Masterarbeit

Kontextvorhersage in einem adaptiven, kontext-sensitiven System

Stephan Mehlhase

12. September 2008

Betreuer:
Prof. Dr. Johannes Fürnkranz
Hans-Peter Zorn
Dr. Ulrich Scholz

Danksagung

Meiner Familie, meiner Freundin und deren Familie sowie meinen Freunden sei an dieser Stelle mein tiefster Dank ausgedrückt. Ohne eure in jeglicher Hinsicht nicht selbstverständliche Unterstützung wäre mein Studium in dieser Form nicht möglich gewesen.

Den Mitarbeitern des European Media Lab in Heidelberg, insbesondere meinen dortigen Betreuern Hans-Peter Zorn und Ulrich Scholz, sowie Professor Johannes Fürnkranz von der TUD gilt mein Dank zum einen für die Ermöglichung dieser extern angefertigten Arbeit innerhalb dieses interessanten Themengebiets, zum anderen wegen dem angenehmen und inspirierenden Arbeitsumfeld in dem ich diese anfertigen konnte.

Abschließend danke ich Christoph, Jens, Markus, Paul, Peter und Sebastian, die mir geholfen haben, diese Arbeit in eine syntaktisch korrekte Form zu bringen und mir Unzulänglichkeiten sowie mögliche Erweiterungen darin aufzeigten.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die zunehmende Miniaturisierung in den letzten zwei Dekaden hat den Computer von einer großen, schreibtisch-beherrschenden Maschine in eine kleine, nahezu überall verwendbaren Komponente verwandelt. Dadurch ist die durch Weiser [53] erstmals formulierte Vision des ubiquitär verteilten Rechners entstanden, in welcher der Mensch von vielen kleinen Computern, die ihn in seinen Aufgaben und Zielen unterstützen, umgeben ist. Auch wenn die Größe einzelner Systeme mit der Zeit abgenommen hat, ist die Leistungsfähigkeit doch weiter stetig gestiegen – selbst wenn sie noch weit von der Leistung eines modernen Rechnersystems entfernt ist – wodurch es ermöglicht wird immer komplexere Programme auf diesen Geräten auszuführen. Neben dieser Entwicklung führt der technische Fortschritt zu immer kleineren und billigeren Sensoren, so dass modernen Mobiltelefonen mittlerweile Unmengen an Daten über die sie umgebende Umwelt zur Verfügung stehen.

Sowohl die Rechenleistung als auch die Interaktionsmöglichkeiten mit mobilen Geräten sind obgleich dieser Entwicklung auch weiterhin im Vergleich zu normalen Rechnern stark eingeschränkt. Ansätze dieser Problematik Herr zu werden umfassen zum einen das Verteilen rechenintensiver Applikationen auf mehrere zumeist in Sachen Leistungsfähigkeit sehr heterogener Geräte, zum anderen aber gerade auch in mobilen Szenarien das Miteinbeziehen verschiedenster Faktoren – zum Beispiel Umwelt und Situation des Benutzers –, um mit diesem Wissen die Applikationen daran in einer Art anzupassen, die es erlaubt die „passende“ Funktionalität unter Verwendung weniger Ressourcen zur Verfügung zu stellen (siehe u. a. [2, 21, 42]).

In den letzten Jahren hat sich neben diesem reaktiven Verhalten auf die aktuelle Umwelt oder dessen Vergangenheit der Ansatz herauskristallisiert auch die Entwicklung der den Benutzer umgebenen Welt vorherzusagen (siehe u. a. [3, 12, 37]).

Diese Vorhersagen werden genutzt um Applikationen die Möglichkeit zu geben proaktiv zu agieren und sich im Vorhinein auf kommende Ereignisse einzustellen. Ein in diesem Szenario oft erwähntes Beispiel ist das einer Haussteuerung, die das Verhalten des Bewohners lernt, um entsprechende Aktionen im Vorhinein ergreifen zu können. Trinkt der Bewohner eines solchen Hauses beispielsweise stets nach dem Duschen einen Kaffee, kann die Steuerung bereits die Kaffeemaschine aktivieren, sobald er in die Dusche steigt.

1.1 Motivation

Bereits im Jahr 2006 gab es in Deutschland mehr Mobiltelefonanschlüsse als Einwohner und knapp 80 % der Deutschen verfügen über eines [8, 50]. In Relation zu dieser Masse an Benutzern ist das Arbeiten mit mobilen Geräten ein in der Informatik eher unterrepräsentiertes Feld. Heutzutage zielt Forschung in diesem Bereich darauf ab, die Mobilität, die ein Benutzer mit solch einem Gerät besitzt, mit in Betracht zu ziehen. In den einleitenden Worten wurde schon deutlich, dass der Entwicklung solcher Systeme eine immense Komplexität inhärent ist. Auf Grund dessen ist die Entwicklung einer Plattform für Anwendungsentwickler, die die gewünschte Funktionalität in einem mobilen Umfeld zur Verfügung stellt, gleichzeitig allerdings den Programmierer von dem erhöhten Aufwand befreit, ein derzeit aktives Forschungsfeld (siehe zum Beispiel [2] oder das später vorgestellte MUSIC Projekt). Solche Systeme sind in der Lage nicht nur auf Ereignisse in der Umwelt zu reagieren, sondern auch aus dem Zustand dieser Schlüsse zu ziehen und sich auf Grund dieser anzupassen.

Die Aktionen des Benutzers zu registrieren und daraus auf einer abstrakteren Ebene zu schließen welcher Beschäftigung er nachgeht ist eine Frage, der derzeit nachgegangen wird (siehe u. a. [23, 28, 30]). Die Vision einer Anwendung, die dem Benutzer stets genau die Funktionen anbietet, die dieser in seiner aktuellen Situation benötigt, ist allerdings, bedingt durch die dem Menschen inhärente Spontanität, nicht realisierbar. Mehr noch, Applikationen dieser Bauart sollten diese sogar berücksichtigen und den Benutzer durch eine auf Grund gemachter Vorhersagen gewählten Aktion nicht behindern [31].[1]

[1] Es gibt noch weiteres beim Entwurf von Anwendungen welche sich Kontextvorhersage zu nutze machen zu beachten. Einen ersten Überblick gibt es in [31].

Unabhängig davon ist die Anpassung der Applikation an das Verhalten der Benutzer ein erstrebenswertes Ziel. Dient es zum einen der Erhöhung des Komforts des Benutzers, kann es zum anderen aber auch – richtig eingesetzt – durch monetäre Interessen motiviert sein. Mozer *et al.* [36] haben beispielsweise gezeigt, dass eine auf neuronalen Netzen basierende, adaptive Heizungssteuerung zu einem geringeren Energieverbrauch führen kann.[2] In einer Zeit weltweit steigender Energiepreise könnte dies für den Anwender durchaus ein lukratives Geschäft sein. Neben diesem finanziellen Aspekt eröffnet sich dadurch die Perspektive, dass auch das mobile Gerät durch solche Applikationen Energie sparen und somit die eigene Laufzeit signifikant steigern könnte.

1.2 Szenario und Ziel

Diese Arbeit verfolgt zwei Ziele: Zum einen soll ein Demonstrator entworfen werden, der die Fähigkeiten der am European Media Lab mitentwickelten, kontextsensitiven MUSIC Middleware aufzeigen kann. Zum anderen soll die allgemeine Problemstellung der Kontextvorhersage sowie Algorithmen zu deren Lösung in Bezug auf den Demonstrator und dem ihn zu Grunde liegende und nachfolgend beschriebene Szenario untersucht werden.

Der Demonstrator soll eine Arbeitsoberfläche für den Benutzer simulieren. Je nach Arbeitssituation, in der sich dieser befindet, sollen in dem Kontext passende Anwendungen präsentiert werden, die helfen die Aufgabe, die der Nutzer verfolgt, zu erledigen. Diese Idee wird durch ein Wiki realisiert in dem spezielle Seitenelemente (sogenannte *Panels*) neben den eigentlichen Inhalten dafür zuständig sein werden, kleine Helferapplikationen anzuzeigen. Würde ein deutschsprachiger Nutzer beispielsweise in dem Wiki ein englischsprachiges Dokument lesen, könnte der Demonstrator einen Übersetzungsservice anbieten.

Die Middleware dient der Entwicklung kontext-sensitiver und adaptiver Applikationen. Das Szenario ist allerdings nicht im eigentlichen Einsatzgebiet von MUSIC angesiedelt. Während die Middleware dazu gedacht ist auf mobilen Geräten Verwendung zu finden, wird sie im Rahmen dieser Arbeit innerhalb einer Webapplikation eingesetzt werden. Der Grund für diese Entscheidung liegt im frühen

[2] Weitere Beispiele für Szenarien, in denen Kontextvorhersage angewandt wird, werden in [31] genannt.

Entwicklungsstadium der Middleware und der damit einhergehenden mangelnden Erprobung auf mobilen Geräten.

1.3 Gliederung dieser Arbeit

Die Grundlage der Betrachtung der Kontextvorhersage bildet die Untersuchung der zu diesem Themengebiet verfügbaren Literatur (Kapitel 2). Anschließend werden in Kapitel 3 verschiedene Algorithmen zur Kontextvorhersage betrachtet und evaluiert. Dazu wird das Problem formal definiert (Abschnitt 3.1) bevor in den folgenden Abschnitten die Algorithmen beschrieben werden. Abschließend werden diese evaluiert werden (Abschnitt 3.7).

Danach verlagert sich der Fokus der Arbeit von der theoretischen Betrachtung der Kontextvorhersage auf den erstellten Demonstrator. Die vorgegebene Plattform auf der dieser arbeitet, wird in Kapitel 4 vorgestellt. Ein besonderes Augenmerk wird dabei auf die verwendete Music Middleware gelegt (Abschnitt 4.3). In Kapitel 5 wird die Architektur des Demonstrators und die Entscheidungen, auf denen diese basiert, genauer beleuchtet. Darauffolgend wird in Kapitel 6 die implementierte Lösung beschrieben. Dazu wird auch insbesondere auf die Strukturierung des Demonstrators in einzelne Osgi Bundles eingegangen (Abschnitt 6.1). In Kapitel 7 wird die Arbeit dann zusammengefasst und ein Ausblick auf weitere offene Fragen gegeben.

Kapitel 2

Relevante Literatur

In diesem Kapitel widmet sich diese Arbeit zunächst der Erkundung der für das Themengebiet der Kontextvorhersage relevanten Literatur. Zunächst wird sich dabei der Definition von Kontext gewidmet, bevor verschiedene Wege vorgestellt werden diesen zu klassifizieren. Nachdem diese Grundlagen erläutert wurden, wird die Literatur, die zum Thema Kontextvorhersage bislang veröffentlicht wurde, genauer untersucht.

2.1 Definition von Kontext

Kontext ist ein im allgemeinen Sprachgebrauch weitverbreiteter Begriff. Aber auch in der wissenschaftlichen Beschäftigung mit der Informatik hat er seit nahezu Anbeginn Bestand. Heutzutage wird in der formalen Informatik von *kontext*-sensitiven und *kontext*-freien Grammatiken gesprochen, in Betriebssystemen gibt es den Begriff des *Kontext*wechsels und im Entwurf grafischer Schnittstellen werden häufig *kontext*-sensitive Menüs erzeugt. Der Kontextbegriff, der dieser Arbeit zu Grunde liegt, hat mit diesen Ideen allerdings nichts oder nur am Rande zu tun, sondern stammt aus dem Themengebiet des „context-aware computing“.

Auf Grund der weiten Verbreitung des Begriffs Kontext existieren zahlreiche Definitionen unterschiedlicher Autoren. Den Anfang machten Schilit *et al.* [47]:

> „Three important aspects of context are: where you are, who you are with, and what resources are nearby“

Kontext wurde damals häufiger über Aufzählungen definiert, beispielsweise zählten Brown *et al.* [10] unter anderem den Ort, die Temperatur sowie die Zeit zum Kontext. Ryan *et al.* [46] sowie Dey [13] nutzten ähnliche Definitionen. Der Nachteil an diesen

ist, dass sie sich in der Regel nur schwer generalisieren lassen und demzufolge stark von dem zu Grunde liegenden Problem abhängig sind.

Brown [9] ging das Problem anders an und definierte Kontext generell als die Umgebung die der Computer kennt. Diese und ähnliche Definitionen [24, 38, 44, 52] – auch wenn sie einen guten Ansatz bilden – basieren auf ebenfalls wenig konkreten Begriffen wie „Umgebung“ oder „Situation“ oder nutzen diese synonym. Die mangelnde Exaktheit verhindert daher die universelle Anwendbarkeit dieser Definitionen.

Die am weitesten verbreitete und akzeptierte Definition lieferten Dey und Abowd [14]:

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

Diese Definition ist es letztlich auch, die dieser Arbeit zu Grunde liegt, denn sie ist flexibel, umfassend aber dennoch exakt. Sie erlaubt beispielsweise von Applikation zu Applikation zu unterscheiden, was zum Kontext gehört; ist für eine Applikation von Bedeutung ob sich eine zweite Person im Raum befindet und gehört damit nach der obigen Definition zum Kontext, kann dies für eine andere völlig unerheblich sein, wodurch diese Information für sie kein Kontext mehr ist. Diese Flexibilität besitzt die von Brown [9] vorgestellte Definition nicht.

Im Rahmen dieser Arbeit wird eine im Vergleich zur Definition leicht geänderte Nomenklatur verwendet, die allerdings im weiteren Verlauf auch im Kontextmodell[1] der Music Middleware Verwendung findet. Kontext ist in dieser Arbeit eine Sammlung von *Kontextelementen*, welche einen Wert besitzen können und sind damit eher das was Dey und Abowd mit „Context“ bezeichnet haben. Als weitere Ergänzung fasst der Begriff der Entität in dieser Arbeit nicht nur Personen, Orte und Objekte, sondern auch abstrakte Datenwerte, welche beispielsweise aus anderen Kontextelementen gewonnen werden können, zusammen. Diese Erweiterung gibt einen Hinweis darauf, dass Kontextelemente in verschiedene Kategorien zusammengefasst werden können.

[1] Der Begriff Kontextmodell bezeichnet die Repräsentation des Kontextes in Applikationen. Eine einführende Übersicht über verschiedene Ansätze bietet [4].

2.2 Kontextkategorisierung

Bereits Schilit *et al.* [47] unterteilten Kontext in „computing“, benutzer-basierten und physikalischen Kontext. Physikalischer Kontext sind Werte wie die Umgebungstemperatur, also physikalische, von Sensoren messbare Größen. Ressourcen wie Bandbreite oder Latenz einer Netzwerkverbindung oder die Präferenzen eines bestimmten Benutzers, fallen in dieser Einteilung hingegen in den Bereich des „computing“ beziehungsweise in den des Benutzerkontextes. Chen und Kotz [11] schlugen vor den zeitlichen Kontext, der Uhrzeit oder Jahreszeit umfasst, zu dieser Kategorisierung hinzuzufügen.

Eine Unterteilung in *primären* und *sekundären* Kontext wurde von Dey und Abowd [14] vorgeschlagen. Primäre Kontextelemente umfassen den Ort, die Zeit, die aktuelle Aktivität sowie die Identität des Benutzers, hingegen sind sekundäre alle daraus abgeleiteten. Eine ähnliche Kategorisierung nutzen Hofer *et al.* [22], indem sie die Kontextelemente in *physische* und *logische* zusammenfassen. Erstere bezeichnen gemessene, physikalische Größen, wohingegen letztere diese Daten durch semantische Informationen erweitern. Grundsätzlich ist diese Unterteilung intuitiv und deswegen weit verbreitet [4], so haben beispielsweise Prekop und Burnett [41] die Klassen zwar mit *extern* beziehungsweise *intern* benannt, repräsentieren allerdings die gleiche Idee.

Anders teilen Brown *et al.* [10] die Kontextelemente ein: Sie unterscheiden zwischen *kontinuierlichen* und *diskreten* Kontext. Bezogen ist dies auf den Zeitpunkt zu dem die Veränderungen des Kontextes registriert wird und nicht auf den Wert. Viele Elemente sind dadurch diskret, da viele Sensoren den gemessenen Wert nur zu diskretisierten Zeitpunkten (zum Beispiel alle 30 Sekunden) weitergeben, selbst wenn sich die eigentlich gemessene Größe kontinuierlich ändert. Der Unterschied zwischen *diskret-wertigen* und *kontinuierlichen* Kontext*werten* ist in dieser Arbeit wichtig, da der Fokus auf diskret-wertigen Elementen liegt und kontinuierliche nicht weiter betrachtet werden. Der Grund dieser Einschränkung liegt im Bezug auf das Szenario und wird in Abschnitt 3.2 genauer dargelegt.

Schmidt *et al.* [48] zeigen eine weitere Möglichkeit auf, indem sie zwischen *expliziten* und *impliziten* Kontext unterscheiden. Gemeint sind damit Elemente die direkt vom Benutzer abgefragt beziehungsweise automatisch gewonnen werden. Beispielsweise könnte die Erkennung der in einem Raum anwesender Personen

durch die Informationen eines Logindialogs gewonnen werden (explizit) oder durch automatische Bilderkennung auf Videodaten (implizit).

In vielen Veröffentlichungen ist zudem die Rede von *hohen* und *niedrigen* Kontextelementen (beispielsweise in [1, 4, 49]). Diese Bezeichnung, obwohl eine genaue Definition nicht gefunden werden konnte, zielt zumeist auf den Abstraktionsgrad des repräsentierten Kontextes ab. Die Temperatur, die ein Sensor misst, ist somit niedriger Kontext, während die aktuelle Aufgabe die der Benutzer ausführt, als hoher Kontext bezeichnet würde.

2.3 Kontextvorhersage

Eine umfassende Arbeit auf dem Gebiet der Kontextvorhersage ist die Dissertation von Mayrhofer [30], in der ein Framework zur Kontextvorhersage beschrieben wird. In Rahmen der Entwicklung wurden zwar bereits Algorithmen evaluiert, allerdings wird dort auch betont, dass die Evaluation auf nur einem Datensatz, wie sie dort durchgeführt wurde, nicht geeignet sei, um allgemeine Schlüsse aus dem Ergebnis ziehen zu können. Zudem ist die in der Arbeit vorhergesagte Kontextklasse eine aus den gemessenen Kontextwerten abstrahierte, wohingegen in dieser Arbeit der Fokus zunächst auf der Vorhersage einzelner Elemente liegt.

Es existieren einige Projekte, die sich mit intelligenten Hausumgebungen beschäftigen und in denen der Frage nachgegangen wird, was der Bewohner als nächstes tun wird. Das „Adaptive House“ Projekt[2] [34] versucht beispielsweise Geräte zu entwickeln, die das Benutzerverhalten mit Hilfe neuronaler Netze lernen. Allerdings sind diese immer spezifisch auf ein konkretes Problem (zum Beispiel das einer adaptiven Lichtsteuerung [35]) zugeschnitten und lassen sich daher schlecht auf einen allgemeinen Kontext ausweiten. Viele Arbeiten zu diesem Thema beschäftigen sich mit spezifischen Kontextelementen – beispielsweise dem Ort – und entwickeln daran angepasste Lösungen, die daher schlecht oder gar nicht zu generalisieren sind [23, 39, 40]. Das „MavHome“ Projekt[3] hingegen kombiniert verschiedene Algorithmen, um die nächsten Aktionen des Benutzers vorherzusagen und entsprechend darauf zu reagieren [12]. Der dort unter anderem entwickelte

[2] Projektwebseite: `http://www.cs.colorado.edu/~mozer/house/`
[3] Projektwebseite: `http://ailab.eecs.wsu.edu/mavhome/index.html`

und eingesetzte „Active Lempel-Ziv“ Algorithmus wird auch in dieser Arbeit näher betrachtet.

Unlängst haben Sigg *et al.* [49] eine weitere Architektur zur Kontextvorhersage vorgestellt und dabei ARMA-, Regel- und Markov-basierte Algorithmen evaluiert. Weiterhin kommen sie zu dem Schluss, dass weniger abstrakte Kontextelemente für die Vorhersage mit Hilfe dieser Verfahren von Vorteil sein können.

Das mathematische Problem der statistischen Zeitreihenvorhersage zielt in eine ähnliche Richtung und einige Verfahren, die in der Literatur betrachtet werden, basieren darauf (siehe zum Beispiel [30] für Details). Die entwickelten Algorithmen haben allerdings eine nicht zu vernachlässigende Komplexität, da sie auf Zeitreihen mit kontinuierlichen Werten angewendet werden, weswegen dieser Ansatz hier nicht weiter verfolgt wurde, zumal eine detaillierte Betrachtung über den Rahmen dieser Arbeit hinaus führen würde.

Kapitel 3

Kontextvorhersage

In diesem Kapitel werden die zu untersuchenden Algorithmen sowie deren Evaluation beschrieben. Dazu bedarf es zunächst einer, im Rahmen dieser Arbeit entwickelten, formalen Fassung des Problems.

3.1 Formale Definition

Dazu ist zunächst eine Repräsentation des Kontextes, der Kontextelemente und ihrer Werte erforderlich, die auch später zur Darstellung der Algorithmen genutzt wird.

Definition 1. *Sei $K \in \Xi$ ein Kontextelement aus einem Gesamtkontext Ξ und $\mathcal{K}_K$ die Menge seiner möglichen Werte, dann gibt die Folge $(k_i)_{i=0,1,\ldots}^{K}$ mit $k_i \in \mathcal{K}_K$ den Wert eines Elements K zum Zeitpunkt i an.*[1]

Augenscheinlich wird die Zeit durch diese Definition diskretisiert, was in dem vorgegebenen Szenario zweckmäßig ist, denn von entscheidendem Interesse ist der nächste Kontextwert, weniger wann dieser auftreten wird. An dieser Stelle sei darauf hingewiesen, dass die Diskretisierung keinesfalls mit äquidistanten Abständen einher gehen muss. Die nächste Definition zeigt die Anforderungen an eine Vorhersagefunktion auf.

Definition 2. *Sei $K_j \in \Xi$, $\left(k_{ij}\right)_{i=0,1,\ldots} = \left(k_i\right)^{K_j}$ sowie $\left(\xi_l\right)_{l=0,1,\ldots} = \left(k_{l,0} \ldots k_{lm}\right)^T$ mit $m = |\Xi|$, dann sind die Funktionen ϕ_n* ideale *Vorhersagefunktionen genau dann, wenn für alle n $\xi_{n+1} = \phi_n\left(\xi_0, \ldots, \xi_n\right)$ gilt.*[2]

[1] Der Index K an der Wertemenge $\mathcal{K}$ wird nur hinzugefügt, sofern der Bezug unklar ist.

[2] Der formalen Vollständigkeit halber sei angemerkt, dass genaugenommen eine ideale Vorhersagefunktion nur für *einen* bestimmten Kontext Ξ definiert ist. Eine *perfekte* Vorhersagefunktion wäre für alle Kontexte eine ideale Vorhersagefunktion.

