Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3093

Ein TaskManager für PerFlows

Thorsten Höger

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. Tobias Unger Dipl.-Inf. M.Sc.(USA) Stephan Urbanski
begonnen am:	1. Oktober 2010
beendet am:	2. April 2011
CR-Klassifikation:	E.1, G.3, H.4.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I.

Grundlagen

1. Einleitung

1.1. Motivation

Die Allgegenwärtigkeit von Computern und ähnlichen Geräten hat in letzter Zeit ungeahnte Ausmaße erreicht. Dabei haben sich sowohl die Leistungsfähigkeit als auch die Verbreitung dieser Geräte vervielfacht. Vor allem im Bereich der Smartphones sind die Rechenleistung und das Speicherangebot massiv gestiegen. Auch eine Internetverbindung kann mittlerweile als dauerhaft gegeben betrachtet werden. Dies liegt einerseits am stark verbesserten Netzausbau aber andererseits auch an der deutlichen Senkung der Preise für Endbenutzer. Als weiterer Punkt sind mittlerweile sehr viele verschiedene Sensoren, wie Mikrofone, Kameras, Beschleunigungssensoren und GPS-Empfänger in nahezu jedem Smartphone integriert. Auf Basis dieser neuen Rahmenbedingungen ergeben sich völlig neue Anwendungsszenarien für mobile Endgeräte. Für Smartphones existieren einige Anwendungen, die Internet und Sensorik verbinden. Als Beispiele sind hier Google Maps und Wikihood genannt. Google Maps bietet dem Benutzer eine Karte seiner aktuellen Umgebung auf Basis der Lokationsinformationen seines Endgerätes. Wikihood verknüpft die Lokationsdaten noch weiter indem es dem Benutzer Wikipediaartikel zu Objekten und Personen mit Bezug zum aktuellen Umfeld anzeigt. Bei falscher Anwendung und unsachgemäßem Umgang durch den Benutzer birgt diese Verknüpfung von Daten auch große Risiken, was eine Aufklärung des Benutzers essentiell macht. Die genannten Anwendungen beschränken sich dabei allerdings auf die Sensordaten eines Endgerätes und können so nur sehr lokal begrenzte Daten erfassen und verwenden. Das Konzept der in Abschnitt 2.2 vorgestellten PerFlows erweitert dieses Modell. PerFlows bieten dem Benutzer eine Grundlage zur Modellierung von persönlichen Abläufen zur Abbildung seiner Alltagsaufgaben. Die Infrastruktur sorgt dann dafür, dass die Aufgaben und Anwendungen entsprechend dem aktuellen Kontext ausgeführt werden können. Das heißt die Infrastruktur orchestriert die verfügbaren Geräte um die Aufgabe auszuführen. Durch diese Automatisierung sollten sich signifikante Zeitersparnisse und ein vermindertes Stresslevel des Anwenders erreichen lassen, was wiederum die Produktivität und Zufriedenheit des Einzelnen erhöht. Als Beispiel betrachten wir Sal die einen PerFlow für das Einkaufen erstellt hat. Wir nehmen an, Sal besitzt ein Fahrzeug mit Navigationssystem und eine digitale Einkaufsliste. Die aktuellen Angebote der Supermärkte sind im Internet abrufbar. Die Ausführungsumgebung der PerFlows würde nun automatisch einen Supermarkt vorschlagen und auf Wunsch das Navigationssystem im Fahrzeug programmieren. Hierbei werden zwei Dinge deutlich: Zum einen wird die Verbindung der einzelnen Komponenten,

Dienste und Geräte automatisch von der Infrastruktur übernommen, zum anderen verfügt der Anwender jederzeit über die Kontrolle, d.h. die Infrastruktur führt keine Aktionen ohne Freigabe durch den Benutzer aus. Ausnahmen hiervon sind nur reversible Vorbereitungsaktivitäten, die bei Stornierung keinen Nachteil für den Benutzer mit sich bringen. Im Beispiel muss Sal die Auswahl des Supermarkts freigeben, um die Programmierung des Navigationssystems einzuleiten. Der Grund hierfür ist die Annahme, dass sich Benutzer im Alltag ungern von elektronischen Helfern bevormunden lassen und Persönlichkeitsrechte wie Selbstbestimmung wahrnehmen wollen. PerFlows sollen lediglich eine Hilfe anbieten, die jederzeit abgelehnt werden kann. Eine Herausforderung im Beispielszenario ist allerdings die Auswahl von momentan günstigen Aufgaben. So ist beispielsweise der Vorschlag einkaufen zu gehen wenig sinnvoll, wenn der Benutzer sich momentan in einer Besprechung befindet, selbst wenn die Aufgabe eine Deadline in naher Zukunft hat. Daraus wird offensichtlich, dass klassische Planungssysteme nicht für PerFlows in Frage kommen können. Erstens verwenden die wenigsten Systeme aktuelle Sensordaten aus dem Kontext des Benutzers und können somit wesentliche Kriterien nicht betrachten, zweitens arbeiten die Systeme in der Regel nicht dynamisch und bewerten die Informationen nicht ständig neu, wie es bei kontextsensitiven Anwendungen zwingend notwendig ist. In der folgenden Arbeit soll nun eine Priorisierung von persönlichen Aufgaben an Hand verschiedener Kriterien entwickelt werden. Diese Kriterien stammen zum einen aus den Eigenschaften der Aufgabe, wie zum Beispiel Deadlines oder Ausführungsorte, zum anderen sollen aber auch Kontextinformationen des Benutzers wie seine aktuelle Position mit in die Berechnung einbezogen werden. In der Arbeit werden vorhandene Planungssysteme auf ihre Einsatzfähigkeit für PerFlows untersucht und auf Basis der daraus gewonnen Ideen und Mängellisten ein Konzept für die Umsetzung eines TaskManagers für PerFlows erstellt. Ein wesentlicher Bestandteil ist die Umsetzung des TaskManagers auf realen Plattformen, um in zukünftigen Arbeiten die Evaluierung von Kriterien in einem realen Szenario zu ermöglichen. Die gewonnenen Erkenntnisse über eine sinnvolle Architektur des Gesamtsystems und die Implementierung für unterschiedliche mobile Plattformen vervollständigen diese Arbeit. Die Datenschutzaspekte wurden hier bewusst ausgelassen, was aber nur der Abgrenzung der Arbeit dient, aber keine Schmälerung der Relevanz dieser Themen bedeutet. Vor einem Einsatz der hier entwickelten Lösung müssen diese Fragestellungen unbedingt bearbeitet werden.

1.2. Ziele der Arbeit

Das Ziel dieser Arbeit gliedert sich in mehrere Bereiche. Zum einen soll ein Datenmodell für die Verwaltung von Aufgaben erstellt werden, das es ermöglicht diese mit Kontextinformationen (z.B. Lokation) anzureichern und sie in PerFlows nutzen zu können. Des Weiteren soll ein Algorithmus entwickelt werden, der diese Aufgaben anhand diverser Faktoren, wie Lokationsinformationen und Zeitvorgaben, dynamisch priorisiert und dem Anwender eine Empfehlung für die Abarbeitungsreihenfolge seiner Aufgaben gibt. Abschließend wird eine

konkrete Implementierung für diverse Endgeräte des Benutzers geplant und ausgeführt. Hierbei werden auf Architektur- und Protokollüberlegungen mit einfließen.

1.3. Aufbau der Arbeit

Die folgende Arbeit gliedert sich in zwölf Abschnitte. Zu Beginn werden die Grundlagen, die für diese Arbeit notwendig sind, erläutert. Dabei werden die Begriffe Aufgabe und Task synonym verwendet. Die Grundlagen beinhalten bestehende Konzepte zur Taskverwaltung, verwandte Arbeiten und das Konzept der PerFlows. Im Weiteren wird dann der Taskbegriff für diese Arbeit definiert und von den bestehenden abgegrenzt. Im Kapitel4 werden verschieden Methoden zur Priorisierung vorgestellt und anschließend der erarbeitete Algorithmus näher erläutert. Das Kapitel 6 betrachtet die Architektur des Gesamtsystems. Hier werden verschiedene Ansätze geprüft und dann die einzelnen Komponenten der gewählten Infrastruktur vorgestellt. Die Implementierung der Serverkomponente schließt sich im Kapitel 7 an. Die Festlegung der Schnittstelle wird danach getroffen. Die Kapitel 9ff widmen sich der Implementierung von Desktop- und Mobileclients, sowie den Erweiterungsmöglichkeiten. Eine Evaluation der entwickelten Komponenten mit einer anschließenden Zusammenfassung der Arbeit und ein Ausblick auf zukünftige Anwendungen im Bereich der PerFlows und darüber hinaus schließen die Arbeit ab.

2. Grundlagen und verwandte Arbeiten

2.1. Motivation

In diesem Kapitel sollen die notwendigen Grundlagen für die vorliegende Arbeit erläutert werden. Des Weiteren werden bestehende Konzepte zur Verwaltung von Aufgaben aus dem wissenschaftlichen und kommerziellen Bereich vorgestellt und die Unterschiede und Gemeinsamkeiten zum hier vorgestellten TaskManager beschrieben. Das Ziel ist es, die hinter dieser Arbeit stehende Idee zu erklären und alle notwendigen Voraussetzungen für die folgenden Kapitel zu schaffen.

2.2. Personal Flows

In seiner wegweisenden und visionären Arbeit im Bereich des *ubiquitous computing "The Computer for the 21st Century"* [Wei91] beschreibt Mark Weiser eine Welt, in der alle Objekte aus dem täglichen Bedarf mit intelligenten Funktionen ausgestattet sind und uns bei unserer Arbeit unterstützen. Die Unterstützung reicht dabei von digitalen Wänden und Projektionstafeln über mobile Endgeräte bis hin zu digitalen Notizzetteln. Zur Zeit der Veröffentlichung waren all diese Ideen noch nicht technisch umsetzbar und reine "Science-Fiction". Durch den dramatischen Anstieg der Anzahl an Computern und computerähnlichen Geräte und der Ausdehnung der Anwendungsbereiche in alle Lebenssituationen kann aus dieser Fiktion aber heute Realität werden. Nahezu jeder hat heute ein Handy und der Anteil an Smartphones wächst stark an. Auch durch die Miniaturisierung von Computern auf Subnotebooks oder der Hype um Tablet-PCs zeigen, dass der Computer als Alltagsgegenstand längst Standard ist.

Ein Vorschlag zur Umsetzung dieser Vision mit heutigen Mitteln findet sich im Paper *"PerFlows for the Computers of the 21st Century"* aus dem Jahre 2009 [UHW+09]. Die Autoren beschreiben darin die Verwendung von aus dem Prozessmanagement abgeleiteten Methoden zur Umsetzung persönlicher Abläufe im täglichen Leben. Dazu führen sie das Konzept der *"Personal Workflows"* oder *PerFlows* ein. Diese Arbeit basiert auf den *Sentient Processes* aus dem Jahre 2006 [UBR06]. Mit Hilfe dieser Flows lassen sich Aufgaben und Ziele einer Person samt den dazu notwendigen Schritten als Prozess modellieren und in einer geeigneten Laufzeitumgebung ausführen. Der für die Ausführung notwendige Benutzerkontext beinhaltet als

eines der wichtigsten Merkmale die Position des Benutzers. Dazu liefert einer der Autoren das Positionierungskonzept des *SmartGPS* [Hub08].

Die Ausführung der PerFlows geschieht dann auf einem verteilten System, bei dem ein mobiles Endgerät des Benutzers die Rolle des Prozesskoordinators übernimmt. Dieser verwaltet die zu erledigenden Prozessschritte der aktiven PerFlows und führt diese bei Aktivierung auf geeigneten Geräten in der aktuellen Umgebung des Benutzers aus. Im Gegensatz zu den genannten Arbeiten kann mittlerweile sogar von einer dauerhaften Internetverbindung der beteiligten Geräte ausgegangen werden, was die Koordination und Verwaltung der Daten und Aufgaben wesentlich erleichtert.

Da ein Großteil der Aufgaben im persönlichen Ablauf von einem Menschen ausgeführt werden und nur einige unterstützenden Aufgaben automatisiert ablaufen, ist die Verwaltung von persönlichen Aufgaben eine Kernkomponente einer PerFlow-Umgebung. Das Prozesssystem darf im Gegensatz zum geschäftlichen Einsatz dem Benutzer keine Aufgaben und Prozessabläufe diktieren, sondern soll als Empfehlung verstanden werden und Abweichungen des Benutzers vom aktuellen Prozess tolerieren und entsprechende Maßnahmen ergreifen. Deshalb dürfen unumkehrbare Prozessschritte nur nach Genehmigung durch den Benutzer ausgeführt werden und nur reversible Aktionen vollautomatisch aktiviert werden.

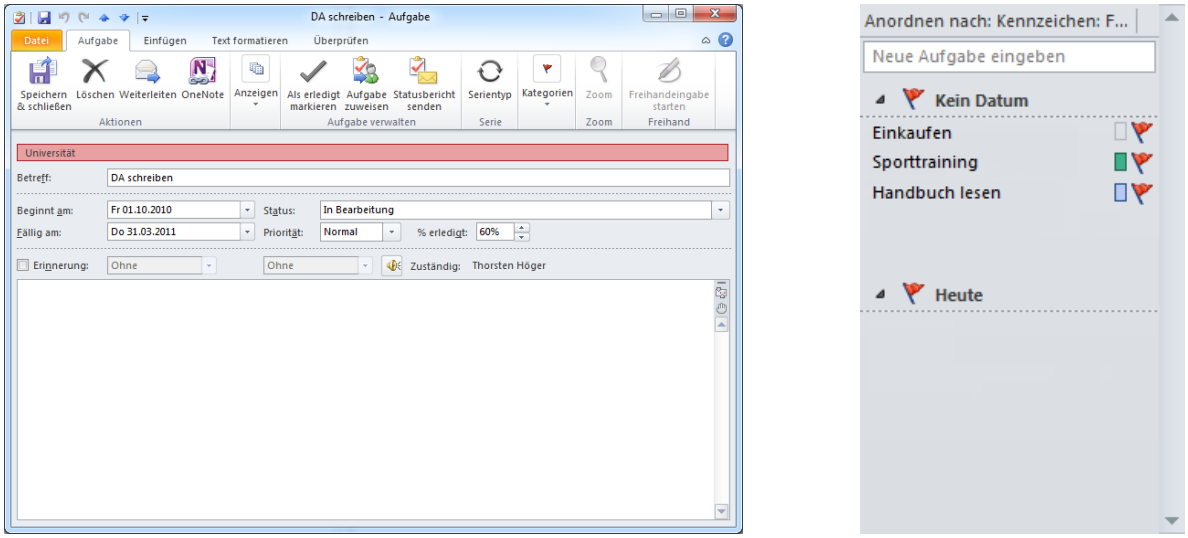
Nach der Vorstellung des Konzepts und der Ausführungsumgebung für PerFlows in [UHW+09] soll nun in dieser Arbeit der notwendige TaskManager für persönliche Aufgaben konzipiert und implementiert werden. Dies berücksichtigt auch die Priorisierung der Aufgaben nach Wichtigkeit und Gelegenheit, sowie die Statusverwaltung der Aufgaben zur Kommunikation mit dem Prozesskoordinator. Im weiteren Verlauf sind mit den Begriffen Flow, Workflow oder Prozess, wenn nicht anders angegeben, immer PerFlows entsprechend des eben vorgestellten Konzepts gemeint.

2.3. Taskkonzepte

Die Verwaltung von Aufgaben im persönlichen oder geschäftlichen Umfeld ist kein neues Thema. Es existieren bereits diverse Lösungen für Taskmanagementsysteme. Dies beinhaltet reine Aufgabenlistensysteme wie zum Beispiel PIM-Lösungen und komplexe Verwaltungsprogramme für die Verwendung in Geschäftsprozessen. In diesem Abschnitt werden nun einige dieser Lösungen vorgestellt und anschließend geklärt, warum eine weitere TaskManagerentwicklung sinnvoll ist. Bei den vorgestellten Lösungen handelt es sich sowohl um theoretische Konzepte als auch um reale Implementierungen, die auf dem Markt verfügbar sind.

2.3.1. PIM Tasks

Unter *Personal Information Manager* versteht man Systeme, die persönliche Daten wie Kontakte, Aufgaben und Notizen verwalten. Diese Art der Aufgabenverwaltung ist sehr weit verbreitet und liefert die Grundlage einer digitalen Erfassung seiner ToDos. Das bekannteste Beispiel in diesem Bereich dürfte Microsoft Outlook sein. Es bietet dem Benutzer die Möglichkeit Aufgaben anzulegen, zu kategorisieren und ihren Fortschritt zu dokumentieren. Screenshots der Anwendung finden sich in [Abb: 2.1].



