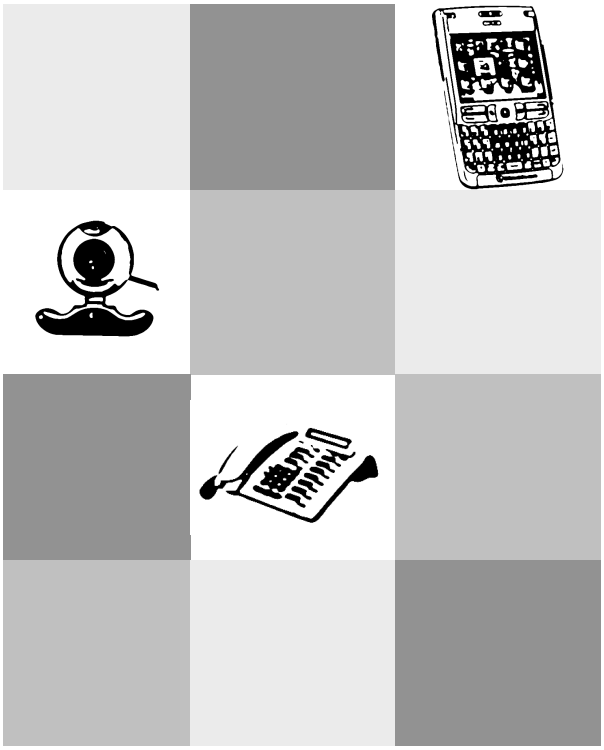
Universität Zürich
Institut für Informatik

Dynamic and Distributed
Information Systems

Framework für ein dynamisches Telefonsteuerungssystem



Diplomarbeit 5. September 2006

Daniel Imhof
von Reussbühl

Matrikelnr.: 99-911-646
da@imhof.com

Betreuer: **Peter Vorburger**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Univesität Zürich
http://www.ifi.unizh.ch/ddis

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, welche mitgeholfen haben, diese Arbeit zu verwirklichen.

Prof. Abraham Bernstein möchte ich danken, dass er mir die Gelegenheit geboten hat, eine Diplomarbeit in einem interessanten und spannenden Umfeld zu schreiben.

Einen speziellen Dank möchte ich an meinen Betreuer Peter Vorburger richten. Er stand in allen Phasen der Diplomarbeit immer zur Verfügung.

Zusammenfassung

Am Arbeitsplatz ist man verschiedenen Quellen der Störung ausgesetzt. Diese Störungen zu reduzieren, ist das Ziel dieser Arbeit. In früheren Versuchen wurde schon gezeigt, dass man mit Sensordaten und der Hilfe von Data Mining Algorithmen auf den Unterbrechbarkeitsgrad einer Person schliessen kann. Doch konnten diese Werte erst im Nachhinein ermittelt werden. Darum soll in dieser Arbeit ein Framework entwickelt werden, welches die ganzen Vorgänge in Echtzeit bewerkstelligen kann. Neben der Programmierung der Softwarekomponenten soll die ganze Applikation auch auf ihre Alltagstauglichkeit hin evaluiert werden.

Abstract

On the job one is exposed to different sources of disruption. The goal of this paper ist to reduce these disturbances. In earlier attempts it was already shown that one can determine the interruptability degree of a person with sensor data and the assistance of data mining algorithms. But these values could be acquired only afterwards. Therefore a framework has to be developed, which can manage the whole procedures in real time. Apart from the programming of the software components the whole application also has to be evaluated on its suitability for daily use.

Inhaltsverzeichnis

1

Einführung

Im normalen Büroalltag wird man leider zu häufig durch verschiedene Ereignisse von seiner eigentlichen Arbeit abgelenkt. Bei der Mehrheit davon handelt es sich um Telefonanrufe. Leider ist man selten in der Lage, im Voraus zu entscheiden, ob der Anruf wichtig ist oder nicht. Zu den wenigen Möglichkeiten sich davor zu schützen, gehören das Umleiten auf eine Mailbox oder falls vorhanden auf das Sekretariat. Entweder man leitet einfach alle eingehen Anrufe um, oder man man aktiviert die Umleitung nur bei Bedarf. Eine solche Massnahme ist aber höchst ineffizient. Zu schnell vergisst man, die Weiterleitungen zu aktivieren bzw. zu deaktivieren.

Von Vorteil wäre jetzt eine Software, welche anhand verschiedener Parameter selbständig entscheiden könnte, ob der Benutzer gestört werden will oder nicht. Die Software könnte dann, anhand der ermittelten Unterbrechbarkeit, das Telefon so steuern, dass der Anruf automatisch am richtigen Ort landet.

In [Zurfluh, 2004] wurde schon gezeigt, dass es möglich ist, anhand von Sensordaten die Unterbrechbarkeit einer Person zu ermitteln. In diesem Experiment wurden die aufgezeichneten Video- und Audiodaten aber erst nach dem Eintreffen eines Ereignisses analysiert. So konnte die Unterbrechbarkeit erst im Nachhinein ermittelt werden, was für unsere Anwendung noch nicht ausreichend ist.

Mit dem jetzigen Experiment soll durch verschiedene Massnahmen erreicht werden, dass man in Echtzeit den aktuellen Zustand der Unterbrechbarkeit ermitteln kann.

1.1 Experimentablauf

Das Experiment durchläuft während der ganzen Dauer drei verschiedene Phasen. Diese betreffen einerseits die Interaktion des Benutzers mit dem Programm und andererseits den eigentlichen Ablauf der Software. Im folgenden werden die verschiedenen Phasen aus den beiden unterschiedlichen Sichten des Benutzers und der Software dargestellt.

1.1.1 Benutzersicht

Aus der Sicht des Benutzers laufen die drei Phasen folgendermassen ab:

1. Bei jedem Eintreffen eines Ereignisses, in unserem Fall ein Telefonanruf, wird dem Benutzer auf einem mobilen Endgerät eine Eingabemaske präsentiert. Auf diesem Gerät muss er dann annotieren, inwiefern er durch diesen Event bzw. Anruf gestört wurde.

In dieser Phase merkt der Benutzer noch keine Veränderung im Telefonverhalten. Alle Anrufe kommen noch zum Telefon durch.

2. Die zweite Phase läuft für den Anwender genau gleich ab wie die erste Phase. Einen Unterschied gibt es nur für die Software.

   Die Telefonanrufe kommen auch weiterhin zum Benutzer.

3. In der dritten und letzten Phase fällt das Annotieren weg. Der Benutzer muss also keine Eingaben mehr vornehmen.

   Auch sollte der Benutzer jetzt ein verändertes Verhalten des Telefons feststellen. Denn nur noch die als wichtig klassifizierten Anrufe kommen durch.

1.1.2 Softwaresicht

Aus der Sicht des Software laufen die drei Phasen folgendermassen ab:

1. In der ersten Phase werden bei jedem Eintreffen eines Ereignisses die fortlaufend aufgezeichneten Originale der Video- und Audiodaten gespeichert. Gleichzeitig werden auf diesen Daten Berechnungen durchgeführt, welche für das spätere Reasoning verwendet werden. Dazu kommen dann noch die Annotationsdaten des Benutzers. Dies alles wird zusammen mit einem Zeitstempel persistent gespeichert.

   Da für ein sinnvolles Online-Reasoning zu diesem Zeitpunkt noch zu wenig Daten zur Verfügung stehen, werden für die Berechnung der Unterbrechbarkeit in dieser Phase noch statische Regeln verwendet.

2. In Phase zwei werden weiterhin alle Daten wie in Phase eins gespeichert. Aber da sich inzwischen einige Daten aus der Vergangenheit angesammelt haben, wird auf die statischen Regeln verzichtet und es kommt ab diesem Zeitpunkt ein Onlinereasoning für die Berechnung der Unterbrechbarkeit zum Einsatz.

3. In der dritten Phase werden die Daten zwar weiterhin gesichert, aber für die Berechnung der Unterbrechbarkeit kommen sie nicht mehr zur Anwendung. Durch die jetzt fehlenden Annotationsdaten würde das keinen Sinn mehr machen. Das Reasoning erfolgt weiterhin online.

1.1.3 Experimentatorsicht

Der Experimentator hat dafür zu sorgen, dass die Software immer im richtigen Modus läuft. Das heisst, er muss den Wechsel zwischen den einzelnen Phasen von sich aus veranlassen. Dieser Entscheid soll zusammen mit dem Benutzer gefällt werden, denn dieser weiss auch über die Anzahl eingegangener Anrufe bescheid.

Um möglichst optimale Resultate für das Experiment zu bekommen, müssen die auf der Festplatte gespeicherten Daten regelmässig analysiert werden. Dies geschieht wie in [Zurfluh, 2004] beschrieben. So kann z.B. die optimale Zeit für das Reasoning bestimmt und gegebenenfalls angepasst werden oder es können andere Merkmale der Audiodaten ermittelt werden.

2

Anforderungen

Damit das Experiment als erfolgreich angesehen werden kann und auch entscheidende Vorteile gegenüber [Zurfluh, 2004] bringt, müssen einige Anforderungen erfüllt werden. Diese beziehen sich entweder auf die Interaktion mit dem Benutzer oder auf die Applikation selbst.

2.1 Benutzerseitige Anforderungen

Um es dem Benutzer so einfach wie möglich zu machen, soll er so wenig wie möglich mit der Software in Kontakt kommen. Aber gewisse Eingaben durch den Benutzer sind erforderlich. Und dies soll möglichst ohne grosse Beeinträchtigung der Arbeit des Benutzers stattfinden.

2.1.1 Bedienbarkeit

Die Software selbst wird der Anwender während des laufenden Betriebes nicht zu Gesicht bekommen. Nachdem sie einmal gestartet wurde, muss nur noch bei den zwei folgenden Phasenwechseln ein Parameter verändert werden. Aber dieser Schritt erfolgt durch den Experimentator, welcher im Normalfall nicht mit dem Benutzer identisch ist.

Der wichtigste Punkt im Sinne der Bedienbarkeit ist für den Anwender sicher das Annotieren. Da es am einfachsten ist, wenn er diese Daten von überall her eingeben kann, ist der Einsatz von mobilen drahtlosen Geräten wie einem Pocket PC erforderlich. Dadurch wird es möglich, dass er das Gerät bei sich tragen kann oder an einem zentralen Ort im Büro deponieren kann. Damit reduziert sich die Interaktion des Benutzers mit der Software auf einen einfachen portablen Client.

2.1.2 Echtzeit

Ein wichtiger Punkt der nichts mit der Bedienung der Software an sich zu tun hat, ist die Anforderung der Echtzeitverarbeitung. Der Wert der Unterbrechbarkeit soll möglichst unmittelbar nach dem Eintreffen eines Telefonanrufes zur Verfügung stehen. Da durch den ermittelten Wert das Verhalten des Telefons gesteuert werden soll, sind Verzögerungen in der Verarbeitung ein grosses Problem.

2.2 Softwareseitige Anforderungen

Auch für die Applikation gibt es einige wichtige Anforderungen, welche erfüllt werden müssen.

