ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Realisierung eines mobilen Assistenten für intelligentes Zeitmanagement

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Simon Löw
simon.loew@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2016

Fassung 11. April 2016

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Durch ein immer flexibler werdendes Arbeitsumfeld mit Gleitzeit, Arbeit im Homeoffice und Eigenverantwortung der Mitarbeiter, steigt die Notwendigkeit für intelligentes Zeitmanagement seit Jahren kontinuierlich an. Während Termine mit fester Uhrzeit problemlos in einer Kalender-Anwendung erfasst werden können, müssen flexible Aufgaben ohne feste Uhrzeit in einer gesonderten To-do-Liste verwaltet werden. Durch die strikte Trennung der Anwendungen und die Tatsache, dass diese über keine eigene Intelligenz verfügen, ist der Anwender dazu gezwungen selbständig seinen Tag zu planen. Dies kann zu Stress, Fehlern und vergessenen Aufgaben führen.

Im Rahmen dieser Arbeit wird ein Konzept für einen mobilen Assistenten zum intelligenten Zeitmanagement entworfen. Dieser vereint Kalender und To-do-Liste in einer Anwendung und schlägt dem Anwender anhand des Kontextes passende Aufgaben vor. Im Folgenden wird zunächst analysiert, welche Kontextinformationen für die Anwendung relevant sind und wie diese erfasst werden können. Anschließend werden bisherige Zeitmanagementsysteme evaluiert und ein Anwendungskonzept für den Assistenten entwickelt. Dabei wird großer Wert auf ein einfaches Oberflächen- und Bedienkonzept gelegt. Schließlich wird eine modular erweiterbare Softwarearchitektur für die Anwendung entworfen und implementiert.

Inhaltsverzeichnis

1

Einleitung

Gutes Zeitmanagement gewinnt in unserer Gesellschaft zunehmend an Bedeutung. Durch ein immer flexibler werdendes Arbeitsumfeld mit Gleitzeit, Arbeit im Homeoffice und eigener Projektverantwortung durch die Mitarbeiter, steigt die Notwendigkeit für intelligentes Zeitmanagement auch für normale Arbeitnehmer an.

Das Mittel der Wahl zur Verwaltung des Tages ist dabei für die Meisten ein Kalender. In diesem lassen sich alle Termine mit einer festen Uhrzeit eintragen. Problematisch ist aber die Vielzahl an flexiblen Aufgaben, die sich nur schlecht in einem Kalender verwalten lassen. Viele Aufgaben verfügen über keine feste Uhrzeit zu der sie bearbeitet werden sollen, sondern nur über ein Fälligkeitsdatum zu dem sie erledigt sein müssen. Solche Aufgaben werden üblicherweise in einer gesonderten To-do-Liste verwaltet. Zwar gibt es schon lange ausgefeilte Techniken wie Get Things Done (siehe Kapitel 4.1.2) um solche flexiblen Aufgaben in To-do-Listen zu verwalten und den Tag zu planen, doch diese erfordern eine längere Einarbeitungszeit und hohe Disziplin bei der Umsetzung.

Auch existieren mobile Anwendungen, zur Verwaltung von Aufgaben und Terminen, doch diese sind im Wesentlichen digitale Versionen klassischer, handgeschriebener To-do-Listen und Kalender und verfügen über keine eigene Intelligenz. Das Problem wird dadurch erschwert, dass bisher eine strikte Trennung zwischen Aufgaben- und Terminverwaltung existiert. Somit ist der Anwender dazu gezwungen permanent zwischen mehreren Anwendungen zu wechseln:

Will der Anwender eine Aufgabe erledigen, so muss er zunächst den Kalender prüfen, um festzustellen wie viel Zeit ihm bis zum nächsten festen Termin bleibt. Anschließend wechselt er zur To-do-Liste und sucht eine Aufgabe, die er in der zur Verfügung stehenden Zeit bearbeiten kann. Dabei muss er aber den Kontext jeder Aufgabe und eine

Vielzahl von Nebenbedingungen beachten (siehe Kapitel 3). So sollte er die dringenden Aufgaben zu erst bearbeiten, muss aber auch sicherstellen, dass die zur Verfügung stehende Zeit ausreicht. Hierfür muss er neben der Dauer der Aufgabe auch Anfahrtswege zum Ort der Aufgabe und zum nächsten Termin beachten und beispielsweise die Routenplanung seines Smartphones benutzen. Aber auch weitere Nebenbedingungen wie Wetterlage oder Öffnungszeiten müssen in Betracht gezogen werden und zum Beispiel über eine Website abgerufen werden.

Die Folge ist, dass der Anwender zwischen verschiedenen Anwendungen wechseln und dabei eine Vielzahl an Informationen im Kopf behalten muss. Dieser Prozess ist aufwendig und fehleranfällig und kann zu Stress führen. Auch besteht die Gefahr, dass wichtige Aufgaben vergessen oder unangenehme Aufgaben aufgeschoben werden.

Es stellt sich also die Frage, wie dieser Prozess vereinfacht und Aufgaben- und Terminverwaltung in einer Anwendung kombiniert werden können. Da ein derartiges Anwendungskonzept bisher nicht existiert, ist zu klären, wie ein Oberflächen- und Bedienkonzept einer solchen Anwendung aussehen könnte. Auch gilt es zu analysieren, welche Kontextinformationen und Nebenbedingungen für die Anwendung relevant sind und wie diese erfasst werden können. Schließlich ist zu untersuchen, welches intelligente Verhalten der Anwendung den Anwender bei seiner Planung unterstützen könnte und in wie weit eine softwaretechnische Umsetzung möglich ist.

1.1 Zielsetzung

Ziel der Arbeit ist es einen mobilen Assistent zu entwerfen, der an Hand des Kontextes intelligente Aufgabenvorschläge präsentiert. Dabei sollen die verschiedenen Nebenbedingungen der Aufgaben beachtet werden und für den Anwender relevante Zusatzinformationen, wie zum Beispiel Anfahrtswege, direkt in der Anwendung angezeigt werden.

Im Rahmen dieser Arbeit soll zunächst ein Überblick über bisherige Zeitmanagementsysteme gegeben und der Begriff des Kontextes klar definiert werden. Anschließend soll untersucht werden, in wie weit sich die so eben beschriebene Anwendung umsetzen

lässt und welche Konzepte klassischer Zeitmanagementsysteme dafür relevant sind. Dabei soll die Anwendung so konzipiert werden, dass sich der beachtete Kontext später beliebig erweitern lässt. Schließlich soll das Konzept in einem Prototypen implementiert werden, wobei aus Zeitgründen nur der minimale noch sinnvolle Kontext beachtet werden soll.

1.2 Struktur der Arbeit

In Kapitel 2 wird zunächst ein Überblick über die zur Implementierung des Assistenten verwendeten Technologien gegeben.

Anschließend wird im Kapitel 3 der Kontextbegriff definiert und untersucht, welche Kontextinformationen für die Anwendung relevant sind. Nach einem Überblick über bisherige Zeitmanagementsysteme werden in Kapitel 4 daraus Designkriterien für die neue Anwendung abgeleitet und auf Basis dieser verschiedene Anwendungskonzepte evaluiert.

Kapitel 5 stellt schließlich das Gesamtkonzept der Anwendung vor. Anschließend werden in den folgenden Kapiteln die einzelnen Anwendungskomponenten im Detail betrachtet. In Kapitel 6 wird die Terminverwaltung kurz besprochen und anschließend die Aufgabenverwaltung in Kapitel 7 beschrieben. Schließlich werden die Implementierung der Kontexterfassung und der Planungsalgorithmus in Kapitel 8 eingehend betrachtet.

Abschließend wird in Kapitel 9 der derzeitige Entwicklungsstand der Anwendung zusammengefasst und ein Ausblick auf mögliche Erweiterungen des Anwendungskonzeptes gegeben.

2

Rahmenbedingung der Implementierung

Bevor der Entwurf der Anwendung im Detail betrachtet werden kann, gilt es zunächst die Rahmenbedingungen der Implementierung festzulegen.

Die praktische Umsetzung erfolgt mittels einer aufgeteilten Client- / Serveranwendung. Dies ermöglicht langfristig die Unterstützung einer breiten Auswahl an Plattformen (Web, iOS, Android, . . .), ohne die Programmlogik mehrfach implementieren zu müssen. Auch ist für den Anwender so eine Nutzung auf mehreren Geräten möglich.

Hinsichtlich der Kontexterfassung ist eine Aufteilung in Client und Server ebenfalls sinnvoll. Der Server kann so die Kontextinformationen verschiedener Clients bündeln und ermöglicht so, auch komplexe Zusammenhänge in der Aufgabenplanung zu erfassen. Sind beispielsweise mehrere Personen an einer Aufgabe beteiligt, so ist es sinnvoll den Aufenthaltsort aller Personen in die Planung einfließen zu lassen (siehe Kapitel 3.2.5).

2.1 Server

Betrachten wir zunächst die Serverseite. Hier wurde Wert darauf gelegt moderne Webtechnologien zu benutzen. Ziel ist es, eine einfache und schnelle Erweiterbarkeit der Anwendung zu garantieren und eine leichte Skalierung der Serveranwendung zu ermöglichen.

2.1.1 REST-Schnittstelle

Um die Kommunikation zwischen Client und Server so einfach wie möglich zu gestalten, wurde eine in weiten Teilen REST-konforme HTTP-Schnittstelle implementiert [1]. Dadurch müssen weder der Client noch der Server Zustandsinformationen speichern. Dies erleichtert die Implementierung des Clients und ermöglicht eine leichtere Skalierbarkeit der Serveranwendung. Auf die Schnittstellen der Termin- und Aufgabenverwaltung wird in den Kapiteln 6 und 7 genauer eingegangen.

Zur Authentifizierung der Clients werden JSON Webtokens (kurz: JWT, [2]) verwendet. Das Vorgehen dabei ist wie folgt:

1. Der Client erfragt vom Benutzer den Benutzernamen und das Passwort und sendet diese Informationen an den Server.
2. Der Server prüft die Korrektheit der Zugangsdaten.
3. Sind die Zugangsdaten korrekt, so werden die Benutzerinformationen in ein JSON-Objekt gepackt und mit einem geheimen Schlüssel signiert.
4. Dieses signierte Objekt wird Base64 codiert an den Client übertragen und dient diesem als Token für weitere Anfragen.
5. Für alle weiteren Anfragen sendet der Client das Token als HTTP-Header an den Server, der anhand der Signatur die Echtheit der Daten prüft.

Das Token kann ohne Sicherheitsbedenken auf dem Client gespeichert werden, da es kein Benutzerpasswort enthält. Der Server kann mit seinem privaten Schlüssel wiederum die Echtheit eines Tokens problemlos überprüfen und Manipulationen des Tokens zuverlässig erkennen. Da die komplette Authentifizierung ohne gespeicherte Zustände auskommt, ist dieses Verfahren für einen REST-Service optimal.

2.1.2 Server: Node.js und Heroku

Für die Implementierung der Serveranwendung wurde Node.js gewählt. Ausschlag gebend dafür war die einfache und schnelle Implementierung eines REST-Services

mittels Express [3], sowie die große Auswahl an Zusatzbibliotheken über npm[1]. Auch ist die Anbindung an REST-konforme APIs von Drittanbietern leicht möglich.

Die Anwendung selbst wird in der Cloud ausgeführt. Somit entfällt einerseits der Betrieb eines eigenen Testservers, andererseits lässt sich im Falle einer späteren Veröffentlichung die Anwendung leicht skalieren. Als Cloud-Anbieter wurde Heroku gewählt. Dieser stellt ein git-Repository zur Verfügung, in das die Node.js Anwendung geladen werden kann und direkt ausgeführt wird [4]. Ein aufwendiges Deployment entfällt somit.

2.1.3 MongoDB

Als Datenbank wurde MongoDB gewählt, da sich ihr dokumentenorientiertes Datenmodell [5] ideal zur Verwaltung von Terminen und Aufgaben eignet. Termine und Aufgaben sind jeweils in sich abgeschlossene Entitäten mit wenig Querverweisen. Sie besitzen jedoch unter Umständen eine komplexe, verschachtelte Struktur. Beispiel hierfür ist eine Termingruppe mit mehreren sich wiederholenden Terminen. In einer relationalen Datenbank müsste ein derartiges Schema normalisiert werden und die Termine in eine extra Tabelle ausgelagert werden (1 zu N Beziehung). Bei jeder Abfrage müsste dann zunächst ein Join ausgeführt werden. Bei komplexen Anfragen, bei denen die Datensätze zum Teil schon vor dem Join gefiltert werden können, wäre ein solches normalisiertes Schema von Vorteil. Im Falle von Terminen mit komplexen Wiederholungsschemata, ist dies aber nicht ohne weiteres möglich[2].

In der MongoDB kann ein Termin jedoch als einzelnes verschachteltes Dokument gespeichert werden. Die Abfrage und das Bearbeiten gestaltet sich dabei einfach und es sind keine Joins nötig. Daher entfällt auch die Konsistenzprüfung zwischen verschiedenen Tabellen und die Datenbank lässt sich leicht horizontal skalieren.

Hinzu kommt, dass MongoDB direkt mit JSON- (bzw. BSON-) Dokumenten arbeitet und so einfach mit Node.js interagieren kann. Um ein einheitliches Schema für die

[1] npm ist eine Paketverwaltung für Node.js Bibliotheken. Über ein Online-Repository können eine Vielzahl an Zusatzbibliotheken geladen werden.

[2] Teil der Programmlogik könnten zwar in SQL implementiert werden. Dies würde aber die Lesbarkeit des Programmcodes und das Debugging erschwert.

Dokumente zu garantieren, wurde Mongoose [6] für das Object Data Mapping (ODM) verwendet. Die in den Kapiteln 6 und 7 vorgestellten Datenbank-Schemata sind dabei in der Mongoose Notation angegeben [7].

Zu beachten sind die unterschiedlichen Formate für Uhrzeiten und Intervalle:

- **Client:** Zum Datenaustausch mit dem Client werden Uhrzeiten und Intervalle als ISO8601 Strings [8] übertragen. Dies stellt einen reibungslosen Datenaustausch mit beliebigen Clients sicher und ermöglicht das leichte Debuggen der Schnittstelle, da die Strings auch für Menschen gut lesbar sind. Der Android Client kann diese Strings dann mittels JodaTime [9] parsen und intern weiter verarbeiten.
- **Server:** Der Server verarbeitet Uhrzeiten und Intervalle mittels der moment.js Bibliothek [10] und verwendet dessen Repräsentation als JavaScript-Objekt.
- **Datenbank:** In der MongoDB werden Uhrzeiten mit dem Datentype `Date` abgespeichert. Für Intervalle gibt es hingegen keinen speziellen Datentyp weshalb diese als ISO8601 String abgespeichert werden.

2.2 Client

Der mobile Assistent selbst wurde für die Android-Plattform implementiert. Grund hierfür ist die weite Verbreitung der Plattform mit einem Marktanteil von zirka 80% [11]. Das Oberflächendesign orientiert sich dabei an den Material Design Richtlinien [12].

Es wurde eine native Anwendung implementiert, um direkt auf die Betriebssystem APIs zugreifen und somit alle Sensoren des Smartphones effizient nutzen zu können [13, 14]. Die Architektur folgt dem Model-View-Controller-Patterns (MVC) [15, 16]. Android gibt hierbei kein festes Schema vor und unterstützt MVC nicht direkt. Mit folgender Aufteilung der Komponenten ist aber eine relativ saubere MVC Implementierung möglich:

- **View:** Die mittels XML beschriebene Benutzeroberfläche repräsentiert die View-Komponente. Jedes Element der Oberfläche kann im Programmcode mittels einer ID referenziert werden. Dies garantiert eine lose Koppelung zwischen View und Controller.

- **Controller:** Der Controller ist entweder eine Activity oder ein Fragment. Der Controller verarbeitet Ereignisse der Benutzeroberfläche und stellt die Schnittstelle zwischen View und Model her. Die eigentliche Programmlogik liegt hingegen im Model und ist nicht Teil des Controllers.
- **Model:** Für die Implementierung wurde auf einen Offline-Modus verzichtet. Da der Kontext nur bei bestehender Internetverbindung vollständig ermittelt werden kann, wäre ein Offline-Modus stark eingeschränkt und kaum benutzbar (siehe Kapitel 3.3.3). Das Model besteht daher im wesentlichen aus Klassen, die für Kommunikation mit dem Server zuständig sind. Dabei wird die Volley Bibliothek [17] verwendet. Die JSON-Objekte werden intern auf Java-Klassen gemapped, dabei kommt GSON [18] zum Einsatz. Dadurch haben die Daten sowohl auf Server- als auch auf Client-Seite die selbe Struktur.

3

Kontext

Um mittels eines kontextsensitiven Algorithmus sinnvolle Aufgabenvorschläge zu ermitteln, muss der Kontext der Anwendung klar definiert sein. Im Folgenden wird untersucht, wie sich der Kontextbegriff im Allgemeinen definieren lässt, und anschließend der Kontext für die Anwendung definiert.

3.1 Definition

Obwohl Kontextsensitivität mittlerweile integraler Bestandteil mobiler Anwendungen ist, gibt es kaum formale Definitionen des Kontextbegriffes. Die meisten Definitionen bestehen aus Aufzählungen konkreter Beispiele und sind für neue Anwendungen somit schwer zu adaptieren [19]. Für diese Arbeit wurde deshalb die sehr abstrakte Definition des Kontextbegriffes von Dey und Abowd übernommen [19]:

"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

Im folgenden wird ein möglicher Kontext für die Anwendung angegeben[1].

[1] Durch die abstrakte Definition des Kontextbegriffs ist eine allumfassende Definition des Anwendungskontextes nur schwer möglich.

3.2 Relevanter Kontext

Die für die Anwendung relevanten Entitäten sind die Zeit, Termine, Aufgaben, der Anwender selbst, Ressourcen und Personen. Im Folgenden werden die Kontextinformationen jeder Entität im Detail betrachtet.

3.2.1 Zeit

Die Zeit selbst als Entität zu betrachten mag zunächst ungewöhnlich erscheinen, ist im Kontext dieser Anwendung aber sinnvoll. Mögliche Kontextinformationen der Zeit sind:

- **UTC-Zeit:** Die aktuelle Uhrzeit bietet die Basis für alle weiteren Kontextinformationen der Zeit, sie muss aber mittels der Zeitzone korrekt interpretiert werden. Für diese Anwendung wird die UTC-Zeit verwendet, da sie in den meisten Anwendungen Standard ist [20].
- **Zeitzone:** Die Zeitzone ermöglicht es die Uhrzeit richtig zu interpretieren und die weiteren Kontextinformationen abzuleiten. Die Zeitzone selbst hängt wiederum von den anderen betrachteten Entitäten ab. Betrachtet man beispielsweise eine Person, so leitet sich die Zeitzone aus dem Aufenthaltsort dieser Person ab.
- **Tageszeit:** Um Aufgaben zu einer bestimmten Tageszeit (morgens, mittags, . . .) vorzuschlagen, muss diese der Anwendung bekannt sein. Die Tageszeit lässt sich aus der UTC-Zeit und der Zeitzone ableiten.
- **Jahreszeit:** Die Jahreszeit wird in der vorliegenden Anwendung nicht verwendet, dennoch sind Szenarien denkbar, in denen die Jahreszeit Einfluss auf den Planungsalgorithmus hat.