(a) Einzelner Task

(b) Komplette Worklist

Abbildung 2.1.: Tasks Microsoft Outlook

Weitere bekannte Beispiele für PIM-Lösungen sind Apple iCal[1] und Mozilla Sunbird[2]. Auch jedes Smartphone bringt im Betriebssystem eine Verwaltung von Aufgaben und Notizen mit, die sich mit Desktoplösungen abgleichen lassen. Diese Systeme sind allerdings nur für die Benutzung durch Menschen konzipiert und lassen sich schlecht automatisiert in Software verwenden. Eine Integration in ein Workflowsystem ist mit diesen Produkten nicht zielführend. Auch bieten diese Systeme sehr eingeschränkte Möglichkeiten, die Aufgaben automatisch zu priorisieren. Stattdessen setzen sie auf eine vorgegebene und statische Priorisierung durch den Anwender. Aus diesen Gründen ist eine Verwendung dieser Komponenten in PerFlows nicht durchführbar, allerdings sollte eine Möglichkeit geschaffen werden, die Aufgaben eines PIM-Systems im TaskManager zu integrieren. Dies ist aber nicht Bestandteil dieser

[1] http://www.apple.com/de/macosx/what-is-macosx/mail-ical-contacts-apps.html
[2] http://www.sunbird-kalender.de

Arbeit. Ein weiterer Nachteil von PIM-Systemen ist nach [OTM03, 2.2], dass es "für eine Verbesserung der persönlichen Produktivität wichtig ist, Aufgaben vorausschauend und unter verschiedenen Blickwinkeln zu evaluieren" und konventionelle Aufgabenplaner diese Anforderung nicht umsetzen können.

2.3.2. WS-HumanTask

Eine weitere Arbeit, die sich mit der Verwaltung von Task beschäftigt ist das Konzept der WS-HumanTasks[AAD+07b]. Es bietet eine Möglichkeit von Menschen auszuführende Aufgaben in Geschäftsprozessen zu verwenden. Dazu wird in einem BPEL-Prozess[3] mit der Erweiterung BPEL4People[AAD+07a] eine *peopleActivity* verwendet, die dann den TaskManager bedient. Hierbei wird sehr stark auf die Interaktion von Taskverwaltung und Prozessumgebung wert gelegt, daher zielt diese Lösung in eine vergleichbare Richtung wie die Verknüpfung von PerFlows mit Personal Tasks. Allerdings ist auf Grund der vielfältigen Einsatzmöglichkeiten im Geschäftsumfeld und der hohen Prozessanforderungen an Geschäftsprozesse die Verwendung von BPEL und Ws-HumanTask zu komplex für den Anwendungsfall der Prozessverwaltung im persönlichen Bereich. Ein wichtiger Unterschied von Geschäftsprozessen zu persönlichen Workflows ist außerdem die gewünschte Prozesstreue im geschäftlichen Umfeld und die flexible, eher als Vorschlag zu betrachtende Ausführung im privaten Umfeld. Auch die Mächtigkeit von WS-HumanTask im Bereich der automatischen Zuweisung von Personen zu Tasks (*potentialOwners*) und die Integration der, bei Bedarf auch multilingualen, Darstellungsbeschreibung von Tasks innerhalb der Definition führen zu dem Schluss, dass die Verwendung von WS-HumanTask im Bereich der PerFlows überdimensioniert und von zu hoher Komplexität ist. Trotzdem beeinflussen die Überlegungen hinter WS-HumanTask die Entwicklung des Konzepts der persönlichen Aufgaben.

2.3.3. Andere Arbeiten

Bereits 1992 haben Toyohide Watanabe und Teruo Fukumura in ihrer Arbeit *A Scheduler of Daily Personal Tasks on the Basis of the Object-oriented Model* [WF92] das Problem der Ablaufplanung von persönlichen Aufgaben als wichtig erkannt und eine objekt-orientierten Lösungsansatz präsentiert. Sie gehen dabei davon aus, dass jede Person einen eigenen "Stundenplan" mit ihren Aufgaben hat und zusätzlich Mitglied in *n* Umgebungen ist, die wiederum Aufgabenpläne beinhalten. Diese Umgebungen können baumartig geschachtelt sein und somit dem einzelnen Benutzer mehrere parallele Zeitpläne bieten. Der Benutzer, von den Autoren *Agent* genannt, muss nun nach dem in [WF92, Fig. 7] beschriebenen Algorithmus diese Zeitpläne verschmelzen um einen einzelnen, aktuellen Zeitplan zu erhalten, der dann

[3] Business Process Execution Language[Org07]

ausgeführt werden kann. Allerdings gibt es auch hier keine Garantie der Ausführbarkeit durch inkompatible Zeitpläne.

Auch im Artikel von Ohmukai, Takeda und Miki [OTM03] liegt das Hauptaugenmerk auf der Erstellung eines Ablaufplans für personenbezogene Aufgaben. Die Autoren wählen hier aber einen anderen Ansatz zur Aufstellung der Pläne. Als Ausgangssituation für den vorgeschlagenen Lösungsweg steht dem System eine Liste von zu erledigenden Aufgaben zur Verfügung, die dann in eine gemeinsame Liste gefüllt werden. Danach versucht das System entstandene Konflikte bei der Ablaufplanung zu lösen, indem es einzelne Aufgaben umplant. Dabei werden die verschiedenen Pläne getrennt erstellt und dann einzeln bewertet, um den idealen Ablaufplan zu finden. Dieser dreistufige Ablauf ist dem menschlichen Entscheidungsprozess nachempfunden, nach dem erst die nötigen Informationen gesammelt werden, danach die möglichen Optionen bestimmt werden und abschließend die beste Lösung gewählt wird.

Ein anderes Ziel verfolgt das Paper *Supporting Flexible Processes Through Recommendations Based on History* [SWDA08]. Die Autoren stellen eine Methode zur Empfehlung von Aufgaben basierend auf Zielfunktionen und Prozesslogs. Dabei definiert ein Benutzer welches Ziel er in einem Prozess erreichen möchte. Zusätzlich liefert er ein Log seiner bisherigen Prozessschritte. Mit Hilfe dieser Daten sucht das System dann für diesen Fall relevante Ablauflisten vergangener Ausführungen desselben Prozesses und versucht anhand der Zielfunktion eine optimale Empfehlung des nächsten Prozessschrittes zu geben. Der grundsätzliche Unterschied zum TaskManager in dieser Arbeit ist die Tatsache, dass der Benutzer seine nächsten Aufgaben nicht kennt sondern erst vom System vorgeschlagen bekommt wogegen der TaskManager bestehende Aufgabeliste anhand der Kontextinformationen des Benutzers priorisiert.

2.4. Zusammenfassung

All diese Konzepte vernachlässigen den aktuellen Aufenthaltsort des Benutzers. Dies geschieht entweder aus mangelnden Informationen über die Lokation oder aus der Annahme einer geschäftlichen Nutzung in einer definierten Umgebung ohne Bezug zu anderen Bereichen des Benutzers. Da aber die Priorisierung von kontextbezogenen Aufgaben im persönlichen Umfeld unabdingbar für den Einsatz in PerFlows ist, kann keines dieser Systeme alle Anforderungen erfüllen. Deshalb wird im Folgenden ein neues Konzept zur Verwaltung von Aufgaben erarbeitet.

Teil II.

Theoretische Konzepte

3. Personal Task

3.1. Motivation

Um den Priorisierungsalgorithmus zu definieren und zu implementieren ist ein Taskmodell notwendig, das die entsprechenden Attribute beinhaltet. Zur Hilfestellung bei der Definition wird zunächst ein Beispielszenario skizziert und dessen Aufgaben abschließend mit dem definierten Modell beschrieben.

3.2. Szenario

Das folgende Szenario dient der Verdeutlichung der Konzepte und Algorithmen in dieser Arbeit. Es bezieht sich dabei auf die Ausgangssituation von Sal aus Weisers Beispiel[Wei91]. Zusätzlich zu ihrem beschriebenen Tagesablauf sollen nun drei Aufgaben gezeigt werden, die sie am Wochenende durchzuführen hat. Als erste Aufgabe will Sal den Familieneinkauf durchführen. Dazu muss sie in den 11 *km* entfernten Supermarkt fahren. Sie startet nun den Einkaufen-Flow, der die Einkaufsliste aus dem digitalen Notizbuch entnimmt und den TaskManager anweist eine Aufgabe einzuplanen. Weiter müssen ihre Kinder um 15:00 Uhr zum Sport im 10 *km* entfernten Sportstadion sein. Diese Aufgabe hat eine harte Zeitgrenze, da Verspätungen einer Nichterfüllung gleichkommen. Als letzte Aufgabe will Sal noch das Handbuch ihres Garagenöffners lesen, das ihr vom Hersteller zugesendet wurde. Diese Aufgabe stuft sie als priorisiert ein, weist ihr aber keine Deadline zu. Dies führt zu einer Unterordnung bei Aufgaben mit nahender Deadline.

3.3. Anforderungen

Um die Eigenschaften der Tasks entsprechend des gezeigten Szenarios einzuordnen, muss das Taskmodell einerseits grundlegende Eigenschaften ausweisen, wie eine Bezeichnung, Start- und Endzeiten sowie Laufzeitinformationen. Andererseits sind aber auch Kontextinformationen wie der Ausführungsort einer Aufgabe wichtig, um bei der Priorisierung eine Zuordnung des Umfelds des Benutzers zu einzelnen Aufgaben zu ermöglichen. Ein weiterer Aspekt ist die Trennung zwischen Taskmodell und Taskinstanz. Dabei wird eine Aufgabe

zuerst als Entwurf mit allen Attributen angelegt und dann bei Bedarf eine konkrete Instanz erzeugt, die dann zur Priorisierung gelangt. Der Zugriff auf Kontextdaten aus dem PerFlow soll ebenfalls möglich sein. Hierzu können entweder Daten aus dem Kontext an den Task angehängt werden oder die Taskanzeige kann auf die Ausführungsumgebung des PerFlows zugreifen und die benötigten Daten direkt referenzieren. Eine nicht-funktionale Anforderung an das Taskmodell ist das Ziel einer niedrigen Komplexität, um dem Einsatzgebiet im persönlichen Umfeld gerecht zu werden. Deshalb gelten hier nicht dieselben Anforderungen wie beispielsweise bei WS-HumanTask [AAD+07b].

3.4. Definition

Der hier verwendete Taskbegriff beinhaltet eine geringere Komplexität als Tasks im gewerblichen Bereich oder in Geschäftsprozessen. Eine Erweiterung zu bisherigen Taskdefinitionen aus PIM-Systemen ist die Verwendung der Lokation in Koordinatenform zur Vereinfachung der digitalen Verarbeitung statt der üblichen textbasierten Ortsangabe. Um dem Benutzer die Verwaltung zu erleichtern gibt es keine verschiedenen Aufgabentypen, sondern nur allgemeine auszuführende Aufgaben. Die Erzeugung von Aufgabenentwürfen wird aus dem TaskManager ausgelagert und kann entweder bei der Erfassung verwaltet werden oder direkt in der PerFlow-Engine vorgehalten werden. Der TaskManager erhält dann nur die Taskinstanzen und unterscheidet somit nicht zwischen Modell und Instanz. Jede Aufgabe ist somit eine einmalig auszuführende Einheit. Kontextdaten werden bei Bedarf direkt aus der Engine geladen oder im Beschreibungsfeld hinterlegt. In einer weiteren Entwicklung könnte der Task allerdings um Ein- und Ausgabedaten erweitert werden, um Kontextdaten aus den PerFlows zu verwenden und diese in einem Anzeigesystem darzustellen ohne Kenntnis von der Ausführungsumgebung zu haben.

Zusammenfassend gilt für PersonalTasks nun:
Personal Tasks sind in sich abgeschlossene, aber trotzdem wahlweise pausierbare, Aufgabenpakete, die von Privatpersonen im persönlichen Umfeld zu bearbeiten sind und mit Kontextinformationen angereichert wurden.

3.5. Eigenschaften

3.5.1. Felder

Die Felder der hier verwendeten Taskdefinition sind in der folgenden Liste aufgezählt. Eine Beschreibung der einzelnen Felder findet sich im Anhang A.

- id
- name
- description
- state
- startTime
- dueTime
- manualPriority
- location
- pausable
- runtime
- estimatedRuntime
- escalation

3.5.2. Status

Der Status einer Aufgabe ist eine der Kerneigenschaften zur Verwaltung von Tasks und im Feld *state* abgelegt. Die möglichen Werte sind in [Tabelle: 3.1] beschrieben. Die möglichen Statusübergängen findet man in [Abb: 3.1].

Zustand	Bedeutung
NotYetStarted	Die Aufgabe wurde noch nicht gestartet und wartet auf Aktivierung
Running	Die Aufgabe ist momentan in Bearbeitung
Waiting	Die Aufgabe ist unterbrochen, da sie auf externe Ergebnisse wartet
Paused	Die Aufgabe ist pausiert und kann fortgesetzt werden.
Reopened	Eine bereits geschlossene Aufgabe wurde wieder geöffnet und kann gestartet werden
Closed	Die Aufgabe ist als erledigt markiert
Deleted	Die Aufgabe wurde gelöscht

Tabelle 3.1.: Aufgabenzustände

3.6. Taskattribute im Szenario

Entsprechend dieser Definition und den Angaben aus dem Szenario ergeben sich die Tasks aus der Liste in [Tabelle: 3.2]. Sie dienen im weiteren Verlauf als Ausgangsbasis für alle beispielhaften Berechnungen und Beschreibungen.

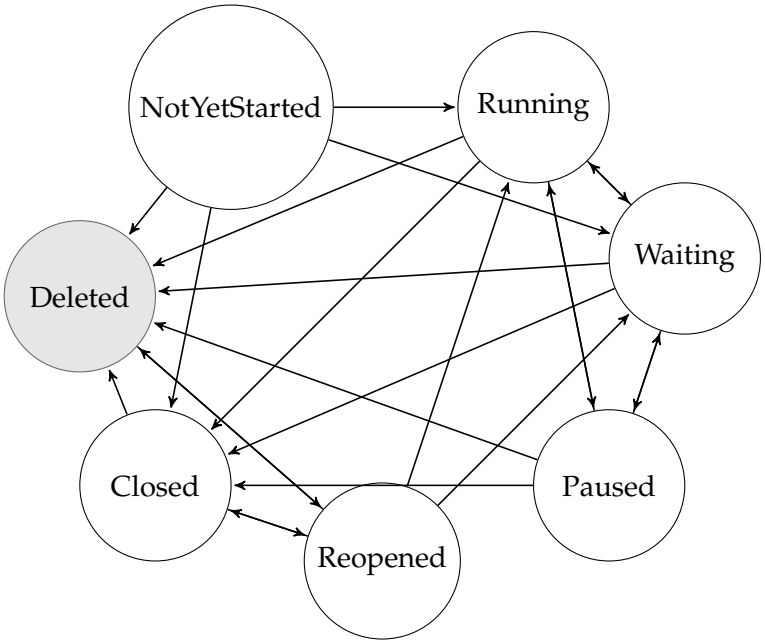


Abbildung 3.1.: Zustandsübergänge von Aufgaben

Attribut	*Task 1*	*Task 2*	*Task 3*
Name	Einkaufen	Training	Handbuch lesen
startTime	–	14:15	–
dueTime	18:00	15:00	–
est. Runtime	30 min	20 min	10 min
location	Supermarkt	Stadion	Zuhause
manualPriority	Medium	Medium	High
escalation	Top	Top	Ignore

Tabelle 3.2.: Sals Beispielaufgaben

3.7. Zusammenfassung

Das hier gezeigte Datenmodell für persönliche Aufgaben beinhaltet alle notwendigen Informationen, um die Priorisierung durchzuführen und eine Integration in die PerFlow Umgebung zu ermöglichen. Es dient als Grundlage für die gesamte weitere Arbeit und findet sich in dieser Form auch in der Implementierung wieder. Anhand des aufgezeigten Szenarios wird die weitere Entwicklung der Komponenten erläutert.

4. Methoden der Priorisierung

4.1. Motivation

In diesem Kapitel werden Methoden der Priorisierung aus verschiedenen Bereichen vorgestellt. Sie dienen als Ideengrundlage für die Entwicklung des Priorisierungsalgorithmus im darauffolgenden Kapitel.

