Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Andreas Krohn

Entwurf und Realisierung eines Systems zur Prognose und Optimierung der Wartungskosten von Geldautomaten auf Basis maschineller Lernverfahren

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Krohn

Entwurf und Realisierung eines Systems zur Prognose und Optimierung der Wartungskosten von Geldautomaten auf Basis maschineller Lernverfahren

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 30. Oktober 2009

Andreas Krohn

Thema der Bachelorarbeit

Entwurf und Realisierung eines Systems zur Prognose und Optimierung der Wartungskosten von Geldautomaten auf Basis maschineller Lernverfahren

Stichworte

Prognose; Maschinelles Lernen; Künstliche neuronale Netze; Wartungsplanung; Geldautomaten

Kurzzusammenfassung

In dieser Bachelorarbeit wird in Kooperation mit der Firma it kompetenz GmbH ein System zur Reduzierung der für den Betrieb von Geldautomaten anfallenden Zinsverluste und Wartungskosten entworfen und prototypisch implementiert. Dazu wurde eine Modul entwickelt, das die Wartungstermine und Füllmengen so wählt, dass die Kosten pro Tag minimal sind. Ein austauschbares Prognoseverfahren liefert dazu die benötigten Abschätzungen zu zukünftigen Abhebebeträgen. Das in dieser Arbeit implementierte Prognoseverfahren nutzt neurale Netze, die mit vergangenen tagesabhängigen Merkmalen und den dort beobachteten Abhebebetrag trainiert werden und anschließend für erneut auftretende Ereignisse einen Abhebebetrag prognostizieren können. Da hierzu keine Echtdaten zur Verfügung stehen, wurde ein Programm entwickelt, das Testdaten generiert. Weiterer Bestandteil dieser Arbeit ist die prototypische Implementierung einer Benutzeroberfläche, die die entworfenen Verfahren für einen Benutzer zugänglich macht.

Andreas Krohn

Title of the paper

Design and implementation of a system to forecast and optimise maintenance costs of ATMs based on machine learning

Keywords

Forecast; Machine Learning; Artificial Neural Networks; Maintenance Schedule; Automated Teller Machines

Abstract

A system to reduce interest loss an maintenance costs for ATMs shall be created in this bachelor thesis. This aim shall be achieved by optimizing the maintenance schedule.

Inhaltsverzeichnis

Abbildungsverzeichnis

Auflistungen

Kapitel 1

Einleitung

Alleine in Deutschland betreiben Banken derzeit über 55.000 Geldautomaten. Es gibt einige Gründe, die Banken zur Bereitstellung dieses Service motiviert. Ein Geldautomat steht dem Kunden rund um die Uhr zur Verfügung und entschärft damit die oft geführte Diskussion um Öffnungszeiten. Außerdem entlasten Geldautomaten die Angestellten am Bankschalter, sodass diese mehr Zeit für Kundenberatung und -betreuung haben.

Die Versorgung der Automaten mit Bargeld erfolgt durch Geldtransporte beauftragter privater Unternehmen. Dafür und für das im Automaten vorgehaltene Geld entstehen den Banken Kosten.

Das Ziel dieser Bachelorarbeit ist es ein System zu entwickeln, das hilft Zinsverluste und unnötige Kosten für zu häufige Wartung (Befüllen und Entleeren) von Geldautomaten zu reduzieren.

Die Arbeit wird in Kooperation mit der Firma it kompetenz erstellt. Die Ergebnisse dieser Arbeit sollen später in bestehende Produkte integriert werden.

Die Motivation zur Erstellung dieser Arbeit sowie die Firma it kompetenz GmbH werden im Folgenden kurz vorgestellt.

1.1 Motivation

Banken und Werttransport-Unternehmen betreiben einen hohen planerischen und finanziellen Aufwand um sicherzustellen, dass der Kunde stets ausreichend gefüllte Geldautomaten vorfindet. Gerechtfertigt wird dieser Aufwand mit dem primären Ziel des Erhalts der Kundenzufriedenheit. Geldautomaten haben sich für Privatpersonen als Bezugsquelle von Bargeld etabliert und werden oft als selbstverständlich betrachtet. Ein leerer Geldautomat, vor allem ein wiederholt leerer, würde auffallen und den Kunden verärgern.

Um einzuschätzen, wann und mit welchem Betrag ein Geldautomat gefüllt werden muss,

benötigt das Geldtransport-Unternehmen oder die Bank ein zuverlässiges Verfahren den Verlauf der Füllstände zu prognostizieren. Über die Geschwindigkeit mit der ein Geldautomat unter verschiedenen Bedingungen geleert ist - das Abhebeverhalten - existieren derzeit lediglich Schätzungen. Damit lässt sich der nötige Füllbetrag und die Fälligkeit einer erneuten Füllung nur sehr grob einschätzen. Im Hinblick auf das primäre Ziel (Kundenzufriedenheit) wird unter diesen Bedingungen die sichere Variante gewählt. Das bedeutet in der Praxis, dass Geldautomaten zu oft oder zu voll aufgefüllt werden. Es besteht also Einsparpotential. Die Entwicklung einer Softwarelösung, die die Erschließung dieses Potentials möglich macht, ist Motivation für das vorliegende Projekt.

1.2 it kompetenz GmbH

Die Firma it kompetenz GmbH wurde im Januar 1995 von den derzeitigen Geschäftsführern

- Matthias Schick
- Thomas Vietze

gegründet. Die Produktpalette umfasst Softwarelösungen im Bereich Geld- und Werttransport. Weiterhin entwickelt die Firma it kompetenz GmbH Individualsoftware im Bereich Logistik und Distribution und stellt ihr Know-How im Gebiet der Client-Server Programmierung sowie Datenbankentwicklung Partnern zur Verfügung (vgl. it kompetenz GmbH (2009)).

Das Produkt zur Planung von Geldtransporten soll um die im Rahmen dieser Arbeit entworfene Funktionalität erweitert werden.

1.3 Aufbau der Arbeit

Im folgenden Kapitel wird auf die aktuelle Situation eingegangen und die momentanen Abläufe beschrieben. Möglichkeiten diese Situation zu verbessern werden in Kapitel 3 herausgearbeitet. In Kapitel 4 wird kurz in das Thema maschinelles Lernen eingeführt und der Einsatz von neuronalen Netzen in dieser Arbeit begründet. Der Entwurf und die Realisierung eines Prototypen, der die entworfenen Ideen umsetzt wird in Kapitel 5 dargestellt. Das Fazit in Kapitel 6 fasst die Ergebnisse dieser Arbeit zusammen und gibt eine Bewertung darüber ab.

Kapitel 2

Ausgangssituation

Die Firma it kompetenz GmbH bietet eine Softwarelösung für Werttansport-Dienstleister an. Die Tourenplanung für Geld- und Werttransporte ist dabei eins der enthaltenen Module. Während einer solchen Tour werden unter anderem Geldautomaten befüllt. Dieser Vorgang wird als Wartung bezeichnet.

Der Auftrag zur Durchführung einer Wartung wird von dem Betreiber - einer Bank - des Geldautomaten erteilt. Der Handlungsspielraum und die Verantwortung, die der Betreiber dem Geldtransport-Unternehmen hierbei einräumt, kann unterschiedlich groß sein. Die Spanne reicht von der expliziten Beauftragung jeder Wartung mit Datum und gewünschter Füllhöhe bis zur vollständigen Delegation der Wartungsplanung und -durchführung an das Geldtransport-Unternehmen unter Bereitstellung tagesaktueller Füllstände der Geldautomaten.

Wartungstermine und Füllmengen bilden den Wartungsplan. Dieser wird nach Vorgabe des Betreibers von Mitarbeitern des Werttransport-Unternehmens erstellt. Welcher Geldbetrag in den Automaten gefüllt wird, ist dabei vom Fassungsvermögen des Automaten begrenzt und von der Abschätzung des planenden Mitarbeiters oder dem Auftraggeber abhängig. Hier sind also menschliche Erfahrungswerte die Grundlage der Wartungsplanung. Die Entscheidung, zu welchem Zeitpunkt und um welchen Betrag ein Geldautomat aufgefüllt wird, trifft der Mitarbeiter momentan ohne Unterstützung von der existierenden Softwarelösung.

2.1 Ablauf einer Wartung

Geldautomaten werden nicht etwa mit losem oder gebündeltem Geld gefüllt, sondern mit verplombten Geldkassetten bestückt. Vor einer Wartung werden die Geldkassetten mit den geplanten Geldbeträgen befüllt. Dies geschieht im Werttransport-Unternehmen unter strengen Sicherheitsvorkehrungen. Das Geld beschafft das Geldtransport-Unternehmen bei einer Landes- oder Bundesbank. Am geplanten Tag fährt dann

ein Team von drei Mitarbeitern in einem abgesicherten Fahrzeug die Route ab. Die Geldkassetten der Automaten entlang dieser Route werden gegen neu bestückte getauscht. Die in den entnommenen Kassetten verbliebenen Geldbeträge werden später gezählt und dem Betreiber gut geschrieben. Auf die Zusammensetzung der hierbei verursachten Kosten wird im nächsten Abschnitt eingegangen.

2.2 Wartungskosten

Der Betrieb eines Geldautomaten verursacht Kosten. Zum Teil sind sie unabhängig vom Geldgeschäft wie zum Beispiel die Kosten für Stromversorgung und Miete. Die Wartungskosten überwiegen jedoch. Sie entstehen im Zusammenhang mit dem Füllen der Geldautomaten. Sie schwanken über die Zeit in Abhängigkeit vom Zinskurs und den Personalkosten.

Die Wartungskosten werden letztendlich von den Banken als Betreiber der Geldautomaten getragen. Die beim eigentlichen Wartungsvorgang verursachten Kosten werden dem Werttransport-Unternehmer pauschal pro Wartung vergütet. Das im Geldautomaten vorgehaltene Geld bringt keine Rendite, sodass hierfür Kosten durch entgangene Zinsen anfallen.

Banken tragen die folgenden Kosten:

- Entgangene Zinsen
- Pauschale pro Füllvorgang an das Werttransport-Unternehmen

Werttransport-Unternehmen kalkulieren ihre Wartungspauschalen auf Grundlage der Kosten für:

- Befüllung der neuen Kasetten
- Bereitstellung des Fahrzeugs
- Personalkosten
- Auszählung der Geldbeträge in entnommenen Kasetten
- Beschaffung des Geldes bei Landes- oder Bundesbank

2.3 Befüllung bei Bedarf

Geldautomaten sind mit ihren Betreibern vernetzt, da beispielsweise bei Abhebevorgängen die Bonität des Kunden geprüft werden muss. Es ist also durchaus möglich, jederzeit den aktuellen Füllstand des Automaten abzurufen oder ihn vom Automaten melden zu

lassen. Hat der Füllstand eine gewisse Grenze unterschritten könnte dann eine Wartung veranlasst werden. Diese Vorgehensweise eliminiert jeglichen Planungsaufwand, hat jedoch gegenüber dem geplanten Durchführen von Wartungen Nachteile. Die Grenze ab der eine Wartung durchgeführt werden soll müsste entweder relativ hoch gewählt werden - was höhere Zinsverluste mit sich bringt - oder die Reaktionszeit für die Vorbereitung und Durchführung einer Wartung reduziert werden - was zu einer weniger effektiven Auslastung der Ressourcen des Geldtransport-Unternehmens führt. Eine Optimierung im Bereich der Planung scheint der beste Weg einer Kostenreduktion zu sein.

Zusammmenfassung

- Maßgeblicher Faktor bei der Planung von Wartungen sind derzeit die Erfahrungswerte der Mitarbeiter
- Die aktuelle Softwarelösung unterstützt nicht bei der Wahl von Wartungstermin und Füllmenge
- Wartungskosten setzen sich aus entgangenen Zinsen und den Kosten für Wartungsvorgänge zusammen
- Die Verantwortung für die Planung von Wartungen kann unterschiedlich zwischen Betreiber und Werttransport-Unternehmer aufgeteilt sein

Kapitel 3

Analyse

Die Wartungsplanung soll zwei wesentliche Ziele erreichen: Ein Geldautomat darf zum einen niemals leer sein und sollte zum anderen einen geringen Zinsverlust verursachen. Diese Vorgaben konkurrieren miteinander. Die kontinuierliche Verfügbarkeit von Bargeld im Geldautomaten lässt sich über höhere eingelagerte Barbestände oder verkürzte Wartungszyklen erreichen. Die Kostensenkung erfordert dagegen, dass nur der vermutete oder erwartete Abhebebetrag im Wartungsintervall vorgehalten wird und dass die Wartungen am optimalen Zeitpunkt stattfinden.

Eine Lösung, die die Kostensenkung *und* die Verfügbarkeit des Geldes garantiert, kann nur durch geschicktes Kombinieren von Füllmenge und Wartungszeitpunkt erreicht werden.

Das System basiert derzeit auf Erfahrungswerten der Beteiligten. Neben den zyklischen Einflüssen (Wochentag, Jahreszeit) spielen auch singuläre Ereignisse eine entscheidende rolle (z.B. „Findet ein Volksfest statt, werden wesentliche höhere Summen abgehoben - die Automaten in der näheren Umgebung sollten vorher aufgefüllt werden“). Diesem komplexen Wissen soll ein Prognosesystem zur Seite gestellt werden. Es soll aus vergangenem Abhebeverhalten lernen und daraus Schlüsse für die Zukunft ziehen.

3.1 Prognose des Abhebeverhaltens

Um einen Wartungsplan aufstellen zu können, wird eine Prognose über das zu erwartenden Abhebeverhalten an den einzelnen Geldautomaten benötigt. Diese Aufgabe erfüllte bisher ein Mitarbeiter des Werttransport-Unternehmens. Hier soll stattdessen automatisch eine, auf kontinuierlich gesammelten Daten gestützte, Prognose erstellt werden. Mit Hilfe dieser Prognose können dann Füllstände prognostiziert und Zeitpunkte für fällige Wartungen bestimmt werden. Hierzu muss ein Weg gefunden werden, die Erfahrung der Mitarbeiter in einem System zu erfassen und nachzubilden.

Die Zusammenhänge, die das Abhebeverhalten in der Realität beeinflussen, können unüberschaubar zahlreich und schwer auszumachen sein. Jedoch liegt die Vermutung nahe, das abhängig von schwerwiegenderen Faktoren tageweise bei gleicher Ausprägung dieser Faktoren ähnliche Summen abgehoben werden. Befinden sich beispielsweise auf Grund eines bestimmten Anlasses mehr Menschen in der näheren Umgebung eines Geldautomaten, steigt die Chance vermehrter Abhebevorgänge.

Zu diesen relevanten Faktoren die jeweilige Abweichung der Abhebemenge einzuschätzen ist als Mensch bei einigen wenigen Faktoren möglich. Überschreitet die Anzahl berücksichtigter Faktoren oder der betrachteten Automaten einen gewissen Wert, so ist die Auswirkung nicht mehr zu überblicken. Dies gilt insbesondere für das gemeinsame Auftreten mehrerer divergierender Faktoren an einem Tag.

Hier bieten sich maschinelle Lernverfahren (siehe Kapitel 4) an. Sie werden mit aus der Vergangenheit bekannten Wertepaaren aus Abhebemenge pro Tag und an diesem Tag aufgetretenen Faktoren trainiert. Erlernte Zusammenhänge von Faktoren und Abhebemenge erlauben es dann Aussagen für die Zukunft zu treffen.

Vermutlich können bereits aufgrund der Wochentage grobe Tendenzen für die zu erwartende Abhebesumme geliefert werden. Da aber durch auftretende Ereignisse verursachte Anomalien berücksichtigt und prognostiziert werden sollen, ist die Pflege der Kalender zu den Geldautomaten zwingend erforderlich.

3.1.1 Beschränkung auf synthetische Daten

Abhebevorgänge an Geldautomaten sind sensible Daten, die von den Quellen (Banken und Werttransport-Unternehmen) aus Gründen des Datenschutzes nicht herausgegeben werden. Um dieses Projekt dennoch durchführen zu können, wird auf die Erstellung künstlicher Füllstände von Geldautomaten zurückgegriffen. Näheres hierzu wird in Abschnitt 5.4 erläutert.

Der erfolgreiche Abschluss dieser Bachelorarbeit wird mit den erwarteten Kosteneinsparungen der Firma it kompetenz GmbH Argumente liefern, mit einem dieser Unternehmen eine Vereinbarung zu treffen, die es ermöglicht mit realen Daten zu arbeiten.