2.2.1 Erweiterbarkeit

Für die Applikation muss die Möglichkeit einer Erweiterung bestehen. Eine solche Anpassung der Software soll möglich sein, ohne an der Architektur selbst etwas zu ändern. Es gibt grundsätzlich zwei Bereiche, in denen die Erweiterbarkeit wichtig ist:

- eintreffende Ereignisse
- Methoden des Reasonings

In diesem Experiment werden nur eingehende Telefonanrufe als Ereignis betrachtet. Eine weitere Möglichkeit wären z.B. Anrufe über Skype oder SIP. Aber auch gänzlich andere Events wie das Klingeln an der Bürotür kommen in Frage.

Der zweite Bereich der Erweiterbarkeit betrifft das Reasoning. In diesem Experiment kommt je nach Phase eine bestimmte Methode zur Berechnung der Unterbrechbarkeit zum Zuge. Hier sollte die Möglichkeit bestehen, diese Methoden zu erweitern oder komplett auszutauschen.

2.2.2 Synchronität

Die Synchronität betrifft hauptsächlich die Annotation. Bei [Zurfluh, 2004] sind Datenaufzeichnung und Annotation noch unabhängig voneinander. Die beiden Bestände mussten bei der Analyse zuerst noch zusammengeführt werden. In unserem Experiment soll das geändert werden. Die Annotation soll immer dann möglich sein, wenn ein Ereignis eingetroffen ist. So soll gewährleistet werden, dass immer das richtige Ereignis bewertet wird. Damit alle Daten zusammenbleiben, sollen beide Teile, Bewegungsdaten und Annotation, in einem gemeinsamen Datensatz gespeichert werden.

2.2.3 abgeschlossene Transaktionen

Die Applikation soll sich ähnlich dem in der Datenbankwelt bekannten ACID-Prinzipes verhalten. Die vier Punkte, welche erfüllt werden müssen, lauten:

- Atomicity: Die Transaktion soll entweder komplett oder gar nicht durchgeführt werden. Auf unser Programm bezogen heisst das, dass nur komplette Datensätze gebraucht werden können. Wenn nicht alle Werte vorhanden sind, kann die Unterbrechbarkeit nicht ermittelt werden. Und wenn die Annotationsdaten fehlen, ist das Onlinereasoning nicht durchführbar.
- Consistency: Eine Transaktion hinterlässt nach Beendigung einen konsistenten Datenzustand.
- Isolation: Transaktionen, welche miteinander laufen, dürfen sich nicht gegenseitig stören. Da ein Datensatz nicht nachträglich geändert wird, ist das auch kein grosses Problem.
- Durability: Die aufgezeichneten Bild- und Audiodateien müssen so gespeichert werden, dass sie auch bei einem Neustart der Applikation noch vorhanden sind.

2.2.4 Konfigurierbarkeit

Da man nicht schon von Beginn weg entscheiden kann, mit welchen Parametern das Programm laufen soll, muss eine Möglichkeit geschaffen werden, verschiedene Werte auf eine einfache Weise anzupassen. Dazu gehören einerseits Parameter für die Aufnahmedauer und den Ort der Speicherung, andererseits soll dort auch die Phase gewählt werden können, in der sich das Experiment momentan befindet. Dazu kommen noch einige andere Werte, die dort gespeichert werden.

3

Designentscheidungen

Die Entscheidungen für das Design des Experimentes und insbesondere der Applikation, hängen sehr stark von der schon existierenden Software von Christian Singer [Singer, 2005] ab. Diese Applikation, welche im Rahmen einer Semesterarbeit entstanden ist, ist schon in der Lage, die aufgenommen Video- und Audiodaten so zu verarbeiten, dass man sie in einfacher Weise für das Reasoning weiterverwenden kann. Weiter sind auch schon Schnittstellen für den Zugriff per RMI[1] vorhanden, so dass sich auch diese Verwendung anbietet.

3.1 verwendete Technologien

Im folgenden Abschnitt werden die Technologien beschreiben, welche beim Aufbau des Experiments verwendet werden.

3.1.1 Programmiersprache

Da die Applikation von Christian Singer [Singer, 2005] schon in Java programmiert ist, liegt es nahe, für die Weiterentwicklung auch Java zu verwenden. Trotz der Verwendung von Java handelt es sich nicht um eine Multiplattform-Software. Durch den Einsatz weiterer Technologien, wie z.B. dem Java Media Framework (JMF) oder der Java Telephony API (JTAPI), ist der Einsatz auf Microsoft Windows beschränkt.

3.1.2 persistente Speicherung

Die bei einem eintreffenden Telefonanruf zu speichernden Daten müssen dauerhaft gespeichert werden. Einerseits werden sie ja für das Reasoning gebraucht und andererseits sollen sie für eine mögliche spätere Verwendung auch noch vorhanden sein.

Die vorhandene Applikation ist schon in der Lage, die vollständigen Daten auf die Festplatte zu speichern. Für die Archivierung der Aufzeichnungen ist dies sicher die optimale Lösung. Aber für Verwendung des Reasonings sind diese Daten nicht optimal. Für das Reasoning kommt

[1] Remote Method Invocation

die Software WEKA, welche im Kapitel 3.1.3 genauer beschrieben wird, zum Einsatz. WEKA akzeptiert folgende Dateiformate als Eingabe: ARFF[2], CSV[3] und Datenbanken. Beim Speichern der Daten werden zuerst die Audio- und Videodaten erfasst. Die Annotationsdaten werden erst kurze Zeit später hinzugefügt. Dadurch muss jeder Datensatz aktualisiert werden. Diese Gegebenheiten sprechen eindeutig für die Verwendung einer Datenbank. WEKA arbeitet grundsätzlich mit jeder Datenbank zusammen, für welche ein Java-Treiber existiert. In diesem Experiment wird MySQL verwendet. MySQL ist kostenlos erhältlich und unterstützt alle für diese Applikation erforderlichen Funktionen.

3.1.3 Reasoning

Für das Reasoning, also das Berechnen der Unterbrechbarkeit einer Person, werden zwei verschiedene Methoden verwendet. In der ersten Phase werden einfache statische Regeln verwendet, welche während der ganzen Phase gleich bleiben. Diese Art von Reasoning erfordert keine weitere Software. Für das Onlinereasoning ab Phase zwei kommt hingegen eine zusätzliche Software ins Spiel.

Es kommt die Data Mining Software WEKA, welche an der Universität von Waikato in Neuseeland entwickelt wurde, zum Einsatz. Bei WEKA handelt es sich um eine Sammlung von maschinenlernbaren Algorithmen für die Anwendung im Data Mining. Grundsätzlich wären auch andere Softwarepakete möglich, aber nur WEKA bietet den Zugriff auf die Algorithmen über eine Java API. Dies erleichtert die Anbindung an die bestehende Applikation enorm.

3.1.4 Telefonschnittstelle

Bei den verwendeten Telefonen handelt es sich um ein Produkt von Siemens. Die Geräte der optiPoint 500 Familie laufen an einer HiPath 3000 Telefonanlage des selben Herstellers. Die Telefone können über eine USB-Schnittstelle mit einem Computer verbunden werden. Siemens liefert dazu die passenden Treiber. Zusätzlich gibt es einen TAPI Treiber, welcher die Steuerung des Telefons aus Windows heraus erlaubt.

Für die Ansteuerung der Telefone aus einer Java-Applikation heraus ist JTAPI von Sun Microsystems vorgesehen. Dabei handelt sich aber nur um ein Interface, welches von Sun zur Verfügung gestellt wird. Von anderen Anbietern existieren aber zwei unterschiedliche Implementationen von JTAPI.

Die erste Implementation nennt sich XTAPI. Sie wir schon seit längerem nicht mehr weiterentwickelt und basiert auf der Version 2.0 der Windows-TAPI. Dadurch fällt die Verwendung mit der Siemens Software, welche die Windows-TAPI in der Version 3.0 voraussetzt, weg.

Eine neuere Variante ist Generic JTAPI (GJTAPI). Sie basiert zwar auch auf XTAPI, soll aber auch mit der Windows-TAPI 3.0 kompatibel sein. In diesem Experiment wird darum auf die Implementation von GJTAPI gesetzt.

3.1.5 Annotation

Die Annotation soll auf einem mobilen Gerät, wie z.B. einem Pocket PC oder Palm, durchgeführt werden. Damit der Client auf den verschiedenen Geräten lauffähig ist, soll hier auch Java eingesetzt werden und zwar die Java 2 Platform Micro Edition (J2ME). Dies garantiert die Funkti-

[2] Attribute-Relation File Format
[3] Character Separated Values

onsfähigkeit auf den meisten aktuellen Geräten und erleichtert auch die Kommunikation mit der Serverapplikation, welche ja auch in Java implementiert ist.

4

Umsetzung

In diesem Kapitel wird die Umsetzung der wichtigen Teile des Experiments beschrieben. Zuerst werden die beteiligten Module in einer groben Übersicht dargestellt. Im zweiten Unterkapitel wird dargestellt, was neu zur Applikation von Christian Singer [Singer, 2005] hinzugekommen ist oder was sich geändert hat. Und im letzten Teil wird die konkrete Umsetzung der einzelnen Module beschrieben.

4.1 Übersicht

Die Applikation basiert grob gesagt auf einer Client-Server Architektur. Es ist eine Serverapplikation vorhanden, an die mehrere Clients angebunden sind. Die Clients und der Server können auf dem selben Rechner ausgeführt werden. Die Kommunikation zwischen dem Server und Clients funktioniert mehrheitlich mit RMI. Über die Ausnahme der einen Socketverbindung wird später noch näher drauf eingegangen. Ein Überblick über die verschiedenen Module bietet die Abbildung 4.1.

4.1.1 Module

Die in Abbildung 4.1 zu sehenden Applikationsteile enthalten verschiedene Module, welche unterschiedliche Funktionen zur Verfügung stellen. In den folgenden Abschnitten werden diese aufgelistet und näher beschrieben.

Server Applikation

Die Server Applikation erledigt die grösste Arbeit. Neben dem ganzen Reasoning verarbeitet sie auch die aufgezeichneten Video- und Audiodaten. Dieser Teil enthält folgende Module:

- Reasoning: stellt die ganze Reasoner-Logik zur Verfügung.
- Datenspeicherung: stellt die Funktionen für das Speichern auf Festplatte und in die Datenbank zur Verfügung.
- RMI-Schnittstelle: bietet eine Schnittstelle nach aussen per RMI.

