Fachhochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Freie wissenschaftliche Arbeit zur Erlangung des akademischen Grades
Bachelor of Science in Wirtschaftsinformatik

Konzeption einer Applikation zur Visualisierung von Statistikdaten im Retail Bereich Self-Checkout

Bachelorthesis

im Fachbereich Wirtschaftswissenschaften II
im Studiengang Wirtschaftsinformatik
der Fachhochschule für Technik und Wirtschaft Berlin

vorgelegt von: Johannes Wolf
Sonnenallee 114
12045 Berlin
Matrikel-Nr: 0510984

Erstbetreuer: Prof. Dr. Burkhard Messer
Zweitbetreuer: Prof. Dr. Ingo Claßen

Abgabetermin: 31.08.2007

Schutzklausel

Die Inhalte dieser Bachelorthesis sind **nicht** zur Veröffentlichung gedacht und unterliegen der Geheimhaltung.

Vorwort

Im Rahmen meines praktischen Studiensemesters bei der Wincor Nixdorf International GmbH war ich von November 2006 bis März 2007 in der Abteilung WN Retail Store Solutions beschäftigt und habe mich dort mit der Planung einer Anwendung zur zentralen Speicherung und Auswertung von Statistikdaten im Bereich Self-Checkout beschäftigt.

Die Aufgabe bestand darin, ein bestehendes Modul der Systemsoftware TPiSCAN zur Erzeugung von Statistikdaten so zu erweitern, dass eine zentralisierte Datenhaltung aufgesetzt wurde. Dies war insofern relevant, da viele Kunden der TPiSCAN Software die vom Statistikmodul erzeugten Daten zu einem späteren Zeitpunkt mit professionellen Statistik-Auswertungstools aufbereiteten, wozu sich die bis dahin erzeugten XML-Dateien nicht gut eigneten.

Resultierend aus meiner Beschäftigung als Praktikant bei Wincor Nixdorf entstand dann auch das Thema dieser Bachelorthesis: „Konzeption einer Applikation zur Visualisierung von Statistikdaten im Retail Bereich Self-Checkout“.

Neben dem Entwurf und der Realisierung einer zentralisierten Datenhaltung in Form einer Datenbank wollte Wincor Nixdorf seinen Kunden auch eine eigene Anwendung zur Visualisierung und Auswertung der Statistikdaten anbieten, damit die Kunden nicht gezwungen sind, auf teure Auswertungstools von Drittanbietern zurückgreifen zu müssen.

Aufbauend auf meinen Erfahrungen aus dem praktischen Studiensemester bei Wincor Nixdorf, habe ich im Rahmen dieser Bachelorthesis das Konzept für eine Applikation entwickelt, welche den ermittelten Anforderungen entspricht und heute fertig implementiert als Bestandteil der TPiSCAN Software an die Kunden ausgeliefert wird.

An dieser Stelle möchte ich den Personen danken, welche mich beim Anfertigen dieser Arbeit und der inhaltlichen Auseinandersetzung mit der Thematik unterstützt haben. Vielen Dank an meine Betreuer seitens der FHTW Berlin, Herrn Prof. Dr. Burkhard Messer und Herrn Prof. Dr. Ingo Claßen und die Mitarbeiter von Wincor Nixdorf, vor allem Herrn Michael Reschke, Herrn Harry-Erich Pape, Herrn Thomas Schicktanz, Herrn Dr. Sönke Kannapinn, Herrn Dr. Andreas Eibeck und Herrn Nicolas Gabron.