3.1.2 Behandlung von statischen und Standortfaktoren

Eine identische Kombination aus aufgetretenen Faktoren kann sich abhängig vom Standort, der Zugänglichkeit und anderen Attributen pro Geldautomat anders auf die Abhebemenge auswirken.

Beispiel:

An einem leicht zugänglichen Geldautomat in einer Fussgängerzone wird mehr Geld abgehoben als an einem in ländlicher Gegend. Dies gilt auch, wenn zeitabhängige

Faktoren wie Datum, ein Wochenmarkt in der Nähe, Schulferien für beide Geldautomaten identisch sind. Der Wochenmarkt im ländlichen Raum zieht relativ konstant die selbe Menge Kunden an. Der in der Fussgängerzone hat in den Schulferien deutlich weniger Kunden, da in der Umgebung hauptsächlich gut situierte Familien mit Schulkindern wohnen, die dann im Urlaub sind.

Der Standort und andere statische Attribute spielen also eine Rolle bei der Prognose von Abhebevorgängen. Diese Tatsache kann auf zwei Weisen berücksichtigt werden.

Allgemeingültige globale Prognosekomponente

Das Prognoseverfahren versucht statische und zeitabhängige Faktoren zu berücksichtigen und daraus auf ein bestimmtes Abhebeverhalten zu schließen. Dieser Ansatz hat den Charme, dass für neu in das Verfahren aufgenommene Geldautomaten lediglich die Ausprägung der statischen Faktoren bekannt sein muss um Prognosen erstellen zu können. Einmal erfasste Zusammenhänge zwischen bestimmten Faktoren und Abhebeverhalten können verallgemeinert werden. Die angestrebte globale Gültigkeit erhöht jedoch die Komplexität des Prognoseverfahrens. Einerseits müssen neben den dynamischen auch die statischen Faktoren identifiziert werden, andererseits ihre Ausprägungen und Auswirkungen erfasst und berücksichtigt werden.

Individuelle Prognosekomponenten

In der Natur statischer Faktoren liegt es, dass sie für den jeweiligen Geldautomaten über die Zeit gesehen konstant sind. Betrachtet man jeden Geldautomaten für sich, verlieren Standortfaktoren ihre Wirkung. Sie beschreiben lediglich den Unterschied zwischen Geldautomaten, jedoch keinerlei Änderungen des Abhebeverhaltens an einem einzelnen Geldautomaten. Angewendet auf das Prognoseverfahren, bedeutet das, dass die Modellierung statischer Einflüsse umgangen werden kann. Hierzu muss die Sicht auf jeweils einen Geldautomaten beschränkt werden. Jede Prognosekomponente wird dafür einem Geldautomaten zugeordnet und an das hier beobachtete Abhebeverhalten in Abhängigkeit dynamische Faktoren angepasst. Dieser Ansatz verspricht ein leichter handhabbares Prognoseverfahren, da Unterschiede zwischen Geldautomaten und deren Auswirkungen weder identifiziert noch modelliert werden müssen.

Es ist weiterhin durchaus denkbar, dass sich für unterschiedliche Geldautomaten unterschiedliche Prognosealgorithmen anbieten. Der mobile, auf Volksfesten aufgestellte Geldautomat ist in seinem Abhebeverhalten vom Besucheraufkommen abhängig, was stark vom Wochentag und Wetter beeinflusst wird. Ein fest installierter Geldautomat im Umfeld des Volksfestes ändert sein Verhalten während der Veranstaltung gegenüber dem alltäglichen stark. Würde eine globale Prognosekomponente erstellt werden, müssten solche Ausnahmen erkannt und behandelt werden. Die Verwendung individueller Komponenten erlaubt es pro Geldautomat das jeweils passende Verfahren zu wählen

und auf diesem Weg eine höhere Flexibilität zu erhalten. Gleichzeitig reduziert sich hier die Komplexität, da eine Prognosekomponente auf einen speziellen Geldautomaten angepasst werden kann und nicht alle möglichen Abhebeverhaltensmuster erlernen und unterscheiden muss.

Entscheidung

Tabelle 3.1 fasst die in den vorigen Abschnitten erarbeiteten Bewertungen zusammen. Auf Grund der Beschränkung auf synthetische Daten (Abschnitt 3.1.1) erhält der Faktor Flexibilität eine herausragende Bedeutung. Der Test des Systems mit realen Daten könnte Änderungsbedarf am Prognoseverfahren offenbaren. In diesem Punkt erleichtert ein flexibler und mit geringer Komplexität behafteter Ansatz die Weiterentwicklung. Aus diesen Gründen fällt die Entscheidung gegen ein globales und zu Gunsten eines auf den jeweiligen Geldautomaten angepassten Prognoseverfahrens.

	global	**individuell**
	alle Faktoren	*nur* dynamische Faktoren
Komplexität	-	+
Benötigte Beispieldaten	+	o
Flexibilität	-	+

+ gut, o neutral, - schlecht

Tabelle 3.1: Bewertung: globale oder individuelle Prognosekomponente

3.2 Möglichkeiten der Kostenbeeinflussung

Anhand der prognostizierten Abhebebeträge sollen Entscheidungen gefällt werden, die die für den Betrieb von Geldautomaten entstehenden Kosten reduzieren. Dabei sind allerdings nicht alle Kostenstellen beeinflussbar. Eine Kostenminimierung kann aber nur durch das Verstellen beeinflussbarer Faktoren erreicht werden. Diese Faktoren sollen im Folgenden identifiziert werden.

Ein Geldtransporter ist auf der Tour zu den Geldautomaten mit mindestens drei Mitarbeitern bestückt. Einer bleibt während der Wartung beim Fahrzeug, zwei Mitarbeiter tauschen die Geldkassetten der Automaten aus (Vier-Augen-Prinzip). Die Versicherung des Transportes macht es erforderlich, dass das Fahrzeug ständig besetzt ist. Das Vier-Augen-Prinzip beim Austausch der Kassetten bietet den Mitarbeitern Schutz vor Überfällen oder unberechtigten Anschuldigungen sich bereichert zu haben. Hier lassen sich keine Einsparungen vornehmen (abgesehen von Gehaltskürzungen, die aber nicht in das hier behandelte Fachgebiet fallen). Diese Einschätzung lässt sich auf das Personal für das Befüllen und Auszählen der Geldkassetten übertragen.

Der Zinssatz, zu dem Geld geliehen werden kann, variiert über die Zeit gesehen und ist extern vorgegeben.

Die bei der Wartung eingefüllte Menge Geld ist variabel. Sie ist jedoch begrenzt. So hat ein Geldautomat ein maximales Fassungsvermögen über das hinaus er nicht gefüllt werden kann. Wäre diese technische Grenze nicht, gäbe es weiterhin den limitierenden Faktor der Versicherung. Die Füllmenge ist also in Grenzen wählbar.

Wartungstermine können verschoben werden, sodass die Wartungspauschale seltener oder häufiger anfällt. Das Verzögern des Auffüllens senkt die Kosten für Wartungspauschalen, erhöht jedoch die Gefahr, dass ein Geldautomat leer läuft und damit Kunden unzufrieden werden. Wird die Frequenz der Wartungen erhöht, steigen gleichzeitig die durch fällige Wartungspauschalen verursachten Kosten. Der Faktor Wartungstermin ist beeinflussbar, es muss jedoch sichergestellt werden, dass der Automat zu keinem Zeitpunkt leer ist.

Was bleibt sind die geschickte Wahl von Wartungszeitpunkt und Füllmenge, sodass über einen bestimmten Zeitraum betrachtet die Summe aus den Kosten für fällige Auffüllvorgänge und den Zinskosten minimiert wird. Die Ermittlung einer möglichst günstigen Kombination von Wartungszeitpunkt und Füllmenge sind Aufgabe der Wartungsplanung und Kostenoptimierung.

3.3 Kostenoptimierung

Die eigentliche Optimierung der Kosten findet als ein der Prognose der Abhebevorgänge nachgelagerter Schritt statt. Ausgehend von einem letzten manuell erfassten realen Füllstand wird für eine ausgewählte Zeitspanne der voraussichtliche Verlauf der Füllstände prognostiziert und ein möglichst kostengünstiger Plan für Wartungstermine und Auffüllmengen erstellt. Die Trennung zwischen Optimierung und Prognose der Abhebevorgänge ist möglich, da sich der Wartungsplan nicht auf die Abhebevorgänge auswirkt.

Die von der Prognosekomponente gelieferten Abhebebeträge werden dazu benutzt den Füllstand der Geldautomaten über die verfügbaren Daten hinaus zu bestimmen. Spätestens wenn der Automat leer ist muss dann eine Wartung durchgeführt werden (besser aber vorher, ein leerer Automat soll ja gerade vermieden werden). Neben dem Zeitpunkt, muss ein Füllbetrag gewählt werden, um eine Wartung zu planen. Dieser Betrag beeinflusst die Zinskosten und die Zeitspanne für die anschließend keine Wartung nötig ist. Wird der Geldautomat bei einer Wartung auf sein maximales Fassungsvermögen gefüllt, ist der Zeitraum in dem keine erneute Wartung fällig ist länger, als wenn er zu drei Vierteln oder zur Hälfte gefüllt wird (vgl. Abb. 3.1). Die Höhe des Füllbetrages ist damit zentrale Stellschraube für die Minimierung der Wartungskosten.

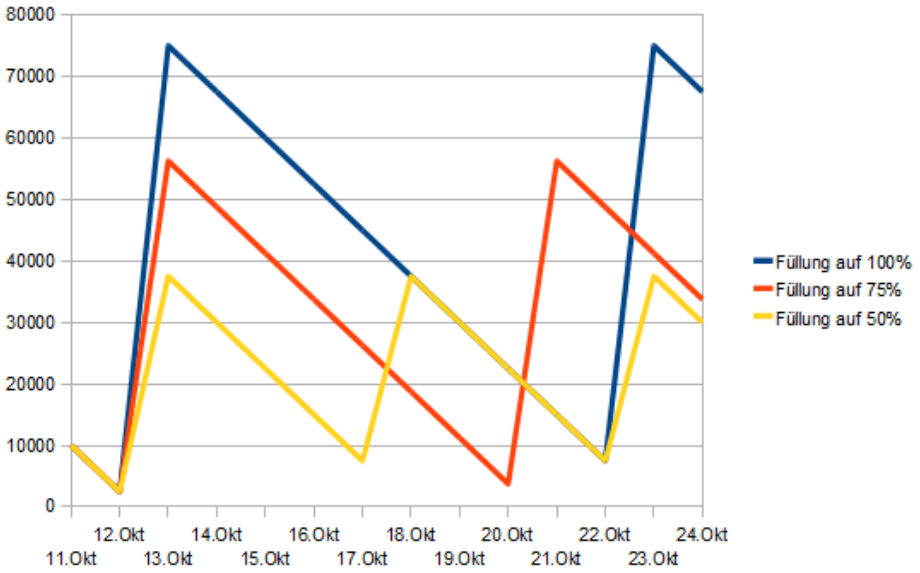

Abbildung 3.1: Einfluss der Füllhöhe auf das Wartungsintervall (idealisiert)

3.3.1 Dimension des Suchraums

Den Wartungsplan zu ermitteln, der zu minimierten Kosten führt, wäre in einem Bruteforce-Verfahren theoretisch möglich. Dieses Verfahren würde bereits erarbeitete Zusammenhänge außer Acht lassen und stattdessen alle Kombinationen aus Wartungstermin und Füllbetrag generieren. Der Suchraum ist begrenzt, da es im Planungszeitraum eine endliche Anzahl möglicher Wartungstermine gibt und die möglichen Füllbeträge endlich sind. Die vollständige Enumeration des Suchraums ist jedoch aufgrund seiner Größe unpraktikabel.

Beispiel:

- Prognose für 14 Tage - maximal $2^{14} = 16.384$ Kombinationen Wartungen zu planen
  - 14 mal Wartung oder keine Wartung
- Mögliche Füllbeträge je Wartung $\frac{20.000}{5} = 4.000$
  - Maximaler Füllstand 20.000 €
  - Kleinste Banknote 5 €
- **Suchraum:** $16.384 \times 4.000 = 65.536.000$ Kombinationen
  - Zu jeder dieser möglichen Wartungsplanung müsste eine Kostenberechnung zu dessen Beurteilung durchgeführt werden.
  - Bei der parallelen Betrachtung von mehreren Automaten um Wartungen an aufeinander folgenden Tagen zusammenzufassen wächst die Komplexität des Problems schlimmstenfalls exponentiell wenn das Kreuzprodukt der

möglichen Planungen gebildet und jede Variante auf Kosten untersucht wird. Dieser Aspekt einer Clusterbildung von Geldautomaten, die gleichzeitig gewartet werden, ist denkbar, wird jedoch auf Wunsch der Firma it kompetenz GmbH außen vor gelassen.

Die Suche lässt sich beschleunigen indem die Extremfälle

- tägliche Wartung bei minimaler Füllung (bei hohem Zinssatz)
- maximale Füllung bei möglichst seltener Wartung (bei niedrigem Zinssatz)

zuerst überprüft werden. Ausgehend von der kostengünstigeren Variante kann dann der Füllbetrag verringert oder erhöht werden, solange die Kosten fallen. Weiterhin sollten Wartungen immer nur dann eingeplant werden, wenn die prognostizierten Abhebevorgänge den Geldautomaten geleert haben. Berechnet das Suchverfahren parallel zur Planung die anfallenden Kosten, lässt sich die Effizienz weiter steigern.Steigen die Kosten gegenüber bereits ermittelten Lösungen, kann die Suche in der aktuellen Richtung dann abgebrochen werden. Basierend auf diesen Ansätzen wird im Folgenden eine Vorgehensweise zum Aufstellen des Wartungsplans konstruiert.

3.3.2 Funktion zur Wartungsplanung

Ausgehend von einem letzten realen Füllstand werden solange prognostizierte Beträge „abgehoben“, bis eine Sicherheitsmarge S unterschritten ist. Der Tag an dem dies eintritt ist damit der erste Wartungstermin w_0.

Die Planungsfunktion sucht anschließend einen nächsten Wartungstermin w_1. Dieser wird basierend auf den anfallenden Kosten gewählt. Die Gesamtkosten für eine Wartungsperiode setzten sich aus den Kosten für die Befüllung (der Wartungspauschale) und den entgangenen Zinsen zusammen. Die Wartungspauschale $W(d)$ kann Preisänderungen unterliegen - es wird jeweils der Preis an w_1 verwendet, um die Schwankungen in die Wahl des optimalen w_1 einfließen zu lassen. Zinskosten fallen für das am jeweiligen Tag im Automaten vorhandene Geld in Abhängigkeit vom Zinssatz $Z(d)$ (in %) an. In die Berechnung der Kosten fließt der Zinssatz des Starttages w_0 ein, da das Geld zu den hier herrschenden Bedingungen geliehen wird. Damit der Automat einen Tag „aushält“, reicht es den für diesen Tag prognostizierten Abhebebetrag $p(d)$ - zuzüglich Sicherheitsmarge S - einzufüllen - für mehrere Tage wird die Summe ihrer Abhebebeträge benötigt. Die Funktion 3.1 zeigt die Berechnung der Gesamtkosten für eine Wartungsperiode $K_t(w_i, w_{i+1})$.

$$K_t(w_i, w_{i+1}) = W(w_{i+1}) + \frac{(w_{i+1} - w_i + 1) \cdot S \cdot Z(w_i)}{360 \cdot 100} + \sum_{d=w_i}^{w_{i+1}} \frac{(d - w_i + 1) \cdot p(d) \cdot Z(w_i)}{360 \cdot 100} \quad (3.1)$$

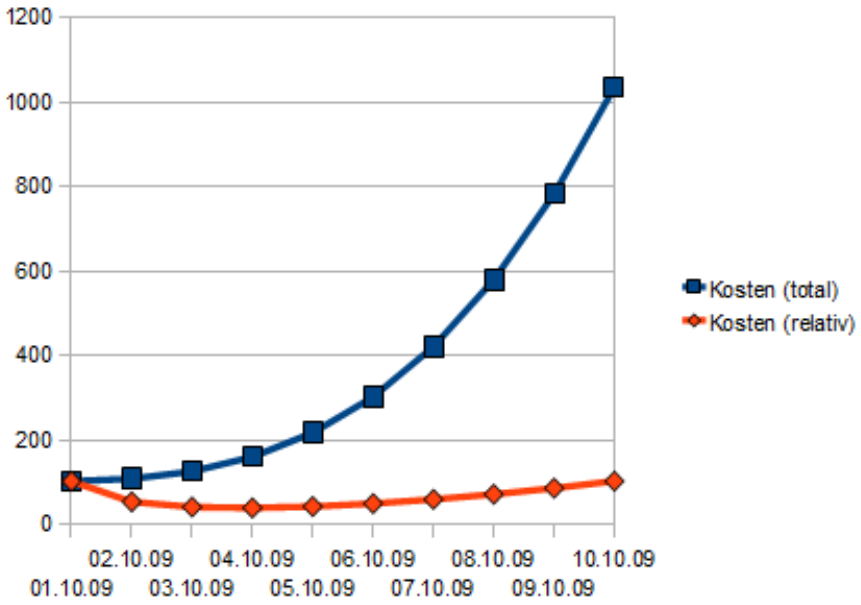

Abbildung 3.2: Totale und relative Kosten (idealisiert)

Der eingefüllte Geldbetrag bei der Wartung wird so gewählt, dass die Kosten pro Tag bis zum nächsten Wartungstermin minimal sind. Berechnet man die Gesamtkosten und teilt sie durch die Anzahl der Tage der Wartungsperiode, so erhält man für eine mögliche Wartungsperiode vergleichbare relative Kosten $K_r(w_i, w_{i+1})$ (siehe Gleichung 3.2). Der Tag an dem die relativen Kosten minimal sind, wird als nächster Wartungstermin w_{i+1} gesetzt.

$$K_r(w_i, w_{i+1}) = \frac{K_t(w_i, w_{i+1})}{(w_{i+1} - w_i + 1)} \tag{3.2}$$

Abb. 3.2 zeigt idealisiert die Entwicklung von relativen und absoluten Kosten bei steigender Dauer einer Wartungsperiode. Die Gesamtkosten steigen kontinuierlich. Die relativen Kosten sinken bedingt durch das Aufteilen der Befüllungskosten auf mehrere Tag zunächst. Nach Erreichen eines Minimums am gesuchten Tag w_{i+1} steigen sie wieder, da die Zinskosten an Gewicht gewinnen.

