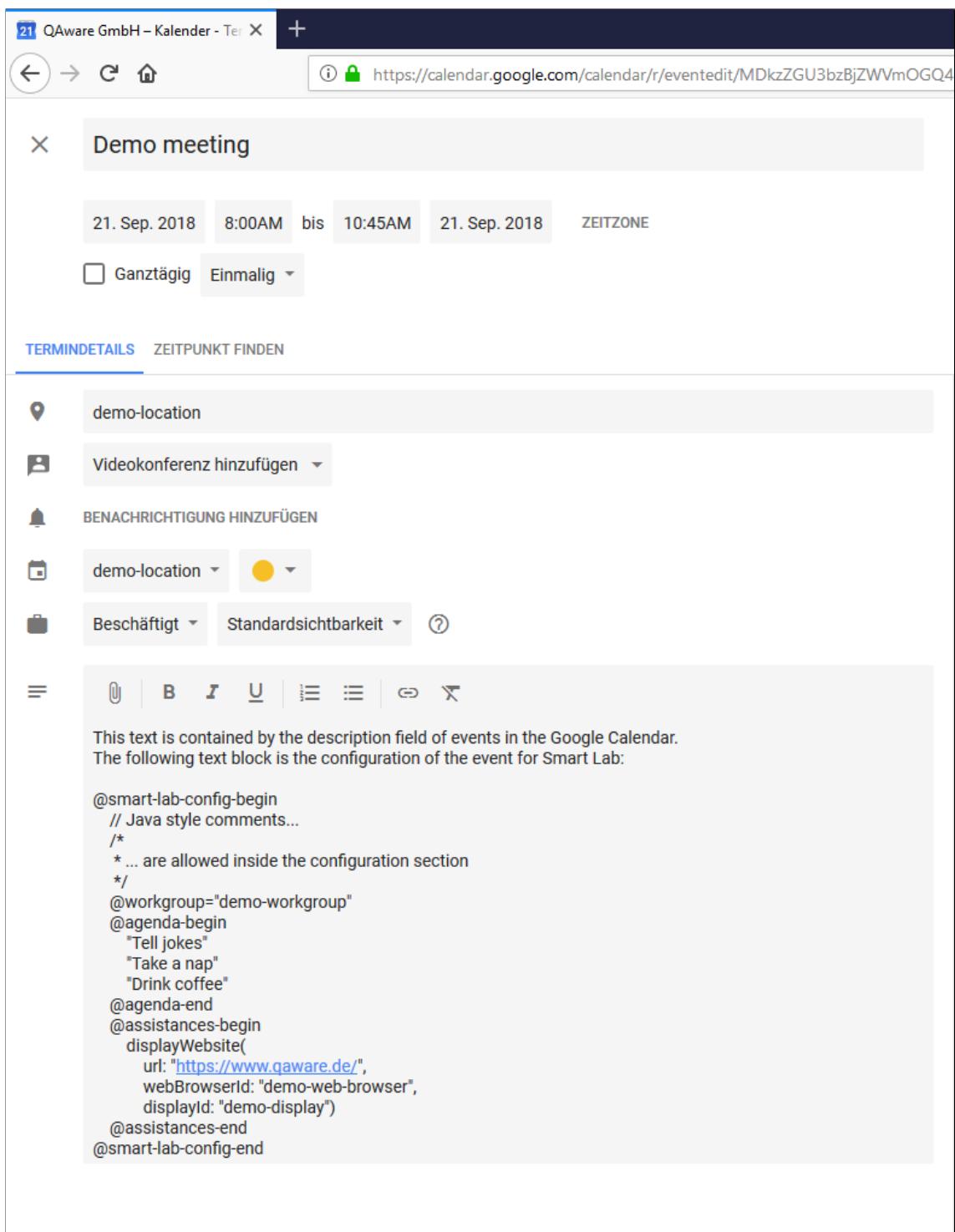


Abbildung 4.5: Die Konfiguration von Events erfolgt beim Kalenderdienst von Google teilweise über ECL. Der im Beschreibungsfeld enthaltene Konfigurationsblock in ECL legt die Arbeitsgruppe, die Agenda und die gewünschten Assistanzen fest.

Arbeitsgruppe. Text, der vor oder nach den äußersten Tags steht, ist für ECL nicht relevant und wird ignoriert. Damit kann z.B. das Beschreibungsfeld eines Termins im Kalenderdienst von Google auch weiterhin als solches verwendet werden, selbst wenn es für ECL genutzt wird. Zusätzlich ist es auch möglich innerhalb eines Konfigurationsblocks Kommentare im Java-Stil einzufügen. Die Parametrierung von Assurances ist in ECL über Key-Value-Paare möglich. Die nötigen Parameter für jede Assurance sind in den jeweiligen Unterkapiteln von Abschnitt 3.7.1 beschrieben.

In Abb. 4.5 ist die Konfiguration der Assurance *website displaying* dargestellt. Diese ist so eingerichtet, dass sie die Internetseite mit der URL <https://www.qaware.de/> öffnet und dafür den Webbrowser-Aktor *demo-web-browser* und den Anzeigegerät-Aktor *demo-display* verwendet (Details zur Implementierung von Akten und deren Adaptern sind in Abschnitt 4.9 zu finden). Ein umfassendes Beispiel für einen Konfigurationsblock in ECL, in welchem alle realisierten Assurances vorkommen, ist in Anhang C aufgeführt. An dieser Stelle sei noch hervorgehoben, dass eine Assurance auch mehrfach pro Konfiguration vorkommen kann, z.B. um mehrere Internetseiten auf unterschiedlichen Anzeigegerät-Aktoren darstellen zu lassen.

Um einen in ECL verfassten Konfigurationsblock in ein entsprechendes Objekt umzuwandeln, mit dem SL umgehen kann, ist natürlich ein Parser notwendig. Hierfür wurde der Parser-Generator *ANTLR* (Another Tool for Language Recognition) verwendet. ANTLR ermöglicht es, einen Parser aus einer formalen Grammatik zu erzeugen, anstatt ihn selbst zu implementieren (vgl. ANTLR 2016). Die Grammatik folgt dabei der EBNF (Erweiterte Backus-Naur-Form) und ist in einer eigenen Datei definiert (vgl. ANTLR 2011).

Listing 4.3 zeigt einen Auszug aus der Grammatik von ECL, in dem die hierarchisch höchsten Sprachregeln der Konfigurationssprache definiert sind. Demnach kann zwischen den äußersten Tags eine beliebige Anzahl von *statements* stehen. Diese können entweder ein *assignment* sein (wie im Fall des Festlegens der Arbeitsgruppe) oder eine *section* aufspannen (wie im Fall des Festlegens der Agenda oder der gewünschten Assurances). Der interessierte Leser sei auf die im Quellcode enthaltene Grammatik-Datei *EventConfigurationLanguage.g4* verwiesen, um einen tieferen Einblick zu erhalten. Ihre umfassende Behandlung würde den Umfang dieses Dokuments sprengen.

Listing 4.3: ECL ist über eine formale Grammatik in der EBNF definiert. Der hier aufgeführte Ausschnitt aus dieser Grammatik zeigt lediglich ihre hierarchisch höchsten Sprachregeln.

```
grammar EventConfigurationLanguage;

eventConfiguration
    : CONFIG_TAG_BEGIN statement* CONFIG_TAG_END EOF
    ;
```

```

statement
: assignment
| section
;

...

```

ANTLR generiert aus der Grammatik mehrere Java-Klassen (oder bei Bedarf auch Quellcode für andere Programmiersprachen), welche unter anderem einen Lexer sowie einen Parser umfassen. Der Lexer ist dafür zuständig, eine Zeichenkette in einen Strom von sogenannten *Tokens* umzuwandeln. Tokens sind Zeichenketten, denen in der Grammatik eine Bedeutung zugewiesen wurde (wie z.B. Schlüsselwörter oder Werte von Variablen). Aus dem Token-Strom erzeugt der Parser wiederum einen Syntaxbaum, der sämtliche Tokens in einer strukturierten Form beinhaltet, welche der Grammatik folgt. Der komplette Baum oder auch nur Teile davon können von ANTLR in syntaktisch korrekter Reihenfolge (d.h. nach dem Prinzip der Tiefensuche) abgewandert werden. Dabei kann benutzerdefinierter Quellcode gemäß dem Besucher-Entwurfsmuster eingehängt werden. So ist es möglich, auf das Vorhandensein und den Inhalt bestimmter Knoten (welche letztendlich Tokens darstellen) zu reagieren. In SL wird während des Abwanderns ein Java-Objekt erzeugt, das einen Konfigurationsblock in ECL repräsentiert. Nach der Fertigstellung dieses Objekts ist der Parse-Vorgang abgeschlossen. (vgl. ANTLR 2016)

4.7. Assurances

Die Logik, welche Assurances kapseln, kommt an verschiedenen Stellen von SL zum Tragen und nicht nur, wie man zunächst vermuten könnte, im *assistance service* (siehe Abb. 3.17). Der *trigger service* muss z.B. entscheiden, ob und wie eine Assurance auf ein Trigger-Signal reagiert, um anschließend den *assistance service* entsprechend aufrufen zu können. Daher wurden die Assurances von SL in drei Komponenten aufgespalten, die jeweils einen bestimmten funktionellen Aspekt kapseln, jedoch unabhängig voneinander verwendet werden können. Der Zweck dieser *Info*-, *Triggerable*- und *Controllable*-Komponenten wird im Folgenden beschrieben.

Info Die Info-Komponente enthält allgemeine Informationen zu einer Assurance wie z.B. deren ID und eventuelle Aliasse. Außerdem können über sie Konfigurationsobjekte für die entsprechende Assurance erzeugt werden. Die Info-Komponente jeder Assurance ist als Bean zur Laufzeit verfügbar und wird an verschiedenen Stellen in der Logik von SL verwendet.

Triggerable Das Verhalten wie eine Assistance auf die verschiedenen Trigger-Signale reagiert, ist in ihrer Triggerable-Komponente gekapselt. Dabei ist die Reaktion auf jedes Signal als eigene Java-Methode verfügbar. Eine solche Methode könnte z.B. einen Aufruf an den *assistance service* absetzen, um eine bestimmte Phase der Assistance ausführen zu lassen. Oder aber sie enthält keinerlei Logik, wenn eine Assistance nicht auf das entsprechende Trigger-Signal reagiert. Die Triggerable-Komponenten stellen damit eine zusätzliche Abstraktionsschicht für die Nutzung des *assistance service* dar. Diese Komponenten sind ebenfalls als Beans zur Laufzeit verfügbar und werden vom *trigger service* verwendet.

Controllable Diese Komponente einer Assistance enthält die Ausführungslogik ihrer einzelnen Phasen. Jeder Phase ist dabei eine Java-Methode zugeordnet, welche die Ausführung der entsprechenden Actions veranlasst und deren Inputs und Outputs korrekt miteinander koppelt (siehe Abb. 3.14). Die Controllable-Komponenten realisieren somit eine zusätzliche Abstraktionsschicht für die Nutzung des *action service*. Sie werden vom *assistance service* verwendet, um die Phasen von Assurances auszuführen. Im Gegensatz zu den Info- und Triggerable-Komponenten werden die Controllables nicht als Beans zur Verfügung gestellt. Sie werden stattdessen für jede aktiv werdende Assistance neu instanziert. Ihr Lebenszyklus deckt sich dabei mit der Aktivität einer Assistance, d.h. er reicht von der Ausführung der Phase *begin* bis zur Ausführung der Phase *end*. Die separate Instanziierung der Controllable-Komponenten ist erforderlich, damit in ihnen eventuell anfallende Daten über die Phasen einer Assistance hinweg gespeichert und verfügbar gemacht werden können. Diese können potenziell für jede aktive Assistance unterschiedlich sein (z.B. die ID einer geöffneten Webbrowswer-Instanz in der Assistance *website displaying*).

Für jede der fünf in Abschnitt 3.7.1 beschriebenen Assurances wurden diese drei Komponenten implementiert. In Abb. 4.6 ist ihre Klassenhierarchie dargestellt. Die Komponenten werden von den Services, welche sie verwenden, in ihren jeweiligen Business-Schichten aufgerufen (siehe Abb. 4.1). Auf eine detaillierte Beschreibung aller konkreten Info-, Triggerable- und Controllable-Implementierungen wird bewusst verzichtet, da es den Rahmen dieses Dokuments sprengen würde. Der interessierte Leser sei hierfür auf den zu dieser Abschlussarbeit gehörenden Quellcode verwiesen (siehe Abschnitt 3.4.6).

Grund für die Trennung der Assurances in diese drei unabhängigen Komponenten sind deren Abhängigkeiten. Insbesondere die Controllable-Komponente hängt von vielen anderen Modulen ab. Diese Abhängigkeiten sollten nur dann von Services eingebunden werden, falls sie wirklich gebraucht werden, da sie gleichzeitig auch die Notwendigkeit ihrer Konfiguration mit sich bringen.

Der Lebenszyklus der Controllable-Komponenten erfordert deren Speicherung, weswegen

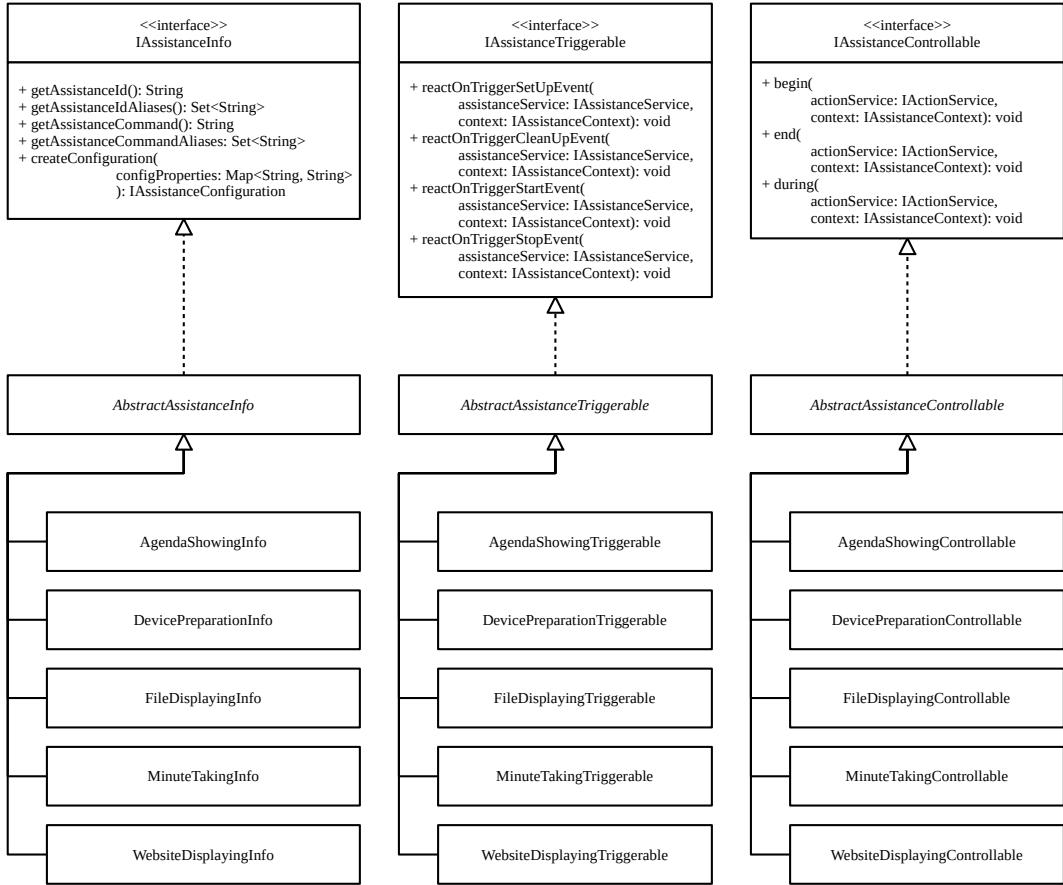


Abbildung 4.6: Die Info-, Triggerable- und Controllable-Komponenten aller implementierten Assists sind in einer Klassenhierarchie organisiert. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

der *assistance service* zustandsbehaftet ist. Daraus folgt, dass dieser Service in der ersten Iteration von SL noch nicht skalierbar ist, indem mehrere Instanzen von ihm hochgefahrt werden.

4.8. Actions

Die Implementierung der Actions folgt in SL einem ähnlichen Schema wie die der Assists. Auch hier wurde eine Aufspaltung in unterschiedliche und unabhängige Komponenten vorgenommen, welche jeweils bestimmte funktionelle Aspekte von Actions kapseln. Jene *Info*-, *Callable*- und *Executable*²-Komponenten werden im Folgenden näher beschrieben.

²Die Executables in diesem Kontext haben nichts mit ausführbaren Dateien gemein.

Info Die Info-Komponente von Actions enthält, genau wie die der Assurances, allgemeine Informationen zu den einzelnen Actions wie z.B. deren ID. Diese Komponente ist zur Laufzeit als Bean für jede Action verfügbar und wird an verschiedenen Stellen in der Logik von SL verwendet.

Callable Die Callable-Komponenten stellen eine zusätzliche Abstraktionsschicht für das Aufrufen des *action service* (siehe Abb. 3.17) dar. Zusätzlich kann mit ihrer Hilfe ein zu einer Action passendes Parameterpaket erzeugt werden (im Quellcode als *ActionArgs* bezeichnet). Callables stehen ebenfalls als Beans zur Laufzeit zur Verfügung und werden vom *assistance service* verwendet, um die Ausführung von Actions vom *action service* anzufordern. Die Ausführung selbst ist dabei wahlgemerkt nicht in den Callables, sondern in der folgenden dritten Komponente gekapselt.