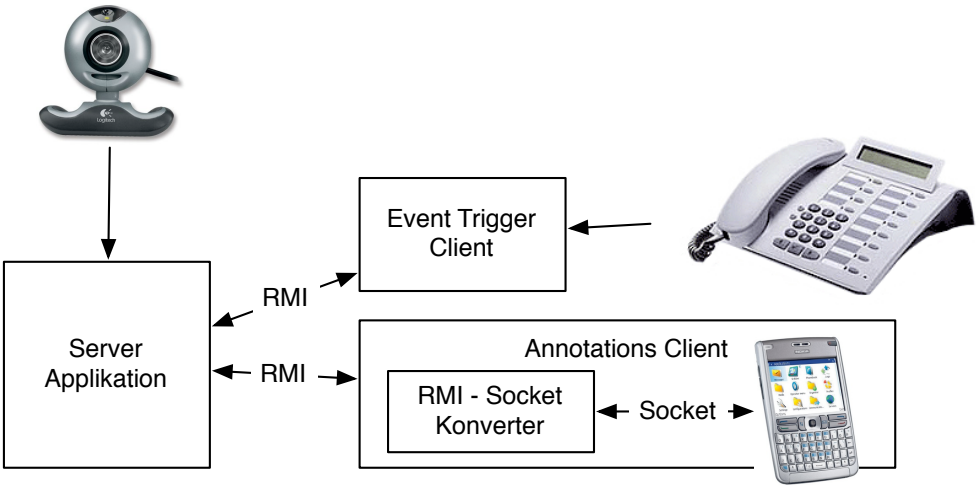


Abbildung 4.1: Versuchaufbau

Event Trigger Client

Der Event Trigger Client ist dazu da, eingehende Anrufe zu erkennen und dem Server zu melden. Gleichzeitig kann er den aktuellen Zustand der Unterbrechbarkeit anfordern und so entsprechend diesem Wert das Telefon ansteuern. Diese Funktionalität ist im folgenden Modul zu finden:

- Telefonsteuerung: erkennt eingehende Anrufe und kann das Verhalten des Telefons steuern.

Annotationsclient

Der Annotationsclient leitet die vom Benutzer erfassten Werte an den Server weiter. Folgende Module sind darin zu finden:

- Annotationsclient: Stellt einen mobilen Client für die Erfassung der Annotationsdaten zur Verfügung.
- Annotations-Bridge: Übersetzt zwischen der RMI basierten Kommunkation auf der Serverseite und der Socket basierten Kommunkation auf der Clientseite.

4.2 Abgrenzung

Da die in diesem Experiment verwendete Applikation auf dem Programm von Christian Singer [Singer, 2005] aufbaut, soll in diesem Abschnitt gezeigt werden, welche Teile schon vorhanden waren, welche geändert wurden und was komplett neu hinzugekommen ist.

4.2.1 Server Applikation

Die Applikation von Christian Singer [Singer, 2005] dient als Grundlage für unsere Serverapllikation. Ein grosser Teil der Funktionalität konnte übernommen werden. Aber für unser Experiment

mussten zuerst einige Teile angepasst oder erweitert werden.

Die grösste Erweiterung fand im Bereich des Reasonings an. In Abbildung 4.2 ist sichtbar, was sich geändert hat.

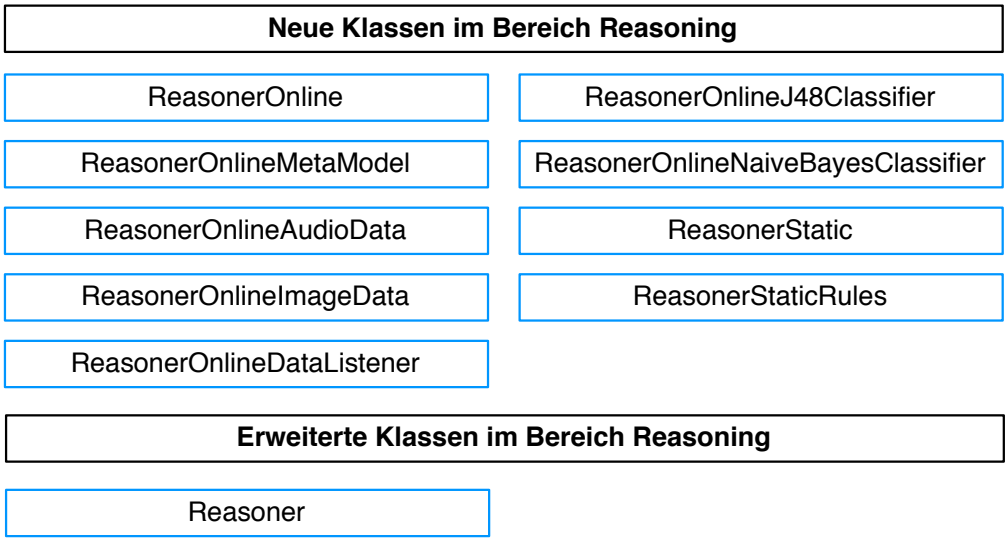


Abbildung 4.2: Abgrenzung Reasoning

Weitere Änderungen kamen im Bereich der Datenspeicherung hinzu. Insbesondere das zusätzliche Speichern in eine Datenbank. Diese Anpassungen sind in Abbildung 4.3 zu sehen.

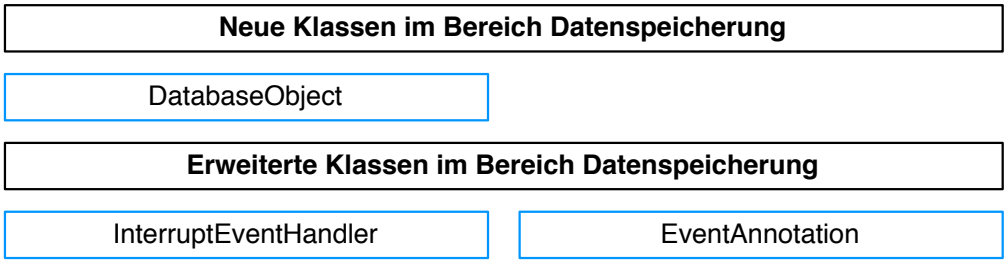


Abbildung 4.3: Abgrenzung Datenspeicherung

4.2.2 Clients

Die Clients wurden komplett neu erstellt. Die beteiligten Klassen sind in Abbildung 4.4 zu sehen.

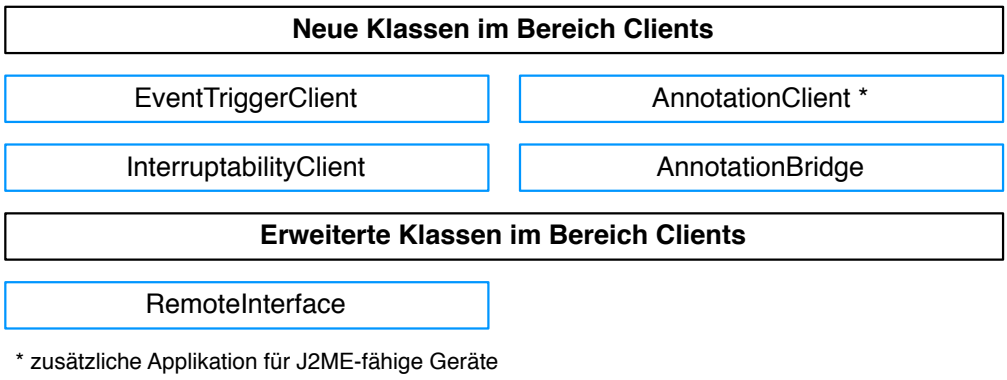


Abbildung 4.4: Abgrenzung Clients

4.3 Umsetzung der Module

In den nächsten Abschnitten werden die wichtigsten Module und Modulgruppen anhand ihrer konkreten Umsetzung beschrieben.

4.3.1 Realtime-Reasoning

Für das Reasoning werden die durch die Applikation aufgezeichneten Video- und Audiodaten verwendet. Um das Ganze besser verstehen zu können, wird zuerst gezeigt, in welcher Form die Daten vorhanden sind.

Videodaten

Während des Experimentes wird immer der gleiche Bereich eines Raumes aufgezeichnet. Das aufgenommene Bild besteht aus 16x16 Teilen. Jedes dieser 256 Teilstücke wird einem Bereich zugeteilt, welcher je eine Zone bildet. Die genaue Beschreibung, wie die Zonen konfiguriert werden, ist im Anhang A.1.2 zu finden. Im Bild 4.5 ist ein Beispiel einer solchen Zoneneinteilung zu sehen. Man erkennt darauf auch was die einzelnen Zonen repräsentieren sollen. Es gibt z.B. den Arbeitsplatz, den Eingangsbereich usw.

Für jede dieser Zonen wird jede Sekunde ein Wert ermittelt, der die Stärke der Bewegung in diesem Bereich repräsentiert. Beim eintreffen eines Ereignisses werden die Werte pro Zone über einen bestimmten im Voraus definierten Zeitraum addiert und zum Schluss davon ein Mittelwert bestimmt.

Audiodaten

Gleichzeitig mit der Videoaufzeichnung, wird auch das Umgebungsgeräusch des überwachten Raumes aufgenommen. Wie bei den Videodaten werden auch hier für jede Sekunde die Werte ermittelt. Aus dem Audiostream werden folgende zwei Werte bestimmt:

- Arithmetisches Mittel
- Varianz

Wie bei den Videodaten werden auch hier die Werte entlang der History-Länge addiert und dann der Mittelwert berechnet.

Annotationsdaten

Den letzten Teil eines Datensatzes machen noch die Annotationsdaten aus. Davon gibt es drei Werte, welche nachfolgend beschreiben werden:

- Unterbrechbarkeit: Beschreibt die Unterbrechbarkeit bei einem eingehenden Anruf auf einer Skala von 1 bis 5. (1 = ohne Probleme unterbrechbar; 5 = gar nicht unterbrechbar)
- Wichtigkeit des Anrufes: Beschreibt die Wichtigkeit eines durchgeführten Telefonates auf einer Skala von 1 bis 5. (1 = unwichtig; 5 = sehr wichtig)
- Anruf annehmen: Sagt aus, ob der Anruf hätte angenommen werden sollen oder nicht. (0 = ablehen; 1 = annehmen)

Das ergibt dann für jedes Ereignis einen Datensatz der ungefähr so aussieht:

```
image1,image2,image3,image4,image5,audio1,audio2,annot1,annot2,annot3
...
12345 ,66343 ,332   ,5332  ,34250 ,-0.324,0.0001,4     ,2     ,0
...
```

Wie mit diesen Werten weiterverfahren wird, hängt von der aktuellen Phase ab. In Phase eins wird das statische Reasoning verwendet und in den Phasen zwei und drei kommt das dynamische Reasoning zum Einsatz. Beide Implementationen werden in den folgenden Punkten genauer beschrieben.