4.2. Anforderungen

Bei der Priorisierung von Tasks müssen diese anhand von Informationen aus den Eigenschaften und Kontextinformationen des Benutzers in eine Reihenfolge gebracht werden, die angibt welcher Task zuerst ausgeführt werden sollte, welcher als nächstes und so weiter. Dazu müssen die relevanten Attribute erkannt werden und ein Verfahren entwickelt werden, das eine sinnvolle Priorisierung erlaubt. Eine essentielle Eigenschaft des Verfahrens muss die dynamische Priorisierung sein. Das heißt Tasks werden bei der Veränderung von Rahmenbedingungen neu priorisiert, um eine aktuelle Reihenfolge zu erhalten. Die Priorisierung sollte versuchen die Aufgaben so einzuplanen, dass diese vor ihrer Fälligkeit erledigt werden können um Verspätungen zu minimieren. Die Performanceanforderungen des Verfahrens beeinflussen später auch nachhaltig die Wahl der Architektur. So ist für eine Berechnung auf dem mobilen Endgerät ein deutlich einfacherer Algorithmus zu entwerfen als für eine zentralisierte Architektur.

4.3. Berechnungsmethoden

Priorisierung spielt in vielen Bereichen eine wichtige Rolle. Beispiele hierfür sind Anwendungen zur Einreihung von Arbeitsschritten in eine Sequenz, die heuristische Bewertung von Daten, oder das Scheduling in Prozessoren. Bei allen Verfahren geht es darum, aus einzelnen Attributen der Objekte ein Reihenfolge abzuleiten, die ausgeführt werden kann oder Objekte anhand ihrer Eigenschaften zu bewerten, um dann entsprechende Aktionen auszuführen. Im Folgenden werden einige der bestehenden Verfahren vorgestellt. Hierbei unterscheiden sich die Verfahren hauptsächlich durch die Anzahl der zur Priorisierung

verwendeten Attribute. Bei den Bewertungsalgorithmen werden beliebige Eigenschaften zur Entscheidung herangezogen, bei den Prozessorplanern liefert normalerweise ein zeitbasiertes Attribut die nötigen Informationen.

4.3.1. Echtzeit-Schedulingverfahren

Auf einer echtzeitfähigen Maschine müssen die einzelnen Aufgabenpakete oder Prozesse in eine Ausführungsreihenfolge für die verfügbaren Ressourcen gebracht werden. Dabei handelt es sich primär um die Vergabe von Rechenzeit auf dem Prozessor. Diese Algorithmen heißen Schedulingverfahren. Im Gegensatz zur restlichen Arbeit werden in diesem Abschnitt die Aufgabenpakete für Prozessoren als Tasks bezeichnet. Die Verfahren *FIFO*, *Least-Laxity-First*, *Fixed Priority* und *Earliest-Deadline-First* sollen hier nun beschrieben werden und ihre Adaptionsmöglichkeit für persönliche Aufgaben untersucht werden. Die Informationen über die Schedulingverfahren stammen aus [Gö10].

FIFO

Eines der einfachsten Verfahren ist das FIFO-Verfahren. Hierbei werden die Tasks entsprechend ihrer Erstellungsreihenfolge eingeplant. Der am frühesten erstellte Task wird zuerst ausgeführt. Dieses Verfahren erlaubt keine Umpriorisierung bei veränderten Rahmenbedingungen, ist aber sehr einfach zu implementieren. Auf Grund fehlender Dynamik ist es für die in dieser Arbeit geplante Anwendung nicht geeignet.

Earliest-Deadline-First

Beim Earliest-Deadline-First-Verfahren (EDF) liegt das Augenmerk auf der Einhaltung von Deadlines. Es gehört daher zu den zeitbasierten Verfahren und plant immer den Task ein, der den frühesten Fertigstellungstermin hat. Das führt dazu, dass Tasks, unter der Voraussetzung der Machbarkeit, immer vor ihrer Deadline begonnen werden. Dieses Verfahren wird sehr häufig in Echtzeitsystemen eingesetzt[Gö10]. Da die Zeitspanne bis zur Fertigstellung nur einer der Aspekte der Aufgabenpriorisierung sein soll, ist dieses Verfahren für den TaskManager nur bedingt geeignet.

Least Laxity First

Das Least-Laxity-First-Verfahren ist eine Abwandlung des EDF-Verfahrens. Dabei werden die Aufgaben nach ihrem Spielraum sortiert und dann der Prozess mit dem geringsten Spielraum gestartet. Unter dem Spielraum eines Prozesses versteht man die Zeitdifferenz zwischen seiner Fertigstellungszeit und seiner geplanten Ausführungszeit und seiner Bereitzeit, also

die Zeitspanne die der Task "zu früh" fertig wäre wenn er sofort starten würde. Der Rechenaufwand bei diesem Verfahren ist relativ hoch, da für jeden Task der Spielraum berechnet werden muss. Außerdem wird bei diesem Verfahren sehr oft umpriorisiert. Es ist aber optimal für unterbrechbare Prozesse geeignet. Auf Grund des Ansatzes, Aufgaben so spät wie möglich, aber immer rechtzeitig einzuplanen, wird es im folgenden Kapitel für die Bewertung von Zeitprioritäten persönlicher Aufgaben verwendet.

Fixed Priority

Das Fixed-Priority-Verfahren gehört wiederum zu den einfacheren Verfahren. Hierbei wird jedem Prozess eine feste Priorität zugewiesen. Der Scheduler plant die Tasks dann so ein, dass immer der Task mit der verbleibenden Höchstpriorität als nächstes startet. Dieser Ansatz wird im hier entwickelten Algorithmus für die Bewertung der manuellen Priorisierung aufgegriffen.

4.3.2. Bewertungsalgorithmen

Im Gegensatz zu den Schedulingverfahren, die die Gesamtheit der Aufgaben betrachten, sind die Bewertungsalgorithmen für die Berechnung eines Wertes für ein einzelnes Objekt ausgelegt. Hierbei wird einem Objekt eine Wertung vergeben, die unabhängig von den weiteren Objekten in der Prüfung ist. Somit eignen sich diese Verfahren bei kontinuierlichen Objektströmen, die auf Basis von Erfahrungswerten betrachtet werden. Als Beispiel einer heuristischen Bewertung von Daten soll die Spamerkennung dienen. Hierbei wird eine eingehende Nachricht mit vielen verschiedenen Filtern und Algorithmen bearbeitet. Dabei vergibt jeder Filter eine Wertung der Nachricht als Dezimalzahl, die die Nachricht als potentiellen Spam, als neutral oder als erwünscht einstuft. Das Spamsystem addiert nun die einzelnen Bewertungen und vergleicht diese Summe gegen einen vordefinierten Schwellwert. Wird nun dieser Schwellwert bei einer Nachricht überschritten, wird diese je nach Konfiguration als Spam markiert oder gelöscht. Eine Umsetzung dieser Art der Spam- und Virenbekämpfung erfolgt beim kombinierten Einsatz von amavisd[1] mit ClamAV[2] und SpamAssassin[3]. Ein Beispiel der Bewertung eingehender Nachrichten lautet wie folgt.

```
amavis[19864]: ... Passed SPAM, ... quarantine: ...
   Hits: 14.175, size: 10292, queued_as: 952F616F00CBA, 4114 ms
amavis[19865]: ... Passed CLEAN ...
   Hits: -2.563, size: 782, queued_as: 4CEA716F00CA8, 291 ms
```

[1] http://www.ijs.si/software/amavisd/
[2] http://www.clamav.net
[3] http://spamassassin.apache.org/

Hierbei wurde die erste Nachricht als SPAM markiert, da sie den Schwellwert von 10 deutlich überschritten hat. Die zweite Nachricht wurde mit einem Wert von $-2,563$ akzeptiert und dem Empfänger zugestellt.

Dieser Ansatz der Bewertung von Objekten durch Vergabe von Punkten und der anschließenden Einordnung anhand der Gesamtwertung soll nun auch im hier entwickelten Algorithmus Anwendung finden. Hierbei dienen die Schedulingverfahren zur Berechnung der Summanden und die Gesamtsumme der Werte danach zur Priorisierung der Aufgaben.

4.4. Zusammenfassung

In diesem Kapitel wurden mehrere Methoden vorgestellt, um Tasks zu priorisieren. Dabei wurden hauptsächlich Konzepte aus dem Prozessorscheduling aufgezeigt. Das Scheduling von persönlichen Aufgaben wird nun aus den Kernideen dieser Verfahren zusammengestellt, wobei hierbei mehrere Verfahren kombiniert werden. Trotz der erhöhten Komplexität des Least-Laxity-First-Verfahrens wird es im Folgenden für die Bewertung von Taskdeadlines eingesetzt. Die Idee der Bewertung von Objekten durch eine Punktvergabe bildet im hier entwickelten Algorithmus die Grundlage der Priorisierung, da dadurch eine sehr hohe Flexibilität gegeben ist und das Punktesystem sehr einfach durch einen erfahrenen Benutzer den eigenen Bedürfnissen angepasst werden kann. Ein signifikanter Unterschied gegenüber den Schedulingverfahren für Prozessoren ist die Tatsache, dass bei PerFlows durchaus einzelne eingeplante Tasks gar nicht zur Ausführung kommen, sondern verfallen oder vorher wieder aus der Priorisierung entnommen werden. Dies muss bei der Entwicklung des Algorithmus beachtet werden.

5. Priorisierungsalgorithmus

5.1. Motivation

In diesem Kapitel soll nun der konkrete Algorithmus zur Priorisierung von Aufgaben erläutert werden. Dazu werden zuerst die relevanten Eigenschaften genannt und die verwendete Gewichtung erläutert. Danach folgt der Algorithmus selbst sowie Angaben zur Neuberechnungslogik. Diese gibt an in welchen Fällen der Algorithmus neu angestoßen wird um eine Neupriorisierung der Aufgabenliste vorzunehmen. Abschließend wird eine Erweiterung des Algorithmus für Aufgabenabläufe vorgestellt.

5.2. Algorithmus

Grundlage für die Priorisierung ist die Taskdefinition gemäß 3.4. Alle folgenden Schritte basieren auf den getroffenen Annahmen und Festlegungen.

Zur Vereinfachung des Algorithmus wird davon ausgegangen, dass die Bewertung eines Tasks unabhängig von den anderen Tasks in der Liste ist. Die Priorisierung ergibt sich anschließend aus der Sortierung der Aufgaben nach ihrer Bewertung. Zur Bewertung werden nun für alle Faktoren verschiedenen Punktzahlen vergeben, die dann für jeden Task aufaddiert werden und so die Bewertung der Aufgabe darstellen. Dabei bedeutet eine größere Zahl eine höhere Bewertung und somit später eine höhere Priorisierung. Der Basiswert für einen Task ohne relevante Faktoren ist 0. Die Bewertung eines Tasks hängt dabei immer vom Kontext des Benutzers ab, für den die Bewertung vorgenommen wird. Ein Task hat somit keine globale Kennzahl, sondern wird für jeden Benutzer getrennt bewertet.

Die Abbildung χ ist somit wie folgt definiert:
Sei $\mathbb{T}$ die Menge der Tasks und $\mathbb{U}$ die Menge der Benutzer im System, dann ist:
$\chi : \mathbb{T} \times \mathbb{U} \rightarrow \mathbb{N}$

Für jeden bewertungsrelevanten Faktor i in einer Aufgabe oder im Kontext des Benutzers wird nun eine Kennzahl α_i bestimmt, die die Wichtigkeit der Aufgabe anhand dieses Blickwinkels bewertet. Dabei sollte der Maximal-, Neutral- und Minimalwert in sinnvollem Zusammenhang mit den übrigen Faktoren stehen, um die Gewichtung nicht zu sehr zu verzerren. Weiter wird für jedes Faktorenpaar i, j eine Funktion ρ_{ij} definiert, die die Kennzahlen

α_i und α_j als Argumente bekommt und als Funktionswert eine Anpassung der Kennzahl auf Grund von Korrelationen der betreffenden Faktoren liefert. So kann eine Anpassungsfunktion bei positiv korrelierten Werten die starke Wirkung der beiden Bewertungsfunktionen teilweise kompensieren. Die Berechnung der Kennzahl χ erfolgt nun anhand folgender Formel.

Seien $\alpha_i(\tau, v)$ die n Bewertungsfunktionen der Attribute und Kontextfaktoren und $\rho_{ij}(\alpha_i, \alpha_j)$ die paarweisen Anpassungsfunktionen bei Korrelation der Faktoren i und j,
dann gilt für einen Task $\tau \in \mathbb{T}$ und einen Benutzer $v \in \mathbb{U}$:

$$\boxed{\chi(\tau, v) = \sum_{i=1}^{n} \alpha_i(\tau, v) + \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij}(\alpha_i(\tau, v), \alpha_j(\tau, v))}$$

Zur Erstellung einer priorisierten Liste wird nun für einen festen Benutzer $v \in \mathbb{U}$ und für alle aktiven Tasks $\tau_i \in \mathbb{T}$ der Wert $\chi_i = \chi(\tau_i, v)$ berechnet. Die Sortierung aller Tasks nach ihrem Wert χ_i bildet nun die Aufgabenliste des Benutzers v.

5.3. Faktoren zur Priorisierung

Bei der Priorisierung von Tasks sind ausgewählte Faktoren zu berücksichtigen. Dazu gehören im Rahmen dieser Arbeit der Aufgabenstatus, der Fälligkeitszeitpunkt, die Position und die manuelle Priorität. Andere Faktoren dienen aber durchaus als Hilfswerte bei der Berechnung der Priorität. In einer Erweiterung könnten hier zum Beispiel noch andere Kontextinformationen wie das aktuelle Fortbewegungsmittel des Benutzer oder das aktuelle Wetter am Ausführungsort mit eingezogen werden. Auf die Annahme einer Korrelation einzelner Faktoren wird hier zur Vereinfachung verzichtet. Die Gewichtung der einzelnen Aspekte in dieser Arbeit wird nun ausführlich behandelt. Der Wertebereich der Kennzahl χ ist im hier umgesetzten Fall $-100 \leq \chi \leq 115$.

5.3.1. Gewichtung der Faktoren

Die folgenden Eigenschaften haben Einfluss auf die Bewertung:

Aufgabenstatus

Der Status einer Aufgabe legt die weitere Bewertung des Tasks fest. Ein laufender Task (Status *Running*) wird, wenn er durch den anfragenden Benutzer bearbeitet wird, höher bewertet, da das Fertigstellen einer Aufgabe sinnvoller ist als eine neue Aufgabe zu beginnen.

Wenn der Task pausierbar ist gilt: $\alpha_1 = 10$ sonst gilt: $\alpha_1 = 30$. Wird ein laufender Task von einem anderen Benutzer bearbeitet, wird er herabgestuft. Es gilt: $\alpha_1 = -25$

Ein Task der sich im Zustand *Paused* befindet wird mit 10 Punkten bewertet um ihn einem neuen Task vorzuziehen. $\alpha_1 = 10$

Befindet sich der Task im Wartemodus (Status *Waiting*) wird er heruntergestuft, da er momentan nicht ausgeführt werden kann. $\alpha_1 = -10$

Spielraum bis zur Fälligkeit

Sollte für einen Task keine Fälligkeit angegeben sein wird eine Wertigkeit von 0 angelegt.

Bei gegebener Fälligkeit berechnet sich der Spielraum λ aus der Differenz der Zeit bis zur Fälligkeit und dem geschätzten Ausführungsende bei sofortigem Beginn ($\lambda = due(\tau) - (time(v) + duration(\tau))$). Gegenüber einer Bewertung nach Fälligkeit hat dies den Vorteil, dass die benötigte Zeit zur Fertigstellung der Aufgabe in die Berechnung mit einfließt. Die Grundlage für diese Bewertung stellt das Least-Laxity-Verfahren dar [Gö10, S. 342].

Bei abgelaufenen Tasks wird der Wert der Eskalationsstrategie berücksichtigt [Tabelle: 5.1].

Eskalation	α_2
Ignore	25
Kill	-40
Top	50

Tabelle 5.1.: Bewertung je nach Eskalationsstrategie

Bei zukünftigen Werten gilt [Tabelle: 5.2].

Spielraum in Minuten	α_2
$0 < \lambda \leq 30$	25
$30 < \lambda \leq 60$	15
$60 < \lambda \leq 180$	10
$180 < \lambda \leq 360$	5
$360 < \lambda \leq 1440$	0
$1440 < \lambda$	-10

Tabelle 5.2.: Bewertung nach Spielraum

Manuelle Priorität

Benutzer haben die Möglichkeit den Aufgaben eine manuelle Priorität zu setzen. Dies lässt einen gewissen Spielraum zu die Priorisierung zu beeinflussen. Es wird wie bei bekannten Taskplanungstools nur zwischen drei Varianten unterschieden. Ein Task kann *High*, *Medium* oder *Low* als Priorität haben. Die soll auch nur eine Möglichkeit für spezielle Priorisierungen sein und nicht flächendeckend verwendet werden, da hierdurch die automatische Priorisierung ad absurdum geführt wäre. Die Wertigkeit der manuellen Bewertung findet sich in [Tabelle: 5.3].