Bei der Suche nach einer optimalen Wartungsplanung muss weiterhin berücksichtigt werden, dass das Fassungsvermögen des Automaten nicht überschritten wird. Weiterhin soll der Benutzer manuell Wartungstermine oder Wartungen mit Termin und Füllbetrag festlegen können. Auf deren Behandlung wird im folgenden Abschnitt eingegangen.

3.3.3 Manuelle Eingriffe

Wie bereits ausgeführt, gestehen Betreiber von Geldautomaten dem Werttransport-Unternehmen unterschiedliche Freiheiten bei der Planung von Wartungen zu. Eingeschränkte Freiheitsgrade, d.h. vom Betreiber vorgegebene Wartungen sollen im System berücksichtigt werden.

Dies geschieht in Form von manuell vorgegebenen Wartungsterminen (nötige Füllhöhe wird vom System vorgeschlagen) und fest vorgegebenen Wartungen (Termin und Füllhöhe vom Betreiber vorgegeben). Trifft das Programm bei der Aufstellung eines Plans auf bereits vorgegebene, zukünftige Wartungstermine, haben diese Vorrang gegenüber anderen, eventuell kostengünstigeren Terminen.

Ein denkbarer Einsatzzweck dieses Features ist das folgende Szenario:

> Ein Geldautomat ist wegen Straßenbauarbeiten vorübergehend für Geldtransporte schlecht erreichbar. Dies wird vom System bei der Planung mangels Kenntnis nicht berücksichtigt. Hier kann der Benutzer für den letzten Tag vor den Bauarbeiten eine Auffüllung vorgeben, damit eine bei schlechter Erreichbarkeit möglicherweise zeit- und damit kostenintensivere Wartung hinausgezögert oder vermieden wird.

Abgesehen von einer solchen Ausnahmesituation gibt es wie bereits angedeutet Banken, die die Planung ihrer Geldautomatenbefüllung im eigenen Haus durchführen und diese Organisation vorerst beibehalten werden.

Zusammenfassung

- Auf individuelle Geldautomaten angepasste Komponenten prognostizieren das Abhebeverhalten. Dafür werden in einem pro Geldautomaten geführten Kalender eingetragene zeitabhängige Faktoren herangezogen.
- Nicht alle kostenverursachenden Umstände im Betrieb eines Geldautomaten sind beeinflussbar. Wartungstermin und Einfüllbetrag sind beeinflussbar. Eine kostengünstige Kombination dieser Parameter zu finden ist Ziel der Kostenoptimierung.
- Die Kostenoptimierung braucht einen Wert anhand dessen sich verschiedene Konstellationen aus Wartungszeitpunkt und Füllbetrag vergleichen lassen. Hierzu werden die Kosten pro Tag verwendet.
- Manuelle Eingriffe werden bei der automatisierten Planung von Wartungen berücksichtigt.

Kapitel 4

Maschinelles Lernen

Maschinelles Lernen hat das Ziel Aspekte des menschlichen Lernens nachzustellen und zu formalisieren. Dabei ist der Begriff *Lernen* schwer präzise zu definieren. Michalski (1986, S. 11) bietet die folgende Definition an:

> Learning is constructing or modifying representations of what is being experienced.[1]

Anhand der Repräsentation der Erfahrungen - des Wissens - lassen sich verschiedene maschinelle Lernverfahren kategorisieren. Man unterscheidet symbolische und subsymbolische Systeme.

- **Symbolische Systeme** stellen Informationen explizit dar. Dies geschieht in Form von Regeln, Logiken oder strukturierten Objekten (Keller, 2000, S. 36). Erlerntes Wissen wie beispielsweise eine aus speziellen Regeln abgeleitete allgemeine Regel wird explizit angelegt und ist oft auch für einen Benutzer verständlich und interpretierbar.

  Ein Beispiel: Die Fakten `kugel(Fussball)`, `kugel(Murmel)`, `rollt(Fussball)`, `rollt(Murmel)` sind gegeben. Daraus könnte ein symbolisches Lernverfahren verallgemeinern, dass etwas, das eine Kugel ist, rollt. Dieser Lernerfolg würde sich in einer expliziten Regel `kugel(x)`$\rightarrow$`rollt(x)` manifestieren.

- **Subsymbolische Systeme** arbeiten dagegen mit impliziten Wissensrepräsentationen. Informationen werden hier beispielsweise im neuronalen Netz durch die Verknüpfung einfacher Einheiten repräsentiert.

[1] Übersetzung aus Morik (1993, S. 4): „Lernen ist das Konstruieren oder Verändern von Repräsentationen von Erfahrungen."

Ein weiteres Kriterium zur Klassifizierung von Lernverfahren ist die Gestaltung des Lernprozesses. Hierbei unterscheidet man das überwachte, unüberwachte und bestärkende Lernen.

- Beim **überwachten Lernen** sind sowohl die Eingaben X als auch die Ausgaben Y bekannt. Ziel der Lernens ist es eine Abbildung $A : X \to Y$ zu finden oder zu approximieren. Der Lernerfolg kann über das Testen der Abbildungsfunktion mit nicht beim Lernen verwendeten Daten überprüft werden. Wurde eine gute Approximation für die zu erlernende Abbildung gefunden, ist die Abweichung der von der Funktion gelieferten Ausgaben von den in den Testdaten enthaltenen gering.
- Beim **unüberwachten Lernen** sind nur Eingabewerte bekannt. Ziel des Lernens ist es ein Modell der Eingabedaten zu erstellen, das beispielsweise Häufungen bestimmter Eingabemuster oder Assoziationen beschreibt.
- Beim **bestärkenden Lernen** werden dem Lernenden lediglich Eingabewerte präsentiert. Er generiert zu der Eingabe eine Ausgabe, die anschließend als richtig oder falsch bewertet wird. Die vollständige, eigentlich erwartete oder richtige Ausgabe zu einer Eingabe ist nicht bekannt. Das Lernverfahren muss also durch Versuch und Irrtum ein günstiges - also öfter richtig als falsch liegendes - Verhalten suchen.

4.1 Beispiele für Einsatzszenarien

Maschinelle Lernverfahren werden zum Beispiel eingesetzt, wenn die Menge der Daten zu groß ist, eine klassische statistische Analyse effizient durchzuführen - beim sog. Datamining. Onlineshops setzen diese Technik ein. Kunden werden in Abhängigkeit von den Produkten in ihrem Warenkorb weitere Artikel vorschlagen, die von Kunden mit ähnlichem Kaufverhalten ausgewählt wurden. Hierbei wird das Ziel verfolgt, durch diesen Service die Zufriedenheit der Kunden zu steigern und mittelbar mehr Umsatz zu generieren. Die Extraktion eines Kundenverhaltens aus vergangenen Bestellungen ist auch mit einer manuellen Auswertung möglich. Sie nimmt aber mehr Zeit in Anspruch als eine maschinelle statistische Analyse und rentiert sich damit aus betriebswirtschaftlicher Sicht nicht.

Ein anderes prominentes Beispiel für den Einsatz maschineller Lernverfahren ist die Handschrifterkennung. Jeder Mensch hat seine Eigenarten in der Handschrift. Maschinelle Lernverfahren bieten sich hier mit ihrer Fähigkeit, aus Beispielen zu lernen, an. Die handschriftliche Eingabe wird per Mustererkennung und automatischer Klassifikation als ein Buchstabe oder Wort erkannt. Fehlinterpretationen einer Eingabe könnten vom Benutzer als solche deklariert werden. Alleine der Hinweis auf das Vorliegen eines Fehlers (entspräche bestärkendem Lernen) oder die Information, was die richtige Interpretation gewesen wäre (entspräche überwachtem Lernen), trägt zum Training und damit zur Verbesserung der Erkennungsleistung bei. In der Handschrifterkennung spielen subsym-

bolische Lernverfahren ihre Vorteile aus. Im kriminalistischen Verfahren der Schriftvergleichung werden unter anderem Schriftmerkmale erhoben. Angesichts der Komplexität und Vielzahl dieser Merkmale erscheint es jedoch aussichtslos, sie mit expliziten Regeln oder Formeln zu beschreiben. Ein subsymbolisches Lernverfahren umgeht dieses Problem und erlernt eine „6“ von einem „G“ oder ein „i“ von einem „j“ zu unterscheiden, ohne deren Charakteristika explizit umschreiben zu müssen. Eine ähnliche Vorgehensweise ist auch für die Stimmerkennung einsetzbar.

4.2 Künstliche neuronale Netze

Künstliche neuronale Netze sind eine Umsetzung eines subsymbolischen Lernverfahrens. Sie orientieren sich in Aufbau und Funktionsweise an biologischen Gehirnen. Biologische Nervenzellen werden in vereinfachter und idealisierter Form als künstliche Neuronen (im Folgenden *Neuronen*) simuliert. Mehrere Neuronen werden zu einem neuronalen Netz verbunden (vgl. Abb. 4.1). Diese Verbindungen sind mit variablen Gewichten behaftet und bilden das vereinfachte künstliche Pendant zu biologischen Synapsen. Das erlernte Wissen manifestiert sich im neuronalen Netz in Form der Struktur und Ausprägung der Verbindungen zwischen den Neuronen und der Anzahl und Anordnung der Neuronen.

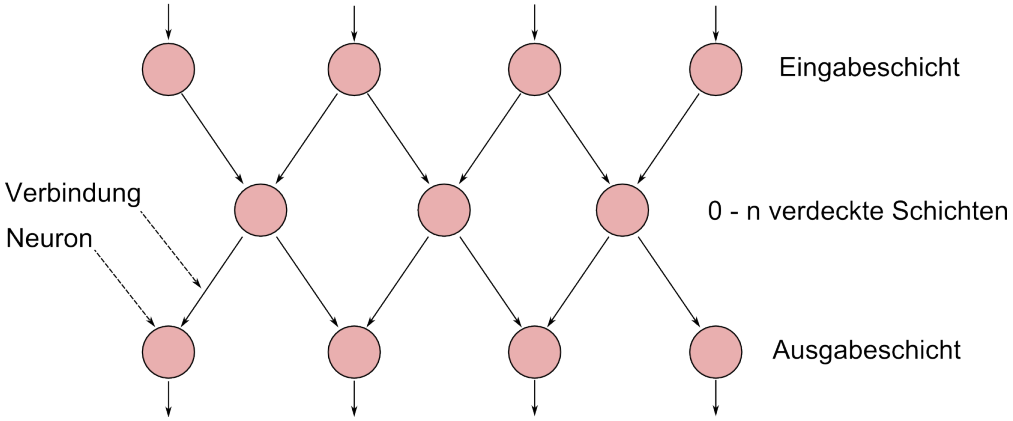

Abbildung 4.1: Schematischer Aufbau eines neuronalen Netzes

Die Nachbildung eines neuronalen Netzes in informationstechnischen Systemen ist stark vereinfacht. Die massive Parallelität, die in biologischen neuronalen Netzen stattfindet, lässt sich beispielsweise nicht oder nur mit großem Aufwand (beispielsweise mit verteilten Systemen) simulieren.

Die Bausteine eines künstlichen neuronalen Netzes - die Neuronen - sind verhältnismäßig einfach aufgebaute Einheiten. Nach Zell (2003, S. 72) sind die Bestandteile eines Neurons die Folgenden (vgl. Abb. 4.2)

- Neuronen besitzen einen Aktivierungszustand $a_j(t)$

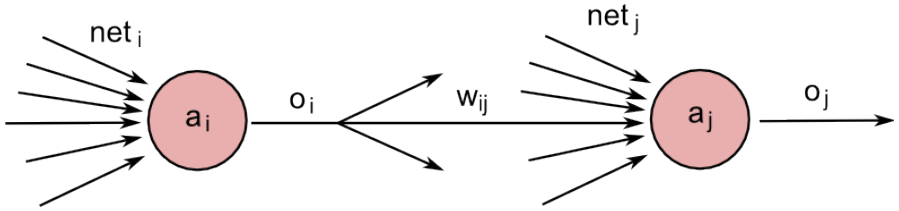

Abbildung 4.2: Darstellung von Zellen eines neuronalen Netzes (Zell, 2003, S. 72)

- Sie nehmen eine Netzeingabe von anderen Neuronen oder von außen entgegen $net_j(t)$. Die Netzeingabe wird über eine Propagierungsfunktion aus den Ausgaben der verbundenen Neuronen berechnet. Dabei wird die Summe der mit dem Gewicht w_{ij} ihrer Verbindung multiplizierten Ausgaben o_i gebildet:

$$net_j(t) = \sum_i o_i(t) \cdot w_{ij} \tag{4.1}$$

- Eine Aktivierungsfunktion f_{act} berechnet aus der Eingabe $net_j(t)$, dem alten Aktivierungszustand $a_j(t)$ und dem Schwellwert θ des Neurons den neuen Aktivierungszustand $a_j(t+1)$.

$$a_j(t+1) = f_{act}(a_j(t), net_j(t), \theta_j) \tag{4.2}$$

- Über die Ausgabefunktion f_{out} wird zu einem Aktiverungszustand a_j ein Ausgabewert o_j berechnet. Der Ausgabewert wird über gewichtete Verbindungen Bestandteil der Netzeingabe weiterer (oder des ursprünglichen) Neuronen.

Das Spektrum der Ausgaben eines Neurons reicht in Abhängigkeit von der nach Wahl der Aktivierungs- und Ausgabefunktion von linear bis binär. Eine binäre Ausgabe wird realisiert, indem unter einem Schwellwert beispielsweise null, darüber eins ausgegeben wird. Soll mit Hilfe des neuronalen Netzes eine qualitative Aussage getroffen werden, bietet es sich sich an eine Schwellwertfunktion einzusetzen. Für quantitative Aussagen eignen sich (semi)lineare Aktivierungs- und Ausgabefunktionen.

Im einfachsten Fall besteht ein neuronales Netz aus lediglich einem Neuron. Sind es mehr, werden die Neuronen anhand ihrer Aufgaben oder der Anordnung ihrer Verknüpfungen in Schichten zusammengefasst. Eingaben nimmt ein Netz über Eingangsneuronen entgegen. Ausgaben werden über Ausgangsneuronen an die Außenwelt übergeben. Zwischen Ein- und Ausgangsschicht können sich beliebig viele versteckte Schichten befinden (vgl. Abb. 4.1) .

Gehen die Verbindungen innerhalb eines Netzes nur in Richtung Ausgabe, spricht man von einem **vorwärtsgekoppelten Netz**. Ein solches Netz kann Verbindungen haben, die Schichten überspringen (*shortcut connections*) oder ebenenweise verbunden sein.