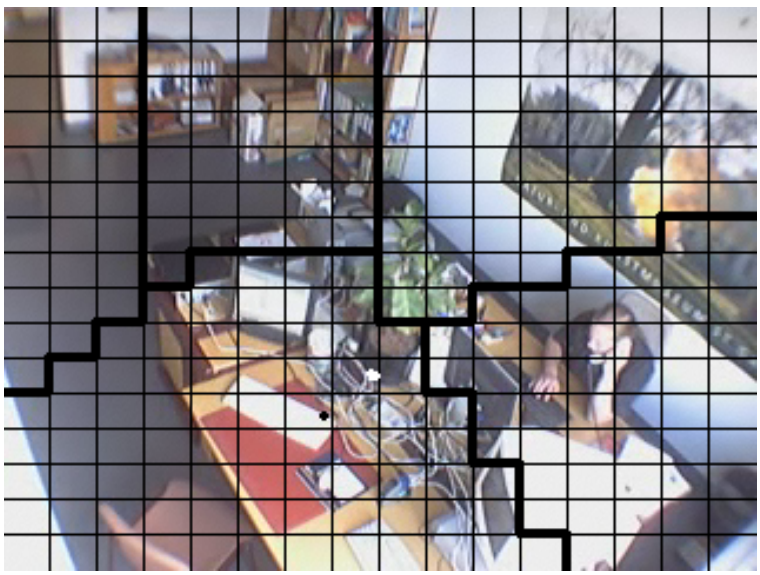


Abbildung 4.5: Zoneneinteilung

4.3.2 Statisches Reasoning

Für das statische Reasoning werden einfache Regeln erstellt, welche während der ganzen ersten Phase gleichbleiben. Für jede Zone wird ein Wert festgelegt, der die Unterbrechbarkeit festlegt. Diese Regeln werden in einer Konfigurationsdatei festgehalten, welche nach diesem Muster aufgebaut ist. Wobei der erste Wert die Zone angibt und der zweite Wert die Unterbrechbarkeit:

```
1:5
2:3
3:1
4:2
5:2
```

Diese Regeln werden einmalig beim Start der Software in ein Array geladen. Beim Eintreffen eines Anrufes wird jetzt überprüft, in welcher Zone die grösste Bewegung vorhanden war. Für diesen Bereich, wird dann der Wert aus dem Array genommen. So hat man auf einfache Weise die Unterbrechbarkeit bestimmt.

In unserem Beispiel hat man in der Zone zwei die grösste Bewegung. Und der Wert der Unterbrechbarkeit für diese Zone beträgt zwei.

Da diese Methode offensichtlich nicht zu überzeugen mag, da sie nur eine einzelne Zone betrachtet und die Audiodaten gar nicht berücksichtigt, gibt es noch das dynamische Reasoning. Da diese Variante aber erst einige gespeicherte Datensätze braucht, kommt sie erst ab Phase zwei zum Einsatz.

4.3.3 dynamisches Reasoning

Das dynamische Reasoning bedient sich der Methoden des Data Minings. Aus den bis anhin gesammelten Datensätzen soll ein Modell erstellt werden, mit dessen Hilfe neu eintreffende Ereignisse klassifiziert werden können. Die einfachste Möglichkeit wäre, aus allen Attributen der Daten ein einziges Modell zu erstellen. Doch in [Zurfluh, 2004] wurde gezeigt, dass für die Audio- und Videodaten nicht das gleiche Modell zum besten Resultat führt. Für die Videoattribute soll demnach der J48 Algorithmus, was einem Decision Tree entspricht, verwendet werden. Und für die Attribute der Audiodaten bringt der NaiveBayes Klassifizierer das beste Resultat. In unserem Experiment wird darum ein Metamodell erstellt, welches beide erwähnten Modelle integriert. Die Abbildung 4.6 zeigt den ganzen Ablauf beim Erstellen des Metamodells.

Instanzen laden

Als erstes werden die in der Datenbank gespeicherten Datensätze geladen. Um die entsprechenden Modelle zu erstellen, werden auch die Annotationsdaten benötigt. Darum werden nur diejenigen Datensätze verwendet, welche diese auch schon enthalten. Das ergibt dann zwei Sätze mit Instanzen. Der erste Satz enthält die Videodaten mit dem Annotationswert der Unterbrechbarkeit und der zweite Satz besteht aus den Audiodaten mit dem selben Wert der Annotation. Die Datensätze sehen dann ungefähr so aus:

```
image1,image2,image3,image4,image5,annot1
...
12345 ,66343 ,332   ,5332  ,34250 ,4
...
```

```
audio1,audio2,annot1
...
-0.324,0.0001,4
...
```

Preprocessing

An den momentan vorhandenen Datensätzen muss nicht mehr viel angepasst werden. Durch die richtige Wahl der Attribute in der Datenbank sind alle Werte im richtigen Format. Einzig muss noch das Zielattribut beider Instanzensets gewählt werden. Das Zielattribut gibt an, welches Attribut später vorausgesagt werden soll. Bei unseren Daten ist das beide Male der erste Annotationswert, also die Unterbrechbarkeit.

10-fold Crossvalidation

Im nächsten Schritt wird auf beiden Datensätzen eine 10-fold Crossvalidation durchgeführt. Dabei wird die ganze Datenmenge in 10 gleich grosse Mengen partitioniert, wobei neun Mengen als Trainings- und die zehnte Menge als Testmenge dienen. Dieses Vorgehen wird jetzt nun solange wiederholt, bis jede der zehn Partitionen einmal als Testmenge verwendet wurde. Die beste Hypothese wird dann als Endergebnis verwendet.

Prediction Appender

Im letzten Schritt vor der Erstellung des Modells werden bei den Audio- und Videodaten die Voraussagen hinzugefügt. Dabei handelt es sich um die Wahrscheinlichkeiten, mit denen eine Instanz zu einer bestimmten Klasse gehört. Zu jeder Instanz gibt es in unserem Fall fünf Werte, welche zusammen 100% ergeben müssen. Dabei haben wir jetzt zwei Datensätze, die folgendermassen aufgebaut sind:

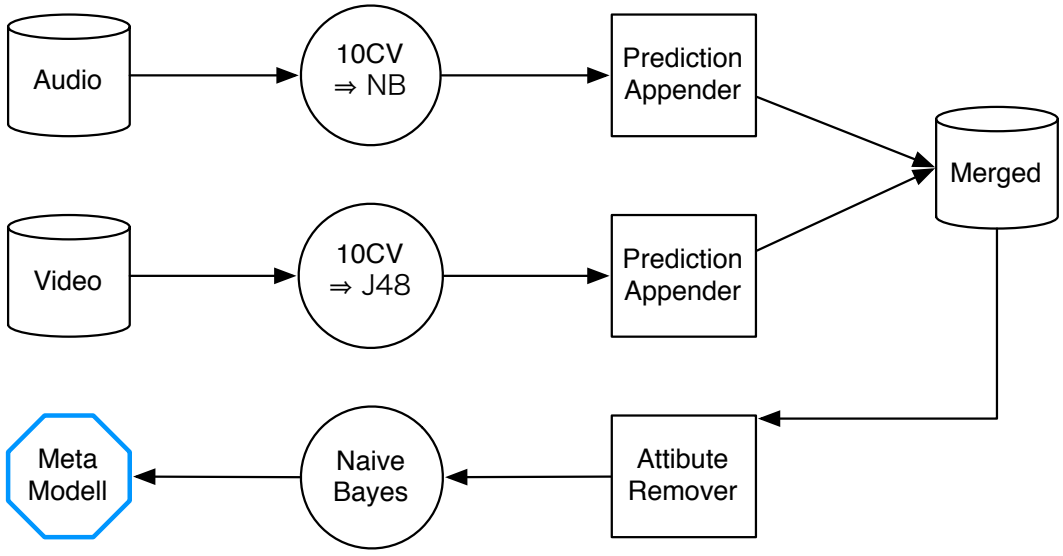


Abbildung 4.6: Ablauf Reasoning

```
image1,image2,image3,image4,image5,annot1,pred1,pred2,pred3,pred4,pred5
...
12345 ,66343 ,332   ,5332  ,34250 ,4     ,0    ,0.25 , 0.25 ,0.5 ,0
...

audio1,audio2,annot1,pred1,pred2,pred3,pred4,pred5
...
-0.324,0.0001,4     ,0.111.0.111,0    ,0.666,0.111
...
```

Meta Modell

Da wir für das Metamodell nur einen Datensatz verwenden können. müssen die beiden Teildatensätze von vorhin zu einem einzigen zusammengefügt werden. Gleichzeitig können die Ursprünglichen Attribute entfernet werden, so dass nur noch fünf Wahrscheinlichkeiten für die Videodaten und fünf Wahrscheinlichkeiten für die Audiodaten bestehen bleiben. Der Annotationswert muss selbstverständlich auch noch erhalten bleiben, weil es sich dabei wieder um das Zielattribut handelt. Das Ganze sieht dann so aus:

```
pred1,pred2,pred3,pred4,pred5,pred1,pred2,pred3,pred4,pred5,annot1
...
0    ,0.25 , 0.25 ,0.5 ,0    ,0.111.0.111,0    ,0.666,0.111,4
...
```

Aus diesem Datensatz wird dann ein Modell erstellt, auf welchem dann schlussendlich die neu zu klassifizierenden Ereignisdaten angewendet werden. Als Algorithmus wird für dieses Metamodell wieder NaiveBayes verwendet.

Audio- und Videomodell

Für das Klassifizieren von neuen Datensätzen müssen noch zwei zusätzliche Modelle erstellt werden. Der Grund liegt in den unterschiedlichen Attributen der Originaldaten und der beim Metamodell verwendeten Daten. Darum wird wieder je ein Modell aus den Audio- und den Videodaten erstellt. Diesmal aber ganz einfach ohne die 10-fold Crossvalidation. Abbildung 4.7 zeigt diesen Vorgang. Mit diesen drei vorhanden Modellen kann jetzt die Unterbrechbarkeit von neuen Ereignissen vorausgesagt werden.

Klassifizierung

Um einen neuen Wert zu klassifizieren geht man am Anfang ähnlich vor wie beim Erstellen der Modelle (siehe Abbildung 4.8). Es wird aber nur der aktuellste Datensatz aus der Datenbank geladen. Dieser wird dann wieder in Audio- und Videoteil getrennt. Der Annotationswert fehlt aber jetzt, weil der zu klassifizierende Datensatz ja noch gar nicht annotiert wurde. Im nächsten Schritt werden die beiden Instanzen durch ihr entsprechendes Modell klassifiziert. Die Audiodaten gehen durch das NaiveBayes-basierte Audiomodell und die Videodaten benutzen das auf J48 aufbauende Videomodell. Die berechnete Klassifizierung interessiert uns aber an dieser Stelle nicht. Wir verwenden wieder die fünf Wahrscheinlichkeiten beider Instanzen. Diese werden wieder zu einer einzigen Instanz zusammengesetzt, welche jetzt aus zehn Werten besteht. Diese Instanz wird zum Schluss durch unser Metamodell klassifiziert. Als Resultat erhalten wir einen Wert zwischen eins und fünf, welcher die Unterbrechbarkeit beim Eintreffen eines Ereignisses repräsentiert.