Berlin, August 2007 Johannes Wolf

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	-	Application Programming Interface
CSS	-	Cascading Style Sheets
DB	-	Datenbank
DBMS	-	Datenbank Management System
DTD	-	Document Type Definition
eEPK	-	erweiterte Ereignisgesteuerte Prozesskette
EDV	-	elektronische Datenverarbeitung
ERM	-	Entity-Relationship-Modell
GPL	-	GNU General Public License
GUI	-	Graphical User Interface
HTTP	-	Hypertext Transfer Protocol
IEEE	-	Institute of Electrical and Electronic Engineers
IPX/SPX	-	Internetwork Packet Exchange / Sequenced Packet Exchange
IT	-	Informationstechnologie
JDBC	-	Java Database Connectivity
JPEG	-	Joint Photographic Experts Group
JSP	-	Java Server Page
JVM	-	Java Virtual Machine
LAN	-	Local Area Network
LGPL	-	GNU Lesser General Public License
MS	-	Microsoft
MVC	-	Model-View-Controller Pattern
OEM	-	Original Equipment Manufacturer
OGNL	-	Object-Graph Navigation Language
OLAP	-	Online Analytical Processing
OS	-	Operating System / Betriebssystem
PDF	-	Portable Document Format
PNG	-	Portable Network Graphics
POS	-	Point of Sales

RDI	-	Retail Device Interface
RMI	-	Remote Method Invocation
RPC	-	Remote Procedure Call
SCO	-	Self-Checkout
SRS	-	Software Requirements Specification
SVG	-	Scalable Vector Graphics
SVN	-	Subversion
SQL	-	Structured Query Language
TCP/IP	-	Transmission Control Protocol / Internet Protocol
UPOS	-	Unified POS
URL	-	Uniform Resource Locator
UTC	-	Universal Time Coordinated
WAN	-	Wide Area Network
WN	-	Wincor Nixdorf
XML	-	Extensible Markup Language

1 Einleitung

Das Ziel dieser Arbeit ist es, auf der Grundlage einer umfassenden Anforderungsanalyse ein geeignetes Konzept für die Visualisierung von Statistikdaten innerhalb der Systemsoftware TPiSCAN von Wincor Nixdorf zu entwerfen. Diese konzeptionelle Arbeit umfasst neben der Betrachtung der Ist-Situation und dem Aufstellen eines betriebswirtschaftlichen Sollkonzeptes auch wichtige Aspekte des Entwurfs einer geeigneten Applikation. Die konzeptionellen Vorüberlegungen innerhalb dieser Arbeit sollen anschließend die konkrete technische Implementierung der Applikation unterstützen.

Die Überlegungen umfassen einerseits den Entwurf einer geeigneten Anwendungs-Architektur, welche in die TPiSCAN Systemumgebung integriert werden kann, andererseits werden geeignete Technologien vorgestellt, mit denen sich die aufgestellten betriebswirtschaftlichen Anforderungen effektiv umsetzen lassen. Des Weiteren werden Überlegungen zur notwendigen Anwendungs-Logik aufgestellt, welche bereits konkrete Vorschläge für die technische Realisierung der Applikation enthalten.

Den Einstieg in die Thematik liefern die ersten vier Kapitel dieser Arbeit. Neben einer Begriffsbestimmung, welche auf die Formulierung des Themas dieser Bachelorthesis eingeht, findet sich hier auch das Unternehmensprofil, welches dabei hilft, die Thematik in die Tätigkeiten der Wincor Nixdorf International GmbH einzuordnen. Schließlich wird mit dem Anwendungsszenario eine Übersicht zu den betriebswirtschaftlichen und technischen Gegebenheiten im Einzelhandelsbereich Self-Checkout vermittelt. Dieser Einstieg soll dabei helfen, die Anforderungsanalyse nachvollziehen zu können und die entwickelten Lösungsansätze besser bewertbar zu machen.

Programmcode und Ausführungen zur Implementierung welche in dieser Arbeit keinen Platz mehr fanden, aber von mir für wichtig erachtet wurden, finden sich im Anhang wieder.