Executable Diese Komponente einer Action enthält die Logik für ihre Durchführung. Es wird dabei zwischen der *entfernten Ausführung* und der *lokalen Ausführung* einer Action unterschieden. Erstere wird direkt vom *action service* abgehandelt, letztere nach einer entsprechenden Weiterleitung von einer Instanz des *delegate service*. Ob eine lokale Ausführung notwendig ist, hängt dabei vom zu verwendenden Aktor ab (denn die Durchführung einer Action beinhaltet immer auch die Verwendung eines Aktors). Die Details über diesen werden während der entfernten Ausführung mit Hilfe des *actuator management service* ermittelt. Sollte es sich um einen Aktor handeln, welcher lediglich lokal angesteuert werden kann, so besteht die entfernte Ausführung lediglich in der Weiterleitung an den entsprechenden Delegaten. Die entfernte Ausführung findet demnach unabhängig vom Aktor immer statt. In Abb. 4.7 sind diese Situationen noch einmal grafisch dargestellt. Die Executables aller Actions stehen zur Laufzeit als Beans zur Verfügung.

In Abb. 4.8 ist die Klassenhierarchie der Komponenten aller 11 implementierten Actions dargestellt. Aufgrund ihrer umfangreichen Zahl wird an dieser Stelle auf eine detaillierte Beschreibung der einzelnen Actions verzichtet. Ihre jeweiligen Funktionen ergeben sich jedoch in der Regel aus den selbstbeschreibenden Namen der Actions. Der interessierte Leser sei für einen tiefergehenden Einblick auf den zu dieser Abschlussarbeit gehörenden Quellcode verwiesen (siehe Abschnitt 3.4.6).

Der Grund für die Aufteilung von Actions in verschiedene Komponenten ist der gleiche wie bei den Assurances (siehe Abschnitt 4.7). Unterschiedliche Teile von SL verwenden lediglich bestimmte Aspekte der Actions. Auf diese Weise müssen nur die benötigten Komponenten konfiguriert und als Abhängigkeiten eingebunden werden.

	Aktor ist per Fernzugriff ansteuerbar	Aktor ist lediglich lokal ansteuerbar
Entfernte Ausführung (durch <i>action service</i>)	Der <i>action service</i> führt die Action selbst aus, indem er den Aktor per Fernzugriff ansteuert.	Der <i>action service</i> leitet die Ausführung an den verantwortlichen Delegaten weiter.
Lokale Ausführung (durch <i>delegate service</i>)	Fehlerfall, der nicht eintreten darf.	Der <i>delegate service</i> führt die Action aus, indem er den Aktor über seine lokale Schnittstelle ansteuert.

Abbildung 4.7: Die Ausführung einer Action erfolgt unterschiedlich, abhängig davon, ob der zu verwendende Aktor per Fernzugriff oder lediglich über eine lokale Schnittstelle ansteuerbar ist.



Abbildung 4.8: Die Info-, Callable- und Executable-Komponenten aller implementierten Actions sind in einer Klassenhierarchie organisiert. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

4.9. Aktoren

Actions stellen eine abstrahierte Form von Vorgängen dar, die von den Mitteln ihrer letztendlichen Ausführung losgelöst sind. Diese Mittel sind Aktoren, welche durch konkrete Geräte, Programme oder Webservices verkörpert werden. Für die Interaktion zwischen SL und Aktoren sind entsprechende Software-Adapter notwendig, deren Implementierung in diesem Abschnitt beschrieben wird. Dabei existiert für jeden konkreten Aktor (z.B. für ein bestimmtes Mikrofon-Modell) ein eigener Adapter, der über eine Typbezeichnung identifiziert und zugeordnet werden kann. Die Adapter unterteilen sich in verschiedene Kategorien (Mikrofone, Anzeigegeräte etc.), erben jedoch alle von einer gemeinsamen Basis-Klasse. Für jede dieser Kategorien (und damit auch für jeden der folgenden Abschnitte) existiert ein eigenes Diagramm, welches die Klassenhierarchie ihrer Adapter darstellt. All jene Diagramme sind in Anhang B zu finden.

4.9.1. Mikrofone

Die Assistance *minute taking* involviert das Aufnehmen von Audiodaten. Daher mussten Mikrofone als Aktoren an SL angebunden werden. Die Klassenhierarchie der Adapter für Mikrofon-Aktoren ist in Abb. B.1 zu sehen. Während der Entwicklung von SL wurden zwei solcher Adapter umgesetzt:

- Dummy-Implementierung, die lediglich fest codierte Audiodaten für Debugging-Zwecke liefern kann
- Adapter für das im Notebook *Lenovo Thinkpad P50* verbaute Mikrofon

Letzterer basiert funktionell auf dem Java Sound API. Dieses dient als Abstraktions-schicht für angeschlossene Audio-Hardware und bietet den Zugriff auf solche über eine einheitliche Schnittstelle an. Die Komplexität des Umgangs mit der Hardware ist dabei weitestgehend hinter Abstraktionsschichten verborgen. In SL wurde die Verwendung des Java Sound API selbst wiederum in einer generischen Mikrofon-Klasse gekapselt. Ihre Benutzung eliminiert den Großteil der Implementierungsarbeit beim Einbinden von neuen Mikrofonen in SL. (vgl. Butz 2007; Oracle o.J.[a],[b],[c])

4.9.2. Speech-to-Text-Services

Nicht nur physische Geräte können Aktoren sein, sondern auch virtuelle Komponenten wie Programme oder Webservices. Aus letzterer Kategorie stammen die Aktoren, welche während der Assistance *minute taking* verwendet werden, um aufgenommene Audiodaten in ein textuelles Transkript umzuwandeln. Die Klassenhierarchie ihrer Adapter ist in Abb. B.2 dargestellt. Während der Entwicklung von SL wurden zwei solcher Adapter umgesetzt:

- Adapter für den Webservice *Watson Speech to Text*, der Teil von IBMs KI-Plattform Watson ist
- Adapter für den Webservice *Remeeting*

Beide Dienste verfügen über ein REST-API, welches von SL angesprochen wird (vgl. IBM o.J.[a]; Remeeting 2018). Die Details dieser Kommunikation sind in den entsprechenden Adapters gekapselt. Dabei kam für Watson die von IBM stammende Java-Implementierung von dessen API zum Einsatz (vgl. IBM o.J.[c]). Für den relevanten Teil des API von Remeeting wurde ein Feign-Client geschrieben, über den HTTP-Aufrufe an den Dienst geschickt werden können³.

4.9.3. Projektablagen

Für das Hoch- und Herunterladen von Daten werden von SL die Projektablagen von Arbeitsgruppen verwendet. Diese Funktionalitäten kommen in den Assurances *minute taking* und *file displaying* zum Tragen, um einerseits Audiodaten sowie deren Transkript zu speichern und andererseits um an zu öffnende Dateien zu gelangen. Die Klassenhierarchie der Adapter für Projektablagen kann Abb. B.3 entnommen werden. Während der Entwicklung von SL wurde ein Adapter für den Onlinedienst GitHub, der webbasierte Repositorys für die Versionsverwaltung *Git* (und auch andere Systeme) zur Verfügung stellt, realisiert. GitHub bietet ein REST-API nach außen an, welches über dessen Java-Implementierung *JCabi GitHub* verwendet wurde (vgl. GitHub o.J.[a]; JCabi 2018). Für die Nutzung des implementierten Adapters wird ein eigener GitHub-Benutzer für SL benötigt (siehe Abschnitt 5.1.1).

4.9.4. Webbrowser

Die Assurances *website displaying* und *agenda showing* involvieren das Öffnen von Internetseiten. Daher mussten Webbrowser ebenfalls als Akteure für SL verfügbar gemacht werden. Die Klassenhierarchie der entsprechenden Adapter ist in Abb. B.4 zu sehen. Für die Ansteuerung von Webbrowsern wurde das Framework *Selenium* verwendet. Dieses ist eigentlich für das automatisierte Testen von Webanwendungen gedacht, kann jedoch auch in anderen Szenarien eingesetzt werden, in denen eine softwareseitige Steuerung von Webbrowsern erforderlich ist (vgl. Selenium 2018). In der ersten Iteration von SL können die beiden Browser *Firefox* und *Chrome* verwendet werden, für welche jeweils ein Adapter implementiert wurde.

³Zum Zeitpunkt der Entstehung dieses Dokuments existierte noch keine dedizierte und vollständige Java-Implementierung des API von Remeeting.

4.9.5. Programme zum Öffnen von Dateien

Programme, die zum Öffnen von Dateien verwendet werden, sind ebenfalls als Aktoren an SL angebunden. Derlei Aktoren kommen in der Assistance *file displaying* zum Einsatz. Für die erste Iteration von SL wurde ein entsprechender Adapter für das Präsentationsprogramm *PowerPoint* implementiert, dessen Klassenhierarchie in Abb. B.5 dargestellt ist. Dabei wurden keinerlei externe Bibliotheken verwendet, sondern nur Java-Bordmittel zum Starten neuer Prozesse (d.h. die Klasse *ProcessBuilder*). Mit den geeigneten Startparametern können so aus Java heraus PowerPoint-Dateien direkt in der Präsentationsansicht geöffnet werden (vgl. Microsoft o.J.[a]).

Weil für jeden Dateityp potenziell ein eigenes Programm als Aktor zum Öffnen benötigt wird, sind somit aktuell lediglich PowerPoint-Dateien mit der Assistance *file displaying* kompatibel. Es wäre auch durchaus möglich gewesen, jegliche Dateien aus Java heraus einfach mit den im Betriebssystem als Standardprogramme ausgewählten Anwendungen zu öffnen. Damit würde jedoch die Kontrolle darüber, welche Programme für SL verfügbar sein sollen, teilweise aus den Händen des Systemadministrators gegeben werden. Zusätzlich wäre so die Funktionalität von SL abhängig von Betriebssystem-Einstellungen. Die Implementierung von Programmen als separate Aktoren lässt dagegen eine feingranulare Kontrolle der möglichen Abläufe zu.

Aus offensichtlichen Gründen kann der implementierte Adapter für PowerPoint nur zusammen mit Betriebssystemen verwendet werden, auf denen PowerPoint selbst auch lauffähig ist.

4.9.6. Anzeigegeräte

Anzeigegeräte sind Aktoren, welche in beinahe allen umgesetzten Assurances zum Einsatz kommen. Dabei können sie zwei Zwecken dienen:

- Sie können zum Anzeigen von Fenstern (z.B. von einem Webbrowser) dienen, die durch eine Assistance geöffnet wurden.
- Sie können über die Assistance *device preparation* an- und ausgeschaltet werden.

Der genaue Typ solcher Aktoren spielt dabei nur für den letzteren Einsatzzweck eine Rolle. Ihre Benutzung für die Anzeige von Fenstern spielt sich rein auf Ebene der Software ab und ist damit unabhängig von der verwendeten Hardware. Die Klassenhierarchie der Anzeigegerät-Adapter ist in Abb. B.6 dargestellt. Es wird in der Implementierung zwar zwischen Beamern und Bildschirmen unterschieden, jedoch besitzen beide Kategorien in der ersten Iteration von SL dieselbe Funktionalität.

Adapter wurden für den Beamer *Hitachi CP-WU8461* und den Fernseher *Telefunk XF48B400* umgesetzt. Beide Geräte besitzen eine Infrarot-Schnittstelle, über die sie nor-

malerweise mit Hilfe einer Fernbedienung gesteuert werden können. Um mit SL kompatibel zu sein, verwenden beide Adapter eine auf dem Linux-Paket *LIRC* (Linux Infrared Remote Control) basierende Java-Implementierung, welche diese Fernbedienung ersetzt. LIRC ermöglicht es, gerätespezifische Infrarot-Signale zu senden und zu empfangen, z.B. für das Ein- und Ausschalten eines Geräts (vgl. LIRC o.J.). Hierfür wird eine LIRC-Konfigurationsdatei pro Gerät benötigt, welche die Details der Kommunikation spezifiziert. Derartige Dateien können entweder bereits fertig aus einem Repository bezogen werden oder über LIRC erzeugt werden, indem die Signale einer physischen Fernbedienung angelernt werden (vgl. LIRC 2017, o.J.). Für den oben genannten Beamer bzw. Fernseher sind die jeweiligen Konfigurationsdateien dem zu dieser Abschlussarbeit gehörenden GitHub-Repository beigefügt (siehe Abschnitt 3.4.6). Die Typbezeichnung eines Aktor-Adapters von SL und der Gerätename, der in einer LIRC-Konfigurationsdatei angegeben ist, sind maßgeblich für ihre Zuordnung. Beide Werte müssen hierfür übereinstimmen.

Natürlich wird ebenfalls ein entsprechender physischer Sender bzw. Empfänger für Infrarot-Signale benötigt, welchen LIRC benutzen kann. Auf die Mittel und Wege von dessen Bereitstellung wird in Abschnitt 5.1.1 näher eingegangen. Weil es sich bei LIRC um ein Linux-Paket handelt, ist die Nutzung von Aktor-Adaptoren, die darauf aufbauen, auf ebendieses Betriebssystem limitiert.

4.10. Grafische Benutzeroberfläche

Die Integration in die bestehende Systemlandschaft eines Arbeitsumfelds ist ein wichtiger Aspekt von SL. Daher wird die Benutzung des Systems so weit wie möglich auf bereits vorhandene GUIs ausgelagert. In der ersten Iteration von SL umfasst das hauptsächlich die Oberfläche des Kalenderdiensts von Google, falls dieser verwendet wird (siehe Abschnitt 4.5).

Darüber hinaus existieren jedoch auch eine Status-Oberfläche (siehe Abschnitt 3.10) und eine Agenda-Übersicht für Events (siehe Abschnitt 3.7.1). Beide Oberflächen wurden als Internetseiten implementiert, welche über Spring Boot ausgeliefert werden. Dabei kam die Bibliothek *Thymeleaf* zum Einsatz. Thymeleaf ist eine sogenannte *Template Engine*, die es erlaubt, zur Laufzeit programmatisch Dokumente in HTML (Hypertext Markup Language), XML (Extensible Markup Language), JavaScript etc. aus vorher angelegten Vorlagen zu generieren (vgl. Thymeleaf 2018a,b).

Abbildungen 4.9 und 4.10 zeigen die beiden implementierten Oberflächen, so wie sie von einem Webbrowser dargestellt werden. Sie basieren hauptsächlich auf HTML und besitzen lediglich einen kleinen Anteil an JavaScript-Code. Wie in den Abbildungen ersichtlich ist, ist die Optik eher schlicht gehalten, da es sich bei SL in der ersten Iteration lediglich um einen Prototypen handelt.

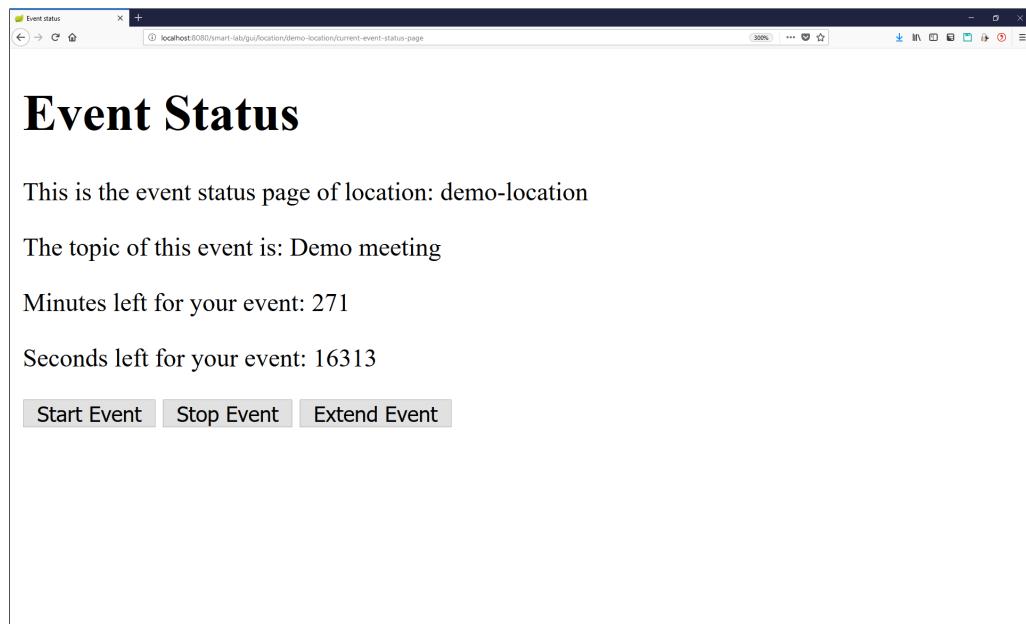


Abbildung 4.9: Die Status-Oberfläche eines Events ist als Internetseite über SL verfügbar.