Sind Verbindungen von Neuronen nicht nur zur Ausgabeschicht gerichtet, wird ein solches Netz als rückgekoppelt bezeichnet. **Rückkopplung** kann direkt (Neuron verwendet

seine Aus- als Teil der Eingabe), indirekt (Ausgabe eines Neurons mit vorangehender Schicht verbunden) oder lateral (Neuron mit einem anderen Neuron der selben Schicht verbunden) sein.

Ein weiterer Spezialfall einer Netzarchitektur ist das **vollständig verbundene Netz**. Dabei besteht zwischen allen Neuronen paarweise eine bidirektionale Verbindung.

4.2.1 Lernen im neuronalen Netz

Das Lernen im neuronalen Netz kann auf unterschiedliche Weise realisiert werden. Die am häufigsten eingesetzte Vorgehensweise ist das Anpassen der Verbindungsgewichte. Theoretisch denkbar wäre weiterhin das Hinzufügen und Entfernen von Neuronen oder Verbindungen zwischen Neuronen. Auch könnten Propagierungsfunktion, Aktivierungsfunktion oder Schwellwert des Neurons verändert werden. In der Forschung und Entwicklung sieht Zell (2003, S. 84) eine wachsende Bedeutung des Löschens und Hinzufügens von Neuronen im Zuge des Lernens. In der Praxis sind die Gewichte momentan *die* Stellschraube, an der Lernverfahren drehen.

Lernverfahren für neuronale Netzen haben ihren Ursprung in der Hebbschen Regel. Sie besagt, dass das Gewicht w_{ij} einer Verbindung von Neuronen j zu i erhöht wird, wenn die Ausgabe o_j des einen und das Aktivierungspotential a_i des anderen Neurons gleich groß ist. Δw_{ij} ist hierbei der Betrag, um den das Verbindungsgewicht verändert wird. η ist eine Konstante - die Lernrate.

$$\Delta w_{ij} = \eta \cdot o_j \cdot a_i \tag{4.3}$$

Überwachtes Lernen

Das überwachte Lernen eines neuronalen Netzes besteht im wesentlichen aus den folgenden Schritten zitiert aus (Kriesel, 2007, S. 53):

> **Eingabe** des Eingabemusters (Aktivierung der Eingabeneuronen),
>
> **Vorwärtspropagierung** der Eingabe durch das Netz, Erzeugung der Ausgabe,
>
> **Vergleich** der Ausgabe mit der korrekten Ausgabe (*Teaching Input*), liefert Fehlervektor (Differenzvektor),
>
> **Verbesserungen des Netzes** werden aufbauend auf den Fehlervektor berechnet.
>
> **Anwendung der Verbesserung** um vorher berechneten Werte.

Die Abbildung von n Verbindungsgewichten auf den Fehler lässt sich als Ebene im $n+1$ dimensionalen Raum darstellen. Die Suche nach dem globalen Minimum oder eines

dem Minimum nahe liegenden Punktes in dieser Ebene ist die Aufgabe eines überwachten Lernverfahrens für neuronale Netze. Diese Aufgabe erfüllen Gradientenverfahren. Backpropagation ist ein solches Gradientenverfahren. Es hat zum Ziel eine Belegung der Gewichte zu finden, für die der Fehler gegenüber den Trainingsdaten minimal wird. Es bringt die Gradientenverfahren eigenen Probleme mit sich.

Lokale Minima Die Suche nach einem Minimum in der Fehlerebene kann es passieren, dass das Lernverfahren ein lokales Minimum findet und dieses nicht mehr verlassen kann. Das globale Minimum der Fehler würde in diesem Fall nicht gefunden werden.

Stagnation auf flachen Plateaus Trifft das Lernverfahren bei der Suche auf ein flaches Plateau, ist es möglich, dass ein weiterer Abstieg in Richtung Minimum stark verlangsamt wird. Die Steigung könnte sogar so gering sein, dass es zum Stillstand kommt.

Verlassen guter Minima Eine steile Steigung könnte das Verfahren zu einem so weiten Sprung veranlassen, dass das ein gutes Minimum verlassen und jenseits der übersehenen Schlucht in der Fehlerebene weiter gesucht wird.

Diese Probleme lassen sich durch Modifikation des Verfahrens umgehen oder zumindest in ihren Auswirkungen abschwächen.

Backpropagation geht beim Training nach dem eingangs erwähnten Ablauf vor. Ein Eingabewert wird an die Eingabeneuronen übergeben. Die Aktivierung wird durch das Netz vorwärtspropagiert und die Ausgabe mit der korrekten Ausgabe verglichen. Der aus diesem Vergleich resultierende Differenzvektor wird anschließend verwendet, um die Verbindungsgewichte anzupassen.

Für das in der vorliegenden Arbeit zu lösende Problem sind sowohl zu den Eingaben als auch zu den erwarteten Ausgaben Trainingsdaten vorhanden. Deshalb eignet sich ein überwachtes Lernverfahren. Auf Umsetzung anderer Möglichkeiten neuronale Netze zu trainieren wird nicht näher eingegangen.

4.3 Support Vector Machine

Eine Support Vector Machine (SVM) ist ein überwacht lernendes maschinelles Lernverfahren, das primär eingesetzt wird, um Objekte zu klassifizieren. Dazu wird eine Menge von Trainingsobjekten und deren Klassenzugehörigkeit als Vektorschar dargestellt. Ziel des Trainings der SVM ist es nun eine Hyperebene zu finden, die so zwischen die Trainingsvektoren gelegt wird, dass diese nach Klassenzugehörigkeit getrennt und der geringste Abstand eines Vektors zur Ebene maximal ist. Den größten Einfluss auf die

Lage der Hyperebene haben die näher der Klassengrenze liegenden Vektoren, die Support oder Stützvektoren. Solche die weiter entfernt - also „sicher“ in der Klasse - liegen spielen keine Rolle für das Training.

Sind die Klassen linear trennbar, kann die trennende Ebene auf Basis linearer Funktionen dargestellt werden. Im Umgang mit nicht linear trennbaren, sich überlappenden Klassen gibt es zwei Ansätze. Zum einen wäre es möglich weiterhin eine linear trennende Ebene zu suchen, aber eine gewisse Anzahl falsch klassifizierter Ausreißer zu tolerieren. Dabei wird versucht die Anzahl falscher Klassifizierungen so gering wie möglich zu halten. Der zweite Ansatz ist die Transformation des Problems in einen Raum höherer Dimension. Dazu bedient man sich eines sog. Kernels, der es erlaubt die Transformation nur implizit vorzunehmen. Ziel dieses Tricks sind eine einfachere Berechnung der Hyperebene und ein drastisch reduzierter Rechenaufwand.

Neben dem Einsatz der SVM zur Klassifikation gibt es Ansätze, das Konzept einer Hyperebene zur Approximation einer Funktion $\vec{x} \rightarrow \mathbb{R}$ zu verwenden (Alpaydin, 2008, S. 239f),(Hastie u. a., 2009, S. 436ff). Dabei wird jedoch im Gegensatz zur Vorgehensweise bei der klassischen SVM versucht den Abstand der Trainingsdaten zur Hyperebene zu minimieren. Ein Parameter dieses Verfahrens eine die Distanz zur Hyperebene, innerhalb derer Abweichungen der Trainigsvektoren nicht als Fehler betrachtet werden. Die Wahl dieser Fehlertoleranzgrenze beeinflusst die Robustheit und die Fähigkeit zur Approximation einer Funktion trotz rauschbehafteter Trainingsdaten. Die im Training positionierte Hyperebene stellt die Approximation der gesuchten Funktion dar.

Die Idee der SVM fand in den vergangenen Jahren viele Anhänger. An den Möglichkeiten und Problemen wird geforscht. Gute Resourcen zu SVM stellen Alpaydin (2008); Hastie u. a. (2009) und Smola u. Schölkopf (2009) dar.

4.4 Auswahl des Verfahrens

Das zu wählende maschinelle Lernverfahren soll eine Abbildung von einer Menge boolescher Variablen (Faktor aufgetreten oder nicht) auf einen numerischen Wert (Abhebebetrag) erlernen. Dazu wird es mit den in der Vergangenheit aufgetretenen Kombinationen aus Faktoren und Abhebebetrag trainiert. Das eingesetzte Lernverfahren muss also ein überwachtes und trainierbar sein.

Es wird erwartet, dass die von dem Lernverfahren zu approximierenden Abbildung von Faktoren auf Abhebebetrag über eine hohe Komplexität verfügt. Diese Komplexität in expliziten Regeln zu formulieren scheint alleine bei kombinatorischer Betrachtung der Faktoren aussichtslos. Die Prognose sollte daher ein subsymbolisches Lernverfahren einsetzen.

Die Anforderungen ein überwacht trainierbares und subsymbolisches Lernverfahren zu

sein erfüllen sowohl neuronale Netze als auch die Regression mittels SVM. Eine Wahl alleine auf Grund dieser Kriterien ist schwierig.

Das Ziel der Arbeit ist jedoch nicht die Entwicklung des perfekten Prognoseverfahrens. Das Prognoseverfahren ist nur das ein Hilfsmittel um eines der primären Ziele - eine optimierte Wartungsplanung - zu erreichen. Diese Optimierung funktioniert unabhängig von dem jeweils verwendeten Prognoseverfahren. Weiterhin ist zu berücksichtigen, dass sich Aussagen im Bezug auf das Prognoseverfahren mangels realer nur auf künstlich generierte Daten beziehen. Daher hat die Entscheidung für oder gegen ein Verfahren für das Ergebnis der Arbeit ein geringes Gewicht.

Unter dieser Prämisse wird prototypisch ein Prognoseverfahren implementiert, das auf neuronalen Netzen basiert. Anhand der an das maschinelle Lernverfahren gestellten Kriterien sind SVM und neuronales Netzwerk gleichwertig. Da aber die Regression mittels SVM momentan noch nicht besonders verbreitet und in der Literatur nur vom Ansatz, nicht aber der Anwendung, beschrieben wird, wird diese Möglichkeit nicht weiter verfolgt.

Das neuronale Netz soll die in den generierten Daten abgebildeten Zusammenhänge aus Faktoren und Abhebebetrag approximieren. Dabei soll eine gewisse Toleranz gegenüber dem applizierten Rauschen toleriert werden. Die Umsetzung des Prognosealgorithmus wird in Abschnitt 5.2.2 auf Seite 28 eingegangen. Im Test mit den generierten Daten liefert das auf neuronale Netze setzende Prognoseverfahren zuverlässig die erwarteten Werte. Es ist also für den hier geforderten Einsatzzweck eine gute Wahl.

Im Zuge der Erweiterung der bestehenden Software für Werttransport-Unternehmen sollte das in dieser Arbeit entwickelte Prognoseverfahren mit realen Daten validiert werden, sobald diese vorhanden sind. Stellen sich hierbei Diskrepanzen heraus, sind die erforderlichen Anpassungen vorzunehmen. Es könnte sich auch als nötig erweisen ein anderes Verfahren - zum Beispiel Regression mittels SVM - einzusetzen. Über die Auslagerung des Prognosealgorithmus in ein Modul (vgl. Abschnitt 5.2.1) ist ein solcher Austausch leicht zu vollziehen.

Kapitel 5

Prototyp
Entwurf und Implementierung

Zur Veranschaulichung der dargestellten Ideen für Prognose und Wartungsplanung wird ein Prototyp entwickelt. Er soll außerdem der Firma it kompetenz GmbH helfen Partner für die Weiterentwicklung zu gewinnen.

Der Prototyp wird logisch in Schichten aufgeteilt. Die Datenbank dient der Persistenz, die Anwendungsschicht greift auf die Datenbank zu und stellt der dritten Schicht - der Benutzerschnittstelle - Funktionen und Daten zur Verfügung. Die Benutzerschnittstelle nimmt Benutzereingaben entgegen, zeigt Daten an und löst Aktionen aus. Beim Entwurf wird Wert auf die Verringerung und Entflechtung der Abhängigkeiten gelegt. Dies ist in der Anwendungsentwicklung allgemein ein erstrebenswertes Gut - im vorliegenden Projekt aufgrund der erwarteten Weiterentwicklung und Eingliederung in bestehende Software umso mehr.

Zur Implementierung des Prototypen wird auf Wunsch der Firma it kompetenz GmbH die Programmiersprache C# und das Datenbanksystem Microsoft SQL Server 2008 eingesetzt.

Im ersten Schritt wird die Datenbank modelliert:

5.1 Modellierung der Datenbank

Knotenpunkt der benötigten Daten ist der Geldautomat (Tabelle `Cashpoint` vgl. Abb. 5.1 auf Seite 24). Zwingend erforderlich ist für ihn die Angabe über das maximale Fassungsvermögen (Spalte `MaximumAmount`). Weiterhin wird ein `Name` erfasst, um das Finden und Wiedererkennen des Automaten für den Benutzer zu erleichtern. Die Funktion zur Ermittlung von Wartungsterminen berücksichtigt eine Sicherheitsmarge, die in der Spalte `Security` abgelegt wird. Bei der Integration in die bestehende Software für

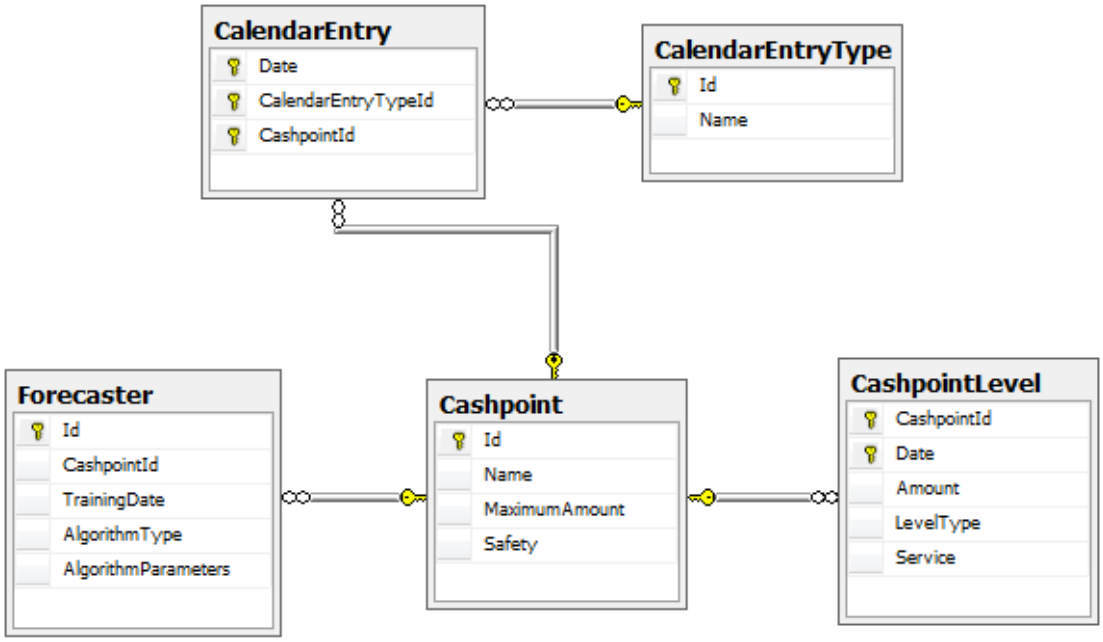

Abbildung 5.1: Datenbankdiagramm

Geld- und Werttransport-Unternehmen könnten dort bereits modellierte Geldautomaten um das `MaximumAmount`- und `Security`-Attribut oder die `Cashpoint`-Entität um die für die bestehende Software benötigten Attribute erweitert werden.

Vergangene und prognostizierte Füllstände der Geldautomaten werden in der Tabelle `CashpointLevel` abgelegt. Das Feld `Amount` enthält den am Datum (Feld `Date`) im Automaten vorhandenen Geldbetrag. Füllstände können auf unterschiedlichem Wege in das System gelangen. Sie können Bestandteil einer Prognose, vom Benutzer eingegeben, importiert oder zu Testzwecken (vgl. Abschnitt 5.4) generiert sein. Abhängig von ihrer Quelle werden sie entweder als fixe Vorgabe oder als bei einem Prognosedurchlauf überschreibbar betrachtet. Die diesbezügliche Kennzeichnung der Füllstände erfolgt im Feld `LevelType`. Weiterhin ist bei einem Füllstand im Feld `Service` vermerkt, ob am betreffenden Tag eine Wartung durchgeführt wurde bzw. durchgeführt werden soll.