2 Begriffsbestimmungen

2.1 Visualisierung von Statistikdaten

2.1.1 Visualisierung

Unter einer Visualisierung wird die „*bildliche Aufbereitung, Darstellung und Kommunikation von Information*“[1] verstanden. „*Der Begriff umfasst alle Arten bildlicher Repräsentation und Sichtbarmachung*“[2]. Ziel dabei ist die Veranschaulichung von abstrakten Daten und Zusammenhängen. Dabei sollen bestimmte Zusammenhänge aufzeigt werden, welche sich zwar aus dem gegebenen Datenbestand ergeben, aber für den Betrachter nicht unmittelbar sichtbar werden.

Eine Visualisierung beinhaltet auch immer eine Interpretation der darzustellenden Daten, da Details des Datenbestandes betont oder weggelassen werden können. Eine kontextunabhängige Visualisierung von Informationen, allein zum Zwecke der Illustration wird eher selten verwendet.

2.1.2 Statistik

Als Statistik bezeichnet man die Zusammenfassung verschiedenster Methoden, welche Massendaten quantifizier- und interpretierbar machen. Es geht also darum, große Datenbestände zu sammeln und auszuwerten, indem die empirischen Daten beschrieben, zusammengefasst und anschließend induktiv Eigenschaften der (Stichproben-) Daten abgeleitet werden.[3] [4]

Die Statistik bedient sich der Visualisierung, um aufbereitete Daten als Ergebnis zu präsentieren. Dazu werden am häufigsten Tabellen und Diagramme verwendet. Eine Tabelle ist eine geordnete Zusammenstellung von Daten, wobei die Daten in Zeilen und Spalten aneinander ausgerichtet sind. Ein statistisches Diagramm hingegen ist

[1] Brockhaus 2007 Visualisierung

[2] Brockhaus 2007 Visualisierung

[3] vgl. Eckstein 2003, S. 2

[4] vgl. Schweizer Aggregation 1990, S. 13

ganz allgemein eine grafische Darstellung von Häufigkeiten zur Erklärung eines Zusammenhangs mit Hilfe geometrischer Figuren.[5]

In mathematischen oder statistischen Diagrammen werden meist Zusammenhänge zwischen zwei oder mehr Messgrößen dargestellt. Typische Vertreter dieser Diagrammtypen sind das Liniendiagramm, das Balkendiagramm oder das Kreisdiagramm. Die Statistik bedient sich aber auch spezieller Diagrammtypen um statistische Maßzahlen grafisch abzubilden.[6] Dazu zählen beispielsweise das Histogramm zur Darstellung der Häufigkeiten bei Klassenbildung[7] und der Box-Whistler-Plot, welcher unter anderem zur Darstellung von Häufigkeitsverteilung, Streuung und Schiefe verwendet wird.[8]

2.2 Self-Checkout im Bereich Retail

Mit dem Begriff Checkout umschreibt man im Bereich des Einzelhandels (engl. „retail“) einen Verkaufsort. Alternativ trifft man in der Literatur auch auf den Begriff Point of Sale (POS). Aus Sicht des Einzelhändlers bezeichnet Checkout die Verkaufsstelle, zumeist einen Kassenarbeitsplatz. Aus Sicht des Käufers ist der Checkout die Einkaufsstelle und stellt deshalb aus Marketingsicht einen wesentlichen Bestandteil des Händlerimages dar.

Immer mehr Händler denken über Prozessveränderungen am Checkout nach, um Prozess- und Personalkosten zu reduzieren und somit dem wachsenden Wettbewerbsdruck entgegenzuwirken. Vor allem der traditionelle Kassenarbeitsplatz bietet dem Einzelhandel enormes Einsparungs- und Optimierungspotential.