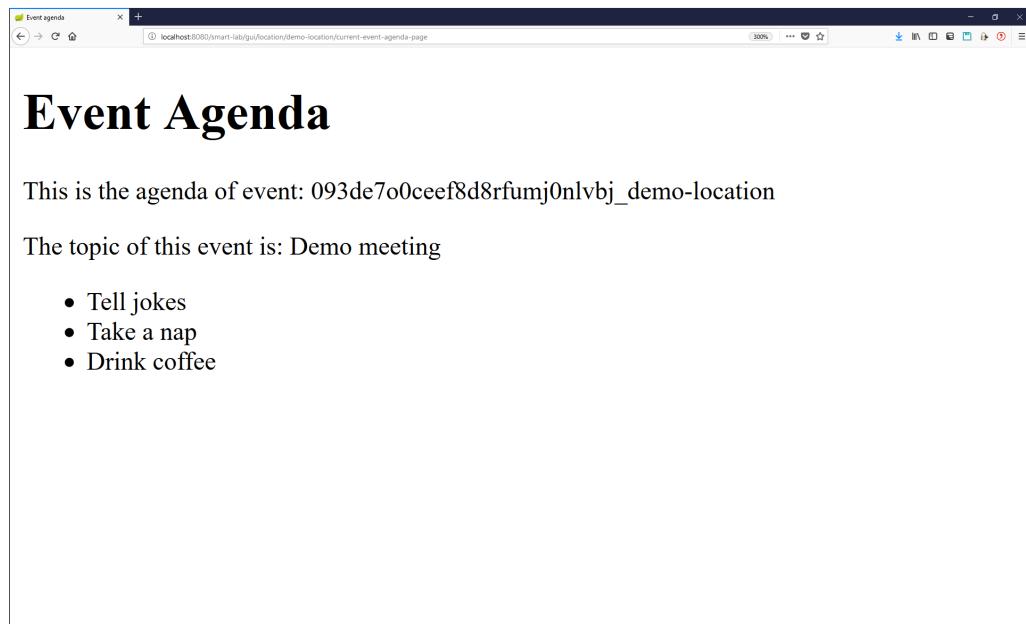


Abbildung 4.10: Die Agenda-Übersicht eines Events ist als Internetseite über SL verfügbar.

4.11. Manipulation von Programmfenstern

Während der Entwicklung von SL wurden auch Funktionalitäten umgesetzt, welche spezifisch für bestimmte Betriebssysteme sind. Neben der Verwendung von LIRC (siehe Abschnitt 4.9.6) und PowerPoint (siehe Abschnitt 4.9.5) umfasst dies insbesondere den Mechanismus zum Manipulieren von Fenstern. Die Assistenzen *website displaying*, *file displaying*

und *agenda showing* starten Programminstanzen und öffnen damit auch neue Fenster. Jede der drei Assurances bietet dem Benutzer zudem die Möglichkeit festzulegen, auf welchem Anzeigegerät diese Fenster positioniert werden sollen. Das Verschieben solcher Fenster und das Verändern ihrer Größe ist ein Vorgang, der im verwendeten Betriebssystem verankert ist und nicht aus Java heraus angesteuert werden kann. Daher wurde für SL mit der Programmiersprache C# eine Möglichkeit implementiert, um die zuvor genannten Funktionalitäten unter dem Betriebssystem Windows nutzen zu können. C# setzt auf Microsofts .NET-Plattform auf und besitzt daher eine bessere Anbindung an Windows als Java. Im Zuge dessen wurden zwei Projekte umgesetzt, welche in diesem Kapitel beschrieben sind.

4.11.1. Programmbibliothek für das Ansprechen des Windows API

Über das Projekt *smart-lab-window-handling* kann eine DLL-Datei (Dynamic Link Library) erzeugt werden, die alle für SL benötigten Aufrufe an das Windows API beinhaltet (z.B. für das Verschieben und Maximieren von Fenstern). Falls SL oder bestimmte Teile davon unter Windows ausgeführt werden, wird diese Datei zur Laufzeit verwendet, um Fenster zu manipulieren. Für das Einbinden und Aufrufen der DLL unter Java kommt *JNA* (Java Native Access) zum Einsatz. JNA erlaubt es, native Funktionen aufzurufen, ohne dass dabei etwas anderes als reiner Java-Code erforderlich ist (vgl. JNA o.J.). Dies setzt jedoch folglich voraus, dass die erzeugte und über JNA eingebundene DLL ebenfalls nativ ist. Um das mit C# zu erreichen, wurde ein *NuGet*⁴-Paket namens *DllExport* verwendet. DllExport macht es möglich, statische Methoden, welche mit einem speziellen Attribut versehen sind, in eine native DLL zu verpacken (vgl. DllExport o.J.). Der Vorgang ist dabei in den Build-Prozess eines C#-Projekts integriert (vgl. DllExport o.J.). In Listing 4.4 ist beispielhaft eine der exportierten Methoden aufgeführt.

Listing 4.4: Die hier dargestellte Methode dient zum gleichzeitigen Verschieben und Maximieren eines Programmfensters. Sie wurde mit einem von DllExport angebotenen Attribut versehen und wird während des Build-Prozesses in eine native DLL verpackt.

```
[DllImport]
public static void MoveToFullScreen(
    IntPtr windowHandle,
    string screenName)
{
    Screen screen = ScreenFromName(screenName);
    MoveToFullScreen(windowHandle, screen);
}
```

⁴NuGet ist ein in Microsofts Entwicklungsumgebung *Visual Studio* integrierter Manager für die Verteilung von Software-Komponenten.

Aus offensichtlichen Gründen kann die erzeugte DLL lediglich unter Windows verwendet werden, da sie letztendlich nur Aufrufe an das Betriebssystem weiterleitet. Dies bedeutet, dass jegliche Funktionalität für das Manipulieren von Fenstern in der ersten Iteration von SL ebenfalls auf Windows-Umgebungen beschränkt ist. Unter anderen Betriebssystemen werden sämtliche Programmfenster von Assistanzen einfach auf dem primären Anzeigegerät geöffnet.

An dieser Stelle sei noch angemerkt, dass NuGet-Pakete normalerweise nicht in einer C#-Projektmappe belassen werden, wenn sie über eine Versionsverwaltung gespeichert wird. Weil DllExport jedoch zwingend für den Build-Prozess des Projekts *smart-lab-window-handling* erforderlich ist, wurde es in dem zu SL gehörenden GitHub-Repository (siehe Abschnitt 3.4.6) manuell der entsprechenden Projektmappe beigefügt. Damit ist unter normalen Umständen eine erneute Installation des Pakets über NuGet nicht erforderlich.

4.11.2. Ermitteln der Bezeichnungen von Anzeigegeräten unter Windows

Wie in Listing 4.4 zu sehen ist, werden Anzeigegeräte in der in Abschnitt 4.11.1 beschriebenen Bibliothek über eine einfache Zeichenkette identifiziert. Diese Windows-interne Bezeichnung muss für jeden Aktor ermittelt werden, welchen SL für das Anzeigen von Programmfenstern verwenden können soll. Zu diesem Zweck wurde mit dem C#-Projekt *smart-lab-display-identifier* ein Hilfsprogramm realisiert, über das die Bezeichnungen aller angeschlossenen Anzeigegeräte ermittelt werden können. Abbildung 4.11 zeigt den Einsatz des Programms, welches im aktiven Zustand auf jedem verfügbaren Anzeigegerät ein Fenster mit dessen Bezeichnung darstellt.

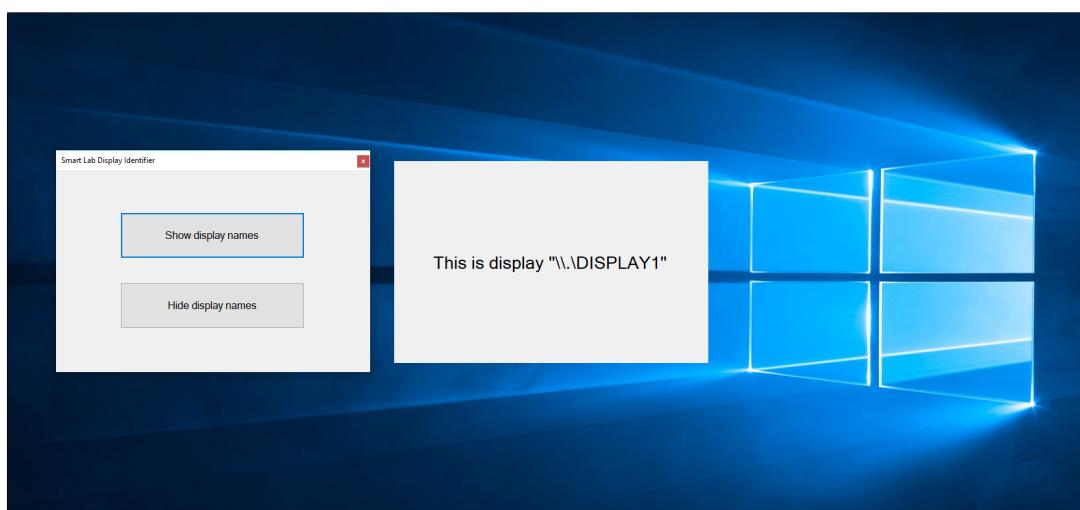


Abbildung 4.11: Die Windows-interne Bezeichnung jedes Anzeigegeräts kann über ein Hilfsprogramm ermittelt werden.

Anschließend muss eine Zuordnung der ermittelten Bezeichnung zu den in SL hinterlegten Anzeigegerät-Aktoren erfolgen. Dies ist erforderlich, da Windows inkompatibel mit deren SL-spezifischen IDs ist und SL wiederum nicht mit den Windows-spezifischen Bezeichnungen umgehen kann. Die Bezeichnungen sind lediglich pro Computer, an den die Anzeigegeräte angeschlossen sind, einzigartig, jedoch nicht über Rechnergrenzen hinweg. Das macht sie ungeeignet für die Identifizierung der Aktoren direkt in SL. Die Zuordnung der beiden Namenssysteme muss in den Konfigurationsdateien der Delegaten erfolgen, welche die entsprechenden Anzeigegeräte ansteuern.

4.12. Erweiterung von Smart Lab

Die Anforderungen und Systemlandschaften verschiedener Arbeitsumfelder sind vielgestaltig. Daher war die Möglichkeit für zukünftige Erweiterungen von SL von Beginn an ein wichtiger Aspekt bei der Entwicklung des Systems. Dessen Architektur ist so gestaltet, dass leicht neue Trigger-Signale, Assurances und Actions hinzugefügt sowie neue Arten von Aktoren und Datenquellen angebunden werden können. Dabei kommen Erweiterungen weitestgehend ohne Änderungen am bestehenden Quellcode aus. Dieser Abschnitt beschreibt in groben Zügen, was für derartige Erweiterungen nötig ist.

4.12.1. Erweiterung um neue Trigger-Signale

Neue Trigger-Signale können realisiert werden, indem der Controller- und Business-Schicht des *trigger service* (siehe Abb. 3.17) entsprechende Methoden zum Auslösen der Signale hinzugefügt werden. Zusätzlich muss in der abstrakten Basisklasse aller Assurances ein Standard-Verhalten als Reaktion auf die neuen Trigger implementiert werden. Das Verhalten existierender Assurances muss so nicht im Einzelnen erweitert werden.

4.12.2. Erweiterung um neue Assurances

Für das Hinzufügen einer neuen Assurance müssen drei neue Klassen implementiert werden. Diese entsprechen den Info-, Triggerable- und Controllable-Komponenten einer Assurance, welche in Abschnitt 4.7 beschrieben sind. Änderungen am *assurance service* müssen nicht vorgenommen werden, weil dieser so weit abstrahiert ist, dass er keine Logik enthält, die für einzelne Assurances spezifisch ist.

4.12.3. Erweiterung um neue Actions

Die Erweiterung um neue Actions folgt prinzipiell einem ähnlichen Schema wie in Abschnitt 4.12.2. Es muss je eine neue Klasse für die Info-, Callable- und Executable-Komponenten einer Action hinzugefügt werden. Auch hier ist der *action service* so weit

abstrahiert, dass er keine Logik enthält, welche für einzelne Actions spezifisch ist und muss somit auch nicht verändert werden.

4.12.4. Anbinden neuer Aktoren

Jeder Typ von Aktor (z.B. ein bestimmtes Mikrofon-Modell) besitzt in SL einen entsprechenden Software-Adapter für seine Ansteuerung. Wenn ein neues Gerät, Programm oder ein neuer Webservice an SL angebunden werden soll, so ist es lediglich erforderlich, einen entsprechenden Adapter dafür zu implementieren. Dieser kann von einer bereits vorhandenen Basisklasse erben, falls der zugehörige Aktor unter eine der in Abschnitt 4.9 beschriebenen Kategorien fällt (siehe dazu die Klassendiagramme in Anhang B). Weil sämtliche Aktor-Adapter anhand ihrer Typbezeichnung aufgelöst und für die Benutzung bereitgestellt werden, sind keine weiteren Änderungen erforderlich, um einen neuen Adapter ins System einzubinden.

4.12.5. Anbinden neuer Datenquellen

Die Anbindung von Datenquellen an SL geschieht über die Repository-Schicht der entsprechenden Services. Wenn neue Datenquellen wie Datenbanken oder Kalenderdienste mit SL kompatibel gemacht werden soll, so müssen für sie Adapter implementiert werden, welche als Repository-Schicht ins System eingehängt werden. Jeder der Datenmanagement-Services verfügt bereits über Basisklassen, von denen neue Adapter erben können (siehe Abb. 4.4 für die entsprechende Klassenhierarchie des *event management service*).

Ein besonderes Augenmerk wurde dabei der Erweiterbarkeit um neue Datenquellen für den *event management service* gewidmet. Eine zusätzliche Anforderung an das Projekt war es, dass auch weniger mächtige Kalenderdienste an SL angebunden werden können. Generell muss eine Event-Datenquelle folgende Informationen speichern können, um kompatibel zu SL zu sein:

- Titel eines Events
- Startzeitpunkt eines Events
- Endzeitpunkt eines Events
- Arbeitsgruppe, welche einem Event zugeordnet ist
- Lokalität, an der ein Event stattfindet
- Agenda eines Events
- Gewünschte Assistsances und deren Konfiguration

Wie in Abschnitt 4.6 beschrieben, können jedoch sämtliche dieser Speicherfunktionen, welche einem Kalenderdienst eventuell fehlen, über ECL kompensiert werden. Die Grammatik der Sprache muss unter Umständen dafür erweitert werden, Anpassungen von Seiten eines Kalenderdiensts sind aber nicht nötig. Damit ist praktisch jeder solche Dienst prinzipiell kompatibel zu SL.

5. Ergebnisse

Das primäre Ergebnis dieser Abschlussarbeit ist der Quellcode von SL. Der Code und dazugehörige Ressourcen wie beispielhafte Konfigurationsdateien und Skripte zum Starten des Systems sind über ein GitHub-Repository verfügbar (siehe Abschnitt 3.4.6). Die Abschnitte dieses Kapitels beschreiben einerseits die Inbetriebnahme des implementierten Systems und zeigen andererseits das Resultat von dessen qualitativer Evaluierung.

5.1. Inbetriebnahme von Smart Lab

Die Inbetriebnahme von SL verläuft in zwei Schritten. Zunächst müssen die Umgebung und die entsprechenden Konfigurationsdateien für das System vorbereitet werden, was in Abschnitt 5.1.1 beschrieben ist. Anschließend kann SL gestartet und verwendet werden. Auf die genaue Startprozedur wird in Abschnitt 5.1.2 näher eingegangen.

5.1.1. Vorbereitende Maßnahmen

Vor dem eigentlichen Starten von SL müssen diverse Vorbereitungen getroffen werden. Diese umfassen das Erstellen von Benutzerkonten für benötigte Webservices, die entsprechende Einrichtung der Systemumgebung und das Bereitstellen von passenden Konfigurationsdateien.

Erstellen von Benutzerkonten für Webservices

Im Folgenden sind alle Webservices beschrieben, für welche unter Umständen ein Benutzerkonto angelegt werden muss:

- Soll Remeeting als Speech-to-Text-Service verwendet werden, so muss ein Benutzerkonto für diesen Dienst erstellt werden. Außerdem muss ein API-Schlüssel erzeugt werden, über den SL auf den Webservice zugreifen kann.
- Kommt Watson als Speech-to-Text-Service zum Einsatz, wird ein entsprechendes Benutzerkonto bei IBMs Cloud-Plattform *Bluemix* benötigt. Zudem muss eine Instanz des Speech-to-Text-Service eingerichtet werden (vgl. IBM o.J.[b]). Dabei werden auch Zugangsdaten für diese spezielle Instanz des Diensts erstellt, über die SL Zugriff erhält.

- Falls GitHub als Projektablage verwendet werden soll, muss für SL ein eigenes GitHub-Benutzerkonto angelegt werden. Für den Zugang zum Dienst ist es außerdem erforderlich, einen Token zu erzeugen, welcher Rechte für den Zugriff auf Repositorys und Einladungen zu solchen gewährt (vgl. GitHub o.J.[b]). Der zu SL gehörende GitHub-Benutzer muss außerdem allen Repositorys, mit denen das System interagieren können soll, als Mitarbeiter hinzugefügt werden. Einladungen werden von SL automatisch angenommen, sobald ein Zugriff auf ein entsprechendes Repository erfolgt.
- In dem Fall, dass Googles Kalenderdienst als Event-Datenquelle eingesetzt werden soll, ist ein entsprechendes Benutzerkonto erforderlich. Zusätzlich muss in diesem ein spezielles Dienstkonti (englisch *service account*) eingerichtet werden, welches den softwareseitigen Zugang durch SL erlaubt. Da der Vorgang aus mehreren Schritten besteht, soll er im Folgenden näher beschrieben werden:
 1. Über die Konsole von Googles Cloud-Plattform muss ein neues Projekt mit aktiviertem Kalender-API erstellt werden (vgl. Google o.J.[a]).
 2. Innerhalb des neuen Projekts muss ein Dienstkonti erstellt werden. Dienstkonten sind für den programmatischen Zugriff auf Teile von Googles Cloud-Angebot vorgesehen. Um Berechtigungsprobleme zu vermeiden, kann zunächst die Rolle *Projekt/Eigentümer* für das Dienstkonti ausgewählt werden. Für den Produktiveinsatz ist dies natürlich nicht zu empfehlen. (vgl. Google 2018b)
 3. Zum Schluss muss für das Dienstkonti ein privater Schlüssel generiert werden. Daraufhin kann eine Datei im JSON-Format (JavaScript Object Notation) heruntergeladen werden, welche die Zugangsdaten für das Dienstkonti enthält.