3.2.2 Termine

Termine sind Fixpunkte innerhalb des Tagesplanes. Sie ermöglichen es, Rückschlüsse auf den Kontext des Anwenders zu ziehen, und sind deshalb relevant für die Anwendung. Zu den Kontextinformationen zählen:

- **Zeit:** Start- und Endzeit des Termins
- **Wiederholungen:** Je nach Implementierung kann man ein Wiederholungsschema als Teil der **Zeit** sehen oder als gesonderte Kontextinformation betrachten.
- **Ort:** Unter Umständen kann der Ort gleichzeitig auch Ressource sein (siehe Abschnitt 3.2.5).
- **Art der Tätigkeit:** Die Art der Tätigkeit kann auf verschiedene Weisen betrachtet werden: So ist beispielsweise eine grobe Einteilung in Kategorien wie Arbeit, Freizeit, Familie, etc. möglich. Die Tätigkeit kann aber auch nach dem Einfluss auf den Kontext des Anwenders klassifiziert werden. In diesem Fall wären Kategorien wie „anstrengend“, „entspannend“, etc. denkbar.
- **Personen** / **Ressourcen:** Beteiligte Personen und benötigte Ressourcen sind wichtiger Teil des Kontextes (siehe Abschnitt 3.2.5).

3.2.3 Aufgaben

Beim Kontext der Aufgaben muss die obige Definition des Kontextbegriffes weiter ausgelegt werden und zwischen zwei Arten von Kontext unterschieden werden: Bei erledigten Aufgaben gilt es, den tatsächlichen Kontext während der Ausführung zu erfassen. Bei anstehenden Aufgaben ist der Kontext hingegen nicht fest definiert. Vielmehr lässt sich für jede Aufgabe der präferierte Ausführungskontext ermitteln. Zu den Kontextinformationen zählen in beiden Fällen:

- **Dauer:** Hier muss zwischen erledigter und offener Aufgabe unterschieden werden: Bei der erledigten Aufgabe ist die tatsächlich benötigte Zeit bekannt, bei der offenen Aufgabe handelt es sich lediglich um die geschätzte Zeit die benötigt wird, um sie zu erledigen.
- **Dringlichkeit/Deadline:** Sehr dringende Aufgaben sollten bei den Vorschlägen bevorzugt werden.
- **Ort:** Unter Umständen kann der Ort gleichzeitig auch Ressource sein (siehe Abschnitt 3.2.5).

- **Art der Tätigkeit:** Analog zu Terminen (siehe Abschnitt 3.2.2), lässt sich die Art der Tätigkeit auf verschiedene Weisen betrachten: So ist beispielsweise eine grobe Einteilung in Kategorien wie Arbeit, Freizeit, Familie, etc. möglich. Die Tätigkeit kann aber auch nach dem Einfluss auf den Kontext des Anwenders klassifiziert werden. In diesem Fall wären Kategorien wie „anstrengend“, „entspannend“, etc. denkbar.
- **Personen** / **Ressourcen:** Beteiligte Personen und benötigte Ressourcen sind wichtiger Teil des Kontextes (siehe Abschnitt 3.2.5).
- **Weitere Nebenbedingungen:** Für Aufgaben sind noch viele weitere Nebenbedingungen denkbar, die hier nicht alle aufgezählt werden können. Beispiele sind Ausführung zu einer bestimmten Tageszeit (morgens, mittags, ...) oder nur an bestimmten Wochentagen.

3.2.4 Anwender

Der Anwender selbst ist eine der wichtigsten Entitäten für die Anwendung. Sein Verhalten spielt eine maßgebliche Rolle für die Auswahl der Vorschläge. Der Kontext des Anwenders bildet zusammen mit den Terminen die Fixpunkte um gute Aufgabenvorschläge zu präsentieren. Der relevante Kontext für den Anwender ist dabei:

- **Zeit:** Sowohl aktuelle Uhrzeit (siehe Abschnitt 3.2.1) als auch die zur Verfügung stehende freie Zeit sind relevant für die Planung.
- **Ort:** Zusammen mit der Uhrzeit ermöglicht der Ort, die Anfahrzeiten zum nächsten Termin und allen Aufgaben zu ermitteln. Dadurch kann für jede Aufgabe geprüft werden, ob sie zeitlich möglich ist.
- **Stimmung**/ **Gesundheitszustand:** Mögliche Zustände sind: wach, fröhlich, müde, gestresst, etc. Zusammen mit der Art der Aufgabe kann ermittelt werden, ob sich eine Aufgabe im Moment anbietet oder nicht. So macht es beispielsweise wenig Sinn einem müden und gestressten Anwender die Steuererklärung als nächste Aufgabe vorzuschlagen, sofern sie nicht sehr dringend zu erledigen ist.

- **Gewohnheiten** / **Arbeitszeiten:** „Geht immer am frühen Abend zum Sport“, „Arbeitet jeden Tag von 8 Uhr bis 17 Uhr“, etc.

3.2.5 Ressourcen / Personen

Für einige Aufgaben können weitere Ressourcen und Personen notwendig sein, um sie erfolgreich zu erledigen. Beispiele dafür sind Sportkleidung, die für einen Workout benötigt wird, oder Übungspartner, mit denen man ein Übungsblatt bearbeitet. Es kann daher sinnvoll sein, für Aufgabenvorschläge die Verfügbarkeit dieser Ressourcen zu prüfen.

Ressourcen können dabei auch direkt mit dem Ort der Aufgabe zusammenhängen. Beispiele dafür sind ein Tennisplatz der gebucht werden muss oder ein Geschäft mit festen Öffnungszeiten.

Da verschiedenste Objekte als Ressource in Frage kommen und die Abfrage der Verfügbarkeit sich stark unterscheidet, wurde in dieser Arbeit auf die Implementierung einer Ressourcenverwaltung verzichtet. Ein einfacher erster Ansatz wäre, Termine als Fixpunkte zu verwenden:

Da bei Terminen davon ausgegangen werden kann, dass die Verfügbarkeit von Ressourcen, sowie die Anwesenheit relevanter Personen im Vorfeld abgeklärt wurde, können diese Ressourceninformationen für die weitere Planung benutzt werden. So ist es zum Beispiel denkbar, nach einer Vorlesung die der Anwender selbst und sein Übungspartner besucht haben, die Bearbeitung des Übungsblattes vorzuschlagen.

3.3 Erfassung des Kontextes

Beim Kontext kann zwischen primärem und sekundärem Kontext unterschieden werden [19]. Primärer Kontext muss dabei direkt erfasst werden, sekundärer Kontext lässt sich aus dem primären Kontext ableiten.

3.3.1 Kontext des Anwenders

Zum primären Kontext des Anwenders zählen die Zeit und der Ort, welche direkt mit dem Smartphone ermittelt werden können. Die Gewohnheiten und Arbeitszeiten des Anwenders könnten entweder direkt eingegeben werden oder durch Analyse der Vergangenheit automatisch gelernt werden. Letzteres ist die bessere Wahl, um dem Anwender die Konfiguration der Anwendung zu ersparen. Da hierbei der Kontext der Termine und erledigten Aufgaben analysiert wird, handelt es sich um sekundären Kontext. Das Erfassen der Stimmung und des Gesundheitszustandes des Anwenders ist hingegen schwierig. Hier sind beispielsweise Fitnesstracking-Armbänder oder Bluetooth Pulsmesser [21] denkbar, die aus dem Puls und Blutdruck Rückschlüsse ziehen (sekundärer Kontext), aber auch aus den bereits erledigten Aufgaben und Terminen des aktuellen Tages könnten Rückschlüsse gezogen werden.

3.3.2 Kontext der Termine und Aufgaben

Im Gegensatz zum Kontext des Anwenders, kann der Kontext eines Termins oder einer Aufgabe jedoch nicht über Sensoren ermittelt werden und muss vom Anwender angegeben werden. Um die Eingabe möglichst einfach zu gestalten, wurde der einzugebende primäre Kontext auf das Minimum begrenzt. Die Schnittmenge zwischen Termin- und Aufgabenkontext ist dabei wie folgt[2]:

- **Name:** Der Name ist zwar nicht direkt Kontext, kann aber dazu verwendet werden, mittels verschiedener Machine Learning Techniken weitere Kontextinformation abzuleiten (z.B. Art der Tätigkeit).
- **Kategorie:** Für die Arbeit wurden die Kategorien auf „Arbeit", „Schule", „Familie", „Freunde/Hobbys" begrenzt. Über die Kategorie können zusammen mit dem Name weitere Kontextinformationen abgeleitet werden (z.B. bevorzugte Uhrzeit, Art der Tätigkeit, . . .).
- **Ort:** Der Ort wird für die Ermittlung des Anfahrtsweges benötigt.

[2] Die Termin- und Aufgaben spezifischen Kontextinformationen werden jeweils in einem eigenen Kapitel behandelt (siehe Kapitel 6 und 7)

Die Idee hinter diesem minimalen Kontext ist, dass es sich dabei um Informationen handelt, die von den meisten To-do- und Kalenderanwendungen bereits jetzt erfasst werden. Dies hat den Vorteil, dass der Anwender kein neues Konzept lernen muss und die Eingabe von Terminen und Aufgaben wie gewohnt abläuft. Auch wird dadurch das Einbinden anderer Kalender erleichtert, da deren Einträge die selben Informationen enthalten (unter Umstände aber in einem anderen Format). Aus diesen drei Informationen ist es zudem einem Menschen leicht möglich weitere Kontextinformationen abzuleiten, weshalb davon ausgegangen werden kann, dass weitere Kontextinformationen auch von einem hinreichend komplexen Machine Learning System abgeleitet werden können.

Termine und Aufgaben werden in den Kapiteln 6 und 7 ausführlich besprochen.

3.3.3 Ableitung des sekundärer Kontextes

Um aus den primären Kontextinformationen Weitere abzuleiten, sind zwei Ansätze denkbar. So könnten zunächst aus den vorhandenen Informationen direkt Schlüsse gezogen werden. Dabei kann es sich um sehr einfache Algorithmen handeln. Beispielsweise kann einer Uhrzeit anhand eines festen Schemas eine Tageszeit (morgens, mittags, etc.) zugeordnet werden (zum Beispiel: $6 \leq Zeit(Stunde) \leq 10 \Rightarrow morgens$). Aber auch komplexe Folgerungen mittels Machine Learning Techniken sind denkbar. Solche Algorithmen ließen sich zwar prinzipiell auf dem Client implementieren, dies wäre aber mit erheblichem Aufwand verbunden, zumal für jede Plattform eine eigene Implementierung erfolgen müsste.

Andererseits können mit Hilfe der vorhandenen Kontextinformationen auch Weitere abgefragt werden. So können beispielsweise mit den GPS-Koordinaten wichtige Ortsinformationen wie Wetter, Zeitzone, Öffnungszeiten, etc. online abgerufen werden. Um diese wichtigen Kontextinformationen ermitteln zu können, ist eine permanente Internetverbindung nötig und somit ein Offline-Modus des mobilen Assistenten ausgeschlossen.

Wie die Kontexterfassung und die Ableitung des sekundären Kontextes implementiert wurde, wird im Kapitel 8 im Detail besprochen.

4

Evaluation verschiedener Ansätze

Für den Entwurf einer Anwendung zum intelligenten Zeitmanagement sind ganz verschiedene Ansätze denkbar. Im Verlauf der Planung und Implementierung der Anwendung wurden sowohl bisherige Zeitmanagementsysteme als auch psychologische Grundlagen betrachtet und, an Hand dieser, Kriterien für die Evaluation verschiedener Anwendungskonzepte formuliert. Anschließend wurden neue Ansätze entworfen und evaluiert.

Im Folgenden wird ein kurzer Überblick über bisherige Zeitmanagementsysteme und die betrachteten psychologischen Studien gegeben. Anschließend werden die entworfenen Anwendungskonzepte in chronologischer Reihenfolge vorgestellt und schließlich in Kapitel 5 der gewählte Ansatz ausführlich besprochen.

4.1 Bisherige Zeitmanagementsysteme und psychologischer Hintergrund

In der Vergangenheit wurden eine Vielzahl von Zeitmanagementsystemen entwickelt. Zu den bekanntesten Vertretern zählen sicherlich Get Things Done (kurz: GTD, [22]), Zen To Done (kurz: ZTD, [23]) und die Pomodoro Technik [24]. Im folgenden werden die Pomodoro Technik und GTD genauer betrachtet, da sie das Anwendungskonzept maßgeblich beeinflusst haben. Auf eine genauere Betrachtung von ZTD wurde verzichtet, da es sich im Wesentlichen um eine vereinfachte Version von GTD handelt und die Konzepte sich nicht sinnvoll als mobile Anwendung umsetzen lassen.

4.1.1 Pomodoro Technik

Die Pomodoro Technik ist kein ausgefeiltes System zur Planung von Aufgaben, sondern ein Technik um die Bearbeitung einer Aufgabe effizienter zu gestalten. Nach den Erkenntnissen der Psychologie und Hirnforschung stellt unser Arbeitsgedächtnis einen Engpass bezüglich gleichzeitiger Gedächtnisinhalte dar und kann schlecht mit Störungen umgehen [24]. Deshalb minimiert die Pomodoro die Anzahl der Unterbrechungen und Aufgabenwechsel. Ziel ist es, beim Bearbeiten einer Aufgabe in den Flow zu geraten. Flow ist dabei wie folgt definiert [25]:

„Klare Ziele, Konzentration und fokussierte Aufmerksamkeit, eine Art von Selbstvergessenheit, Verlust des Zeitgefühls, direktes und sofortiges Feedback, Ausgewogenheit zwischen den eigenen Fähigkeiten und der gestellten Herausforderung, ein Gefühl von Selbstkontrolle, das Gefühl, dass die Arbeit selbst eine Belohnung darstellt, die Verbindung von Handeln und Bewusstsein."

Um diesen Zustand zu erreichen, sollen Unterbrechungen und der regelmäßige Blick auf die Uhr vermieden werden. Das Vorgehen dabei ist simpel. Man wählt eine Aufgabe aus, stellt einen Wecker auf 25 Minuten (ein Pomodoro[1]) und bearbeitet die Aufgabe bis der Wecker läutet. Dabei soll man sich ganz auf die Aufgabe konzentrieren und auf jede Unterbrechung verzichten. Dies fördert die Konzentration auf die aktuelle Aufgabe. Gleichzeitig verhindert der Wecker, dass man sich ganz in der Bearbeitung der Aufgabe verliert und die Zeit komplett vergisst. Sobald der Wecker läutet, sollte man eine kurze Pause machen und wieder den Überblick über das große Ganze gewinnen. Dann bearbeitet man entweder die gleich Aufgabe für weiter 25 Minuten oder wählt eine Neue.

Wie oben bereits erwähnt, gibt die Pomodoro-Technik keinen Rahmen für die Verwaltung und Auswahl der Aufgaben vor. Zwar wird in der Literatur zum Teil auch darauf eingegangen, wie man seine Aufgabenliste strukturiert [24], die Pomodoro Technik ist aber dennoch kein System zur Strukturierung und Planung des Tages. Innerhalb eines Zeitmanagementsystems, das dem Tag Struktur gibt, kann die Pomodoro Technik aber

[1] Der Name leitet sich vom italienischen Wort für Tomate ab. Der Erfinder der Pomodoro Technik verwendete ursprünglich einen Küchenwecker in Tomatenform.

sehr nützlich sein: Das Zeitmanagementsystem gibt dabei vor was gemacht werden soll, die Pomodoro-Technik wie.

Auch kann ein Pomodoro als Einheit für die Planung sinnvoll sein. Da man mit der Zeit ein Gefühl dafür entwickelt, was man in der Dauer eines Pomodoro schafft, kann diese Schätzung akkurater sein, als Zeitangaben in Minuten und Stunden.

4.1.2 Get Things Done (GTD)

Get Things Done ist hingegen ein komplettes Zeitmanagementsystem, mit dem sich alle Aufgaben erfassen und strukturieren lassen. Dabei wird nach folgendem Schema vorgegangen [22]:

1. **Sammeln:** Zusammentragen aller Aufgaben aus den verschiedenen Eingangsquellen (Postfach, Email, etc.)
2. **Durcharbeiten:** Aufgaben genau definieren. Ist überhaupt eine Handlung nötig? Was sind die konkret notwendigen Handlungsschritte?
3. **Organisieren:** Aufgaben in Kategorien einteilen
4. **Durchsicht/Kontrolle:** Überblick über alle Aufgaben in allen Listen/Kategorien gewinnen
5. **Erledigen:** Tatsächliche Ausführung einer Aufgabe. Die Aufgabe kann dabei zum Beispiel an Hand des Kontextes, der verfügbaren Zeit, dem eigenen Energielevel und der Priorität gewählt werden.

Durch die feste Struktur, die vielen notwendigen Gewohnheitsänderungen und die daraus resultierende langen Einarbeitungszeit [23], ist GTD in seinem vollen Umfang nicht gut für eine mobile Anwendung geeignet. Dennoch bietet GTD viele interessante Ansätze.

Zentraler Gedanke hinter dem Schritt **Sammeln** ist, dass alle Aufgaben im System erfasst werden, um so den Kopf für die eigentlichen Tätigkeiten frei zu haben. GTD geht dabei soweit, dass alle noch so kleinen Aufgaben und Ideen erfasst werden. Hierfür wird eine gesonderte Liste mit dem Titel „Irgendwann/ Vielleicht“ geführt, die alle unwichtigen nicht genau spezifizierten Aufgaben erfasst. Für die vorliegende Anwendung

wurde aber auf die Kritik von ZTD eingegangen. Würde man tatsächlich alles noch so unwichtige und ungenau definierte in das System einspeisen, wird man irgendwann von der Fülle der sinnlosen Aufgaben erschlagen. Deshalb wurde für die vorliegende Anwendung das Anlegen neuer Aufgaben möglichst einfach gestaltet. Gleichzeitig sollen die Eingabemasken aber nur das Anlegen konkreter Aufgaben ermöglichen. Dadurch wird der Anwender dazu angeregt nur Aufgaben einzutragen, die wichtig sind und er tatsächlich erledigen möchte. Die Schritte **Durcharbeiten** und **Durchsicht/Kontrolle** sind deshalb nur bedingt relevant und die Anwendung kümmert sich selbständig um die Verwaltung der Aufgaben.

Das **Organisieren** der Aufgaben ist hingegen ein sehr interessantes Konzept. Hierbei rät GTD zum führen von Kontextlisten. Aufgaben werden dabei je nach Kontext einer bestimmten Liste hinzugefügt (Arbeit, Freizeit, Telefonanrufe, . . .). Dadurch kann der Anwender zu jedem Zeitpunkt die passende Kontextliste betrachten und eine Aufgabe auswählen. Eine mobile Anwendung könnte dieses Konzept optimal unterstützen, da Aufgaben nicht starr einer einzigen Liste zugeteilt werden müssen, sondern die Eignung für einen bestimmten Zeitpunkt dynamisch aus allen Kontextinformationen ermittelt werden kann.

Schließlich ist auch das **Erledigen** der Aufgaben für die Anwendung relevant. Hier setzt GTD auf einen reaktiven Ansatz der sich gut in einer mobilen Anwendung umsetzen lässt. GTD sieht von einer strikten Planung der Aufgaben ab, da sich die meisten Tage nicht gut vorausplanen lassen. Ganz im Gegenteil: Plant der Anwender für seinen Tag viele Aufgaben ein, kann es passieren, dass er beim Erledigen aller nicht dringenden Aufgaben, die einzige Dringende des Tages übersieht [22]. In GTD verwalten man nur feste Termine in einem Kalender, Entscheidungen für eine Aufgabe werden hingegen immer spontan im Hinblick auf den aktuellen Kontext getroffen.

4.1.3 Zen To Done (ZTD)

Zen To Done ist im wesentlichen eine Vereinfachung von Get Things Done. Es handelt sich um kein starres System und der Anwender muss nicht mehrere Gewohnheiten gleichzeitig ändern, was zu einer deutlich niedrigeren Einstiegshürde führt [23]. Auf

die Kritik an GTD wurde im letzten Abschnitt bereits kurz eingegangen, weshalb im Folgenden nur noch zwei neue Konzepte von ZTD besprochen werden.