Der Ausdruck (k_{ij}) repräsentiert die Folge der Werte (k_i) des j-ten Kontextelements K_j. Demzufolge repräsentiert jeder Vektor ξ_i die Werte aller Kontextelemente zu einem Zeitpunkt i, indem der j-te Eintrag ξ_{ij} den Wert des j-ten Kontextelements K_j angibt.[3] Die Folge (ξ_i) ist dementsprechend die zeitliche Abfolge aller Kontextwerte. Eine Funktion ist also eine ideale Vorhersagefunktion, sobald sie aus einer Aufzeichnung der letzten Kontextwerte den folgenden berechnen kann. Entsprechend ist es natürlich das Ziel, eine (berechenbare) Funktion f zu finden, die für jedes (endliche) n möglichst gut ϕ_n approximiert.[4]

3.2 Algorithmen zur Kontextvorhersage

In diesem Abschnitt werden zunächst die Anforderungen an und die Struktur der folgenden, zu evaluierenden Verfahren erläutert:

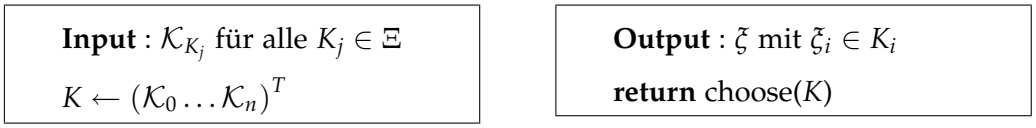
1. Statischer und sequentieller Majority-Vote
2. Ein auf N-Grammen basierender Algorithmus
3. Hidden Markov Model
4. Active Lempel-Ziv

Teilweise wurden diese bereits an anderer Stelle untersucht [12, 30]. Allerdings wurde eingangs schon erwähnt, dass die Ergebnisse dieser Evaluationen im Allgemeinen nicht gelten müssen (siehe Abschnitt 2.3). Dennoch, die gewählten Verfahren haben sich auch dort als gut geeignet herausgestellt.

Die im Folgenden dargestellten Anforderungen an die Algorithmen ergeben sich aus dem Szenario. Wie weiter oben bereits erwähnt, werden ausschließlich kategorische Werte betrachtet, denn die vorhandenen Kontextelemente – beispielsweise die aktuelle Seite des Benutzers, der Benutzername oder die Sprache des Benutzers – sind diskretwertig. Weiterhin ist der Demonstrator eine Webapplikation

[3] Die Notation ist an dieser Stelle vielleicht ein wenig verwirrend, denn auch die einzelnen Elemente werden über den Index adressiert. Im Nachfolgenden wird aber eindeutig sein, worauf sich ein Index bezieht.

[4] Auch hier sollte bemerkt werden, dass die Funktion f für möglichst viele Kontexte die idealen Vorhersagefunktionen gut approximieren sollte.



(a) (b)

Abbildung 3.1: Initialisierung (a) und Vorhersage (b) eines auf Zufall basierenden Algorithmus

und als solche ist die Zeit, die das Vorhersageverfahren braucht, von entscheidender Wichtigkeit. Ein Benutzer wird nur sehr ungern mehrere Sekunden auf die aufgerufene Webseite warten wollen. Gleichzeitig entstehen durch die Bewegung des Benutzers auf der Webseite ständig neue Kontextdaten, so dass es sich anbietet, einen inkrementell lernenden Algorithmus zu verwenden, um die Vorhersagegüte zu verbessern anstatt sie ungenutzt verfallen zu lassen. Zusätzliche Nachteile der nicht inkrementell arbeitenden Verfahren ergeben sich aus der Menge der während der Lernphase zur Verfügung stehenden Daten. Diese sind in ihrem Umfang meist so stark begrenzt, dass diese Verfahren auf dieser Menge schnell den inkrementellen Varianten, die mit der Zeit auf immer mehr Daten zurückgreifen können, unterlegen sind.

Die vorgestellten Verfahren werden zur Erläuterung stets in drei Schritten dargestellt. Der erste ist die Initialisierung (`init`), die die Übergabe arbiträrer Parameter nutzt, um die internen Datenstrukturen und Variablen zu erzeugen. Darauffolgend wird die eigentliche Vorhersage getätigt (`predict`), bevor dem Algorithmus der als nächstes auftretende Wert präsentiert wird (`present`). Zur Verdeutlichung werden in Abbildung 3.1 diese Schritte an einem Verfahren demonstriert, das stets ohne etwas zu lernen einen zufälligen Wert zurück gibt. Die Initialisierung (Abbildung 3.1a) nimmt die Mengen aller Werte aller Kontextelemente entgegen, die als Ergebnis zurückgegeben werden dürfen und speichert diese in der Variable K. Aus diesen wird während der Vorhersage (Abbildung 3.1b) je ein Wert pro Element zufällig ausgewählt. Der daraus resultierende Gesamtvektor bildet die Vorhersage. Die Funktion, der der nächste Kontextwert präsentiert würde, bleibt bei diesem Algorithmus leer.

Zu einer Algorithmenbetrachtung gehört die Angabe der theoretischen, asymptotischen Laufzeit- sowie Speicherkomplexität [27]. Darin wird $n = |\Xi|$ für die Anzahl der verschiedenen Kontextelemente benutzt werden, die in den meisten Anwendungsfällen gering genug ist, dass auch eine lineare Abhängigkeit nicht zwangsläufig problematisch ist. Die Anzahl der bislang gesehenen Kontextwerte m ist in dieser Hinsicht allerdings unvorteilhafter, denn in dem gewählten Szenario soll das Vorhersageverfahren potenziell sehr lange laufen und so ist eine Unabhängigkeit von m wünschenswert. Die letzte entscheidende Größe ist $C = \max\{|K| \, | K \in \Xi\}$, die die Kardinalität der größten Wertemenge aller Kontextelemente angibt. Diese Zahl ist durch die zuvor gegebene Definition nicht beschränkt und kann insbesondere auch unendlich groß werden, so dass eine Abhängigkeit von C nicht sehr erstrebenswert ist. Es wird sich allerdings zeigen, dass eine theoretische Abhängigkeit von C in der Praxis durchaus zu handhaben ist.

Für das auf Zufall basierende Verfahren liegt die obere Schranke der Speicherkomplexität in $\mathcal{O}(nC)$, da während der Initialisierung alle Wertemengen aller Kontextelemente gespeichert werden. Dadurch ist augenscheinlich, dass dieser Algorithmus für große Werte von C nicht mehr ausführbar ist. Durch eine geschicktere Implementierung mit einer dynamischen Datenstruktur wäre es zwar nicht möglich den theoretischen Verbrauch zu senken, allerdings würde der tatsächlich genutzt Speicher nur von der Anzahl der *präsentierten* Kontextwerte abhängen.[5] Das Abspeichern während der Initialisierung führt dementsprechend auch zu einer Laufzeitkomplexität von $\mathcal{O}(nC)$ dieser Methode. Die der Vorhersage liegt allerdings nur in $\mathcal{O}(n)$, da für jedes Kontextelement nur ein Wert zufällig gewählt werden muss. Die Methode, der die Werte präsentiert werden, liegt offensichtlich in $\mathcal{O}(1)$.[6]

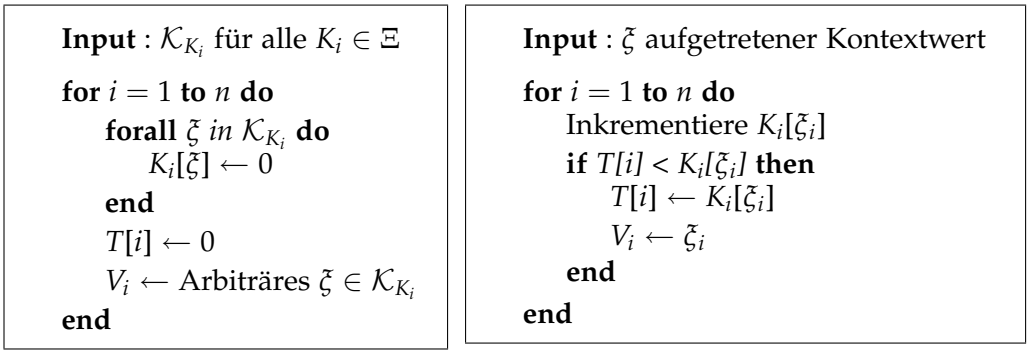


Abbildung 3.2: Initialisierung (a) und Präsentationsfunktion (b) des sequenziellen Majority Vote Algorithmus.

3.3 Statischer und sequenzieller Majority Vote

Der englische Ausdruck „Majority Vote“ könnte im deutschen etwa durch „Wahl der Mehrheit“ übersetzt werden und bezeichnet ein Verfahren, bei dem das Objekt ausgewählt wird, für welches die Mehrheit der Wähler gestimmt hat. Auf dieser Basis arbeitet die als erste vorgestellte Prozedur, für dessen Evaluation sich die Frage stellte, wie das Objekt mit dem häufigsten Vorkommen ermittelt werden sollte. Einerseits existiert die Möglichkeit den Evaluationsdatensatz vorher zu untersuchen und mit genau den Wert den Algorithmus zu initialisieren, der am häufigsten darin vorkommt. Dieser Ansatz wird in dieser Arbeit als *statischer* Majority Vote bezeichnet, da stets derselbe Wert vorhergesagt wird. Andererseits bedeutet diese Idee aber, vorher den gesamten Datensatz kennen zu müssen, was in der Praxis nicht möglich ist. Deswegen verfolgt der *sequenzielle* Majority Vote den Ansatz genau den Kontextwert vorherzusagen, der dem Algorithmus bis dato

[5]Die Schranke von $\mathcal{O}(nC)$ bleibt damit bestehen, denn in einem (theoretisch) unendlichen langen Lauf würden alle C Werte dem Algorithmus präsentiert werden.

[6]Die für die Praxis relevante Variante muss die Präsentationsmethode die Datenstruktur gegebenenfalls um ein neues Element erweitern und überprüfen, ob dieser Wert schonmal gesehen wurde. Dies lässt sich in $\mathcal{O}(n \log C)$ lösen, wobei C auch hier nur die theoretische obere Schranke ist und die Anzahl der gesehenen Werte eingesetzt werden muss. Die Initialisierung läge dann in $\mathcal{O}(1)$.

Input : ξ aufgetretener Kontextwert

for $i = 1$ **to** n **do**
    $K \leftarrow$ Element das durch $window_i$ repräsentiert wird.
    Inkrementiere $K[\xi_i]$
    Füge ξ_i an $window_i$ an
    Entferne wenn nötig das erste Element aus $window_i$
end

Abbildung 3.3: Präsentationsfunktion des N-Gramm Algorithmus

am häufigsten präsentiert wurde. Offensichtlich geben beide Verfahren auf großen (aber endlichen) Datensätze ab einem Punkt stets das gleiche Ergebnis zurück.

Die Komplexität der statischen Version ist nicht sonderlich aufregend. Sowohl die Speicher-, als auch die Laufzeitkomplexität der Methoden liegt in $\mathcal{O}(1)$ (Präsentation und Vorhersage) oder $\mathcal{O}(n)$ (Initialisierung), da der Kontextwert, der n Elemente umfasst und stets zurückgegeben wird, gespeichert werden muss. Interessanter ist hingegen die Betrachtung der sequenziellen Variante. In der in Abbildung 3.2a gezeigten Initialisierung ist ersichtlich, dass die Datenstruktur K auf eine Größe von $\mathcal{O}(nC)$ anwächst und somit die Speicherkomplexität dominiert. Allerdings kann diese auch hier, ähnlich wie beim bereits erwähnten Zufallsalgorithmus, durch eine dynamischere Datenstruktur auf einen Wert, der in der Praxis nur von den gesehenen Kontextwerten abhängt, vermindert werden. Für die Laufzeitkomplexität der Initialisierung, die auch in $\mathcal{O}(nC)$ liegt, gilt das analog. Die Methode, der die auftretenden Kontextwerte präsentiert werden (Abbildung 3.2b), zählt in der Struktur K wie oft ein Wert für jedes Kontextelement bereits aufgetreten ist. Der dabei am häufigsten aufgetretene Wert wird in der Variable V gespeichert, während der entsprechende Zähler in T behalten wird. Die Laufzeitkomplexität dieser Methode liegt also in $\mathcal{O}(n)$, während die Vorhersage sogar in $\mathcal{O}(1)$ liegt, da nur V zurückgegeben werden muss.

for $i = 1$ **to** n **do**
    $K \leftarrow$ Element das durch $window_i$ repräsentiert wird.
    $\xi_i \leftarrow$ Element mit höchstem Wert in K
end
return ξ

Abbildung 3.4: Vorhersagefunktion des N-Gramm Algorithmus

3.4 N-Gramm Algorithmus

In der statistischen Sprachverarbeitung bezeichen N-Gramme eine Sequenz von N Worten [26, 29]. Der Ausdruck N-Gramm kann aber leicht auf eine Sequenz der Länge N einer beliebigen Grundmenge Σ verallgemeinert werden. Die Grundidee des vorgestellten Verfahrens ist, dass die Wertemenge $\mathcal{K}$ diese Menge darstellt und die Häufigkeit aller auftretenden N-Gramme über diesem Alphabet gespeichert werden. Eine Vorhersage kann dann gewonnen werden, indem die letzten $N-1$ aufgetretenen Werte betrachtet werden und genau der als Vorhersage zurückzugeben wird, der zusammen mit diesen das N-Gramm bildet, das bislang am häufigsten vorgekommen ist. Dies wird in dieser Form auch in der Sprachverarbeitung durchgeführt [26, 29].

In der Initialisierung des Algorithmus wird zunächst eine Datenstruktur angelegt, in der die Häufigkeit aller N-Gramme gezählt werden kann, sowie eine Variable *window*, in der die letzten $N-1$ gesehenen Werte gespeichert werden. Wird dem Verfahren ein neuer Wert präsentiert (Abbildung 3.3), wird dieser mit den in *window* enthaltenen Werten zu einem neuen N-Gramm zusammengesetzt, der entsprechende Zähler inkrementiert und die in der Fenstervariablen gespeicherten Daten aktualisiert. Während der Vorhersage (Abbildung 3.4) wird der Wert gesucht, der zusammen mit diesen das häufigste N-Gramm bildet.

Die Speicherkomplexität birgt, ähnlich den Majority Vote Algorithmen, auch hier das größere Problem, denn um die Häufigkeit aller N-Gramme zu speichern wird eine Speichermenge benötigt, die asymptotisch in $\mathcal{O}\left(nC^N\right)$ liegt, da für jede der N Stellen in jedem der n Worte maximal C mögliche Zeichen existieren. Durch geeignete Implementierung kann der Wert von C in der Praxis auf den der bislang

gesehenen Kontextwerte reduziert werden. Für die Laufzeitkomplexität gestaltet sich ein ähnliches Problem, denn das Vorhersagen läuft in $\mathcal{O}(nC)$, kann allerdings auf $\mathcal{O}(n)$ reduziert werden, sofern ein Mehraufwand von $\mathcal{O}(nC^{N-1})$ an Speicher in Kauf genommen werden kann. Das Präsentieren eines neuen Wertes liegt hingegen mit $\mathcal{O}(nN)$ im akzeptablen Bereich.

3.5 Hidden Markov Modell

Hidden Markov Modelle (Hmm) werden in vielen Bereichen des maschinellen Lernens verwendet, eine besonders weite Verbreitung haben sie allerdings sowohl in der Sprachverarbeitung, als auch in der Bilderkennung (siehe zum Beispiel [43]). Eine kurze Übersicht über die dem Hmm zu Grunde liegenden Ideen ist unumgänglich, um die Beschränkungen, die diese Methode mit sich bringt, zu verstehen. Obwohl die Form der folgenden Darstellung, von der in [7] abweicht, ist sie inhaltlich daran orientiert.

Definition 3. *Die Wahrscheinlichkeit, dass eine* Zufallsvariable X *den Wert x annimmt wird mit* $\Pr[X = x]$ *beschrieben. Tritt der Fall ein, dass eine weitere Zufallsvariable* Y *gleichzeitig den Wert y annimmt, wird dies durch* $\Pr[X = x, Y = y]$ *dargestellt.*

Um eine aufgeblähte Darstellung zu vermeiden, wird, soweit es eindeutig möglich ist, auf die Schreibweise $\Pr[x]$ zurückgegriffen, die eine Kurzform für $\Pr[X = x]$ ist. Weiterhin ist der Begriff der bedingten Wahrscheinlichkeit für das folgende unerlässlich.

Definition 4 (Bedingte Wahrscheinlichkeit)**.** *Die Wahrscheinlichkeit, dass eine Zufallsvariable* X *den Wert x annimmt, wenn bekannt ist, dass die Zufallsvariable* Y *den Wert y besitzt, wird mit* $\Pr[x|y]$ *dargestellt.*

Die bedingte Wahrscheinlichkeit, dass x eintritt – gegeben y – berechnet sich aus den Einzelwahrscheinlichkeiten zu:

$$\Pr[x|y] = \frac{\Pr[x, y]}{\Pr[y]}$$

Wichtig für die Notation eines Hmm ist ferner die einer Wahrscheinlichkeitsdichte.

Definition 5. *Die Funktion $p(x)$ heißt* Wahrscheinlichkeitsdichte *sofern folgendes erfüllt ist:*

$$p(x) \geq 0$$
$$\int p(x)\,\mathrm{d}x = 1$$
$$\Pr[\mathrm{Z} \leq z] = \int_{-\infty}^{z} p(x)\,\mathrm{d}x$$

Dem Hmm liegt ein stochastischer Prozess zu Grunde, der, wie der Name bereits impliziert, die Markov-Eigenschaft besitzt, dessen Wert also zum Zeitpunkt $n+1$ nur von vorherigen n Werten abhängt.

Definition 6. *Seien $\mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{D}$ Zufallsvariablen und $\zeta = (x_1 \cdots x_D)^T$ die Repräsentation der Werte dieser Variablen. Die Folge $\zeta_1, \ldots, \zeta_N$ hat die* Markov-Eigenschaft *wenn gilt*

$$p(\zeta_1, \ldots, \zeta_N) = \prod_{n=1}^{N} p(\zeta_N | \zeta_1, \ldots, \zeta_{n-1})$$

Hängt die Wahrscheinlichkeit sogar ausschließlich von den letzten k Beobachtungen ab, ergibt sich eine Markov-Kette *k*-ter Ordnung.

Ein Hmm besteht aus einer Menge von „versteckten“ Zuständen $\mathbf{Z}$, zwischen denen das Modell im zeitlichen Verlauf wechselt. Diese Abfolge wird durch eine Markov-Kette (meist erster Ordnung) modelliert und die daraus resultierenden Zustandsübergangswahrscheinlichkeiten in Form einer Matrix A angegeben, in der der Eintrag A_{jk} die Wahrscheinlichkeit des Übergangs in Zustand k zum Zeitpunkt i bezeichnet, gegeben das sich das System zum Zeitpunkt $i-1$ im Zustand j befindet.

$$A_{jk} = p(z_i = k | z_{i-1} = j)$$

Jeder Zustand z besitzt eine Emissionswahrscheinlichkeit $p(\zeta | z, \phi)$, die die Wahrscheinlichkeit, dass in diesem Zustand eine Beobachtung ζ gemacht wird, bezeichnet. Der Parameter ϕ beinhaltet etwaige, zusätzliche Parameter dieser Verteilung.[7]

[7] Handelt es sich bei der Emissionswahrscheinlichkeit beispielsweise um eine Gauss-Verteilung, beinhaltet ϕ die Varianz σ als auch den Erwartungswert μ.

Zur vollständigen Angabe eines Hmm gehört abschließend noch, in welchem Zustand sich das System zum Zeitpunkt o befindet. Auch dies wird stochastisch durch die Variable π angegeben, dessen k-tes Element $\pi_k = p(z_0 = k)$ die Wahrscheinlichkeit angibt, dass sich das System zu Beginn in Zustand k befindet.

Für eine gegebene Beobachtungssequenz $\Gamma = \{\zeta_1, \ldots, \zeta_N\}$ kann mit Hilfe des Hmm

$$p(\Gamma, \mathbf{Z}|\Theta)$$

berechnet werden, wobei $\Theta = \{\pi, A, \phi\}$ die Menge der Parameter ist, die es zunächst so zu optimieren gilt, dass der Gesamtausdruck maximal wird. Diese Aufgabe wird durch den Baum-Welch oder Forward-Backward Algorithmus erledigt [5, 43], der die zufällig initialisierten Parameter maximiert, indem er eine „expectation maximization“ (EM) Technik nutzt. Eine genauere mathematische Darstellung würde an dieser Stelle allerdings über den Rahmen dieser Arbeit hinausgehen, deswegen sei an dieser Stelle nochmal auf das Buch von Bishop [7] hingewiesen, das eine sehr detaillierte und mathematisch genaue Abhandlung des Themas beinhaltet.

Mit Hilfe dieses Verfahrens kann ein Hmm aber in der Regel nicht inkrementell arbeiten, da die Parameter vor einer Vorhersage festgelegt sein sollten und so bedarf es eines gesonderten Trainingsdatensatzes um dies zu bewerkstelligen. Erst danach ist eine Vorhersage möglich, indem der Wert vorhergesagt wird, der, ähnlich dem N-Gramm Verfahren, mit den bisher gesehenen Werten die höchste kumulierte Emissionswahrscheinlichkeit aufweist. Der Viterbi-Algorithmus [51] übernimmt mit Hilfe dynamischer Programmierung die Aufgabe, die Wahrscheinlichkeit für eine Beobachtungssequenz bei gegebener Zustands- und Parametermenge zu berechnen. Auch an dieser Stelle wird von einer detaillierteren mathematischen Behandlung abgesehen und auf [7] verwiesen.

Das Lernen mit Hilfe des Baum-Welch Algorithmus kann in $\mathcal{O}\left(N^2 l\right)$ liegen [45], dabei bezeichnet N die Anzahl der Zustände in $\mathbf{Z}$ und l die Kardinalität der Menge Γ. Die Komplexität des Lernens spielt für den späteren Betrieb allerdings eine untergeordnete Rolle, da dies nur einmal vor Beginn durchgeführt werden muss. Wichtiger ist die Laufzeitkomplexität des Vorhersagens, welche von dem Viterbialgorithmus abhängt, der in $\mathcal{O}\left(N^2 k\right)$ liegen kann [33], wobei k die Länge der Beobachtungssequenz darstellt. Da dieser Wert für n verschiedene Abfolgen berechnet werden muss, ist die Gesamtkomplexität in $\mathcal{O}\left(nN^2 k\right)$. Sofern die Beob-

achtungssequenz auf eine fixe Länge begrenzt werden kann, stellt dies durchaus eine akzeptable Größe dar.

Wie bereits erwähnt ist das HMM nicht inkrementell lernbar, es wurde dennoch genauer untersucht, da es in vielen ähnlichen Bereichen sehr gute Ergebnisse liefert. Zusätzlich musste während der Auswertung zunächst eine gute Größe für die Menge **Z** gefunden werden, da diese bereits vor dem Lernen festgelegt sein muss. Auf diese Aspekte wird in der Evaluation in Unterabschnitt 3.7.4 noch genauer eingegangen. Die Implementierung der Verfahren übernahm die Jahmm Bibliothek[8].