Einrichten der Systemumgebung

Im Folgenden sind alle Anpassungen der Systemumgebung beschrieben, die unter Umständen für SL vorgenommen werden müssen:

- Falls Googles Kalenderdienst als Event-Datenquelle verwendet werden soll, müssen über dessen Weboberfläche für alle in SL hinterlegten Lokalitäten separate Kalender erstellt werden. Die Kalender müssen für das im vorigen Abschnitt beschriebene Dienstkonti mit der Berechtigung *Termine ändern* freigegeben werden, wofür dessen E-Mail-Adresse benötigt wird. Diese ist in der heruntergeladenen JSON-Datei zusammen mit den Zugangsdaten aufgeführt. Neu erstellte Kalender sind unter Umständen erst nach einigen Minuten über die API-Implementierung, welche SL benutzt, erreichbar.

- Sollen Assistanz-Dienste verwendet werden, die das Öffnen eines Webbrowsers involvieren, so müssen für Selenium sogenannte *Driver*-Dateien bereitgestellt werden. Hierbei handelt es sich um ausführbare Dateien und nicht, wie man vermuten könnte, um Treiber, welche sich in das Betriebssystem integrieren. Die Driver sind spezifisch für jeden über Selenium ansteuerbaren Webbrowser und können über den Webauftritt des Frameworks heruntergeladen werden (vgl. Selenium o.J.). Ihre Bereitstellung muss auf denjenigen Rechnern erfolgen, auf denen ein Delegat von SL mit der entsprechenden Funktionalität ausgeführt werden soll.
- Sollen unter Windows 10 Assistanz-Dienste verwendet werden, welche die Benutzung eines Mikrofons involvieren, muss deren Nutzung erst für SL freigegeben werden. Hierfür muss in den Datenschutzeinstellungen für Mikrofone die Option *Zulassen, dass Apps auf Ihr Mikrofon zugreifen* aktiviert werden. Diese Einstellung muss auf denjenigen Rechnern vorgenommen werden, auf denen ein Delegat von SL mit der entsprechenden Funktionalität ausgeführt werden soll.
- Sollen Assistanz-Dienste verwendet werden, welche die Benutzung von PowerPoint involvieren, muss das Programm konsequenterweise zuvor installiert werden. Die Installation muss auf denjenigen Rechnern erfolgen, auf denen ein Delegat von SL mit der entsprechenden Funktionalität ausgeführt werden soll.
- Damit Assistanz-Fenster neu positionieren können, muss die in Abschnitt 4.11.1 beschriebene DLL-Datei bereitgestellt werden. Diese kann über ein ebenfalls in Abschnitt 4.11.1 erwähntes C#-Projekt erzeugt werden. Die Bereitstellung der Datei muss auf denjenigen Rechnern erfolgen, auf denen ein Delegat von SL mit der entsprechenden Funktionalität ausgeführt werden soll.
- Falls Akteure über Infrarot-Signale angesteuert werden sollen, muss LIRC zuvor installiert und eingerichtet werden. Neben dem Linux-Paket wird auch entsprechende Hardware für das Senden (und Empfangen, falls Fernbedienungen angelernt werden sollen) der Signale benötigt.

Es existieren zahlreiche Online-Ressourcen, die den Bau einer derartigen Hardware-Schaltung und die Einrichtung von LIRC beschreiben. Zwar ist es unwahrscheinlich, dass man genau die in einer Anleitung beschriebenen elektronischen Bauteile zur Verfügung hat, jedoch spielen Art und Integrationsdichte der Bauteile prinzipiell keine große Rolle. Relevant ist lediglich, dass Sender und Empfänger für Trägersignale im Frequenzbereich von 38 kHz ausgelegt sind, welchen die meisten über Infrarot-Signale steuerbaren Geräte verwenden. So wurde im Rahmen dieser Abschlussarbeit die benötigte Schaltung lose auf einer Anleitung aus dem Internet basierend mit

gerade verfügbaren Bauteilen zusammengesetzt und an dem Einplatinencomputer *Raspberry Pi 3 Modell B+* betrieben (vgl. o.A. 2017).

Beruhend auf ebenjener Anleitung wurde auch die Einrichtung von LIRC vorgenommen¹. Die LIRC-Konfigurationsdateien für die implementierten Aktoren (siehe Abschnitt 4.9.6) sind dem Quellcode von SL beigelegt und müssen LIRC für die Benutzung verfügbar gemacht werden.

Die Einrichtung von LIRC und das Anbinden der Hardware-Schaltung muss für diejenigen Rechner erfolgen, auf denen ein Delegat von SL mit der entsprechenden Funktionalität ausgeführt werden soll.

Konfiguration von Smart Lab

Bevor SL gestartet werden kann, muss das System konfiguriert werden (im Sinne von Abschnitt 3.2). Hierfür wird ein Set aus Java-Properties-Dateien verwendet, welches vom *configuration service* (siehe Abb. 3.17) geladen und an die anderen Dienste verteilt wird. Viele Einstellungen in den Dateien beziehen sich auf Punkte aus den vorigen beiden Abschnitten. So müssen z.B. Pfade von einzubindenden Dateien und Zugangsdaten für Webservices angegeben werden. In dem zu dieser Abschlussarbeit gehörenden GitHub-Repository (siehe Abschnitt 3.4.6) sind Beispiele für solche Sets aus Konfigurationsdateien zu finden, welche als Basis für eigene Konfigurationen genutzt werden können (wobei ihr Inhalt höchstwahrscheinlich angepasst werden muss). Der *configuration service* ordnet die Konfigurationsdateien über ihre Namen den jeweiligen Diensten zu. Der Name einer Datei und der Anwendungsname eines Service müssen somit übereinstimmen. Eine Ausnahme bildet die Konfigurationsdatei *application.properties*, welche an jeden anfragenden Service verteilt wird (vgl. Spring Cloud Config o.J.[a],[b]).

Erzeugen der benötigten JAR-Dateien

Alle Anwendungen und Bibliotheken, welche die Services von SL bilden, müssen in Form von JAR-Dateien bereitgestellt gestellt. Sämtliche dieser Dateien können aus dem Quellcode erzeugt werden, indem auf das oberste Modul *smart-lab* (siehe Abb. 4.2) der Maven-Befehl `mvn clean install` angewandt wird.

5.1.2. Starten von Smart Lab

Wenn alle erforderlichen Maßnahmen aus Abschnitt 5.1.1 getroffen wurden, kann SL gestartet werden. Alle Services des Systems können dabei über das Ausführen von JAR-

¹An dieser Stelle sei noch darauf hingewiesen, dass Konfiguration sowie Benutzung von LIRC ab Version 0.94 in großem Umfang verändert wurden. Somit können diverse Online-Ressourcen zu diesen Themen unter Umständen mittlerweile veraltet sein.

Dateien, die Spring Boot Anwendungen enthalten, hochgefahren werden. Das zu dieser Abschlussarbeit gehörende GitHub-Repository (siehe Abschnitt 3.4.6) enthält Beispiel-Skripte für das Starten einer Systemkonfiguration, welche folgendes beinhaltet:

1. Der *configuration service*
2. Zwei Delegaten (von denen einer für die Ausführung auf einem anderen Rechner als die restlichen Services gedacht ist)
3. Das monolithische Kernsystem von SL

Die Skripte zeigen beispielhaft die Parametrierung der Startbefehle und können als Basis für eigene Skripte verwendet werden. Die zu übergebenden Werte umfassen z.B. das Verzeichnis der Konfigurationsdateien für den *configuration service* und die Adresse, über welche dieser von anderen Anwendungen erreicht werden kann. Die Dienste müssen in der oben aufgeführten Reihenfolge gestartet werden, damit ihre Konfiguration korrekt erfolgt und das Kernsystem nicht versucht auf noch nicht vorhandene Delegaten zuzugreifen. Nachdem alle Services korrekt hochgefahren wurden, kann SL verwendet werden. Hierfür müssen lediglich noch Events angelegt werden, in denen der Einsatz von Instances konfiguriert ist.

Das Starten des Systems als Microservices ist in der ersten Iteration von SL noch experimenteller Natur. Soll es dennoch auf diese Weise hochgefahren werden, so müssen entsprechend statt dem monolithischen Kernsystem die darin enthaltenen Dienste separat gestartet werden. Zusätzlich wird dann auch der *discovery service* (siehe Abb. 3.17) benötigt, damit die Microservices miteinander kommunizieren können.

Falls SL oder Teile davon zu Testzwecken aus einer IDE (Integrated Development Environment) heraus gestartet werden sollen, so muss zuvor die Verarbeitung von Annotationen aktiviert werden. Dies ist notwendig, da der Quellcode von SL Java-Annotationen² enthält, welche die Generierung von weiterem Code steuern, der zur Compilezeit schon bekannt sein muss. Die Aktivierung gestaltet sich je nach IDE unterschiedlich und es wird an dieser Stelle dem Leser überlassen, den für ihn passenden Weg zu ermitteln.

5.2. Evaluierung

Um den letztendlichen Mehrwert, den SL bietet, zu beurteilen, wurde eine qualitative Evaluierung des Systems durchgeführt. Dabei wurde der gesamte implementierte Funktionsumfang einer Gruppe von insgesamt acht Testpersonen demonstriert. Diese haben die

²Annotationen dieser Art stammen hauptsächlich aus der Programmiersprache *Project Lombok*. Strukturen wie Konstruktoren oder Getter- bzw. Setter-Methode, welche im Quellcode häufig wiederkehren und meist zwingend erforderlich sind, können damit auf das Angeben von Java-Annotationen reduziert werden (vgl. Project Lombok o.J.).

einzelnen Funktionalitäten anschließend in Bezug auf ihre Nützlichkeit und das Gesamtsystem in Bezug auf seine Benutzbarkeit bewertet.

Die Gruppengröße der befragten Personen ist natürlich in keiner Weise statistisch repräsentativ. Da es sich bei SL in der ersten Iteration jedoch lediglich um einen Prototypen handelt, welcher als Proof of Concept dienen soll, ist eine umfassende und endgültige Bewertung aber auch nicht sinnvoll. Vielmehr sollen die gesammelten Meinungen als Wegweiser für mögliche zukünftige Entwicklungen dienen.

Die Evaluierung ist kein zentrales Element dieser Abschlussarbeit und in Umfang sowie Komplexität recht beschränkt. Daher wurde bewusst darauf verzichtet, Versuchsaufbau, Durchführung und Ergebnisse in separaten Kapiteln zu beschreiben, auch wenn dies eigentlich dem normalen Vorgehen in wissenschaftlichen Dokumenten entspricht. Stattdessen sind alle drei jener Punkte in diesem Abschnitt aufgeführt. Die Schlüsse, die aus den Ergebnissen der Evaluierung gezogen wurden, sind dabei in den Ausblick in Kapitel 6 eingearbeitet.

Der für die Demonstration verwendete Versuchsaufbau ist in Abb. 5.1 dargestellt. Insgesamt kamen zwei Delegaten zum Einsatz, welche von einem monolithischen Kernsystem angesteuert wurden. Das Ausführen der Software wurde von einem Notebook³ (*Lenovo Thinkpad P50*) und einem Einplatinencomputer (*Raspberry Pi 3 Modell B+*) übernommen. Neben der internen Hardware der Geräte sowie der auf ihnen installierten Software wurden als Akteure noch ein Beamer (*Hitachi CP-WU8461*) und ein zusätzlicher Bildschirm verwendet. Die genaue Modellbezeichnung von letzterem ist hier nicht weiter relevant, weil er im Gegensatz zum Beamer nicht für eine direkte Ansteuerung durch SL vorgesehen war. Das Senden von Infrarot-Signalen an den Beamer wurde über eine speziell hierfür gebaute Schaltung realisiert (siehe Abschnitt 5.1.1). Diese von LIRC gesteuerte Schaltung zählt zwar an sich nicht als Akteur im Sinne von SL. Sie ist in Abb. 5.1 jedoch trotzdem als ein solcher gekennzeichnet, da sie vom Akteur-Adapter des verwendeten Beamers genutzt wurde.

Die Konfigurationsdateien, der Inhalt der Datenquellen und die Startskripte für den in Abb. 5.1 gezeigten Versuchsaufbau sind in dem zu dieser Abschlussarbeit gehörenden GitHub-Repository enthalten. Sie befinden sich in separaten Verzeichnissen in jeweils einem Ordner mit dem Namen *monolith_kurt_goedel_feedback_session*⁴. Daneben ist auch ein Konfigurationsblock in ECL beigelegt, welcher sämtliche implementierten sowie

³Das Notebook ist gleichzeitig für die Ausführung des *configuration service*, des Kernsystems und eines Delegaten verantwortlich. Diese Konstellation ist normalerweise nicht sinnvoll, da die ersten beiden eigentlich für die Ausführung auf einem zentralen Server gedacht sind und nur das System des Delegaten direkt mit lokalen Akteuren verbunden sein sollte (siehe Abschnitt 3.12). Zur Vereinfachung des Versuchsaufbaus wurde dies aber ignoriert.

⁴Der Name gründet auf der Ausführung von SL als Monolith (siehe Abschnitt 4.2) sowie auf einem Meetingraum der QAware GmbH, für den die in den Ordner enthaltenen Dateien gelten. Dieser ist nach dem Mathematiker und Philosophen *Kurt Gödel* benannt.

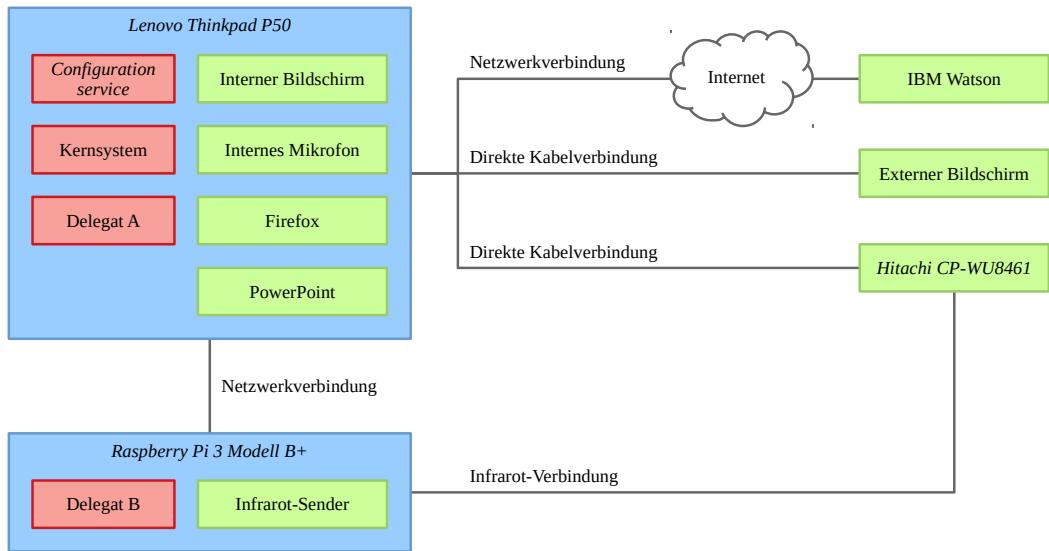


Abbildung 5.1: Die Evaluierung von SL wurde mit diesem Versuchsaufbau durchgeführt. Blaue Komponenten stellen Rechner dar, rote Komponenten sind Teile von SL und grüne Komponenten stehen für Akteure.

demonstrierten Assurances enthält. Deren genaue Parametrierung kann zusätzlich ebenso Anhang C entnommen werden.

Tabelle 5.1 zeigt den ersten Teil der Evaluierung, in welchem die Nützlichkeit verschiedener Funktionalitäten von SL bewertet wurde. Bewertet wurden alle implementierten Assurances (siehe Abschnitt 3.7.1), die lediglich als Konzept ausgearbeitete Assurance *image preservation* (siehe Abschnitt 3.7.2) und die Möglichkeit Events über ihre Status-Oberfläche auf einfache Art und Weise verlängern zu können (siehe Abschnitt 3.10).

Im zweiten Teil der Evaluierung wurde die Benutzbarkeit von SL bewertet, was in Tabelle 5.2 dargestellt ist. Der Fokus hierbei lag auf unterschiedlichen Methoden für die Konfiguration von Events. Dabei haben die Testpersonen eine Auswahl von verschiedenen solcher Methoden in eine Prioritätenreihenfolge ihrer Bevorzugung gebracht. Neben der implementierten Möglichkeit der Konfiguration über ECL umfasst die Auswahl noch weitere Methoden, welche potenzielle Weiterentwicklungen von SL darstellen. Im Konkreten sind das die Konfiguration von Events über eine dedizierte von SL bereitgestellte grafische Oberfläche, über einen Chatbot oder über einen Sprachassistenten.

Im dritten und letzten Teil der Evaluierung hatten die Testpersonen die Möglichkeit, Kritik und Vorschläge zu allen sonstigen Aspekten von SL abzugeben. Dabei hat sich in Gesprächen eine Reihe von Punkten herauskristallisiert, welche ein besonderes Gewicht besitzen. Diese Punkte sind im Folgenden aufgeführt:

Tabelle 5.1: Bei der Evaluierung der Nützlichkeit von SL wurden verschiedene Funktionalitäten mit jeweils einer Stimme pro Testperson bewertet. Die in der Tabelle aufgeführten Zahlen geben dabei an, wie viele Stimmen eine Funktionalität für einen bestimmten Grad an Nützlichkeit erhalten hat.