ZTD regt zum einen dazu an, neue Routinen zu entwickeln. Ziel ist es dabei, kleine tägliche (oder wöchentliche) Aufgaben zusammenzufassen und immer zu einem festen Zeitpunkt zu erledigen. So könnte man zum Beispiel jeden Morgen um 8 Uhr alle E-Mails prüfen und Abends um 18 Uhr ein zweites Mal. Dadurch wird man nicht ständig durch neue E-Mails abgelenkt und erlernt gleichzeitig eine Gewohnheit, über die man langfristig nicht mehr nachdenken muss. Da sinnvolle Gewohnheiten sehr subjektiv sind, kann eine Anwendung dies nur schwer unterstützen und ausgefeilte Konzepte hierfür gehen über den Rahmen der Arbeit hinaus. Gerade wenn es um Routinen wie die oben erwähnte E-Mail-Routine geht, so ist ein Eintrag als fester Termin jedoch eine einfache Lösung.

Ein weiterer Ansatz von ZTD ist es, den Fokus auf langfristige Ziele zu richten. So wird verhindert, dass wichtige, große Aufgaben beim Bearbeiten der vielen kleinen Aufgaben untergehen. Um langfristige Ziele mit vielen Teilaufgaben zu unterstützen ist hingegen eine komplexe Anwendung nötig. Da ein komplexes Projektmanagementsystem nicht Ziel dieser Arbeit ist wurde auf eine genauere Betrachtung dieses Aspektes verzichtet.

4.1.4 Entscheidungen und Selbstkontrolle

Beim Entwurf einer Zeitmanagementanwendung, stellt sich schnell die Frage, wie viele Entscheidungen dem Nutzer überlassen werden sollen und was die Anwendung übernehmen soll. Hier lohnt sich ein Blick in die Psychologie:

Es zeigt sich, dass eine große Auswahl an Möglichkeiten eher kontraproduktiv ist. Eine Person, die aus einer großen Auswahl an Produkten wählen kann, neigt eher dazu nichts zu kaufen. Trifft diese Person hingegen eine Wahl, so ist die Zufriedenheit mit der Entscheidung deutlich geringer, je größer die Auswahl ist [26]. Überträgt man diese Erkenntnisse auf das Thema dieser Arbeit, so birgt eine zu große Entscheidungsvielfalt die Gefahr, dass der Anwender sich für keine Aufgabe entscheidet und die Arbeit aufschiebt. Trifft er hingegen eine Wahl, so ist es wahrscheinlich, dass er sie hinterher

bereut. Es ist denkbar, dass dies langfristig zu einer negativen Assoziation mit der Anwendung führt. Eine kleine Auswahl erhöht hingegen die Wahrscheinlichkeit das eine Aufgabe gewählt wird und führt gleichzeitig zu einer größeren Zufriedenheit mit der Entscheidung.

Die Anzahl der Entscheidungen drastisch zu begrenzen ist auch unter anderen Gesichtspunkten sinnvoll. In Studien zeigte sich, dass Entscheidungen viel Energie kosten und begrenzte mentale Ressourcen aufbrauchen [27]. Trifft man viele Entscheidungen sinkt die Qualität folgender Entscheidungen drastisch. Auch die Selbstkontrolle ist davon beeinträchtigt: Es zeigt sich, dass Probanden, die eine große Zahl von Entscheidungen treffen mussten, im darauf folgenden Versuch deutlich geringere Selbstkontrolle hatten [27]. Dies hat natürlich gravierende Konsequenzen für jedes Zeitmanagementsystem: Muss der Anwender zu viele Entscheidungen treffen, leidet nicht nur die Qualität darauf folgender Entscheidungen zur Wahl der nächsten Aufgabe, sondern auch die Ausführung der nächsten Handlung, da der Anwender nicht mehr über genug Geduld oder Konzentration verfügt.

Es scheint also sinnvoll die Zahl der Entscheidungen zu begrenzen. Aber auch hier sind psychologische Effekte zu beachten. Weitere Versuche zeigten, dass vorgegebene Entscheidungen, ebenfalls Energie des Probanden kosteten und seine Selbstkontrolle schmälern [27]. Nicht nur Entscheidungen zu treffen sondern auch Entscheidungen anderer zu akzeptieren kostet Ressourcen. Deshalb ist es wichtig die Balance zu finden. Die ideale Anwendung lässt dem Anwender eine Wahl und zwingt ihm keine Entscheidungen auf, minimiert aber die Zahl und die Komplexität der Entscheidungen.

4.2 Zielsetzung

Alle bisher vorgestellten Zeitmanagementsysteme wurden vor der breiten Verfügbarkeit von Smartphones entworfen und verwenden deshalb als Hilfsmittel vor allem Stift und Papier. Würde man diese Konzepte direkt auf eine mobile Anwendung übertragen, wären die Vorteile im Vergleich zu einem analogen Ansatz gering. Aber aus den Ideen hinter diesen System lassen sich dennoch sinnvolle Kriterien für die neue Anwendung ableiten:

1. **Fokus/Flow:** Es sollte immer nur eine Aufgabe gleichzeitig bearbeitet werden. Die Anwendung soll dieses Verhalten fördern, in dem jede Ablenkung von der aktuellen Aufgabe vermieden wird. Auch die Integration eines Pomodoro-Timers ist denkbar.
2. **Sammeln:** Die Anwendung soll das Speichern neuer Aufgaben möglichst einfach gestalten. Aufgaben, die dem Anwender in den Sinn kommen, können so sofort abgespeichert werden und müssen nicht im Kopf behalten werden. Es sollen aber nur konkret definierte Aufgaben in der Anwendung erfasst werden.
3. **Kontextsensitivität/reaktives Verhalten:** Wie in Kapitel 1 bereits erwähnt, soll die Anwendung Aufgaben anhand des Kontextes vorschlagen. Dieses Konzept findet sich in analoger Form bereits in den Kontextlisten von GTD. Die Anwendung soll dabei aber den Kontext selbst ermitteln, ohne dass der Anwender Aufgaben in speziellen Kontextlisten speichern muss. Vorschläge werden dem Anwender dabei dynamisch präsentiert und wie bei GTD werden die Entscheidungen für eine Aufgabe spontan gefällt.
4. **Keine festes Schema:** Die Anwendung soll dem Anwender kein starres Schema vorgeben, sonder sich nahtlos in seinen bisherigen Alltag integrieren und keine Gewohnheitsänderungen erzwingen.

Aus den psychologischen Studien lassen sich zudem die folgenden Kriterien ableiten:

5. **Entscheidungsfreiheit:** Dem Anwender sollen kein Aufgaben aufgezwungen werden.
6. **Begrenzte Anzahl der Entscheidungen:** Der Anwender soll mit möglichst wenigen Entscheidungen konfrontiert werden.

Viele Konzepte der bisherigen Zeitmanagementsystem lassen sich entweder nur bedingt durch eine Anwendung abdecken oder würden den Rahmen der Arbeit sprengen. So kann ein System dem Anwender nicht abnehmen selbst zu entscheiden, welche Aufgaben er in sein System übernimmt und welche Aufgaben er als unwichtig klassifiziert und nicht erledigt. Auch das Erreichen langfristiger Ziele kann durch eine Software nur schwer abgedeckt werden, sieht man von komplexen Projektmanagmentsystemen

ab. Dennoch ließe sich folgende Konzepte in die Anwendung integrieren, die aber aus Zeitgründen im aktuellen Entwurf nicht umgesetzt wurden:

7. **Routinen:** Die Anwendung könnte zum Beispiel Gewohnheiten aus den vergangenen Tagen lernen und sie bei den Vorschlägen berücksichtigen (siehe Kapitel 3.2.4).
8. **Langfristige Ziele:** Das Erreichen langfristiger Ziele könnte dadurch unterstützt werden, dass der Anwender Zusammenhänge zwischen einzelnen Aufgaben und einem langfristigen Ziel im System speichern könnte. Dadurch hat er seine langfristigen Ziele im Blick und sieht gleichzeitig den Fortschritt bei jedem Projekt.

4.3 Starrer Tagesplan

Ein naheliegender erster Ansatz ist, den Tag des Nutzer komplett zu planen. Ein derartiger Algorithmus würde zunächst alle noch nicht erledigten Aufgaben aus der Datenbank laden und anschließend für jede Aufgabe die optimale Lücke zwischen den Terminen suchen. Um dem Anwender am Ende einen perfekten Plan präsentieren zu können, muss der Algorithmus auch zukünftige Tage betrachten und zum Beispiel Aufgaben über die ganze nächste Woche verteilen. Dabei sind verschiedene Kriterien zu beachten:

- Es sollten alle Deadlines eingehalten werden und Aufgaben möglichst frühzeitig erledigt werden.
- Die Arbeitslast sollte zwischen allen Tagen gleichmäßig verteilt werden.
- Die Anordnung sollte unnötig lange Fahrzeiten vermeiden.[2]

Die Liste ließe sich noch beliebig verlängern, würde man den kompletten Kontext (siehe Kapitel 3) beachten. Die Probleme liegen dabei aber auf der Hand: Bei einem derartigen Ansatz handelt es sich um ein Optimierungsproblem mit einer Vielzahl an Nebenbedingungen. Ein Algorithmus zur Lösung dieses Problems hätte im schlimmsten Fall exponentielle Laufzeit und wäre somit nicht praxistauglich. Zudem kann nicht davon

[2] Es handelt sich dabei um ein verändertes Taveling Salesman Problem, mit dem Zuhause des Anwenders als Start- und Endpunkt und den Terminen mit festen Zeiten und Orten als Fixpunkte.

ausgegangen werden, dass der Anwender den vorgeschlagenen Plan einhalten kann oder einhalten will: Termine können länger dauern als geplant oder der Anwender möchte eine bestimmte Aufgabe im Moment nicht erledigen, da er beispielsweise zu müde dazu ist. Auch ist das weite Vorausplanen in die Zukunft problematisch, da sich bis dahin noch viele Termine ändern können. Die Folge ist, dass der Algorithmus seinen Plan permanent an die geänderten äußeren Bedingungen anpassen muss, was bei einer exponentiellen Laufzeit aber mit sichtbaren Verzögerungen einher gehen würde.

Betrachten wir den Ansatz nun unter den zuvor festgelegten Kriterien:

1. **Fokus/Flow:** Da der Anwender keine Entscheidungen treffen muss und für jeden Zeitpunkt exakt eine Aufgabe geplant ist, führt er im Idealfall genau diese Aufgabe aus. Da sich der Algorithmus darum kümmert, dass alle Termine erreicht und alle Aufgaben rechtzeitig erledigt werden, kann der Anwender sich ganz auf das Bearbeiten der Aufgabe konzentrieren und ein Flow-Erlebnis wird möglich.

3. **Kontextsensitivität/reaktives Verhalten:** Zwar wird bei der Planung der Kontext berücksichtigt, dynamisch auf Änderungen zu reagieren ist aber schwierig.

4. **Kein festes Schema:** Der Anwender ist dazu gezwungen regelmäßig mit der Anwendung zu interagieren, um keine wichtige Aufgabe zu vergessen. Außerdem muss er jede noch so kleine Aufgabe eintragen, damit die Planung funktioniert. Daher gibt die Anwendung ein zu starres Schema vor.

5. **Entscheidungsfreiheit:** Der Anwender hat keinerlei Entscheidungsfreiheit und muss sich komplett nach der Anwendung richten.

6. **Begrenzte Anzahl der Entscheidungen:** Der Anwender ist mit keinen Entscheidungen konfrontiert.

Der Vorteil des Ansatzes liegt auf der Hand. Würde der Anwender jede noch so kleine Aufgabe und jeden Termin eintragen, so würde die Anwendung den perfekten Tag planen. Der Anwender könnte entspannt die Aufgabe bearbeiten, die die Anwendung im vorgibt, und so in den Flow geraten. Dabei hat er die Sicherheit, dass alle Aufgaben rechtzeitig und zum optimalen Zeitpunkt erledigt werden.

Die Nachteile überwiegen die Vorteile aber bei weitem. Der Ansatz ist nicht besonders flexibel, gibt ein zu starres Schema vor und nimmt dem Anwender jegliche Entscheidungsfreiheit. Auch ist dieser Ansatz in der Praxis kaum umsetzbar. Es ist äußerst unwahrscheinlich und nicht erstrebenswert, dass der Anwender jede Kleinigkeit in der Anwendung als Aufgabe einträgt. Deshalb wurde dieser Ansatz, ganz unabhängig von den technischen Schwierigkeiten, nicht gewählt.

4.4 Vorschläge für jede Lücke

In Anbetracht dessen, dass der erste Ansatz zu starr ist und den Anwender bevormundet, ist der naheliegende nächste Ansatz Aufgaben nicht fest einzuplanen, sondern nur vorzuschlagen. Dazu könnte man die Lücken zwischen den Terminen betrachten und für jede Lücke Aufgaben vorschlagen. Eine Funktion könnte anhand des Kontextes für jede Aufgabe und jede Lücke einen Wert berechnen, der angibt, wie gut die Aufgabe passt beziehungsweise wie wahrscheinlich die Ausführung durch den Anwender ist. Für jede Lücke werden dann die besten Vorschläge präsentiert und der Anwender kann dynamisch eine Aufgabe für den aktuellen Zeitpunkt auswählen, aber auch Aufgaben im späteren Tagesverlauf oder für die nächsten Tage einplanen.

Für das Vorausplanen von Aufgaben sind zwei unterschiedliche Ansätze denkbar:

1. Der Anwender könnte eine Lücke auswählen. Die Aufgaben wird somit zwischen zwei Terminen oder bereits geplanten Aufgaben eingeplant. Die Anwendung merkt sich dabei keine feste Uhrzeit sondern die Aufgabe (oder den Termin) davor und danach.
2. Der Anwender legt einen bestimmten Startzeitpunkt für die Aufgabe fest.

4.4.1 Wahl der Lücke

Ermöglicht man dem Anwender die Wahl einzelner Lücken, so erreicht man für ihn die größtmögliche Flexibilität. Trifft er im Voraus keine Auswahl, so werden ihm für den aktuellen Moment die besten Vorschläge angezeigt. Trifft er hingegen ein Auswahl, so

wird keine feste Uhrzeit vermerkt, sondern nur die Aufgaben (oder der Termin) davor und danach. Dies ist ein sehr intuitiver Ansatz, der zum Beispiel den folgenden Eintrag ermöglicht: „Nach dem Banktermin (um 14 Uhr) und vor dem Abendessen (um 18 Uhr) mache ich einen einstündigen Workout.“ Da im Normalfall weder die Dauer des Termins noch die Anfahrt zum Ort der Aufgabe genau abgeschätzt werden können, ist das Eintragen eines festen Startzeitpunktes meist nicht sinnvoll. Gleichzeitig kann über die Wahl der Lücke aber dennoch im voraus Struktur in den Tag gebracht werden.

Problem bei einem derartigen Ansatz ist die aufwendige Implementierung. Kennt die Anwendung nur die Lücke einer geplanten Aufgabe, erschwert dies das Berechnen neuer Vorschläge erheblich. Es ist für die Anwendung nicht ersichtlich, ob sie eine gewählte Aufgabe als nächsten Schritt anzeigen oder vorher noch weiter Vorschläge präsentieren soll. Daher müssen beim Ermitteln der Aufgabe alle möglichen Szenarien in Betracht gezogen werden. Da der Anwendung nur die Lücke bekannt ist, muss ein relativ großes Zeitfenster für die Ausführung betrachtet werden. Dabei kann die Aufgabe zu Beginn oder am Ende des Zeitfensters ausgeführt werden:[3]

Wählt der Anwender für ein Lücke von 14:00 - 18:00 Uhr eine einstündige Aufgabe, so müssten sowohl Vorschläge für den Zeitraum zwischen 14 Uhr und 17 Uhr als auch für 15 Uhr bis 18 Uhr angezeigt werden. Je nachdem ob der Anwender die Aufgabe zu Beginn oder am Ende des Zeitfensters ausführen will. Wird dann ein weiterer Aufgabenvorschlag ausgewählt, entsteht wieder eine neue Lücke, die betrachtet werden muss.

Abgesehen von der hohen Komplexität der Implementierung erfüllt dieser Ansatz auch einige der zuvor festgelegten Kriterien nicht. Zwar wird der Anwender nun weder bevormundet (5. Kriterium), noch wird ihm ein festes Schema aufgezwungen, aber er ist stattdessen mit einer Vielzahl von Entscheidungen konfrontiert. Dies macht die Darstellung komplex und kann den Anwender leicht überfordern (Verletzung des 6. Kriteriums). Auch verleitet die Möglichkeit zum Vorausplanen dazu, sich von der aktuellen Tätigkeit ablenken zu lassen und sich mit der Planung zu beschäftigen (Verletzung des 1. Kriteriums). Wird zudem zu viel vorausgeplant geht die Dynamik der ganzen Anwendung verloren (Verletzung des 3. Kriteriums).

[3] Die Aufgabe kann zwar auch zu einem beliebigen Zeitpunkt dazwischen ausgeführt werden, dies ist für die Aufgabenvorschläge aber nicht relevant.

4.4.2 Wahl eine festen Uhrzeit

Lässt man den Anwender beim Einplanen der Aufgabe eine feste Uhrzeit wählen, so vereinfacht das sowohl die Anzeige als auch die Implementierung des Algorithmus. Für die Anwendung ist nun klar ersichtlich, wann der Anwender die Aufgabe erledigen möchte, und diese kann wie ein fester Termin behandelt werden. Analog zur Wahl der Lücke werden aber viele der oben definierten Kriterien verletzt. Zudem büßt die Anwendung nochmals an Dynamik ein.

4.5 Vorschläge für den aktuellen Moment

Schließlich wurde ein Ansatz evaluiert, bei dem eine begrenzte Anzahl an Vorschlägen ausschließlich für den aktuelle Moment präsentiert werden. Wählt der Anwender eine Aufgabe aus, so steht diese in der Anwendung im Vordergrund und der Anwender wird erst wieder mit Vorschlägen konfrontiert, wenn die Aufgabe erledigt ist. Somit muss der Anwender nur eine Entscheidung für den Moment treffen und die ganze Anwendung gewinnt an Dynamik. Da dieses Modell letzten Endes gewählt wurde, wird es detailliert in Kapitel 5 betrachtet.

5

Gesamtkonzept

Für die Anwendung wurde ein Ansatz gewählt, bei dem Vorschläge nur für den aktuellen Moment präsentiert werden. Anhand des aktuellen Benutzerkontextes und der darauf folgenden Termine werden (maximal) drei Aufgabenvorschläge präsentiert. Der Anwender kann sich dann für einen Vorschlag entscheiden oder die Vorschläge auch einfach ignorieren und auf den nächsten Termin warten. Erkennt die Anwendung, dass bis zum nächsten Termin keine Aufgabe mehr möglich ist, so wird der nächste Termin zusammen mit der Anfahrt dort hin angezeigt.