Priorität	α_3
High	10
Medium	0
Low	-10

Tabelle 5.3.: Wertigkeit der manuellen Priorität

Entfernung Task - Benutzer

Die Einbeziehung der Entfernung zwischen dem Benutzer $loc(v)$ und dem Ausführungsort der Aufgabe $loc(\tau)$ ist zentraler Bestandteil der vorliegenden Arbeit. Somit wird diesem Wert auch eine große Relevanz zugemessen. Als Entfernung wird die Großkreisdistanz verwendet. Die Berechnung erfolgt über die Semiversus-Formel mit einem angenommenen Erdradius von $6371km$.

Mit Breitengrad $\phi_{1,2}$ und Längengrad $\lambda_{1,2}$ gilt für die Entfernung δ:

$$\delta = 2 \times 6371\ km \times \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Für die Bewertung der Entfernung gilt nun die [Tabelle: 5.4]

Entfernung in km	α_4
$0 < \delta \leq 1$	25
$1 < \delta \leq 10$	15
$10 < \delta \leq 100$	0
$100 < \delta \leq 1000$	-15
$1000 < \delta$	-25

Tabelle 5.4.: Wertigkeit der Entfernung

Zeitpunkt: 12:00 Uhr
Aktueller Ort: Zuhause

Task 1	Task 2	Task 3	Worklist
$\alpha_1 = 0$	inaktiv	$\alpha_1 = 0$	Task 3 (35)
$\alpha_2 = 5$		$\alpha_2 = 0$	Task 1 (5)
$\alpha_3 = 0$		$\alpha_3 = 10$	
$\alpha_4 = 0$		$\alpha_4 = 25$	

Zeitpunkt: 14:20 Uhr
Aktueller Ort: Zuhause

Task 1	Task 2	Task 3	Worklist
$\alpha_1 = 0$	$\alpha_1 = 0$	$\alpha_1 = 0$	Task 2 (40)
$\alpha_2 = 5$	$\alpha_2 = 25$	$\alpha_2 = 0$	Task 3 (35)
$\alpha_3 = 0$	$\alpha_3 = 0$	$\alpha_3 = 10$	Task 1 (5)
$\alpha_4 = 0$	$\alpha_4 = 15$	$\alpha_4 = 25$	

Zeitpunkt: 15:00 Uhr
Aktueller Ort: Stadion

Task 1	Task 2	Task 3	Worklist
$\alpha_1 = 0$	fertig	$\alpha_1 = 0$	Task 1 (35)
$\alpha_2 = 10$		$\alpha_2 = 0$	Task 3 (25)
$\alpha_3 = 0$		$\alpha_3 = 10$	
$\alpha_4 = 25$		$\alpha_4 = 15$	

Zeitpunkt: 17:00 Uhr
Aktueller Ort: Zuhause

Task 1	Task 2	Task 3	Worklist
fertig	fertig	$\alpha_1 = 0$	Task 3 (35)
		$\alpha_2 = 0$	
		$\alpha_3 = 10$	
		$\alpha_4 = 25$	

Tabelle 5.5.: Bewertung der Beispielaufgaben

5.4. Priorisierung der Szenariotasks

Aus diesem Algorithmus und dem Szenario aus Abschnitt 3.2 ergeben sich für die Tasks die Bewertungen und Reihenfolgen aus [Tabelle: 5.5]. Dazu wurden vier verschiedene Kontextsituationen innerhalb Sals Tag verwendet. Man sieht, dass zu Beginn die Aufgabe 2 noch nicht eingeplant wurde, da sie eine Startzeit von 14.15 Uhr hat. Auf Grund der späten Deadline und der Entfernung von 11 km wird das Einkaufen sehr niedrig eingestuft. Zum zweiten Zeitpunkt ist dann die Aufgabe 2 aktiv und als hochprior eingeplant. Sal fährt nun ihre Kinder ins Stadion und erledigt damit diese Aufgabe. Um 15.00 Uhr ist dieser Task nicht mehr im System, aber auf Grund der veränderten Position von Sal wird ihr der TaskManager nun vorschlagen die Einkäufe zu erledigen. Da das Handbuch zu Hause liegt hat Task 3 an Priorität verloren. Nach erledigtem Einkauf wieder zu Hause angekommen, verbleibt um 17.00 Uhr nur eine Aufgabe im System, die Sal nun erledigen kann.

5.5. Pseudocode Implementierung

Der Code zur Erstellung der Worklist besteht aus zwei Komponenten. Ein Teil ist die Berechnung der Kennzahl eines einzelnen Tasks. Dieser Algorithmus findet sich in Listing 5.2, der andere Teil fasst die Berechnungen aller Tasks zusammen und generiert die Worklist. Dies ist in Listing 5.1 dargestellt. Dieser Listengenerator muss dann für alle Benutzer im System aufgerufen werden, für die eine Aufgabenliste erstellt werden soll.

Algorithmus 5.1 Priorisierung von Tasks

```
procedure PRIORIZETASKS(Tasklist 𝕋, User v)
    List χ                                              // Erstelle Worklist
    for all Task τ_i ∈ 𝕋 do
        prio(χ_i) ← EVALUATEPRIORITY(τ_i, v)
        task(χ_i) ← τ_i
    end for
    return sort(χ, prio)                                // Sortiere Tasks nach Kennzahl
end procedure
```

5.6. Neuberechnungslogik

In den vorangehenden Abschnitten wird erläutert, wie die Tasks priorisiert werden und Worklists erstellt werden. Da sich die Position des Benutzers oder die Eigenschaften von Aufgaben aber kontinuierlich ändern, muss die Berechnung der Worklist ständig erneuert werden. Grundsätzlich bieten sich zwei Möglichkeiten die Neuberechnung auszulösen. Zum einen kann bei Änderung von Parametern die Berechnung aktiv angestoßen werden. Zum anderen kann die Berechnung in einem gewissen Zeitintervall regelmäßig durchgeführt werden. Bei der aktiven Berechnung muss nicht auf den Ablauf des Zeitgebers gewartet werden und die Ergebnisse liegen direkt nach Änderung der Eigenschaften vor. Da die Zeit aber einer der entscheidenden Faktoren in diesem Algorithmus ist und somit zu den Kontextparametern gehört, ist in dieser Arbeit eine zeitgesteuerte Neuberechnung vorgesehen. In einer Erweiterung ließe sich trotzdem zusätzlich aktiv bei Änderung von anderen Parametern eine Neuberechnung anstoßen lassen. Die hier implementierte Lösung berechnet alle Prioritäten in einem zweiminütigen Intervall neu. Dies sollte für die Demonstration des Konzepts ausreichen.

5.7. Erweiterung: zeitliche Abläufe

Eine Erweiterung des aufgezeigten Priorisierungsalgorithmus ist die Erstellung von idealen Abläufen. Ziel ist es, aus den Aufgaben eines Benutzers einen zeitlichen Ablauf seiner Aufgaben zu generieren. Aus Komplexitäts- und Laufzeitgründen wird dabei nicht der ideale Ablauf bestimmt, da hierbei jede mögliche Reihenfolge überprüft werden müsste. Stattdessen wird nach jedem Priorisierungsschritt die am höchsten bewertete Aufgabe ausgewählt. Diese wird dann aus der Worklist entfernt und der Kontext des Benutzers angepasst. Dazu wird die Position des Benutzers temporär auf den Ausführungsort der entfernten Aufgabe gesetzt und der Zeitstempel um die geschätzte Ausführungsdauer der Aufgabe verschoben. Danach wird ein weiterer Priorisierungslauf mit den verbleibenden Aufgaben gestartet. Dies wird wiederholt, bis alle Aufgaben in den Ablauf einsortiert wurden. Der Algorithmus findet sich

Algorithmus 5.2 Bewertung von Tasks

```
procedure evaluatePriority(Task τ, User υ)
    int prio = 0
    if state(τ) = Running then
        if user(τ) = υ then
            if pausable(τ) = true then
                prio ← prio + 10
            else
                prio ← prio + 30
            end if
        else
            prio ← prio − 25
        end if
    end if
    if state(τ) = Paused then
        prio ← prio + 10
    end if
    if state(τ) = Waiting then
        prio ← prio − 10
    end if
    prio ← prio+ getLaxityPoints(τ, υ)
    prio ← prio+ getPriorityPoints(τ, υ)
    prio ← prio+ getDistancePoints(τ, υ)
    return prio;
end procedure
```

in Listing 5.3. Der so erzeugte Aufgabenablauf wird dann dem Benutzer zur Ausführung vorgeschlagen. Durch die dynamische Neuberechnung des Ablaufs bei jeder Änderung bleibt dem Benutzer aber erstens die Möglichkeit doch eine andere Aufgabe zuerst zu bearbeiten, was dann durch den Aufgabenstatus auch berücksichtigt wird, und zweitens können weitere Aufgaben der Liste hinzugefügt werden, die dann beim nächsten Durchlauf in den idealen Ablauf integriert werden. Der Algorithmus besitzt allerdings zwei Einschränkungen. Er geht davon aus, dass ein Benutzer immer nur eine Aufgabe gleichzeitig ausführen kann und bei der temporären Kontextanpassung werden keine Zeitaufwände für das Erreichen der neuen Lokation berücksichtigt.

Algorithmus 5.3 Erstellung von Abläufen

```
procedure createSeries(Tasklist 𝕋, User v)
    List χ ← prioritizeTasks(𝕋, v)          // Priorisierung durchführen
    Task τ ← χ.first;                        // ersten Task auswählen
    𝕋 ← 𝕋.remove(τ)
    time(v) ← time(v) + duration(τ)          // Kontext anpassen
    loc(v) ← loc(τ)
    return τ|| createSeries(𝕋, v)            // rekursiver Aufruf
end procedure
```

5.8. Zusammenfassung

Der in diesem Kapitel vorgestellte Algorithmus stellt das zentrale Ergebnis dieser Arbeit dar. Er erlaubt es Aufgaben anhand ihrer Kontextinformationen für einen Benutzer zu priorisieren. Dieser Algorithmus wurde dann am Beispielszenario demonstriert und eine Pseudocodeimplementierung vorgestellt. Zum Abschluss wurde der Algorithmus um die Erzeugung idealer Abläufe erweitert.

6. Architektur des TaskManagers

6.1. Motivation

In diesem Kapitel soll die Architektur des Systems dargestellt werden. Zu Beginn werden die Anforderungen an das Gesamtsystem definiert und mögliche Lösungsansätze diskutiert. Anschließend wird das gewählte Konzept näher erläutert und der Systementwurf, der die Aufteilung in einzelne Komponenten beschreibt und die Definition der Schnittstellen zur Kommunikation der Komponenten beschrieben. Im Anschluss werden verschiedene Konzepte für Clientapplikationen dargestellt und die entsprechenden Theorien erarbeitet. Ziel ist es, eine klare Struktur des TaskManagers aufzuzeigen, um eine ideale Interoperabilität der verschiedenen Implementierungen zu gewährleisten.

6.2. Anforderungen

Die Anforderungen an die Architektur des TaskManagers leiten sich einerseits aus den gewünschten Zielplattformen für Benutzer und andererseits aus Sicherheitsthemen und Schnittstellenüberlegungen ab. Da eine möglichst große Zahl von Plattformen für die Benutzerinteraktion angestrebt ist und hier in kurzen Zeitabständen neue Systeme hinzukommen, sollte die Architektur eine relativ leichte Adaption auf verschieden Geräte erlauben. Auch dürfen die Benutzerdaten nicht an ein konkretes Gerät gebunden sein. Auf Seiten der Infrastruktur ist eine Interaktion mit der PerFlow-Plattform notwendig sowie die Anbindung von Kontextinformationssystemen. Datenschutzüberlegungen beschränken dabei die Möglichkeiten der Ablage und des Zugriff auf Benutzerdaten. Die konkreten Anforderungen sollen nun genauer beleuchtet werden.

Mehrbenutzerfähigkeit

Da Aufgaben mehreren Benutzer gleichzeitig zugewiesen werden können und diese dann die Zuständigkeit beim Start übernehmen, muss das System eine Verwaltung von Benutzerdaten beinhalten und muss die Daten auch mehreren Benutzer zugänglich machen. Eine redundante Verwaltung aller Aufgaben pro Benutzer in einem eigenen System ist nicht sinnhaft und auch schlecht durch die Ausführungsumgebung koordinierbar.

Mobile Endgeräte

Die Grundidee der PerFlows geht von mobilen Geräten aus, die der Benutzer bei sich trägt. Eine Architektur muss darauf ausgelegt sein, auf Geräten mit beschränkten Ressourcen zur Ausführung zu kommen. Die Ortsveränderlichkeit mobiler Endgeräte ist ebenso zu berücksichtigen. Dies impliziert Einschränkungen bei der Internetkonnektivität und verbietet das Voraussetzen einer gewissen Umgebung in der sich der Anwender bei der Benutzung aufhalten muss.

Verschiedene Clients pro Benutzer

Je nach Aufenthaltsort und Zweck der Verwendung des TaskManagers wird der Anwender verschiedene Geräte zur Bedienung des Systems verwenden. Das Abrufen der Aufgaben und Listen wird eher über mobile Endgeräte erfolgen, wogegen die Verwaltung und Bearbeitung von Aufgabendetails auf Desktopsystemen stattfinden wird. Es muss also möglich sein, von wechselnden Endgeräten auf die Daten desselben Benutzers zuzugreifen ohne diese vorher synchronisieren zu müssen.

Kurze Ladezeiten von Aufgabenlisten

Die Abfrage der aktuellen Aufgabenliste ist, wie für Smartphoneanwendungen typisch, keine längere Verwendung der Applikation, sondern spielt sich innerhalb weniger Sekunden ab. Um das Nutzererlebnis zu steigern und somit überhaupt eine sinnvolle Verwendung des Systems zu ermöglichen, muss die Anwendung die Daten sehr schnell aufbereiten und anzeigen. Der Benutzer hat kein Verständnis dafür, nach dem Start der Applikation mehrere Sekunden auf seine Aufgabeliste warten zu müssen.

Schnittstelle zur PerFlow-Engine

Da der TaskManager eine Komponente der PerFlow-Umgebung ist, muss er mit anderen Bestandteilen wie dem Prozesskoordinator, der Benutzerverwaltung oder anderen Endgeräten interagieren können. So muss die PerFlow-Engine beispielsweise über eine Schnittstelle neue Aufgaben im TaskManager anlegen können, wenn eine entsprechende Prozessaktivität bearbeitet wird. Hierzu muss der TaskManager allerdings durch diese Komponenten auffindbar und auch ständig erreichbar sein.

Datenschutz und Privatsphäre

Da es sich bei den persönlichen Aufgaben um Objekte innerhalb der Privatsphäre eines Menschen handelt, muss mit einer gesteigerten Sorgfalt mit ihnen verfahren werden. So kann zum Beispiel keine ungeschützte, zentrale Ablage der Daten im Internet erfolgen, da sonst jedermann Zugriff auf die Tasks anderer hätte. Auch der psychologische Aspekt der Ablage der Daten im *Netz* ist bei der Wahl der Architektur zu bedenken. Für den Benutzer wäre eine Ablage auf dem Gerät oder auf einer Maschine im persönlichen Zugriff wünschenswert.

Einfaches Deployment

Auf Grund der Schnelllebigkeit mobiler und stationärer Endgeräte und dem häufigen Austausch dieser Komponenten muss die Integration neuer Geräte in das Gesamtsystem sehr einfach sein und auch von unerfahrenen Benutzern problemlos zu bewerkstelligen sein. Die Integration neuer Komponenten darf vor allem auf den bestehenden Geräten keine weitere Anpassung erfordern.

6.3. Lösungsansätze

Zur Realisierung des TaskManagers stehen nun verschiedene Architekturkonzepte zur Auswahl, welche aber auf die Erfüllung der eben erläuterten Anforderungen untersucht werden müssen. Dazu wird das Konzept kurz beschrieben und auf die Tauglichkeit für das hier aufgezeigte Szenario geprüft. Die Spannweite der Architekturansätze reicht hierbei von einer monolithischen Anwendung bis hin zu einem P2P-System ohne zentrale Komponente.

6.3.1. Konzepte

Monolith

Eine Realisierung wäre die Erstellung einer monolithischen Anwendung. Dazu wird eine Applikation entwickelt, die auf einem Geräte ausgeführt wird und die gesamte Verwaltung von der Erfassung über die Priorisierung bis hin zur Anzeige der Aufgabenlisten beinhaltet. Die Datenhaltung befindet sich hierbei ebenfalls direkt in der Anwendung. Dieser Ansatz ist für das Konzept untauglich, da die Anforderungen an die Unterstützung mehrerer Endgeräte nicht erfüllt werden kann ohne den gesamten Datenbestand auf alle beteiligen Geräte zu replizieren und auf jedem die Priorisierung getrennt durchzuführen. Die Konsistenz der Daten und Listen kann hierbei nicht garantiert werden. Auch die Anbindung an die Ausführungsplattform der PerFlows lässt sich in dieser Architektur schlecht umsetzen.