3.6 Active Lempel Ziv

Das Active Lempel-Ziv Verfahren (ALZ) wurde bereits im MavHome Projekt verwendet [12, 20] und basiert auf dem Lempel-Ziv Kompressionsverfahren (LZ78) [54]. Dieses komprimiert eine Zeichenkette $x_1, \ldots, x_n$ über einem Alphabet Σ, indem es diese in m Wörter $w_1, \ldots, w_m$ einteilt, so dass für jedes $j > 0$ gilt, dass das längste Präfix von w_j (d. h. alle Buchstaben des Wortes bis auf den letzten) gleich einem Wort w_i mit $1 \leq i < j$ ist [20, 54]. Auf Grund dieser Präfixeigenschaft, kann aus diesen Wörter nun leicht ein Trie erzeugt werden, welcher dann zum en- und dekodieren benutzt wird. Letzteres ist allerdings für das hier vorgestellte Verfahren unerheblich. Abbildung 3.5 zeigt, wie die Präfixeigenschaft erzeugt wird. Zu Grunde liegt eine Triestruktur, die für jeden Knoten einen Zähler bereithält, der im Originalverfahren zur Kodierung benötigt wird, im Fall des ALZ wird dieser allerdings anders ausgewertet. Zusätzlich beinhalten die Variablen *dictionary* und *phrase* alle bislang erzeugten Wörter beziehungsweise stets ein Wort welches bereits in *dictionary* enthalten ist. In dieser und den folgenden Abbildungen bezeichnet der Punkt das Konkatenieren einer Zeichenkette mit einem Zeichen. Die Präfixeigenschaft wird also hergestellt, indem in der Variable *phrase* die einzeln gelesenen Zeichen solange aneinandergefügt werden, bis das entstehende Wort im Wörterbuch nicht mehr existiert. Daraufhin wird es hinzugefügt und die Zähler im Trie werden entsprechend erhöht.

Um eine schöne, theoretische Eigenschaft des LZ78 und damit später auch des ALZ zu zeigen, wird der Begriff der „finite state predictability“ (FS Vorhersagbar-

[8] Verfügbar unter http://www.run.montefiore.ulg.ac.be/~francois/software/jahmm/

```
Data : dictionary, phrase ← null
repeat
    Warte auf nächstes Zeichen v
    if phrase.v in dictionary then
        phrase ← phrase.v
    else
        Füge phrase.v zu dictionary hinzu
        Inkrementiere Zähler für jedes Präfix von phrase
        phrase ← null
    end
until Forever
```

Abbildung 3.5: LZ78: Erzeugen der Wörter mit Präfixeigenschaft.

keit) eingeführt. Dieses Maß gibt für eine unendliche Sequenz den minimalen, asymptotischen Fehleranteil an, der mit einem auf endlichen Zuständen basierendem Verfahren erreicht werden kann. Der Wert liegt zwischen 0 für perfekte Vorhersagbarkeit und $1/2$ für perfekte Nicht-Vorhersagbarkeit. Eingeführt wurde dieses Maß von Feder *et al.* [15], die auch zeigen, dass ein universeller, auf Markov-Ketten basierender Algorithmus, der deren Ordnung mit einer „passenden" Rate variiert asymptotisch die FS Vorhersagbarkeit erreicht. In der gleichen Arbeit wird zudem auch gezeigt, dass mit Hilfe des LZ78 genau solch ein probabilistisches Modell erzeugt werden kann [15, 20].

Feder *et al.* [15] haben festgestellt, dass der LZ78 gegen diesen optimalen Wert nur sehr langsam konvergiert. Durch die Erhöhung der Zähler aller Präfixe aller Suffixe von *phrase*, anstatt nur für das durch *phrase* repräsentierte Wort, konnten Bhattacharya und Das [6] dieses Problem bereits beim Entwurf ihres „LeZi Update" Algorithmus lösen. Diese Lösung wurde in abgewandelter Form später von Gopalratnam und Cook [20] für ALZ übernommen. Neben dieser Änderung erkannten sie, dass der Zähler der Wörter, die über die Grenze ragen, in der *phrase* zurückgesetzt wird, nicht weiter erhöht wurde. Behoben werden konnte dies durch ein sich in der Größe dynamisch anpassendes Fenster, indem der Zähler für alle sich darin befindlichen Wörter mit der von Bhattacharya und Das vorgeschlagenen Technik zu erhöhen.

```
Data : dictionary, phrase, window ← null
max_length ← 0
Input : ξ Kontextwerte

for i = 1 to n do
    if phrase.v in dictionary then
        phrase ← phrase.v
    else
        Füge phrase.v zu dictionary hinzu
        Setze max_length wenn nötig neu
        phrase ← null
    end
    Füge v zu window hinzu
    if Länge von window > max_length then
        Entferne erstes Element von window
    end
    Inkrementiere Zähler aller Präfixe aller Suffixe von window
end
```

Abbildung 3.6: `present`-Funktion des ALZ

Abbildung 3.6 zeigt die resultierende Präsentationsfunktion, die den Trie für den gesehenen Kontextwert aktualisiert. Die Ähnlichkeit zum LZ78 ist offensichtlich. Die dynamische Größe des Fensters *window* wird über die Länge des längsten bisher registrierten Wortes festgelegt, denn diese ist es, die mit der Ordnung der Markov-Kette korreliert [20]. Aus dem erzeugten Trie wird der nächste Kontextwert vorhergesagt, indem die „Predictor by Partial Match“ (PPM) Technik benutzt wird. Ein PPM berechnet die Wahrscheinlichkeit, in dem es Markov-Ketten verschiedener Ordnungen miteinander in einer gewichteten Summe verrechnet [20]. Zur Vorhersage wird diese Summe für jedes Zeichen des Alphabets berechnet und das mit der höchsten Wahrscheinlichkeit wird zurückgegeben. Ein Zeichen z wird dafür mit jedem Suffix s (der Länge nach absteigend sortiert) von *window* konkateniert. Anschließend wird die Wahrscheinlichkeit berechnet, indem die Zähler der durch $s.z$ und s repräsentierten Trieknoten durcheinander geteilt werden. Dann wird ein Faktor berechnet mit dem das Ergebnis für das nächste Suffix multipliziert wird.

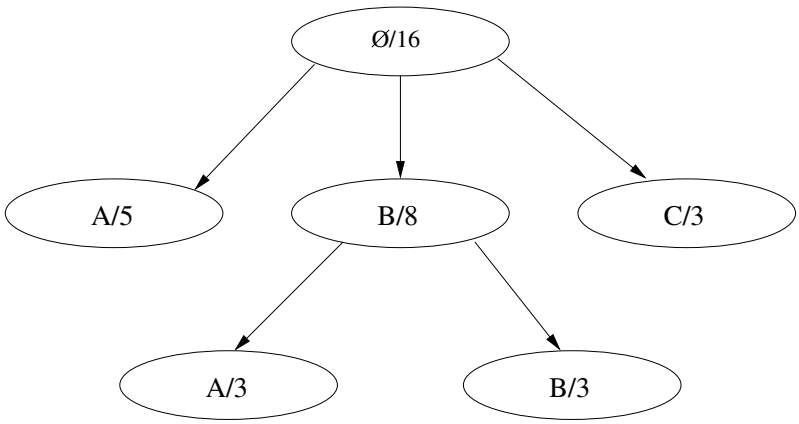


Abbildung 3.7: Beispiel eines von ALZ erzeugten Tries. Nach dem Schrägstrich sind die Werte des Zählers angegeben.

Dieser ergibt sich indem die Summe der Zähler von $s.x$ über alle Zeichen x aus dem Alphabet berechnet wird und erst von dem Zähler von s subtrahiert und anschließend durch ihn dividiert wird. Dieser Faktor bezieht sich insbesondere auch auf den Faktor des nächsten Suffixes. Seien x ein Buchstabe und $s_0, \ldots, s_n$ Suffixe, wobei s_0 das leere Wort darstellt, $v_n, \ldots, v_o$ die Verhältnisse zwischen $s.x$ und s, sowie $f_{n-1}, \ldots, f_0$ die Faktoren der einzelnen Ebenen, dann ergibt sich die Wahrscheinlichkeit p als Summe zu

$$p = v_n + f_{n-1}\left(v_{n-1} + f_{n-2}\left(\ldots\left(v_1 + f_0\left(v_o\right)\right)\ldots\right)\right)$$

Zur Verdeutlichung wird ein kurzes Beispiel diesen Vorgang erläutern. In Abbildung 3.7 ist ein Baum ersichtlich, der von ALZ hätte erzeugt werden können. Das längste gesehene Wort ist zwei Zeichen lang, was der Fenstergröße entspricht. Bestehe nun das Fenster aus *ab*, dann sind die Suffixe *b* und das leere Wort. Zunächst wird die Wahrscheinlichkeit berechnet, dass das Zeichen *a* als nächstes folgt. Das Verhältnis zwischen Wort und Suffix ist $3/8$, denn der Zähler von *ba* ist 3 und der von *b* ist 8. Der Faktor für das nächste Suffix beträgt $2/8$, denn die Summe der Zähler der Wörter *ba* und *bb* ist 6. Im Baum wird nun auf die nächsthöhere Ebene zurückgegriffen und das Verhältnis zwischen *a* und dem leeren Wort ist $5/16$. Da

for $i = 1$ **to** n **do**
  forall *Buchstaben b im Alphabet* **do**
    $P_i[b] \leftarrow 0$
    $F_i[b] \leftarrow 1$
  end
  phrase $\leftarrow$ *phrase*$_i$
  for $j =$ *Länge von* phrase $- 1$ **to** 0 **do**
    suffix $\leftarrow$ Suffix der Länger j von *phrase*
    parent $\leftarrow$ Trieknoten der *suffix* repräsentiert
    sum $\leftarrow$ Summe der Zähler aller Kinder von *parent*
    forall *Kinderknoten k von* parent **do**
      $P_i[k] \leftarrow f_i[k] \cdot \frac{\text{Zähler von } k}{\text{Zähler von } parent}$
      $f_i[k] \leftarrow f_i[k] \cdot \frac{\text{Zähler von } parent - sum}{\text{Zähler von } parent}$
    end
  end
end
$\xi \leftarrow$ An der Stelle i den Wert mit dem höchsten Wert in P_i
return ξ

Abbildung 3.8: Vorhersagefunktion des ALZ

es danach kein weiteres Suffix mehr gibt, braucht der nächste Faktor nicht mehr berechnet werden und die Gesamtwahrscheinlichkeit ergibt sich zu

$$\frac{3}{8} + \frac{2}{8}\left(\frac{5}{16}\right) = 0.453125$$

Für den Buchstaben b ergibt sich entsprechend eine Wahrscheinlichkeit von 0.5. Für den Buchstaben c ergibt sich $0/8 + 2/8 \cdot 3/16 = 0.046875$, da dieser mit einem Suffix b bislang noch nicht registriert wurde. Einen Hinweis darauf, dass es sich bei den errechneten Werten tatsächlich um Wahrscheinlichkeiten handelt, gibt die Tatsache, dass sie sich zu 1 aufsummieren. Auf Grund der Ergebnisse würde der Buchstabe b als der nächste vorausgesagt. In Abbildung 3.8 ist das gesamte Verfahren nochmal zusammengefasst.

Zu guter Letzt wird die Komplexität dieses Algorithmus betrachtet. Der Speicher ist dabei wieder das größere Problem, da diese von der Größe der Triestruktur und dessen Zunahmegeschwindigkeit abhängt. Gasieniec *et al.* [18] haben den LZ78 Algorithmus untersucht und kommen zu der Erkenntnis, dass die Triestruktur mit der Länge l, die in unserem Fall die Anzahl der gesehenen Kontextwerte ist, des zu komprimierenden Strings mit $\mathcal{O}(l \log l)$ wächst. Auch wenn der ALZ die Datenstruktur etwas schneller wachsen lässt, scheint diese Einschätzung auch hier zu gelten. Diese Problematik wird etwas genauer in Unterabschnitt 3.7.5 betrachtet. Auch die exakte Bestimmung der Laufzeitkomplexität der Funktionen ist schwierig. Beide hängen aber unmittelbar von der Länge des längsten Wortes ab, die gleichzeitig auch die Tiefe des Tries angibt.

3.7 Evaluation

Nachdem die einzelnen Algorithmen nun vorgestellt wurden, werden sie in diesem Abschnitt auf Basis der gestellten Anforderungen, aber auch auf Grund ihrer Vorhersagegüte untersucht.

3.7.1 Experimentdesign

Die durchgeführten Experimente gehen von einer Menge von aufgenommenen Kontextwerten eines Kontextelements aus. Diese werden dem Algorithmus nach und nach präsentiert. Vor jedem weiteren Wert wird zunächst die Vorhersage abgefragt und kontrolliert ob diese mit dem nächsten aufgezeichneten Wert übereinstimmt. Um die Güte zu quantifizieren, kam die 0/1-*Loss*-Funktion zum Einsatz, die auch von Mayrhofer [30] benutzt wurde.

$$\pi(T) = \frac{1}{n} \sum_{(\xi,k)\in T} l(\xi, k)$$

wobei n die Anzahl der bisher präsentierten Elemente ist und T eine Menge von Tupeln (ξ, k), in dem ξ der tatsächliche und k der vorhergesagte Kontextwert ist. Weiterhin ist $l(\xi, k)$ definiert als:

$$l(\xi, k) = \begin{cases} 0 & \xi = k \\ 1 & \xi \neq k \end{cases}$$

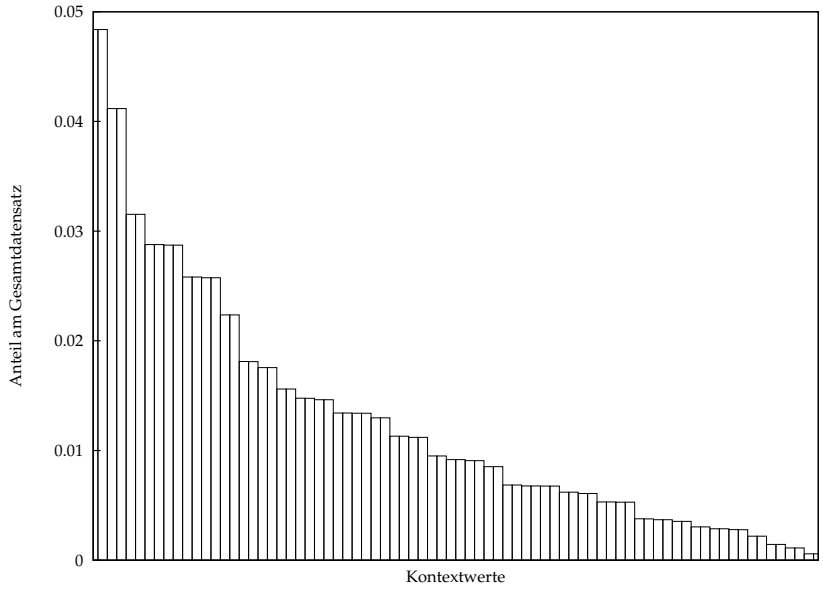


Abbildung 3.9: Prozentualer Anteil der einzelnen Kontextwerte im gesamten MavHome Datensatz. Die Klassen sind nach Auftrittshäufigkeit sortiert.

Die Funktion π gibt also den Anteil an fehlerhaft vorhergesagten Kontextwerten an. Die verschiedenen Algorithmen werden verglichen, indem zum einem der Wert der Lossfunktion, nachdem der gesamte Datensatz präsentiert wurde, ausgewertet wird, zum anderen wird aber auch der Verlauf betrachtet, um daraus Rückschlüsse auf das Verhalten der Vorhersage machen zu können. Die Verfahren wurden an Hand folgender Datensätze evaluiert:

1. Der größte verwendete Datensatz[9] ist beim MavHome Projekt entstanden. Der Datensatz wurde vom 3. Januar 2005 bis zum 20. Februar 2005 von einem in der mit Sensoren ausgestatteten Wohnung lebenden Menschen erzeugt. Für die Evaluation wurden die Daten der 38 verschiedenen Bewegungssensoren extrahiert. Diese zeigen jeweils an, ob in dem entsprechenden Bereich Bewegung stattgefunden hat oder nicht. Für jeden Bewegungssensor gibt es eine Unterscheidung zwischen „Bewegung“ und „keine Bewegung“, so dass

[9]Der Datensatz ist unter `http://ailab.eecs.wsu.edu/mavhome/datasets/mavpad2005.zip` verfügbar.

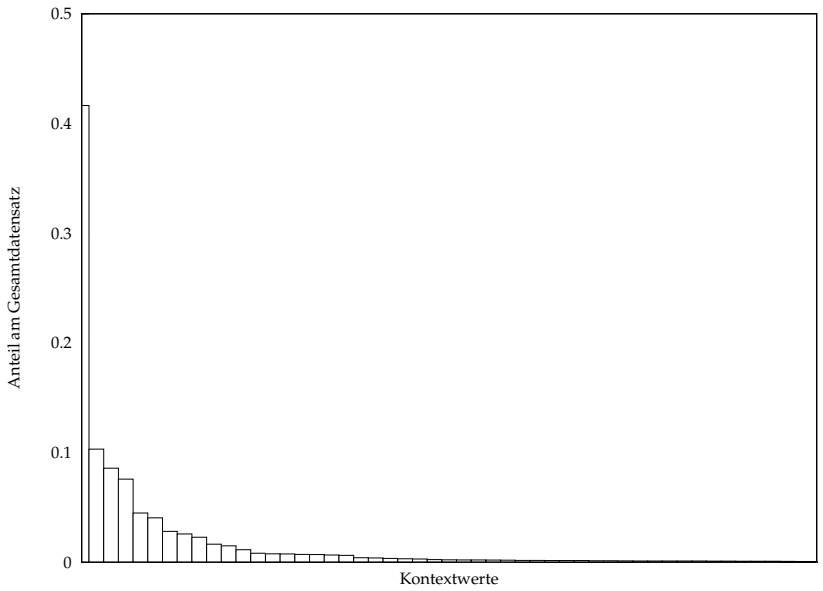


Abbildung 3.10: Prozentualer Anteil der einzelnen Kontextwerte im aktivem Fenster Datensatz von Mayrhofer. Die Klassen sind nach Auftrittshäufigkeit sortiert.

insgesamt 76 verschiedene Werte existieren. Der Datensatz umfasst insgesamt 331 428 Kontextwerte. In Abbildung 3.9 ist zu sehen, dass dieser Datensatz relativ ausgewogen verteilt ist, denn kein Zustand hat einen Anteil größer als 5 % an den Daten.

2. Des Weiteren wurden aus dem von Mayrhofer [30] erzeugten Datensatz drei einzelne, kategorische Datensätze extrahiert. Die Daten wurden mit einem Laptop gesammelt der über einen längeren Zeitraum mit 29 verschiedenen Sensoren ausgestattet war und alle 30 Sekunden den Wert der Kontextelemente gespeichert hat. Daraus wurden die Anzahl der in der Nähe befindlichen Rechner (Peers), das gerade aktivierte Fenster und der gerade aktive WLAN Access Point (AP) extrahiert und als separate Datensätze gespeichert. Alle haben einen Umfang von 87 637 aufgezeichneten Kontextwerten. Die Verteilung des aktiven Fenster Datensatzes ist in Abbildung 3.10 ersichtlich. Sie ist im Gegensatz zum MavHome Datensatz nicht sehr homogen und bereits die ers-

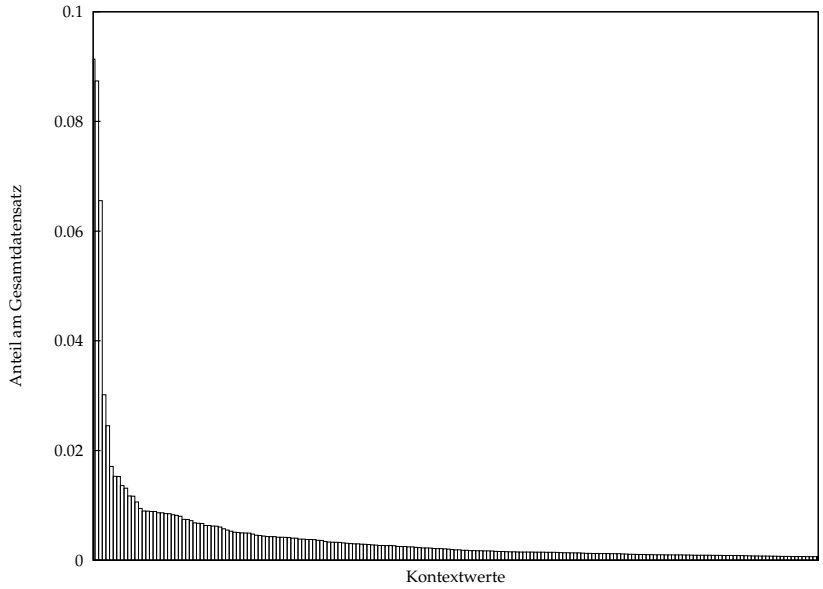


Abbildung 3.11: Prozentualer Anteil der einzelnen Kontextwerte im Wiki Datensatz. Die Klassen sind nach Auftrittshäufigkeit sortiert und es werden nur die 200 häufigsten Klassen gezeigt.

ten 4 Kontextwerte sind in über 80 % der Fälle aktiv. Die anderen Datensätze sind in Anhang A dargestellt, dort verhält es sich allerdings ähnlich.

3. Zu guter Letzt wurden in einem Zeitraum von etwa 3 Monaten 46 997 Zugriffe auf ein firmeninternes Wiki aufgezeichnet. Diese Zugriffe stammen von 17 verschiedenen Nutzern auf 585 verschiedenen Seiten. Dieser Gesamtdatensatz wurde nach einzelnen Nutzern aufgeteilt, so dass ein Vorhersagealgorithmus auch für einzelne Nutzer getestet werden konnte. Die Ergebnisse für einzelne Nutzer wichen aber nicht signifikant vom Gesamtdatensatz ab, was letztlich darin begründet ist, dass die Seite nicht sehr stark frequentiert war und deshalb wahrscheinlich meist nur ein Benutzer gleichzeitig auf den Seiten unterwegs war. Die Verteilung ist in Abbildung 3.11 sichtbar, auch hier ist ein Kontextwert mit fast 10 % bei 585 verschiedenen Möglichkeiten stark gehäuft, die Verteilung ist aber insgesamt dennoch homogener als die in den von Mayrhofer extrahierten Daten.

Datensatz	Majority-Vote Sequenziell	Statisch
MavHome	0.9675	0.9514
Wiki	0.9087	0.9086
Aktives Fenster	0.5837	0.5836
Aktiver AP	0.5124	0.5123
Aktive Peers	0.1868	0.1868

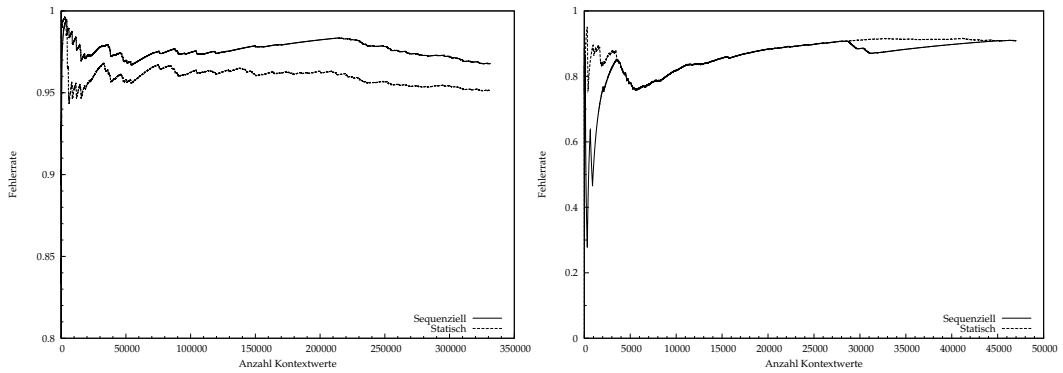
Tabelle 3.1: Ergebnisse der Evaluation der verschiedenen Majority Vote Algorithmen auf den 5 Datensätzen. Angegeben ist die Lossfunktion nach der Präsentation aller Kontextveränderungen.