Zu jedem Geldautomaten eine Liste von Kalendereinträgen geführt (Tabelle `CalendarEntry`). Ein Kalendereintrag entspricht einem an einem bestimmten Tag für einen Automaten aufgetretenen oder erwarteten Faktoren. Um einmal identifizierte Faktoren nicht bei jeder Bearbeitung des Kalenders neu erfassen zu müssen, werden diese zentral verwaltet und in der Tabelle `CalendarEntryType` abgelegt.

Wie in Abschnitt 3.1 herausgearbeitet, können Prognosenkomponenten trainiert werden. Die hierbei angepassten Parameter werden in der Tabelle `Forecaster` gespeichert. Um dem Prognosealgorithmus eine Entscheidungshilfe zu geben, ob zu einem späteren Zeitpunkt ein erneutes Training fällig ist, wird außerdem das Datum des letzten Trainings gespeichert. Es soll die Möglichkeit geschaffen werden, neben der hier implementierten,

auf künstliche neuronale Netze setzenden, Prognosetechnik andere Verfahren zu integrieren. Zu einer Prognosekomponente wird daher der Klassenname der verwendeten Algorithmusimplementierung in der Spalte `AlgorithmType` abgelegt.

Der Zugriff auf die Datenbank und die Manipulation der dort abgelegten bzw. abzulegenden Daten erfolgt in der Anwendungsschicht.

5.2 Anwendungsschicht

Die Anwendungsschicht stellt der Benutzerschnittstelle Daten und Funktionen zur Verfügung.

Der eigentliche Zugriff auf die Datenbank und die Abbildung der Datensätze auf Objekte wird mit Hilfe des Tools Fluent NHibernate realisiert. Fluent NHibernate benötigt je Datenklasse eine Mappingklasse. Sie spezifiziert eine Abbildung von Klassen und Attributen auf Datenbanktabllen und -spalten. Listing 5.1 zeigt beispielhaft die Mappingklasse für die Prognosekomponenten. Sie leitet sich von der `ClassMap`-Klasse ab. Im Constructor findet die Abbildung statt. `Id(..)` definiert den Primärschlüssel, `Map(..)` weist Attributen Spalten zu, `References(..)` konfiguriert die Eigenschaft `Cashpoint` so, dass sie ein anhand der Spalte `CashpointId` identifiziertes Cashpoint-Objekt referenziert.

```
public sealed class ForecasterMap : ClassMap<Forecaster>
{
    public ForecasterMap()
    {
        Id(x => x.Id);
        Map(x => x.AlgorithmParameters);
        Map(x => x.AlgorithmType);
        Map(x => x.TrainingDate);
        References(x => x.Cashpoint).Column("CashpointId")
                                    .ForeignKey();
    }
}
```

Listing 5.1: Beispiel: Mappingklasse

Die Verbindung zur Datenbank wird jeweils nur für die Dauer eines lesenden oder schreibenden Zugriffs auf die Datenbank geöffnet. Fluent NHibernate unterstützt diese Konzept durch die Bereitstellung von Sessionobjekten. In der Klasse `SessionFactory` wird Fluent NHibernate konfiguriert (u.a. Setzen von Datenbank- und Servername). Die Klasse stellt weiterhin eine einfache Möglichkeit bereit eine neue Session zu öffnen. Listing 5.2 zeigt beispielhaft, wie in dem ein neuer Geldautomat erstellt, eine Session geöffnet und das Geldautomat-Objekt gespeichert wird.

```
var cashpoint = new Cashpoint();
cashpoint.Name = "Test";
cashpoint.MaximumAmount = 750000;
cashpoint.Safety = 5000;

using(var session = SessionFactory.OpenSession())
```

```
{
    session.SaveOrUpdate(cashpoint);
}
```

Listing 5.2: Bespiel: Verwendung von SessionFactory und Session

Der `using`-Block ist eine Spezialität der C#-Sprache. Das Argument (`var session = ..`) wird beim Eintritt erstellt und beim Verlassen des Blocks gelöscht. Das Sessionobjekt sorgt bei seiner Zerstörung dafür, dass eventuell noch nicht geschriebene Änderungen in die Datenbank befördern werden und die Datenbankverbindung dann geschlossen wird.

Zu den zu speichernden Daten werden die passenden Datenklassen erstellt. Dies sind der Geldautomat (`Cashpoint`), dessen Füllstände (`CashpointLevel`), Prognosekomponenten (`Forecaster`) und Kalendereinträge (`CalendarEntry`). Wie die Klassennamen bereits vermuten lassen, werden die Datenfelder der Klassen auf die gleichnamigen Tabellen abgebildet. Abb. C.1 auf Seite IX zeigt die Datenklassen mit ihren Attributen, so wie ihren Assoziationen.

Die Berechnung der Wartungskosten benötigt die jeweils aktuellen Kosten für eine Wartung sowie den Zinssatz. In der momentanen Implementierung sind die Pauschale für eine Wartung und die Zinskosten über die Zeit gesehen konstant. Da aber Schwankungen der Wartungspauschale und des Zinskurs Realität sind, wird mit der Klasse `CostRates` eine Möglichkeit geschaffen, in Abhängigkeit von einem Datum die jeweils aktuellen Kostenfaktoren zu verwenden. Die Klasse `CostRates` könnte beispielsweise den Zinskurs aus einer Onlinequelle (z.B. Webservice einer Bank) und die Wartungspauschale aus der Datenbank des Werttransport-Unternehmens holen und der Kostenberechnung bereitstellen.

Der Algorithmus zur Prognose zukünftiger Abhebevorgänge soll austauschbar sein. Aus diesem Grund wird in der Klasse `Forecaster` nur auf eine Implementierung der `IAlgorithm`-Schnittstelle (vgl. Anhang D) verwiesen. Die Bedeutung der Schnittstellen und der Mechanismus zum Laden einer Implementierung wird im folgenden Abschnitt erläutert.

5.2.1 Anatomie der Prognosekomponente

Prognoseobjekte werden pro Geldautomat erstellt (siehe Abschnitt 3.1.2). Sie werden - sofern der Algorithmus dies erfordert - mit vergangenen Füllständen und den Einträgen des Kalenders trainiert. Resultate des Trainings, beispielsweise ein trainiertes neuronales Netz, werden in der Datenbank gespeichert.

Ein Prognoseobjekt besteht aus zwei Teilen, aus einem `Forecaster`-Objekt und einem Prognosealgorithmus. Ersteres stellt die Assoziation mit einem Geldautomaten her, lädt einen Algorithmus, ermöglicht die Speicherung der Algorithmusparameter und reicht Funktionsaufrufe an den Algorithmus weiter. Letzterer wird zur Laufzeit mit Hilfe von MEF

aus im Anwendungsverzeichnis abgelegten Klassenbibliotheken geladen. Der verwendet Algorithmus wird über seinen Klassennamen identifiziert.

Der Prognosealgorithmus übernimmt die Aufgabe, aus einer Menge von Kalendereinträgen einen Abhebebetrag zu prognostizieren (Methode: `ForecastWithdrawal`). Benötigt der Algorithmus vorheriges Training, so kann dies über den Rückgabewert der Funktion `NeedsTraining` signalisiert werden. Ein Trainingslauf wird bei Bedarf über `RunTraining` angestoßen. Wurde ein Prognosealgorithmus bereits trainiert, so können die Resultate (Parametereinstellungen etc..) in serialisierter Form über die Funktion `SerializeParameters` übergeben werden. Sie werden dann gespeichert und bei erneuter Durchführung einer Prognose wiederhergestellt und an den Prognosealgorithmus über die Funktion `LoadParameters` übergeben.

Die vom Prognoseobjekt gelieferten Abhebebeträge werden von dem Kostenoptimierungsverfahren genutzt um Wartungen zu planen.

Damit die Algorithmusimplementierung von MEF geladen werden kann, müssen einige Voraussetzungen erfüllt werden.

Fabrikklasse

Je Algorithmus muss eine Fabrikklasse bereitgestellt werden. Fabrikklassen müssen ein `AlgorithmInformation`-Attribut haben, die Schnittstelle `IAlgorithmFactory` implementieren und diese Schnittstelle für MEF sichbar machen („exportieren“). Listing 5.3 zeigt hierfür ein Beispiel.

- Das `Export`-Attribut signalisiert MEF, dass die Klasse eine Schnittstelle - `IAlgorithmFactory` - exportiert.
- Das `AlgorithmInformation`-Attribut liefert Metadaten zur Beschreibung des bereitgestellten Algorithmus.
- Die einzige Funktion der Fabrikklasse - `CreateAlgorithm` - erstellt eine neue Instanz des Algorithmus.

```
[Export(typeof(IAlgorithmFactory))]
[AlgorithmInformation(Description = "A sample forecast algorithm",
                      Type = typeof(DummyAlgorithm))]
public class DummyFactory : IAlgorithmFactory
{
    public IAlgorithm CreateAlgorithm()
    {
        return new DummyAlgorithm();
    }
}
```

Listing 5.3: Beispiel: Fabrikklasse

Algorithmus

Der eigentliche Prognosealgorithmus wird in einer Klasse realisiert, die das Interface `IAlgorithm` implementiert. Hier findet das Training und das Prognostizieren der Abhebebeträge statt. Die Klasse in Listing 5.4 implementiert die nötigen Funktionen und deutet an, was in ihnen jeweils geschehen soll.

```
public class DummyAlgorithm : IAlgorithm
{
    public void LoadParameters(string parameters)
    {
        // "parameters" deserialisieren und übernehmen
    }

    public string SerializeParameters()
    {
        string parameters = parameterSerialisierung();
        return parameters;
    }

    public bool NeedsTraining(Cashpoint cashpoint, DateTime? lastTraining)
    {
        bool needsTraining = parameterPruefung(cashpoint, lastTraining);
        return needsTraining;
    }

    public void RunTraining(Cashpoint cashpoint)
    {
        /* Algorithmus mit den in "cashpoint" vorhandenen Füllständen
         * und Kalendereinträgen trainineren, Resultate so
         * aufbereiten und ablegen, dass sie über "SerializeParameters()"
         * zum Speichern abgeholt werden können
         **/
    }

    public int ForecastWithdrawal(Cashpoint cashpoint,
                IEnumerable<CalendarEntryType> entryTypes)
    {
        int withdrawal = prognoseAlgorithmus(cashpoint, entryTypes);
        return withdrawal;
    }
}
```

Listing 5.4: Beispiel: Algorithmus-Implementierung

5.2.2 Prognosealgorithmus Basiernd auf künstlichen neuronalen Netzen

Eingabedaten für den Prognosealgorithmus sind die gesetzten Faktoren im Kalender. Als Ausgabe wird ein zu der übergebenen Faktorkonstellation passender Abhebebetrag erwartet, eine Zahl zwischen Null und dem Fassungsvermögen des Geldautomaten. Es muss also eine trainierbare Funktion realisiert werden, die eine Menge von booleschen Variablen (Faktor gesetzt oder nicht gesetzt) auf eine Zahl abbildet. Diese Aufgabe soll ein künstliches neuronales Netz erfüllen. Auf die Gründe für diese Wahl wird in Abschnitt

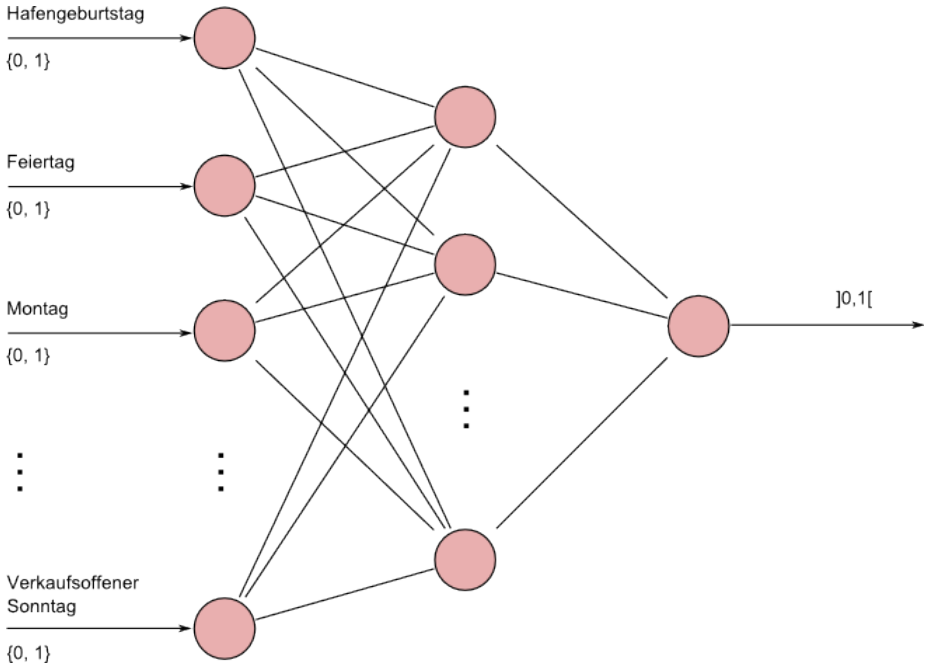

Abbildung 5.2: Schematische Darstellung des Prognosenetzwerks

4.4 auf Seite 21, auf Funktionsweise und Ausprägungen neuronaler Netze in Abschnitt 4.2 ab Seite 17 eingegangen.

Es wird ein vorwärtsgekoppeltes Netz verwendet. Die Anzahl der Eingangsneuronen entspricht der Anzahl verfügbarer Faktoren. Eine verdeckte Schicht (hidden layer) mit ebenso vielen Neuronen hat sich als gute Wahl herausgestellt. Die Ausgabeschicht besteht aus einem einzelnen Neuron (vgl. Abb. 5.2). Als Aktivierungsfunktion wird die logistische Funktion gewählt. Sie ist differenzierbar (wichtig für das Training) und hat als Wertebereich das offene Intervall $]0,1[$. Der vom Ausgangsneuron gelieferte Wert aus diesem Intervall wird auf das geforderte Intervall (Zwischen Null und Fassungsvermögen des Geldautomaten) skaliert.

Zu der zu lösenden Aufgabe liegen sowohl die Eingaben (Kalendereinträge) als auch die erwarteten Ausgaben (Abhebebeträge) vor. Daher bietet es sich an das Netz mit einem überwachten Lernverfahren zu trainieren. Um das Training ausführen zu können, müssen die Füllstände zunächst in Abhebebeträge umgerechnet werden. Dazu wird jeweils die Differenz zweier auf einander folgender Füllstände gebildet. Der Abhebebetrag A eines Tages d_i ist die Differenz aus dem Füllstand F an vorherigen Tag d_{i-1} und dem am Tag d_i.

$$A(d_i) = F(d_{i-1}) - F(d_i) \tag{5.1}$$

An Tagen, an denen eine Wartung durchgeführt wurde, kann dieser Abhebebetrag nicht ermittelt werden. Die Veränderung des Füllstandes gegenüber dem Vortag stehen in diesem Fall in keinem Zusammenhang mit den zu erlernenden Einflüssen. Daher müssen

skalierter Betrag	Faktoren
$0,0142$	{Montag, Wochenmarkt}
$0,0091$	{Dienstag}
$0,0087$	{Mittwoch, Schulferien}
$0,0164$	{Donnerstag, Schulferien, Kinopremiere}
$\vdots$	$\vdots$

Tabelle 5.1: Beispiel: Trainingsdaten

diese Tage aus den Trainingsdaten ausgefiltert werden. Das gleiche gilt für Füllstände, die das Resultat einer früheren Prognose sind.

Das neuronale Netz liefert wie oben erwähnt als Ausgabe lediglich einen Zahlenwert aus dem Intervall $]0,1[$ die Abhebebeträge können also nicht direkt als erwartete Ausgaben zum Training verwendet werden. Sie werden skaliert, indem sie durch das Fassungsvermögen des Geldautomaten dividiert werden. Die Genauigkeit des Datentyps des Ausgabewertes des neuronalen Netzes lässt dies ohne relevante Verluste zu. Das Fassungsvermögen des Geldautomaten eignet sich gut als Referenzgröße, da dies baulich bedingt ist und sich über die Lebensdauer des Automaten nicht ändert.

Die Zuordnung der aufgetretenen Faktoren zu den skalierten Abhebebeträgen komplettiert die Trainingsdaten. Sie könnten dann wie in Tabelle 5.1 angedeutet aussehen.