Eine Aufgabe kann sich dabei in 4 verschiedene Zuständen befinden:

1. **Offen:** Die Aufgabe ist noch nicht erledigt und wurde nicht ausgewählt. Sind alle Aufgaben entweder offen oder erledigt, so werden in der Anwendung die drei besten Aufgabenvorschläge präsentiert. Wird ein Aufgabenvorschlag ausgewählt wechselt er in den Zustand **Geplant**. Sind zum aktuellen Zeitpunkt keine offenen Aufgaben vorhanden oder keine Aufgabe möglich, wird der nächste Termin angezeigt.
2. **Geplant:** Existiert eine geplante Aufgabe, so wird diese als erstes in der Anwendung angezeigt. Um den Anwender nicht von seiner aktuellen Tätigkeit abzulenken, werden keine weiteren Aufgabenvorschläge angezeigt. Da die Aufgabe noch nicht begonnen wurde, wird noch keine Zeit für den Beginn der Aufgabe in der Datenbank gespeichert. Wird eine geplante Aufgabe begonnen (siehe Abschnitt 5.3.2) wechselt sie in den Zustand **In Bearbeitung**.
3. **In Bearbeitung:** Befindet sich eine Aufgabe in Bearbeitung, so werden keine weiteren Aufgabenvorschläge angezeigt. Wechselt eine Aufgabe in diesen Zustand,

so wird die aktuelle Uhrzeit als Startzeitpunkt in der Datenbank gespeichert. Eine Aufgabe in Bearbeitung kann entweder abgebrochen werden und wechselt wieder in den Zustand **Offen** oder sie wird als **Erledigt** markiert.

4. **Erledigt:** Die Aufgabe wird zusammen mit der Start- und Endzeit in der Datenbank archiviert. Da nun weder eine Aufgabe geplant ist, noch eine bearbeitet wird, werden wieder Aufgabenvorschläge angezeigt (Zustand: **Offen**).

Dieser Ansatz hat verschiedene Vorteile die im folgenden betrachtet werden. Anschließend wird auf das Konzept der Benutzeroberfläche im Detail eingegangen.

5.1 Oberflächenkonzept

Betrachten wir nun das Konzept der Benutzeroberfläche. Ziel beim Design der Oberfläche war es, sowohl die Anzahl der Ansichten (bei Android: Activities und Fragments), als auch die Zahl der Bedienelemente und Informationen innerhalb der Ansichten zu reduzieren. Die Anwendung soll den Benutzer nicht überfordern und auch in Stresssituationen schnell und einfach bedienbar sein. Die Ansichten sind:

- **Tagesansicht:** Die Tagesansicht ist die zentrale Komponente der Anwendung. Hier wird der nächste Handlungsschritt und die Termine des aktuellen Tages präsentiert. Über einen Button können schnell neue Termine und Aufgaben angelegt werden.
- **Eingabemaske - Termin:** Sie ermöglicht neu Termine anzulegen. Mit einem Klick kann zur Eingabemaske für Aufgaben gewechselt werden, wobei Informationen die sowohl Termine als auch Aufgaben betreffen übernommen werden (Name, Ort, . . .). Das Design dieser Oberfläche wird im Kapitel 6 genau betrachtet.
- **Eingabemaske - Aufgabe:** Sie verhält sich analog zur Termineingabemaske und wird im Kapitel 7 im Detail betrachtet.

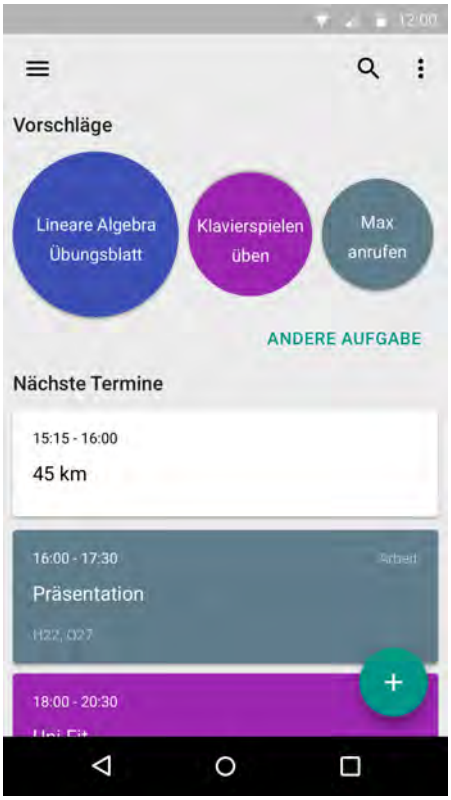

Abbildung 5.1: Tagesansicht mit Todo-Vorschlägen als nächstem Schritt

5.2 Tagesansicht

Zentrale Komponente der Anwendung ist die Ansicht für den aktuellen Tag. Das Ziel dieser Ansicht ist es, dem Benutzer übersichtlich den nächsten Handlungsschritt anzuzeigen. Alle im Moment nicht relevanten Informationen sind nur unnötige Ablenkung und sollten vermieden werden. Das Design der ganzen Anwendung orientiert sich dabei an den Material-Design-Richtlinien von Google [12].

Die Ansicht selbst ist in zwei Hälften aufgeteilt (siehe Abbildung 5.1). Im oberen Teil wird der nächste Handlungsschritt präsentiert. Dies können Aufgabenvorschläge, eine gewählte Aufgabe oder der nächste Termin sein. Für Termine und gewählte Aufgaben wird zudem der Anfahrtsweg angezeigt. Im unteren Teil der Ansicht werden alle festen

Termine des Tages chronologisch sortiert angezeigt und geben so einen Überblick über den bevorstehenden Tag.

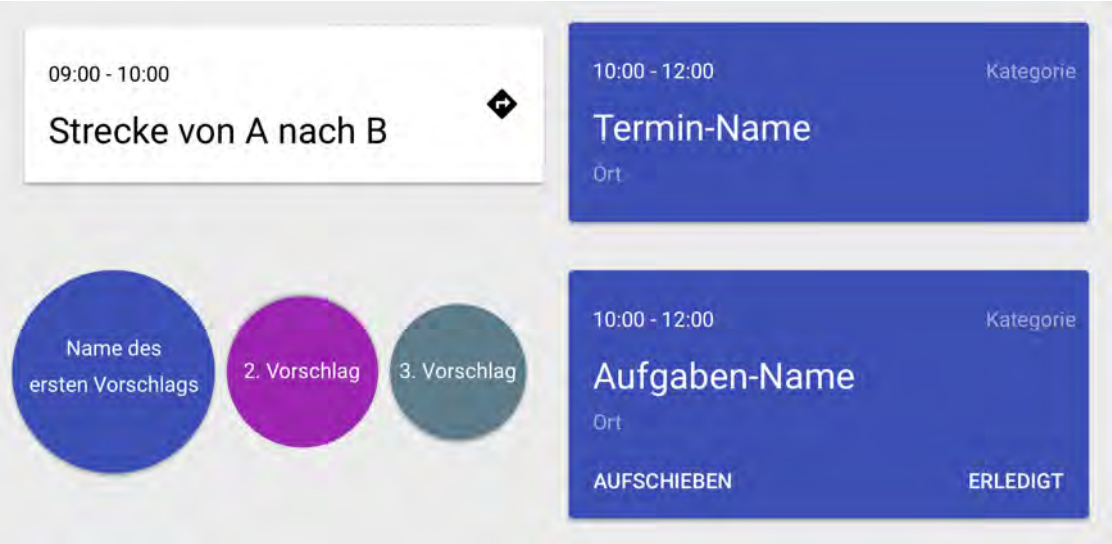

Abbildung 5.2: Karten (von links, oben): Anfahrtswege, Termin, Aufgabenvorschläge, ausgewählte Aufgabe

Die verschiedenen Informationen und Entitäten werde dabei in Form von Karten [28] angezeigt. In Abbildung 5.2 werden die verschiedenen Karten gegenübergestellt.

- **Anfahrtswege:** Es soll sowohl die Strecke als auch die geschätzte Fahrzeit angezeigt werden. Auf Klick soll die Routenplanung gestartet werden.
- **Termine:** Es wird der Name, der Start- und Endzeitpunkt, der Ort und die Kategorie angezeigt.
- **Aufgabenvorschläge:** Es werden lediglich die Namen der drei Vorschläge angezeigt. Wird ein Vorschlag angeklickt, so wird die jeweilige Aufgabe ausgewählt (siehe nächste Karte).
- **Ausgewählte Aufgabe:** Es wird der Name, der Startzeitpunkt (falls vorhanden), die Dauer, die Kategorie und der Ort angezeigt. Die Aufgabe kann entweder als erledigt markiert oder aufgeschoben werden.

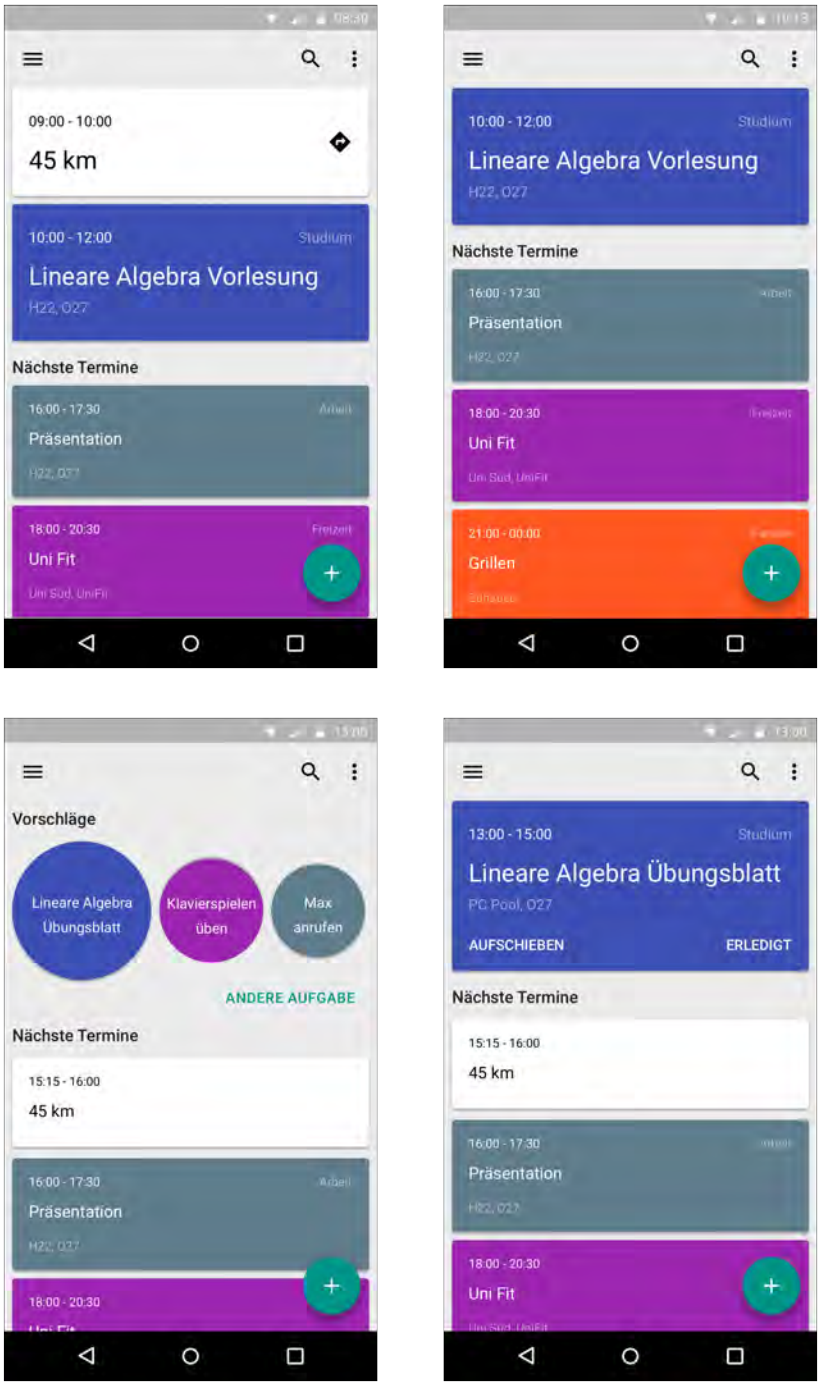

Abbildung 5.3: Tagesansichten in verschiedenen Situationen

Um die verschiedenen Entitäten auch optisch unterscheiden zu können, wurden verschiedene Formen und Farben für die Karten gewählt. Informationen zur Anfahrt sind dabei die am wenigsten relevanten und deswegen durch eine schlichte, weiße Karte repräsentiert.

In Anlehnung an klassische Kalenderanwendungen werden Termine als rechteckige Karte dargestellt. Die Farbe entspricht dabei der jeweiligen Kategorie. Hier wurden intensive Farben aus der Material-Design-Farbpalette [29] gewählt, um die Karten klar vom Hintergrund abzuheben. Anhand der unterschiedlichen Farben gewinnt der Anwender zudem schnell einen ersten Eindruck über den bevorstehenden Tag. So kann er beispielsweise schnell abschätzen, wie arbeitsreich sein Tag wird und dies in die Wahl der Aufgaben mit einfließen lassen.

Um dynamische Aufgabenvorschläge von den starren Terminen abzugrenzen, werden Vorschläge als runde Karten dargestellt. Wird ein Vorschlag ausgewählt, ist er fest im Tagesplan und erscheint, wie die Termine, als rechteckige Karte. Die Hintergrundfarben der Karten entspricht wieder der Farbe der Kategorie.

Da die Oberfläche durch die farbigen Karten bereits sehr bunt ist und um den Anwender nicht vom eigentlichen Inhalt abzulenken, wurde ein sehr schlichtes Design in verschiedenen Graustufen gewählt. Die Navigationsleiste am oberen Rand der Ansicht, ist hierbei entgegen der Design Richtlinien teil des Hintergrundes.

Der Plus-Button am rechten unteren Rand bietet die Möglichkeit schnell neue Aufgaben und Termine einzutragen und startet die Eingabemasken. Diese werden in den folgenden Kapiteln noch ausführlich vorgestellt.

In Abbildung 5.3 werden die verschiedenen Situationen nochmal gegenübergestellt. So steht in der ersten Ansicht ein Termin bevor, der Anwender befindet sich aber noch nicht am richtigen Ort. In der zweiten Ansicht befindet sich der Anwender gerade mitten in einem Termin. In der dritten Ansicht werden Aufgabenvorschläge angezeigt. Und in der Vierten wurde schließlich eine Aufgabe ausgewählt. Der Anwender befindet sich dabei bereits am richtigen Ort, weshalb keine Karte für die Anfahrt angezeigt wird.

5.3 Client-Serverinteraktion

Innerhalb der Tagesansicht sind die folgenden Interaktionen möglich:

1. Abrufen des Tagesplans (Aufgabenvorschläge und nächste Termine)
2. Auswahl eines Aufgabenvorschlags
3. Aufschieben einer gewählten Aufgabe
4. Aufgabe als erledigt markiert

Nun stellt sich die Frage, wie die Interaktion zwischen Client und Server ablaufen und die Schnittstelle konzipiert werden soll. Hier ist vor allem das Abrufen des Tagesplans interessant und wird im folgenden betrachtet. Die Schnittstelle für die Aufgabenverwaltung (Punkt 2 - 4) wird in Kapitel 7 im Detail besprochen, weshalb hier nur kurz auf die Besonderheiten bei der Auswahl eines Aufgabenvorschlages eingegangen wird.

5.3.1 Abrufen des Tagesplans

Beim Tagesplan stellt sich die Frage, ob der Client diesen abrufen soll oder der Server bei geänderten Vorschlägen einen neuen Plan an den Client pusht. Ein Ansatz, bei dem der Server Änderungen an den Client sendet, würde dem reaktiven Konzept der Anwendung entgegen kommen und der Anwender würde so immer sofort über Änderungen informiert werden. Seitens der Implementierung hätte dies aber gravierende Konsequenzen. Ein naheliegendes Vorgehen wäre, immer dann neue Vorschläge zu berechnen, wenn sich der Kontext in größerem Umfang ändern. Ein Teil der Kontextinformationen muss aber explizit von anderen Webservices erfragt werden und viel Kontextinformationen sind abhängig von der Zeit und ändern sich somit ständig (insbesondere die Zeit selbst). Daher wäre der Server gezwungen in sehr kurzen Zeitabständen für jeden Benutzer den Kontext neu zu ermitteln und neue Vorschläge zu berechnen. Dies ist einerseits mit Node.js schwer zu implementieren, würde aber ganz unabhängig von der Implementierung den Server stark belasten. Gleichzeitig werden so aber auch viele unnötige Vorschläge berechnet, weil der Anwender zum Beispiel sein Smartphone ausgeschaltet hat und so nicht über die Änderungen informiert werden kann. Schließlich wäre eine derartige

Serveranwendung auch schwer zu skalieren, da sich die einzelnen Serverinstanzen absprechen müssten, wer die Vorschläge für einen bestimmten Benutzer berechnet.

Daher wurde die API so implementiert, dass der Client den Plan beim Server abruft:

REST Befehl	Beschreibung
GET /now?location=<lat>,<lon>	Abrufen des Tagesplans

Tabelle 5.1: HTTP-Schnittstelle zum Abrufen der Aufgabenvorschläge und nächsten Termine (Tagesplan)

Der Client fragt dabei in regelmäßigen Abständen den Plan beim Server ab. Um die Anzahl der Serveraufrufe zu minimieren, sendet er dabei direkt die aktuelle Position des Benutzers an den Server. Der Server berechnet darauf hin einen neuen Plan und sendet ihn als Antwort an den Client. Dadurch ist sichergestellt, dass nur Vorschläge berechnet werden, die der Anwender tatsächlich zu Gesicht bekommt. Gleichzeitig muss der Server nicht für jeden Benutzer einen Timer starten, um regelmäßig den Plan zu berechnen. Betrachten wir nun die Auswahl eines Aufgabenvorschlags.

5.3.2 Auswahl eines Aufgabenvorschläge

Wird eine Aufgabevorschlag ausgewählt, so sind die beiden folgenden Situationen denkbar:

1. Der Anwender befindet sich bereits am richtigen Ort, um mit der Aufgabe zu beginnen. Somit sollte direkt in den Zustand **In Bearbeitung** (siehe Seite 31) gewechselt werden.

2. Der Anwender wählt eine Aufgabe aus für deren Bearbeitung er sich an einem anderen Ort befinden muss. In diesem Fall sollte zunächst die Anfahrt angezeigt werden. So lange der Anwender sich noch nicht am richtigen Ort befindet, sollte die Aufgabe lediglich in den Zustand **Geplant** wechseln. Anschließend wird auf dem Smartphone ein Geofence [30] für den entsprechenden Ort eingerichtet. Das Smartphone prüft dann selbständig, ob es sich in der Nähe des Ortes befindet und sendet ein Event an die Anwendung, sobald der Ort erreicht ist. Befindet sich

der Anwender schließlich am richtigen Ort, wird die Aufgabe als **In Bearbeitung** markiert und die aktuelle Uhrzeit in der Datenbank hinterlegt. Dadurch kann bei späteren Analysen die tatsächlich Aufgabendauer aus der Datenbank gelesen werden, ohne dass sie durch die Anfahrtszeit verfälscht ist.

Mit diesem Ansatz muss der Client zwar einen Teil der Programmlogik kennen, dafür werden aber sowohl die Ressourcen des Clients, als auch des Servers geschont. Geofences sind hinsichtlich ihres Akkuverbrauches stark optimiert und verbrauchen deutlich weniger Energie, als wenn die GPS-Position explizit angefragt wird. Zudem wird der Server nur dann informiert, wenn sich der Zustand einer Aufgabe tatsächlich ändert.

Der alternative Ansatz wäre, die GPS-Position regelmäßig an den Server zu senden. Der Server würde dann prüfen, ob die GPS-Position mit dem gesuchten Ort übereinstimmt und den Zustand der Aufgabe entsprechend anpassen. In diesem Fall wäre die Programmlogik zwar komplett im Server implementiert, dieser müsste aber eine Vielzahl an Clientanfragen bearbeiten, von denen die meisten zu keiner Zustandsänderung führen. Gleichzeitig würde die regelmäßige Abfrage der GPS-Position zusammen mit der Übertragung an den Server, den Akku des Smartphones stark belasten.