Funktionalität	Sehr nützlich	Mäßig nützlich	Kaum nützlich
Assistance <i>minute taking</i>	6	1	1
Assistance <i>website displaying</i>	2	6	0
Assistance <i>file displaying</i>	4	3	1
Assistance <i>agenda showing</i>	4	3	1
Assistance <i>device preparation</i>	4	3	1
Assistance <i>image preservation</i>	6	2	0
Verlängerung von Events über deren Status-Oberfläche	3	3	2

Tabelle 5.2: Bei der Evaluierung der Benutzbarkeit von SL wurde eine Auswahl von Methoden zur Konfiguration von Events durch die Testpersonen in eine Prioritätenreihenfolge gebracht. Die Prioritätsklassen reichen von 1 (stark bevorzugt) bis 4 (wenig bevorzugt). Die in der Tabelle aufgeführten Zahlen geben dabei an, wie oft einer Methode eine bestimmte Priorität zugewiesen wurde.

Methode zur Konfiguration von Events	Prioritätsklassen			
	1	2	3	4
Textuell über die Oberfläche eines Kalenderdiensts mit ECL	3	2	2	1
Grafisch über eine dedizierte Weboberfläche	5	1	1	1
Textuell über einen Chatbot	0	3	5	0
Sprachlich über einen Sprachassistenten	0	2	1	5

- Die Syntax der ECL ist bisher noch zu kompliziert und sollte weiter vereinfacht werden.
- Als Alternative zum Hochladen von Daten in eine Projektablage sollten Assists sie auch per E-Mail verschicken können.
- Eine Konfiguration von Events über einen Sprachassistenten ist ungeeignet, da die Kommunikation mit einem solchen in der Regel langatmig ausfällt. Dem schnellen Konfigurieren von Events sollte ein hoher Stellenwert eingeräumt werden.
- Die Implementierung der beiden von SL bereitgestellten Internetseiten für den Event-Status und die Agenda-Übersicht (siehe Abschnitt 4.10) sollte überarbeitet werden. Neben der Verbesserung ihrer Optik sollten sie zu vollständigen Webanwendungen

umfunktioniert werden, um z.B. auch eventuelle Fehlermeldungen anzeigen zu können.

- Die Status-Oberfläche eines Events sollte dessen nahendes Ende signalisieren (z.B. über ein akustisches Signal).
- Bei der Verlängerung eines Events über seine Status-Oberfläche sollten die Teilnehmer eines eventuellen Folge-Events über die Verschiebung ihres eigenen Termins benachrichtigt werden.

6. Fazit und Ausblick

Die zunehmende Technologisierung unserer Gesellschaft verspricht, die Art und Weise wie Menschen gemeinsam an Problemen arbeiten zu verändern. Durch digitale Assistenten kann der Arbeitsalltag effektiver und effizienter gestaltet werden. Dies ist auch notwendig, da Projekte der modernen Arbeitswelt oft so komplex sind, dass sie nur noch durch Teams bearbeitet werden können. Gleichzeitig können solche Teams mit Hilfe der umfassenden Kommunikationsmöglichkeiten unserer heutigen Zeit hochgradig verteilt arbeiten. Negative Folgeerscheinungen dieser Trends sind ein organisatorischer und kommunikativer Overhead sowie eine indirekte Interaktion zwischen Menschen in der verteilten Kollaboration. Smarte Assistenz-Systeme versuchen dem entgegenzuwirken, indem sie den Benutzern automatisierbare Anteile ihrer Tätigkeit abnehmen und bisherige technologiebedingte Grenzen bei der Zusammenarbeit abbauen.

Die in Kapitel 2 vorgestellten aktuellen Entwicklungen zeigen, dass ein tatsächlicher Bedarf für solche Systeme besteht. Einige der weltweit größten Software-Unternehmen beschäftigen sich aktuell mit diesem Themengebiet. Dabei haben mit Sicherheit auch die sprunghaften Fortschritte der letzten Jahre im Bereich der KI, Computerlinguistik und VR- (Virtual Reality) bzw. AR-Technologien diese Entwicklung begünstigt. Erstere beide versprechen, die Grenzen zwischen der Interaktion mit einem Menschen und der Interaktion mit einer Maschine verschwinden zu lassen. Letztere beide haben das Potenzial, den Aspekt der Verteilung in der verteilten Kollaboration komplett zu eliminieren. Durch virtuelle Avatare können Personen miteinander arbeiten, ganz so als ob sie an demselben physischen Ort wären. Gleichzeitig ist durch VR und AR ein völlig neuer Zugang zu bisherigen Software-Werkzeugen für die Kollaboration denkbar. Diese müssen nicht mehr zwingend an physische Eingabe- und Anzeigegeräte gebunden sein, sondern können stattdessen gänzlich im virtuellen Raum auf eine Art und Weise wirken, welche die Zusammenarbeit maximal unterstützt.

Die oben genannten Technologien sind noch nicht reif, um einen menschlichen Assistenten gänzlich zu ersetzen oder die verteilte Kollaboration komplett hinter einem virtuellen Vorhang verbergen zu können. Jedoch ist davon auszugehen, dass das Erreichen dieser Ziele lediglich eine Frage der Zeit darstellt.

Hauptziel dieser Arbeit war es, ein eigenes smartes Assistenzsystem und dessen Anwendungsfälle zu konzipieren sowie prototypisch zu implementieren. Mit SL wurde dieses

Ziel erreicht und gezeigt, dass ein solches System auch in einem kleineren Maßstab als diejenigen aus Kapitel 2 und ohne Involvieren einer KI-Komponente seinen Zweck erfüllen kann. Zusätzlich bietet SL den Mehrwert, dass es durch die gesetzten Schwerpunkte (siehe Abschnitt 3.4) eine Nische zwischen diesen Systemen ausfüllt. Die Evaluierung hat dabei gezeigt, dass ein Großteil der implementierten Funktionalitäten als nützlich empfunden wird. Dennoch ist SL in der ersten Iteration nicht für den Produktiveinsatz geeignet und muss hierfür zunächst erweitert werden. Eine zwingend notwendige Weiterentwicklung ist das Verbessern der Systemsicherheit durch das Einführen eines Authentifizierungsmechanismus und eines Autorisierungskonzepts. Doch auch abseits davon existieren viele Punkte, an denen zukünftige Iterationen von SL anknüpfen können. Im Folgenden sollen einige dieser potentiellen Erweiterungsmöglichkeiten beschrieben werden.

Systemsicherheit Die erste Iteration von SL besitzt keinen Mechanismus, um zu überprüfen, ob ein Benutzer berechtigt ist, eine bestimmte Aktion durchzuführen. Die angebotenen Services können ohne Einschränkung von jedem genutzt werden, um z.B. Trigger-Signale zu senden oder die hinterlegten Arbeitsgruppen zu ändern. Sinnvoller wäre es, das System exklusiv über ein spezielles Gateway erreichbar zu machen. Hierfür kann z.B. die Bibliothek *Zuul* verwendet werden, welche genau wie Feign und Eureka (siehe Abschnitt 4.4) ebenfalls zum Netflix-Stack gehört. Die Benutzung eines Gateways würde folgende Vorteile bieten:

- Ein Gateway kann selektiv Funktionalitäten des dahinterliegenden Systems anbieten. Dienste, die ausschließlich zu internen Zwecken verwendet werden (wie z.B. der *assistance service* und der *action service* aus Abb. 3.17), wären so von außen nicht erreichbar.
- Ein Gateway dient als zentraler Zugriffspunkt und eignet sich somit zur Authentifizierung und Überprüfung der Autorisierung.

Die Einführung eines Authentifizierungsmechanismus würde einerseits verhindern, dass Personen unzulässigerweise Services von SL verwenden. Über ein Autorisierungs- und Rollenkonzept könnte andererseits sichergestellt werden, dass Benutzer nur Zugriff auf bestimmte Daten haben. Beispielsweise sollten Benutzer nur Daten von Arbeitsgruppen ändern können, zu denen sie auch selbst gehören. Bei der Erweiterung um ein Autorisierungskonzept muss allerdings darauf geachtet werden, dass auch externe Webservices dabei einbezogen werden. Aktuell ist es z.B. möglich eine Arbeitsgruppe zu hinterlegen und dabei eine Projektablage zu spezifizieren, welche eigentlich zu einer anderen Arbeitsgruppe gehört. Auf diese Weise können über Assurances unbefugt Daten aus jeglichen GitHub-Repositorys bezogen (oder auch hinzugefügt) werden, in denen der GitHub-Benutzer von SL als Mitarbeiter spezifiziert wurde.

Eine ebenfalls notwendige Maßnahme zur Erhöhung der Systemsicherheit ist die verschlüsselte Datenübertragung mittels HTTPS (Hypertext Transfer Protocol Secure). Mit Spring Boot realisierte REST-Services können prinzipiell auch HTTPS verwenden, was in der ersten Iteration von SL jedoch zur Begrenzung des Umfangs dieser Abschlussarbeit vernachlässigt wurde. Für die Einführung der verschlüsselten Kommunikation wäre es außerdem notwendig, alle Dienste von SL mit entsprechenden Zertifikaten auszustatten.

Anwendungsfälle Die erste Iteration von SL ist auf das Bereitstellen von Assurances während Events beschränkt. Durch eine Ausweitung dieses Fokus würde sich ein neues Feld von Anwendungsfällen auftun. Weil alle bisher implementierten Trigger-Signale auf Events zugeschnitten sind, müsste somit auch das API des *trigger service* erweitert werden. Denkbar wäre z.B. ein Trigger, der signalisiert, wenn eine bestimmte Person eine Lokalität betritt, sodass diese entsprechend vorbereitet werden kann. Für das Auslösen des Triggers wäre natürlich ein entsprechender Mechanismus notwendig, welcher die Identifizierung von Personen übernimmt (etwa das Auslesen einer Schlüsselkarte oder das Auswerten von Videodaten über eine KI-Komponente).

Die Evaluierung der Nützlichkeit von SL (siehe Tabelle 5.1) deutet darauf hin, dass offenbar gerade Assurances, welche die Aufnahme und Verteilung von Daten übernehmen, für Benutzer ansprechend sind (d.h. *minute taking* und *image preservation*). Damit ist die Implementierung der bisher lediglich als Konzept existierenden Assurance *image preservation* ein geeigneter Kandidat für eine zukünftige Erweiterung von SL.

Kompatibilität Neben dem Erweitern der Kompatibilität um einzelne neue Akteure, ist auch die Anbindung ganzer Systeme an SL denkbar. Dies gilt sowohl für die Seite der Akteure als auch für die Seite der Trigger.

Letztere können z.B. durch eine Kompatibilität mit dem Webservice *IFTTT* (If This Then That) profitieren. Dieser dient zur funktionellen Verknüpfung unterschiedlicher anderer Services (vgl. IFTTT o.J.[a]). Hierzu kann eine Bedingung, die ein Service anbietet, mit einer Reaktion gekoppelt werden, welche wiederum von einem weiteren Dienst angeboten wird¹ (vgl. IFTTT o.J.[a]). Das Prinzip ist also recht ähnlich zu dem von SL. Die Anzahl der zu IFTTT kompatiblen Services ist dabei ebenso wie die Zahl von deren verfügbaren Bedingungen und Reaktionen enorm (vgl. IFTTT o.J.[b]). Über sogenannte *Webhooks* können auch benutzerdefinierte HTTP-Aufrufe als Reaktionen festgelegt werden (vgl. IFTTT o.J.[a]). Auf diese Weise könnten IFTTT-Bedingungen für das Auslösen von SL-Triggern verwendet werden, was deren

¹Ein Beispiel hierfür wäre „Wenn ich eine E-Mail erhalte, dann lasse meine (smarte) Beleuchtung blinken“.

vorstellbare Diversität ohne nennenswerten Implementierungsaufwand stark erhöhen würde.

Auf Seite der Aktoren ist eine Erweiterung von SL denkbar, welche eine Kompatibilität zu Gebäudeautomationssystemen wie openHAB (Open Home Automation Bus) herstellt. Auf diese Weise würde SL in einen größeren Automatisierungskontext eingebunden werden und könnte in Assurances auch auf klassische Funktionalitäten der Gebäudeautomation, z.B. das Öffnen und Schließen von Fenstern, zurückgreifen.

Benutzbarkeit Die Komplexität der Bedienung ist eines der Hauptprobleme von SL und stellt eine Hürde für Benutzer dar. Die Konfiguration von Events kann ausschließlich über ECL stattfinden. Dies bedeutet, dass vom Benutzer Wissen über die Syntax der Sprache und über valide Werte für die Parametrierung der Assurances vorausgesetzt wird. Beides ist für eine gute Nutzererfahrung nicht zumutbar. Daher wäre eine zusätzliche Software-Schicht, welche die Benutzbarkeit von SL verbessert, eine sinnvolle Weiterentwicklung des Systems.

Dies wurde auch durch die Evaluierung bestätigt (siehe Tabelle 5.2), in der eine dedizierte grafische Oberfläche am stärksten favorisiert wurde. Das vorausgesetzte Wissen würde hier komplett entfallen, da es keine zu verstehende Syntax mehr geben würde und valide Werte für die Parametrierung als Auswahl präsentiert werden könnten. Zwar hat auch die Konfiguration über ECL in der Evaluierung akzeptabel abgeschnitten, jedoch meist unter der Bedingung, dass ihre Komplexität weiter reduziert wird. Dies kann z.B. über einstellbare Standardwerte für die Konfiguration von Assurances erreicht werden. So könnte etwa an einer bestimmten Lokalität immer ein bestimmtes Mikrofon für die Aufnahme von Audiodaten verwendet werden. Die Bedienung über einen Chatbot oder Sprachassistenten würde zwar dem heutigen Trend folgen, wurde aber in der Evaluierung weniger favorisiert. Der Grund hierfür ist, dass eine langatmige und umständliche Kommunikation befürchtet wurde.

Technisches SL kann als Monolith oder als Microservices ausgeführt werden (siehe Abschnitt 4.2). Letztere Variante ist bislang jedoch experimentell. Zwar lassen sich alle Services von SL auf diese Weise starten, ihre Kommunikation verläuft aber noch nicht fehlerfrei. Weil sie Daten speichern, sind manche der Dienste zusätzlich zustandsbehaftet und damit auch gleichzeitig nicht durch das Hinzufügen neuer Service-Instanzen skalierbar (dies gilt z.B. für den *assistance service* aus Abb. 3.17). Um ein vollständig in Containern ausführbares und skalierbares System zu erhalten, müssen die eben genannten Defizite in einer zukünftigen Iteration von SL zunächst beseitigt werden.

Die Aufnahme von Audiodaten erfolgt durch das Java Sound API standardmäßig in unkomprimierter Form. Aktuell bedeutet dies, dass 10 Minuten an aufgenomme-

nen Daten ca. 25 MB an Speicher beanspruchen, was für die Protokollierung eines durchschnittlichen Meetings nicht akzeptabel ist. Der freie Audiocodec *Speex* ist auf die Komprimierung von Daten spezialisiert, die menschliche Sprache enthalten (vgl. Valin 2007). Eine sinnvolle Weiterentwicklung von SL ist somit die Integration der ebenfalls frei verfügbaren Java-Implementierung dieses Codecs namens *JSpeex*.

Die Testabdeckung von SL ist unzureichend. Es existieren lediglich Integrationstests für die Datenmanagement-Services, welche im Maven Modul *smart-lab-integration-test* (siehe Abschnitt 4.2) enthalten sind. Zur Verbesserung der Quellcode-Qualität empfiehlt es sich, weitere Integrationstests und vor allem Modultests hinzuzufügen.

Zusätzlich zu den in den vorigen Absätzen erwähnten Punkten sind im Quellcode von SL To-do-Markierungen hinterlegt. Diese beschreiben weitere technische Defizite, welche nicht gravierend genug sind, um sie an dieser Stelle im Detail zu erläutern. Für zukünftige Iterationen von SL sollte ihre Beseitigung dennoch in Betracht gezogen werden.

Der Umfang der vorangehenden Auflistung zeigt deutlich, dass noch ausreichend Potenzial für weitere Iterationen von SL existiert. Die im dritten Teil der Evaluierung ermittelten Kritiken und Anregungen der Testpersonen sind dabei ebenso zu berücksichtigen. Um diese Arbeit zu schließen, sei noch einmal darauf hingewiesen, dass der Code von SL quell offen ist und für den interessierten Leser über das zu dieser Abschlussarbeit gehörende GitHub-Repository verfügbar ist (siehe Abschnitt 3.4.6).

A. HTTP-Endpunkte der verschiedenen Services von Smart Lab

Tabelle A.1: Das API des *GUI service*.

Ressource	HTTP-Methode	Auswirkung
<code>smart-lab/gui/location/<ID>/current-event-status-page</code>	GET	Ruft die Status-Oberfläche des aktuell an der spezifizierten Lokalität laufenden Events auf.
<code>smart-lab/gui/location/<ID>/current-event-agenda-page</code>	GET	Ruft die Agenda-Übersicht des aktuell an der spezifizierten Lokalität laufenden Events auf.

Tabelle A.2: Das API des *job service*.