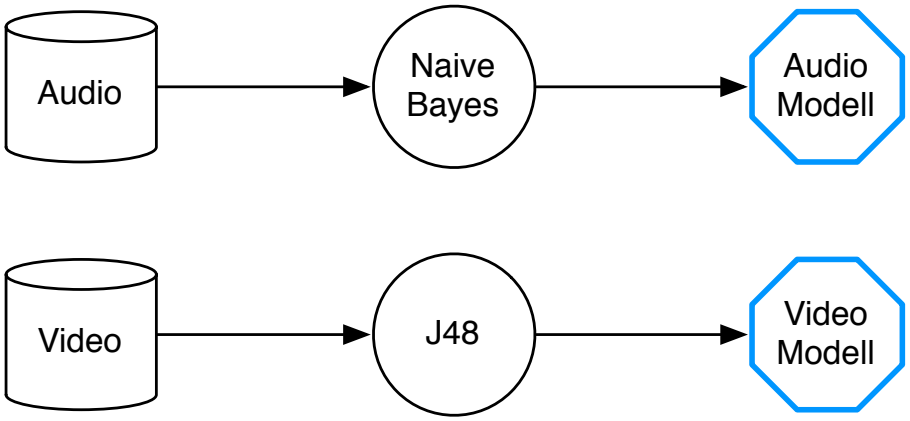


Abbildung 4.7: Audio- und Videomodell

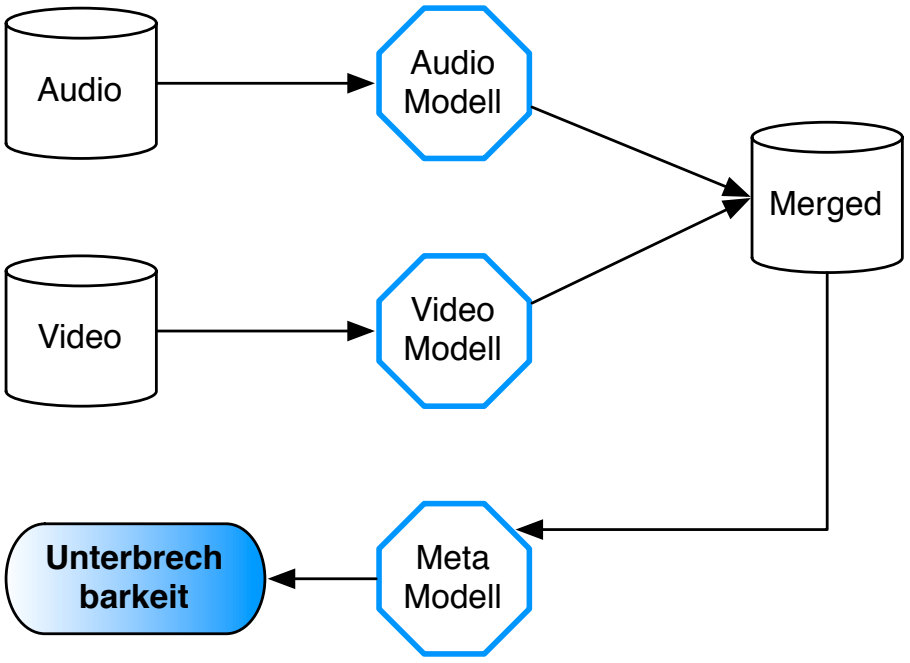


Abbildung 4.8: Klassifizierung

4.3.4 Datenbankspeicherung

Wie schon früher angedeutet, existiert neben der normalen Speicherung auf die Festplatte auch noch ein Modul, welches die Daten direkt in eine Datenbank speichert. Da für das Reasoning die Daten so schnell als möglich zur Verfügung stehen sollen, muss die Datenbankspeicherung vor der Festplattenspeicherung geschehen. In Originalzustand ist das Speichern auf die Harddisk nicht optimal gelöst. Denn die zeitliche Reihenfolge läuft folgendermassen ab:

1. Original- und Differenzbilder auf Festplatte schreiben
2. Berechnung der Bewegungsraster aus den Videodaten
3. Bewegungsraster auf Festplatte schreiben
4. komplette Audiodatei auf Festplatte schreiben
5. Merkmalsextraktion aus den Audiodaten
6. Audiomerkmale auf Festplatte schreiben

Da die Daten in diesem Fall erst nach dem fünften Schritt für die Datenbank zur Verfügung stehen, vergehen mehrere Minuten, bis der Datensatz in die Datenbank übertragen werden kann. Um diese Wartezeit, welche das Gebot der Echtzeitverarbeitung stark verletzt, zu umgehen, muss der ganze Ablauf geändert werden. Die für die Datenbank nötigen Berechnungen sollen ganz am Anfang durchgeführt werden und die grossen Datenmengen für die Festplatte sollen an den Schluss verschoben werden. Der neue Ablauf sieht dann folgendermassen aus:

1. Berechnung der Bewegungsraster aus den Videodaten
2. Videodaten in Datenbankobjekt speichern
3. Merkmalsextraktion aus den Audiodaten
4. Audiodaten in Datenbankobjekt speichern
5. Datenbankobjekt in Datenbank übertragen
6. Bewegungsraster auf Festplatte schreiben
7. Original- und Differenzbilder auf Festplatte schreiben
8. Audiomerkmale auf Festplatte schreiben
9. komplette Audiodatei auf Festplatte schreiben

Mit diesem neu gestalteten Ablauf können die Daten ohne merkliche Verzögerung in die Datenbank gespeichert werden. Um die einzelnen Daten in der Datenbank zu speichern, wird zuerst ein neues Datenbankobjekt erzeugt. Darin werden dann fortlaufend die Teildaten gespeichert (Schritt 2 und 4) und erst zum Schluss wird alles in die Datenbank geschrieben (Schritt 5). Die Annotationsdaten kommen erst in einem späteren Schritt hinzu. Durch einen vorhandenen Zeitstempel können diese Daten aber ohne Probleme dem richtigen Datensatz hinzugefügt werden.

Datenbankschema

Damit WEKA beim Laden der Datensätze auch automatisch den richtigen Attributtyp zuordnen kann, muss die Datenbank schon entsprechend eingerichtet sein. WEKA betrachtet z.B. float und int als numerische Datentypen und varchar als nominalen Datentyp. Diese Unterscheidung ist für das Erstellen von Modellen von grosser Bedeutung. Denn gewisse Algorithmen lassen nur nominale Attribute als Zielattribut zu.

Je nach Attribut muss also ein entsprechender Datentyp gewählt werden:

- ID: int
- Timestamp: varchar
- Audiodaten: float
- Videodaten: int
- Annotationsdaten: varchar

Es ist auch zu beachten, dass genügend Felder für die Videodaten vorhanden sind. Pro konfigurierte Zone muss ein Feld vorhanden sein. Sind zu wenig Felder vorhanden, ist die Applikation nicht lauffähig. Zu viele Felder hingegen sind kein Problem und werden einfach ignoriert. Ein genaues Beispiel eines Tabellenschemas für die Datenbank ist im Anhang zu finden.

4.3.5 Annotationsclient

Der Annotationsclient besteht aus zwei Komponenten. Einerseits aus dem Client selbst und andererseits aus einem Modul, welches die Kommunikation zwischen Client und Server regelt. Dies ist darum nötig, weil die Verbindung zum Server auf die Verwendung von RMI ausgelegt ist. Der Client hingegen, welcher in J2ME implementiert ist, beherrscht nur die Kommunikation über Sockets. Diese Einschränkung auf Sockets ist eine Limitation von J2ME und kann nicht umgangen werden. In den nächsten zwei Punkten werden die beiden Module anhand ihrer Eigenschaften näher betrachtet.

Client

Wie schon beschrieben ist der Annotationsclient in J2ME implementiert. Dadurch kann er auf allen möglichen mobilen Geräten eingesetzt werden. In diesem Experiment wurde er auf einem Nokia E61 ausprobiert. Aber auch Windows Mobile und Palm basierte Geräte sind für den Einsatz geeignet.

Der Client stellt vier verschiedene Bildschirme zur Verfügung. Der erste Bildschirm, zu sehen in Abbildung 4.9 links, wird angezeigt, während auf ein eintreffendes Ereignis gewartet wird. Sobald ein Telefonanruf registriert wird, wird auf die Annotationsbildschirme weitergeschaltet. Im ersten Annotationsbildschirm, zu sehen in Abbildung 4.9 rechts, muss der Benutzer seine Unterbrechbarkeit annotieren. In Abbildung 4.10 sind die beiden weiteren Annotationsbildschirme für die Wichtigkeit und die Annahme des Anrufes zu sehen. Nach dem Abschluss der Annotation wird automatisch wieder auf den Wartebildschirm umgeschaltet.

Abbildung 4.9: Wartebildschirm und Annotationsbildschirm 1

Abbildung 4.10: Annotationsbildschirme 2 und 3

Socket - RMI Übersetzer

Damit der mobile Client überhaupt davon erfährt, dass ein neues Ereignis eingetroffen ist, braucht es eine weitere Komponente. Das AnnotationBridge genannte Modul arbeitet als Übersetzer für die vom Server verwendete RMI-Kommunkation und der Socket-Kommunkation vom Client. In Abbildung 4.11 ist das Zusammenspiel von Server, Bridge und Client zu sehen.

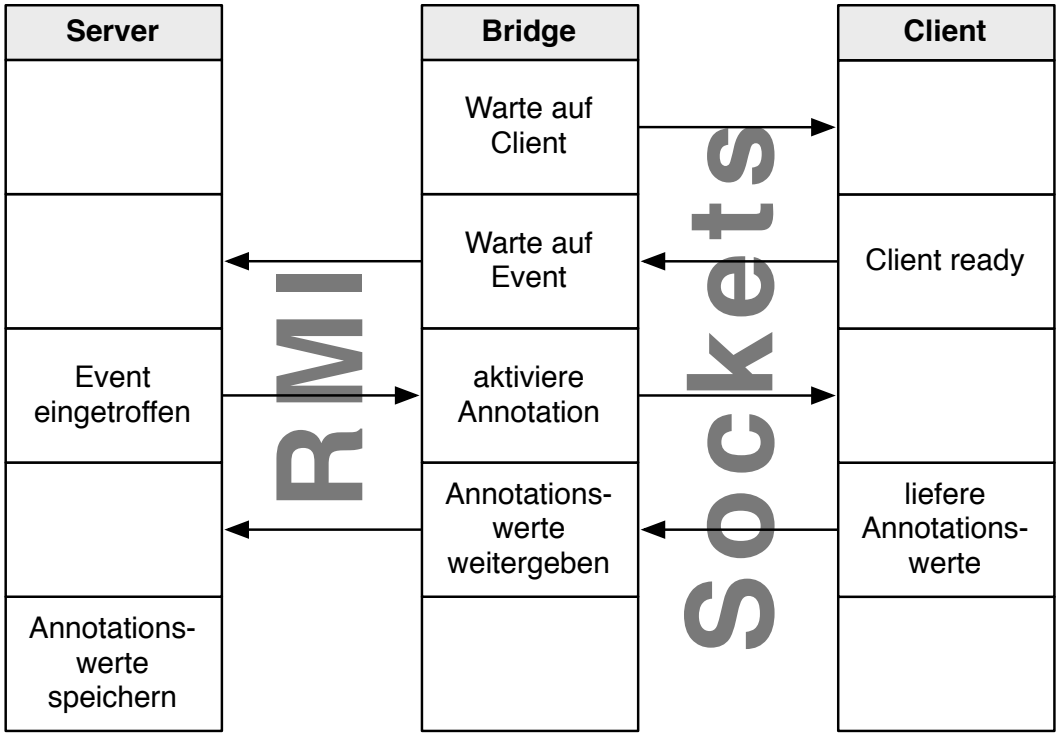


Abbildung 4.11: Socket - RMI Übersetzer

4.4 Event Trigger Client

Der Event Trigger Client hat die Funktion, Ereignisse zu erkennen und dann die entsprechende Handlung anzustossen. Weiter kann er, mit dem vom Server erhaltenen Resultat, das Verhalten des Telefons beeinflussen.