5.4 Vorteile des Ansatzes

Betrachten wir den Ansatz nun im Hinblick auf die in Abschnitt 4.2 festgelegten Ziele:

1. **Fokus/Flow:** Es werden nur Vorschläge für den aktuellen Zeitpunkt präsentiert. Wird eine Aufgabe ausgewählt, so steht diese im Vordergrund und es ist keine weitere Interaktion mit der Anwendung nötig. Insbesondere werden keine weiteren Aufgabenvorschläge angezeigt. Der Anwender ist somit nicht mit weiteren Entscheidungen konfrontiert und wird nicht dazu verleitet, über zukünftige Aufgaben nachzudenken. Stattdessen kann er sich ganz auf das hier und jetzt konzentrieren und die ausgewählte Aufgabe ohne Ablenkung erledigen.
2. **Sammeln:** Das leichte Anlegen neuer Aufgaben hat nicht direkt mit dem gewählten Ansatz zu tun, sondern ist primär eine Eigenschaft der Eingabemasken. Beim

Entwurf der Oberfläche wurde aber dafür gesorgt, dass die Eingabemasken schnell über den Plus-Button erreichbar sind (siehe Abschnitt 5.1).

3. **Kontextsensitivität / reaktives Verhalten** Die Anwendung betrachtet den aktuellen Kontext und den nächsten Termin. Da es keinen starren Plan für die Zukunft gibt, kann die Anwendung dynamisch auf Änderungen des Kontextes reagieren und die Vorschläge anpassen. Ein Beispiel dazu wäre, dass der Anwender spontan durch die Fußgängerzone läuft. Hätte er den Tag starr vorausgeplant, könnte die Anwendung nicht darauf reagieren. Durch den neuen Ansatz kann die Anwendung jetzt aber auf den geänderten Kontext reagieren und zum Beispiel eine Aufgabe in einem Geschäft in der Nähe vorschlagen.
4. **Kein festes Schema:** Die Anwendung gibt kein festes Schema vor. Aufgaben können zu jedem beliebigen Zeitpunkt angelegt werden, eine komplexe Verwaltung der Aufgaben ist nicht nötig und Vorschläge werden immer dynamisch zu jedem Zeitpunkt präsentiert. Der Anwender muss also keine neuen Verhaltensweisen lernen, sondern kann ab und zu einen Blick auf die Anwendung werfen und dann eine Aufgabe wählen.
5. **Entscheidungsfreiheit:** Die Anwendung reagiert dynamisch auf den Kontext und präsentiert passende Vorschläge. Für welchen der Vorschläge sich der Anwender entscheidet und ob er sich überhaupt für das Erledigen einer der Aufgaben, entscheidet liegt bei ihm. Dem Anwender wird also keine Entscheidung aufgezwungen.
6. **Begrenzte Anzahl an Entscheidungen:** Der Anwender muss nur eine Entscheidung für den aktuellen Moment treffen. Dabei werden ihm bereits die drei besten Vorschläge präsentiert. Dadurch ist der Anwender in jedem Moment nur mit einer einzigen, nicht übermäßig komplexen, Entscheidung konfrontiert und seine mentalen Ressourcen werden geschont. Durch die geringe Komplexität hat der Anwender die Sicherheit, eine gute Entscheidung getroffen zu haben und kann sich so komplett auf die gewählte Aufgabe konzentrieren.

Auch unter technischen Gesichtspunkten hat dieser Ansatz viele Vorteile. Da es immer nur eine Aufgabe geben kann, die als nächster Schritt geplant ist oder gerade ausgeführt

wird, sind viele Anwendungsfehler ausgeschlossen. Auch kann es so maximal eine geplante Aufgabe geben, die vergessen wurde und nun aufgeschoben werden muss. Die Anwendung muss daher nur wenige Spezialfälle beachten. Des weiteren sind die Vorschläge zuverlässiger, da es einfacher ist den aktuellen Benutzerkontext zu erfassen, als den zukünftigen Kontext vorauszusagen.

6

Termine

Feste Termine bilden Fixpunkte innerhalb des Tagesplanes und geben den Rahmen für die Aufgabenvorschläge vor. Ein Termin ist eine Tätigkeit, die zu einem fest vorgegebenen Zeitpunkt stattfindet. Dabei kann zwischen verschiedenen Arten von Terminen unterschieden werden:

1. **Einzelner Termin:** Ein einmaliges Ereignis mit fester Uhrzeit und fester Dauer. Beispiele hierfür sind Prüfungstermine, Konzerte, Arzttermine, etc.
2. **Termin mit endlicher Anzahl unregelmäßiger Wiederholungen:** Ein Beispiel ist ein mehrtägiges Seminar von Freitagmittag bis Sonntagabend. Hierbei handelt es sich nicht um Wiederholungen im klassischen Sinn, da sich sowohl die Uhrzeiten, als auch das Intervall ändern kann. Sinnvoll ist hier, die Uhrzeiten jeweils einzeln einzutragen. Um jedoch die Eingabe zu erleichtern und der Anwendung den logischen Zusammenhang deutlich zu machen, ist eine Eingabe als Termingruppe sinnvoll.
3. **Termine mit (un-)endlicher Wiederholung mit festem Intervall:** Beispiele hierfür sind wöchentlich stattfindende Vorlesungen oder Meetings, aber auch komplexere Termin sind denkbar. So soll zum Beispiel auch eine Vorlesung möglich sein, die jede Woche Montag und Mittwoch stattfindet. Deshalb sollte auch die Eingabe mehrerer Starttermine, zusammen mit einem festen Intervall und einem optionalen Enddatum der Wiederholungen unterstützt werden. Auch Ausnahmen vom Intervall sollen möglich sein, da ein Termin entfallen oder zu einer anderen Zeit, an einem anderen Ort stattfinden könnte.

All diese Arten von Terminen sollen unterstützt werden. Wichtig dabei ist, die Eingabe für den Benutzer möglichst einfach zu gestalten. So ist es beispielsweise nicht praktikabel den Anwender zunächst aus den drei Kategorien wählen zu lassen, da die Namensgebung der einzelnen Kategorien schwierig ist und verwirrend sein kann. Es wurde deshalb eine einzige Eingabemaske für alle drei Kategorien konzipiert.

6.1 Eingabemaske

Die Eingabemaske ist zweigeteilt (siehe Abbildung 6.1). Im oberen Teil werden Kontextinformationen erfragt, die sowohl bei Terminen als auch bei Aufgaben relevant sind (siehe Kapitel 3.3.2). Über ein Dropdown-Menü kann gewählt werden, ob es sich um einen Termin oder eine Aufgabe handelt. Wechselt man zwischen Termin- und Aufgabeneingabe, werden die gemeinsamen Kontextinformationen übernommen. Die Hintergrundfarbe des oberen Teils ist die Farbe der gewählten Kategorie. So entspricht der obere Teil der Ansicht im Wesentlichen der, der Karten in der Tagesansicht, und stellt so einen optischen Zusammenhang zwischen beiden her.

Im unteren Teil werden die Uhrzeiten des Termins erfragt. Für die Eingabe dieser Informationen wurde eine sehr minimalistische Ansicht gewählt (siehe Abbildung 6.1). Zunächst können mehrere feste Termine eingegeben werden. Wird nur einer angegeben, so entspricht dies der ersten Kategorie (einzelner Termin). Werden mehrere Termine angelegt, so fällt er in die zweite Kategorie (unregelmäßige Wiederholung). Wird schließlich der **Wiederholungen**-Switch aktiviert, ist der Termin aus der dritten Kategorie und alle Termine der Liste werden in festem Intervall wiederholt, wobei ein Enddatum über den **Bis**-Switch gesetzt werden kann.

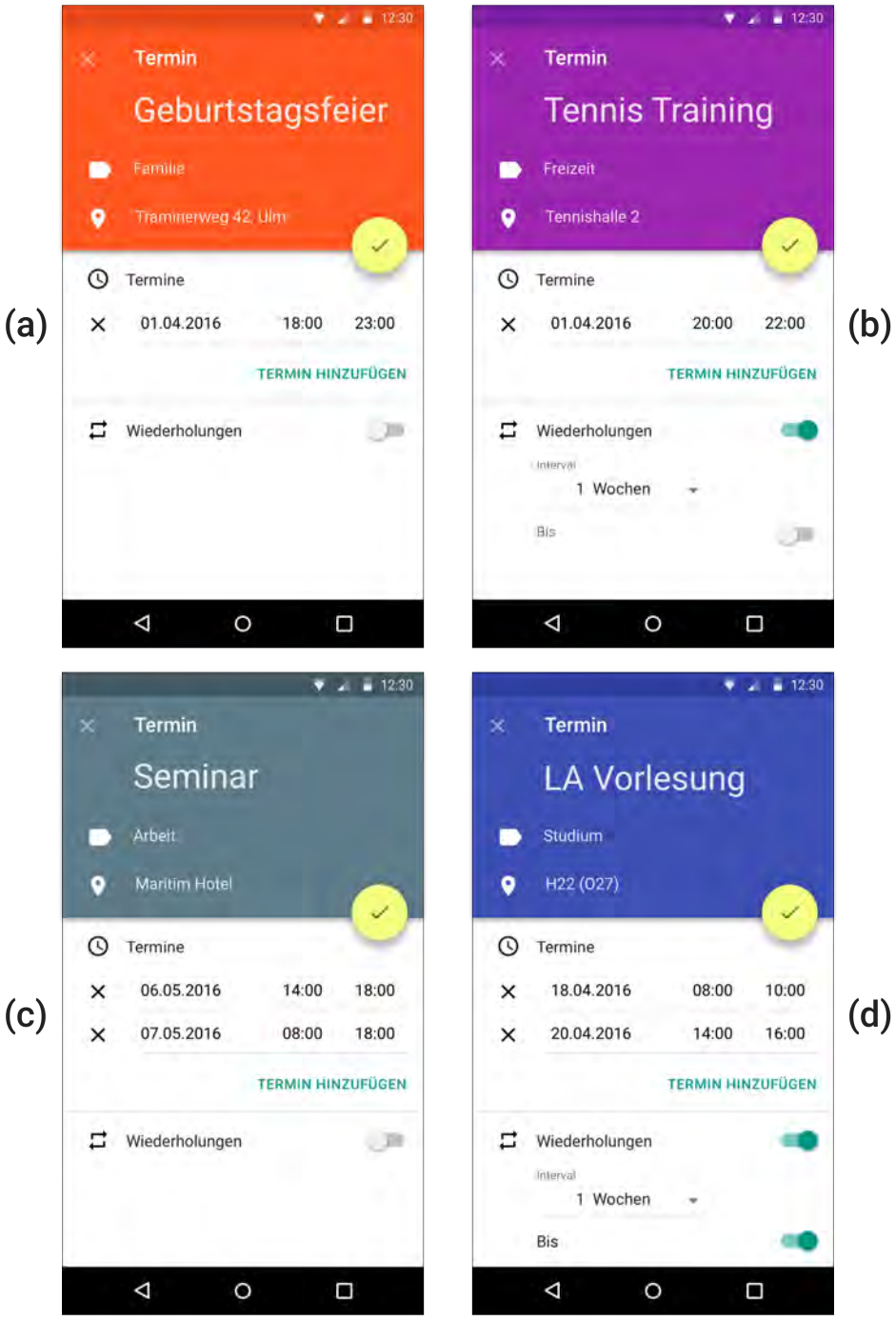

Abbildung 6.1: Eingabemasken - Termine: einmaliger Termin (a), Termin mit festem Wiederholungsintervall (b), Termingruppe (c), Termingruppe mit festem Wiederholungsintervall (d)

6.2 Implementierung

Im Folgenden werden einige interessante Details der Server-Implementierung besprochen. Die Schwierigkeit dabei ist, ein möglichst einfaches Schema zur Darstellung aller Arten von Terminen zu finden.

6.2.1 iCalendar Format

Das iCalendar Format [31] ist der derzeitige Standard für den Austausch von Terminen und Kalendern zwischen verschiedenen Anwendungen. Es liegt daher zunächst nahe, das iCalendar Format zur Speicherung und zum Austausch der Daten in Betracht zu ziehen.

Bei genauer Betrachtung werden aber verschiedene Schwächen des iCalendar Formats deutlich:

- Bei iCalendar handelt es sich um ein komplett eigenständiges Format. Es kann daher nicht auf einen standard JSON- oder XML-Parser zurückgegriffen werden.
- Es handelt sich um ein textbasiertes Format, das sich nur umständlich in ein Datenbank Schema umwandeln lässt.
- Das Textformat müsste auf dem Server erst geparsed und in ein JSON Dokument überführt werden um weiter verarbeitet zu werden.
- Auch der Client kann nicht direkt mit dem Format arbeiten, sondern muss es zunächst aufwendig parsen.
- Obwohl es sich um einen Standard handelt, wird das Format sehr unterschiedlich unterstützt und interpretiert. Selbst wenn die Anwendung das iCalender Format unterstützen würde, wäre ein reibungsloser Datenaustausch nicht garantiert [32].
- Da das iCalendar Format ein bestehender Standard ist, kann es nicht ohne weiteres erweitert werden und schränkt so Kreativität beim entwickeln einer neuen Anwendung ein.

Aus diesen Gründen fiel die Entscheidung zu Gunsten eines eigenen JSON Formates aus. Dies lässt sich direkt in Node.js verarbeiten und in der MongoDB speichern. Auch Android kann JSON Dokumente leicht verarbeiten und in Java-Objekte umwandeln (siehe Kapitel 2), so dass die Entitäten in allen Anwendungsschichten die gleiche Struktur aufweisen. Dies vereinfacht die Entwicklung und das Debugging drastisch. Zudem lässt sich ein JSON Dokument leicht um weitere Felder erweitern und so im Laufe der Entwicklung an neue Anforderungen anpassen.

Auch für eine zukünftige Webversion der Anwendung oder eine API für Drittanbieter ist ein Datenaustausch mittels JSON von Vorteil und mittlerweile der De-facto-Standard.

Sollte später die Einbindung andere Kalender nötig sein, ließe sich das iCalender Format (unter Umständen mit Informationsverlust) in das JSON Format konvertieren. Eine iCalendar Schnittstelle für die Anwendung selbst macht aber keinen Sinn, da die Anwendung zu stark von einem klassischen Kalender abweicht. Zwar ließen sich Termine im iCalendar Format austauschen, aber für Aufgaben existiert keine sinnvolle Repräsentation und die Anzeige in einer normalen Kalenderanwendung würde so alle Vorteile zunichtemachen.

6.2.2 Schema

Das folgende Schema ermöglicht es, alle oben beschriebenen Termintypen in der Datenbank zu speichern. Das Schema ist dabei in der Notation von Mongoose verfasst (siehe Kapitel 2).

Wie in Kapitel 3.3.2 besprochen, verfügen Termine und Aufgaben über die gemeinsamen Kontextinformationen Name, Ort und Kategorie. Um dies in der Datenbank wieder zu spiegeln und einen einheitlichen Zugriff auf diese Informationen, unabhängig vom Typ der Entität, zu ermöglichen, wurden diese Informationen in ein eigenes Unterschema ausgelagert. Da Informationen über den Eigentümer einer Entität ebenfalls unabhängig vom Typ sind, sind auch diese Informationen Teil des Unterschemas.

```
1 let metaSchema = new mongoose.Schema({
2   titel : {
3     type: String,
```

```
4       required: true,
5       trim: true
6     },
7     category : {
8       type: String,
9       required: true,
10      enum: ['family', 'friends', 'work', 'school']
11    },
12    location: {
13        type: mongoose.Schema.Types.ObjectId,
14        ref: 'Location',
15        required: false
16    },
17    owner: {
18        type: mongoose.Schema.Types.ObjectId,
19        ref: 'User',
20        required: true
21    }
22  });
```

Listing 6.1: Gemeinsame Kontextinformationen für Termine und Aufgaben

Informationen zu Wiederholungen und Ausnahmen davon wurden ebenfalls in eigene Schemata ausgelagert. Ausnahmen vom Wiederholungsintervall werden dabei in einer Art Map gespeichert. Über das Feld `date` wird das ursprüngliche Datum des Termins referenziert, über die Felder `newDate`, `newLocation` und `canceled` können dann die Uhrzeit und der Ort geändert oder der Termin gestrichen werden.

```
1   let repetitionSchema = mongoose.Schema({
2     interval: {
3       type: String, // ISO8601 Intervall
4       required: true
5     },
6     end: {
7       type: Date,
8       required: false
9     }
10  });
11
```

```
12 // Schema fuer Ausnahmen (anderer Ort, andere Uhrzeit, ...)
13 let exceptionSchema = mongoose.Schema({
14   date: {
15     type: Date,
16     required: true
17   },
18   newDate: {
19     type : {
20       start: {type: Date, required: true},
21       end: {type: Date, required: true}
22     },
23     required: false
24   },
25   newLocation: {
26       type: mongoose.Schema.Types.ObjectId,
27       ref: 'Location',
28       required: false
29   },
30   canceled: {
31     type: Boolean,
32     required: true,
33     default: false
34   }
35 });
```

Listing 6.2: Schema für Wiederholungen mit Ausnahmen

Schließlich werden alle Schemata zum Termin-Schema zusammengefasst und somit ein kompletter Termin als einzelnes Dokument in der Datenbank gespeichert. Termine können so leicht abgerufen und atomar bearbeitet werden. Im Feld `dates` werden dabei alle Termine der Termingruppe gespeichert. Wird nur ein Termin eingetragen und `repetition` nicht gesetzt, so handelt es sich um einen einzelnen Termin und somit die 1. Kategorie. Werden hingegen mehrere Termine in `dates` eingetragen, handelt es sich um eine Termingruppe (2. Kategorie). Wird schließlich `repetition` gesetzt, handelt es sich um einen sich wiederholenden Termin der 3. Kategorie.

```
1 let appointmentSchema = mongoose.Schema({
2   meta: {
```

```
3       type: metaSchema,
4       required: true
5     },
6     dates: {
7       type : [{
8         start: {type: Date, required: true},
9         end: {type: Date, required: true}
10      }]
11    },
12    repetition: {
13      type: repetitionSchema,
14      required: false
15    },
16    exceptions: [exceptionSchema]
17  });
```

Listing 6.3: Termin-Schema

6.2.3 HTTP Schnittstelle

Für Termine wurde folgende REST-konforme HTTP-Schnittstelle implementiert:

HTTP Aufruf	Beschreibung
POST /appointment	Termin anlegen
PUT /appointment/:id	Termin bearbeiten
DELETE /appointment/:id	Termin löschen
PUT /appointment/:id/:date	Termin mit dem Startzeitpunkt :date ändern (nur bei wiederholenden Terminen).
DELETE /appointemnt/:id/:date	Termin mit dem Startzeitpunkt `:date` löschen (nur bei wiederholenden Terminen).

Tabelle 6.1: HTTP-Schnittstelle der Terminverwaltung

Über die Schnittstelle kann sowohl die komplette Termingruppe, als auch ein einzelner Termin, bei sich wiederholenden Terminen, geändert werden. Das Ändern aller zukünftigen Termine, wird nicht direkt in der API unterstützt. Der Client kann aber zu nächsten den bestehenden Termin ändern und das Enddatum der Wiederholungen auf

den aktuellen Tag setzen. Dann legt er einen neuen Termin mit den geänderten Daten an.

Beim Anlegen eines neuen Termins wird ein, dem Schema 6.3 entsprechendes, JSON-Dokument an den Server übertragen. Als Antwort erhält der Client das Dokument zusammen mit einer eindeutigen ID. Da mit jedem POST-Befehl ein neuer Termin in der Datenbank hinterlegt wird und eine neue ID generiert wird, ist dieser Befehl nicht idempotent. Würde der Client selbständig eine eindeutige ID generieren, könnte auch der POST-Befehl idempotent gestaltet werden.