Ressource	HTTP-Methode	Auswirkung
<code>smart-lab/api/job/<ID></code>	GET	Ruft Informationen über die spezifizierte asynchrone Aufgabe ab.
<code>smart-lab/api/jobs/all</code>	GET	Ruft Informationen über alle im System angelegten asynchronen Aufgaben ab.
<code>smart-lab/api/job</code>	POST	Legt eine neue asynchrone Aufgabe im System an.
<code>smart-lab/api/job/<ID>/processing</code>	POST	Markiert die spezifizierte asynchrone Aufgabe als in Bearbeitung befindlich.
<code>smart-lab/api/job/<ID>/finished</code>	POST	Markiert die spezifizierte asynchrone Aufgabe als abgeschlossen.
<code>smart-lab/api/job/<ID>/failed</code>	POST	Markiert die spezifizierte asynchrone Aufgabe als fehlgeschlagen.

Tabelle A.3: Das API des *trigger service*.

Ressource	HTTP-Methode	Auswirkung
<code>smart-lab/api/trigger/ set-up-current-event/location/<ID></code>	POST	Löst das Trigger-Signal <i>set up event</i> für das aktuell an der spezifizierten Lokalität laufende Event aus.
<code>smart-lab/api/trigger/ set-up-current-event/workgroup/<ID></code>	POST	Löst das Trigger-Signal <i>set up event</i> für das aktuell laufende Event der spezifizierten Arbeitsgruppe aus.
<code>smart-lab/api/trigger/ clean-up-current-event/location/<ID></code>	POST	Löst das Trigger-Signal <i>clean up event</i> für das aktuell an der spezifizierten Lokalität laufende Event aus.
<code>smart-lab/api/trigger/ clean-up-current-event/workgroup/ <ID></code>	POST	Löst das Trigger-Signal <i>clean up event</i> für das aktuell laufende Event der spezifizierten Arbeitsgruppe aus.
<code>smart-lab/api/trigger/ start-current-event/location/<ID></code>	POST	Löst das Trigger-Signal <i>start event</i> für das aktuell an der spezifizierten Lokalität laufende Event aus.
<code>smart-lab/api/trigger/ start-current-event/workgroup/<ID></code>	POST	Löst das Trigger-Signal <i>start event</i> für das aktuell laufende Event der spezifizierten Arbeitsgruppe aus.
<code>smart-lab/api/trigger/ stop-current-event/location/<ID></code>	POST	Löst das Trigger-Signal <i>stop event</i> für das aktuell an der spezifizierten Lokalität laufende Event aus.
<code>smart-lab/api/trigger/ stop-current-event/workgroup/<ID></code>	POST	Löst das Trigger-Signal <i>stop event</i> für das aktuell laufende Event der spezifizierten Arbeitsgruppe aus.

Tabelle A.4: Das API des *person management service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/person/<ID>	GET	Ruft Informationen über die spezifizierte Person ab.
smart-lab/api/persons	GET	Ruft Informationen über die spezifizierten Personen ab.
smart-lab/api/persons/all	GET	Ruft Informationen über alle im System angelegten Personen ab.
smart-lab/api/person	POST	Legt eine neue Person im System an.
smart-lab/api/persons	POST	Legt mehrere neue Personen im System an.
smart-lab/api/person/<ID>	DELETE	Löscht die spezifizierte Person aus dem System.

Tabelle A.5: Das API des *actuator management service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/actuator/<ID>	GET	Ruft Informationen über den spezifizierten Aktor ab.
smart-lab/api/actuators	GET	Ruft Informationen über die spezifizierten Aktoren ab.
smart-lab/api/actuators/all	GET	Ruft Informationen über alle im System angelegten Aktoren ab.
smart-lab/api/actuator	POST	Legt einen neuen Aktor im System an.
smart-lab/api/actuators	POST	Legt mehrere neue Aktoren im System an.
smart-lab/api/actuator/<ID>	DELETE	Löscht den spezifizierten Aktor aus dem System.

Tabelle A.6: Das API des *location management service*.

Ressource	HTTP-Methode	Auswirkung
<code>smart-lab/api/location/<ID></code>	GET	Ruft Informationen über die spezifizierte Lokalität ab.
<code>smart-lab/api/locations</code>	GET	Ruft Informationen über die spezifizierten Lokalitäten ab.
<code>smart-lab/api/locations/all</code>	GET	Ruft Informationen über alle im System angelegten Lokalitäten ab.
<code>smart-lab/api/location</code>	POST	Legt eine neue Lokalität im System an.
<code>smart-lab/api/locations</code>	POST	Legt mehrere neue Lokalitäten im System an.
<code>smart-lab/api/location/<ID></code>	DELETE	Löscht die spezifizierte Lokalität aus dem System.
<code>smart-lab/api/location/<ID>/events</code>	GET	Ruft Informationen über alle Events an der spezifizierten Lokalität ab.
<code>smart-lab/api/location/<ID>/current-event</code>	GET	Ruft Informationen über das aktuell an der spezifizierten Lokalität laufende Event ab.
<code>smart-lab/api/location/<ID>/extend-current-event</code>	POST	Verlängert das aktuell an der spezifizierten Lokalität laufende Event.

Tabelle A.7: Das API des *workgroup management service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/workgroup/<ID>	GET	Ruft Informationen über die spezifizierte Arbeitsgruppe ab.
smart-lab/api/workgroups	GET	Ruft Informationen über die spezifizierten Arbeitsgruppen ab.
smart-lab/api/workgroups/all	GET	Ruft Informationen über alle im System angelegten Arbeitsgruppen ab.
smart-lab/api/workgroup	POST	Legt eine neue Arbeitsgruppe im System an.
smart-lab/api/workgroups	POST	Legt mehrere neue Arbeitsgruppen im System an.
smart-lab/api/workgroup/<ID>	DELETE	Löscht die spezifizierte Arbeitsgruppe aus dem System.
smart-lab/api/workgroup/<ID>/events	GET	Ruft Informationen über alle Events der spezifizierten Arbeitsgruppe ab.
smart-lab/api/workgroup/<ID>/current-event	GET	Ruft Informationen über das aktuell laufende Event der spezifizierten Arbeitsgruppe ab.
smart-lab/api/workgroup/<ID>/extend-current-event	POST	Verlängert das aktuell laufende Event der spezifizierten Arbeitsgruppe.

Tabelle A.8: Das API des *event management service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/event/<ID>	GET	Ruft Informationen über das spezifizierte Event ab.
smart-lab/api/event/current/at/location/<ID>	GET	Ruft Informationen über das aktuell an der spezifizierten Lokalität laufende Event ab.
smart-lab/api/event/current/of/workgroup/<ID>	GET	Ruft Informationen über das aktuell laufende Event der spezifizierten Arbeitsgruppe ab.
smart-lab/api/events	GET	Ruft Informationen über die spezifizierten Events ab.
smart-lab/api/events/all/current	GET	Ruft Informationen über alle aktuell laufenden Events ab.
smart-lab/api/events/all/at/location/<ID>	GET	Ruft Informationen über alle Events an der spezifizierten Lokalität ab.
smart-lab/api/events/all/of/workgroup/<ID>	GET	Ruft Informationen über alle Events der spezifizierten Arbeitsgruppe ab.
smart-lab/api/events/all	GET	Ruft Informationen über alle im System angelegten Events ab.
smart-lab/api/event	POST	Legt ein neues Event im System an.
smart-lab/api/events	POST	Legt mehrere neue Events im System an.
smart-lab/api/event/<ID>	DELETE	Löscht das spezifizierte Event aus dem System.
smart-lab/api/event/<ID>/shorten	PUT	Verkürzt das spezifizierte Event.
smart-lab/api/event/<ID>/extend	PUT	Verlängert das spezifizierte Event.
smart-lab/api/event/<ID>/shift	PUT	Verschiebt das spezifizierte Event.

Tabelle A.9: Das API des *assistance service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/assistance/begin	POST	Führt die Phase <i>begin</i> einer Assistance aus.
smart-lab/api/assistance/end	POST	Führt die Phase <i>end</i> einer Assistance aus.
smart-lab/api/assistance/during	POST	Führt die Phase <i>during</i> einer Assistance aus.

Tabelle A.10: Das API des *action service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/action/<ID>/execute	POST	Führt die spezifizierte Action aus.

Tabelle A.11: Das API des *delegate service*.

Ressource	HTTP-Methode	Auswirkung
smart-lab/api/delegate/execute/ action/<ID>/with/actuator/<Typ>	POST	Führt die spezifizierte Action unter Verwendung des spezifizierten Aktor-Typs aus.

B. Klassenhierarchien der verschiedenen Kategorien von Aktor-Adaptoren

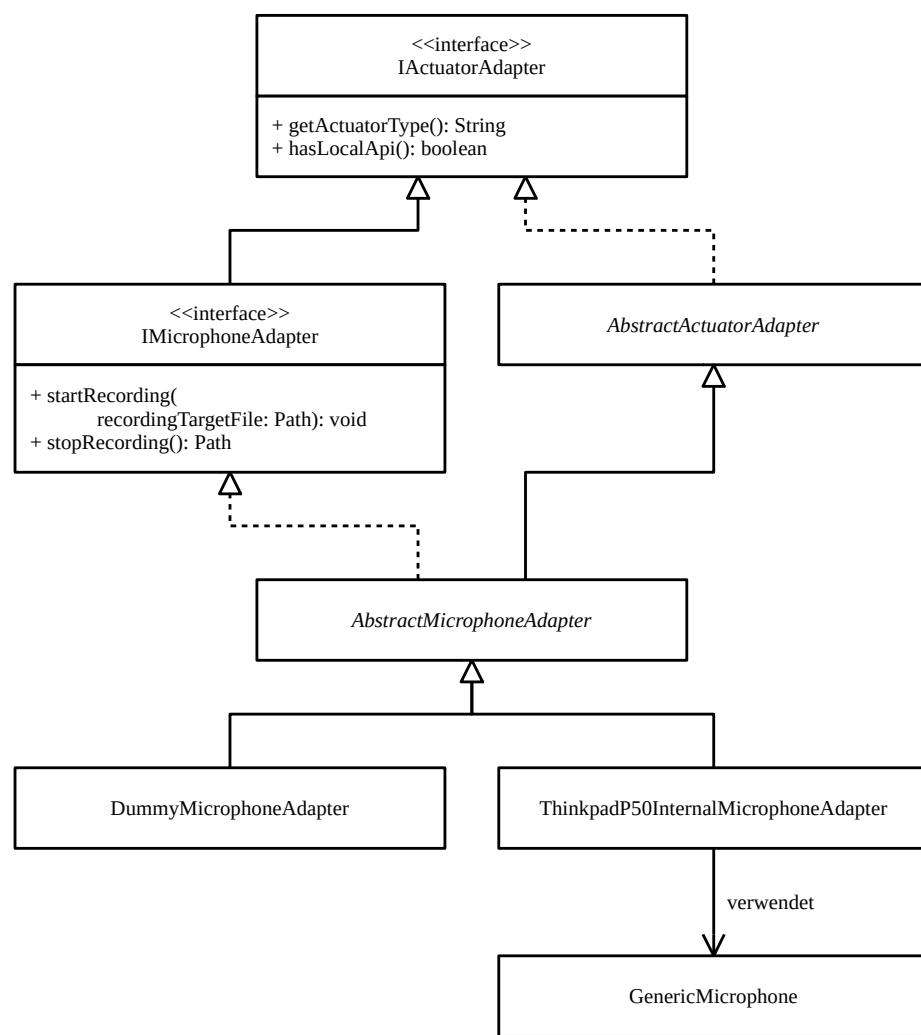


Abbildung B.1: Die Klassenhierarchie der implementierten Adapter für Mikrofon-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

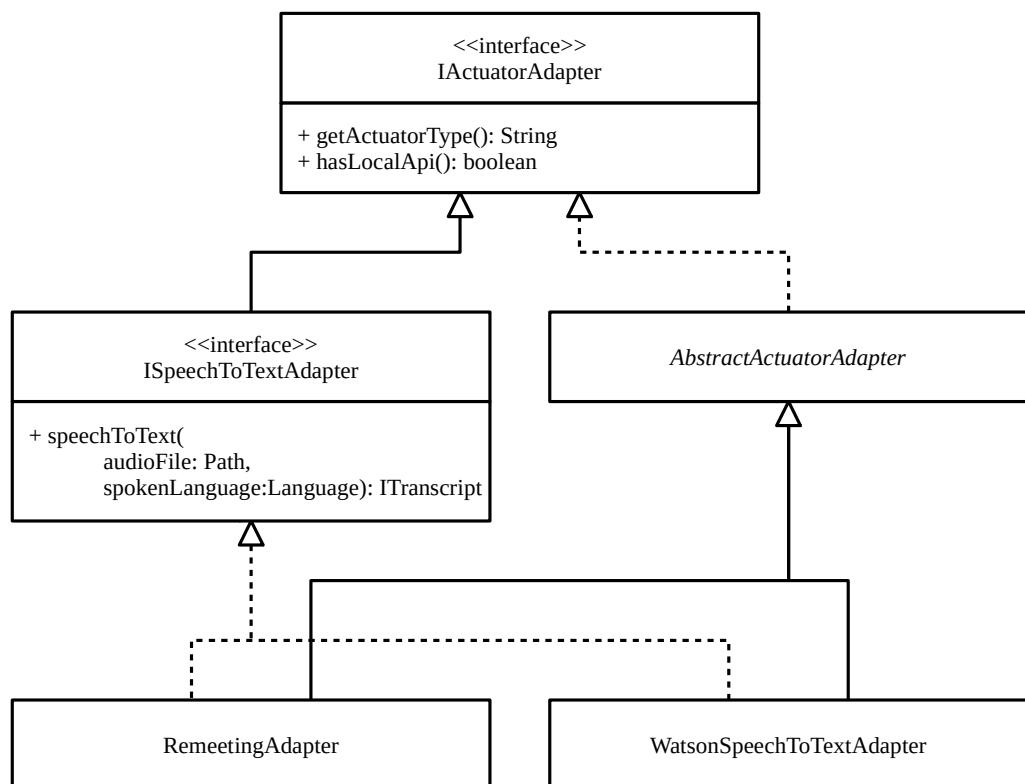


Abbildung B.2: Die Klassenhierarchie der implementierten Adapter für Speech-to-Text-Service-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

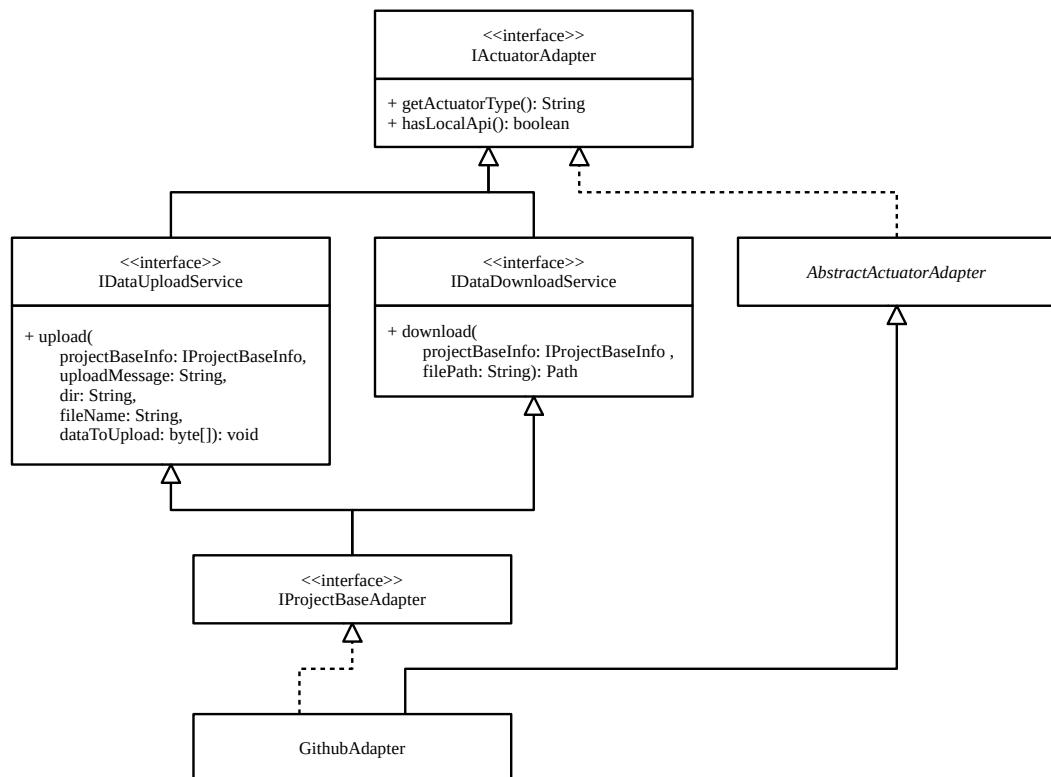


Abbildung B.3: Die Klassenhierarchie der implementierten Adapter für Projektablage-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