Die Angaben zur Verteilung sind wichtig, um die Ergebnisse der Evaluation richtig einschätzen zu können. Sollte ein Algorithmus beim aktiven Fenster Datensatz eine Vorhersagegüte von 40 % erreichen, klingt das absolut gesehen zwar einigermaßen akzeptabel, ist jedoch, angesichts dessen, dass der statische Majority Vote die gleiche Leistung in $\mathcal{O}(1)$ erreicht, nicht mehr sehr beeindruckend.

3.7.2 Majority-Vote Algorithmen

In Tabelle 3.1 sind alle Ergebnisse der Majority Vote Algorithmen zusammengefasst. Wie zu erwarten war, hat die statische Variante genau den, dem Verhältnis der in der Verteilung am häufigsten vorkommenden Klasse entsprechenden Anteil korrekt vorhergesagt. Bemerkenswert sind die Werte für die sequenzielle Variante. Es wurde zwar erwartet, dass die Ergebnisse der beiden Varianten nah bei einander liegen, allerdings war auch die Idee naheliegend, dass die sequenziellen insgesamt besser abschneiden würden, da sie sich an die Daten anpassen können. Offenbar ist das nicht der Fall und der Effekt zu vernachlässigen.

In Abbildung 3.12 ist allerdings auch ersichtlich, dass die beiden Werte gerade zu Beginn der jeweiligen Experimente durchaus differieren können. Zu beachten ist, dass die Ordinate der beiden Bilder unterschiedlich skaliert ist. In Abbildung 3.12a sind die beiden Algorithmen zwar mehr oder weniger die ganze Zeit über gleichauf, die statische Variante ist aber zu jedem Zeitpunkt besser als die sequenzielle. Die Verläufe der beiden Kurven sind relativ ähnlich, es sieht so aus, als ob auch die



(a) (b)

Abbildung 3.12: Ergebnisse der Majority Vote Algorithmen auf dem MavHome Datensatz (a) und dem Wikidatensatz (b).

sequenzielle Variante öfter genau die Klasse vorhersagt, die auch die statische ausgibt (erkennbar an einem exakt gleichen Verlauf der beiden Kurven). Andererseits scheint die sequenzielle gerade am Anfang durch einige falsche Vorhersagen viel Boden zu verlieren. Eine ganz andere Situation stellt sich in Abbildung 3.12b dar, in der die sequenzielle Variante die statische zu Beginn ausstechen kann. Interessant ist ebenfalls der kleine Ausbruch auf Höhe der 30 000 Marke, sowie die Ungleichheit am Anfang. Die sequenzielle Ausführung neigt in diesen Bereichen oft dazu den vorhergesagten Wert zu ändern, wohingegen die statische Variante dies natürlich nicht tut. Dies hilft auf Strecken, bei denen die Klasse oft vorkommt (erkennbar durch den steilen Anstieg in der statischen Variante). Die in Abschnitt B.1 gezeigten Ergebnisse auf den Mayrhoferdatensätzen zeichnen ein ähnliches Bild. Zu Anfang unterscheiden sich die Ergebnisse, fangen allerdings schnell an gegen denselben Wert zu streben.

Da die Ergebnisse auf lange Sicht doch sehr ähnlich sind, die statische Variante aber deutlich schneller und einfacher zu berechnen ist, wird diese als Grundlage für die Evaluation der weiteren Algorithmen herhalten müssen. Insgesamt ist es natürlich so, dass beide Algorithmen auf den Datensätzen zu schlechte Ergebnisse erzielen um in der Praxis in Betracht gezogen zu werden.

N =	1	2	3	4	5
MavHome	0.6747	0.5492	0.4932	0.4949	0.5090
Wiki	0.1761	0.2325	0.2812	0.3369	0.3845
Aktives Fenster	0.1412	0.1514	0.1729	0.1999	0.2278
Aktiver AP	0.0005	0.0007	0.0009	0.0010	0.0012
Aktive Peers	0.0071	0.0078	0.0084	0.0107	0.0114

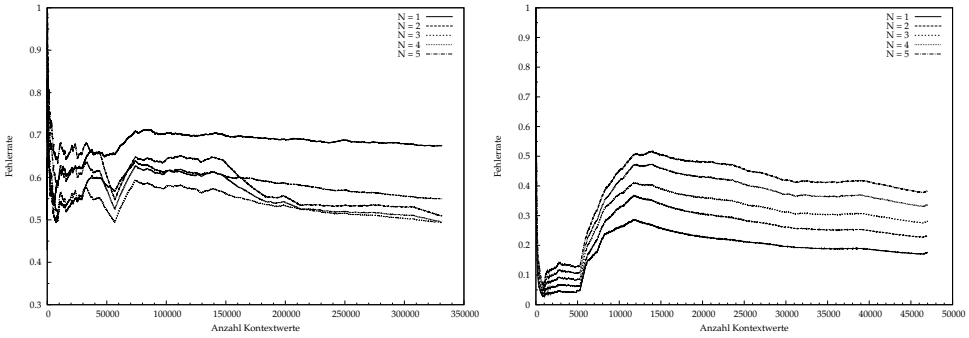
Tabelle 3.2: Ergebnisse des N-Gramm Algorithmus für verschiedene Werte von N

3.7.3 N-Gramm Algorithmus

Bei diesem Verfahren galt es zum einen die allgemeine Vorhersagequalität zu evaluieren, zum anderen aber auch zu überprüfen, welche Werte von N noch realisierbar sind und welche sich nicht mehr zum Einsatz empfehlen.

In Tabelle 3.2 sind die Ergebnisse des N-Gramm Verfahrens ersichtlich. Offensichtlich ist, dass die Ergebnisse deutlich besser als die der Majority Vote Algorithmen sind (siehe Tabelle 3.1). Interessant ist zudem der Unterschied zwischen dem MavHome-Datensatz und den restlichen, denn bei ersteren hat die Erhöhung von N zunächst einen positiven Effekt, wohingegen bei letzteren die Auswirkungen genau gegenteilig sind. Die Verschlechterungen beim Datensatz des aktiven Access Points oder des aktiven Peers ist allerdings so gering, dass sie zu vernachlässigen sind, zumal die Vorhersagequalität auf diesen Daten generell sehr gut ist, was sicherlich darin begründet ist, dass die Klassen nicht sehr homogen verteilt sind. Eine Erklärung dafür, warum dieser Effekt gerade beim MavHome-Datensatz auftritt könnte sein, dass menschliche Bewegungsmuster, die in einem Haus aufgezeichnet wurden, deutlich komplexer sind als beispielsweise die Bewegungen auf einer Webseite.

In Abbildung 3.13 ist die Entwicklung der Lossfunktion für den MavHome- und Wiki-Datensatz dargestellt. Die Grafiken für die restlichen Datensätze befinden sich in Abschnitt B.2. Auf Abbildung 3.13a fällt auf, dass im Gegensatz zum Wiki-Datensatz die 5 Kurven eine durchaus unterschiedliche Form haben. Die starken Schwankungen zu Beginn könnten als Lernphase interpretiert werden. Allerdings spricht der Fakt, dass sich die verschiedenen Varianten in diesem Bereich meist schon gleich verhalten dagegen. Weiterhin ist die Stelle um 60 000 interessant, an



(a) (b)

Abbildung 3.13: Ergebnisse des N-Gramm Algorithmus auf dem MavHome- und Wiki-Datensatz

der alle Varianten mit $N > 1$ einen mehr oder minder großen Knick nach unten machen. Eine ähnliche Situation ergibt sich zwischen 150 000 und 180 000, an der alle Algorithmen mit $N > 3$ nochmal gute Vorhersagen machen. Dies könnte ein Hinweis darauf sein, dass an diesen Stellen Muster auftreten, die mit einem zu kleinen Wert für N nicht registriert werden können. Ganz zu Beginn ist erkennbar, dass die Varianten mit kleinen Werten für N in diesem Bereich besser abschneiden, und im Falle von $N = 3$ diesen Vorsprung auch nicht mehr herzugeben scheinen. Allerdings ist auch zu sehen, dass die Algorithmen für $N > 3$ sich der $N = 3$ Kurve annähern und auf einem noch größeren Datensatz könnten sie vielleicht im weiteren Verlauf wieder bessere Ergebnisse erreichen.

Ein etwas anderes Bild zeichnet Abbildung 3.13b, in welchem die Kurven für alle Werte von N nahezu parallel laufen. An diesen Ergebnissen zeigt sich aber deutlicher, dass die Implementierungen mit hohen N-Werten dazu neigen, die Muster geringer Länge zu übersehen. Dies ist besonders gut an der Steigung zwischen 5 000 und 11 000 zu sehen, in der zwar alle Varianten ansteigen, die $N = 1$ Variante aber ständig kurze Muster wieder zu erkennen scheint, da sie an dieser Stelle nicht so stark ansteigt, wie die anderen. Es zeigt sich auch, dass die Kurven für große N-Werte nicht so abrupte „Richtungswechsel“ vornehmen, sondern etwas geglättet sind. Dies könnte auf dort vorhandene längere, sich wiederholende

Sequenzen zurückzuführen sein. Bei denen im Anhang gezeigten Datensätzen verhält es sich wie beim Wiki-Datensatz.

Insgesamt konstruieren die Ergebnisse ein gutes Bild von den Varianten. Im MavHome-Szenario ist die Vorhersagequalität allerdings nur begrenzt, denn ein Ergebnis mit einer Fehlerquote von nahezu 50 % kann für den Benutzer störende Auswirkungen haben. Die Vorhersage der nächsten Wikiseite mit einer Quote von über 80 % ist allerdings durchaus für die Praxis geeignet.

3.7.4 Hidden Markov Model

Auf Grund seiner Einschränkungen bedurfte das Hmm einer gesonderten Evaluation. Es ist wie oben erwähnt kein inkrementelles Lernverfahren und deshalb war es nötig eine Trainingsmenge zu erstellen mit der das Modell gelernt werden konnte, bevor es auf einer Validationsmenge Vorhersagen tätigen durfte. Um eine Beeinflussung der Ergebnisse durch eine ungünstige Wahl der Trainingsmenge auszuschließen, wurde auf eine 5-fache Kreuzvalidierung zurückgegriffen. In diesem Verfahren wird der gesamte Datensatz in fünf Mengen aufgeteilt. In jedem der fünf Durchläufe wird nun eine als Validationsmenge, der Rest als Trainingsmenge genutzt. Die Ergebnisse werden dann gemittelt. Weiterhin musste bedacht werden, dass das Ergebnis zusätzlich auch von der durch Zufall bestimmten Initialisierung des Hmm abhängen kann, so dass jeder Durchgang zusätzlich noch fünf mal durchgeführt wurde.

Um die für den Baum-Welch Algorithmus erforderliche Menge von Beobachtungssequenzen zu erzeugen, wurde ein Fenster der Länge l über die Daten geschoben. Nachdem das Modell damit gelernt wurde, wurde die Kontrollmenge sequenziell abgearbeitet, indem die letzten $l - 1$ aufgetretenen Werte gespeichert wurden und zur Vorhersage das Zeichen gesucht wurde, das die Zeichenkette der Länge l erzeugt, welche die höchste Wahrscheinlichkeit hat.

Während der Auswertung hat sich gezeigt, dass die gewählte Implementierung des Hmms Probleme mit den erzeugten, großen Datenmengen hat, so dass zunächst die Menge der Daten deutlich reduziert wurde. Aus dem MavHome-Datensatz wurden so die ersten 5 000 Einträge ausgewählt, um zumindest grob die Parameter des Hmm zu bestimmen, aber selbst diese Auswertungen nahmen sehr viel Zeit in Anspruch.

Zustände	5 000	10 000
12	0.733	—
14	0.7356	0.8107
16	0.721	0.8293
18	0.7178	0.8088
20	0.7319	0.8631
22	0.7358	0.8775
24	0.7525	—
26	0.7625	—
28	0.8354	—
30	0.851	—

Tabelle 3.3: Ergebnisse des Hidden Markov Modell auf einem Ausschnitt des MavHome-Datensatzes

In Tabelle 3.3 sind die Ergebnisse dieses Versuchs ersichtlich. Die Ergebnisse liegen im Bereich über 0.7, es sind also nur in 30 % die Vorhersagen korrekt. Die besten Vorhersagen wurden bei 18 versteckten Zuständen erzielt, was nicht unplausibel klingt, denn der MavHome-Datensatz besteht aus 2·19 Sensoren. Für eine weitere Auswertung auf den Datensatz mit 10 000 Werten wurden nur die Verfahren mit 14–22 Zuständen berücksichtigt, da auch in diesem Bereich die vorher besten Ergebnisse erzielt wurden. Das Ergebnis auf dem größeren Datensatz ist erstaunlich, denn die Vorhersagequalität nimmt sogar noch deutlich ab. Aus diesen Ergebnissen zusammen mit der Nicht-Erfüllung einiger Anforderungen lässt sich klar sagen, dass ein Hmm auf diesen Daten nicht sehr geeignet zu sein scheint. Infolgedessen wurde dieser Ansatz nicht weiter verfolgt.

3.7.5 Active Lempel Ziv

Die Ergebnisse des Alz Verfahrens sind in Tabelle 3.4 zusammengefasst. Auf Grund der in Abschnitt 3.6 dargelegten theoretischen Überlegungen von Feder *et al.* [15], könnte erwartet werden, dass dieser Algorithmus zumeist besser vorhersagt als beispielsweise der N-Gramm Algorithmus. Dies ist offensichtlich nicht zwingend der Fall. Zwar wird auf dem MavHome-Datensatz eine Verbesserung erzielt, auf

	ALZ
MavHome	0.3919
Wiki	0.2035
Aktives Fenster	0.1818
Aktiver AP	0.0008
Aktive Peers	0.0087

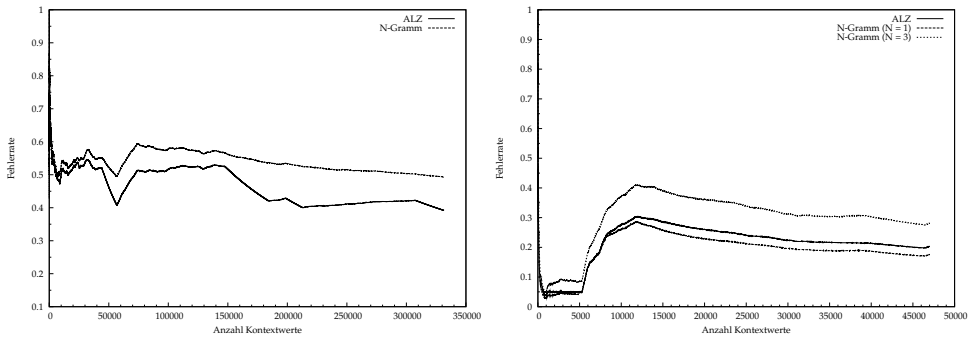
Tabelle 3.4: Ergebnisse des ALZ Verfahrens

den anderen Datensätze aber jeweils eine Verschlechterung. Anscheinend ist eine Verbesserung nicht möglich, sofern der N-Gramm Algorithmus sich schon mit steigendem N nicht verbessern konnte. Diesen Zusammenhang näher zu untersuchen, in dem weitere Datensätze mit komplexeren Mustern erzeugt werden bleibt allerdings an dieser Stelle offen.

In Abbildung 3.14a ist der Vergleich zwischen ALZ und dem N-Gramm Algorithmus auf dem MavHome-Datensatz ersichtlich. Für diesen Graph wurde jeweils der niedrigste Wert aus den fünf verschiedenen N-Gramm Durchläufen verwendet. Es zeigt sich, dass der ALZ stets ein besseres Ergebnis zeigt. Zudem zeigt er auch all die Charakteristika die in Abbildung 3.13a nur einzelne Varianten gezeigt haben. Dies kann dadurch begründet werden, dass der Algorithmus im übertragenen Sinne stets mehrere Werte für N betrachtet.

Etwas anders stellt sich die Lage auf Abbildung 3.14b dar. Der ALZ lässt zwar alle Verfahren mit $N \geq 2$ hinter sich, ist in seiner Leistung aber ein wenig schlechter als der N-Gramm Algorithmus für $N = 1$. Ein deutlicher Unterschied zwischen den beiden entsteht allerdings erst ab etwa dem 8 000. Datensatz. Zuvor liegen sie in etwa gleich auf. Ab dem 12 000. Datensatz scheinen sie sich auch wieder recht gleich zu Verhalten. Die Graphen für die restlichen Daten befinden sich in Abschnitt B.3. Im (nicht dargestellten) Vergleich zu den N-Gramm Varianten zeichnet sich erneut das gleiche Bild. Von der Vorhersagequalität scheint auch dieser Algorithmus gut geeignet, um zur Vorhersage eingesetzt zu werden. Allerdings braucht er spürbar mehr Zeit als der N-Gramm Algorithmus, wobei er dennoch mehrere Tausend Vorhersagen pro Sekunde schafft.

Bereits in der theoretischen Laufzeit wurde der Speicher kritisch beäugt. Die Speicherkomplexität ist für LZ78 mit $\mathcal{O}(n \log(n))$ angegeben (siehe Abschnitt 3.6).



(a) (b)

Abbildung 3.14: Vergleich des N-Gramm Algorithmus mit dem ALZ Algorithmus auf dem MavHome-Datensatz (a) und dem Wiki-Datensatz (b)

Da der MavHome-Datensatz der größte ist, wurde der Speicherverbrauch bei der Vorhersage gemessen und ist in Abbildung 3.15 zu sehen. Der Speicher wurde alle 1 000 Vorhersagen mit Hilfe der von Java bereitgestellten Möglichkeiten gemessen und ist auf Grund des Speichermodells leicht fehlerbehaftet. Gemessen wurde der Speicherverbrauch in Megabyte (MB). Zusätzlich wurde eine Funktion $f(x) = ax \log(x) + bx + c \log(x)$ automatisch auf die Daten gefittet. Es ist ersichtlich, dass diese Funktion gut an die Werte angepasst werden kann und scheint zu bestätigen, dass der Speicherverbrauch in der vorher angegebenen Komplexitätsklasse liegt. Es waren auch Speicherprobleme beim N-Gramm Algorithmus vermutet worden, allerdings liegt der Speicherverbrauch nach Abschluss des MavHome-Datensatzes bei etwa 11 MB, einem Wert, der im Gegensatz zu den hier auftretenden Werten zu vernachlässigen ist.

Interessant am Speicherverlauf ist das Plateau, dass sich zwischen 150 000 und 200 000 gebildet hat. An dieser Stelle ist in Abbildung 3.14a ein starke Abnahme der Lossfunktion zu sehen. Die Erklärung für den weniger starken Speicherzuwachs in dieser Zeit ist, dass der Speicher nur wächst, wenn neue Elemente in den Trie aufgenommen werden. An dieser Stelle scheinen sich viele Ereignisse zu wiederholen, die bereits vorher schon erkannt wurden.

Ein Weg diesem Speicherverbrauch beizukommen, wäre die Technik des Pruning, sprich Teile des Tries abzuschneiden und so Speicher zu sparen. Im maschinellen

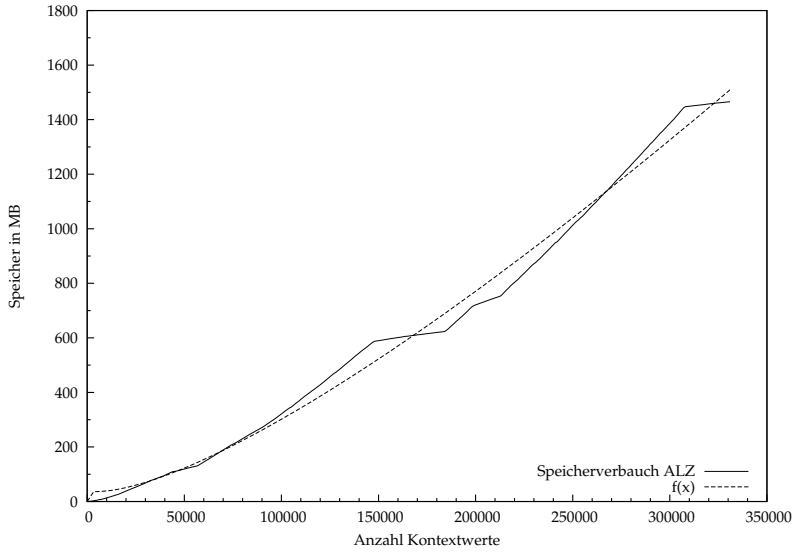


Abbildung 3.15: Speicherverbrauch des ALZ Verfahrens auf dem MavHome-Datensatz in Megabyte. Die Funktion f ist eine eingepasste Funktion des Typs $\mathcal{O}(n \log(n))$.

Lernen wird dabei meist auch das Ziel verfolgt, das Verfahren dadurch etwas zu generalisieren, was dazu führt, dass es sich nicht zu stark an die vorhandenen, möglicherweise verrauschten Trainingsdaten anpasst.

Ein erster einfacher Ansatz zum Pruning bestand darin nach ein paar gesehenen Kontextwerten die Blätter aus dem Trie zu entfernen, bei denen das Verhältnis zwischen eigenem Zähler und dem Zähler des Elternknoten unterhalb eines Pruningfaktors φ lag. So ist bei $\varphi = 0.1$ der Speicherverbrauch zwar unterhalb der 10 MB Marke, allerdings leidet die Vorhersagequalität deutlich darunter und die Lossfunktion endet mit einem Wert über 0.6. Ein Wert von $\varphi = 0.02$ hingegen, brachte die Funktion auf 0.49 zurück (und ist damit in etwa so gut wie der beste N-Gramm Algorithmus), allerdings war ein Speicherverbrauch von annähernd 128 MB messbar. Dies ist im Vergleich zwar nur 10 % des ursprünglichen Wertes, allerdings doch noch deutlich mehr als der N-Gramm Algorithmus benötigt.

Insgesamt hinterlässt der ALZ also einen zwiespältigen Eindruck und kann die aus der Theorie gesteckten Erwartungen nicht gänzlich erfüllen. Er schneidet in

der Vorhersagegüte zwar nicht sonderlich schlecht ab, allerdings ist sein Ressourcenverbrauch gegenüber den N-Gramm Versionen enorm. Andererseits hat er in komplexeren Szenarien, wie im MavHome deutlich bessere Ergebnisse erzielen können.

Kapitel 4

Plattform

Nachdem bis hierher die Kontextvorhersage und deren Verfahren im Vordergrund standen, wendet sich diese Arbeit nun der Entwicklung des Demonstrators zu. Dieser basiert auf einer Reihe von Technologien, die in diesem Kapitel kurz vorgestellt werden. Die Beschreibungen soll eine kurze Einführung geben, die genügt um die späteren Entscheidungen, die auf Grund einer dieser Technologien gefallen sind, nachvollziehen zu können. Insbesondere erheben sie weder Anspruch auf Vollständigkeit noch auf einen hohen Detailgrad.

4.1 XWiki

In Abschnitt 1.2 wurde bereits erläutert, dass die verschiedenen, im Wiki dargestellten Panels mit unterschiedlichen Varianten ausgestattet sein sollen, die von der Middleware je nach Situation automatisch ausgewählt werden.

XWiki[1] ist eine Java-Anwendung und bedarf eines Applikationsservers, der allerdings optional in Form des Jetty-Servers mitgeliefert wird. Das Wiki bietet weitreichende Möglichkeiten sowohl die Funktionalität als auch die Darstellung zu erweitern. Im Rahmen dieser Arbeit ist die Erweiterbarkeit mit Hilfe in Java geschriebener Plugins, die tief in die Verarbeitung einzelner Anfragen eingreifen können, von besonderer Wichtigkeit. Gleiches gilt für die Möglichkeit die Oberfläche des Wikis mit Hilfe der Skriptsprache Velocity vielfältig zu ändern.