7

Aufgaben (To-dos)

Die Aufgabenverwaltung ist der zentrale Bestandteil der Anwendung. Aufgaben haben keinen festen Startzeitpunkt, aber unter Umständen Nebenbedingungen, die den Ausführungszeitpunkt einschränken. Zunächst werden mögliche Anwendungsszenarien betrachtet. Anschließend wird ein Datenbankschema für die Aufgaben entworfen und auf wichtige Details der Implementierung eingegangen.

7.1 Anwendungsszenarien und Nebenbedingungen

Für Aufgaben sind verschiedenste Anwendungsszenarien denkbar:

1. **Einmalige Aufgaben ohne Nebenbedingungen:** Der einfachste Fall ist eine Aufgabe, die einmal stattfindet und über kein festes Fälligkeitsdatum (Deadline) verfügt. Die möglichen Ausführungszeitpunkt sind nur durch die Anfahrtswege und die Dauer begrenzt.
2. **Einmalige Aufgabe mit Fälligkeitsdatum:** Verfügt die einmalige Aufgabe über ein festes Fälligkeitsdatum, so sollte die Anwendung sicherstellen, dass die Aufgabe an Priorität gewinnt, je näher das Fälligkeitsdatum rückt, und bei den Vorschlägen bevorzugt wird. Wie im ersten Fall, müssen zusätzlich die Anfahrtswege und die Dauern beachtet werden. Ein Beispiel ist eine Übungsblatt, das bis zu einem bestimmten Datum abgegeben werden muss.
3. **Aufgaben mit sich wiederholender Deadline:** Es sind auch Aufgaben mit einer sich wiederholenden Deadline denkbar. Ein Beispiel dafür ist ein Übungsblatt das jeden Freitag bis 12:00 Uhr abgegeben werden muss. Trägt man dieses als sich

wiederholende Aufgabe ein, so muss die Aufgabe nur einmal zu Semesterbeginn angelegt werden und der Anwender wird jede Woche an die Bearbeitung des neuen Übungsblattes erinnert.

4. **Aufgaben mit bestimmter Quote:** Betrachtet man zum Beispiel den Besuch im Fitnessstudio, stellt man fest, dass keines der bisherigen Szenarien dies adäquat abdeckt. Hier kann die Eingabe einer bestimmten Quote sinnvoll sein, wie zum Beispiel „3 mal pro Woche“. Diese ließe sich noch um weitere Nebenbedingungen erweitern, wie zum Beispiel „3 mal pro Woche, mit mindesten einem Tag Abstand“.

5. **Aufgaben mit Teilaufgaben:** Ziel dieser Arbeit ist es nicht, ein komplettes Projektmanagementsystem zu entwickeln. Dennoch können Teilaufgaben in begrenzten Umfang sinnvoll sein. So wäre zum Beispiel ein einfacher Ansatz im Stile der Get Things Done Methode [22] denkbar: Man ermöglicht dem Anwender zu einer Aufgabe Teilaufgaben mit Namen und Dauer hinzuzufügen. Die Anwendung zieht dann als möglichen Aufgabenvorschlag immer nur die erste noch nicht erledigte Teilaufgabe heran. Da sich diese Teilaufgabe wie eine einmalige Aufgabe (1. Kategorie) verhält, müsste der Planungsalgorithmus nicht angepasst werden.

Für Aufgaben sind noch viele weitere Nebenbedingungen denkbar, die teilweise in Kapitel 3 kurz erwähnt wurden. Im folgenden Entwurf und der Implementiert des Prototypen wurden die ersten vier Kategorien umgesetzt, auf die Unterstützung von Teilaufgaben wurde hingegen verzichtet. Obwohl Teilaufgaben von praktischer Relevanz sind, sind sie im Rahmen dieser Arbeit von untergeordnetem Interesse, da ihr Verhalten dem einzelner Aufgaben ähnelt.

7.2 Eingabemaske

Alle zuvor beschriebenen Aufgabentypen sollen über eine einzige Eingabemaske angelegt werden können. Gleichzeitig darf der Benutzer nicht durch zu viele Möglichkeiten überfordert werden. Deshalb verfügt die Eingabemaske zunächst nur über ein Feld das die Dauer erfragt (siehe Abbildung 7.1). Über die drei Switches **In Teilaufgaben zerlegen**, **Fälligkeitsdatum** und **Wiederholungen** kann der Anwender dann implizit eine

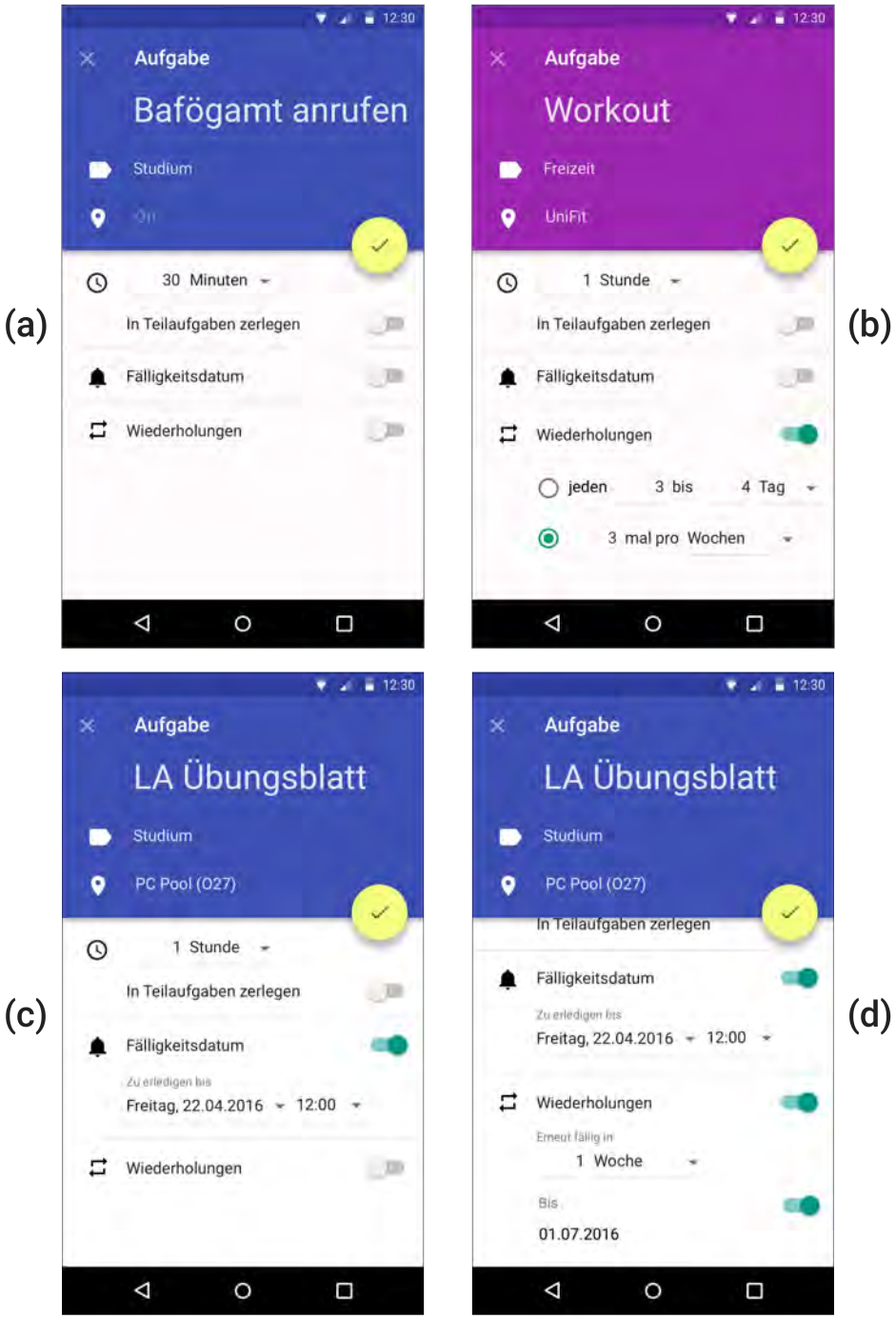

Abbildung 7.1: Eingabemasken - Aufgaben: einmalige Aufgabe (a), Aufgabe mit fester Quote (b), Aufgabe mit Fälligkeitsdatum (c), wiederholende Aufgabe (d)

der obigen Kategorien wählen und es werden nur die jeweils relevanten Eingabefelder angezeigt [33, 34]:

(a) Keiner der Switches aktiviert: Es handelt sich um eine einmalige Aufgabe ohne Fälligkeitsdatum.

(b) **Wiederholungen** aktiviert, **Fälligkeitsdatum** nicht aktiviert: In diesem Fall kann es sich nur um eine Aufgabe mit einer festen Quote handeln (4. Kategorie). Statt nach einem Wiederholungsintervall zu fragen, wird vom Anwender eine Quote und ein dazugehöriges Intervall erfragt (zum Beispiel: 3 mal pro Woche).

(c) **Fälligkeitsdatum** aktiviert: Es handelt sich somit um eine einmalige Aufgabe mit Fälligkeitsdatum (2. Kategorie). Vom Anwender wird daher ein konkretes Fälligkeitsdatum abgefragt.

(d) **Fälligkeitsdatum** aktiviert, **Wiederholungen** aktiviert: Es handelt sich um eine regelmäßige Aufgabe mit festem Fälligkeitsdatum (3. Kategorie). Vom Anwender wird daher ein konkretes Fälligkeitsdatum und ein Wiederholungsintervall für das Fälligkeitsdatum abgefragt. Optional kann ein Enddatum für die Wiederholungen eingetragen werden (**Bis**-Switch).

Über den Switch **In Teilaufgaben zerlegen** könnte zudem eine Aufgabe mit Teilaufgaben angelegt werden. In diesem Fall wird das Feld zur Eingabe der Dauer ausgeblendet, stattdessen können Teilaufgaben mit einem Namen und einer Dauer hinzugefügt werden.

Vorteil dieses Ansatzes ist, dass der Anwender nicht mit verschiedenen Arten von Aufgaben konfrontiert wird, sondern über eine einzige Eingabemaske alle möglichen Aufgaben eintragen kann. Dabei stellt die Anwendung sicher, dass nur sinnvolle Kombinationen eingegeben werden können.

7.3 Schema

Das folgende Schema ermöglicht alle oben beschriebenen Aufgabentypen in der Datenbank zu speichern. Das Schema ist dabei in der Notation von Mongoose angegeben (siehe Kapitel 2).

Wie in Kapitel 3.3.2 besprochen, verfügen Termine und Aufgaben über die gemeinsamen Kontextinformationen: Name, Ort und Kategorie. Um dies in der Datenbank wieder zu spiegeln und einen einheitlichen Zugriff auf diese Informationen unabhängig vom Typ der Entität zu ermöglichen, wurden diese Informationen in ein eigenes Unterschema ausgelagert.

```
let metaSchema = new mongoose.Schema({
  titel : {
    type: String,
    required: true,
    trim: true
  },
  category : {
    type: String,
    required: true,
    enum: ['family', 'friends', 'work', 'school']
  },
  location: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Location',
      required: false
  },
  owner: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true
  }
});
```

Listing 7.1: Gemeinsame Kontextinformationen für Termine und Aufgaben

Analog zu Terminen werden auch Aufgaben durch ein einzelnes Dokument in der Datenbank repräsentiert. Alle Nebenbedingungen einer Aufgabe werden dabei im Feld `constraints` zusammengefasst.

```
let repetitionSchema = new mongoose.Schema({
  interval: {
    type: String, // ISO8601 Intervall
    required: true
```

```
5     },
6     end: {
7       type: Date,
8       required: false
9     }
10  });
11
12  let deadlineSchema = new mongoose.Schema({
13    date: {
14      type: Date,
15      required: true
16    },
17    repetition: {
18      type: repetitionSchema,
19      required: false
20    }
21  });
22
23  let quotaSchema = new mongoose.Schema({
24    value: {
25      type: Number,
26      required: true
27    },
28    {
29      type: String,
30      required: true,
31      enum: ["week', 'month']
32    },
33    end: {
34      type: Date,
35      required: false
36    }
37  });
38
39  let constraintSchema = new mongoose.Schema({
40    duration: {
41      type: String, // ISO8601 Intervall
42      required: true
```

```
43    },
44    deadline: {
45      type: deadlineSchema,
46      required: false
47    },
48    quota: {
49      type: quotaSchema,
50      required: false
51    }
52  });
53
54  let todoInstanceDateSchema = new mongoose.Schema({
55      start: Date,
56      end: Date
57  });
58
59  let todoSchema = new mongoose.Schema({
60    meta: {
61      type: metaSchema,
62      required: true
63    },
64    constraints: {
65      type: constraintSchema,
66      required: true
67    },
68    instance_date: {
69      type: todoInstanceDateSchema,
70      required: false
71    },
72    _inExecution: {
73      type: Boolean,
74      required: true,
75      default: false
76    },
77    _next: {
78      type: Boolean,
79      required: true,
80      default: false
```

```
81    }
82 });
```

Listing 7.2: Aufgaben-Schema

7.4 Implementierung

Wie in Kapitel 5 besprochen, können sich Aufgaben in einem der folgenden vier Zustände befinden:

1. **Offen:** Noch nicht erledigt und somit als Vorschlag relevant
2. **Geplant:** Als nächste Aufgabe ausgewählt, aber noch nicht begonnen
3. **In Bearbeitung:** Die Aufgabe wird gerade bearbeitet. Dabei sollte die Startzeit für spätere Analysen gespeichert werden.
4. **Erledigt:** Die Aufgabe wurde erledigt. Die Start- und Endzeit soll in der Datenbank gespeichert werden.

Um diese Zustände zu repräsentieren, verfügen Aufgaben über die Felder `_next`, `_inExecution` und `instance_date`.

Ist das `_next` Feld `true`, so ist die Aufgabe im Zustand **Geplant**. Ist `_inExecution` `true` so ist die Aufgabe **In Bearbeitung**. Dabei enthält `instance_date.start` die Uhrzeit, zu der die Ausführung begonnen wurde.

Nun wäre prinzipiell ein Feld `_done` denkbar, das die Aufgabe als erledigt markiert, während in `instance_date.end` das Ende der Ausführung vermerkt wird. Doch dieser Ansatz scheitert an sich wiederholenden Aufgaben. In diesem Fall müsste eine Liste an Uhrzeiten gespeichert werden. Was einerseits die Abfrage erschwert und andererseits das Größenlimit für einzelne Dokumente überschreiten könnte. Wird in der Zukunft etwas an der sich wiederholenden Aufgabe geändert, so wären außerdem auch die bereits erledigten Instanzen der Aufgabe davon betroffen. Um all diese Probleme zu umgehen, werden erledigte Aufgaben in eine zweite Collection kopiert und `instance_date` entsprechend gesetzt. Diese Dokumente erhalten eine neue ID, speichern aber auch

ihre ursprüngliche ID mit ab. Dadurch kann bei späteren Analysen der Zusammenhang zwischen den einzelnen Instanzen und der ursprünglichen Aufgabe erkannt werden.

7.5 HTTP-Schnittstelle

Für Aufgaben wurde folgende HTTP-Schnittstelle implementiert:

HTTP Aufruf	Beschreibung
POST /todo	neue Aufgabe anlegen
PUT /todo/:id	Aufgabe bearbeiten (erledigte Instanzen sind davon nicht betroffen)
DELETE /todo/:id	Aufgabe löschen (erledigte Instanzen sind davon nicht betroffen)
POST /todo/:id/plan	Aufgabe als nächsten Schritt planen
POST /todo/:id/start/:date	Aufgabe bearbeiten (date ist Startzeit)
POST /todo/:id/done/:date	Aufgabe erledigt (Instanz wird in das Archiv kopiert, date ist Endzeit)

Tabelle 7.1: HTTP-Schnittstelle der Aufgabenverwaltung

Bis auf das anlegen neuer Aufgaben sind alle HTTP Befehle idempotent.[1] Die Befehle zum planen, beginnen und abschließen von Aufgaben, sind dabei keine klassischen CRUD-Operationen. Würde der Client die Felder der Aufgabe direkt bearbeiten und anschließend per `PUT /todo/:id` die Aufgabe aktualisieren, würde das sowohl die Implementierung des Clients als auch des Servers verkomplizieren. Die hier vorgestellte Schnittstelle beugt hingegen einer Vielzahl von Anwendungsfehlern vor.

[1] Für idempontentes Anlegen siehe Kapitel 6.2.3

8

Kontexterfassung und Planungsalgorithmus

Zusammen mit dem Planungsalgorithmus ist die Kontexterfassung die zentrale Komponente der Anwendung. Ziel der Implementierung ist, beide Komponenten modular und leicht erweiterbar zu gestalten. So kann im Rahmen dieser Arbeit ein kleiner Teil der Kontexterfassung implementiert werden und in der weiteren Entwicklung der Anwendung schrittweise erweitert werden. Gleiches gilt für den Algorithmus, der leicht um die Beachtung neuer Kontextinformationen erweitert werden kann.

Im folgenden wird zunächst die Kontexterfassung im Detail besprochen und dann auf die Implementierung des Algorithmus eingegangen.

8.1 Evaluierte Ansätze

Die Softwarerachitektur der Kontexterfassung ist vor allem durch das Context Toolkit [35] und „Modular Context Processing and Provisioning“ [36] inspiriert.

8.1.1 Context Toolkit

Zentrale Komponente des Context Toolkits sind sogenannte Kontext-Widgets[1]. Diese abstrahieren von der konkreten Erfassung des Kontextes und stellt ihre Kontextinformationen über eine einheitliche Schnittstelle bereit. Ein Beispiel hierfür wäre ein Orts-Widget,

[1] Der Name Kontext-Widget wurde in Anlehnung an GUI-Widgets gewählt, da sich ihr Verhalten ähnelt.

das den aktuellen Ort des Benutzers zur Verfügung stellt. Andere Anwendungskomponenten können dann von diesem Widget den Ort abfragen, ohne etwas über die konkret verwendeten Sensoren wissen zu müssen. Ob das Widget intern GPS-Daten verwendet, per Bluetooth die Nutzerposition erfasst, etc. ist für den Aufrufer der Schnittstelle nicht sichtbar. Dadurch kann die Implementierung eines Widgets geändert werden, ohne dass die restliche Anwendung angepasst werden muss.

Weitere Komponenten des Context Toolkits sind Interpreter. Diese interpretieren Kontextinformationen und reichern diese um weitere Informationen an. Ein Beispiel hierfür ist ein Interpreter, der Geokoordinaten in Adressen übersetzt. Dabei ist keine Erfassung des Kontextes nötig, stattdessen müssen vorhandene Informationen (z.B. mit Hilfe von Webservices wie Google Maps) nur richtig interpretiert werden. Ein Interpreter kann dabei auch mehrere Kontextinformationen in Verbindung zu einander bringen und zum Beispiel anhand des Pulses und des Ortes Rückschlüsse auf die Tätigkeit einer Person ziehen.

Schließlich gibt es noch Aggregators, die die Kontextinformationen verschiedener Widgets und Interpreter abrufen und zusammenfassen. Verwendet eine Anwendungskomponente eine Kombination aus verschiedenen Kontextinformationen, so kann sie einen entsprechenden Aggregator aufrufen und muss nicht mit jedem Widget einzeln kommunizieren.