Als Trainingsverfahren wird Backpropagation eingesetzt. Die Implementierung des Trainingsverfahrens sowie des neuronalen Netzes wird nicht eigenhändig neu entwickelt, sondern aus dem AForge.NET-Framework verwendet. Es standen weitere Implementierungen (Beitrag bei CodeProject[2], fann[3], encog[4], JavaNNS[5]) zur Auswahl. Die Wahl fiel auf das AForge.NET-Framework, da es leicht zu integrieren und weit entwickelt (stabil) ist, die benötigten Funktionalitäten liefert und zudem kostenlos inklusive Quellcode verfügbar ist.

Instanziierung und Training des Netzes

Das künstliche neuronale Netz wird bei jedem Tainingslauf neu erstellt. Dies ist notwendig, da neue Faktoren in das System aufgenommen worden sein könnten und eine Wiederverwendung eines bestehenden Netzes neben einer geringen Einsparung an Rechenzeit keine Vorteile bietet.

Der folgende Quellcodeausschnitt zeigt die Instanziierung des Netzes. Es wird ein Netz mit den oben genannten Dimensionen und einer sigmoiden Aktivierungsfunktion erstellt. Die Gewichte der Neuronenverbindungen werden mit zufälligen Werten vorbelegt um

[2] `http://www.codeproject.com/KB/cs/BackPropagationNeuralNet.aspx`
[3] `http://leenissen.dk/fann/`
[4] `http://code.google.com/p/encog-cs/`
[5] `http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/welcome_e.html`

das von Zell (2003, S. 110f) beschriebene *symmetry breaking* Problem zu umgehen. Es besagt, dass das Training mittels Backpropagation bei Initialisierung der Gewichte einer Schicht im neuronalen Netz mit gleich großen Werten fehlschlagen könnte, da sich keine Unterschiede in der Belegung der Gewichte herausbilden.

```
int inputNeurons = entryTypes.Count;
int hiddenNeurons = entryTypes.Count;
int outputNeurons = 1;
ActivationNetwork network = new ActivationNetwork(
                new SigmoidFunction(),
                inputNeurons,
                new[] { hiddenNeurons, outputNeurons });

network.Randomize();
```

Listing 5.5: Instanziierung des Netzes

Die Trainingsdaten werden in zufälliger Reihenfolge in zwei Mengen aufgeteilt. Zwei Drittel der Daten werden zum eigentlichen Trainieren mittels Backpropagation, das verbleibende Drittel zum Validieren des Netzzustandes im Laufe des Trainings eingesetzt. Das Training wird solange fortgesetzt, bis die bei der Validierung ermittelte Genauigkeit ausreichend ist oder eine gewisse Anzahl Iterationen überschritten ist. Stellt die Validierung einen hohen Fehler fest, werden die Gewichte des Netzes zufällig neu gewählt. Dies führt schneller zu besser trainierten Netzen, als ein längeres Training eines „schlecht“ initialisierten Netzes. Neben dem hohen Zeitbedarf birgt ein zu langes Training weiterhin die Gefahr der Überanpassung des Netzes - also ein Auswendiglernen der Trainingsdaten.

Das trainierte Netz wird anschließend serialisiert und in einer Variable zwischengespeichert. Dieser Wert wird über die `SerializeParameters`-Funktion der `IAlgorithm`-Schnittstelle abgefragt und in der Datenbank abgelegt. Wird die Prognosekomponente zu einem späteren Zeitpunkt erneut benötigt, wird das serialisierte Netz über die `LoadParameters`-Funktion an den Algorithmus übergeben und kann dort wieder in das trainierte Netz deserialisiert werden.

```
BackPropagationLearning trainer = new BackPropagationLearning(network);
bool terminate = false;
int iteration = -1, iterations = 3000;

while(!terminate) {
  trainer.RunEpoch(trainingInput, trainingOutput);
  double error = ValidateNetwork(network, validateInput, validateOutput);
  if(error > 0.7) {
    network.Randomize();
  }
  terminate = error < 0.005 || iteration > iterations;
  iteration++;
}
```

Listing 5.6: Training des Netzes

5.2.3 Wartungsplanung und Kostenoptimierung

Die Generierung des Wartungsplans wird vom Benutzer angestoßen. Hierbei gibt er das Enddatum des zu erstellenden Plans an und für welche Geldautomaten dies durchgeführt werden soll.

Das Planungsverfahren erstellt nun für die gewählten Geldautomaten ausgehend von einem Ausgangstag den Wartungsplan. Der Ausgangstag ist der letzte Tag für den ein realer (also vom Benutzer erfasster) Füllstand vorhanden ist. Ausgehend von diesem Tag und Füllstand wird in einer **ersten Phase** das Datum der ersten zu planenden Wartung ermittelt. Dies geschieht, indem für den jeweils folgenden Tag der Abhebebetrag prognostiziert und der Füllstand um diesen Betrag verringert wird. Die hierbei generierten Füllstandsprognosen werden über das `LevelType`-Attribut gekennzeichnet und in die Liste der Füllstände des Geldautomaten eingetragen. Fällig ist die Wartung, wenn der im Automaten verbleibende Geldbestand den für den folgenden Tag prognostizierten Abhebebetrag oder die Sicherheitsmarge des Geldautomaten unterschreitet.

Ausgehend von dem ersten Wartungstermin w_i werden in der **zweiten Phase** der nötige Füllbetrag und der folgende Wartungstermin w_{i+1} ermittelt. Hierzu wird die in Abschnitt 3.3.2 entwickelte Funktion $K_r(w_i, w_{i+1})$ eingesetzt. Sie liefert für jede Kombination aus w_i und Tag d die durchschnittlich pro Tag anfallenden Kosten. Als w_{i+1} wird nun der Tag d gewählt, für den die Kosten minimal sind.

Die prognostizierten Füllstände sowie der für w_i ermittelte Auffüllbetrag werden in die Liste der Füllstände des Geldautomaten eingetragen. Sie werden über das `LevelType`-Attribut als Prognose, der Füllstand am Tag w_i zusätzlich als Wartung, gekennzeichnet.

Solange w_{i+1} noch vor dem vom Benutzer angegebenen Enddatum der Planung liegt, wird die zweite Phase mit w_{i+1} als Startwert erneut durchlaufen.

Behandlung manuell geplanter Wartungen

Stößt die Planungsfunktion auf eine vom Benutzer vorgegebene Wartung, wird die Suche nach möglicherweise kostengünstigeren Terminen abgebrochen und der vorgegebene Termin verwendet. Sind Termin und Füllbetrag vorgegeben, läuft das Verfahren bei der ersten Phase mit vorgegebenem Wartungstermin und Füllbetrag als Ausgangswerten weiter. Ist lediglich der Termin festgelegt, wird der optimale Füllbetrag in einem Durchlauf der zweiten Phase ermittelt.

5.3 Benutzeroberfläche

Das Graphical User Interface (GUI) soll die von der Anwendungsschicht gelieferten Daten für den Benutzer darstellen und die Manipulation derselben ermöglichen. Um das

GUI sinnvoll zu gestalten werden zunächst die darzustellenden Daten in einem Entwurf skizziert (vgl. Abb. 5.3). Zentrale Rolle bei der Interaktion mit dem Benutzer spielt ein Kalender. Er stellt in den Zeilen Geldautomaten, in den Spalten Tag und die hier gesetzten Faktoren und Füllstände dar. So wird der Bezug zwischen Geldautomaten, seinen Füllständen, dem Auftreten von Faktoren und dem Datum hergestellt. Über oben angeordnete Menüeinträge und Toolbars sollen weitere Aktionen wie das Anlegen eines Geldautomaten oder die Wartungsplanung ausgelöst werden können.

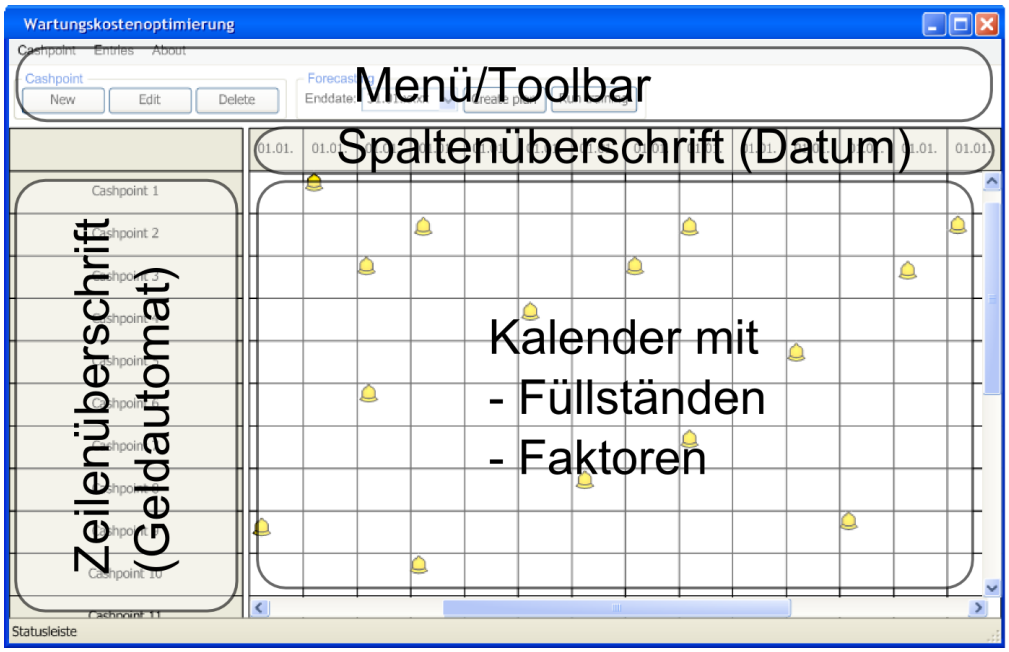

Abbildung 5.3: Prototyp: Benutzeroberfläche (Hauptfenster)

Die eigentliche Benutzerschnittstelle wird mittels Windows Presentation Foundation (WPF) - einem im .NET-Framework enthaltenen Grafik-Framework - erstellt. Das endgültige Aussehen des Hauptfensters des GUIs ist in Abbildung 5.4 zu sehen. Über die linke Spalte lassen sich Geldautomaten für eine Aktion - beispielsweise die Durchführung eines Trainingslaufs - selektieren. Zu ausgewählten Geldautomaten gehörende Zeilen werden in ihrer Höhe vergrößert und farblich hinterlegt, um die Selektion für den Benutzer leichter ersichtlich zu machen.

In einer Navigationstoolbar (vgl. Abb. 5.5 (links)) kann der Benutzer den zu zeigenden Ausschnitt des Kalenders bestimmen. Entweder, durch das Betätigen der Schaltflächen zum Blättern (1) oder durch Wahl des Start- oder Enddatums (2) in einem Popup (3).

Mit den Schaltflächen der Cashpoint-Toolbar kann der Benutzer Geldautomaten anlegen, bearbeiten und löschen. Für das Anlegen und Bearbeiten wird ein Dialog (vgl. Abb. 5.6 (links)) verwendet, der die Eingabe des Namens, des Fassungsvermögen und der Sicherheitsmarge des anzulegenden oder zu bearbeitenden Geldautomaten erlaubt. Über die Schaltfläche Save werden Eingaben gespeichert, über Reset die noch ungespeicherten Änderungen zurückgenommen.

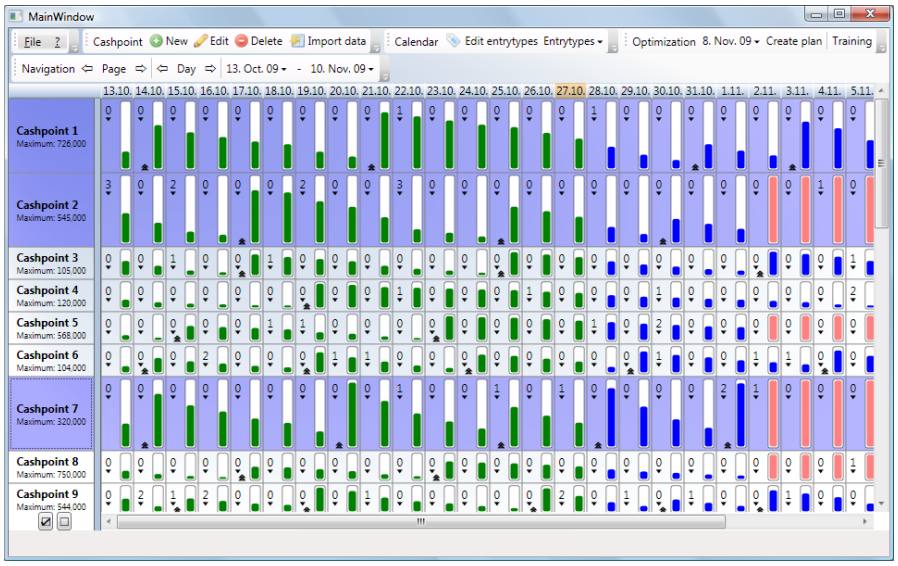

Abbildung 5.4: Benutzeroberfläche (Hauptfenster)

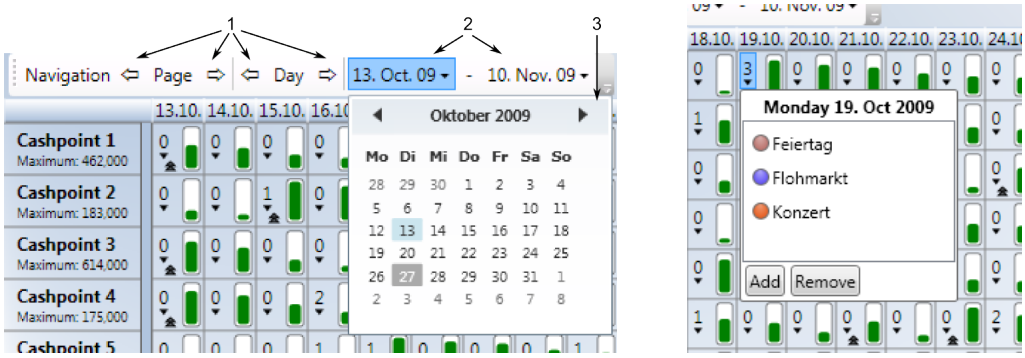

Abbildung 5.5: Navigationsleiste (links) und Popup zum Setzen von Faktoren (rechts)

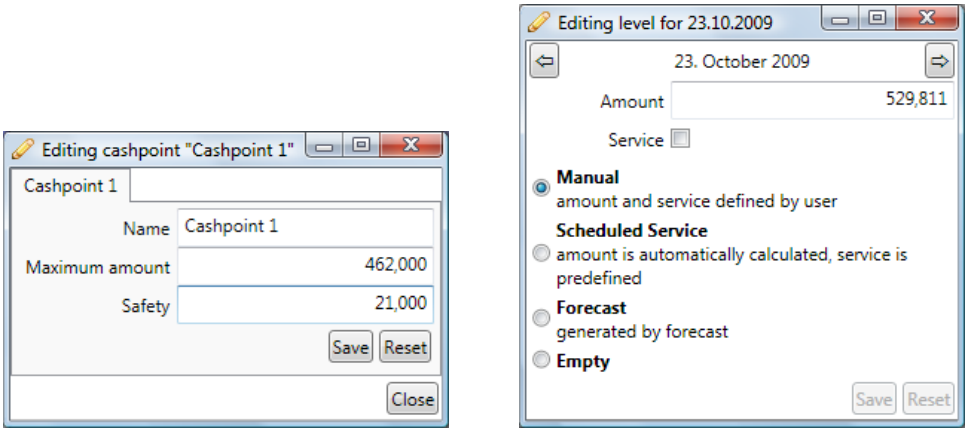

Abbildung 5.6: Dialoge zum Bearbeiten der Attribute eines Geldautomaten (links) und des Füllstandes (rechts)

Eine Zelle der Kalendertabelle zeigt am rechten Rand den Füllstand. Die Farbe wird dabei abhängig vom Typ des Füllstandes (prognostiziert oder real) gewählt. Prognosen sind blau, reale Daten grün und Tage ohne erfasste Füllstände rot hinterlegt. Am unteren Rand zeigt ein Symbol ▲ an, wenn am Tag eine Wartung geplant ist. Weiterhin zeigt eine Zahl in der linken oberen Ecke die Anzahl für diesen Tag eingetragener Faktoren an. Ein Mausdruck auf diese Zahl öffnet ein Unterfenster in dem Faktoren gesetzt oder entfernt werden können (vgl. Abb. 5.5 (rechts)).