[1] Verfügbar auf http://www.xwiki.org

4.2 OSGi

Die Plattformspezifikation OSGi[2] ist von der OSGi Alliance festgelegt, einem Konsortium namhafter Firmen wie SAP, IBM oder Sun. Das Ziel dieser Vereinigung ist es in Form des OSGi Standards eine Plattform für Java Entwickler zu schaffen, mit der eine leichte Entwicklung von komponentenbasierten und serviceorientierten Anwendungen ermöglicht wird. In der Spezifikation sind nur die Schnittstellen und deren Verhalten festgelegt. Den Kern von OSGi bildet das sogenannte Framework, welches die grundlegende Funktionalität, meist in Form einer Konsole, bereitstellt.[3] Es ermöglicht die Installation von verschiedenen Komponenten, sowie das Starten und Stoppen einzelner Dienste. Es gibt zahlreiche Projekte, die dieses Framework bereits implementiert haben. Diese Arbeit stützt sich auf das vom Eclipse Team genutzte Equinox[4].

Eine auf OSGi basierende Applikation ist in mehrere, separat agierende Komponenten zerlegt, die untereinander über Schnittstellen kommunizieren können. Jedes dieser sogenannten Bundles bildet eine eigene Jar Datei, die dann in das Framework geladen werden kann. Sie enthalten im Manifest neben den im Java-Standard festgelegten Attributen weitere OSGi-spezifische. Diese kontrollieren das Verhalten des Frameworks während der Installation der einzelnen Bundles. Die Angaben beinhalten unter anderem die (Java-)Pakete, die aus anderen Bundles importiert sowie für andere Bundles exportiert werden. Der Unterschied zwischen nur bundleintern verfügbaren und von extern eingebundenen Paketen ist eine Erweiterung gegenüber des normalen Java-Verhaltens, welche den modularen Entwurf von Anwendungen unterstützen kann.

Die von einer Komponente bereitgestellten Schnittstellen zur Kommunikation mit anderen Bundles, werden in der OSGi-Terminologie „Services“ genannt und bilden ein weiteres wichtiges Konzept der Spezifikation. Natürlich können ein oder mehrere Bundles verschiedene Implementierungen der gleichen Schnittstelle anbieten. Das Framework ist in diesem Fall für die Auswahl der Instanz des Dienstes zuständig, der an eine, die Schnittstelle importierende Klasse weitergegeben

[2] Verfügbar auf http://www.osgi.org

[3] Für eine detaillierte Einführung in den Aufbau des OSGi Standards siehe http://www.osgi.org/About/Technology

[4] Verfügbar auf http://www.eclipse.org/equinox/

wird.[5] Dadurch wird die Austauschbarkeit einzelner Komponenten gefördert und die Modularität der gesamten Anwendung kann gesteigert werden. Sofern die Implementierung der einzelnen Komponenten es zulässt, ist es sogar möglich einzelne Bundles auszutauschen, ohne dass die gesamte Anwendung gestoppt werden müsste.

In der Spezifikation sind neben dem gerade beschriebenen Kern weitere Dienste definiert. Eine Implementierung des Frameworks selbst enthält meist nicht alle dieser Services und entsprechend müssen weitere, teilweise von anderen Anbietern bereitgestellte Bundles dafür eingesetzt werden. Beispielsweise existiert im Standard zwar eine Schnittstelle für einen Http Service, der das Hinzufügen und Entfernen von Servlets in einen Applikationsserver erlaubt, realisiert wird dieser allerdings durch das Bundle des Applikationsservers Jetty und nicht durch Equinox. Generell unterstützt die Definition eines Dienstes im Standard die Austauschbarkeit der Implementierung desselben.

Einer dieser Dienste sind die sogenannten „declarative services“ (DS), die sich mit Hilfe des „Inversion of Control“ (IoC) Patterns[6] um die automatische Erzeugung und Verwaltung der verfügbaren Services kümmert und dadurch den Export und Import vereinfacht. In einer XML Datei werden die Klassen eines Bundles angegeben, die bestimmte Dienste exportieren oder importieren. Die DS werten diese Datei aus und erzeugen automatisch eine Instanz der Klasse, sobald die benötigten Dienste im Framework registriert wurden. Anschließend werden Letztere importiert (und an die neu erzeugte Instanz weitergereicht) und die realisierten Services werden exportiert. Die Modellierung der Abhängigkeiten erlaubt bestimmte Dienste als optional zu kennzeichnen. Die Möglichkeit auch alle existierenden Implementierungen eines Services zu importieren kann genutzt werden, um auf einfache Weise publish-subscribe Verfahren zu realisieren.

[5] Genau genommen wird diese Entscheidung nicht allein vom Framework gefällt, sondern kann durch Angabe von Prioritäten oder bestimmten Paketversionen beeinflusst werden. Theoretisch ist es sogar möglich, dass der Entwickler eines Bundles selbst aus dem Pool der verfügbaren Services eine bestimmte Implementierung wählt. Dies würde dem System allerdings die Dynamik nehmen und es ad absurdum führen.

[6] Inversion of Control bezeichnet ein Paradigma, in dem ein System, wie in unserem Fall OSGi, den Kontrollfluss einer Anwendung bestimmt, indem es an gewissen Punkten vom Benutzer bereitgestellten Code aufruft. Für eine genauere Einführung siehe zum Beispiel [16, 25]

4.3 Music

Music (Self-adapting applications for *M*obile *US*ers *I*n ubiquitous *C*omputing environments)[7] ist eine sich derzeit in Entwicklung befindende, kontext-sensitive Middleware. Das Ziel ist die Möglichkeit zu bieten adaptive Applikationen zu entwickeln, die auf mobilen Geräten laufen und sich leicht auf mehrere Geräte verteilen lassen. Beispielsweise sind Anwendungen denkbar, die im Büro ihre Arbeit mit Hilfe der dort vorhandenen Infrastruktur durchführen. Sobald diese Umgebung verlassen wird, benötigen sie Letztere aber nicht mehr und passen sich den zur Verfügung stehenden Ressourcen an.

Die Middleware nutzt Osgi als Unterbau und jede größere Komponente ist darin als ein einzelnes Bundle realisiert. Im Rahmen dieser Arbeit sind insbesondere die Adaptions- und Kontextmiddleware von Bedeutung. Die weiteren Komponenten, die sich unter anderem um die Kommunikation verschiedener Music Instanzen miteinander oder das Ressourcenmanagement kümmern, werden deswegen nicht näher erläutert.

4.3.1 Adaption

Auf Music basierende Anwendungen bestehen grundsätzlich aus zwei Teilen. Zum einen aus einer abstrakten Beschreibung des Aufbaus in Form hierarchisch aufgebauter Pläne, zum anderen aus der Realisierung dieser durch sogenannte Varianten [19][8]. Die Pläne beinhalten Informationen darüber, welche Ressourcen benötigt werden, welche Kontextelemente von Bedeutung sind oder durch welche Klasse eine Variante realisiert wird. Die Adaption findet mit Hilfe der ebenfalls im Plan enthaltenen Utility-Funktion die Variante, die in der aktuellen Situation am geeignetsten ist – für die die Utility-Funktion also den höchsten Wert annimmt.

Durch diesen Aufbau ist folgendes Szenario denkbar: Eine Anwendung auf einem mobilen Gerät dient zur Abfrage eines web-basierten Kalenders und besteht aus zwei Varianten, die sich im Ressourcenverbrauch deutlich unterscheiden, da sie entweder eine langsame Gprs Verbindung oder eine schnelle Wlan Verbindung nutzen. Sobald das Gerät an ein Netzteil angeschlossen wird und damit plötzlich

[7] Projektwebseite: `http://www.ist-music.eu`

[8] [19, 32] beziehen sich auf die Madam Middleware, welche den direkten Vorgänger von Music darstellt. Die in den Artikeln vorgestellten Konzepte sind jedoch weitestgehend unverändert übernommen worden.

über „unendlich“ viel Energie verfügt, kann die WLAN-Variante gewählt werden, die viele Ressourcen verbraucht. Wird das Gerät vom Netzteil getrennt, erhält auf Grund des jetzigen „Ressourcenmangels“ die energiesparende GPRS-Variante einen höheren Utility-Wert.

Eine auf MUSIC basierende Applikation durchläuft verschiedene Phasen. Dieser Lebenszyklus besteht aus mehreren Schritten:

1. Sobald eine Applikation geladen wird, werden alle Pläne dieser Applikation, die über eine bestimmte Schnittstelle abgefragt werden können, durch den sogenannten Bundlemanager installiert.

2. Soll eine Applikation gestartet werden, wird eine erste Adaption gestartet, in der sie das erste Mal mitberücksichtigt wird.

3. Solange die Applikation läuft, treten verschiedene Änderungen des Kontextes auf die eventuell eine neue Adaption auslösen.

4. Sobald eine Anwendung ihren Dienst getan hat, wird eine neue Adaption ausgelöst, die dafür sorgt, dass die nun freigewordenen Ressourcen neuverteilt werden können.

5. Letztendlich wird eine Applikation ganz aus der Middleware entfernt, indem ihre Pläne durch den Bundlemanager gelöscht werden.

Während der Adaption werden die Utility-Funktionen verschiedener Varianten ausgewertet. Sobald die mit dem höchsten Wert gefunden wurde, wird der entsprechende Plan durch den Konfigurator in die Tat umgesetzt. Dieser ist für die Erzeugung der Instanzen, der in den Plänen angegebenen Klassen, das Verbinden dieser mit den anderen Komponenten der Applikation, das Entfernen eventuell vorhandener alter Verbindungen zwischen einzelnen, nicht weiter genutzten Komponenten und der Zerstörung Letzterer zuständig.

4.3.2 Kontext

Neben den Plänen ist auch das Kontextmodell hierarchisch aufgebaut. In diesem wird zwischen (Kontext-)Elementen und (Kontext-)Werten unterschieden. Während ein Element andere Elemente und Werte beinhalten kann, ist ein Kontextwert ein Paar aus Name und dazugehörigem Wert. Ein Kontextwert, der beispielsweise

die Temperatur repräsentiert, würde „Temperatur“ als Name und die eigentliche Temperatur (zum Beispiel „20“) als Wert beinhalten.

In diesem Modell dient ein Kontextsensor dazu mehrere dieser Elemente zu verwalten. Eine kleine Wetterstation könnte also einen Sensor darstellen, der neben der oben erwähnten Temperatur auch den Luftdruck misst und all diese Werte als einzelne Kontextwerte zur Verfügung stellt und entsprechend aktualisiert. Die Aktualisierung wird an den Kontextmanager weitergegeben, der diese an daran interessierte Komponenten weiterleitet [32][9]. Unter anderem ist die Adaptionsmiddleware eine dieser Komponenten, so dass sie bei der Änderung relevanter Kontextelemente eine Adaption auslösen kann.

4.4 Burlap

Eine Konsequenz aus der Wahl von XWiki und Music ist, dass das Wiki sowie die Middleware nicht im gleichen Applikationsserver und damit auch nicht in der gleichen virtuellen Maschine laufen. Daraus ergibt sich zwangsläufig eine Situation, in der die beiden Anwendungen miteinander kommunizieren müssen. Das Problem der Kommunikation verschiedener Prozesse wurde schon vielfach aufgegriffen und es existiert eine Vielzahl an Lösungen. Da das Wiki bereits eine Webanwendung ist, ist es naheliegend für die Kommunikation auf Webservices zurückzugreifen. Im Rahmen dieser Arbeit wurde für das Protokoll auf die nicht sehr weit verbreitete Burlap Bibliothek[10] zurückgegriffen. Auf die Möglichkeiten des Soap Standards, der in zahlreichen Implementierungen wie Jax-ws oder Axis vorliegt, wurde zu Gunsten einer möglichst einfachen Verwendung verzichtet. Burlap nutzt zur Kommunikation ein leichteres Xml Format.

Um die Bibliothek möglichst einfach in verschiedenen Bundles zu nutzen, wurden zu dem Manifest der herunterladbaren Jar Datei die entsprechenden Osgi Attribute hinzugefügt.

[9] Siehe Fußnote 8

[10] Verfügbar auf `http://www.caucho.com/resin-3.0/protocols/burlap.xtp`

Kapitel 5

Architektur

Im vorherigen Kapitel wurde bereits angesprochen, dass der Demonstrator in zwei große Komponenten aufgeteilt werden kann, welche über das Burlap Protokoll miteinander kommunizieren. Das Wiki und die darin enthaltenen Plugins bildet die eine und der MUSIC-basierte Teil die andere Komponente (siehe Abbildung 5.1 auf Seite 59). In diesem Kapitel geht es um die Frage, wie diese im Groben aufgebaut sind und vor allem wie sie interagieren, um die gestellte Aufgabe zu lösen.

Die Anwendungen, die den eigentlichen Inhalt der Panels im Wiki generieren, werden in dieser Arbeit als *Widgets* bezeichnet. Ein Beispiel für solch ein Widget könnte dem Benutzer helfen Informationen zu bestimmten Begriffen zu finden. Je nachdem was der Anwender in seiner aktuellen Situation bevorzugt, könnte dies zum einen durch eine Google-Suche und zum anderen durch eine Suche bei Wikipedia realisiert sein.

Der Aufbau des Demonstrators richtet sich nach den im Folgenden präsentierten Fragestellungen.

5.1 Auswahl der Widgets

Die erste Frage, die es zu beantworten gilt, ist, wie die den Inhalt erzeugenden Widgets ausgewählt werden. Die Alternativen sind entweder die anzuzeigenden Panels in geeigneter Form im Wiki zu kodieren oder ein dynamisches System die Widgets auswählen zu lassen. Für eine Demonstration der Fähigkeiten der MUSIC Middleware, wäre die erste Lösung sicherlich ausreichend gewesen. Gewählt wurde dennoch der dynamischere Ansatz, da dieser besser um weitere Widgets zu erweitern ist.

Die Komponente „Get Widget Service“ (Gws) übernimmt in der realisierten Architektur diese Rolle. Die explizite Anforderung daran lautet aus dem Benutzer und dessen aktuell aufgerufener Seite eine Liste von Widgets zu generieren, die in seiner Situation am geeignetsten sind.

Dies ermöglicht die gesamte Anwendungslogik auf der Music-Seite anzusiedeln, wohingegen das Wiki lediglich die Aufgabe erhält, die Inhalte zu präsentieren. Aus dieser Argumentation heraus erwächst auch die Antwort auf die Frage, wie die Widgets zwischen Gws und Wiki transferiert werden. Die Formulierung scheint zu implizieren, dass an dieser Stelle das gesamte Objekt übertragen würde. Dies würde allerdings auf beiden Seiten zusätzliche Komplexität entstehen lassen. Um dies zu verhindern, wurde die Entscheidung getroffen, Referenzen auf die Widgets in Form von Urls zu übertragen, mit denen das Wiki auf den gewünschten Inhalt zugreifen kann. Zusätzlich erlaubt dieser Aufbau das Wiki durch einen anderen Client zu ersetzen, ohne die Funktionalität des Demonstrators zu beeinflussen.

5.2 Aufbau der Panels

In Kapitel 4 wurde bereits erwähnt, dass die Panels des Wikis mit Hilfe der Skriptsprache Velocity eine dynamische Natur erhalten. Dies hat den Vorteil auf die Interna des Wikis zugreifen zu können, ohne jedoch Änderungen am Applikationsserver durchführen zu müssen, wie dies etwa im Fall der „richtigen“ Plugins nötig ist.

Die sich aus dem vorherigen Abschnitt ergebene Fragestellung zu den Panels ist, wie es möglich wird die einzelnen Inhalte darzustellen. Die naheliegendste Idee ist, für jedes Widget ein eigenes Panel zu bauen. Bestechend daran ist, dass einzig das Panel für die Darstellung zuständig ist, was dem zuvor geäußerten Ansatz, den Demonstrator möglichst unabhängig von der Anzeige zu machen, entgegen käme. Das Suchwidget würde beispielsweise nur eine Methode anbieten müssen, die dem Panel die Adresse des zu verwendenden Sucherservices liefert. Dieses erzeugt dann eigenständig das restliche Formular.

Ein anderer Ansatz ist den Code auf Seiten des Widgets zu erzeugen und einem Panel die Aufgabe zu geben diesen anzuzeigen. Dies hat den Vorteil einer simpleren Implementierung, denn das Abrufen und Darstellen lässt sich deutlich einfacher bewerkstelligen, als der tiefe Eingriff in die Funktionsweise des Wikis, der nötig

wäre, um das passende Panel in Abhängigkeit der zurückgegebenen Widgets auszuwählen und anzuzeigen.

Realisiert wurde die zweite Variante auf Grund der einfacheren Implementierung. Der Nachteil, dass durch diese Wahl die zuvor vom Client gewonnene Unabhängigkeit zumindest teilweise wieder verloren geht ist nicht von der Hand zuweisen. Dieser wiegt allerdings nicht so schwer, da das Wiki immernoch leicht durch einen beliebigen Client, der HTML darstellen kann, ausgetauscht werden kann. In dem zu Grunde liegenden Szenario ist dies keine großartige Einschränkung.

5.3 Realisierung der Widgets

Die Widgets halten mehrere Alternativen ihrer eigentlichen Funktion bereit und je nach Situation des Benutzers soll die richtige Variante angezeigt werden. Die Beschreibung legt nahe die Widgets als MUSIC-basierte Applikationen zu modellieren. Die Middleware bringt, wie in Abschnitt 4.3 erläutert, die Methoden mit, um mit dieser Problemstellung fertig zu werden.

Ein sich aus dieser Wahl ergebenes Problem ist, dass die Middleware keine Unterstützung für von mehreren Benutzern gleichzeitig genutzte Applikationen mitbringt. Dies ist nicht unwichtig, da die Widgetapplikation prinzipiell für zwei unterschiedliche, aber zur selben Zeit darauf zugreifende Benutzer verschiedene Varianten zurückgeben kann. Eine Modellierung der Anwendung, die diesen Umstand bereits berücksichtigt, würde dem Ziel MUSIC als Basis zu verwenden zu wider laufen, denn einzig die Middleware sollte für die Anpassung an den Kontext zuständig sein. Die Konsequenz daraus ist, dass jedes Widget für jeden Benutzer einzeln gestartet wird. Bei zwei Nutzern A und B, gäbe es das Suchwidget dementsprechend einmal für A und einmal für B. Im Hinblick auf den Ressourcenverbrauch und die Performanz ist diese Variante nicht optimal. Im Demonstrator sind diese Aspekte aber zweitrangig gegenüber dem Ziel die Funktionalität der Middleware zu zeigen.

Diese Entscheidung zieht nach sich, dass sich eine Komponente um die automatische Erzeugung der Widgets für jeden Benutzer kümmert. Dieser Mechanismus und die genauere Struktur der MUSIC Applikationen werden in Kapitel 6 genauer dargestellt. Es gilt dabei zu bedenken, dass aus den zuvor getroffenen Entschei-

dungen auch folgt, dass ein Widget nicht nur eine MUSIC Applikation ist, sondern zugleich einen Webservice bereitstellen muss.

5.4 Kontext und Features

Die Erzeugung und Verarbeitung des Kontextes ist eine weitere wichtige Frage in der Architektur. Die Daten, die in jeder Situation erzeugt werden, sind der Name des Benutzers und dessen aktuelle Seite. Dies allein sind zu wenige Daten, um daraus sinnvolle Schlüsse ziehen zu können. Deswegen werden aus diesen weitere Daten extrahiert und zu einer Menge von „Features“ zusammengefasst. Jedes einzelne Datum, beispielsweise der Name des Benutzers oder der der aktuellen Seite, ist ein Feature.[1]

Die Widgets sind, wie gerade erwähnt, MUSIC Applikationen. Eine Adaption findet nur dann statt, wenn sich der von der Middleware überwachte Kontext ändert. Um die Applikation adaptiv auf die Features reagieren lassen zu können, müssen diese als Kontext in MUSIC exportiert werden. Dazu wird ein Kontextelement angelegt, das den kompletten Featuredatensatz der erzeugt wird beinhaltet. Von dort an kümmert sich der interne Kontextmanager um die entsprechende Verarbeitung und Weitergabe dieser Informationen innerhalb der Middleware.

Wie bereits bei der Frage zur Modellierung der Widgets ist auch an dieser Stelle zu erwähnen, wie mit der Tatsache, dass mehrere Benutzer gleichzeitig auf das System zugreifen könnten umgegangen wird. Die Auswirkungen auf die Struktur, in der der Kontext gespeichert wird, sind vergleichsweise weniger gravierend. Naheliegend ist, für jeden Nutzer ein eigenes Kontextelement zu erzeugen und unter diesen die entsprechenden Werte der Features anzusiedeln. Ein anderer Ansatz wäre, nur ein globales Kontextelement anzulegen und darunter die Werte aller Nutzer zu speichern, wobei deren Namen durch den Benutzernamen so erweitert sind, dass sie eindeutig bleiben. Ein Fehler in der Middleware verhinderte die Implementierung des ersten Ansatzes, da Änderungen der Werte unterhalb der einzelnen Benutzerelemente nicht richtig in der Middleware weiterverteilt wurden und so keine Adaption angestoßen wurde. Eine Lösung dieses Problems, die nach dessen Behebung erlaubt auf den anderen Ansatz zu wechseln wird in Abschnitt 6.4 erläutert.

[1] Abschnitt 6.3 enthält eine Auflistung aller derzeit generierten Features.

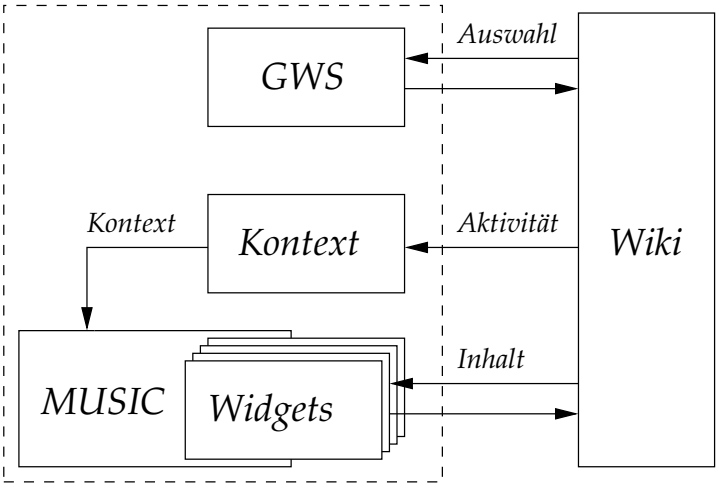


Abbildung 5.1: Architekturdiagramm des Demonstrators. Das Wiki lässt zunächst durch den Gws die anzuzeigenden Widgets auswählen und fragt Letztere nach deren Inhalt. Jede Aktivität des Benutzers auf dem Wiki wird an die Kontextschnittstelle weitergeleitet und von dort in die Music Middleware exportiert.

Die Kontextvorhersage in diesen Mechanismus zu integrieren, lässt sich relativ einfach bewerkstelligen. Eine weitere Komponente, die für die Vorhersage einzelner Elemente zuständig ist, würde sich am Music Kontextmanager anmelden und von dort über Änderungen des entsprechenden Kontextwertes benachrichtigt werden. Sobald diese auftreten werden sie registriert und eine neue Vorhersage generiert. Spezielle Kontextelemente beinhalten dann die zukünftigen Werte. Auf diese Art können zum einen die Utility-Funktionen der Widgets, zum anderen aber auch andere Komponenten mit Hilfe des Music Kontextmechanismusses darauf zugreifen.