Um Kontextinformationen an verschiedensten Orten mit Sensoren erfassenden zu können, wurde das Context Toolkit als verteiltes System konzipiert. Die Widgets erfassen dabei die Sensorwerte direkt und stellen sie über das Netzwerk zur Verfügung. Damit Anwendungskomponenten innerhalb des Netzwerks lokalisiert werden können, gibt es ein zentrales Verzeichnis (genannt Discoverer), bei dem sich die Anwendungskomponenten registrieren müssen. Die lose Kopplung zwischen den Komponenten ist einerseits ein Vorteil, da so ein flexibel erweiterbares System entsteht, andererseits führt dies aber auch zu einem komplexen verteilten System. Die Komponenten müssen dabei mit typischen Fehlern in verteilten Systemen umgehen können, wie beispielsweise schlecht synchronisierte Uhren, Konsistenzprobleme, nicht verfügbaren Anwendungskomponenten oder Netzwerkpartitionen. Die Probleme werden dadurch erschwert, dass

Anwendungskomponenten mit beliebigen anderen Komponenten interagieren können. Fehler sind deshalb schwer zu lokalisieren und zu reproduzieren.

8.1.2 Modular Context Processing and Provisioning

„Modular Context Processing and Provisioning“ [36] folgt hingegen einem etwas anderen Ansatz. Zentrale Komponente der Anwendung ist ein sogenannter Context Broker (CxB). Dieser vermittelt zwischen den Anwendungen (Context Consumer, kurz: CxC), die die Kontextinformationen verarbeiten, und den Komponenten, die die Kontextinformationen zur Verfügung stellen. Kontextinformationen stammen dabei in erster Linie von Context Providern (CxP). Diese sind mit den Widgets und Interpretern des Context Toolkits vergleichbar. Sie registrieren sich beim Context Broker und teilen ihm mit, welche Context Informationen sie zur Verfügung stellen und welche Kontextinformationen sie selbst dazu benötigen. Die Context Provider kommunizieren dann nicht direkt miteinander, sondern werden vom Broker aufgerufen. Dieser akquiriert zunächst alle benötigten Kontextinformationen von den anderen Context Providern und ruft dann den Context Provider auf. Dies hat den Vorteil, dass Context Provider nicht direkt mit den Problemen verteilter Systeme konfrontiert sind. Die Interaktion zwischen den Komponenten ist klar definiert und somit leicht zu debuggen. Und schließlich ermöglicht dieser Ansatz dem Broker auch Kontextinformationen zu cachen und so die Ausführung zu beschleunigen. Auch für den Context Consumer ist dieser Ansatz vorteilhaft, da sich nur an den Broker wenden muss und keine Kenntnisse über die Details der Kontextakquise benötigt.

Um Sensoren mit asynchroner Kommunikation zu unterstützen gibt es schließlich noch Context Sources (CxS). Diese pushen ihre Kontextinformationen selbständig an den Broker, der diese Informationen dann intern cached. Ein Beispiel hierfür wäre ein Android Smartphone, das in periodischen Abständen seine GPS-Position an den Server sendet.

Innerhalb des ganzen Systems werden Informationen im ContextML Format [37] ausgetauscht. Die Kontextinformationen werden in sogenannten Scopes zusammengefasst und sind einer bestimmten Entität zugeordnet. Alle Information innerhalb eines Scopes sind stark zusammenhängend und Änderungen innerhalb des Scopes erfolgen atomar. Ein Beispiel wäre die Entität `Person` mit dem Scope `Ort`. Innerhalb des Scopes `Ort` gibt

es dann die GPS-Koordinaten `latitude` und `longitude`. Da die GPS-Koordinaten zusammen den Ort bestimmen und sich immer gleichzeitig ändern, werden sie in einem Scope zusammengefasst.

8.2 Architektur

Bei der vorliegenden Anwendung wurden beide Ansätze kombiniert. Die Architektur ist dabei in drei Komponenten unterteilt: Context Broker, Context Provider und Context Consumer.

8.2.1 Datenmodell und Gesamtkonzept

Das Datenmodell orientiert sich an ContextML, verzichtet aber auf Entitäten. Stattdessen gibt es im aktuellen Datenmodell nur sogenannte Scopes, die im wesentlichen den Kontextinformationen der Entitäten entsprechen (siehe Kapitel 3). Ein Modell das die Scopes in Relation zu den Entitäten setzten würde, hätte mehrere Probleme. Betrachten wir hierzu ein Beispiel aus der Definition von ContextML [37]: Der Consumer erfragt dabei das Wetter für die Entität Benutzer. Gefragt ist hierbei natürlich das Wetter am Ort des Benutzers. Dieser ist dem Consumer aber nicht bekannt und soll vom Broker ermittelt werden. Eine weitere denkbare Anfrage, wäre das Wetter am Ort einer Aufgabe. In ContextML würde dann der Scope Wetter für die Entität Aufgabe erfragt werden. Das Wetter ist aber streng genommen weder Kontextinformation des Benutzers ,noch einer Aufgabe, sondern gehört zur Entität Ort (siehe Kapitel 3).

Würde man Kontextinformationen aber streng kapseln, hätte man weitere Probleme. Betrachtet man beispielsweise die Stimmung einer Person, so könnte diese vom Wetter abhängen. Das Wetter ist aber Scope innerhalb der Orts-Entität, die wiederum Teil des Benutzerkontextes ist. Würde man die Kontextinformationen also sowohl in Entitäten als auch in Scopes organisieren, wäre es für den Broker sehr komplex, alle notwendigen Kontextinformationen zu ermitteln. So müssten die Context Provider die Möglichkeit haben, Scopes einer anderen Entität zu erfragen, und der Broker müsste aus einer

Entität weitere Entitäten ableiten können. Dies würde zu einer komplexen Architektur führen und ist für den Anwendungsfall dieser Arbeit nicht zielführend.

Stattdessen wurde ein einfacheres Modell gewählt, bei dem lediglich mit Scopes gearbeitet wird. Jede Entität ist dabei selbst ein Context Provider. So ist zum Beispiel eine Aufgabe Context Provider mit dem Scope `gps`, da sie selbst über Ortsinformationen verfügt. Ein Context Consumer kann beim Broker einen bestimmten Scope erfragen, indem er den Namen des Scopes und ihm bekannte Scopes an diesen sendet. Dies lässt sich am Besten am vorherigen Beispiel verdeutlichen:

Will der Consumer das Wetter am Ort des Benutzers wissen, so sendet er den Benutzerkontext zusammen mit dem Scope `weather` an den Broker. Der Benutzerkontext besteht dabei primäre aus dem Scope `identity`, der die Benutzer-ID enthält. Über diese ID kann ein Interpreter den Ort des Benutzers ableiten und an den Wetter-Interpreter weiterleiten. Dieser nimmt die Ortsinformation entgegen und erfragt das Wetter. Will der Consumer jedoch das Wetter am Ort einer bestimmten Aufgabe wissen, so verläuft die Anfrage aus seiner Sicht analog. Er erfragt den Scope `weather` und sendet die ihm bekannten Scopes der Aufgabe mit. Da der Aufgaben-Scope aber bereits Ortsinformationen enthält, kann der Broker direkt den Wetterinterpreter aufrufen und das Wetter erfragen.

Dieses Modell ermöglicht komplexe Anfragen beim Broker, ist aber gleichzeitig einfach in der Implementierung. Statt sowohl mit Entitäten und Scopes zu arbeiten, muss jeder Context Provider nur wissen, welche Scopes bzw. Kontextinformationen für ihn relevant sind. So benötigt der Context Provider nur den Scope `gps` um das Wetter für einen bestimmten Ort abzufragen. Ob die Ortsinformationen von einem Benutzer, einer Aufgabe oder einem Termin stammen, ist für ihn nicht relevant. Im folgenden werden nun Context Broker und Provider im Detail betrachtet.

8.2.2 Context Broker

Der Broker ist Mittelsmann zwischen Context Consumer und Context Provider. Auf eine Verteilung der Anwendung wurde verzichtet. Stattdessen verfügt der Broker über eine

Lookup-Tabelle mit lokalen Context Providern. Dabei werden die Elemente über den Namen ihres Scopes referenziert. Jedes Element verfügt über eine Funktion, die den gesuchten Scope ableiten kann, und eine Liste an dafür benötigten Scopes. Ein Beispiel für diese Tabelle ist der folgende Ausschnitt aus der implementierten Anwendung:

```
{
  "utc" : {
    func: getUtc,
    requires: []
  },
  "gps" : {
    func: getGps,
    requires: ["identity"]
  },
  "timezone" : {
    func: getTimezone,
    requires: ["gps"]
  },
  "time" : {
    func: getTime,
    requires: ["utc", "timezone"]
  },
  ...
}
```

Listing 8.1: Lookup-Tabelle des Context Brokers

Die Scopes sind `utc`, `gps`, `timezone` und `time`. Der Scope `time` enthält weitere Informationen zur aktuellen Uhrzeit, wie Beispielsweise die Tageszeit (Morgens, Mittags, ...). Um diese Informationen ableiten zu können, wird die aktuelle Uhrzeit (`utc`) und die Zeitzone (`timezone`) benötigt. Die Zeitzone hängt wiederum vom Ort (`gps`) ab. Alle Kontextinformationen in der richtigen Reihenfolge abzurufen, wäre für den Context Consumer sehr komplex und aufwendig. Stattdessen übernimmt diese Aufgabe der Broker selbst.

Erfragt der Context Consumer vom Broker den Scope `time`, ist der Ablauf wie folgt (Abbildung 8.1):

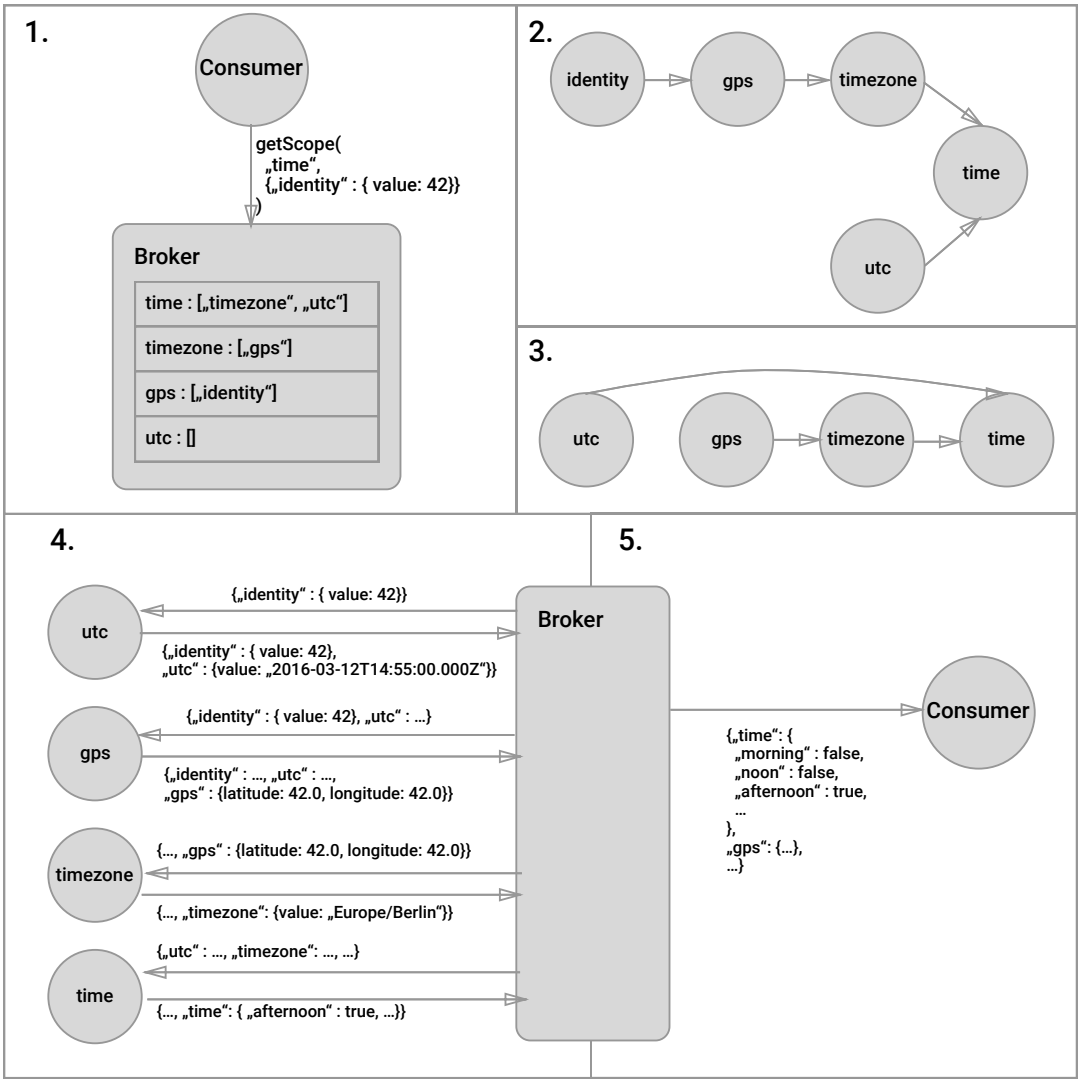

Abbildung 8.1: Abfrage des Scopes `time`

1. Der Context Consumer ruft den Broker auf und übergibt den Namen des gesuchten Scopes (`time`) und bekannte Kontextinformationen. In diesem Fall sind zwei Möglichkeiten denkbar. Einerseits kann der Consumer direkt GPS Koordinaten an den Broker übergeben (`gps`) oder er übermittelt die Identität des Benutzer (`identity`) und der Broker leitet dessen aktuelle GPS Position selbständig ab.
2. Der Broker sucht in seiner Tabelle den Scope `time` und fragt dessen Abhängigkeiten ab (`utc`, `timezone`). Vom Consumer übergebene Scopes werden dabei sofort aus den Abhängigkeiten gestrichen. Er geht dabei rekursiv vor und ermittelt für jede Abhängigkeit erneut die Abhängigkeiten. Am Ende entsteht so ein gerichteter Abhängigkeitsgraph.
3. Per topologischer Sortierung [38] wird nun die Aufrufreihenfolge für die Funktionen ermittelt.
4. Nun werden die Funktionen in der ermittelten Reihenfolge aufgerufen und so der Kontext schrittweise erweitert, bis am Ende `time` abgeleitet werden kann.
5. Schließlich wird der komplette Kontext, inklusive aller ermittelten Scopes, an den Consumer übertragen.

Alle Aufrufe erfolgen dabei asynchron. Der Consumer erhält sofort ein Promise [39], das am Ende der obigen Abfolge erfüllt oder abgelehnt wird.

8.2.3 Context Provider

Die Implementierung der Context Provider ist maßgeblich durch das Context Toolkit beeinflusst. Ein Context Provider kann dabei entweder ein Context Widget oder ein Interpreter sein, wobei der Broker nicht zwischen diesen beiden Arten unterscheiden muss. Der Unterschied liegt nur in den Abhängigkeiten. Ein Context Provider ohne Abhängigkeiten entspricht einem Context Widget, da die Kontextinformation selbst aus der Datenbank oder per Sensorwert ermittelt werden. Ein Context Provider mit Abhängigkeiten ist hingegen ein Interpreter, da neue Kontextinformationen aus den übermittelten Scopes abgeleitet werden.

Context Provider sind einfache Funktionen die lokal auf dem Server ausgeführt werden. Der Broker ruft die Funktion dann mit den ihm bekannten Scopes auf und erhält sofort ein Promise als Rückgabewert. Kontextinformationen werden dabei immer als Objekt ausgetauscht. Dabei sind die Attribute die Name der Scopes und enthalten Informationen zum jeweiligen Scope als Objekt. Betrachten wir hierzu den Aufruf des `time` Providers in Abbildung 8.1 (4. Schritt):

```
let timePromise = broker["time"].func({
  "utc" : {
    value : "2016-03-15T10:24:00.000Z"
  },
  "timezone" : {
    value : "Europe/Berlin"
  },
  "gps" : {
    latitude : 48.411880,
    longitude: 9.959027
  },
  "identity" : {
    value: "567698489f5ef7c00933b56a"
  }
})

timePromise.then( (ctx) => {
  // ctx sind sowohl alle bisherigen Scopes, als auch 'time':
  /*
  {
    "time" : {
      value : "2016-03-15T11:24:00.000+01:00",
      morning : true,
      noon : false,
      afternoon : false,
      evening : false,
      night : false
    },
    "utc" : {
      value : "2016-03-15T10:24:00.000Z"
    },
```

```
32      "timezone" : {
33        value : "Europe/Berlin"
34      },
35      "gps" : {
36        latitude : 48.411880,
37        longitude: 9.959027
38      },
39      "identity" : {
40        "value: "567698489f5ef7c00933b56a"
41      }
42    }
43    */
44  })
```

Listing 8.2: Abrufen des `time` Scopes

Context Sources werden in der Implementierung indirekt unterstützt. Um im Broker nur ein Interface zum Abrufen von Scopes implementieren zu müssen, werden alle Informationen über Context Provider zur Verfügung gestellt. Derzeit sind die Android-Smartphones mit ihren GPS-Daten die einzigen Context Sources. Sendet das Android-Smartphone GPS-Daten an den Server, so werden diese in der Datenbank zusammen mit der Benutzer-ID abgelegt. Der `gps`-Provider kann dann anhand der Benutzer-ID (`identity`-Scope) die letzte bekannte GPS-Position abfragen.

8.3 Algorithmus

Sind die Kontextinformationen erfasst, stellt sich nun die Frage, wie ein Planungsalgorithmus davon Gebrauch machen und intelligente Vorschläge berechnen kann. Hierbei wurde wieder großer Wert auf Modularität und leichte Erweiterbarkeit gelegt.

8.3.1 Ablauf

Die Berechnung der Vorschläge läuft in mehreren Schritten ab:

1. Es werden alle noch nicht erledigten Aufgaben und der nächste feste Termin aus der Datenbank geladen.
2. Der minimale Kontext des Benutzers und des nächsten Termins werden aus den Entitäten extrahiert (siehe Kapitel 8).
3. Es wird über alle Aufgaben iteriert (die Schleife kann dabei parallelisiert werden):
   a) Der minimale Kontext der Aufgabe wird extrahiert (siehe Kapitel 8).
   b) Eine Bewertungsfunktion wird aufgerufen. Dabei wird der Kontext des Benutzers, der Aufgabe und des nächsten Termins übergeben.
4. Anhand der Dringlichkeit und der Bewertung der Aufgaben, werden die drei besten Vorschläge ausgewählt.

8.3.2 Bewertungsfunktion

Die Bewertungsfunktion selbst ist wieder modular aufgebaut. Das Konzept ist dabei durch die Express-Middelware [40] inspiriert. Zur Bewertungsfunktion können dabei beliebige Module hinzugefügt werden. Jedes Modul ist eine Funktion, die den Benutzerkontext, den nächsten Termin und eine Aufgabe als Parameter erhält. Zudem erhält jede Funktion eine Callback-Funktion `next(rating)`. Ein Modul kann dann, anhand des Kontextes, eine Wertung zwischen 0 und 1 für die Aufgabe berechnen und anschließend `next()` mit dem berechneten Wert aufrufen. Die Bewertung wird dann zwischengespeichert und das nächste Modul aufgerufen. Nach der Ausführung aller Module werden die Bewertungen multipliziert und ergeben die Gesamtbewertung[2]. Gibt ein Modul eine Bewertung von 0, so ist die Gesamtbewertung automatisch Null und die Aufgabe wird nicht vorgeschlagen.

Es kann somit zwischen 3 Arten von Modulen unterschieden werden:

- **Binäre Entscheidung:** Ein Modul kann entscheiden, ob eine Aufgabe möglich ist (1) oder nicht (0). Beispiel hierfür ist ein Modul, das die Dauer der Aufgabe sowie

[2] Die Bewertung ist dabei keine klassische Wahrscheinlichkeit. Zwar wäre die bedingte Wahrscheinlichkeit $P(A|C)$ mit den Ereignissen A „Die Aufgabe wird gewählt“ und C als Kontext durchaus interessant, lässt sich aber nicht ohne weiteres berechnen, da zum Beispiel $P(C)$ nicht bekannt ist.

den Startzeitpunkt des nächsten Termins betrachtet und so entscheidet, ob die verbleibende Zeit ausreicht oder nicht.