Client-Server

Eine weitere Möglichkeit besteht in der Realisierung als Client-Server-Anwendung. Dabei existiert ein zentraler Server, der die Datenhaltung übernimmt und die Berechnung der Aufgabenlisten durchführt. Jegliche Datenmanipulation durch Clients wird zentral auf dem Server durchgeführt und steht sofort allen Clients zur Verfügung. Durch die zentrale Ablage der Daten auf dem Server gibt es allerdings Verzögerungen beim Abruf der Aufgabenlisten durch den notwendigen Serverzugriff, der bei fehlender Internetkonnektivität gar nicht stattfinden kann. Dieses Problem muss bei einer Client-Server-Architektur unbedingt berücksichtigt werden. Die Anbindung an die Ausführungsumgebung stellt bei diesem Konzept keine Probleme dar, allerdings muss der Anwender einen eigenen TaskManager-Server betreiben.

Cloud

Durch die ständige Verbindung zum Internet kann auch eine Implementierung als Cloudanwendung angedacht werden. Dies ist die logische Konsequenz aus den Überlegungen zur Client-Server-Architektur und somit gelten dieselben Vor- und Nachteile. Ein weiterer Vorteil besteht darin, dass der Benutzer sich nicht um eine Wartung oder Anbindung der Serverkomponente kümmern muss. Die Datenraten und Rechenleistungen einer Cloudanwendung lassen sich nahezu beliebig skalieren und erfordern keine nachträgliche Aufrüstung bei steigenden Performanceanforderungen. Der große Nachteil dieser Architektur ist allerdings die zentrale Ablage der persönlichen Daten vieler Nutzer an einer zentralen und *öffentlich* zugänglichen Stelle. Zwar ist auch ein privater Server über dieselben Wege für jeden erreichbar, trotzdem ist diese Zentralisierung von privaten Daten für die Anwender im Allgemeinen nur bis zu einem gewissen Grad tragbar. Für die erfolgreiche Umsetzung der PerFlow-Vision sollten aber alle Aufgaben und Prozesse im System hinterlegt sein, was eine Unvereinbarkeit mit der zentralen Datenhaltung darstellen würde. Somit scheidet zum jetzigen Zeitpunkt eine Realisierung dieser Architektur aus. Allerdings zeigen Projekte wie Facebook[1] oder einige Googledienste eine wachsende Bereitschaft der Benutzer auch private Daten in öffentlichen Netzen zu lagern. Auch arbeiten die Betreiber von Cloud-Diensten sicher bereits an einer Lösung der Datenschutzproblematik bei Cloud-Anwendungen.

Peer-to-Peer

Der Verzicht auf zentrale Komponenten und die Umsetzung des TaskManagers als Peer-to-Peer-System soll die möglichen Konzepte abschließen. Hierbei bilden alle Geräte eines Benutzers oder einer logischen Einheit von Benutzern ein P2P-Netzwerk, das die Aufgaben

[1] http://www.facebook.com

gemeinsam verwaltet und priorisiert. Durch die Verteilung der Daten und das Fehlen einer Verwaltungsinstanz bestehen aber hinsichtlich der Datenkonsistenz beim Abruf der Aufgabenliste ähnliche Probleme wie bei einer monolithischen Anwendung. Die Verwaltung des P2P-Netzwerks birgt außerdem eine nicht zu unterschätzende Komplexität, die für diesen relativ einfachen Anwendungsfall den Nutzen übersteigt.

6.3.2. Auswahl

Nach einer Abwägung der in den Konzepten beschriebenen Vor- und Nachteile wurde für die Implementierung in dieser Arbeit eine Client-Server-Architektur gewählt[Abb: 6.1]. Diese lässt sich außerdem leicht implementieren und beschleunigt die Entwicklung einer prototypischen Implementierung. Auf Grund der genannten Bedenken zur Offenlegung privater Daten wurde der Cloudansatz nicht weiter verfolgt. Das Verständnis von Datenschutz und Privatsphäre unterliegt momentan aber einem massiven Wandel und bringt teilweise übertrieben Einschätzungen und Interpretationen mit sich[Kir10]. In weiteren Entwicklungen könnte also eine Cloudanwendung als klarer Favorit für die Wahl der Architektur dienen. Die Umwandlung der Client-Server-Struktur in eine Cloudapplikation ist aber bei Bedarf durch die Wahl einer entsprechenden Plattform sehr einfach umzusetzen.

Abbildung 6.1.: Gesamtarchitektur

6.4. Serverarchitektur

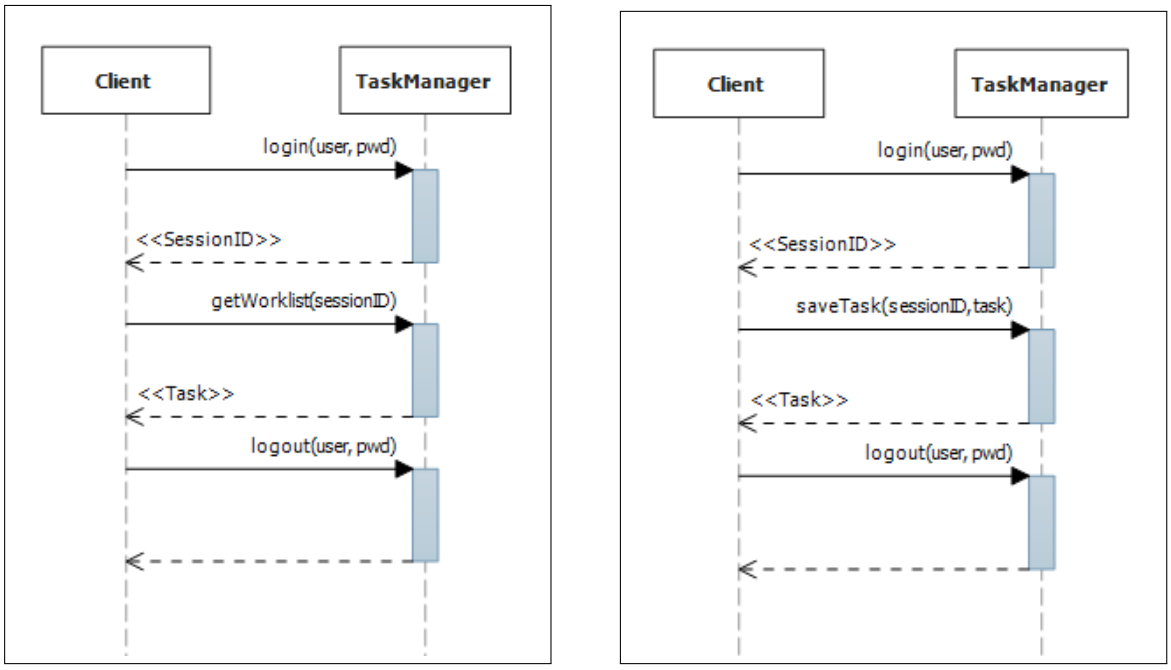
Das Serversystem des TaskManagers gliedert sich in einzelne Komponenten, um eine Modularisierung zu ermöglichen. Die Aufteilung erfolgt nach dem Three-Tier-Prinzip. Die zentrale Geschäftslogik wird durch eine Persistenzschicht unterstützt, die die Datenhaltung übernimmt. Die Schnittstellen zu den Clients sind klar definiert, um eine Anbindung verschiedener Anzeige- und Verwaltungskomponenten zu ermöglichen. Da aus Komfortgründen eine Anbindung an andere Benutzerverwaltungen möglich sein soll, wird diese vom eigentlichen TaskManager entkoppelt. Dies verhindert zum Beispiel eine Redundanz bei der Verwaltung von Benutzern im PerFlows-Szenario. Die Verwaltung von Kontextinformationen, wie etwa Lokationsdaten von Benutzern wird ebenfalls vom Tasksystem getrennt. Hier könnte im Produktivbetrieb eine umfangreichere Kontextlösung verwendet werden, wie zum Beispiel die Nexus Plattform[GBH+05]. Aus diesem Grund ist auch die Aktualisierung der Kontextdaten durch die Sensoren nicht Teil der Taskclients, sondern muss separat durch eine weitere Anwendung erfolgen. Dies betrifft sowohl mobile Sensoren in Smartphones als auch stationäre Sensoren wie RFID-Baken oder Ortungssysteme.

6.5. Schnittstellendefinition

Die Schnittstellen bieten dem Client die Möglichkeit mit dem Server zu interagieren. Bei den Schnittstellen handelt sich um die Benutzerverwaltung, die Taskverwaltung und die Kontextverwaltung. Die Schnittstelle ist als reiner Prototyp entworfen und darf in dieser Form nicht produktiv verwendet werden, da essentielle Sicherheitsfunktionen fehlen. So verlangt die Benutzerverwaltung zum Beispiel keine Authentifizierung zur Bearbeitung von Systembenutzern. Im realen Umfeld sollte die Benutzerverwaltung durch ein externes System implementiert werden. Die Authentifizierung des Benutzers geschieht durch ein Sessionkonzept. Dazu wird beim Login eine Benutzersitzung erzeugt und dem Benutzer die *SessionID* mitgeteilt. Bis zum Logout kann er dann die Funktionen der Task- und Kontextverwaltung unter Angabe der *SessionID* verwenden. In den Abbildungen 6.2(a) und 6.2(b) werden einige exemplarische Interaktionen beschrieben. Die vollständige Schnittstellendefinition inklusiv aller Methode findet sich im Anhang B.

6.6. Clientkonzepte

Bei der Realisierung der Clients gibt es eine Menge verschiedener Nutzungsszenarien. Daher gibt es mehrere unterschiedliche Clientkonzepte die im Folgenden kurz erläutert werden. Dabei wird berücksichtigt, welche Geräte ein Benutzer haben könnte und auf welchen die Verwendung des TaskManagers sinnvoll erscheint.



(a) Abfragen der Aufgabenliste

(b) Anlegen eines neuen Tasks

Abbildung 6.2.: Beispiele der Client-Server Interaktion

6.6.1. Desktopclients

Eine Anwendung ist die vollwertige Administration des Taskplaners, um neue Tasks anzulegen, bestehende zu verändern und Benutzereinstellungen zu setzen. Dieser Client ist typischerweise eine Desktopanwendung mit vollem Funktionsumfang. Eine Realisierung als Weblösung wäre in diesem Szenario ebenfalls denkbar. Bei Desktopanwendungen kann von einem leistungsstarken Host mit bestehender Internetverbindung ausgegangen werden, was wenige Einschränkungen bei der Implementierung bedingt. So können Daten im Speicher vorgehalten werden und Serverzugriffe können jederzeit zügig stattfinden.

6.6.2. Mobile Clients

Ein weiteres Szenario ist die Verwendung von mobilen Endgeräten zur Anzeige der Worklist und einzelner Aufgaben, wie zum Beispiel Smartphones, Tablets, oder Ähnliches. Eine Bearbeitung von Tasks findet nur im begrenzten Rahmen statt. Hier sind hauptsächlich Statusänderungen interessant, aber keine komplette Editierfunktion. Dieser Clienttyp hat allerdings die größte Implementierungsvielfalt. Für die meisten Geräte müssen eigene Versionen entwickelt werden, da sich die Plattformen unterschiedlicher Programmiersprachen und

Betriebssysteme bedienen und auch die Bedienkonzepte der Plattform zu berücksichtigen sind.

Auf diesen Geräten steht nur eine begrenzte Rechenleistung und Speichermenge zur Verfügung, das heißt, die Anwendung sollte nur die aktuelle benötigten Daten vorhalten und auf Grund eingeschränkten Multitaskings auch jederzeit mit Programmunterbrechungen umgehen können. Die Verwendung der Internetverbindung muss optimiert werden, da nicht zu jedem Zeitpunkt eine bestehende oder schnelle Verbindung angenommen werden kann. Auch die Menge der Steuerdaten sollte gering gehalten werden, um eine übermäßige Nutzung von Internetressourcen zu verhindern.

6.6.3. Anzeigeclients

Ein dritter Typ Client ergibt sich aus der Möglichkeit reine Anzeigeobjekte zu entwickeln. Diese dienen rein der Information des Benutzers und geben keine Möglichkeit der Bearbeitung oder Interaktion. Beispiele hierfür sind Mac OS X Widgets oder Windows Sidebar Gadgets.

Da diese Clients durchgehend aktiv sind, aber nicht die Vordergrundaktivität des Benutzers darstellen, ist eine ressourcenschonende Implementierung zwingend notwendig. Es sollte darauf geachtet werden, keine zu großen Mengen an Arbeitsspeicher zu verwenden. Auch die Prozessor- und Netzwerklast sollte gering gehalten werden, um den Benutzer bei seiner Tätigkeit nicht zu beinträchtigen.

6.7. Zusammenfassung

In diesem Kapitel wurden die grundlegenden Konzepte für die Entwicklung des TaskManagers entworfen. Außerdem ergeben sich einige Anforderungen an die Implementierung aus den getroffenen Vorüberlegungen. Durch die Nutzung verschiedenster Clientanwendungen muss die Schnittstelle weitgehend plattform- und sprachunabhängig sein. Außerdem sollten die Repräsentation der Daten einen geringen Overhead an Steuer- und Protokolldaten enthalten. Dies ermöglicht die schnelle Übertragung der Daten in langsamen Netzwerken wie GPRS oder UMTS und eine schnellere Startzeit der Anwendung.

Teil III.

Implementierung

7. Serverimplementierung

7.1. Motivation

In diesem Kapitel wird die Implementierung der Serverkomponente vorgestellt. Es handelt sich dabei um die zentrale Einheit des TaskManagers, die die Verwaltung der Aufgaben und die Priorisierung übernimmt. Sie bietet dann Schnittstellen um Clients die Anfrage und Bearbeitung von Daten zu ermöglichen. Die Realisierung erfolgt mit Hilfe eines Application Servers und einer nachgeschalteten Datenbank.

7.2. Gesamtarchitektur

Das Gesamtsystem des TaskManagers gliedert sich in einzelne Komponenten um eine Modularisierung zu ermöglichen. Die Aufteilung erfolgt nach dem Three-Tier-Prinzip. [Abb: 7.1]

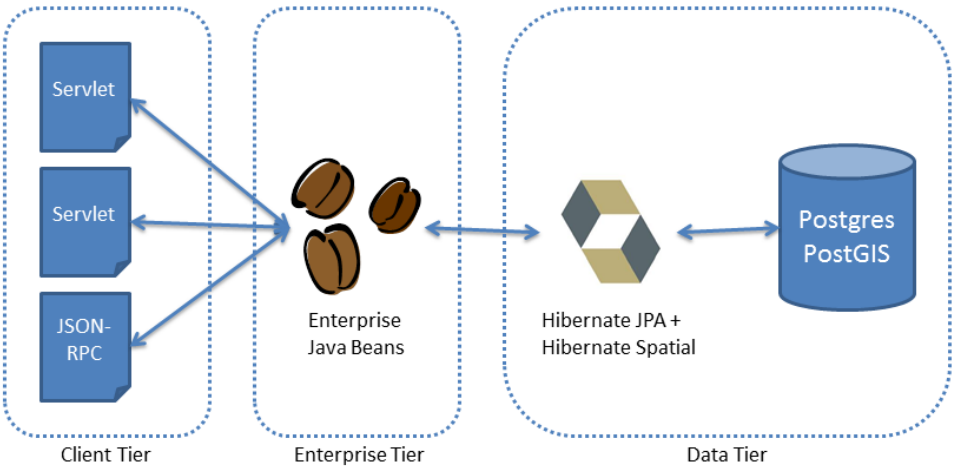


Abbildung 7.1.: Komponentengliederung des Servers

7.2.1. Datenbank

Als Datenbank wird eine PostgreSQL[1] in der Version 9.0 eingesetzt. Auf dieser Basis wird die PostGIS[2] Erweiterung genutzt. Sie bietet eine Unterstützung für Geometriedaten in der Datenbank. Dies erleichtert die Verwaltung und ermöglicht eine intelligentere Nutzung der Daten.

7.2.2. Application Server

Als Server wird ein JBoss AS in der Version 6.0[3] verwendet. Dies ist die neueste Communityversion des Application Servers von RedHat Enterprise. Er implementiert JavaEE in der Version 6[4] und bietet eine Anbindung an die Datenbank mit Hilfe von JPA 2.0. Als konkrete Datenbankanbindung wird Hibernate[5] verwendet, das für diesen Zweck um Hibernate Spatial[6] erweitert wurde um die Geometriedaten der PostGIS-Datenbank zu verwenden.