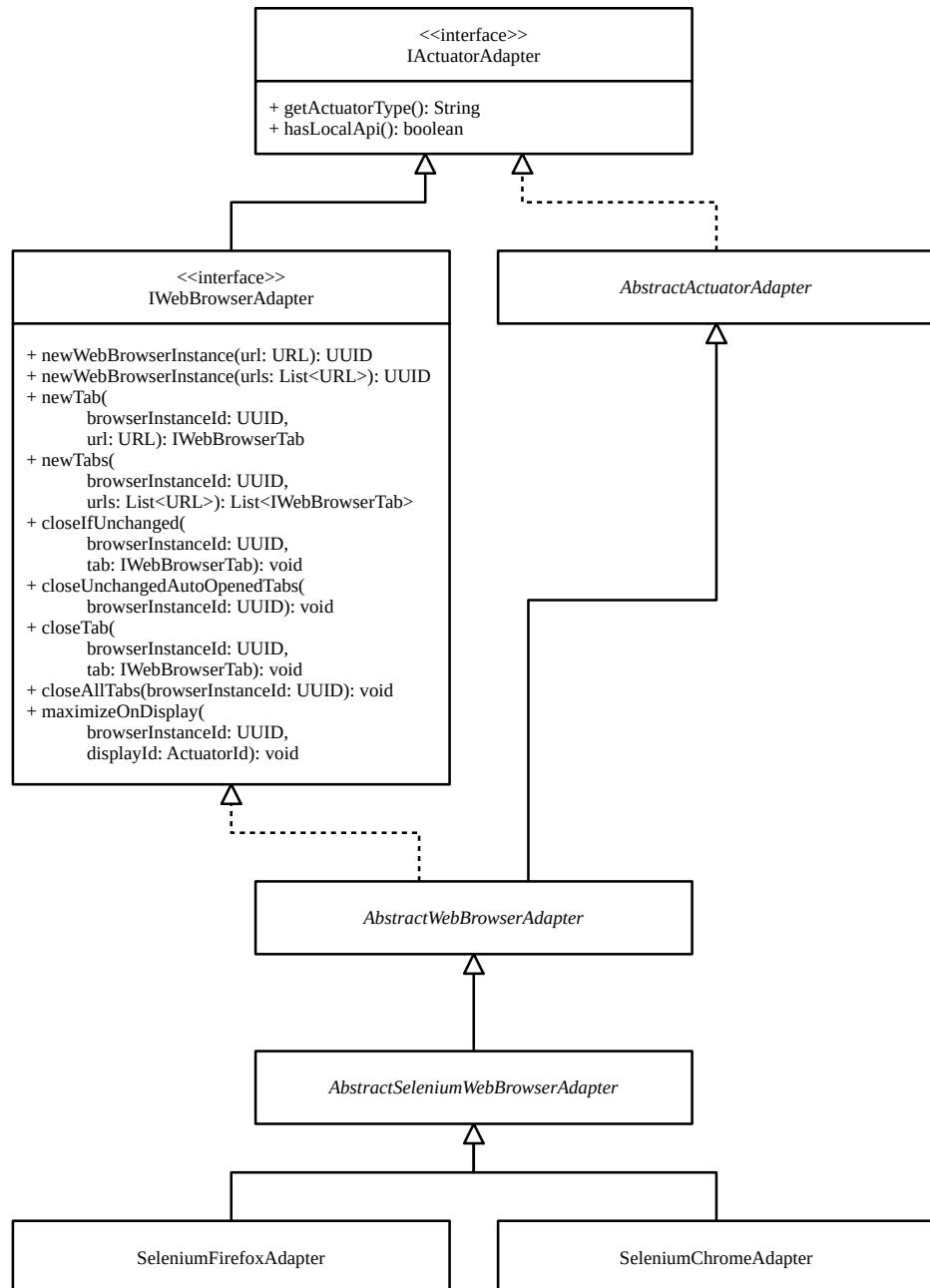


Abbildung B.4: Die Klassenhierarchie der implementierten Adapter für Webbrowser-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

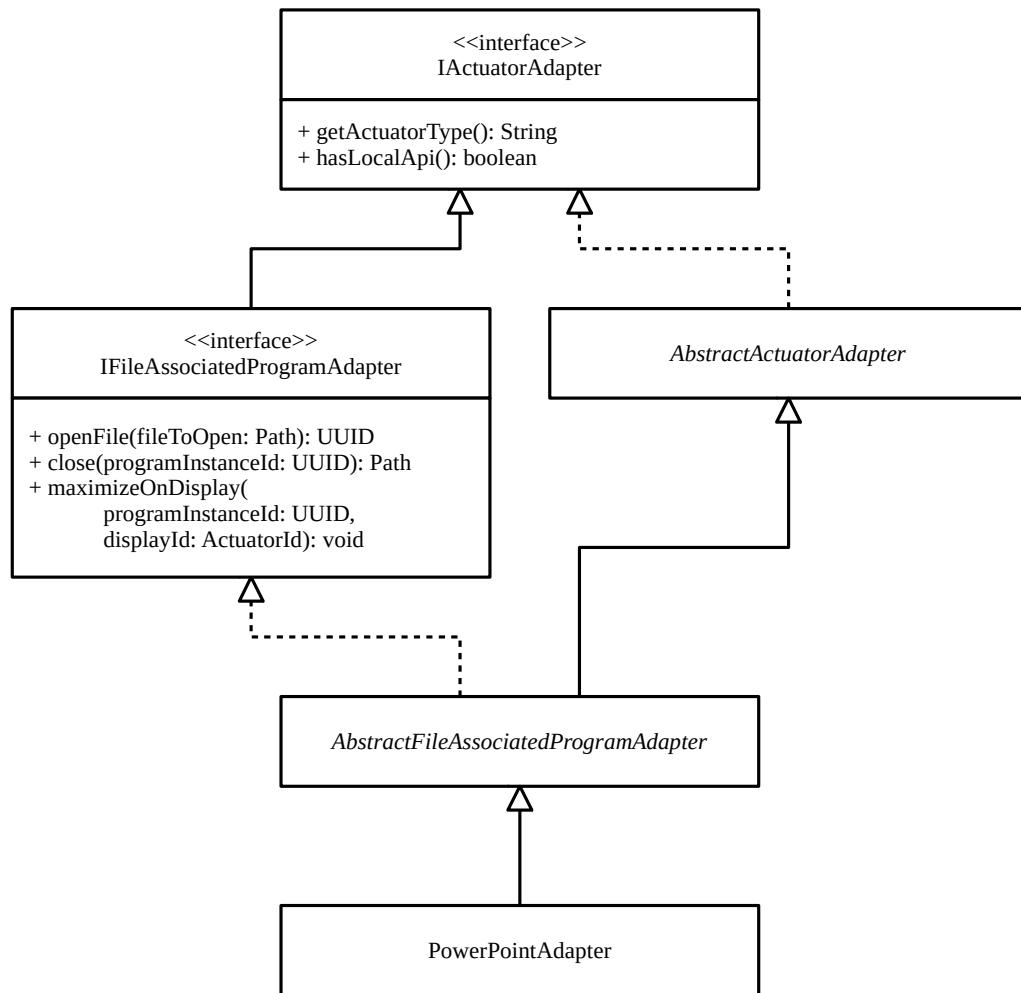


Abbildung B.5: Die Klassenhierarchie der implementierten Adapter für Programm-Aktoren zum Öffnen von Dateien. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

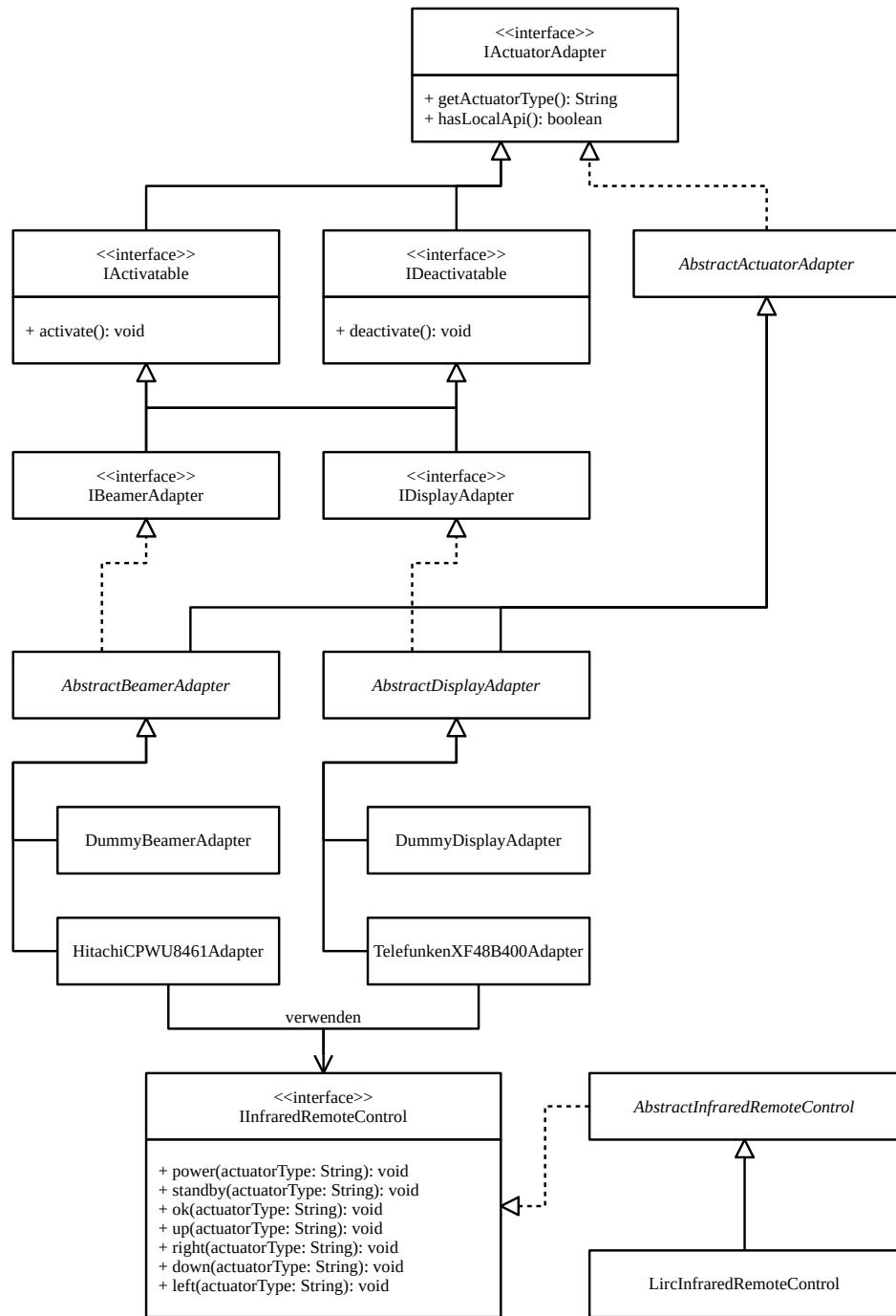


Abbildung B.6: Die Klassenhierarchie der implementierten Adapter für Anzeigegerät-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.

C. Konfiguration über ECL

Listing C.1: Im hier aufgeführten Konfigurationsblock in ECL sind die Arbeitsgruppe und Agenda festgelegt sowie alle implementierten Assistanzien aktiviert. Diese Konfiguration wurde auch für die Evaluierung von SL verwendet (siehe Abschnitt 5.2).

```
@smart-lab-config-begin
  @workgroup="demo-workgroup"
  @agenda-begin
    "Tell jokes"
    "Take a nap"
    "Drink coffee"
  @agenda-end
  @assistances-begin
    showAgenda(
      webBrowserId: "demo-web-browser",
      displayId: "demo-display")
    displayFile(
      file: "slides/smart_lab.pptx",
      programId: "demo-power-point",
      displayId: "demo-beamer")
    prepareDevice(
      deviceId: "demo-beamer")
    displayWebsite(
      url: "https://www.qaware.de/",
      webBrowserId: "demo-web-browser",
      displayId: "demo-display")
    takeMinutes(
      spokenLanguage: "EN_US",
      uploadDir: "/minutes",
      microphoneId: "demo-microphone")
  @assistances-end
@smart-lab-config-end
```

Quellenverzeichnis

- Amazon.com, Inc. (o.J.[a]). *Alexa for Business*. URL: <https://aws.amazon.com/de/alexaforbusiness/> (besucht am 27.02.2018).
- (o.J.[b]). *Alexa helps you around the workplace*. URL: <https://d1.awsstatic.com/product-marketing/A4B/Alexa%20helps%20you%20around%20the%20workplace.f853c32d564510c5295cef9997b28e92d4c1e4e2.png> (besucht am 07.10.2018).
 - (Dez. 2017). *AWS re:Invent 2017 - Introducing Alexa for Business*. URL: <https://www.youtube.com/watch?v=I-KEn6gKm7E> (besucht am 28.02.2018).
- ANTLR (Dez. 2016). *ANTLR 4 Documentation*. URL: <https://github.com/antlr/antlr4/blob/4.6/doc/index.md> (besucht am 24.06.2018).
- (Juli 2011). *Quick Starter on Parser Grammars - No Past Experience Required*. URL: <https://theantlrguy.atlassian.net/wiki/spaces/ANTLR3/pages/2687210/Quick+Starter+on+Parser+Grammars+-+No+Past+Experience+Required> (besucht am 23.06.2018).
- Ballew, Joli (Aug. 2018). *Cortana: Everything You Need to Know About Microsoft's Virtual Assistant*. URL: <https://www.lifewire.com/microsoft-cortana-4147978> (besucht am 24.08.2018).
- Butz, Andreas (2007). *Toneinbindung und Tonbearbeitung in Java*. Ludwig-Maximilians-Universität München. URL: https://www.medien_ifi.lmu.de/lehre/ss07/mt/mtB4.pdf (besucht am 10.05.2018).
- Davis, Collin und Akersh Srivastava (Dez. 2017). *AWS re:Invent 2017 - Voice is everywhere (ALX204)*. Amazon.com, Inc. URL: <https://www.youtube.com/watch?v=NMVeBgZ6kXM> (besucht am 27.02.2018).
- DllExport (o.J.). *DllExport GitHub-Repository*. URL: <https://github.com/3F/DllExport> (besucht am 20.07.2018).
- Feign (o.J.). *Feign GitHub-Repository*. URL: <https://github.com/OpenFeign/feign> (besucht am 09.05.2018).

- GitHub, Inc. (o.J.[a]). *GitHub Developer REST API v3*. URL: <https://developer.github.com/v3/> (besucht am 22.05.2018).
- (o.J.[b]). *Personal access tokens*. URL: <https://github.com/settings/tokens> (besucht am 08.10.2018).
- Google LLC (o.J.[a]). *Anwendung für Google Calendar API in der Google API Console anmelden*. URL: <https://console.developers.google.com/flows/enableapi?apiId=calendar&pli=1> (besucht am 08.10.2018).
- (Sep. 2018a). *Google Calendar API - Java Quickstart*. URL: <https://developers.google.com/calendar/quickstart/java> (besucht am 10.10.2018).
- (o.J.[b]). *Google Jamboard*. URL: https://lh3.googleusercontent.com/h83Sym_hiFaayWsXa28W3FIa1JzUOu_kJBfUF5shRYJwq_SOWNRA9aUf3UHvPUROCJiJdLYRyAj0GG900dn90KnvH6-XTPQfuYWI=w920 (besucht am 07.10.2018).
- (o.J.[c]). *Google Jamboard*. URL: <https://gsuite.google.com/products/jamboard/> (besucht am 09.03.2018).
- (o.J.[d]). *Google Jamboard Help*. URL: <https://support.google.com/jamboard/#topic=7383643> (besucht am 09.03.2018).
- (Okt. 2018b). *Using OAuth 2.0 for Server to Server Applications*. URL: <https://developers.google.com/identity/protocols/OAuth2ServiceAccount> (besucht am 10.10.2018).
- Hachman, Mark (Mai 2018). *Cortana gets the power of sight in Microsoft's future vision of conference rooms*. URL: <https://www.pcworld.com/article/3270645/windows/cortana-gets-the-power-of-sight-in-microsofts-future-vision-of-conference-rooms.html> (besucht am 16.05.2018).
- Herrera, Esteban (Juni 2016). *Introduction to testing with BDD and the Spock Framework*. URL: <https://www.pluralsight.com/guides/introduction-to-testing-with-bdd-and-the-spock-framework> (besucht am 18.04.2018).
- Hoy, Matthew B. (2018). „Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants“. In: *Medical Reference Services Quarterly* 37.1. PMID: 29327988, S. 81–88. DOI: 10.1080/02763869.2018.1404391. eprint: <https://doi.org/10.1080/02763869.2018.1404391>. URL: <https://doi.org/10.1080/02763869.2018.1404391>.
- IBM Corporation (o.J.[a]). *Speech to Text - Getting started tutorial*. URL: <https://console.bluemix.net/docs/services/speech-to-text/getting-started.html#gettingStarted> (besucht am 25.05.2018).
- (o.J.[b]). *Speech to Text - IBM Cloud*. URL: <https://console.bluemix.net/catalog/services/speech-to-text> (besucht am 08.10.2018).

IBM Corporation (o.J.[c]). *Watson APIs Java SDK GitHub-Repository*. URL: <https://github.com/watson-developer-cloud/java-sdk> (besucht am 25.05.2018).

IFTTT (o.J.[a]). *IFTTT Help Center*. URL: <https://helpifttt.com/hc/en-us> (besucht am 08.10.2018).

– (o.J.[b]). *See all services - IFTTT*. URL: <https://ifttt.com/services> (besucht am 08.10.2018).

JCabi (Juni 2018). *Object Oriented Github API*. URL: <https://github.jcabi.com/> (besucht am 21.06.2018).

JNA (o.J.). *JNA GitHub-Repository*. URL: <https://github.com/java-native-access/jna> (besucht am 19.07.2018).

Kapko, Matt (Feb. 2018). *Cortana explained: How to use Microsoft's virtual assistant for business*. URL: <https://www.computerworld.com/article/3252218/collaboration/cortana-explained-why-microsofts-virtual-assistant-is-wired-for-business.html> (besucht am 17.05.2018).