5.5 Zusammenfassung

Abbildung 5.1 zeigt die Gesamtarchitektur des Systems. Diese zeigt zunächst die grobe Unterteilung in die zwei Komponenten aus Wiki und dem Music basierenden Teil. Der Aufbau der in diesem Bild zu sehenden Komponenten wird im

nächsten Kapitel näher erläutert. Der Gws kommuniziert nur mit dem Wiki, indem er die anzuzeigenden Widgets auswählt und eine Liste dieser zurückgibt. Die Kommunikation mit der Kontextverwaltung läuft nur in eine Richtung, denn das Wiki soll nur den Kontext für Music bereitstellen und kümmert sich auch um die Aggregation der weiteren Features. Die wichtigsten Komponenten bilden die Widgets, die von Music verwaltet werden, und bidirektional mit dem Wiki kommunizieren.

Bei jeder Aktion des Benutzers im Wiki laufen die folgenden Schritte parallel ab:

1. Die Kontextkomponente wird darüber informiert, dass sich der Benutzer nun auf der neuen Seite befindet.

2. Für die Darstellung der neuen Seite, fragt das Wiki den Gws nach einer Liste von für den Benutzer interessanten Widgets. Nachdem es diese erhalten hat, fragt es die Widgets nach den entsprechenden Inhalten.

Im folgenden Kapitel wird auf die Details der hier dargestellten Komponenten eingegangen.

Kapitel 6

Design und Implementierung

Während im letzten Kapitel die allgemeine Architektur des Demonstrators erläutert wurde, dreht sich dieses um die näheren Details der einzelnen Komponenten. Dazu wird zunächst die Aufteilung des Demonstrators in einzelne OSGi Bundles aufgezeigt. Darauffolgend werden, anhand des im vorherigen Kapitel dargestellten Aufbaus, die drei Hauptkomponenten – Widget, Gws und Kontextschnittstelle – näher beschrieben. Abschließend wird kurz auf die im Wiki implementierten Komponenten eingegangen.

6.1 OSGi Bundles

MUSIC basiert auf OSGi und daher ist es naheliegend auch den Demonstrator darauf aufbauen zu lassen. Darüber hinaus hat das Framework den Vorteil, dass Komponenten leicht ausgetauscht werden können. Zusätzlich gestaltet sich dadurch die Kommunikation mit der MUSIC Middleware recht einfach.

Tabelle 6.1 listet alle Bundles mit einer kleinen Beschreibung ihrer Funktion auf. Darin sind zunächst diejenigen zu sehen, die den eigentlichen Demonstrator bilden. Diesen folgen die implementierten Widgets. Im unteren Teil der Tabelle sind die Bibliotheken, die als Bundles zur Verfügung gestellt werden, mit Ausnahme von MUSIC und den dazugehörigen Bibliotheken, dargestellt. Im Folgenden werden die wichtigsten Bundles des Demonstrators im Detail erläutert.

6.1.1 InterfaceBundle

Das InterfaceBundle bildet die Basis der Entwicklung des Demonstrators und enthält alle wichtigen Schnittstellen. Auf ihn zugreifende Clients benötigen zur

Bundle	Beschreibung
InterfaceBundle	Sammlung wichtiger Schnittstellen
WidgetBaseBundle	Grundlage zur Entwicklung neuer Widgets
GetServletBundle	Implementierung des Gws
ContextAggregator	Implementierung der externen Kontextschnittstelle
ContextBridge	Brücke zwischen Kontextschnittstelle und Music
WidgetRegistry	Schnittstelle, um auf Widgets zuzugreifen
UserRegistry	Schnittstelle, um auf die Benutzer zuzugreifen
RecommenderLearnerBundle	Implementierung des Lernverfahrens des Gws
SIHARecommenderBridge	Brücke zum Siha Recommender
TestWidget	Ein Widget zum Testen der Funktionalität
SearchWidget	Gibt ein Suchfenster aus
TranslateWidget	Bietet Übersetzungsmöglichkeiten an
Log4j und Log4jProperties	Loggingmechanismus und dessen Konfiguration
Burlap OSGi	Burlapbibliothek als Osgi-Bundle

Tabelle 6.1: Übersicht über alle erzeugten OSGi Bundles

Kommunikation nur die Jar Datei dieses Bundles, um Zugriff auf die externen Schnittstellen des Demonstrators zu haben. Sofern sich diese nicht ändern, müssen diese Clients nicht neu kompiliert werden, falls eine der Komponenten ausgetauscht werden sollte.

Das Bundle enthält zunächst die drei in der Architektur dargelegten Schnittstellen zum Wiki:

ContextType	Java Klasse
INTEGER	Integer, int
LONG	Long, long
FLOAT	Double, double
STRING	String
BOOLEAN	Boolean, boolean

Tabelle 6.2: Übersicht über die durch `ContextType` repräsentierten Kontexttypen

1. Der Gws wird durch die Schnittstelle `WidgetGetService` repräsentiert. Diese enthält die Methode `getWebservices`, welche den Benutzernamen und die aktuelle Seite als Parameter übergeben bekommt. Zurück gibt sie eine Liste von Stringobjekten, die die URLs der anzuzeigenden Widgets beinhalten.

2. `ContextAggregator` implementiert die Kontextschnittstelle. Sie bietet vier Methoden an, mit denen die Kontextwerte manipuliert werden können.

   Von diesen dienen zwei Methoden dazu einen neuen Kontextwert hinzuzufügen. Während `registerGlobalContextElement` den Namen, den Typ und den initialen Wert übergeben bekommt, erhält `registerContextElement` zusätzlich einen Benutzernamen als Parameter. Beide Methoden legen einen neuen Kontextwert an, dessen Typ über ein Element der ebenfalls im InterfaceBundle enthaltenen Aufzählung `ContextType` festgelegt wird. Der anfängliche Kontextwert muss mit diesem Typ übereinstimmen, also in die entsprechende Javaklasse umwandelbar sein (siehe Tabelle 6.2). Der Unterschied zwischen den beiden Methoden liegt darin, dass die zuerst genannte einen globalen Wert erzeugt. Dieser Wert ist für alle Benutzer derselbe. Im Gegensatz dazu legt die zweite einen nur für den entsprechenden Nutzer gültigen Wert an.

   Die beiden weiteren Methoden `updateGlobalContextElement` und `updateContextElement` erhalten als Parameter den während der Registrierung angegebenen Namen, sowie den aktualisierten Kontextwert. An Letztere muss zusätzlich ein Benutzernamen übergeben werden. Der Typ des übergebenen Wertes muss dem bei der Registrierung angegebenen entsprechen. Der Unterschied zwischen globalen und nutzerbasierten Werten ist analog zu dem oben erläuterten.

InterfaceBundle	
Exportierte Pakete:	de.emld.music.interfaces
Importierte Pakete:	—
Exportierte Services:	—
Importierte Services:	—

Tabelle 6.3: Zusammenfassung des InterfaceBundles

3. Die dritte externe Schnittstelle stellen die Widgets dar, welche durch die Klasse `WidgetAccess` umgesetzt werden. Genau wie die Schnittstelle des Gws beinhaltet auch sie nur eine Methode `getWidgetHTML`, die in diesem Fall für das Abrufen des entsprechenden Inhalts für das Wiki Panel zuständig ist. Sie gibt ein Stringobjekt zurück, das den vollständigen HTML Code beinhaltet.

Ein Client ist mit diesen drei Schnittstellen in der Lage auf den vollen Funktionsumfang des Demonstrators zuzugreifen. Zusätzlich beinhaltet das Bundle weitere, nur intern verwendete.

Eine in verschiedenen Komponenten verwendete Schnittstelle ist die `UserRegistry`, die es erlaubt über alle im Wiki angelegten Nutzernamen zu iterieren, indem sie die in Java enthaltene Schnittstelle `Iterable<String>` erweitert. Die Schnittstelle `FeatureExtractor` ermöglicht hingegen die Erzeugung der Features aus dem Namen des Benutzers und dessen aktueller Seite. Die Methode `getFeaturesFor` bekommt diese als Parameter übergeben und gibt eine Liste aller Features zurück. Letztere bestehen jeweils aus einem Namen und einem Wert und werden durch die ebenfalls im Bundle enthaltene Klasse `Feature` repräsentiert. Der zuvor bereits erwähnte Kontexttyp wird durch die Aufzählung `ContextType` repräsentiert, deren Elemente die Grundtypen von Java repräsentieren (siehe Tabelle 6.2).

Alle Klassen des Bundles liegen im Paket `de.emld.music.interfaces`, welches auch exportiert wird. Importiert wird kein Paket, genauso werden Services weder imnoch exportiert. Diese Angaben sind in Tabelle 6.3 zusammengefasst.

6.1.2 GetServletBundle

Der in GetServlet Bundle implementierte und in Tabelle 6.4 zusammengefasste Gws stellt die Schnittstellen `WidgetLearnerService` und `WidgetRegistry` bereit

GetServletBundle	
Exportierte Pakete:	de.emld.music.gws
Importierte Pakete:	de.emld.music.interfaces
	com.caucho.burlap.server
	javax.servlet
	org.apache.log4j
	org.osgi.service.http
Exportierte Services:	de.emld.music.interfaces.WidgetGetService
Importierte Services:	de.emld.music.interfaces.WidgetLearnerService
	de.emld.music.interfaces.FeatureExtractor
	de.emld.music.interfaces.WidgetRegistryService
	org.osgi.service.http.HttpService

Tabelle 6.4: Zusammenfassung des GetServletBundles

und nutzt diese neben dem FeatureExtractor intern. Erstere definiert eine Methode getWebservicesFor, die eine Liste von Features übergeben bekommt und die Namen der Widgets – nicht etwa deren Adressen – in Form einer Liste von Strings zurückgibt. Eine Klasse, die diese Schnittstelle implementiert, stellt also die Kernfunktionalität des Gws zur Verfügung.

Das Umwandeln der Widgetnamen in die vom Gws zurückgegebenen Adressen übernimmt die WidgetRegistry Schnittstelle. Sie umfasst insgesamt drei Methoden. registerWidget erhält Namen sowie Adresse des Widgets und speichert diese. Die Adresse des Widgets kann einen Platzhalter %USER% enthalten. Die Methode getWidgetURL bekommt neben dem Namen des Widgets den Benutzernamen, durch den der Platzhalter ersetzt werden soll. Ein Widget kann, sollte es beendet werden, mit unregisterWidget wieder entfernt werden.

6.1.3 ContextAggregator

Die Kontextschnittstelle wird durch das ContextAggregator Bundle bereitgestellt. Dieses Paket exportiert die Schnittstelle ContextBridge, welche die Verbindung zu einem Kontextsystem herstellt. Angelehnt an den Aggregator hat auch die Brücke vier Methoden, je zwei zum Hinzufügen neuer Kontextwerte und zwei zum Ändern

WidgetBaseBundle	
Exportierte Pakete:	de.emld.music.widgets
Importierte Pakete:	de.emld.music.interfaces
	com.caucho.burlap.server
	javax.servlet
	javax.servlet.http
	org.apache.log4j
	org.istmusic.mw.adaptation.cnfiguration
	org.istmusic.mw.kernel
	org.istmusic.mw.manager
	org.istmusic.mw.model
	org.istmusic.mw.model.connectable
	org.istmusic.mw.model.property
Exportierte Services:	—
Importierte Services:	de.emld.music.widgets.WidgetDescription
	de.emld.music.interfaces.WidgetRegistryService
	de.emld.music.interfaces.UserRegistry
	org.osgi.service.http.HttpService
	org.istmusic.mw.manager.IBundleManagement
	org.istmusic.mw.manager.IApplicationController
	org.istmusic.mw.kernel.IRepository

Tabelle 6.5: Zusammenfassung des WidgetBaseBundles

dieser Werte. Diese unterscheiden sich jeweils durch einen weiteren Parameter, der angibt, unter welchem Kontextelement der übergebene Wert gespeichert werden soll. Letzterer wird durch die Klasse Value aus MUSIC repräsentiert.

6.1.4 WidgetBaseBundle

Das WidgetBaseBundle (Tabelle 6.5) ist ein weiterer wichtiger Bestandteil, denn es bildet die Grundlage zur Entwicklung von Widgets. Es beinhaltet die Klasse WidgetDescription, die, wie der Name schon sagt, eine Beschreibung des Widgets

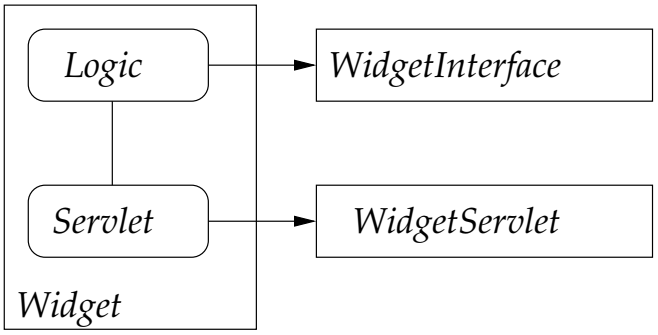


Abbildung 6.1: Aufbau der Music Applikation eines Widgets

darstellt und insbesondere dafür zuständig ist mit Hilfe des sogenannten „Factory Pattern“ [17] dieses für einen bestimmten Nutzer zu instantiieren.

Weiterhin bietet das Bundle die abstrakte Klasse `AbstractWidgetApplication` an, welche die von der Middleware geforderten Details bereits implementiert. Der Service `WidgetCreator` startet alle via OSGi bei ihm registrierten Widgets automatisch für jeden Benutzer. Dazu nutzt dieser die `Widget`- als auch die `UserRegistry` neben einigen weiteren Music Diensten. Die Funktionsweise dieses Erstellungsmechanismus wird in Abschnitt 6.2 näher erläutert.

6.1.5 Weitere Bundles

Die weiteren in Tabelle 6.1 auf Seite 62 ersichtlichen Bundles implementieren im Großen und Ganzen die Schnittstellen der im Detail vorgestellten Bundles. Eine Zusammenfassung aller weiteren Bundles findet sich in Anhang C.

6.2 Widgets

In diesem Abschnitt wird erläutert, wie die Widgets aufgebaut sind und was ein Programmierer zu tun hat, um weitere zu implementieren, bevor auf die bereits implementieren Widgets eingegangen wird.

Ein Widget muss prinzipiell zwei Dinge leisten: Zum einen soll es als Music Applikation realisiert sein, zum anderen soll sein Dienst über die Schnittstelle `WidgetAccess` als Webservice zur Verfügung gestellt werden. Das WidgetBase-

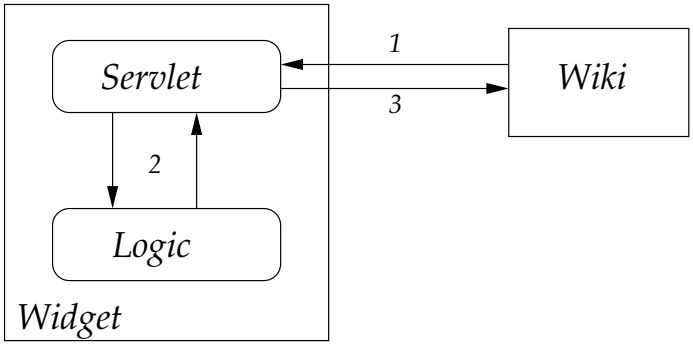


Abbildung 6.2: Ablauf der Abfrage des Inhalts eines Widgets durch das Wiki. Nach Kontaktierung des Servlets durch das Wiki (1), kommuniziert dieses mit der Applikationslogik (2) und gibt den Inhalt abschließend zurück (3).

Bundle soll dem Programmierer die Details der Middleware abnehmen und so eine schnelle und einfache Erzeugung von weiteren Widgets ermöglichen. Abbildung 6.1 zeigt den Aufbau der durch die Klasse `AbstractWidgetApplication` realisierten MUSIC Applikationen. Die Hauptapplikation beinhaltet demnach zwei Komponenten, die zwecks Kommunikation miteinander verbunden sind. Die Komponente „Servlet“ repräsentiert den Webservice, während „Logic“ den Teil darstellt, der die eigentliche Applikationslogik darstellt. Der Webservice wird durch eine Instanz der `WidgetServlet` Klasse realisiert. Die einzelnen Varianten eines Widgets unterscheiden sich dementsprechend durch unterschiedliche Implementierungen der Logikkomponente. Diese müssen jeweils der Schnittstelle `WidgetInterface` gehorchen.

Der Ablauf bei Abruf des Widgets ist in Abbildung 6.2 dargestellt. Das Wiki hat die URL, unter der das Servlet erreichbar ist, vom GWS erhalten und ruft mit Hilfe von Burlap die Methode `getWidgetHTML` der `WidgetAccess` Schnittstelle auf (1). Das Servlet selbst ist durch MUSIC mit der eigentlichen Widgetlogik verbunden, die durch die abstrakte Klasse `WidgetInterface` repräsentiert wird. Diese implementiert die in den MUSIC Schnittstellen `IConfigurable` und `IConnectable` definierten Methoden. Gleichzeitig stellt sie eine `getUser` Methode bereit, die den Benutzer, für den dieses Widget erzeugt wurde, zurückgibt. Die Methode `getHTML` wird durch

das Widget aufgerufen, um die Anzeige abzurufen (2). Das Ergebnis wird dann an das Wiki zurückgegeben (3).

Wie bereits erwähnt wird die Applikation zunächst durch die Klasse `AbstractWidgetApplication` repräsentiert. Diese implementiert die Schnittstelle `IBundle`, die die Grundlage einer jeden MUSIC Applikation ist. Sie definiert vier Methoden, die dazu dienen Komponenten und Pläne einer Applikation innerhalb von MUSIC bekannt zu machen. Der Konstruktor dieser abstrakten Klasse erhält den Namen des Widgets, den Benutzer für den das Widget erzeugt wird, die Utility-Funktion, sowie eine Sammlung der Klassennamen aller Varianten des Widgets. Aus diesen Informationen wird der beschriebene Kompositionsplan erzeugt. Während der Planerzeugung wird die abstrakte Methode `getImportantContextElements` genutzt, um die für die Utility-Funktion benötigten Kontextelemente an MUSIC zu übergeben. Die Klasse `WidgetCoordinator` übernimmt die von der Middleware geforderte Realisierung der Kompisitionskomponente, hat allerdings in diesem Demonstrator keinerlei Funktion.

Der Webservice wird durch die Klasse `WidgetServlet` realisiert, die die Klasse `BurlapServlet` erweitert, wodurch mit Hilfe von Burlap die Schnittstelle `WidgetAccess` exportiert wird. Die Schnittstellenmethode `getWidgetHTML` ruft auf der aktuellen Variante des Widgets die `getHTML` Methode auf und gibt das Ergebnis zurück (vgl. auch Abbildung 6.2).

MUSIC benötigt, wie in Abschnitt 4.3 dargelegt, eine Utility-Funktion, um eine Variante auszuwählen. Für ein Widget muss diese durch den Programmierer angelegt werden. Die abstrakte Klasse `WidgetUtility` abstrahiert den Zugriff auf den von MUSIC bereitgestellten Kontext, so dass die darin definierte Funktion `evaluate` aufgerufen wird und eine Menge von Features übergeben bekommt. Neben den Features, die aus den Kontextelementen, die über die abstrakte Methode `getContextElementsToExtract` abgerufen wurden, extrahiert werden, enthält diese Menge ein weiteres Feature namens „variant“ in dem eine Zahl die aktuelle Variante identifiziert. Der Wert dieses Features richtet sich nach der Reihenfolge, in der die Varianten an den Konstruktor der `AbstractWidgetApplication` übergeben wurden. Die Rückgabe der Utility-Funktion sollte ein Wert zwischen 0 und 1 sein, der die „Nützlichkeit“ der Variante für den Benutzer widerspiegelt.

Das Registrieren und Starten der Applikationen innerhalb von MUSIC übernimmt der `WidgetCreator` (Abbildung 6.3). Mit Hilfe der DS werden alle `WidgetDescription` Services an diese Klasse übergeben. Diese Beschreibung beinhaltet Namen

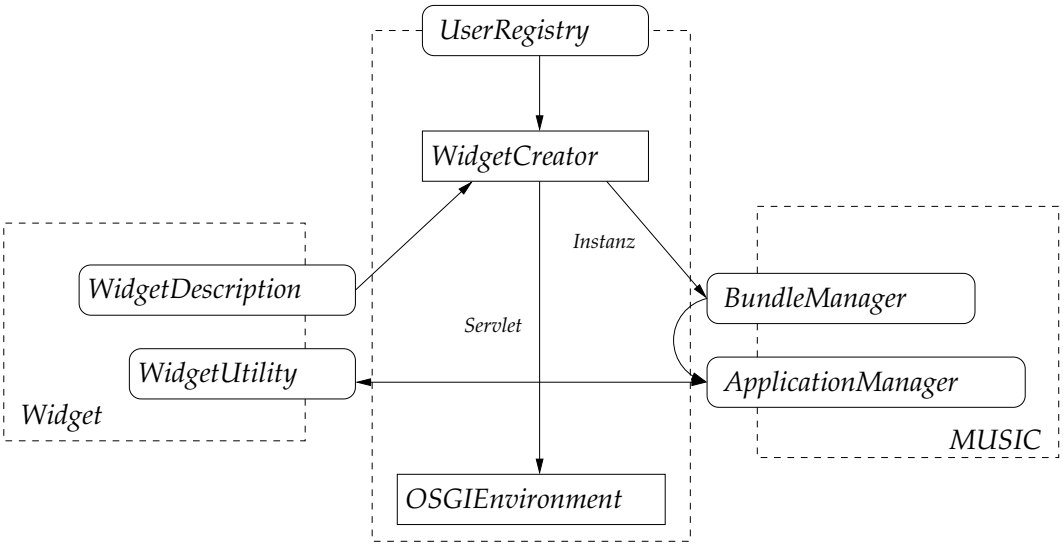


Abbildung 6.3: Schematischer Ablauf des WidgetCreators

und Adresse des Widgets und bietet die Möglichkeit eine Instanz des Widgets mit Hilfe der `getInstanceFor` Methode zu erzeugen. Sie erhält den Benutzernamen und gibt eine Instanz der `AbstractWidgetApplication` zurück. Mit Hilfe der `UserRegistry` wird über alle Benutzer iteriert und für jeden ein Widget erzeugt. Ist dieses geschehen, wird es zunächst am Bundle- und Applikationsmanager innerhalb von MUSIC angemeldet und anschließend gestartet. Darauf folgend wird das Servlet durch die Klasse `OSGIEnvironment` innerhalb des Applikationsservers registriert, wodurch auf den Webservice von außen zugegriffen werden kann. Abschließend wird das Widget an der `WidgetRegistry` angemeldet und kann verwendet werden.

Zusammenfassend müssen folgende Schritte durchgeführt werden, um ein Widget zu erzeugen:

1. `AbstractWidgetApplication` und `WidgetUtility` erweitern.
2. Die entsprechenden Varianten mit Hilfe von `WidgetInterface` implementieren.
3. Danach wird eine `WidgetDescription` implementiert, die es erlaubt Instanzen der Applikation zu erzeugen.
4. Als letztes wird der Service `WidgetDescription` an die DS exportiert.