4.4.1 Ereignisablauf

Beim Eintreffen eines Anrufes laufen folgende Schritte ab:

1. Event auslösen: Mit der RMI-Methode triggerEvent() wird die Datenspeicherung auf der Serverseite ausgelöst.
2. Unterbrechbarkeit anfordern: Mit calculateInterruptabilityLevel() kann dann der Grad der Unterbrechbarkeit angefordert werden.

3. Modell aktualisieren: Als letzter Schritt kommt das Aktualisieren der Modelle für das nächste Reasoning. Dies geschieht mit dem Befehl updateReasoner().

Mit dem aus Schritt zwei erhaltenen Wert kann man jetzt das Verhalten des Telefons steuern. Je nach Grad der Unterbrechbarkeit kann das Telefon normal beim Benutzer läuten, man leitet es weiter an eine andere Nummer oder auf einen Anrufbeantworter oder man lehnt den Anruf einfach ab.

4.4.2 Telefonsteuerung

Für die Ansteuerung des Telefons wird, wie Eingangs erwähnt, GJTAPI verwendet. GJTAPI unterstützt, im Gegensatz zu XTAPI, den TAPI Treiber von Siemens. Mit dieser Software funktionieren ankommende und abgehende Anrufe ohne Probleme. Aber einige wichtige Funktionen sind momentan nicht in GJTAPI implementiert. Dazu gehören z.B. die Funktionalitäten des Weiterleitens oder Ablehnens. Laut dem Entwickler wurde dies wegen der Komplexität und der fehlenden Nachfrage nicht implementiert[1]. Durch diese Tatsache lässt sich die Steuerung des Telefons nicht wie erwünscht bewerkstelligen. Auf mögliche Lösungen wird in Kapitel 7 eingegangen.

[1] http://sourceforge.net/forum/message.php?msg_id=3683676

5

Evaluation

In diesem Kapitel wird die ganze Applikation bezüglich ihrer Funktionstauglichkeit evaluiert. Dies wird einerseits über den Speicherverbrauch und andererseits über die Ausführungsgeschwindigkeit beurteilt.

Die ganzen Versuche für die Evaluation wurden auf einem Rechner mit folgenden Daten durchgeführt:

- Prozessor: Intel Pentium M 1200 MHz
- Hauptspeicher: 768 MB
- Festplatte: 1.8”4200 U/min

5.1 Speicherverbrauch

Der Speicherverbrauch betrifft zwei Arten von Speicher. Als erstes wird der Bedarf an Festplattenspeicher betrachtet und im zweiten Schritt der Verbrauch an Hauptspeicher.

5.1.1 Festplatte

Bei jedem Ereignis das eintrifft, werden die Video- und Audioaufzeichnungen entsprechend der Länge der History aufgezeichnet. Für jeden Datensatz werden folgende Objekte auf die Festplatte gespeichert:

- Originalbild (pro Sekunde)
- Differenzbild (pro Sekunde)
- kompletter Audiostream
- Audio Features
- Bewegungsmatrizen

Da die einzelnen Bilder mit einer Auflösung von 320x240 Bildpunkten gespeichert werden, ist auch schon bei einem kurzen Wert für die History mit einer grossen Menge an Speicherplatz zu rechnen.

Es wurden Versuche mit verschiedenen Längen durchgeführt. Es wurde bei zwei Minuten gestartet und sollte immer in Schritten von zwei Minuten erhöht werden. Da die Daten mindestens 15 Minuten in die Vergangenheit reichen sollen, sollte dort auch etwa der Schluss sein. Im Diagramm 5.1 kann der benötigte Festspeicher pro Ereignis (in MB), abhängig von der Länge der History, abgelesen werden. Für jede Länge der History wurden fünf Datensätze erstellt und der Mittelwert bestimmt.

Die Versuche konnten aber nicht bis zu den erwünschten 15 Minuten durchgeführt werden. Durch den enormen Hauptspeicherverbrauch konnten nur die Werte bis acht Minuten berechnet werden. Wenn man annimmt, dass der Festplattenverbrauch linear weiterwächst, muss man bei unserem Zielwert von 15 Minuten mit 88 MB pro Datensatz rechnen. Um den ganzen Verbrauch während eines längeren Versuches zu ermitteln, rechnen wir mit 10 Anrufen pro Arbeitstag. Dadurch würden über ein Jahr gesehen, knapp 23 GB Festplattenplatz gebraucht. Bei aktuellen Festplattengrössen stellt das also kein nennenswertes Problem dar

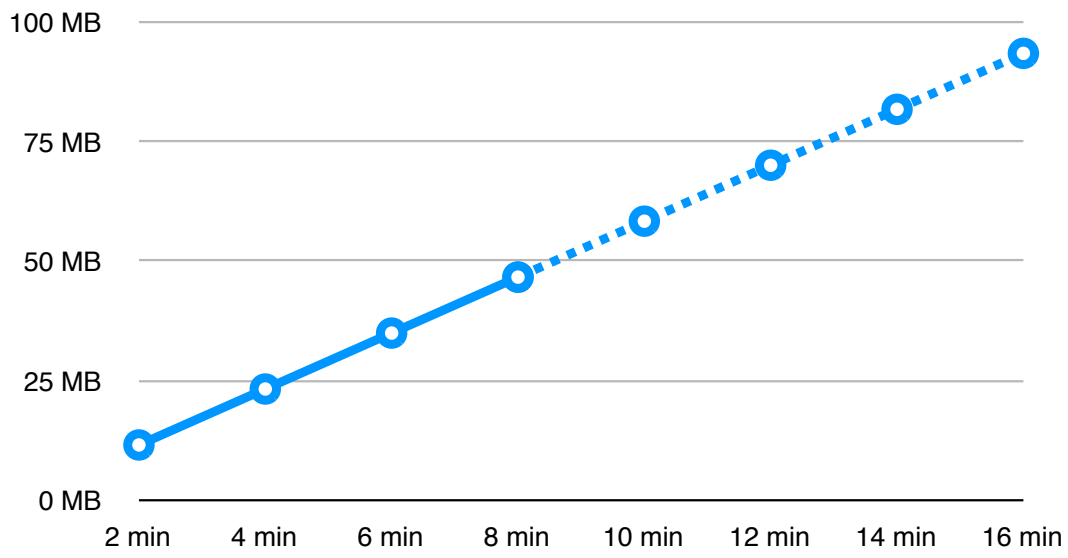


Abbildung 5.1: Festplattenverbrauch (in MB)

5.1.2 Hauptspeicher

Wie in Kapitel 5.1.1 schon angedeutet, spielt der Hauptspeicher (RAM) für diese Applikation eine wichtige Rolle. Dies ist darum der Fall, weil fortlaufend die letzten Minuten der Aufzeichnung in einem Ringbuffer behalten werden. Je nach Länge der gewählten Aufzeichnungsdauer werden bis zu einem Gigabyte und mehr Hauptspeicher dafür verbraucht. Standardmässig stehen der VM von Java nur 128 MB zur Verfügung. Dieser Wert reicht aber gerade mal für eine Dauer von 60 Sekunden. Da die Aufzeichnung aber mindestens 15 Minuten in die Vergangenheit zurückreichen sollte, muss der VM mehr Speicher zugeteilt werden.

Auch wenn die Applikation noch startet, ist sie noch lange nicht funktionsfähig. Da beim Eintreffen eines Ereignisses der ganze Buffer im RAM kopiert wird, erhöht sich der Verbrauch drastisch. Zusätzlichen Speicher benötigt auch das Reasoning und weitere Objekte, welche während er Laufzeit erstellt werden. Diese fallen aber nicht allzu stark ins Gewicht.

Wie oben schon erwähnt wurde, reicht der standardmässige Speicher von 128 MB gerade mal für eine History von einer Minute. Mit 1024 MB Heap Speicher[1] erreicht man eine Länge von acht Minuten. Wenn man jetzt annimmt, dass der Speicher linear steigt, müsste man für eine History von 15 Minuten fast zwei GB Speicher verwenden. Im Diagramm 5.2 kann der benötigte Hauptspeicher, abhängig von der Länge der History, abgelesen werden.

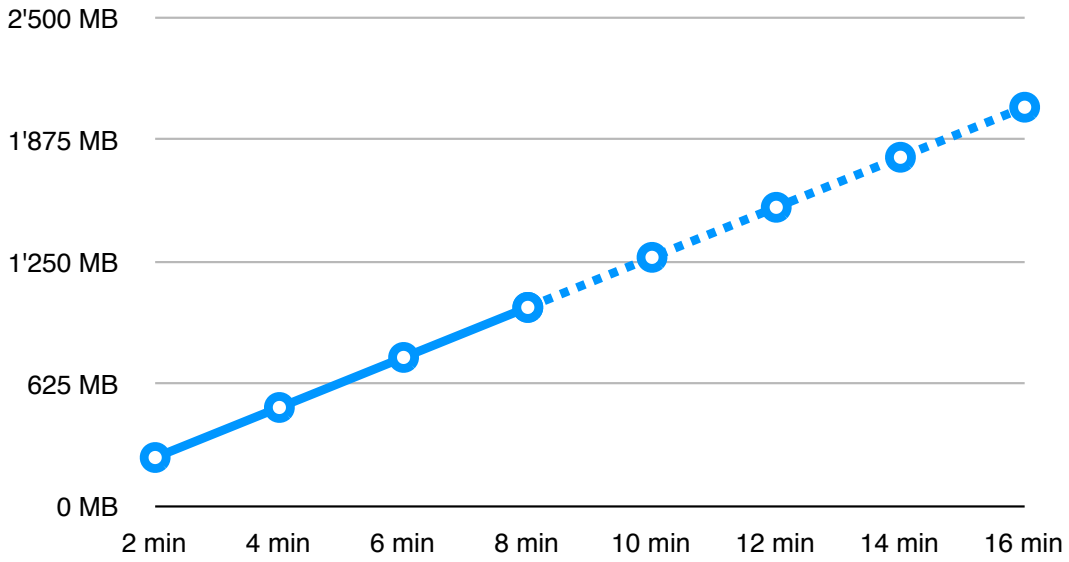


Abbildung 5.2: Hauptspeicherverbrauch (in MB)

5.1.3 Analyse

Wie man in den zwei Abschnitten zum Speicherverbrauch sehen kann, ist die Applikation enorm speicherhungrig. Während der Festplattenverbrauch grundsätzlich mit grösseren Festplatten zu lösen ist, gibt es beim Hauptspeicher gewisse Probleme. Man kann zwar auch mehr RAM einbauen und der VM zuweisen, aber die Applikation wird umso träger, je mehr Hauptspeicher sie verbraucht. Bei unserem Testsystem war bei einer History von acht Minuten das System an seinem Anschlag. Das ganze System war dadurch mehr oder weniger unbedienbar. Durch die Verwendung von schnelleren Rechnern mit mehr Speicher könnte das Problem aber unter Umständen entschärft werden.