Per Doppelklick auf eine Kalenderzelle, wird ein Dialog zur Bearbeitung des Füllstandes geöffnet (vgl. Abb. 5.6 (rechts)). Die erste Zeile zeigt das Datum des aktuell bearbeiteten Füllstandes an und enthält Schaltflächen um einen früheren oder folgenden Füllstand zu bearbeiten, ohne den Dialog zu verlassen. Dem Füllstand des bearbeiteten Tages kann einer der folgenden Modi zugewiesen werden:

Manual Sowohl der Betrag als auch die Angabe, ob eine Wartung stattfindet, sind extern vorgegeben. Dies bedeutet entweder, dass es sich um einen vergangenen Tag handelt und die Werte den vom Betreiber übermittelten Daten entstammen oder, dass eine Wartung manuell vorgegeben ist.

Scheduled Service Für den ausgewählten Tag ist eine Wartung vorgesehen. Der Füllbetrag soll jedoch automatisch ermittelt werden. Bei Auswahl dieser Option wird das `Service` Attribut des `CashpointLevel`-Objekts auf `true` gesetzt.

Forecast Betrag und die Entscheidung, ob eine Wartung durchgeführt werden soll, werden vom Planungsverfahren festgelegt.

Empty Weder Betrag noch eine Angabe über eine eventuelle Wartung sind gesetzt. Diese Option entspricht dem Löschen des betreffenden `CashpointLevel`-Objekts.

Befindet sich das `CashpointLevel`-Objekt im „Manual“-Modus, erlaubt der Dialog die Eingabe von des am betreffenden Tag im Automaten vorhandenen Geldbetrags und die Angabe, ob eine Wartung stattgefunden hat oder stattfinden wird. Andernfalls werden diese Werte - soweit gesetzt - nur angezeigt. Änderungen am Modus bewirken, dass - wenn nötig - ein `CashpointLevel`-Objekt angelegt oder gelöscht wird und die passenden Werten für `LevelType`, `Amount` und `Service` gesetzt werden. Wie im Dialog zum Bearbeiten eines Geldautomaten sind die Save und Reset Schaltflächen vorhanden, um Änderungen zu speichern oder rückgängig zu machen.

Die Schaltflächen der Optimization-Toolbar im Hauptfenster stoßen für die ausgewählten Geldautomaten die Erstellung eines Wartungsplans und das Training der Prognosekomponenten an. Per Dropdown wählt der Benutzer das Datum aus, bis zu dem der Plan reichen soll und startet das Erstellen des Plans für die selektierten Geldautomaten mit der Schaltfläche Create plan. Die Training-Schaltfläche erstellt und trainiert Prognosekomponenten für die ausgewählten Geldautomaten. Die Aktionen Planen und Trainieren können je nach Anzahl der gewählten Geldautomaten einige Sekunden in Anspruch nehmen. Um dem Benutzer währenddessen das Weiterarbeiten zu ermöglichen

und Rückmeldungen über den Fortschritt zu geben, werden sie im Hintergrund ausgeführt. Statusmeldungen und der aktuelle Fortschritt werden in einer Leiste am unteren Rand des Hauptfensters ausgegeben (vgl. Abb. 5.7).

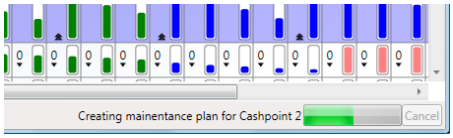

Abbildung 5.7: Statusanzeige

5.4 Generierung von Testdaten

Entwicklung und Test macht es erforderlich über Testdaten - also Geldautomaten, Faktoren und Füllstände - zu verfügen. Reale Daten stehen nicht zur Verfügung (siehe Abschnitt 3.1.1). Stattdessen werden Testdaten durch ein eigenständiges Programm generiert. Diese Anwendung implementiert den Anwendungsfall in Anhang B.3. Da sie nur für Entwicklung und Test verwendet wird, kommt die Anwendung ohne grafische Oberfläche aus. Im späteren eigentlichen Einsatz werden die Daten vom Benutzer eingegeben, aus Dateien importiert oder von anderen Anwendungen in der Datenbank abgelegt.

Wie im Anwendungsfall beschrieben werden zunächst die nötigen Parameter vom Benutzer abgefragt:

Leeren der Datenbank Im Zuge von Test und Entwicklung kann es hilfreich sein, in einer definierten, „sauberen“ Umgebung zu arbeiten. Zu diesem Zweck wird dem Benutzer angeboten die Datenbank vor dem Einfügen neuer Geldautomaten zu leeren.

Geldautomaten: Anzahl Um keine bereits erfassten Daten zu überschreiben, legt das Programm neue Geldautomaten an, die mit generierten Füllständen und Kalendereinträgen versehen werden.

Historie: Anzahl der Monate Hauptzweck der Anwendung ist das Generieren von Füllständen und Kalendereinträge. Dies wird für die hier eingegebene Anzahl Monate m durchgeführt.

Zukunft: Anzahl der Tag Um den Umgang des Prognoseverfahrens mit dem Auftreten von Faktoren testen zu können, werden für die angegebene Anzahl Tage in die Zukunft zufällig Kalendereinträge für die neuen Geldautomaten generiert.

Nach der Bestätigung der Eingaben durch den Benutzer führt das Programm die gewünschten Aktionen durch.

Hat der Benutzer dies ausgewählt, wird zunächst die Datenbank geleert.

Das Programm erstellt und speichert die gewünschte Anzahl neuer Geldautomaten. Das Fassungsvermögen der neuen Geldautomaten beträgt dabei jeweils einen zufälligen Wert aus dem Intervall $[10.000, 800.000]$. Anschließend werden nacheindander für die Geldautomaten Daten generiert.

Für die vergangenen m Monate werden pro Tag zufällig eine beliebige Anzahl Kalendereinträge gemacht. Es wird eine Tabelle T angelegt, die jeder auftretenden Kombination aus Faktoren einen zufällig gewählten Abhebebetrag zuordnet. Ausgehend von einem Anfangsfüllstand F_a generiert das Programm anschließend Füllstände. Der Füllstand für einen Tag d_i wird aus dem des Vortages $F(d_{i-1})$ und dem zu den aufgetretenden Kalendereinträgen $E(d_i)$ passenden Abhebebetrag aus der Tabelle berechnet (vgl. Formel 5.3). Der Abhebebetrag wird mit einem Rauschen r behaftet, um die Komplexität der generierten Daten zu erhöhen.

$$F(d_0) = F_a \tag{5.2}$$

$$F(d_i) = F(d_{i-1}) - T(E(d_i)) \cdot (1 + r) \tag{5.3}$$

Sinkt der Füllstand auf null, wird ein Wartungstermin generiert, der den Automaten auf einen zufälligen Wert zwischen dem halben und vollen Fassungsvermögen auffüllt.

Sind die geforderten Daten generiert, teilt das Programm dieses dem Benutzer mit und kann beendet werden.

Kapitel 6

Fazit

Das Einsparungen möglich sind, zeigt die Betrachtung der idealisierten Kostenentwicklung im Bezug auf die Länge einer Wartungsperiode (vgl. Abb. 3.2). Ist die Periode zu kurz gewählt, überwiegt die Pauschale für die Befüllung, ist sie zu lang gewählt, entstehen zu hohe Zinsverluste. Durch die Wahl des Wartungstermins im Minimum der pro Tag anfallenden Kosten werden unnötige Kosten vermieden und die möglichen Einspareffekte genutzt.

Ziel der Arbeit war es unter anderem ein Verfahren zu entwickeln, das Zinsverluste und Kosten für Wartungen reduziert. Dazu wurde ein Wartungsplanungsmodul realisiert, das einen Wartungsplan so aufstellt, dass die pro Tag anfallenden Kosten minimal sind. Das Ziel der Verringerung der Kosten ist also erreicht.

Das Planungsverfahren ist auf prognostizierte Abhebebeträge angewiesen. Diese werden momentan von einer nur mit generierten Daten getesteten Komponente geliefert. Im Umgang mit realen Daten könnte sich diese Prognosekomponente als unzureichend oder ein anderes maschinelles Lernverfahren als treffender erweisen. In diesem Fall muss das Prognoseverfahren angepasst oder ein neuer Prognosealgorithmus entworfen werden, der die gewünschte Genauigkeit und Zuverlässigkeit liefert. Das Planungsverfahren an sich ist davon jedoch unberührt, da es - gute Prognosen vorausgesetzt - einen optimalen Wartungsplan entwirft.

Es wurde eine prototypische Implementierung erstellt, die in ihrer jetzigen Form von der Firma it kompetenz GmbH zur Demonstration des Konzepts bei potentiellen Partnern genutzt werden kann. Außerdem stellt dieser Prototyp aufgrund seiner Flexibilität eine gute Ausgangsbasis für eine Weiterentwicklung zu einem Produkt oder zur Integration in die bestehende Software für Geldtransport-Unternehmen dar.

Kapitel 7

Glossar

AForge.NET Eine Sammlung - in C# geschriebener - Bibliotheken für Entwicklung und Forschung im Bereich künstlicher Intelligenz und künstlichen Sehens. Unter anderem stellt das AForge.NET Framework eine Implementierung künstlicher neuronaler Netze bereit. Diese wird in der vorliegenden Arbeit für die Prognose verwendet. Weitere Informationen unter (AForge.NET, 2009). 23, 24

C# Eine von Microsoft im Rahmen ihres .NET-Frameworks entwickelte Programmiersprache. Syntaktisch an Java und C angelegt.. 16

Faktor Bezeichnet im Zusammenhang dieser Arbeit Tatsachen, die einen Einfluss auf das Abhebeverhalten eines Geldautomaten haben. Unterschieden wird dabei zwischen statischen Faktoren (z.B. Standort) und dynamischen (z.B. Feiertag). 8

Fluent NHibernate Eine Erweiterung zu NHibernate. Erleichtert das Erstellen des Mappings von Klassen auf relationale Entitäten. Laut (Fluent NHibernate, 2009):

> Fluent NHibernate offers an alternative to NHibernate's standard XML mapping files. Rather than writing XML documents, you write mappings in strongly typed C# code. This allows for easy refactoring, improved readability and more concise code.

Näheres in Abschnitt **??**.. II

GUI Graphical User Interface. 27

Mappingklasse Fluent NHibernate verwendet statt der in NHibernate üblichen Xml-Dokumente Klassen - die Mappingklassen - zur Zuweisung von Klassenattributen zu Tabellenspalten. Eine Mappingklasse wird von der generischen `ClassMap<T>`-Klasse abgeleitet, wobei `T` der Typ der abzubildenden Klasse ist. Ein Beispiel ist in Listing 5.1 auf Seite 25 zu finden. 18

MEF Managed Extensibility Framework. II

Serialisierung Laut (Wikipedia, 2009)

> [..] Abbildung von Objekten auf eine externe sequenzielle Darstellungsform [..] für das Erreichen von Persistenz [..]

Die Umkehrung der Serialisierung ist die Deserialisierung. Sie erstellt ein neues Objekt, das dem ursprünglichen Objekt wertgleich ist. Identisch zum serialisierten ist dieses neue Objekt nicht, da es sich um eine neue Instanz mit gleichen Werten, aber eben nicht um das ursprüngliche Objekt selber handelt.. VIII, 20, 25

WPF Windows Presentation Foundation. 27

Anhang A

Anwendungsfalldiagramm

Aus den Spezifikationen der Anwendung ergibt sich das in Abbildung A.1 dargestellte Anwendungsfalldiagramm. Es soll die Beziehungen einiger zentraler Anwendungsfälle anschaulich machen. Die mit „«uses»“ gekennzeichneten Pfeile sollen andeuten, dass ein Anwendungsfall (Anfang des Pfeils) einen anderen Anwendungsfall benutzt (Pfeilspitze). Nähere Erläuterungen zu den hier gezeigten Anwendungsfällen befinden sich in Anhang B.

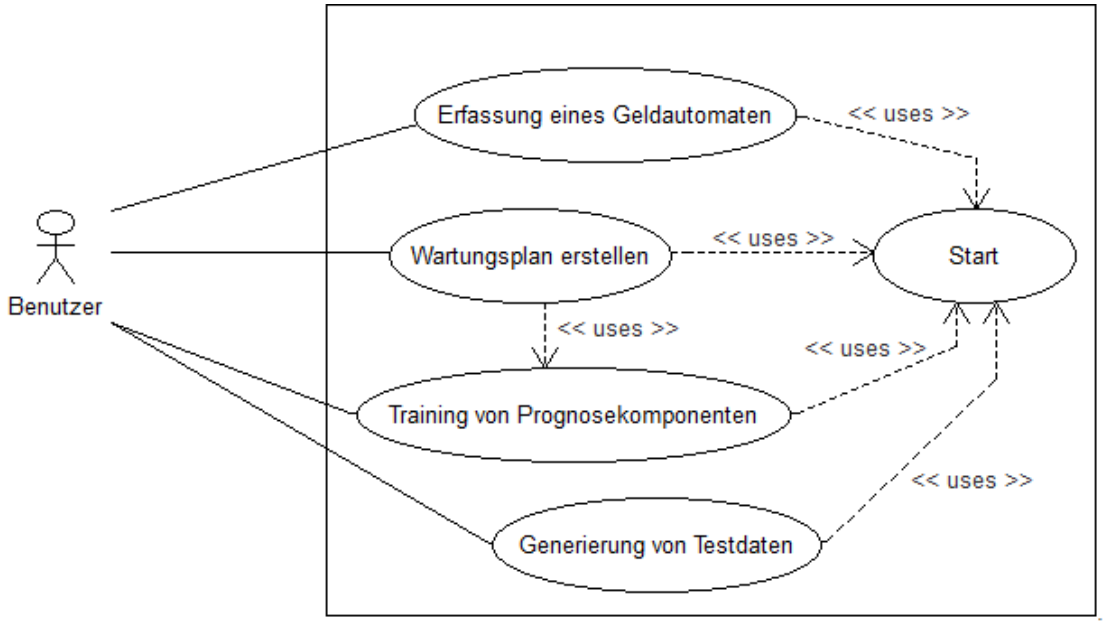

Abbildung A.1: Anwendungsfalldiagramm

Anhang B

Anwendungsfälle

Zu den nachfolgend aufgeführten Anwendungsfällen werden neben dem typischen Ablauf alternative Abläufe aufgezeigt. Auf die Angabe eines Autors wird verzichtet, da sich dieser bereits durch die Bachelorarbeit ergibt. Die Darstellung der Anwendungsfälle ist teilweise vereinfacht und dient der exemplarischen Spezifizierung der Anwendungsfunktionalitäten.

B.1 Start

Akteure

- Anwender der Wartungskostenoptimierung

Auslöser

- Start der Anwendung

Kurzbeschreibung

Die Anwendung wird gestartet, die Konfiguration gelesen, die Verbindung zur Datenbank wird hergestellt, die Implementierungen für Prognosealgorithmen werden gesucht und geladen.

Vorbedingungen

Nachbedingungen

- Datenbankverbindung ist konfiguriert
- Mindestens eine Implementierung eines Prognosealgorithmus ist vorhanden und geladen

Typischer Ablauf

Server- und Datenbankname werden aus einer Konfigurationsdatei gelesen. Eine SessionFactory für Fluent NHibernate wird konfiguriert, auf die angegebene Datenbank zuzugreifen. Die Mappingklassen werden ausgelesen und von Fluent NHibernate verarbeitet. Mittels Managed Extensibility Framework (MEF) werden die vorhandenen Prognosealgorithmusimplementierungen gesucht. Die Anwendung startet.

Alternativer Ablauf: Datenbank nicht erreichbar, Server- oder Datenbankname falsch

Schlägt der Versuch eine Datenbankverbindung zu konfigurieren fehlt, kann die Anwendung kann nicht gestartet werden. Der Benutzer wird über den vorliegenden Fehler informiert.

Alternativer Ablauf: Keine Prognosealgorithmen vorhanden

Um Prognosen erstellen zu können, muss mindestens eine Implementierung eines Prognosealgorithmus verfügbar sein. Sind keine Prognosealgorithmusimplementierungen vorhanden, wird der Start der Anwendung mit Fehlermeldung abgebrochen. Das Erstellen von Prognosen spielt eine zentrale Rolle für die Wartungskostenoptimierung. Es wäre möglich, die Anwendung ohne Prognosealgorithmus zu starten, aber angesichts dessen Bedeutung unsinnig.