6.2.1 Übersetzungswidget

Das Übersetzungswidget bietet die Möglichkeit einzelne Wörter von der Sprache der aktuell gewählten Seite in die des Benutzers zu übersetzen. Das Widget hält für jede mögliche Sprachkombination eine Variante bereit. Diese geben jeweils ein Formular aus, in dem der Nutzer einen Ausdruck eingeben kann, der dann mit Hilfe eines Übersetzungsdienstes übersetzt wird. Zu Demonstrationszwecken wurden die Sprachkombinationen Deutsch–Englisch und Deutsch–Französisch implementiert.

Der Feature Extraktor erzeugt Kontextwerte, die sowohl die Sprache des Nutzers als auch die der aktuellen Seite repräsentieren. Anhand dieser Werte, die aus der Datenbank des Wikis gewonnen werden, kann die Utility-Funktion entscheiden, welche Variante am sinnvollsten ist. Da das Wiki von Haus aus kein Wissen über die Sprache einer Seite besitzt, muss diese per Hand in die Datenbank eingetragen werden. Für eine Demonstration ist diese Einschränkung akzeptabel, denn diese Arbeit lässt sich im Vorfeld ausführen. Dass die gewählte Variante nur von diesen (fixen) Kontextelementen abhängt und nicht etwa von einer durch den Benutzer wählbaren Präferenz erscheint zunächst unflexibel. Dies ist aber dem Demonstrationszweck geschuldet, denn so zeigt dieses Widget eine Adaption nur auf Grund von Kontextänderungen, also ohne Eingriff des Benutzers.

6.2.2 Suchwidget

Das Suchwidget soll dem Benutzer ermöglichen aus dem Wiki heraus auf verschiedenen Webseiten zu recherchieren. Die Varianten zeichnen ein Formular, in welches ein Suchwort eingegeben wird und das auf die Ergebnisseite der entsprechenden Suche führt. Gleichzeitig soll es möglich sein, die bevorzugte Suchmaschine zu ändern. Bereits implementierte Varianten führen auf die Seiten von Google oder (der deutschen) Wikipedia.

Im Gegensatz zum vorher beschriebenen hängt dieses Widget also nicht nur vom automatisch erzeugten Kontext ab, sondern ist durch den Benutzer steuerbar. Dazu wurde neben den Varianten zusätzlich ein sogenanntes „FeedbackServlet“ implementiert, das sich auf den bereitgestellten Applikationsserver installiert. Bei Aufruf dieses Servlets, das für jeden Benutzer einmal existiert, wird die präferierte Variante auf die per Parameter übergebene gesetzt. Dies geschieht, indem ein Kontextelement für jeden Benutzer existiert, welches entsprechend aktualisiert

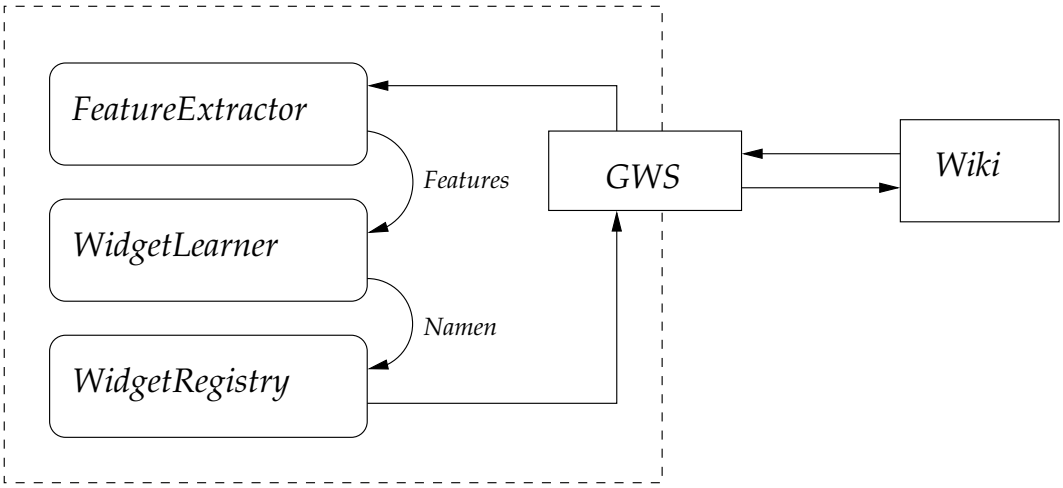


Abbildung 6.4: Schematischer Ablauf des Gws

wird. Gleichzeitig fällt die Utility-Funktion auf Grund dieses Kontextelements die Entscheidung für die genutzte Variante.

6.3 Gws

Die Zuständigkeit des Gws ist die Auswahl der Widgets, die der Benutzer auf der aktuellen Seite benötigt. Dazu wird diesem der Nutzername, sowie dessen aktuelle Seite übergeben, aus welchen mit Hilfe des `FeatureExtractor` weitere Features gewonnen werden. Ein lernendes System (`WidgetLearner`) bekommt diese übergeben und gibt eine Liste mit Namen der anzuzeigenden Widgets zurück, die mit Hilfe der `WidgetRegistry` in Adressen umgewandelt und dann zurückgegeben werden (Abbildung 6.4).

Insgesamt hat dieser Service folglich 3 Variationspunkte. Der `FeatureExtractor` bildet den ersten und erlaubt es so später die Menge der Features beliebig zu erweitern. Die Auswahl der passenden Widgets in Form des `WidgetLearner` dynamisch zu gestalten hat den Vorteil auch dieses System leicht auszutauschen, um so verschiedene Ansätze ausprobieren zu können. Der Extraktor als auch das lernende System sind derzeit im `RecommenderLearnerBundle` realisiert. Die `WidgetRegistry` ist in einem separaten Bundle implementiert.

Der derzeit verwendete Feature-Extraktor nutzt eine Verbindung zur Datenbank des Wikis, um weitere Daten über die Seite und den Nutzer zur Verfügung zu stellen. Die extrahierten Features umfassen:

- Name der aktuellen Seite
- Tag der aktuellen Seite[1]
- Die Sprache der aktuellen Seite
- Der Benutzername
- Die Sprache des aktuellen Benutzers
- Die Aktion, die der Nutzer auf dieser Seite gerade durchführt[2]

Die genannten Features sind in der Aufzählung `FeatureVocabulary` zusammengefasst.

Die grundlegende Idee hinter dem `RecommenderLearner` ist die Definition von Regeln, die mit einer bestimmten Situation eine Menge von Widgets assoziieren. Beispielsweise ist eine Regel denkbar, deren Bedingung umfasst, dass der aktuelle Nutzer „Stephan“ heißt. Ist diese Bedingung in der übergebenen Featuremenge erfüllt, so feuert die Regel und die damit assoziierten Widgets werden zurückgegeben. Sobald mehrere Regeln feuern, werden die Widgets zurückgegeben, die am häufigsten vorkamen. In dem Bundle wird dieses erweitert, indem ein die `Recommender` Schnittstelle implementierender kollaborativer Filter genutzt wird, um die Seiten zu finden, die für den Nutzer zusätzlich interessant sind. Die daraus resultierenden Featuremengen werden einzeln an den Regelmechanismus übergeben und schließlich werden mehrere Widgets zurückgegeben (siehe Abbildung 6.5). In der derzeitigen Implementierung nutzt dieses System statisch festgelegte Regeln. Ein richtiges Regellernverfahren an dieser Stelle einzusetzen, könnte sich allerdings als vorteilhaft für das Lernen des Benutzersverhaltens herausstellen. Eine genaue

[1] Dieses Tag soll in erster Linie eine Klassifizierung der Seite darstellen und existiert derzeit nicht in der Datenbank, sondern lediglich im Quelltext des Demonstrators. Denkbar ist aber eine Verwendung, in der dieses Tag die Tätigkeit kennzeichnet, die zumeist auf dieser Seite ausgeführt wird. Da das XWiki von Haus aus diese Daten nicht liefern kann, bedarf es hier einer manuellen Annotation der Daten.

[2] Genauer gesagt wird diese Information mit der Anfrage übermittelt. Unterschieden wird dabei zwischen „Lesen“ und „Verändern“ der aktuellen Wikiseite.

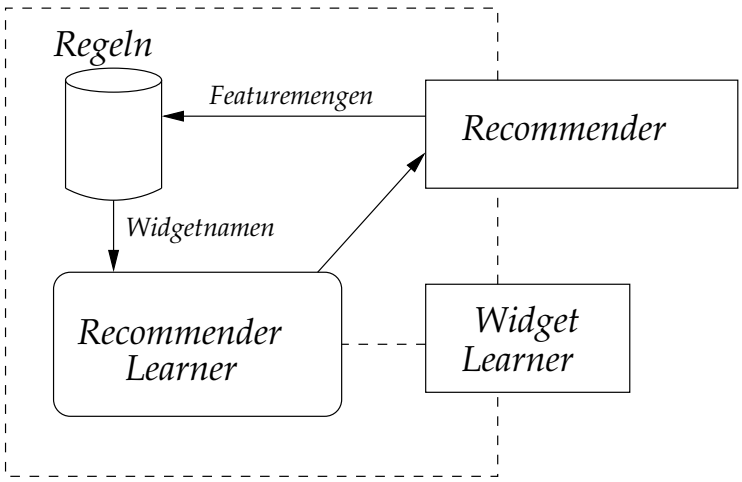


Abbildung 6.5: Schematischer Ablauf innerhalb des RecommenderLearner

Evaluation verschiedener Algorithmen in diesem Szenario bleibt für zukünftige Arbeiten offen.

Die `Recommender` Schnittstelle wird derzeit im `SIHARecommenderBridge` Bundle implementiert. Dieses ruft über einen JAX-WS Webservice den im firmeninternen Netz vorhandenen Recommender auf, welcher kollaboratives Filtern nutzt, um neue Vorhersagen zu erhalten.

6.4 Kontext

Die Kontextschnittstelle besteht im Wesentlichen aus zwei Elementen, dem `ContextAggregator` und der `ContextBridge`. Die Methoden dieser Schnittstellen sind bereits in Unterabschnitt 6.1.1 näher erläutert worden. Während der Initialisierung erzeugt der Aggregator ein Kontextelement unter dem die restlichen Kontextelemente und -werte bei Bedarf angelegt werden. Kommt eine Registrierungs- oder Aktualisierungsanfrage, wird diese an die Kontextbrücke weitergeleitet. Wird das Kontextelement, das die aktuelle Seite eines Nutzers repräsentiert, aktualisiert, werden die weiteren Features extrahiert und deren Kontextwerte entsprechend erneuert (siehe Abbildung 6.6).

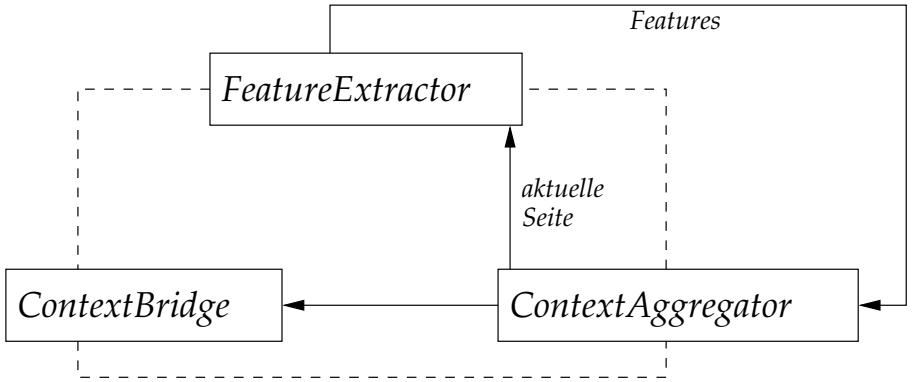


Abbildung 6.6: Schematischer Ablauf in der Kontextschnittstelle

Während der Implementierung des Demonstrators ergab sich ein Problem aus dem Entwicklungsstadium der Middleware. Ein Fehler in MUSIC verhinderte, dass die Änderung von Kontextwerten unterhalb geschachtelter Kontextelemente nicht dazu führte dass eine Adaption ausgelöst wurde. Dieser Fehler wurde umgangen, indem die Methoden für die benutzerbasierten Kontextelemente den Benutzernamen direkt in den Namen des Kontextwertes mit einflochten und daraus einen globalen Kontext erzeugten. So würde aus dem Kontextwert `external/stephan/currentPage` beispielsweise `external/stephan_currentPage`, wobei der Schrägstrich, ähnlich wie bei Verzeichnisangaben, die einzelnen Kontextelemente voneinander abtrennt. Nach Korrektur der Middleware bleibt so ein Wechsel möglich, ohne die Semantik der Funktionen zu ändern.

In Abschnitt 5.4 wurde die grundsätzliche Idee erläutert, wie die Kontextvorhersage in den Demonstrator integriert werden könnte. Im Detail könnte diese Komponente Instanzen mehrere Vorhersagealgorithmen bereithalten und bei jeder Änderung eines Kontextwertes diesen entsprechend weitergeben. Dies würde, wie an einigen anderen Stellen in der Architektur auch, dazu führen, dass pro Nutzer pro Kontextwert ein Verfahren gestartet werden würde. Ebenso wie in den anderen Fällen könnte dies zu Einbußen in der Gesamtleistung führen, so dass spätestens durch diesen Schritt die Nutzung der Verteilungsmöglichkeiten der MUSIC Plattform nötig wäre. Auf den Einbau der Vorhersage in den Demonstrator wurde im Rahmen dieser Arbeit aus Zeitgründen verzichtet.

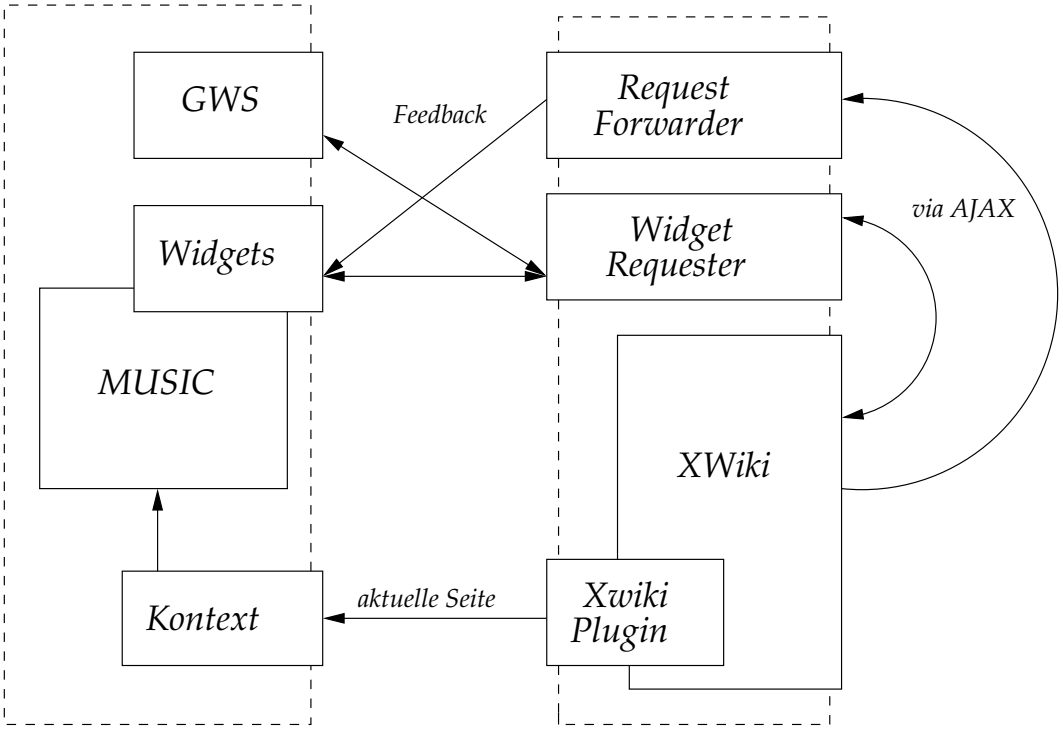


Abbildung 6.7: Schematischer Ablauf inklusive Wiki

6.5 XWiki Plugin

Im letzten Abschnitt dreht sich dieses Kapitel um XWiki und der Darstellung einzelner Panels. Wie bereits in Abschnitt 5.2 erläutert, ist das Panel in der Lage verschiedene Widgets darzustellen. Zunächst wurde mit Hilfe von Velocity solch eines realisiert. Allerdings ist die Zeit, die die Middleware benötigt, um eine Adaption durchzuführen, zu hoch, als dass es möglich gewesen wäre nach einer Anfrage zu warten bis diese abgeschlossen ist, und erst dann zu beginnen die Seite (und damit das Panel) zu erzeugen.

Um dieses Problem zu lösen, wurde auf AJAX zurückgegriffen. Mit dieser Technik ist es möglich den Inhalt des Widgets zu aktualisieren, sobald die Adaption abgeschlossen ist. Um das zu realisieren nutzt AJAX eine „Polling“ genannte Technik,

bei der der Server ständig nach neuen Inhalten gefragt wird.[3] Das Hauptproblem dieser Lösung ist, dass Ajax Anwendungen durch den Browser in der Regel nicht erlaubt wird, auf die Webservices unter anderen Domains zuzugreifen. Selbst der Zugriff auf einen Applikationsserver, der auf der gleichen Server läuft, aber über einen anderen Port erreichbar ist, ist nicht ohne Weiteres möglich. Um auch dieses Problem zu lösen, wurde ein weiterer Webservice geschaffen, der, wenn er eine Anfrage erhält, diese an den entsprechenden Server weiterleitet und das Ergebnis der Anfrage zurückgibt. Zusätzlich wird diese Anwendung in den Applikationsserver des Wikis installiert und erlaubt so, dass die Ajax Anwendung darauf zugreifen kann.

Dieses Problem ist nicht allein durch Ajax verschuldet, sondern liegt schon in der Behandlung von Javascript durch die Browser. So zeigt sich dieses Problem ebenfalls in dem Fall, dass ein Widget Feedback erhält. Auch für diesen Fall wurde ein Webservice geschaffen, der Anfragen an die entsprechend installierten Feedbackwidgets weiterleiten kann, der aber im Gegensatz zu dem vorherigen nicht auf eine Antwort warten muss.[4]

Neben dem Panel fängt ein XWiki Plugin jeden Seitenaufruf innerhalb des Wikis ab und gibt diesen an den Demonstrator weiter, um den Kontext zu aktualisieren. Abbildung 6.7 veranschaulicht das Verhalten des Wikis.

[3] Neben Polling gibt es auch weitere Methoden, die allerdings hier nicht in Betracht gezogen wurden, da sie für eine einfache Demonstration zu komplex umzusetzen wären. Die auf dem Server dadurch entstehende Auslastung allerdings bedarf einer genaueren Betrachtung, sobald dieses System in ein praktisch verwendbares übergeht.

[4] Die Implementierung dieser Weiterleitungswidgets ist trivial, daher entfällt an dieser Stelle eine genauere Erläuterung.

Kapitel 7

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit teilte sich zum einen in die Untersuchung der Methoden zur Kontextvorhersage und zum anderen in den Entwurf eines Demonstrators für die kontext-sensitive MUSIC Middleware.

Für Ersteres wurde die zum Thema bereits existierende Literatur vorgestellt. Daraufhin wurden die Verfahren genauer untersucht, die in dem für den zweiten Teil entworfenen Szenario einsetzbar sind. Die Implementierung der gewählten Algorithmen, auf die sich die Evaluation stützt, wurde anhand der beschriebenen Funktionsweisen vorgenommen. Die Ergebnisse der Evaluation auf mehreren verschiedenen Datensätzen zeigte Überraschendes: Algorithmen zu deren Gunsten die theoretische Grundlage oder lange praktische Verwendung sprach, schnitten oft nicht besser als der auf N-Grammen beruhende Ansatz ab. Insgesamt zeigte sich, dass im für den Demonstrator gewählten Szenario Kontextvorhersage durchaus einsetzbar ist. Die Vorteile, die sich durch die Anwendung dieser Verfahren ergeben, zeigen sich beispielsweise wenn das ursprüngliche Szenario auf mobile Geräte ausgeweitet wird. Auf solchen Geräten ist der Platz für die Anzeige sehr beschränkt, so dass nur ein Widget angezeigt wird. Der vorhergesagte Wert könnte in einem solchen Fall die Auswahl des anzuzeigenden Widgets beeinflussen, um wirklich das für den Benutzer nützlichste auszuwählen.

Die Realisierung des Demonstrators basierte auf der durch OSGI vorgelebten Aufteilung der Applikation in Komponenten und die daraus resultierende Fähigkeit, einzelne Komponenten leicht austauschen zu können. Nachdem diese Grundsatzentscheidung getroffen wurde, konnte der Aufbau entwickelt werden, der die grundlegende Frage beantworten musste, wie das Wiki, das als Webapplikation in einem Applikationsserver läuft, mit dem auf OSGI basierenden MUSIC System zusammenarbeiten kann. Auf Grund der vorgestellten Überlegungen zum Aufbau

ist es dem Demonstrator möglich unabhängig vom Wiki zu funktionieren. Die einzelnen Komponenten konnten nach Festlegung der Schnittstellen unabhängig in einzelnen Bundles entwickelt werden und die Funktionsweise des Demonstrators ist in sehr vielen Teilen innerhalb der verwendeten Architektur frei austauschbar. Gleichzeitig ist es möglich, Widgets schnell und einfach zu implementieren, was an zweien demonstriert wurde. Zudem wurde aufgezeigt wie Kontextvorhersage in den Demonstrator integriert werden könnte. Darüber hinaus lässt sich festhalten, dass die MUSIC Plattform durchaus geeignet ist, um in anderen Bereichen als dem eigentlich angedachten eingesetzt zu werden.

7.1 Ausblick

Es verbleibt noch der Ausblick auf weiter anfallende Arbeit sowohl an dem Demonstrator, als auch an dem Problem der Kontextvorhersage. Der entwickelte Demonstrator kann zwar die einzelnen Fähigkeiten der MUSIC Middleware zeigen, zu einer richtigen Demonstration gehört aber weiterhin die Entwicklung eines vollständigen Szenarios, in dem gezeigt wird, dass verschiedene Benutzer einen verschiedenen Kontext haben können, und wie MUSIC sich an diesen anpassen kann.

Die offensichtlichste Erweiterung des Demonstrators ist, einzelne Kontextwerte vorherzusagen und Widgets zu kreieren, die von diesen Vorhersagen praktischen Gebrauch machen können. Da das prinzipiell nötige Vorgehen in dieser Arbeit bereits skizziert wurde, verbleibt die aus Zeitgründen nicht mehr vollzogene Realisierung. Weiterhin dient die folgende Aufzählung als Übersicht an interessanten Punkten zur Erweiterung des Demonstrators:

- Die dynamische Auswahl von Widgets basiert zur Zeit auf einem statischen Regelsystem. Dies könnte mit Hilfe von Regellernalgorithmen erweitert werden. Es verbleibt dabei auch zu evaluieren, inwiefern Benutzerpräferenzen durch Regellernverfahren modelliert werden können. Denkbar an dieser Stelle wären alle Typen von Klassifikationsalgorithmen.
- Ähnliches gilt für die Umsetzung der Utility-Funktion. Auch hier wäre der Einsatz maschineller Lernverfahren denkbar, um die Funktion nicht nach starren Regeln auszuwerten, sondern an Benutzerpräferenzen anzupassen. Es bleibt an dieser Stelle aber zu beachten, dass durch die häufige Auswertung

der Utility-Funktion diese keine großen Performanzanforderungen aufweisen sollte.