Die Realisierung der Geschäftslogik findet in SessionBeans statt, die direkt mit der Persistenzschicht interagieren können. Die Schnittstelle für die Clients wird durch ein Servlet angeboten, das die empfangen Daten an die entsprechenden EJBs delegiert. Die Neuberechnung der Taskprioritäten wird durch den JavaEE 6 Scheduler gesteuert.

7.2.3. Weboberfläche

Die grundlegenden Funktionalitäten des Systems sind ebenfalls als rudimentäre Weboberfläche ausgeführt. Diese Webanwendung basiert auf der Verwendung von Servlets und Java Server Pages[7]. An dieser Stelle sind auch einige zusätzliche Schnittstellen zur Anbindung an standardisierte Protokolle implementiert. Ein Beispiel hierfür ist der GeoRSS[8] Feed der Worklist eines Benutzers.

Die Weboberfläche des TaskManagers bietet einzelne Funktionalitäten auf einem allgemeinzugänglichen Weg über den Browser. Dies beinhaltet vor allem die Anzeige der Worklist eines Benutzers, die Anzeige von Aufgabendetails sowie die Erstellung einer Übersichtskarte über die Aufgaben auf der aktuellen Worklist ([Abb: 7.2]). Auch die Anzeige der zeitlichen

[1] http://www.postgresql.org
[2] http://postgis.refractions.net
[3] http://www.jboss.org
[4] http://www.oracle.com/technetwork/java/javaee/tech/index.html
[5] http://www.hibernate.org
[6] http://www.hibernatespatial.org
[7] http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[8] http://www.georss.org

Abläufe (Abschnitt: 5.7) erfolgt über die Webdarstellung. Hierzu wird eine Karte eingeblendet auf der sich der Ablauf als Pfad darstellt zusammen mit einer einblendbaren textuellen Erläuterung. Zur Visualisierung der Karte kann hier der Datenbestand von OpenStreetMap[9] verwendet werden oder die Satellitenbilder von Google Maps eingeblendet werden, die über die Kartenbibliothek OpenLayers[10] angezeigt werden.

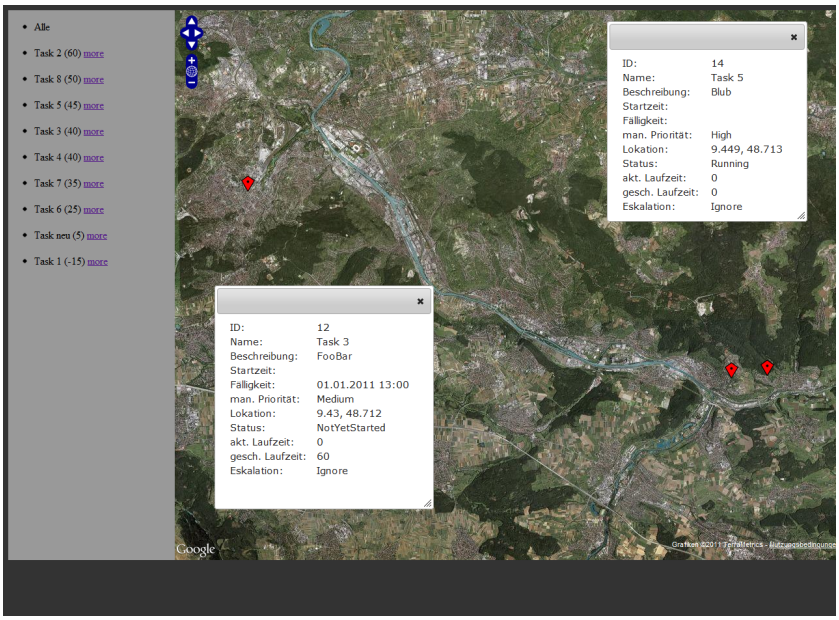


Abbildung 7.2.: Kartenanzeige mit geöffnetem Taskdialog

7.3. Zusammenfassung

In diesem Kapitel wurde die Implementierung der Serversoftware erläutert. Die Serverkomponente bietet die Kernfunktionalität für das System. Alle Datenverwaltungsaufgaben werden hier übernommen. Die Clients interagieren immer mit diesem zentralen Punkt.

[9]http://www.openstreetmap.de/
[10]http://openlayers.org/

8. Schnittstellenumsetzung

8.1. Motivation

In diesem Kapitel sollen die verschiedenen Möglichkeiten zur Umsetzung des Schnittstellenprotokolls aus Abschnitt 6.5 verglichen werden. Dazu werden die Vor- und Nachteile der Implementierungen betrachtet und die Unterstützung durch die verschiedenen Clientplattformen geprüft. Anschließend wird das für diesen Zweck beste Protokollframework verwendet.

8.2. Protokollalternativen

Es gibt eine Vielzahl unterschiedlicher Ansätze das Protokoll zu implementieren. Auf Grund der Verwendung mehrerer verschiedener Clientplattformen ist die Verwendung von Binärprotokollen in diesem Fall nicht ratsam. Die Interoperabilität der Systeme steht an oberster Stelle. Deshalb fallen auch programmiersprachenabhängige Lösungen wie zum Beispiel Java-RMI aus. Die Lösung sind hier standardisierte Protokolle aus dem Bereich der Web Services. Aus diesem Segment sollen nun *XML-RPC* [Win99], *SOAP* und *JSON-RPC* [Mat10] untersucht werden. Eine Verschlüsselung der Daten während der Übertragung zwischen Client und Server ist in einer Produktivumgebung durchaus anzustreben, wurde aber im Rahmen dieser Arbeit weder berücksichtigt noch implementiert.

XML-RPC

XML-RPC ist ein Protokoll, das es Anwendungen auf verschiedenen Plattformen erlaubt Methodenaufrufe über das Internet durchzuführen. Als Transportprotokoll wird dabei HTTP verwendet, um eine allgemeingültige und stabile Basis zu haben. Darin wird dann die Darstellung der Daten und Methodenaufrufe in XML eingebettet. XML-RPC hat das Ziel so einfach wie möglich zu sein aber trotzdem nicht die Mächtigkeit zu weit einzuschränken auch komplexe Datenobjekte zu verwenden.[Win99]

SOAP

Als Weiterentwicklung von XML-RPC wurde 1999 die erste Version von SOAP, damals noch als Abkürzung für *Simple Object Access Protocol*, veröffentlicht. Es bietet, wie sein Vorgänger auch, die Möglichkeit Methodenaufrufe über das Netzwerk zu tätigen. Die Datenrepräsentation ist ebenfalls ein XML-Format, das jedoch deutlich komplexer ist als die Darstellung eines XML-RPC Aufrufs. Als Transportprotokoll können neben HTTP auch viele andere Protokolle dienen, wie zum Beispiel SMTP oder JMS. Diese Wahlfreiheit vergrößert die Einsatzmöglichkeiten von SOAP und hat dazu geführt, dass es de facto Standard in Anwendungssystemen im Geschäftsumfeld ist. Ein weiterer Vorteil von SOAP ist die explizite Beschreibung der Schnittstelle und ihrer Objekte, Transportmöglichkeiten, Endpunkte und sonstigen Metadaten.[WCL+05]

Diese Flexibilität und Sicherung der Stabilität führen aber auch dazu, dass die Implementierung von SOAP in Anwendungen deutlich komplexer ist als andere RPC Methoden. Auch der Datenoverhead der einzelnen Methodenaufrufe ist deutlich höher, was gerade in Verbindungen mit niedriger Datenrate zum Nachteil einer SOAP-basierten Lösung werden kann.

JSON-RPC

JSON-RPC ist ein einfaches RPC-Protokoll basierend auf der *JavaScript Object Notation* [Cro06]. Es ist transportunabhängig und kann über die verschiedensten Wege übermittelt werden. Da die Austauschobjekte reinen Text beinhalten, kann dies von einfachen TCP/UDP Sockets, HTTP POST Requests oder SMTP Inhalten übernommen werden. Durch die Verwendung von JSON ist die Verwendung plattformunabhängig und der Metadatenoverhead deutlich geringer als zum Beispiel bei XML-Nachrichten. Auch die Verfügbarkeit von Bibliotheken für verschiedene Programmiersprachen und Softwareplattformen ist deutlich höher als bei XML. JSON enthält allerdings keine Beschreibungssprache wie zum Beispiel XSD[W3C04] und bietet somit keine strenge Typsicherheit in der Schnittstelle.

8.3. Pro/Contra je Zielplattform

In diesem Abschnitt werden die Vor- und Nachteile der verschiedenen Protokolle im Hinblick auf die geplanten Zielplattformen aufgezeigt und abschließend eine Entscheidung gefällt, welches Protokoll für die Implementierung verwendet wird.

Server / JBoss-AS

Die Serverkomponente hat vergleichsweise wenig Probleme mit den verschiedenen Transportmethoden. Während die SOAP Unterstützung direkt mit der Annotation *@WebService* aktiviert werden kann und der Server die WSDL und XSD Dokumente selbstständig erstellt müssen bei XML-RPC und JSON-RPC entsprechende Webendpunkte als Servlets implementiert werden, was aber ein minimaler Mehraufwand ist. Aus Sicht der Serverimplementierung ist somit keine Präferenz für eine Methode zu wählen.

RCP / Java

Bei Java Clients muss für SOAP Schnittstellen eine Sammlung von Datenobjekte und Proxyklassen erstellt werden. Dies kann aber durch einen Generator direkt aus der WSDL geschehen. Eine Implementierung für XML-RPC steht von Apache[1] zur Verfügung. Diese hat allerdings Probleme bei der Verwendung von Listen als Argumente und Rückgabewerte. Bei der Verwendung von JSON-RPC muss ein entsprechendes JSON-Objekt erstellt werden[2] und als Text serialisiert werden. Dieser kann dann beliebig an eine Serverschnittstelle versendet werden. Programmatisch bedingt hier JSON-RPC den größten manuellen Aufwand, bietet aber auch die größte Flexibilität bei Veränderungen der Schnittstelle.

BlackBerry

Die Entwicklung auf dem BlackBerry basiert auf Java. Allerdings wird als Laufzeitumgebung eine Abwandlung von JavaME verwendet. Die Verwendung von XML stellt eine größere Hürde dar und lässt sich auch nicht automatisch generieren oder verwenden. Auf Grund der Speicheroptimierung lassen sich auch nur SAX-Parser verwenden, was die Entwicklung nochmal erschwert. Eine SOAP Implementierung ist äußerst schwerfällig, da alle Nachrichten "von Hand" erstellt werden müssen und dies sehr fehlerträchtig ist. Die Unterstützung von JSON hingegen ist kein Problem, da eine Implementierung für JavaME vorliegt [3]. Somit bedingt das Ziel einer BlackBerry Lösung die Verwendung von JSON-RPC.

iOS / Objective-C

Unter Objective-C ist die Verwendung von XML und SOAP ebenfalls nicht trivial. Trotz des eigentlichen de facto Standards XML ist eine Unterstützung seitens des Frameworks alles

[1] http://ws.apache.org/xmlrpc/
[2] http://www.json.org/java/index.html
[3] https://github.com/upictec/org.json.me/

andere als gut. SOAP fällt auf Grund der manuell zu erstellenden Nachrichten wie beim BlackBerry aus. Für JSON existiert eine Implementierung für iOS [4], die von Apple für den Datenaustausch mit WebServices empfohlen wird. Auch der *Apple Push Notification Service* verwendet zur Kommunikation mit den Endgeräten das JSON Datenformat[Mar10].

8.4. Auswahl

Eines der wichtigen Ziele des TaskManagers ist die Verwendung von mobilen Endgeräten. Da diese meist über Funknetzwerke an das Internet angebunden sind und nur begrenzte Datenraten anbieten, sollte der Overhead von Metadaten im Transportprotokoll möglichst minimiert werden. Auch muss das Protokoll von vielen Plattformen unterstützt werden, da es für mobile Endgeräte eine Vielzahl von Plattformen und Programmiersprachen gibt. Aus diesem Grund wird für die hier vorgestellte Arbeit JSON-RPC als Transportprotokoll verwendet. Zur Übermittlung der Nachrichten wird das HTTP-Protokoll eingesetzt. Diese Wahl hält auch eine Erweiterung auf Android offen, da auch dort JSON gut unterstützt wird, während die Implementierung für XML[5] und SOAP[6] in ihrer Entwicklung stehen geblieben sind und keine saubere Unterstützung anbieten. Durch die Redundanz des Feldnamens im schließenden Tag eines XML Elements ist ein XML Dokument auch deutlich größer als ein inhaltlich identisches JSON Objekt. Dies führt wiederum zu einer längeren Übertragung in langsamen Netzwerken und zu einem höheren Datenaufkommen bei volumenbasierten Abrechnungsmodellen der Nutzer.

8.5. Zusammenfassung

Dieses Kapitel hat die Alternativen der Schnittstellenimplementierung aufgezeigt und die gewählte Lösung näher erläutert. Dazu wurden die Vor- und Nachteile der Alternativen im Bezug auf die verschiedenen Zielplattformen beleuchtet und die JSON-Bibliothek als für diesen Einsatz optimal angesehen.

[4] http://code.google.com/p/json-framework/
[5] http://kxml.sourceforge.net/
[6] http://ksoap2.sourceforge.net/

9. Desktop Client

9.1. Motivation

Da die Bedienung mobiler Endgeräte im Normalfall deutlich langsamer abläuft als die Nutzung eines Desktop PCs, geschieht die Administration des TaskManagers über einen Desktopclient. Aus Gründen der Entwicklungsgeschwindigkeit wurde im Rahmen dieser Arbeit eine RichClient Anwendung entwickelt. Als Basis diente hierzu das Eclipse RCP Framework, das im Folgenden kurz beschrieben wird.

9.2. Eclipse Plattform

Die Eclipse RCP[1] bietet dem Entwickler ein vollständiges Framework zur schnellen Entwicklung von Desktopanwendungen. Es liefert die gesamte Ausführumgebung inklusive Lifecyclemanagement der UI Komponenten, sodass der Entwickler nur noch die Oberflächenmasken implementieren muss, um sie danach mit der Businesslogik zu verknüpfen. Als Oberflächen Toolkit kommt dabei SWT[2] zum Einsatz, um eine dem nativen Betriebssystem ähnliche Gestaltung zu erreichen.

9.3. Funktionalität Rich Client

Die Rich Client Anwendung bietet unter anderem die Möglichkeit Aufgaben zu verwalten. Dazu wird dem Benutzer eine Liste all seiner Aufgaben angezeigt und es ihm ermöglicht, neue Aufgaben zu erstellen, vorhandene Aufgaben zu editieren oder auch bestehende Aufgaben zu löschen. Das Anlegen einer Aufgabe geschieht über einen Editor, in dem der Benutzer die Basisdaten der Aufgabe erfasst und danach die gewünschten Benutzer als mögliche ausführende Personen zuweist. Diese erhalten dann die Aufgabe ebenfalls in ihrer Worklist auf jedem Client oder in der Aufgabenübersicht in ihrem Desktop Client. Über den Editor können auch die Daten bestehender Aufgaben angezeigt und verändert werden und

[1] http://www.eclipse.org
[2] http://www.eclipse.org/swt

die Zuweisungen von Benutzern aktualisiert werden. Statusübergänge sind, sofern möglich, direkt per Knopfdruck durchführbar. Das Löschen von Aufgaben erfolgt entsprechend der Taskdefinition durch den Übergang der Aufgabe in den Status *Deleted*.

Ein weiterer Anwendungsfall des Desktop Clients ist die Anlage und Verwaltung von benannten Positionen im System durch die Identifizierung von Koordinaten mit einem Namen. Dazu steht eine Übersicht der vorhandenen Lokationen zur Verfügung sowie eine interaktive Karte zur Erstellung neuer Positionen. Zur Anzeige von Karten werden auch hier wieder Daten von OpenStreetMap verwendet.