[1] java -Xmx1024m ArtificialReceptionist.java

5.2 Zeit

Die zweite wichtige Ressource ist die Zeit. Zu den Bereichen, in denen die Zeit eine Rolle spielt, gehören das Speichern der Daten, das Erstellen der Modelle und das Klassifizieren neuer Daten.

5.2.1 Speichervorgang

Für den Speichervorgang betrachten wir einerseits die Zeit, bis die Daten in der Datenbank gespeichert sind (Abbildung 5.3) und andererseits die Dauer bis sie komplett auf der Festplatte sind (Abbildung 5.4). Beide Werte konnten auch nur bis zu einer History von acht Minuten ermittelt werden. Bei den Zahlen darüber handelt es sich um geschätzte Werte.

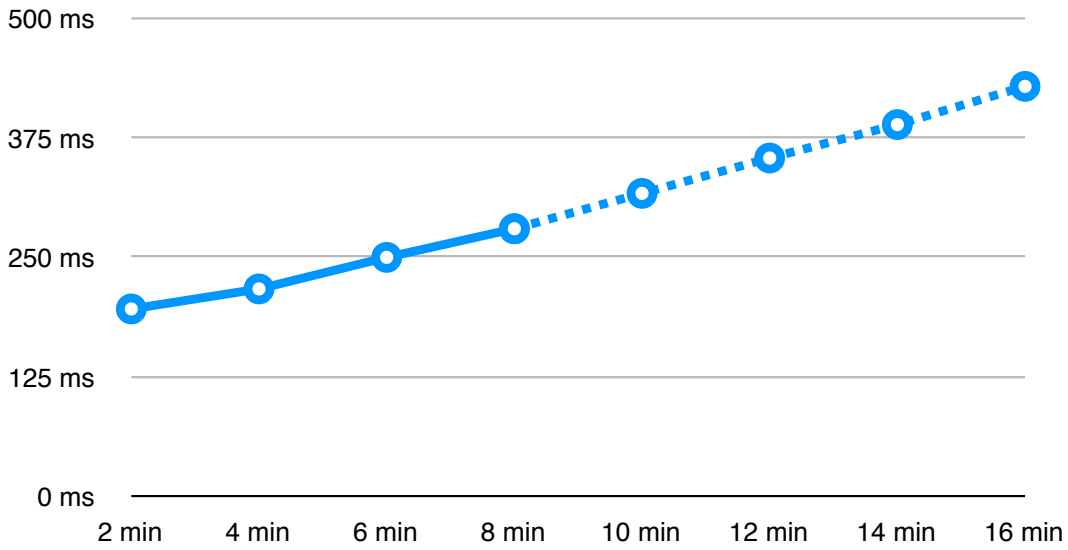


Abbildung 5.3: Dauer der Datenbankspeicherung (in ms)

Die Zeitdauer ist beim Speichern in die Datenbank sehr gering. Auch bei unseren gewünschten 15 Minuten vergeht deutlich weniger als eine Sekunde. Das Speichern auf die Festplatte hingegen ist sehr zeitintensiv. Da das Wachstum hier nicht linear ist, ist die Zeitdauer schwierig abzuschätzen. Aber es muss mit mehreren Minuten gerechnet werden. Da dieser Vorgang aber im Hintergrund abläuft, wird der Programmablauf dadurch nicht gestört.

5.2.2 Modellerstellung

Die Zeit für die Modellerstellung beinhaltet alle Schritte des Vorgangs. Es beginnt beim Laden der Daten aus der Datenbank und endet mit der Erstellung des Metamodells. Die Messungen wurden mit Schrittweiten von 50 Datensätzen durchgeführt. Jede Messung wurde fünf mal durchgeführt und davon wurde der Mittelwert bestimmt. Das Resultat ist in Abbildung 5.5 zu sehen.

Die Zeitdauer für das Erstellen der Modelle liegt im Bereich von einer Sekunde. Es ist zu beobachten, dass die Kurve sehr flach ansteigt. Auch bei noch mehr Datensätzen wird die Dauer nicht mehr allzu stark zunehmen.

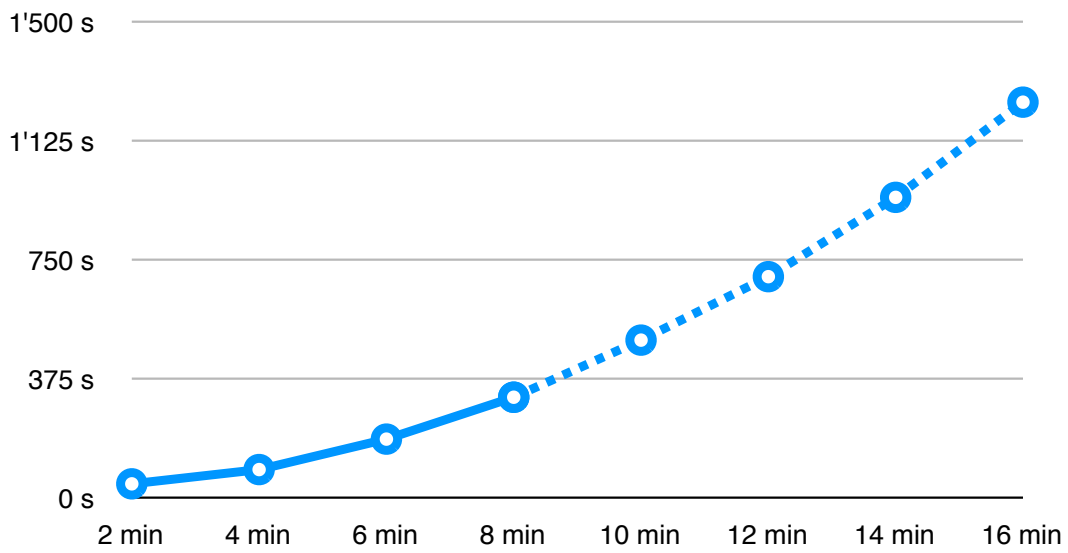


Abbildung 5.4: Dauer der Festplattenspeicherung (in s)

Da die Modellerstellung erst nach dem Eintreffen und Klassifizieren eines Ereignisses geschieht, ist diese Zeitdauer aber auch nicht wirklich ein grosses Problem.

5.2.3 Klassifizieren

Das Klassifizieren eines neuen Ereignisses ist ein wichtiger Vorgang. Das Speichern in die Datenbank und das Klassifizieren bilden zusammen diejenige Zeit, welche für die Verzögerung der Telefonsteuerung verantwortlich ist. Ein möglichst tiefer Wert ist hier also erwünscht.

Und wie wir in Abbildung 5.6 sehen können, ist diese Zeit extrem klein. Die Dauer des Klassifizierens ändert sich auch nicht mit der Grösse des Modells bzw. mit der Anzahl verwendeten Datensätzen.

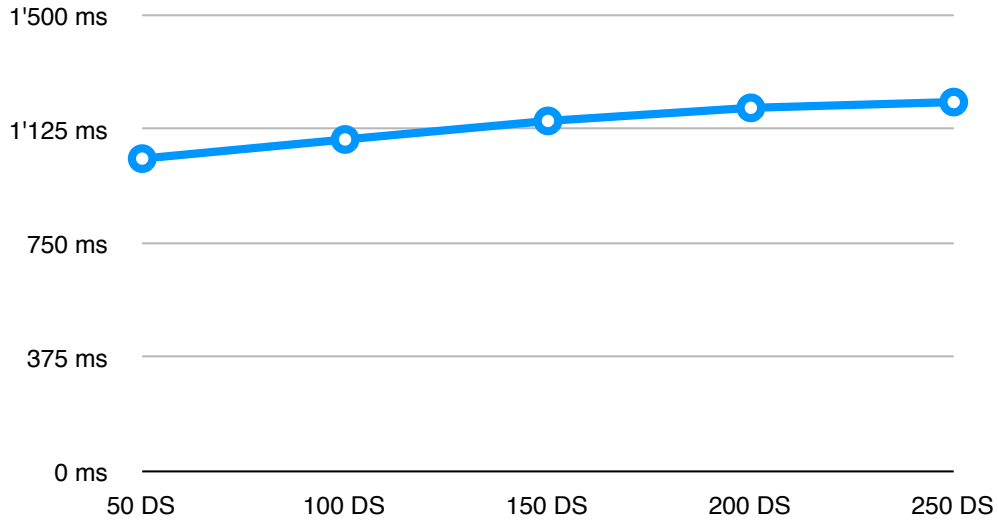


Abbildung 5.5: Dauer der Modellerstellung (in ms)

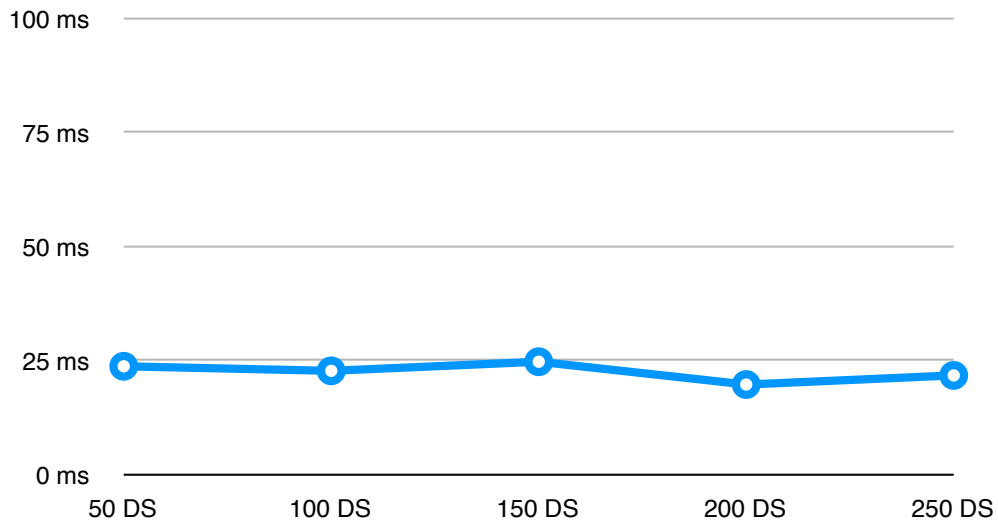


Abbildung 5.6: Dauer des Klassifizierens (in ms)

6

Diskussion und Schlussfolgerung

In diesem Kapitel soll betrachtet werden, inwiefern das Experiment erfolgreich war. Auch werden Lösungsansätze besprochen, falls etwas nicht optimal war.