Ein weiterer Ansatzpunkt ist die Evaluation in Form einer Nutzerstudie, wie Benutzer auf Systeme, die proaktiv handeln reagieren. Insbesondere ist für eine Algorithmenevaluation der Punkt entscheidend, welchen prozentualen Anteil von falsch vorhergesagten Werten der Benutzer tolerieren wird. Eine weitere interessante, wenn auch vielleicht eher philosophische Frage ist, wie Benutzer auf Maschinen reagieren, die auf gesammelten Kontextdaten arbeiten. Die aktuell aufbrandende Diskussion über den von Google entwickelten Browser Chrome zeigt deutlich, dass eine gewisse Skepsis solchen Systemen gegenüber besteht. Diesen Aspekt genauer zu untersuchen wäre sicherlich nicht nur für die Informatik ein interessantes Feld.

Das Hidden Markov Model, das in der Evaluation im Rahmen dieser Arbeit eher schlecht abschloß, bietet, im Gegensatz zu den anderen untersuchten Algorithmen, eventuell die Möglichkeit anhand der versteckten Zustände weitere Aussage zu tätigen. So könnten diese als „Situation“ in der sich der Benutzer befindet interpretiert werden und dadurch weiteren Aufschluss über dessen Verhalten liefern.

Das Problem der Kontextvorhersage bedarf insgesamt einer theoretischen Grundlage. Während Feder *et al.* [15] durch die Einführung der FS Vorhersagbarkeit begannen, scheint die Auseinandersetzung mit diesem Problem auf formaler Ebene nicht stark vorangegangen zu sein. Das Gebiet der statistischen Zeitreihenvorhersage geht in eine verwandte Richtung und eine grundlegende formale Definition sollte auf den dort gemachten Überlegungen basieren. Zu dieser Grundlage zählen allerdings nicht nur eine formale Definition, sondern auch eine genauere analytische Beschäftigung mit den zur Evaluation herangezogenen und vorhergesagten Daten und deren Klassifikation. Durch eine derart gerichtete Forschung könnte eine allgemeingültige Evaluation der Vorhersagealgorithmen erfolgen. Die hier erfolgte Evaluation beschränkte sich auf eine relativ kleine Auswahl von Algorithmen. Eine etwas breiter angelegte Auswahl könnte weiterhin noch neuronale Netze oder verschiedene Klassifikationsalgorithmen beinhalten.

Literaturverzeichnis

[1] Ailisto, Heikki ; Alahuhta, Petteri ; Haataja, Ville ; Kylloenen, Vesa ; Lindholm, Mikko: Structuring context aware applications: Five-layer model and example case. In: *Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing*, 2002

[2] Alia, Mourad ; Eliassen, Frank ; Hallsteinsen, Svein ; Stav, Erlend: MADAM: towards a flexible planning-based middleware. In: *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. Shanghai, 2006, S. 96

[3] Anagnostopoulos, Christos ; Mpougiouris, Panagiotis ; Hadjiefthymiades, Stathes ; Ilisia, Panepistimioupolis: Prediction intelligence in context-aware applications. In: *Proceedings of the 6th international conference on Mobile data management*, 2005

[4] Baldauf, Matthias ; Dustdar, Schahram ; Rosenberg, Florian: A Survey on Context-Aware systems. In: *International Journal of Ad Hoc and Ubiquitous Computing* 2 (2007), Jun, Nr. 4, S. 263–277

[5] Baum, Leonard E. ; Petrie, Ted ; Soules, George ; Weiss, Norman: A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. In: *The Annals of Mathematical Statistics* 41 (1970), Feb, Nr. 1, S. 164 – 171

[6] Bhattacharya, Amiya ; Das, Sajal K.: LeZi-Update: An Information-theoretic framework for personal mobility tracking in PCS networks. In: *ACM/Kluwer Wireless* (2002)

[7] Bishop, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. New ed. Springer, Berlin, 2 2008. – ISBN 9780387310732

[8] BITKOM: *Mehr Handys als Einwohner in Deutschland*. Aug 2006. – URL http://www.bitkom.org/de/presse/43408_40990.aspx

[9] Brown, Peter J.: The Stick-e Document: a Framework for Creating Context-aware Applications. In: *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communication*, URL http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume8/issue2/2point1.pdf, 1996, S. 361–365

[10] Brown, Peter J. ; Bovey, John D. ; Chen, Xian: Context-aware Applications: from the Laboratory to the Marketplace. In: *IEEE Personal Communications* Bd. 4. Oktober 1997, S. 58 – 64

[11] Chen, Guanling ; Kotz, David: A Survey of Context-Aware Mobile Computing Research / Darthmouth College. Hanover, NH, USA, 2000 (TR2000-381). – Forschungsbericht

[12] Cook, Diane J. ; Youngblood, Michael ; Heierman, Edwin O. ; Gopalratnam, Karthik ; Rao, Sira ; Litvin, Andrey ; Khawaja, Farhan: MavHome: An Agent-based Smart Home. In: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003 (PerCom 2003)*, 2003, S. 521– 524

[13] Dey, Anind K.: Context-Aware Computing: The CyberDesk Project / AAAI 1998 Spring Symposium on Intelligent Environments. 1998 (SS-98-02). – Forschungsbericht. – 51 – 54 S

[14] Dey, Anind K. ; Abowd, Gregory D.: Towards a Better Understanding of Context and Context-Awareness / Georgia Institute of Technology. URL http://www.it.usyd.edu.au/~bob/IE/99-22.pdf, 1999 (GIT-GVU-99-22). – Forschungsbericht

[15] Feder, Meir ; Merhav, Neri ; Gutman, Michael: Universal Prediction of Individual Sequences. In: *IEEE Transactions on Information Theory* 38 (1992), Nr. 4

[16] Fowler, Martin: *Inversion of Control Containers and the Dependency Injection pattern*. Jan 2004. – URL http://www.martinfowler.com/articles/injection.html

[17] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John M.: *Design Patterns: Elements of Reuseable Object-Oriented Software*. Addison-Wesley Professional, Nov 1994

[18] Gasieniec, Leszek ; Karpinski, Marek ; Plandowski, Wojciech ; Rytter, Wiojciech: Efficient Algorithms for Lempel-Ziv Encoding. In: *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory*, 1996

[19] Geihs, Kurt ; Ullah Khan, Mohammad ; Reichle, Roland ; Solberg, Arnor ; Hallsteinsen, Svein ; Merral, Simon: Modeling of component-based adaptive distributed applications. In: *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, 2006, S. 718–722

[20] Gopalratnam, Karthik ; Cook, Diane J.: Active LeZi: An Incremental Parsing Algorithm for Device Usage Prediction in the Smart House. In: *Proceedings of the Florida Artificial Intelligence Research Symposium*, May 2003, S. 38 – 42

[21] Henricksen, Karen ; Induslka, Jadwiga ; McFadden, Ted ; Balasubramaniam, Sasitharan: Middleware for Distributed Context-Aware Systems. In: *7th International Symposium on Distributed Objects and Applications (DOA)* Bd. 3760 of LCNS. Agia Napa, Cyprus : Springer, Nov 2005

[22] Hofer, Thomas ; Schwinger, Wieland ; Pichler, Mario ; Leonhartsberger, Gerhard ; Altmann, Josef ; Retschitzegger, Werner: Context-awareness on mobile devices - the hydrogen approach. In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003

[23] Horvitz, Eric ; Koch, Paul ; Kadie, Carl M. ; Jacobs, Andy: Coordinate: Probabilistic Forecasting of Presence and Availability. In: *Proceedings of the Eighteenth Conference on Uncertainty and Artificial Intelligence*, Morgan Kaufmann Publishers, 2002, S. 224–233

[24] Hull, Richard ; Neaves, Philip ; Bedford-Roberts, James: Towards situated computing. In: *First International Symposium on Wearable Computers*, 1997, S. 146 – 153

[25] Johnson, Ralph E. ; Foote, Brian: Designing Reusable Classes. In: *Journal of Object-Oriented Programming* 1 (1988), Jun, Nr. 2, S. 22 – 35

[26] Jurasfky, Daniel S. ; Martin, James H.: *Speech and Language Processing*. Prentice-Hall, 2000

[27] Knuth, Donald E.: *The Art of Computer Programming*. Bd. 1. Reading, Massachusetts : Addison-Wesley, 1997

[28] Loughran, Steve: Towards an Adaptive and Context Aware Laptop / Hewlett-Packard Labs. 2001 (HPL-2001-158). – Forschungsbericht

[29] Manning, Christopher D. ; Schütze, Hinrich: *Foundations of statistical natural language processing*. MIT Press, 1999

[30] Mayrhofer, Rene: *An Architecture for Context Prediction*, Johannes Kepler University of Linz, Austria, Dissertation, 2004

[31] Mayrhofer, Rene: Context Prediction based on Context Histories: Expected Benefits, Issues and Current State-of-the-Art. In: Prante, T. (Hrsg.) ; Meyers, B. (Hrsg.) ; Fitzpatrick, G. (Hrsg.) ; Harvel, L. D. (Hrsg.): *Proc. ECHISE 2005: 1st International Workshop on Exploiting Context Histories in Smart Environments*, URL http://www.mayrhofer.eu.org/downloads/publications/ECHISE2005-Context-Prediction-based-on-Context-Histories.pdf, May 2005

[32] Mikalsen, Marius ; Floch, Jacqueline ; Paspallis, Nearchos ; Papadopoulos, George A. ; Ruiz, Pedro A.: Putting Context in Context: The Role and Design of Context Management in a Mobility and Adaptation Enabling Middleware. In: *Proceedings of the 7th International Conference on Mobile Data Management, 2006. MDM 2006.* IEEE (Veranst.), May 2006, S. 76–76

[33] Müller-Hannemann, Matthias: *Einführung in die Bioinformatik*. Jul 2006. – URL http://zuseex.algo.informatik.tu-darmstadt.de/lehre/2006ss/bioinf/mat/bioinformatik.pdf

[34] Mozer, Michael C.: The Neural Network House: An Environment that Adapts to its Inhabitants. In: Coen, M. (Hrsg.): *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*. Menlo Park, CA : AAAI Press, 1998, S. 110 – 114

[35] Mozer, Michael C.: *Lessons from an adaptive house*. S. 273 – 294. In: Cook, Diane J. (Hrsg.) ; Das, Sajal K. (Hrsg.): *Smart environments: Technologies, protocols, and applications*. Hoboken, NJ : J. Wiley & Sons, 2005

[36] Mozer, Michael C. ; Dodier, Robert H. ; Vidmar, Lucky: The Neurothermosthat: Adaptive Control of Residential Heating Systems. In: Mozer, Michael C. (Hrsg.) ; Jordan, M.I. (Hrsg.) ; Petsche, T. (Hrsg.): *Advances in Neural Informatioln Processing Systems* Bd. 9. MIT Press, 1997, S. 953 – 959

[37] Nurmi, Petteri ; Martin, Miquel ; Flanagan, John A.: Enabling proactiviness through Context Prediction. In: *Proc. Workshop on Context Awareness for Proactive Systems*, URL `http://www.cs.helsinki.fi/u/ptnurmi/papers/caps2005.pdf`, 2005

[38] Pascoe, Jason: Adding Generic Contextual Capabilities to Wearable Computers. In: *2nd International Symposium on Wearable Computers (ISWC 1998)*, 1998

[39] Petzold, Jan ; Bagci, Faruk ; Trumler, Wolfgang ; Ungerer, Theo: Global and Local State Context Prediction. In: *Proceedings of the Workshop on Artificial Intelligence in Mobile Systems 2003 (AIMS 2003) in Conjunction with the Fifth International Conference on Ubiquitous Computing 2003*, URL `http://www.informatik.uni-augsburg.de/de/lehrstuehle/sik/publikationen/papers/2003_aims_pet/2003_aims_pet_pdf.pdf`, 2003

[40] Petzold, Jan ; Pietzowski, Andreas ; Bagci, Faruk ; Trumler, Wolfgang ; Ungerer, Theo: Prediction of Indoor Movements Using Bayesian Networks. In: *Location- and context-awareness. International workshop*, Springer, 2005

[41] Prekop, Paul ; Burnett, Mark: Activities, Context and Ubiquitous Computing. In: *Special Issue on Ubiquitous Computing Computer Communications* 26 (2003), Nr. 11

[42] Preuveneers, Davy ; Berbers, Yolande: Adaptive Context Management using a Component-based Approach. In: *5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)* Bd. 3543 of LCNS. Athen : Springer, Jun 2005, S. 14–26

[43] Rabiner, Lawrence R.: A tutorial on hidden Markov models and selected applications inspeech recognition. In: *Proceedings of the IEEE* Bd. 77, Feb 1989, S. 257 – 285

[44] Rodden, Tom ; Chervest, Keith ; Davies, Nigel ; Dix, Alan: Exploiting Context in HCI Design for Mobile Systems. In: *Workshop on Human Computer Interaction with Mobile Devices*, 1998

[45] Russel, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 2nd. Prentice Hall, 2005

[46] Ryan, Nick ; Pascoe, Jason ; Morse, David: Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant. In: Gaffney, V. (Hrsg.) ; Leusen, M. van (Hrsg.) ; Exxon, M. (Hrsg.): *Computer Applications in Archaeology*. 1997

[47] Schilit, Bill N. ; Adams, Norman ; Want, Roy: Context-Aware Computing Applications, URL `http://nano.xerox.com/want/papers/parctab-wmc-dec94.pdf`, 1994

[48] Schmidt, Albrecht ; Beigl, Michael ; Gellersen, Hans-W.: There is more to context than location. In: *Computer and Graphics* 23 (1999), Nr. 6, S. 893 – 901

[49] Sigg, Stephan ; Haseloff, Sandra ; David, Klaus: Prediction of Context Time Series. In: *5th Workshop on Applications of Wireless Communications*, 2007

[50] Statistisches Bundesamt: *80-Prozent-Marke bei der Handy-Ausstattung überschritten*. Mai 2007. – URL `http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Presse/pm/zdw/2007/PD07__019__p002.psml`

[51] Viterbi, Andrew J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE Transactions on Information Theory* 13 (1967), Apr, Nr. 2, S. 260–269

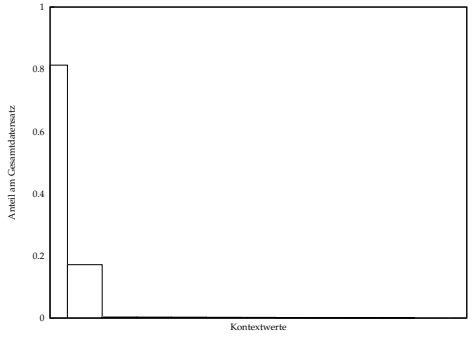
[52] Ward, Andy ; Jones, Alan ; Hopper, Andy: A New Location Technique for the Active Office. In: *IEEE Personal Communications* Bd. 4. 1997, S. 42 – 47

[53] Weiser, Mark: The computer of the twenty-first century. In: *Scientific American* 1496 (1991), S. 94 – 100

[54] Ziv, Jacob ; Lempel, Abraham: Compression of individual sequences via variable rate coding. In: *IEEE Transactions on Information Theory* IT-24 (1978), S. 530 – 536

Anhang A

Verteilung der Mayrhoferdaten



(a) Der dargestellte Sensor ist der für die Anzahl der aktiven Rechner in der Umgebung. Es gibt 12 verschiedene Kontextwerte und es ist offensichtlich, dass zwei Werte fast alle Werte ausmachen. Die Werte sind nach Häufigkeit sortiert.

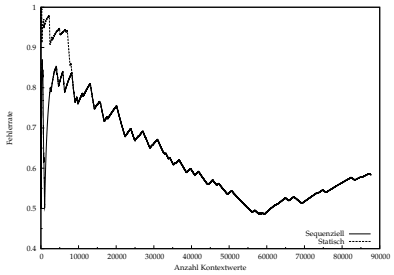
(b) Der Sensor ist diesem Bild gibt die MAC Adresse des WLAN Routers an, zu dem der Laptop verbunden war. Auch hier sind die Werte nach Auftrittshäufigkeit sortiert. Zwei Kontextklassen dominieren ebenfalls den Datensatz.

Abbildung A.1

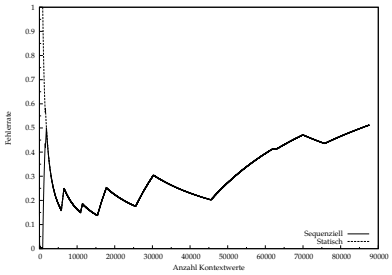
Anhang B

Evaluationsergebnisse

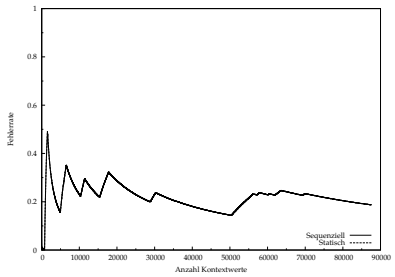
B.1 Majority Vote



(a) Ergebnisse auf dem aktiven Fenster Datensatz



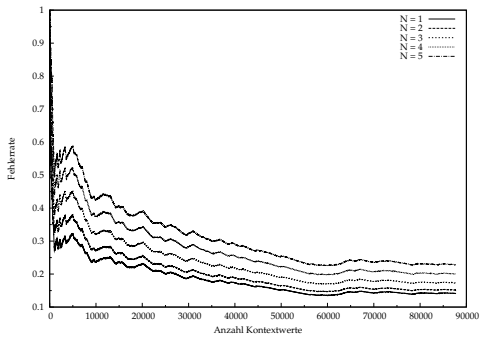
(b) Ergebnisse auf dem aktuellen Router Datensatz



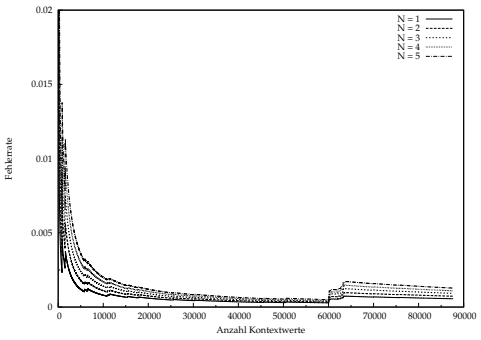
(c) Ergebnisse auf dem Anzahl der Peers Datensatz

Abbildung B.1

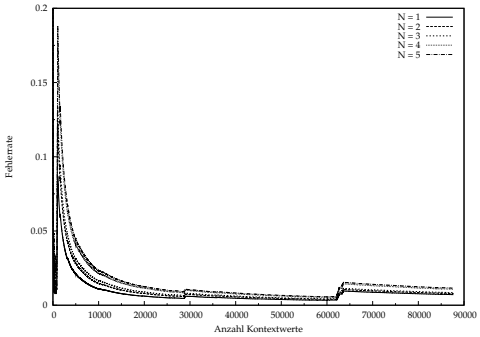
B.2 *N*-Gramm Algorithmus



(a) Ergebnisse auf dem aktiven Fenster Datensatz



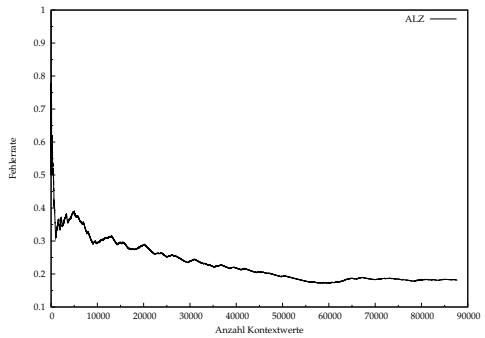
(b) Ergebnisse auf dem aktuellen Router Datensatz



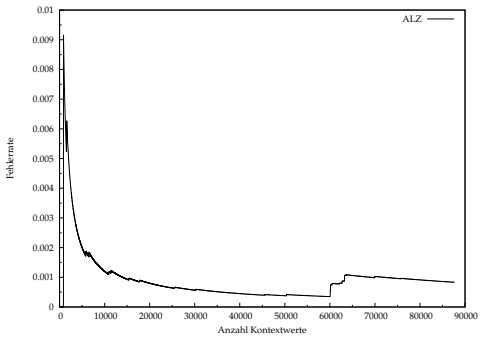
(c) Ergebnisse auf dem Anzahl der Peers Datensatz

Abbildung B.2

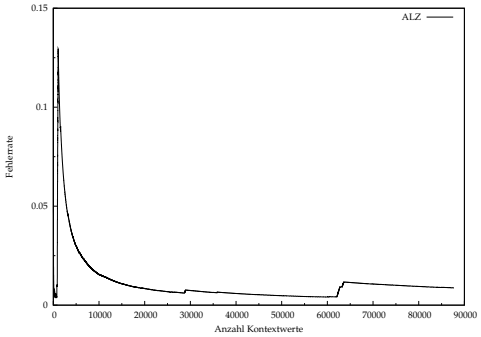
B.3 Active Lempel-Ziv



(a) Ergebnisse auf dem aktiven Fenster Datensatz



(b) Ergebnisse auf dem aktuellen Router Datensatz



(c) Ergebnisse auf dem Anzahl der Peers Datensatz

Abbildung B.3

Anhang C

Zusammenfassung der Bundles

C.1 ContextAggregator

Exportierte Pakete:	—
Importierte Pakete:	com.caucho.burlap.server
	de.emld.music.interfaces
	javax.servlet
	org.apache.log4j
	org.istmusic.mw.context
	org.osgi.service.http
Exportierte Services:	de.emld.music.interfaces.ContextAggregator
Importierte Services:	org.osgi.service.http.HttpService
	de.emld.music.interfaces.ContextBridge

Tabelle C.1: Zusammenfassung des ContextAggregator Bundles

C.2 WidgetRegistry

Exportierte Pakete:	—
Importierte Pakete:	de.emld.music.interfaces
	org.apache.log4j
Exportierte Services:	de.emld.music.interfaces.WidgetRegistry
Importierte Services:	—

Tabelle C.2: Zusammenfassung des WidgetRegistry Bundles

C.3 ContextBridge

Exportierte Pakete:	—
Importierte Pakete:	de.emld.music.interfaces
	javax.servlet
	org.apache.log4j
	org.istmusic.mw.context
	org.istmusic.mw.context.event
	org.istmusic.mw.context.sensor
Exportierte Services:	org.istmusic.mw.context.sensor.ContextSensor
	de.emld.music.interfaces.ContextBridge
Importierte Services:	—

Tabelle C.3: Zusammenfassung des ContextBridge Bundles

C.4 RecommenderLearner

Exportierte Pakete:	de.emld.music.recommender
Importierte Pakete:	de.emld.music.interfaces
	org.apache.log4j
Exportierte Services:	de.emld.music.interfaces.FeatureExtractor
	de.emld.music.interfaces.WidgetLearnerService
Importierte Services:	de.emld.music.interfaces.ContextAggregator
	de.emld.music.interfaces.UserRegistry
	de.emld.music.recommender.Recommender

Tabelle C.4: Zusammenfassung des RecommenderLearner Bundles

C.5 SIHARecommenderBridge

Exportierte Pakete:	—
Importierte Pakete:	de.emld.music.interfaces
	de.emld.music.recommender
	org.apache.log4j
Exportierte Services:	de.emld.music.recommender.Recommender
Importierte Services:	—

Tabelle C.5: Zusammenfassung des SIHARecommenderBride Bundles

C.6 UserRegistry

Exportierte Pakete:	—
Importierte Pakete:	de.emld.music.interfaces
	org.apache.log4j
Exportierte Services:	de.emld.music.interfaces.UserRegistry
Importierte Services:	—

Tabelle C.6: Zusammenfassung des UserRegistry Bundles

Anhang D

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, im September 2008 Stephan Mehlhase