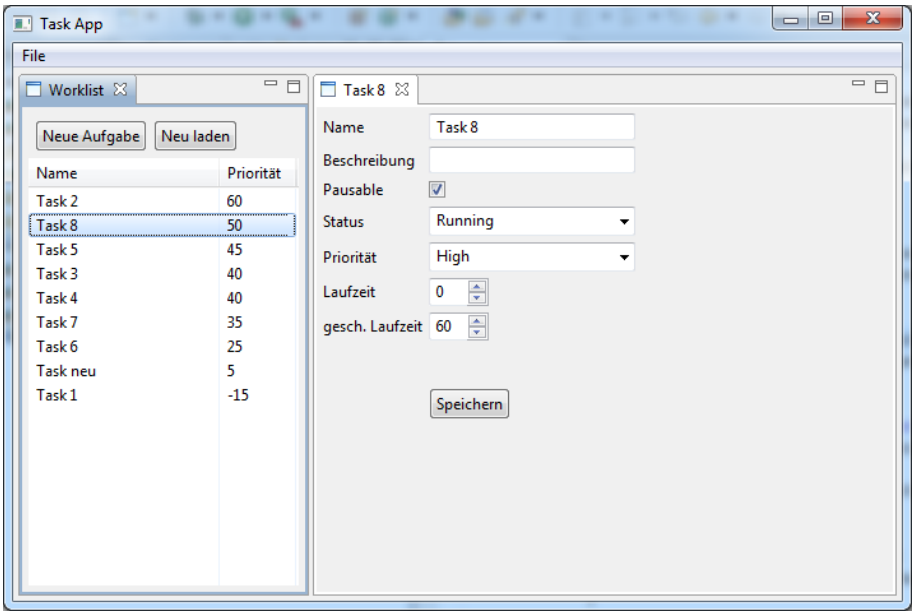


Abbildung 9.1.: Hauptansicht mit Aufgabenliste und Taskeditor

9.4. Zusammenfassung

Auf Grund der Mächtigkeit und vielseitigen Bedienbarkeit des Desktop PCs sind die Clientanwendungen für diese Plattform am vielseitigsten und bieten auch den vollen Funktionsumfang inklusive Erstellen und Bearbeitung von Aufgaben. Eine Erweiterung des Rich-Clients ist noch die Integration der Benutzerverwaltung, die hier nur rudimentär vorgesehen ist aber nicht implementiert wurde.

10. Mobile Client

10.1. Motivation

Die Verwendung des TaskManagers soll die Verwaltung von persönlichen Aufgaben erleichtern und eine Priorisierung anhand der aktuellen Kontextinformationen vornehmen. Um diesen Kontext zu erfassen sollte die Bedienung des Tasksystems nicht an einen Desktopcomputer gebunden sein, sondern von überall zugreifbar sein. Auf Grund der weiten Verbreitung von Smartphones liegt eine Bereitstellung der Aufgabenverwaltung als Applikation auf mobilen Endgeräten nahe. Dabei sind aber einige Spezialitäten von mobilen Geräten zu beachten. Zum ersten sind die Ausführungsumgebungen nicht standardisiert, so dass jedes Smartphone potentiell eine eigene Implementierung benötigt. Zum anderen sind die Hardwareausstattungen dieser Geräte leistungsärmer als Desktops und somit unterstützen diese auch nicht den gesamten Funktionsumfang von Desktopprogrammen und müssen sparsamer mit Ressourcen wie Prozessorleistung, Speicher und Energie umgehen. Beispiele sind hier das Positionsbestimmungsintervall auf Grund des hohen Strombedarfs eines GPS-Empfängers, geringeres lokales Caching von Daten wegen verfügbaren Arbeitsspeichers und das Transportformat um die Datenmenge gering zu halten, die über die Funkschnittstelle übertragen wird.

In der vorliegenden Arbeit sollen zwei Plattformen herausgegriffen werden und die Taskverwaltung beispielhaft umgesetzt werden. Auf Grund der Verfügbarkeit der Geräte wurden hierfür ein BlackBerry 9700[1] von Research In Motion und ein iPad[2] der Firma Apple verwendet. Die Betriebssysteme der beiden Geräte sind RIM OS 5.0 für den BlackBerry und iOS 4.2 für das iPad.

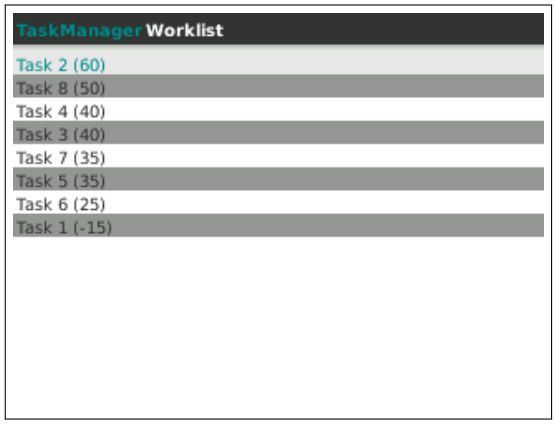
10.2. BlackBerry Client

Bei der Clientsoftware für den BlackBerry müssen einige Unterschiede zu einem normalen Desktop PC beachtet werden. Zum einen ist das Display des BlackBerry deutlich kleiner ($6,2cm$) und hat mit 480×360 Pixeln auch eine geringere Auflösung als ein Desktop PC mit

[1] http://de.blackberry.com/devices/blackberrybold9700/
[2] http://www.apple.com/de/ipad/

1280×1024 auf einem Monitor mit $48cm$ Diagonale. Dies führt dazu, dass nicht mehrere Teile der Anwendung auf einmal angezeigt werden können und somit ein Navigationskonzept für die Anwendung benötigt wird. Außerdem hat das Gerät allgemein ein anderes Bedienkonzept, welches in der Anwendung ebenfalls verwendet werden sollte um eine intuitive Bedienung zu ermöglichen. Da das Hauptaugenmerk auf der mobilen Nutzung liegt kann auch die WLAN-Verbindung unbeachtet gelassen werden und mit EDGE oder UMTS Datenraten gerechnet werden. Diese bedingen eine sinnvolle und bedachte Nutzung von Serverabfragen und wenig Overhead bei der Datenübermittlung. Als weitere Schwierigkeit kommt hinzu, dass die Entwicklung der App zwar in Java erfolgt, aber die Bibliothek nur eine RIM spezifische Abwandlung von Java Mobile ist. Dieses RIM-JDK bietet einige der Vorteile von Java5 oder Java6 nicht an, wie zum Beispiel Generics. Auch die Oberflächenentwicklung unterscheidet sich deutlich von der Arbeit mit dem Standard Widget Toolkit.

Der Funktionsumfang der BlackBerry Lösung beschränkt sich daher auf das Anzeigen der eigenen Worklist ([Abb: 10.1(a)]) und einer Detailansicht einzelner Aufgaben ([Abb: 10.1(b)]). Eine Manipulation von Daten ist in dieser ersten Version nicht enthalten.



(a) Aufgabenliste auf dem BlackBerry

(b) BlackBerry Taskdetails

Abbildung 10.1.: BlackBerry Client

10.3. iOS Client

Die Entwicklung auf dem iPad unterliegt grundsätzlich denselben Einschränkungen wie beim BlackBerry. Dazu kommt noch ein völlig anderes Bedienkonzept der iOS-Familie und die damit verbundenen Entwicklungsvorgaben seitens des Herstellers und die Tatsache, dass das iPad ausschließlich einen berührungsempfindlichen Monitor hat und jegliche Interaktion mit dem Benutzer darüber stattfindet. Dadurch müssen alle Oberflächenkomponente so

angelegt und entworfen sein, dass sie mit dem Finger bedient werden können, was eine untere Schranke der Elementgrößen darstellt. Die Programmierung selbst findet in der Sprache Objective-C statt, die für Entwickler anderer Sprachen nicht sehr intuitiv ist und einige Fallstricke bietet. In dieser Arbeit wurde auch der Funktionsumfang der iPad-Lösung soweit reduziert, dass nur die Anzeige der Worklist und die Aufgabendetails ([Abb: 10.2]) verfügbar sind. Als Erweiterung gegenüber der Smartphonelösung könnte aber auf Grund des größeren Displays (25cm mit 1024×768) eine Anzeige der Worklist als interaktive Karte umgesetzt werden, wie sie auch in der Weboberfläche verfügbar ist. Dies wurde im Rahmen dieser Arbeit nicht implementiert.

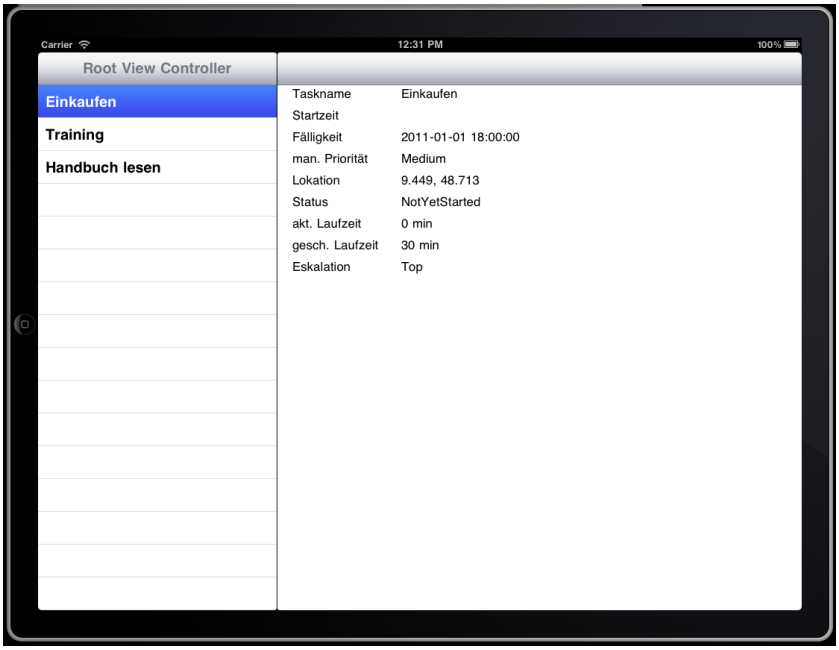


Abbildung 10.2.: Taskliste mit Taskdetails als Splitview

10.4. Zusammenfassung

Die Entwicklung für mobile Endgeräte stellt den Entwickler vor gewisse Herausforderungen bei der Verwendung von Ressourcen wie dem verfügbaren Anzeigebereich, der Internetanbindung und vorhandener Rechenleistung. Dies verlängert die benötigte Zeit zur Entwicklung von Anwendungsfunktionalität erheblich im Gegensatz zu Desktopanwendungen. Trotzdem ist im Bereich der kontextbasierten Anwendungen eine Unterstützung von mobilen Endgeräten zwingend erforderlich.

11. Erweiterungen

Zu den in den vorherigen Kapiteln aufgezeigten Lösungen gibt es natürlich beliebig viele Erweiterungen und weitere Zielplattformen. Hier sollen nun einige Ideen aufgezeigt werden, die an das entworfene Tasksystem angebunden werden könnten.

11.1. OS X Widget

Unter Mac OS X gibt es das sogenannte Mac OS X Dashboard[1] ([Abb: 11.1]). Es handelt sich dabei um eine halbtransparente Ebene, die per Tastendruck aktiviert werden kann. Auf ihr befinden sich kleine Minianwendungen, die Widgets genannt werden und zur Unterhaltung oder zum Zugriff auf nützliche Funktionalitäten einzelner Programme dienen. Die Entwicklung von Widgets erfolgt in HTML und JavaScript. Eine Erweiterung des TaskManagers wäre nun die Entwicklung eines Dashboard Widgets zur Anzeige der Worklist des Benutzers. Außerdem könnte eine Anzeige von Aufgabendetails integriert werden.

11.2. Windows Gadget

Vergleichbar mit dem eben vorgestelleten Dashboard gibt es seit Windows Vista die sogenannte Windows Sidebar[2]. Dies ist eine vertikale Leiste, die standardmäßig am rechten Bildschirmrand erscheint und ebenfalls Minianwendungen enthält. Diese von Microsoft als Gadgets bezeichneten Fenster werden wie unter Mac OS als Webseiten entworfen und können sich per JavaScript Daten beschaffen. Durch die Verwendung von JSON als Datenformat des TaskManagers ist eine Integration sehr leicht und stellt eine praktische Ausweitung der Anzeigeclients dar. Die [Abb: 11.2] zeigt einen Designentwurf für ein TaskManager-Gadget.

[1] http://support.apple.com/kb/HT2492
[2] http://windows.microsoft.com/de-DE/windows-vista/Windows-Sidebar-and-gadgets-overview

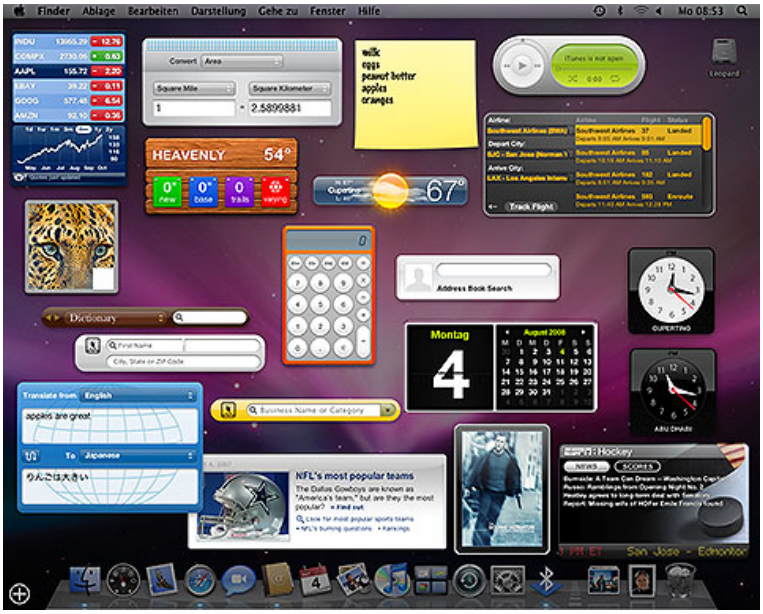


Abbildung 11.1.: Mac OS X Dashboard

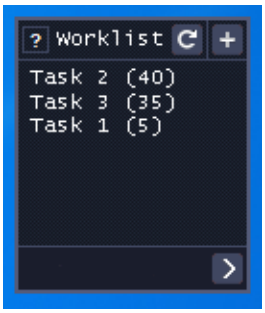


Abbildung 11.2.: Draftshot der Anwendung

11.3. Android App

Im Rahmen dieser Arbeit wurden Clients für BlackBerry und iPad geschrieben. Die dritte Plattform, mit dem momentan größten Zuwachs, ist Android. Die Programmierung für androidfähige Endgeräte erfolgt mit dem javabasierten Android-SDK[3] von Google. Auf Grund dieser zunehmenden Verbreitung muss für eine erfolgreiche Verwertung der hier erarbeiteten Lösung unbedingt eine Androidanwendung entwickelt werden.

[3]http://developer.android.com/sdk/index.html

Teil IV.

Abschluss

12. Zusammenfassung und Ausblick

12.1. Evaluation

Im Folgenden sollen nun noch einige Erkenntnisse und Erfahrungen, die bei der Implementierung gemacht wurden, zusammengefasst werden.

Datenmodell

Die Datenmodelldefinition aus Abschnitt 3.4 konnte vollständig in die Implementierung übernommen werden und hat alle Beispielszenarien abbilden können. Bei der Implementierung wurden noch ein paar Felder hinzugefügt, die aber semantisch keine Relevanz haben, sondern nur implementierungstechnisch für eine einfachere Verwaltung sorgen.

Algorithmus

Auch bei der Implementierung des Priorisierungsalgorithmus mussten keine wesentlichen Anpassungen vorgenommen werden. Die Umsetzung des Algorithmus findet sich im Server und läuft dort periodisch ab [Abb: 12.1]. Der Zeitaufwand für die gesamte Neuberechnung liegt im Bereich von Millisekunden. Da keine Mengengerüste für einen realen Einsatz vorliegen und die Testberechnungen schnell genug waren, wurden keine genauen Zeitmessungen durchgeführt. Die aktuelle Implementierung des Algorithmus bietet aber sicher noch Spielraum für eine Steigerung der Performance.

Server

Die Schwierigkeiten bei der Installation der Serverkomponenten in einer Zielumgebung liegen hauptsächlich an der Datenbank. Die Erweiterung der PostgreSQL-Datenbank um die PostGIS-Komponenten bedarf dem genauen Befolgen der Installationsanleitung und ist relativ fehleranfällig. Die restlichen Komponenten des Servers sind durch einfaches Entpacken von Archivdateien problemlos auszuliefern. Das initiale Datenbankschema muss anschließend manuell erstellt werden, da die automatische Erstellung durch Hibernate Probleme bei der Anlage der Geodatenfelder hat.