B.2 Erfassung eines Geldautomaten

Akteure

- Anwender der Wartungskostenoptimierung

Auslöser

- Betätigung der New Schaltfläche der Cashpoint-Toolbar in der Benutzeroberfläche

Kurzbeschreibung

Ein neues Geldautomat-Objekt wird angelegt. Hierbei werden eine Bezeichnung, die Sicherheitsmarge und das maximale Fassungsvermögen erfasst. Der Benutzer bestätigt seine Eingaben und löst damit das Speichern und Anzeigen des Geldautomaten aus.

Vorbedingungen

- Anwendungsfall B.1

Nachbedingungen

- Neuer Geldautomat ist in der Datenbank angelegt und in der Benutzeroberfläche angezeigt.

Typischer Ablauf

Ein Dialog fordert zur Eingabe von Bezeichnung, Sicherheitsmarge und maximalem Fassungsvermögen des anzulegenden Geldautomaten auf. Der Benutzer gibt die erforderlichen Daten ein und bestätigt die Erstellung mit Save. Der neue Geldautomat wird angelegt und in der Datenbank gespeichert. Anschließend wird die Anzeige aktualisiert, sodass der neue Geldautomat angezeigt wird.

Alternativer Ablauf: Abbruch

Der Benutzer bricht durch das Schließen des Eingabedialogs ohne voriges Speichern ab. Es wird kein Geldautomat angelegt.

B.3 Generierung von Testdaten

Akteure

- Anwender der Wartungskostenoptimierung

Auslöser

- Start des Kommandozeilenprogramms zur Erzeugung von Testdaten

Kurzbeschreibung

Unter Angabe einiger Parameter werden Geldautomaten, Faktoren und zu jedem Geldautomaten eine Historie der Füllstände generiert. Um dabei eine reproduzierbare Testumgebung schaffen zu können, wird angeboten die Datenbank vor dem Einfügen neuer Daten zu leeren.

Vorbedingungen

- Anwendungsfall B.1

Nachbedingungen

- Optional: Datenbank wurde entleert.
- Neue Geldautomaten angelegt.
- Fest definierte Menge an Faktortypen angelegt, sofern nicht vorhanden.
- Faktoren für vergangene und optional zukünftige Tage im Kalender der neuen Geldautomaten angelegt.
- Füllstandhistorie zu neu angelegten Geldautomaten angelegt.

Typischer Ablauf

Zunächst wird der Benutzer zur Eingabe der Parameter für die auszuführenden Aktionen aufgefordert. Dies sind:

- Soll die Datenbank geleert werden?

- Wie viele Geldautomaten sollen angelegt werden?
- Für wie viele Monate in die Vergangenheit sollen Füllstände und Kalendereinträge generiert werden?
- Für wie viele Tage in die Zukunft sollen Kalendereinträge generiert werden?

Anschließend werden die gesammelten Parameter zusammengefasst dargestellt. Bestätigt der Benutzer diese, stößt er damit die gewünschten Aktionen an. Der Abschluss der Datengenerierung wird dem Benutzer anschließend mitgeteilt und das Programm beendet.

Alternativer Ablauf: Abbruch

Bestätigt der Benutzer die gemachten Angaben nicht, wird das Programm ohne Veränderung der Datenbank beendet.

B.4 Training von Prognosekomponenten

Akteure

- Anwender der Wartungskostenoptimierung

Auslöser

- Betätigung der Training Schaltfläche nach Auswahl eines oder mehrerer Geldautomaten

Kurzbeschreibung

Für die ausgewählten Geldautomaten werden Prognosekomponenten angelegt und trainiert.

Vorbedingungen

- Anwendungsfall B.1

Nachbedingungen

- Selektierten Geldautomaten ist jeweils mindestens eine trainierte Prognosekomponente zugewiesen

Typischer Ablauf

Für die selektierten Geldautomaten wird je eine bereits vorhandene Prognosekomponente ausgewählt. Sind für einen Geldautomaten noch keine Prognosekomponenten angelegt, wird dies an dieser Stelle nachgeholt. Die gewählten oder neu angelegten Prognosekomponenten werden trainiert und dabei eventuell erzeugte Parameterwerte abgerufen und gespeichert. Die Benutzeroberfläche informiert den Benutzer über den Fortschritt der ausgeführten Aktionen.

B.5 Wartungsplan erstellen

Akteure

- Anwender der Wartungskostenoptimierung

Auslöser

- Betätigung der Create plan Schaltfläche nach Auswahl eines oder mehrerer Geldautomaten und eines Enddatums.

Kurzbeschreibung

Für die ausgewählten Geldautomaten wird ein bis zum ausgewählten Enddatum reichender Wartungsplan erstellt.

Vorbedingungen

- Anwendungsfall B.1

Nachbedingungen

- Für jeden ausgewählten Geldautomaten sind für die Zeitspanne vom letzten realen Füllstand bis zum gewählten Enddatum Füllstände und Wartungen eingetragen.

- Über das Enddatum hinaus reichende generierte Füllstände eines früheren Durchlaufs der Wartungsplanung sind entfernt.

Typischer Ablauf

Der Benutzer wählt einen oder mehrere Geldautomaten und ein Enddatum für den zu erstellenden Plan aus. Über die Create plan Schaltfläche wird die Wartungsplanung für die selektieren Geldautomaten gestartet. Die Planung läuft in den folgenden Schritten ab:

- Laden der Prognosekomponenten zu den selektieren Geldautomaten
- Ist zu einem der Geldautomaten kein trainiertes Prognoseobjekt vorhanden, wird dies analog zu Anwendungsfall B.4 nachgeholt.
- Suche eines letzten realen Füllstandes je gewähltem Geldautomat
- Prognose der Abhebebeträge für die Tage vom letzten Füllstand bis zum Enddatum
- Aufstellen der Wartungspläne für die geforderte Zeitspanne mit Hilfe der prognostizierten Abhebebeträge nach dem in Abschnitt 5.2.3 beschriebenen Verfahren.
- Löschen eventuell vorhandener, über das Enddatum hinaus reichender Füllstände eines vorigen Planungsdurchlaufs.
- Aktualisierung der angezeigten Daten.

Anhang C

UML-Diagramm

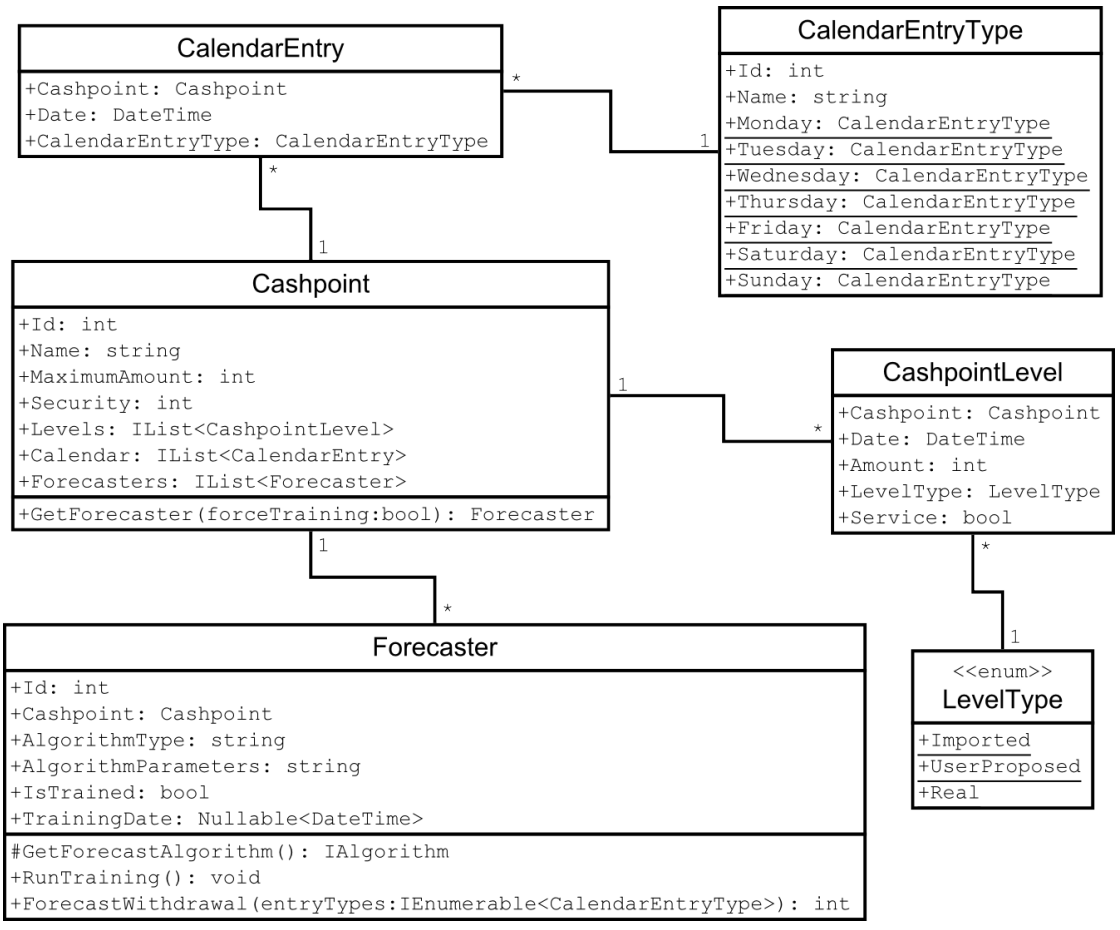

Abbildung C.1: UML-Diagramm der persistenten Klassen

Anhang D

Prognosealgorithmus - Schnittstellen

IAlgorithmFactory

IAlgorithmFactory {

IAlgorithm CreateAlgorithm(); gibt eine neue `IAlgorithm`-Instanz zurück

}

IAlgorithmInformation

IAlgorithmInformation {

string Description { get; } gibt eine Beschreibung des Prognosealgorithmus zurück

Type Type { get; } gibt den Typ des Prognosealgorithmus zurück - dieser wird zur Identifizierung des Algorithmus verwendet

}

IAlgorithm

IAlgorithm {

void LoadParameters(string parameters); weist den Algorithmus an, die übergebenen Parameter zu laden

string SerializeParameters(); momentan gesetzte Parameter in serialisierter Form zur Speicherung zurückgeben

bool NeedsTraining(Cashpoint cashpoint, DateTime? lastTraining); zeigt an, ob dieses Algorithmusobjekt trainiert werden muss

void RunTraining(Cashpoint cashpoint); startet einen Trainingslauf

int ForecastWithdrawal(Cashpoint cashpoint, IEnumerable<CalendarEntryType> entryTypes); prognostiziert einen Abhebebetrag in Abhängigkeit von einer Menge Kalendereinträgen

}

Anhang E

Eingesetzte Anwendungen und Bibliotheken

Für die Entwicklung der vorliegenden Arbeit wurden verschiedene Anwendungen und Bibliotheken eingesetzt, die hier kurz vorgestellt werden.

Visual Studio 2008 Die von Microsoft entwickelte integrierte Entwicklungsumgebung Visual Studio ist das Standardwerkzeug für die Entwicklung von Projekten, die auf das .NET Framework aufsetzen.

.NET Framework 3.5 Unter dem Begriff .NET Framework fasst ihr Urheber - Microsoft - eine Laufzeitumgebung, eine Sammlung von Klassenbibliotheken, einen Compiler und viele weitere Werkzeuge zusammen.

Microsoft SQL Server 2008 - Express Microsofts SQL Server ist ein relationales Datenbanksystem, dass in unterschiedlichen Varianten verfügbar ist. Die Express Version ist im Bezug auf Datenbankgröße, genutztem Arbeitsspeicher und auf die Verwendung nur eines Prozessors begrenzt und hauptsächlich für Entwicklungszwecke gedacht. Hiermit entworfene Datenbanken lassen sich auf vollwertige Versionen des SQL Servers übertragen.

Fluent NHibernate Das von James Gregory geleitete Open Source-Projekt erweitert das objektrelationale Mappingframework NHibernate um eine vereinfachte Konfiguration. Die Abbildung von Klassen und deren Eigenschaften auf Datenbanktabellen und Spalten geschieht statt mit Xml-Dateien in sog. Mappingklassen. Der Vorteil dieses Ansatzes liegt in der leichteren Refaktorisierung beispielsweise bei der Änderung der Bezeichnung einer Eigenschaft. Der Compiler stellt hierbei sicher, dass auch das Mapping geändert wird. In einer Xml-Datei wie NHibernate sie verwendet, muss eine solche Änderung manuell nachgepflegt werden.

Aforge.NET Eine Sammlung - in C# geschriebener - Bibliotheken für Entwicklung und Forschung im Bereich künstlicher Intelligenz und künstlichen Sehens. Unter anderem stellt das AForge.NET Framework eine Implementierung künstlicher neurona-

ler Netze bereit. Diese wird in der vorliegenden Arbeit für die Prognose verwendet. Weitere Informationen unter AForge.NET (2009)

famfamfam Silk Icons Mark James stellt auf seiner Webseite (`http://www.famfamfam.com/`) mehrere Sammlungen von kostenlos zu benutzenden Icons bereit. Diese wurden bei der Umsetzung der Benutzeroberfläche eingesetzt.

MEF Das Managed Extensibility Framework vereinfacht die Erstellung einer erweiterbaren Anwendung. Es stellt Funktionalitäten zur Suche nach und zum Laden von Modulen eines Systems bereit. Es wurde von Microsoft unter `http://mef.codeplex.com/` als Vorabversion inklusive Quellcode bereitgestellt und wird in das kommende .NET Framework 4.0 aufgenommen. In der vorliegenden Arbeit wird es zum Finden und Laden der Implementationen eines Prognosealgorithmus verwendet.

WPF Toolkit Das WPF Toolkit ist eine kostenlose und im Quellcode verfügbare Sammlung von Steuerelementen. Der Kalender zur Wahl eines Datums in der Benutzeroberfläche entstammt dieser Bibliothek.

tortoisegit Git ist ein verteiltes Versionsverwaltungstool. Die Integration in den Windows-Explorer - tortoisegit - befindet sich derzeit in Entwicklung, konnte aber im Umfang dieser Arbeit erfolgreich eingesetzt werden. Es bot sich vor allem aus dem Grund an, dass unabhängig von einem Server die Fortschritte der Entwicklung in einem Versionsmanagement gepflegt werden können.

Literaturverzeichnis

[AForge.NET 2009] AFORGE.NET: *AForge.NET :: Framework*. http://www.aforgenet.com/framework/. Version: 2009

[Alpaydin 2008] ALPAYDIN, Ethem: *Maschinelles Lernen*. München : Oldenbourg, 2008

[Fluent NHibernate 2009] FLUENT NHIBERNATE: *Fluent NHibernate wiki / Main Page*. http://wiki.fluentnhibernate.org/Main_Page. Version: 2009

[Hastie u. a. 2009] HASTIE, Trevor ; TIBSHIRANI, Robert ; FRIEDMAN, Jerome: *The Elements of Statistical Leraning*. New York : Springer, 2009

[it kompetenz GmbH 2009] IT KOMPETENZ GMBH: *Firma*. http://itkompetenz.de/firma.html. Version: 2009. – [Online; Stand 10. Oktober 2009]

[Keller 2000] KELLER, Hubert B.: *Maschinelle Intelligenz : Grundlagen, Lernverfahren, Bausteine intelligenter Systeme*. Braunschweig : Vieweg, 2000

[Kriesel 2007] KRIESEL, David: *Ein kleiner Überblick über Neuronale Netze*. 2007. – erhältlich auf http://www.dkriesel.com – Version:EPSILON-DE

[Michalski 1986] MICHALSKI, Ryszard S.: Understanding The Nature Of Learning: Issues And Research Directions. In: *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, 1986, 3–25

[Morik 1993] MORIK, Katharina: Maschinelles Lernen / Universität Dortmund, Fachbereich Informatik, Lehrstuhl VIII. Version: Februar 1993. http://hdl.handle.net/2003/2574. Dortmund : Universität Dortmund, Februar 1993 (1). – Forschungsbericht

[Smola u. Schölkopf 2009] SMOLA, Alex ; SCHÖLKOPF, Bernhard: *Kernel-Machines.Org*. http://www.kernel-machines.org/. Version: 2009. – [Online; Stand 20. Oktober 2009]

[Wikipedia 2009] WIKIPEDIA: *Serialisierung — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Serialisierung&oldid=61641192. Version: 2009. – [Online; Stand 14. Oktober 2009]

[Zell 2003] ZELL, Andreas: *Simulation neuronaler Netze*. München : Oldenbourg, 2003

Index

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 30. Oktober 2009	
Ort, Datum	Unterschrift