6.1 Echtzeit

In Kapitel 5.2 wurde gezeigt, dass die Komponente Zeit kein Problem für das Experiment darstellt. Weder das Erstellen von Modellen für das Reasoning noch das Klassifizieren von Ereignissen dauern länger als eine Sekunde. Auch bei vielen Datensätzen erhöht sich die Verarbeitungsdauer nicht merklich. Einzig das Speichern der Daten ist zeitintensiv. Aber durch eine geschickte Anordnung der einzelnen Schritte, sind die wichtigen Daten für die Datenbank auch in Sekundenbruchteilen verfügbar. Einzig das Speichern der Bilder und Audiostreams kann mehrere Minuten dauern. Aber dieser Vorgang läuft im Hintergrund weiter, während die Applikation normal weiterläuft.

Fazit: Anforderungen erfüllt.

6.2 Umfang

Der Umfang betrifft die Länge der History. In Kapitel 5.1 wurde schon über die maximale Länge diskutiert. Ein optimaler Wert für die History wäre 15 Minuten. Aber auf dem Testrechner konnte dieser Wert nicht erreicht werden. Es waren maximal acht Minuten möglich um noch einen normalen Rechnerbetrieb zu gewährleisten. Die limitierende Komponente ist in diesem Fall ist der Hauptspeicher. Mit einem Rechner der 2 GB Hauptspeicher der Virtual Machine zur Verfügung stellen kann, sollte eine History von 15 Minuten aber möglich sein.

Fazit: Anforderungen knapp erfüllt.

6.3 Telefonsteuerung

Das Sorgenkind des ganzen Experimentes ist die Ansteuerung des Telefons. Die einzige Möglichkeit für eine Ansteuerung der vorhanden Siemens Telefone bietet GJTAPI. Doch diese JTAPI-Implementation bietet nur rudimentäre Methoden wie Anrufe annehmen und Nummern wählen. Die für dieses Experiment wichtigen Funktionen wie z.B. weiterleiten sind nicht vorhanden. Dadurch ist es nicht möglich, das Telefon, abhängig von der ermittelten Unterbrechbarkeit, zu steuern. Mögliche Lösungen für dieses Problem werden im Kapitel 7 vorgestellt. Anhand von Beobachtungen ist anzunehmen, dass auch bei einer vorhanden Weiterleitungsmöglichkeit das Problem nicht vollständig gelöst werden kann. Denn bei einem eingehenden Anruf klingelt das Telefon schon, bevor das Event durch den TAPI-Treiber erkennt wird. Das Telefon würde also in jedem Fall läuten, was eigentlich vermieden werden sollte. Es ist anzunehmen, dass die Telefonanlage den ankommenden Anruf einfach an alle angemeldeten Geräte meldet. Das Telefon selbst und der TAPI-Treiber werden so als zwei verschiedene Geräte betrachtet.

Fazit: Anforderungen nicht erfüllt.

7

Ausblick

Auch wenn dieses Experiment durch diese Arbeit abgeschlossen ist, sind Erweiterungen oder Anpassungen im Rahmen anderer Arbeiten möglich. Einige Möglichkeiten werden in den folgenden Unterkapiteln genauer beschreiben.

7.1 Experiment durchführen

Ein wichtiger Schritt ist die Durchführung des Experimentes in einer richtigen Umgebung. In diesem Experiment wurde zwar gezeigt, dass die Software grundsätzlich funktioniert. Aber ob das Reasoning auch wirklich verwertbare Resultate erzeugt, kann erst in einem Versuch unter realen Bedingungen ermittelt werden. Um ein besseres Ergebnis zu erreichen, sollte das Experiment mit mehreren Testpersonen durchgeführt werden.

Aber ohne eine funktionierende Telefonsteuerung macht das Experiment natürlich noch nicht viel Sinn. Darum sollte eine andere Lösung für dieses Problem gesucht werden.

7.2 Alternative für Telefon

Da das Steuern des Telefons mit dem TAPI-Treiber nicht zufriedenstellend funktioniert, muss eine Alternative gefunden werden. Eine Möglichkeit wäre die Verwendung eines Clients, welcher auf dem Session Initiation Protocol (SIP) oder Asterisk aufbaut.

7.3 Panel

Die Annahme, dass die Störung im Büro nur durch das Telefon kommt, ist natürlich ziemlich vereinfacht. Viele Leute werden wohl einfach mal vorbeischauen, ob die Person am Arbeitsplatz ist. Darum sollte es auch eine Möglichkeit geben, die Besucher schon vor der Tür oder noch besser am Gebäudeeingang über die Verfügbarkeit der betreffenden Person zu informieren.

Dazu kann man z.B. am Eingang ein Display platzieren, welches in regelmässigen Abständen die Verfügbarkeit der betreffenden Personen ermittelt und anzeigt. Die Unterbrechbarkeit wird dann nicht nur bei einem Anruf ermittelt, sondern, unabhängig von Ereignissen, alle paar Minuten.

7.4 User Interface

Dass das Aussehen einer Anwendung einen grossen Einfluss auf die Bedienbarkeit hat, ist unbestritten. Der in diesem Experiment verwendete Annotationsclient bietet nur einfach gestaltete Auswahllisten mit mehreren numerischen Elementen, dessen Bedeutung unter Umständen nicht immer klar ersichtlich ist. Eine Verbesserung kann z.B. durch den Einsatz von Symbolen erreicht werden. Auch beim vorher angesprochenen Panel ist das Design ein wichtiger Punkt für die Akzeptanz beim Besucher.

7.5 weitere Sensoren

Neben der verwendeten Kamera, welche Video- und Audioaufzeichnungen macht, ist auch der Einsatz anderer Sensoren denkbar. Da im Institut für Informatik überall Bewegungsmelder vorhanden sind, drängt sich diese Verwendung geradezu auf. Natürlich muss hierbei zuerst abgeklärt werden, inwiefern diese Sensoren überhaupt von einem Rechner ansprechbar sind. Da die verwendeten Bewegungsmelder über vier getrennte Bereiche verfügen, könnte man wie in Abbildung 4.5 verschiedene Zonen definieren. Man kriegt daraus selbstverständlich nicht ganz so gute Bewegungsdaten wie von der Kamera, aber um rauszufinden, ob überhaupt jemand da ist, sind sie sehr gut geeignet.

A

Konfigurationsdateien

A.1 Server

In diesem Abschnitt werden die einzelnen Konfigurationsdateien erklärt. Alle Konfigurationsdateien der Server-Applikation liegen im Unterverzeichnis "_config".

A.1.1 Parameter der Hauptapplikation

Folgende Parameter können unter ArtificialReceptionist.config eingestellt werden:

PARAMETER	BESCHREIBUNG
server_ip	IP-Adresse des Rechners auf dem die Applikation läuft
history_length	Aufzeichnugsdauer der Daten auf die Festplatte
image_length	History für Reasoning der Videodaten (kleiner als history_length)
audio_length	History für Reasoning der Audiodaten (kleiner als history_length)
zones	Anzahl Zonen der Bildaufzeichnung
data_dir	Verzeichnis der zu speichernden Daten
db_host	Netzwerkadresse des Datenbankservers
db_name	Datenbankname
db_user	Benutzername der Datenbank
db_password	Passwort der Datenbank
phase	Phase der Software (1-3)

A.1.2 Raster für Zoneneinteilung

Die einzelnen Zonen werden in der Datei CustomRaster.config definiert. Das Bild wird in 64 kleine Bereiche geteilt, wobei jeder Dieser Flächen zu einer Zone zugeordnet wird. Für das Beispiel aus Kapitel 4.3.1 sieht die Datei folgendermassen aus:

```
row1= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3
row2= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3
row3= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3
row4= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3
row5= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3
```

```
row6= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3
row7= 1,1,1,2,2,2,2,2,3,3,3,3,3,3,5,5
row8= 1,1,1,2,4,4,4,4,3,3,3,3,5,5,5,5
row9= 1,1,1,4,4,4,4,4,3,3,5,5,5,5,5,5
row10=1,1,4,4,4,4,4,4,4,5,5,5,5,5,5,5
row11=1,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5
row12=4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5
row13=4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5
row14=4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5
row15=4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5
row16=4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5
```

A.1.3 statische Reason-Regeln

Für das statische Reasoning wird jeder Zone ein Wert der Unterbrechbarkeit zugeordnet. Die Regeln sind zu finden in der Datei CustomRules.config. Für das Beispiel aus Kapitel 4.3.1 kann die Datei folgendermassen aussehen:

```
1=5
2=3
3=1
4=2
5=2
```

A.2 Client

Die Konfigurationsdatei befindet sich im JAR-File des Annotationsclients. Um Änderungen durchführen zu können, muss das JAR-File zuerst entpackt und anschliessend wieder erstellt werden.

A.2.1 Parameter des Annotationsclient

Folgende Parameter können in der Datei server.cfg eingestellt werden:

PARAMETER	BESCHREIBUNG
host	IP-Adresse des Rechners auf dem die Applikation läuft.

A.3 Datenbank

Damit die Applikation ohne Fehler läuft. muss auch die Datenbank richtig konfiguriert werden.

A.3.1 Datenbankschema

Für unser Beispiel aus Kapitel 4.3.1 sieht das Datenbankschema folgendermassen aus:

```
CREATE TABLE  `weka` (
 `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
 `eventtime` VARCHAR( 20 ) ,
```

```
  `image1` INT,
  `image2` INT,
  `image3` INT,
  `image4` INT,
  `image5` INT,
  `audio1` FLOAT,
  `audio2` FLOAT,
  `annotation1` VARCHAR( 1 )
  `annotation2` VARCHAR( 1 )
  `annotation3` VARCHAR( 1 )
);
```

B

Verwendete Software

Folgende zusätzliche Software wurde für dieses Experiment verwendet[1]:

WEKA

Data Mining Software der Universität von Waikato, Neuseeland.
http://www.cs.waikato.ac.nz/ ml/weka/

JTAPI

Java Telephony API von Sun Microsystems.
http://java.sun.com/products/jtapi/

GJTAPI

Generic Java Telephony API. Eine Implementation von JTAPI.
http://gjtapi.sourceforge.net

J2ME

Java 2 Platform, Micro Edition.
http://java.sun.com/javame/index.jsp

MySQL

Datenbankverwaltungssoftware der Firma MySQL AB.
http://www.mysql.com

Siemens TAPI Treiber

TAPI Treiber für die optiPoint 500 Telefone von Siemens.
http://www.hipath.de

[1] Die Internetadressen waren bei der Abgabe dieser Arbeit gültig

Literaturverzeichnis

[Singer, 2005] Singer, C. (2005). *Pilot-Versuch zu Kosten von Unterbrechungen am Arbeitsplatz*. Institut für Informatik, Universität Zürich.

[Zurfluh, 2004] Zurfluh, A. (2004). *The Artificial Secretary - Voraussage des Telefonieverhaltens in einem Bureau anhand von Sensordaten*. Institut für Informatik, Universität Zürich.