Clients

Durch die gute Unterstützung von JSON auf den Zielplattformen konnte die Umsetzung für die mobilen Endgeräte und den Desktopclient sehr schnell erfolgen. Dabei ergab sich die Hauptschwierigkeit jeweils aus dem Einrichten und Einarbeiten in die Entwicklungsumgebungen und die plattformeigenen Konzepte und Sprachen.

```
12:00:00,051 [...] [task=Task [Task 1], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=5]
12:00:00,051 [...] [task=Task [Task 3], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=35]


14:20:12,064 [...] [task=Task [Task 1], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=5]
14:20:12,064 [...] [task=Task [Task 2], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=40]
14:20:12,064 [...] [task=Task [Task 3], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=35]


15:00:03,633 [...] [task=Task [Task 1], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=35]
15:00:03,633 [...] [task=Task [Task 3], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=25]


17:00:01,527 [...] [task=Task [Task 3], user=SystemUser [Sal],
                   role=Creator, calculatedPriority=35]
```

Abbildung 12.1.: Konsolenausgabe der Priorisierung

12.2. Zusammenfassung der Arbeit

In der vorliegenden Arbeit wurde die Notwendigkeit der automatischen Verwaltung von persönlichen Aufgaben aufgezeigt und das Konzept der PerFlows vorgestellt. Hierzu wurde das Paper von Mark Weiser näher betrachtet und das Szenario von Sal aufgegriffen. Danach wurden Bezüge zu verwandten Arbeiten aufgezeigt und andere Definitionen von Aufgaben und ihrer Priorisierung erläutert. Das Hauptaugenmerk lag hierbei auf WS-HumanTask und drei Arbeiten zur Aufgabenpriorisierung. Da sich keines der bestehenden Konzepte als vollständig passend erwiesen hat, wurde anschließend ein eigenes Datenmodell für Tasks entwickelt und ein Algorithmus zur Priorisierung entworfen.

Im Weiteren folgten dann Überlegungen zur Architektur des gesamten Tasksystems und der Entwurf der Schnittstelle für den TaskManager, sowie notwendiger Zusatzkomponenten. Im Kapitel 7 konnte dann eine funktionsfähige Demonstrationsimplementierung umgesetzt werden. Dazu wurden Plattformen für den Server und die Schnittstellenbeschreibung gewählt

und die Software implementiert. Als Clients wurden eine Desktopanwendung sowie mobile Oberflächen für BlackBerry und iPad umgesetzt. Abschließend folgte eine Vorstellung möglicher Erweiterungen für die Implementierung. Zum Abschluss soll nun noch ein Ausblick auf eine Weiterführung und Verwendung gewagt werden.

12.3. Ausblick

Die hier vorgestellte Lösung lässt sich natürlich beliebig erweitern und weiteren Bedürfnissen und Anforderungen anpassen. Eine notwendige Erweiterung, die aus Gründen des Arbeitsumfangs nicht umgesetzt wurde, ist die Schaffung einer Möglichkeit die Lokationseigenschaft des Tasks funktional zu erweitern. So sollte es möglich sein die Position eines Tasks an ein dynamisches Objekt zu binden um beispielsweise Aufgaben zu definieren, die als Ausführungsort den Aufenthaltsort einer Person haben. Damit ließen sich geplante Treffen mit realen Personen als Aufgabe definieren, die dann durch die Kontextinformationen bei Eintreten einer Gelegenheit hoch priorisiert werden. Weiter könnten Tasks auch mehrere Ausführungspositionen haben um Alternativen zu erfassen. Beispiele wären hier Aufgaben für zu tätigende Einkäufe, die in einem beliebigen Supermarkt stattfinden können.

Weiter muss in einem nächsten Schritt der entworfene TaskManager an die Ausführungsumgebung von PerFlows angebunden werden, um durch die Ausführung von Prozessen die entstehenden Aufgaben in der Worklist sichtbar zu machen und eine Einbeziehung in die Priorisierung zu ermöglichen. Dazu muss der PerFlow-Koordinator als Client Zugriff auf die Schnittstelle des TaskManagers erhalten und Tasks erstellen und verändern. In diesem Zug sollte die Taskdefinition um Ein- und Ausgabedaten erweitert werden, um dem Benutzer Prozesskontextdaten zur Verfügung zu stellen und mehr Daten vom Benutzer in den Prozess einbringen zu können. Auch sollte ein Callback-Handler implementiert werden, um dem PerFlow-Koordinator Statusänderungen von Aufgaben mitzuteilen, damit dieser den Prozess entsprechend fortsetzen kann.

Teil V.

Anhang

A. Taskdatenmodell

id
Die ID ist eine eindeutige Zahl zur Identifikation eines Tasks im gesamten System. Sie ist der Primärschlüssel eines Tasks.

name
Der Name benennt die Aufgabe in kurzen prägnanten Worten. Er besteht aus einer frei wählbaren, nicht leeren, Zeichenfolge.

description
Die Beschreibung enthält einen optionalen, längeren Text, der den Inhalt der Aufgabe näher erläutert. Das Beschreibungsfeld hat keine Einschränkungen bezüglich Inhalt oder Länge.

startTime
Die Startzeit definiert den frühstmöglichen Zeitpunkt um diese Aufgabe zu beginnen. Eine Aufgabe mit einer zukünftigen Startzeit wird in der Worklist nicht berücksichtigt, sondern erst mit Erreichen aktiviert.

dueTime
Die Fälligkeitszeit gibt den Zeitpunkt an, an dem die Aufgabe abgeschlossen sein muss. Das Verhalten einer unfertigen Aufgabe nach diesem Zeitpunkt wird durch das Feld *escalation* charakterisiert.

manualPriority
Die manuelle Priorität bietet dem Benutzer die Möglichkeit Einfluss auf die automatische Sortieren der Aufgaben zu nehmen. Er kann einzelne Aufgabe als wichtig markieren um sie höher einzustufen. Mögliche Prioritäten sind *High*, *Medium*, *Low*.

location
Die Lokation gibt den Ort an, an dem die Aufgabe ausgeführt werden muss. Dieses optionale Feld bietet die notwendigen Informationen um Aufgaben kontextabhängig zu modellieren und zu priorisieren. Die Position wird als Koordinatentripel aus Längengrad, Breitengrad und Höhe (über NN) erfasst. Als Bezugskoordiantensystem dient [NIMoo].

pausable
Diese Feld gibt an, ob eine Aufgabe pausiert werden kann oder ob sie nach dem Beginn zu Ende geführt werden muss.

runUser
Bei einer bereits gestarteten Aufgabe wird hier angegeben, welcher Benutzer sie ausführt. Dies ist relevant um die Priorisierung der Aufgabe für verschiedene zugewiesene Personen zu berechnen.

runtime
Die Laufzeit gibt an, wie lange die Ausführung eines Task bisher läuft oder insgesamt gedauert hat. Die Angabe erfolgt in Minuten.

estimatedRuntime
Dieses Feld gibt die geschätze Laufzeit für die Aufgabe an. Anhand dieser Schätzung wird der Spielraum einer Aufgabe bei der Priorisierung berechnet. Auch der Ablauf von Sequenzen hängt von diesem Wert ab. Die Angabe erfolgt ebenfalls in Minuten.

escalation
Das Eskalationsverhalten gibt an, wie mit einem überfälligen Task verfahren wird. Bei *Ignore* wird die Aufgabe wie eine sofort fällige behandelt. Eine Eskalation vom Typ *Kill* verwirft eine überfällige und somit hinfällige Aufgabe und der Typ *Top* sorgt für eine starke Erhöhung der Priorität der Aufgabe um eine rasche Ausführung zu fordern.

B. Schnittstellendefinition

B.1. Benutzerverwaltung

Methode: login
Parameter: *String login, String pwd*
Rückgabe: *String SessionID*
Die Methode meldet einen Benutzer am System an und gibt eine SessionID zurück, die für alle anderen Methoden zur Authentifizierung und Authorisierung dient.

Methode: logout
Parameter: *String login, String pwd*
Rückgabe: *void*
Die Methode dient zur Abmeldung eines Benutzers. Seine Session ist danach nicht mehr gültig.

Methode: validate
Parameter: *String sessionID*
Rückgabe: *SystemUser*
Die Methode dient zur Prüfung einer SessionID. Ist diese gültig wird der entsprechende Benutzer zurückgegeben.

Methode: getByLogin
Parameter: *String loginName*
Rückgabe: *SystemUser*
Die Methode liefert den Benutzer mit dem angegebenen Loginnamen zurück.

Methode: saveUser
Parameter: *Systemuser user*
Rückgabe: *SystemUser*
Die Methode speichert den übergebenen Benutzer und liefert die gespeicherte Version wieder zurück.

Methode: deleteUser
Parameter: *SystemUser user*
Rückgabe: *void*
Die Methode löscht den angegebenen Benutzer aus dem System.

Methode: getUsers
Parameter: -
Rückgabe: *List(SystemUser)*
Die Methode liefert eine Liste aller Systembenutzer zurück.

B.2. Kontextverwaltung

Alle Methoden der Kontextverwaltung benötigen als ersten Parameter die SessionID des anfragenden Benutzers um die Berechtigungen für die Ausführung zu prüfen. Als Hilfskonstrukt für oft verwendete Positionen gibt es das Objekt der *NamedLocation*. Es beinhaltet einen Namen und die Koordinaten der Position.

Methode: getNamedLocation
Parameter: *String sessionID, String name*
Rückgabe: *NamedLocation*
Die Methode sucht eine benannte Position nach ihrem Namen.

Methode: getNamedLocations
Parameter: *String sessionID*
Rückgabe: *List(NamedLocation)*
Die Methode liefert eine Liste aller benannten Positionen.

Methode: saveNamedLocation
Parameter: *String sessionID, NamedLocation loc*
Rückgabe: *NamedLocation*
Die Methode speichert oder aktualisiert eine benannte Position.

Methode: deleteNamedLocation
Parameter: *String sessionID, NamedLocation loc*
Rückgabe: *void*
Die Methode löscht die angegebene benannte Position.

Methode: updateUserLocation
Parameter: *String sessionID, long userID, Location loc*
Rückgabe: *void*
Mit Hilfe dieser Methode kann die Position des angegebenen Benutzers im System aktualisiert werden.

Methode: updateUserLocationName
Parameter: *String sessionID, long userID, String name*
Rückgabe: *void*
Im Gegensatz zur vorherigen Methode kann hier der Name einer benannten Position verwendet werden.

Methode: updateUserLocationAtTime
Parameter: *String sessionID, long userID, Location loc, long time*
Rückgabe: *void*
Mit dieser Methode kann die Position zu einer gewissen Zeit gesetzt werden.

Methode: updateUserLocationNameAtTime
Parameter: *String sessionID, long userID, String name, long time*
Rückgabe: *void*
Diese Methode dient zur Positionsänderung zu einer gewissen Zeit in eine benannte Position.

Methode: getUserLocation
Parameter: *String sessionID, long userID*
Rückgabe: *Location*
Diese Methode liefert die letzte bekannte Position des Benutzers.

Methode: getUserLocationAtTime
Parameter: *String sessionID, long userID, long time*
Rückgabe: *Location*
Diese Methode liefert die Position des Benutzers zu einem bestimmten Zeitpunkt.

Methode: getUserTrack
Parameter: *String sessionID, long userID*
Rückgabe: *List(Location)*
Diese Methode liefert eine Liste aller Wegpunkte des Benutzers.

Methode: getUserTrackAtTime
Parameter: *String sessionID, long userID, long start, long end*
Rückgabe: *List(Location)*
Mit Hilfe dieser Methode lassen sich Wegpunkte eines Benutzers in einem gewissen zeitlichen Bereich abfragen.

B.3. Taskverwaltung

Auch die Methoden der Taskverwaltung benötigen als ersten Parameter die SessionID des anfragenden Benutzers zur Authorisierung.

Methode: getTask
Parameter: *String sessionID, long taskID*
Rückgabe: *Task*
Diese Methode liefert Details zu einer bestimmten Aufgabe.

Methode: getTasks
Parameter: *String sessionID, bool showDeleted*

Rückgabe: *List(Task)*
Diese Methode dient zur Abfrage aller zugewiesenen Aufgaben. Die Anzeige von gelöschten Aufgaben ist optional.

Methode: saveTask
Parameter: *String sessionID, Task task*
Rückgabe: *Task*
Mit Hilfe dieser Methode kann eine Aufgabe angelegt oder verändert werden.

Methode: deleteTask
Parameter: *String sessionID, Task task*
Rückgabe: *Task*
Mit Hilfe dieser Methode kann eine Aufgabe gelöscht werden.

Methode: getWorklist
Parameter: *String sessionID*
Rückgabe: *List(Task)*
Diese Methode liefert die priorisierte Worklist des anfragenden Benutzers und beinhaltet damit die Kernfunktionalität dieser Arbeit.

Methode: processTask
Parameter: *String sessionID, long taskID, String state*
Rückgabe: *bool*
Mit Hilfe dieser Methode lässt sich eine gewählte Aufgabe von ihrem aktuellen Zustand in einen neuen Zustand überführen. Der Rückgabewert zeigt an, ob diese Weiterschaltung möglich war oder nicht.

Literaturverzeichnis

[AAD+07a] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller. WS-BPEL Extension for People v1. 2007. URL http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf. (Zitiert auf Seite 16)

[AAD+07b] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, M. Zeller. WS-HumanTask Specification v1. 2007. URL http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf. (Zitiert auf den Seiten 16 und 22)

[Cro06] D. Crockford. JSON Specification RFC 4627. 2006. URL http://www.ietf.org/rfc/rfc4627.txt. (Zitiert auf Seite 52)

[Gö10] P. Göhner. Automatisierungstechnik I, 2010. SKRIPT. (Zitiert auf den Seiten 26 und 31)

[GBH+05] M. Grossmann, M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, T. Schwarz. Efficiently Managing Context Information for Large-Scale Scenarios. *Pervasive Computing and Communications, IEEE International Conference on*, 0:331–340, 2005. doi:http://doi.ieeecomputersociety.org/10.1109/PERCOM.2005.17. (Zitiert auf Seite 42)

[Hub08] E. Huber. *SmartGPS - Lokationsmodell für PerFlows*. Master's thesis, Universität Stuttgart, 2008. (Zitiert auf Seite 14)

[Kir10] M. Kirkpatrick. Facebook's Zuckerberg Says The Age of Privacy is Over. 2010. URL http://www.readwriteweb.com/archives/facebooks_zuckerberg_says_the_age_of_privacy_is_ov.php. (Zitiert auf Seite 41)

[Mar10] C. Marcellino. Data in Your iPhone App, 2010. LECTURE 9 CS193P at Stanford University. (Zitiert auf Seite 54)

[Mat10] Matt. JSON-RPC Specification. 2010. URL http://groups.google.com/group/json-rpc/web/json-rpc-2-0. (Zitiert auf Seite 51)

[NIM00] NIMA. World Geodetic System 1984. 2000. URL http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf. (Zitiert auf Seite 71)

[Org07] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (Zitiert auf Seite 16)

[OTM03] I. Ohmukai, H. Takeda, M. Miki. A Proposal of the Person-centered Approach for Personal Task Management. In *Proceedings of the 2003 Symposium on Applications and the Internet*, pp. 234 – 240. IEEE Computer Society, 2003. doi:10.1109/SAINT.2003.1183055. URL http://portal.acm.org/citation.cfm?id=827273.829223. (Zitiert auf den Seiten 16 und 17)

[SWDA08] H. Schonenberg, B. Weber, B. F. van Dongen, W. M. P. van der Aalst. Supporting Flexible Processes through Recommendations Based on History. In *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pp. 51–66. Springer, 2008. URL http://dblp.uni-trier.de/db/conf/bpm/bpm2008.html#SchonenbergWDA08. (Zitiert auf Seite 17)

[UBR06] S. Urbanski, C. Becker, K. Rothermel. Sentient processes - process-based applications in pervasive computing. In *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, pp. 4 pp. –611. 2006. doi:10.1109/PERCOMW.2006.124. (Zitiert auf Seite 13)

[UHW+09] S. Urbanski, E. Huber, M. Wieland, F. Leymann, D. Nicklas. PerFlows for the computers of the 21st century. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pp. 1 –6. 2009. doi:10.1109/PERCOM.2009.4912887. (Zitiert auf den Seiten 13 und 14)

[W3C04] W3C. XML Schema. 2004. URL http://www.w3.org/standards/techs/xmlschema#w3c_all. (Zitiert auf Seite 52)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005. (Zitiert auf Seite 52)

[Wei91] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991. URL http://nano.xerox.com/hypertext/weiser/SciAmDraft3.html. (Zitiert auf den Seiten 13 und 21)

[WF92] T. Watanabe, T. Fukumura. A scheduler of daily personal tasks on the basis of the object-oriented model. In *TENCON '92. "Technology Enabling Tomorrow : Computers, Communications and Automation towards the 21st Century.' 1992*

IEEE Region 10 International Conference., pp. 618 –622 vol.2. 1992. doi:10.1109/TENCON.1992.272028. (Zitiert auf Seite 16)

[Win99] D. Winer. XML-RPC Specification. 1999. URL http://www.xmlrpc.com/spec. (Zitiert auf Seite 51)

Alle URLs wurden zuletzt am 28.03.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Thorsten Höger)