Lang, Robert D. und Lenore E. Benessere (Juni 2018). „Virtual Assistants in the Workplace: Real, Not Virtual Pitfalls and Privacy Concerns“. In: *Journal of Internet Law* 21.12. URL: <http://www.damato-lynch.com/app/uploads/2018/07/Virtual-Assistants-in-the-Workplace-Real-Not-Virtual-Pitfalls-and-Privacy-Concerns.pdf>.

LIRC (Feb. 2017). *Remotes Database*. URL: <http://lirc-remotes.sourceforge.net/remotes-table.html> (besucht am 28.07.2018).

– (o.J.). *Welcome to the LIRC 0.10.0rc1 Manual*. URL: <http://www.lirc.org/html/index.html> (besucht am 28.07.2018).

Microsoft Corporation (o.J.[a]). *Command-line switches for Microsoft Office products*. URL: https://support.office.com/en-us/article/command-line-switches-for-microsoft-office-products-079164cd-4ef5-4178-b235-441737deb3a6?ocmsassetID=HA010153889&CTT=1&CorrelationId=b9563055-15ea-4339-832d-2d0e76a7f7f1&ui=en-US&rs=en-US&ad=US#ID0EAABAAA=PowerPoint,_PowerPoint_Viewer (besucht am 13.07.2018).

– (o.J.[b]). *Cortana Dev Center*. URL: <https://developer.microsoft.com/de-de/cortana> (besucht am 30.05.2018).

– (o.J.[c]). *Cortana, Ihre persönliche digitale Assistentin*. URL: <https://www.microsoft.com/de-de/windows/cortana> (besucht am 16.05.2018).

– (Mai 2018a). *Microsoft Build 2018 - Modern Meetings Demo*. URL: <https://www.youtube.com/watch?v=dbb3ZgAp9TA> (besucht am 16.05.2018).

- Microsoft Corporation (Mai 2018b). *Modern Meetings Demo*. URL: <https://developer.microsoft.com/en-us/events/build/content/modern-meetings-demo?playlist=9f769669-4695-4eba-a14c-6ac7bb0001ba> (besucht am 16.05.2018).
- Nadareishvili, Irakli et al. (Juni 2016). *Microservice Architecture. Aligning Principles, Practices, and Culture*. 1. Aufl. O'Reilly Media.
- Niederwieser, Peter (Jan. 2015). *Testing Java, Groovy, Spring and Web Applications with Spock*. URL: <https://www.infoq.com/presentations/groovy-test-java-spock> (besucht am 17.04.2018).
- Niederwieser, Peter et al. (Sep. 2018). *Spock Framework Reference Documentation*. URL: <http://spockframework.org/spock/docs/1.2/index.html> (besucht am 10.10.2018).
- o.A. (Juni 2017). *Raspberry Pi: Mit LIRC Infrarot-Befehle senden*. URL: <https://indibit.de/raspberry-pi-mit-lirc-infrarot-befehle-senden-irsend/> (besucht am 28.07.2018).
- Oostergo, Milo (Dez. 2017). *AWS re:Invent 2017 - Building smart meeting rooms with Alexa for Business (BAP309)*. Amazon.com, Inc. URL: <https://www.youtube.com/watch?v=rby3Lce5bWo> (besucht am 27.02.2018).
- Oracle Corporation (o.J.[a]). *Capturing Audio*. URL: <https://docs.oracle.com/javase/tutorial/sound/capturing.html> (besucht am 11.05.2018).
- (o.J.[b]). *Java Sound Programmer Guide*. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/sound/programmer_guide/contents.html (besucht am 11.05.2018).
 - (o.J.[c]). *Using Files and Format Converters*. URL: <https://docs.oracle.com/javase/tutorial/sound/converters.html> (besucht am 11.05.2018).
- PInvoke.net (o.J.). *pinvoke.net: the interop wiki!* URL: <http://www.pinvoke.net/> (besucht am 20.07.2018).
- Project Lombok (o.J.). *Lombok features*. URL: <https://projectlombok.org/features/all> (besucht am 08.10.2018).
- Remeeting (Jan. 2018). *Documentation*. URL: <https://remeeting.com/api/docs/asr/v1/> (besucht am 21.05.2018).
- Richards, Mark (Feb. 2015). *Software Architecture Patterns. Understanding Common Architecture Patterns and When to Use Them*. 1. Aufl. O'Reilly Media.
- Sarkar, Abhijit (Jan. 2017). *Spring Cloud Netflix Eureka - The Hidden Manual*. URL: <https://blog.asarkar.org/technical/netflix-eureka/> (besucht am 09.05.2018).

- Selenium (Aug. 2018). *Selenium Documentation*. URL: <https://www.seleniumhq.org/docs/> (besucht am 02.06.2018).
- (o.J.). *Third Party Drivers, Bindings, and Plugins*. URL: <https://www.seleniumhq.org/download/> (besucht am 29.06.2018).
- Spring Cloud Config (o.J.[a]). *Spring Cloud Config*. URL: https://cloud.spring.io/spring-cloud-config/multi/multi_spring-cloud-config.html (besucht am 29.07.2018).
- (o.J.[b]). *Spring Cloud Config GitHub-Repository*. URL: <https://github.com/spring-cloud/spring-cloud-config/blob/master/docs/src/main/asciidoc/spring-cloud-config.adoc> (besucht am 29.07.2018).
- Spring Cloud Netflix (o.J.[a]). *Declarative REST Client: Feign*. URL: https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html (besucht am 09.05.2018).
- (o.J.[b]). *Service Discovery: Eureka Server*. URL: https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-eureka-server.html (besucht am 10.05.2018).
- (o.J.[c]). *Spring Cloud Netflix*. URL: <https://github.com/spring-cloud/spring-cloud-netflix/blob/master/docs/src/main/asciidoc/spring-cloud-netflix.adoc> (besucht am 05.05.2018).
- Thymeleaf (Juni 2018a). *Tutorial: Thymeleaf + Spring*. URL: <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html> (besucht am 02.07.2018).
- (Juni 2018b). *Tutorial: Using Thymeleaf*. URL: <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html> (besucht am 02.07.2018).
- Valin, Jean-Marc (Dez. 2007). *The Speex Codec Manual Version 1.2 Beta 3*. URL: <https://www.speex.org/docs/manual/speex-manual.pdf> (besucht am 08.10.2018).
- Webb, Phillip et al. (o.J.). *Spring Boot Reference Guide*. URL: <https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/> (besucht am 10.10.2018).

Abbildungsverzeichnis

2.1. Öffentliche Echo-Geräte können z.B. an den Wänden von Räumen oder Gängen platziert werden, um für alle Mitarbeiter verfügbar zu sein. (Bild: Amazon o.J.[b])	4
2.2. Die in der Mitte des Tischs befindliche Hardware ist für die Aufnahme von Audio- und Videodaten zuständig. Im Hintergrund ist zu sehen, wie ein textuelles Transkript des Meetings in Echtzeit erstellt wird. (Bild: Microsoft 2018a)	7
2.3. Googles Jamboard folgt dem aktuellen Trend der Bedienung über Toucheingaben. Das Gerät ist eine mobile Arbeitsstation für kollaborative Arbeit. (Bild: Google o.J.[b])	8
3.1. Die hier dargestellten Konzepte liegen SL zugrunde und sind auf bestimmte Art und Weise miteinander verknüpft.	9
3.2. SL orchestriert an einer Lokalität (z.B. in einem Meetingraum) vorhandene Akteure, um nutzbringende Assistenzfunktionen bereitzustellen. Dabei werden Daten und Signale von den Lokalitäten empfangen und gegebenenfalls mit Hilfe von Webservices weiterverarbeitet. Im Gegenzug empfangen die lokalen Akteure Instruktionen von SL.	11
3.3. Verschiedene Arten der Ausführung einer Tätigkeit.	16
3.3. Verschiedene Arten der Ausführung einer Tätigkeit.	17
3.4. SL stellt jeder Assistance Kontextinformationen bereit, welche diese bei ihrer Ausführung nutzen kann. Orange dargestellte Teile des Kontexts sind obligatorisch, wohingegen gelb dargestellte Teile optional sind.	20
3.5. SL kann an verschiedene Datenquellen angebunden werden, deren Informationen für den Zusammenbau eines Assistance-Kontexts verwendet werden. Die hier gezeigten Services und Komponenten, welche als Datenquellen verwendet werden, sind lediglich ein Beispiel für eine mögliche Konfiguration des Systems.	21
3.6. SL kennt vier verschiedene Trigger-Signale, die den zeitlichen Rahmen von Events markieren.	23
3.7. Ein Beispiel für die Ausführungsreihenfolge der drei Phasen einer Assistance.	25

3.8.	Die vollständige Definition des Ausführungsverhaltens der Assistance <i>minute taking</i>	26
3.9.	Die vollständige Definition des Ausführungsverhaltens der Assistance <i>website displaying</i>	29
3.10.	Die vollständige Definition des Ausführungsverhaltens der Assistance <i>file displaying</i>	30
3.11.	Die vollständige Definition des Ausführungsverhaltens der Assistance <i>agenda showing</i>	31
3.12.	Die vollständige Definition des Ausführungsverhaltens der Assistance <i>device preparation</i>	32
3.13.	Die vollständige Definition des Ausführungsverhaltens der Assistance <i>image preservation</i>	34
3.14.	Die Kopplung der Actions in der Phase <i>end</i> der Assistance <i>minute taking</i> . Die Inputs und Outputs der Actions sind dabei über Pfeile dargestellt und die Reihenfolge ihrer Ausführung über Zahlen.	35
3.15.	Ein Entwurf der Weboberfläche für den Event-Status einer Lokalität.	40
3.16.	Der Ablauf der fachlichen Ausführungslogik von SL. Der rechte Zweig des Diagramms stellt dabei eine asynchrone Ausführung dar, die unabhängig vom Hauptast abläuft.	42
3.17.	SL basiert auf einer serviceorientierten Architektur. Blaue Komponenten realisieren die fachlichen Prozesse von SL und gehören zu dessen Kernsys- tem. Die roten Komponenten sind ebenfalls dem Kernsystem zugeordnet, erfüllen aber rein technische Zwecke und sind deshalb auch an alle anderen Services angebunden (ihre Verbindungslien wurden daher der Übersicht- lichkeit halber weggelassen). Orangene Komponenten gehören zwar zu SL, sind jedoch nicht Teil des Kernsystems. Grüne Komponenten sind nicht Bestandteil von SL, aber an das System angebunden.	44
4.1.	Die Services von SL sind in Schichten aufgeteilt, welche miteinander kom- munizieren.	50
4.2.	Der Quellcode von SL besteht aus einer Hierarchie aus Maven-Modulen. Grüne Pfeile zeigen die Beziehung „ist ein Untermodul von“ an. Rote und gelbe Pfeile symbolisieren die Beziehung „erbt von“ (zur Vermeidung von mehrdeutigen Pfeilverläufen wurden zwei verschiedene Farben verwendet). .	51
4.3.	Die Mechanismen der Kommunikation zwischen den Diensten von SL sind abhängig davon, ob das System als Monolith oder als Microservices gestartet wurde.	55

4.4. Die Repository-Schicht des <i>event management service</i> ist in einer Klassen-	
hierarchie organisiert. Spezifika der Komponenten wie Instanz- und Klas-	
senvARIABLEn sowie Methoden wurden aus Gründen der Übersichtlichkeit	
wEGGELASSEN. Grau hinterlegte Klassen sind spezifisch für die Datenhaltung	
von Events.	57
4.5. Die Konfiguration von Events erfolgt beim Kalenderdienst von Google teil-	
weise über ECL. Der im Beschreibungsfeld enthaltene Konfigurationsblock	
in ECL legt die Arbeitsgruppe, die Agenda und die gewünschten AssISTANCES	
fest.	59
4.6. Die Info-, Triggerable- und Controllable-Komponenten aller implementierten	
AssISTANCES sind in einer Klassenhierarchie organisiert. Aus Gründen der	
Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces	
aufgeführt.	63
4.7. Die Ausführung einer Action erfolgt unterschiedlich, abhängig davon, ob	
der zu verwendende Aktor per Fernzugriff oder lediglich über eine lokale	
Schnittstelle ansteuerbar ist.	65
4.8. Die Info-, Callable- und Executable-Komponenten aller implementierten	
Actions sind in einer Klassenhierarchie organisiert. Aus Gründen der Über-	
sichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces auf-	
geführt.	66
4.9. Die Status-Oberfläche eines Events ist als Internetseite über SL verfügbar. .	71
4.10. Die Agenda-Übersicht eines Events ist als Internetseite über SL verfügbar. .	71
4.11. Die Windows-interne Bezeichnung jedes Anzeigegeräts kann über ein Hilfs-	
programm ermittelt werden.	73
5.1. Die Evaluierung von SL wurde mit diesem Versuchsaufbau durchgeführt.	
Blaue Komponenten stellen Rechner dar, rote Komponenten sind Teile von	
SL und grüne Komponenten stehen für AktoREn.	83
B.1. Die Klassenhierarchie der implementierten Adapter für Mikrofon-Aktoren.	
Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der überge-	
ordneten Interfaces aufgeführt.	98
B.2. Die Klassenhierarchie der implementierten Adapter für Speech-to-Text-	
Service-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Me-	
thoden der übergeordneten Interfaces aufgeführt.	99
B.3. Die Klassenhierarchie der implementierten Adapter für Projektablage-Aktoren.	
Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der überge-	
ordneten Interfaces aufgeführt.	100

B.4. Die Klassenhierarchie der implementierten Adapter für Webbrowser-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.	101
B.5. Die Klassenhierarchie der implementierten Adapter für Programm-Aktoren zum Öffnen von Dateien. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.	102
B.6. Die Klassenhierarchie der implementierten Adapter für Anzeigegerät-Aktoren. Aus Gründen der Übersichtlichkeit sind lediglich die Methoden der übergeordneten Interfaces aufgeführt.	103

Tabellenverzeichnis

5.1. Bei der Evaluierung der Nützlichkeit von SL wurden verschiedene Funktionalitäten mit jeweils einer Stimme pro Testperson bewertet. Die in der Tabelle aufgeführten Zahlen geben dabei an, wie viele Stimmen eine Funktionalität für einen bestimmten Grad an Nützlichkeit erhalten hat.	84
5.2. Bei der Evaluierung der Benutzbarkeit von SL wurde eine Auswahl von Methoden zur Konfiguration von Events durch die Testpersonen in eine Prioritätenreihenfolge gebracht. Die Prioritätsklassen reichen von 1 (stark bevorzugt) bis 4 (wenig bevorzugt). Die in der Tabelle aufgeführten Zahlen geben dabei an, wie oft einer Methode eine bestimmte Priorität zugewiesen wurde.	84
A.1. Das API des <i>GUI service</i>	91
A.2. Das API des <i>job service</i>	91
A.3. Das API des <i>trigger service</i>	92
A.4. Das API des <i>person management service</i>	93
A.5. Das API des <i>actuator management service</i>	93
A.6. Das API des <i>location management service</i>	94
A.7. Das API des <i>workgroup management service</i>	95
A.8. Das API des <i>event management service</i>	96
A.9. Das API des <i>assistance service</i>	97
A.10. Das API des <i>action service</i>	97
A.11. Das API des <i>delegate service</i>	97

Listingverzeichnis

4.1. Spring Boot erlaubt eine einfache Realisierung von REST-Endpunkten mit Java. Der hier dargestellte Endpunkt stammt aus dem Controller des <i>assistance service</i> (siehe Abb. 3.17) und dient zum Starten der Phase <i>begin</i> einer Assistance. Zur besseren Lesbarkeit wurden alle Konstanten des eigentlichen Quellcodes durch ihre entsprechenden String-Literale ersetzt.	49
4.2. Die netzwerkbasierte Kommunikation mit dem <i>action service</i> wird programmatisch über einen Feign-Client ermöglicht. Zur besseren Lesbarkeit wurden alle Konstanten des eigentlichen Quellcodes durch ihre entsprechenden String-Literale ersetzt.	54
4.3. ECL ist über eine formale Grammatik in der EBNF definiert. Der hier aufgeführte Ausschnitt aus dieser Grammatik zeigt lediglich ihre hierarchisch höchsten Sprachregeln.	60
4.4. Die hier dargestellte Methode dient zum gleichzeitigen Verschieben und Maximieren eines Programmfensters. Sie wurde mit einem von DllExport angebotenen Attribut versehen und wird während des Build-Prozesses in eine native DLL verpackt.	72
C.1. Im hier aufgeführten Konfigurationsblock in ECL sind die Arbeitsgruppe und Agenda festgelegt sowie alle implementierten Assurances aktiviert. Diese Konfiguration wurde auch für die Evaluierung von SL verwendet (siehe Abschnitt 5.2).	104

Abkürzungsverzeichnis

AfB	Alexa for Business
ANTLR	Another Tool for Language Recognition
API	Application Programming Interface
AR	Augmented Reality
AWS	Amazon Web Services
BOM	Bill of Materials
DLL	Dynamic Link Library
EBNF	Erweiterte Backus-Naur-Form
ECL	Event Configuration Language
FaF	Fire-and-Forget
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IFTTT	If This Then That
IoT	Internet of Things
JAR	Java Archive
JNA	Java Native Access
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz

LIRC Linux Infrared Remote Control

openHAB Open Home Automation Bus

PC Personal Computer

POM Project Object Model

REST Representational State Transfer

SL Smart Lab

SQL Structured Query Language

URL Uniform Resource Locator

USB Universal Serial Bus

VR Virtual Reality

XML Extensible Markup Language

(Diese Seite wurde absichtlich leer gelassen)