- **Wahrscheinlichkeit:** Ein Modul kann aber auch die Wahrscheinlichkeit für die Wahl einer Aufgabe berechnen. Beispiel dafür ist ein Modul, das die Gewohnheiten des Benutzers mit dem Kontext der Aufgabe vergleicht. Dabei sollte eine Bewertung von 0 vermieden werden, um zum Beispiel dringende Aufgaben auch unabhängig von den Benutzergewohnheiten zu zulassen.
- **Kontextanreicherung:** Ein Modul kann zudem auch selbständig Kontextinformationen mittels des Brokers abrufen. Dadurch kann nach der Erweiterung der Kontexterfassung einfach ein neues Modul zur Bewertungsfunktion hinzugefügt werden, ohne dass der restliche Programmcode angepasst werden muss.

8.3.3 Auswahl der Aufgaben

Für die derzeitige Implementierung wurde ein einfaches Modell zur Beachtung der Dringlichkeit verwendet. Dazu werden die Aufgaben in 3 Gruppen eingeteilt:

1. Aufgaben die in den nächsten 24 Stunden erledigt werden müssen.
2. Aufgaben die in den nächsten 24 bis 48 Stunden zu erledigen sind.
3. Alle anderen Aufgaben.

Als Vorschläge werden zunächst die dringendsten Aufgaben aus der 1. Kategorie in Betracht gezogen. Können so bereits drei sinnvolle Vorschläge gefunden werden, werden die 2. und 3. Kategorie nicht betrachtet. Ist es jedoch nicht möglich so drei Vorschläge zu finden, werden Aufgaben der 2. Kategorie in Erwägung gezogen. Können dann nach wie vor keine drei Vorschläge gefunden werden, so werden passende Vorschläge in der dritten Kategorie gesucht. Die gefundenen Vorschläge werden anschließend gemäß ihrer Bewertung sortiert.

Durch diesen Ansatz werden dringende Aufgaben immer vorrangig behandelt. Dies hat den Vorteil, dass dringende Aufgaben zum frühest möglichen Zeitpunkt eines Tages vorgeschlagen werden und der Anwender rechtzeitig an diese erinnert wird. Generell

ist die Konzentration und Selbstdisziplin morgens höher und die Wahrscheinlichkeit, dass die Aufgaben rechtzeitig erledigt werden, steigt. Da die Vorschläge aber dennoch nach ihrer Bewertung sortiert werden, ist dem Anwender sofort klar, welche Aufgabe für den Moment am geeignetsten ist. So kann er eine dringende, aber schlecht passende, Aufgabe auch später erledigen.

Andere Modelle zur Beachtung der Dringlichkeit sind denkbar, aber erheblich komplexer. Die Probleme dabei sind ähnlich wie beim starren Tagesplan im Abschnitt 4.3. Gibt es mehrere dringende Aufgaben, so müssen alle möglichen Tagesverläufe durchgespielt werden, um zuverlässig zu erkennen, ob eine Aufgabe zwingend jetzt erledigt werden muss oder auch später am Tag möglich ist. Dabei sind mehrere Szenarien denkbar. So könnten zum Beispiel eine bestimmte Reihenfolge nötig sein, um alle Aufgaben zu erledigen. Oder es kann Fälle geben, in denen es nicht mehr möglich ist alle dringenden Aufgaben zu erledigen. Nicht nur das Erkennen dieser Situationen, sondern auch die Darstellung für den Benutzer, ist schwierig. Generell sollten diese Situationen aber selten auftreten und sind normalerweise die Folge eines ausgeprägten Aufschiebeverhaltens (Prokrastination) durch den Anwender. Es scheint daher nicht sinnvoll viel Aufwand in diesen Spezialfall zu stecken.

8.4 Horizontale Skalierbarkeit

Im Gegensatz zu den in Kapitel 8.1 vorgestellten Ansätzen handelt es sich bei der derzeitigen Implementierung des Servers nicht um ein verteiltes System[3]. Die Serveranwendung lässt sich aber dennoch problemlos horizontal skalieren. Da alle statischen Kontextinformationen in der Datenbank liegen, kann die Anwendung problemlos mehrfach gestartet werden. Ruft der Client den Server auf, wird er per Loadbalancing auf eine Serverinstanz weitergeleitet. Alle Aufrufe die etwas an den Daten ändern, speichern diese direkt in der Datenbank. Werden nur Informationen abgerufen, so werden per Datenbankaufruf immer die neusten Informationen zurückgegeben. Dynamisch gene-

[3] Da es sich um eine Client-Server-Anwendung handelt, ist das System insgesamt natürlich verteilt. Hier geht es jedoch nur um die Implementierung des Servers, insbesondere hinsichtlich des Brokers und der Context Provider.

rierte Inhalte werden derzeit nicht gecached, aber auch dies wäre problemlos möglich. So könnten die Informationen beispielsweise in einer Redis-Datenbank [41] gecached werden und zwischen den Serverinstanzen geteilt werden.

9 Ausblick

Das in dieser Arbeit vorgestellte Anwendungskonzept ermöglicht bereits weite Teile der Aufgaben- und Terminplanung zu automatisieren. Gerade im Hinblick auf die neusten Entwicklungen im Bereich Maschine Learning sind aber noch einige Erweiterungen der Anwendung möglich. Im Folgenden wird zunächst der derzeitige Entwicklungsstand der Anwendung zusammengefasst. Anschließend werden mögliche Erweiterungen des Anwendungskonzeptes kurz vorgestellt.

9.1 Entwicklungsstand der Anwendung

Zusammenfassend lässt sich sagen, dass die hier beschriebene Architektur für den vorliegenden Anwendungsfall zu guten Ergebnissen geführt hat. Das modulare Konzept aus Kapitel 8 wurde dabei im vorliegenden Prototypen vollständig umgesetzt. Die Anwendung kann daher leicht um weitere Kontextinformationen und daraus resultierende Planungsfunktionen erweitert werden. Im derzeitigen Prototyp wurde als Kontext nur die Zeit und der Ort erfasst und für die Planung verwendet. Bei der Berechnung der Vorschläge wird sowohl die zur Verfügung stehende Zeit, als auch Anfahrtszeiten an die entsprechenden Orte beachtet. Hinsichtlich der Termin- und Aufgabenverwaltung kann die Serveranwendung als vollständig betrachtet werden. Auf der Client-Seite standen die dynamischen Aufgabenvorschläge im Fokus. Die im Verlauf der Arbeit beschriebenen Ansichten (siehe Kapitel 5) und Eingabemasken (siehe Kapitel 6 und 7) wurden dabei komplett umgesetzt. Auf eine typische Kalenderansichten mit Wochen-, Monatsansicht, etc. wurde beim Prototypen aus Zeitgründen verzichtet. Für die Veröffentlichung der

Anwendung sind diese Funktionen zwar relevant, spielen aber für die in dieser Arbeit vorgestellten Konzepte keine Rolle.

9.2 Erweiterung des Anwendungskonzeptes

Für das Anwendungskonzept sind verschiedene Erweiterungen denkbar, von denen nun einige vorgestellt werden.

9.2.1 Benötigte Ressourcen

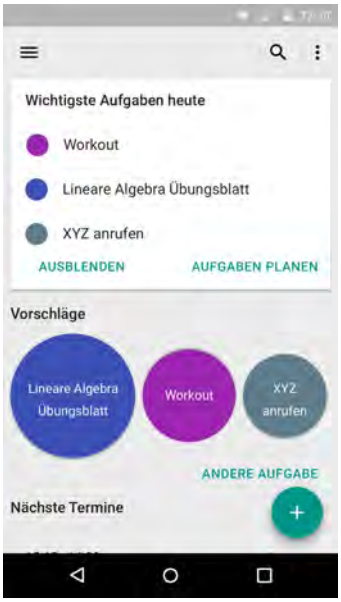

Abbildung 9.1: Tagesansicht mit einer Karte, die die dringendsten Aufgaben des Tages anzeigt

Da die Aufgabenvorschläge immer dynamisch präsentiert werden, kann es zu Situationen kommen, in denen der Anwender eine Aufgabe zwar bearbeiten möchte, die notwendigen Ressourcen dazu aber nicht zur Verfügung stehen. Wird dem Anwender zum Beispiel nach der Arbeit ein Workout im Fitnessstudio vorgeschlagen, so ist dies nur möglich, wenn er Sportkleidung bei sich hat.

Um dieses Problem zu umgehen, gibt es verschiedene Möglichkeiten. Zunächst kann davon ausgegangen werden, dass die meisten Ressourcen sich am Ort der Aufgabe befinden. So könnte die Sportkleidung zum Beispiel bereits im Schließfach des Fitnessstudios liegen. Für Ressourcen die explizit bereit gestellt werden müssen, bietet sich folgender einfacher Ansatz an:

Dem Anwender werden morgens, mittels einer Karte, die wichtigsten Aufgaben des Tages präsentieren (Abbildung 9.1). So gewinnt er einerseits einen Überblick, was ihn im Laufe des Tages erwartet, anderseits kann er gleich die notwendigen Ressourcen bereitstellen. Da der Planungsalgorithmus (siehe Kapitel 8.3) sowieso ermitteln muss, was die wichtigsten Aufgaben sind, ließe sich eine derartige Ansicht leicht in die Anwendung integrieren.

9.2.2 Pomodoro Timer

Um das fokussierte Bearbeiten der aktuellen Aufgabe zu unterstützen, wäre es denkbar für jede Aufgabe einen Pomodoro Timer zur Verfügung zu stellen. Die Aufgaben-Karte würde dabei einen Start-Button erhalten, über den der Pomodoro Timer gestartet wird (Abbildung 9.2). Nach 25 Minuten wird der Anwender informiert und kann entscheiden, ob er die Aufgabe nun vollständig erledigt hat, die weitere Bearbeitung auf später verschieben möchte, oder noch ein weiteres Pomodoro an der Aufgabe arbeiten möchte.

Abbildung 9.2: Pomodoro Karten (von links): Anwender noch nicht am richtigen Ort, Pomodoro noch nicht begonnen, Ausführung eines Pomodoro

Auf Seite des Servers müsste die Anwendung dahingehend erweitert werden, dass teilweise erledigte Aufgaben unterstützt werden und statt einer festen Start- und Endzeit, mehrere Zeiten gespeichert werden können. Auch müsste der Planungsalgorithmus erweitert werden, um Aufgaben auch dann einplanen zu können, wenn sie sich innerhalb eines Zeitslots nicht komplett erledigen lassen.

Da diese Funktion aber Einsteiger einerseits verwirren kann, andererseits wahrscheinlich auch nicht von allen Anwendern und für alle Anwendungsfälle gewünscht ist, sollte diese Funktion nur optional in den Einstellungen aktivierbar sein.

9.3 Machine Learning und Data Mining

Viele Erweiterungen des Anwendungskonzeptes wären durch die Anwendung diverser Machine Learning Techniken möglich. Die größte Schwierigkeit dabei ist, eine ausreichende Anzahl an Trainings- und Testdaten zu erhalten.

9.3.1 Intelligenten Eingabemasken

Die Eingabe neuer Termine und Aufgaben ließe sich dadurch vereinfachen, dass die Anwendung an Hand des Namens selbst die übrigen Felder ausfüllt. Dies sollte zumindest beim Ort und bei der Dauer zu guten Ergebnissen führen. Das Erkennen der gewünschten Uhrzeit oder des Fälligkeitsdatums ist hingegen schwieriger, da der Algorithmus dafür Muster in der Vergangenheit erkennen und auf die Zukunft übertragen müsste.

Schwierig ist dabei, dass die Informationen abhängig vom Benutzer sind. Die Anwendung müsste daher für einen längeren Zeitraum benutzt werden, bevor gute Vorschläge ermittelt werden können. Hier stellt sich auch die Frage, wie die Anwendung aus den Daten lernen soll. Da von einer geringen Zahl an Trainingsdaten auszugehen ist, ist ein Onlinealgorithmus eher nicht zu empfehlen. Sinnvoller ist es, gelegentlich in einem Hintergrundprozess, auf Basis aller gesammelten Daten, ein neues Modell zu berechnen. Dabei ist zu erwarten, dass einfache Ansätze (zum Beispiel mittels eines bayessches

Netzes) bessere Ergebnisse liefern als Komplexe (Neuronale Netze, etc.), da die Anzahl der Trainingsdaten auch nach längerer Nutzung der Anwendung zu gering ist.

9.3.2 Intelligente Kontexterfassung

Interessant wäre zudem, aus den erfassten primären Kontextinformationen weitere Informationen intelligent abzuleiten. So wären beispielsweise die Gewohnheit und die aktuelle Stimmung eines Anwenders interessant für den Planungsalgorithmus. Auch wie eine Aufgabe vom Anwender empfunden wird, wäre interessant. Wäre dem Planungsalgorithmus bekannt, ob eine Tätigkeit auf den Anwender entspannend wirkt oder eher als stressig empfunden wird, könnten die Vorschläge der Stimmung des Anwender angepasst werden.

Schwierig dabei ist, dass für jeden Anwender ein eigenes Modell abgeleitet werden muss und fraglich ist, wie akkurat die Ergebnisse letzten Endes sind. So ist zum Beispiel das Stressempfinden jedes Anwender unterschiedlich und auch gleiche Tätigkeiten können von unterschiedlichen Anwendern ganz unterschiedlich wahrgenommen werden. So kann der Workout im Fitnessstudio für den erfahrenen Sportler entspannend sein, für den Einsteiger ist er hingegen anstrengend und stressig.

Aus welchen Kontextinformationen zuverlässig neue abgleitet werden können und in wie weit sich dies für die weitere Planung verwendet lässt, ist daher nicht trivial und war im zeitlichen Rahmen der Arbeit nicht möglich.

Literaturverzeichnis

[1] Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine (2000)

[2] Jones, M., Bradley, J., Sakimura, N.: JSON Web Token (JWT). RFC 7519, RFC Editor (2015) http://www.rfc-editor.org/rfc/rfc7519.txt.

[3] StrongLoop: Express. http://expressjs.com (2016) Abgerufen: 04.04.2016.

[4] Heroku Inc.: Getting Started on Heroku with Node.js: Deploy the app. https://devcenter.heroku.com/articles/getting-started-with-nodejs#deploy-the-app (2016) Abgerufen: 30.03.2016.

[5] MongoDB, Inc.: Introduction to MongoDB > Documents. https://docs.mongodb.org/manual/core/document/ (2016) Abgerufen: 04.04.2016.

[6] LearnBoost Inc.: Mongoose. http://mongoosejs.com (2016) Abgerufen: 14.03.2016.

[7] LearnBoost Inc.: Mongoose - Schemas. http://mongoosejs.com/docs/guide.html (2016) Abgerufen: 14.03.2016.

[8] ISO: Date and time format - ISO 8601. ISO 8601:2004, International Organization for Standardization, Geneva, Switzerland (2004)

[9] Joda.org: Joda-Time. http://www.joda.org/joda-time/ (2016) Abgerufen: 14.03.2016.

[10] Moment.js: Moment.js. http://momentjs.com (2016) Abgerufen: 30.03.2016.

[11] Gartner: Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015. http://www.gartner.com/newsroom/id/3215217 (2016) Abgerufen: 30.03.2016.

[12] Google: Material Design. https://www.google.com/design/spec/material-design/introduction.html (2016) Abgerufen: 14.03.2016.

[13] Schickler, M., Reichert, M., Pryss, R., Schobel, J., Schlee, W., Langguth, B.: Entwicklung mobiler Apps: Konzepte, Anwendungsbausteine und Werkzeuge im Business und E-Health. Springer-Verlag (2015)

[14] Geiger, P., Schickler, M., Pryss, R., Schobel, J., Reichert, M.: Location-based mobile augmented reality applications: Challenges, examples, lessons learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 383–394

[15] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education (1994)

[16] Schickler, M., Pryss, R., Schobel, J., Reichert, M.: An engine enabling location-based mobile augmented reality applications. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 363–378

[17] Google Inc.: Transmitting Network Data Using Volley. `http://developer.android.com/training/volley/index.html` (2016) Abgerufen: 14.03.2016.

[18] Google Inc.: gson. `https://github.com/google/gson` (2016) Abgerufen: 14.03.2016.

[19] Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a Better Understanding of Context and Context-Awareness. In: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing. HUC '99, London, UK, UK, Springer-Verlag (1999) 304–307

[20] McCarthy, D.D., Seidelmann, K.P.: Time: From Earth Rotation to Atomic Physics. John Wiley & Sons (2009)

[21] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using vital sensors in mobile healthcare business applications: Challenges, examples, lessons learned. In: 9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps. (2013) 509–518

[22] Allen, D., Fallows, J.: Getting Things Done: The Art of Stress-Free Productivity. Penguin Publishing Group (2015)

[23] Babauta, L.: Zen To Done. Leo Babauta (2008)

[24] Nöteberg, S.: Die Pomodoro-Technik in der Praxis: der einfache Weg, mehr in kürzerer Zeit zu erledigen. dpunkt-Verlag (2011)

[25] Csikszentmihaly, M.: Flow and the Foundations of Positive Psychology. Springer Netherlands (2014)

[26] Iyengar, S.S., Lepper, M.R.: When choice is demotivating: Can one desire too much of a good thing? Journal of personality and social psychology **79** (2000) 995

[27] Vohs, K.D., Baumeister, R.F., Twenge, J.M., Schmeichel, B.J., Tice, D.M., Crocker, J.: Decision fatigue exhausts self-regulatory resources—but so does accommodating to unchosen alternatives. Manuscript submitted for publication (2005)

[28] Google: Components - Cards. `https://www.google.com/design/spec/components/cards.html` (2016) Abgerufen: 14.03.2016.

[29] Google: Style - Color. `https://www.google.com/design/spec/style/color.html` (2016) Abgerufen: 14.03.2016.

[30] Google Inc.: Creating and Monitoring Geofences. `http://developer.android.com/training/location/geofencing.html` (2016) Abgerufen: 14.03.2016.

[31] Desruisseaux, B.: Internet Calendaring and Scheduling Core Object Specification (iCalendar). RFC 5545, RFC Editor (2009) `http://www.rfc-editor.org/rfc/rfc5545.txt`.

[32] Wikipedia: ICalendar — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=ICalendar&oldid=707478321` (2016) Abgerufen: 28.03.2016.

[33] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-driven data collection with smart mobile devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[34] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards process-driven mobile data collection applications: Requirements, challenges, lessons learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[35] Dey, A.K., Abowd, G.D., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. Hum.-Comput. Interact. **16** (2001) 97–166

[36] Knappmeyer, M., Tönjes, R., Kiani, S.L., Baker, N.: Modular Context Processing and Provisioning: Prototype Experiences. In: Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems. CASEMANS '10, New York, NY, USA, ACM (2010) 8:53–8:58

[37] Knappmeyer, M., Kiani, S.L., Frà, C., Moltchanov, B., Baker, N.: ContextML: A light-weight context representation and context management schema. In: Wireless Pervasive Computing (ISWPC), 2010 5th IEEE International Symposium on. (2010) 367–372

[38] Schöning, U.: Algorithmik. Springer Spektrum (2001)

[39] bluebird: Why Promises? `http://bluebirdjs.com/docs/why-promises.html` (2016) Abgerufen: 09.04.2016.

[40] Express: Writing Middleware. `http://expressjs.com/en/guide/writing-middleware.html` (2016) Abgerufen: 14.03.2016.

[41] Redis. `http://redis.io` (2016) Abgerufen: 09.04.2016.

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Simon Löw Matrikelnummer: 752634

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Simon Löw