Das Prinzip des klassischen Self-Checkout (SCO) ist simpel: Der Kunde scannt den Barcode seiner Ware selbst und packt sie anschließend in Einkaufstüten oder legt sie auf ein Transportband. Eine integrierte Sicherheitswaage kontrolliert dabei das Gewicht der gescannten Waren. Die Bezahlung erfolgt eigenständig und möglichst ohne Fremdunterstützung. Trotzdem werden Self-Checkout Terminals typischerweise von

[5] vgl. Hartung 1986, S. 21-23

[6] vgl. Eckstein 2003, S. 19

[7] vgl. Hartung 1986, S. 27-28

[8] vgl. Hartung 1986, S. 835-838

einer Aufsichtsperson - dem so genannten Attendant - an einer Attendant-Station beaufsichtigt und bei Bedarf unterstützt.[9]

Neben den klassischen SCO Konzepten kommt neuerdings der Prozesstrennung von beispielsweise Bezahlen und Scannen eine immer wichtigere Bedeutung zu, da durch die Parallelisierung der Prozesse eine deutliche Durchsatzerhöhung beim Checkout erreicht werden kann.

Abb. 1: Terminal "iSCAN POS Tower 100 Scan&Bag"
(Quelle: WN Website S&B 2007, Seite 1)

Letztendlich erhofft sich der Handel neben den angesprochenen Einsparungen vor allem auch eine Steigerung der Kundenzufriedenheit. SCO soll zu verkürzten Warteschlangen führen, mehr Privatsphäre bieten und dem Kunden die volle Kontrolle über den Checkout-Prozess überlassen.

[9] vgl. WN Website SCO 2007

3 Unternehmensprofil

3.1 Das Unternehmen Wincor Nixdorf

Die Wincor Nixdorf International GmbH mit Hauptsitz in Paderborn ist der führende europäische Hersteller und weltweit die Nummer drei bei Geldautomaten, Kiosksystemen und Computerkassen, den so genannten Point of Sale Systemen (POS). Zum Produktspektrum gehören außerdem Software-Lösungen, Dienstleistung und Consulting für Handelsunternehmen, Banken, Lotteriegesellschaften und Firmen aus verwandten Bereichen.[10]

Wincor Nixdorf wurde am 1. Oktober 1999 als eigenständige Gesellschaft gegründet. Vorgänger waren die Siemens-Nixdorf Retail & Banking Systems GmbH (1998-1999), die Siemens-Nixdorf Informationssysteme AG (1990-1998) und die Nixdorf Computer AG (1952-1990) des deutschen Computerpioniers Heinz Nixdorf. Die Umbenennung in Wincor Nixdorf fand 1999 nach dem Kauf des Unternehmens durch die amerikanische Beteiligungsgesellschaft Kohlberg Kravis Roberts & Co. (KKR) statt, welche 90% der Unternehmensanteile übernahm.[11]

Seit Mai 2004 wird die Aktie der Muttergesellschaft Wincor Nixdorf Holding AG an der Frankfurter Börse gehandelt. Seit dem Börsengang hat KKR seine Anteile wieder verkauft. Heute ist Wincor Nixdorf in 34 Ländern mit eigenen Tochtergesellschaften und in mehr als 60 Ländern über Partnerunternehmen vertreten. Knapp 8116 Mitarbeiter weltweit erwirtschafteten im 1. Halbjahr 2006/2007 (1. Oktober 2006 – 31. März 2007) einen Umsatzerlös von 1,085 Milliarden Euro und ein operatives Ergebnis (EBITA) von 90 Millionen Euro, was einer Brutto-Umsatzrentabilität von 8,3% entspricht.[12]

[10] vgl. WN Website Company 2007

[11] vgl. WN Website Historie 2007

[12] vgl. WN Halbjahresbericht 06/07

3.2 Geschäftsfeld

Die Unternehmens-Vision besagt: „*Wir wollen das Unternehmen Wincor Nixdorf zum führenden Anbieter von IT-Services und -Lösungen für das Filialgeschäft von Banken und Handelsunternehmen machen!*“[13]. Die Unternehmens-Vision bringt zum Ausdruck, dass die unternehmerische Tätigkeit von Wincor Nixdorf auf den beiden Säulen Banken (Banking) und Handel (Retail) aufbaut. Auch im Geschäftsbericht wird die Unterteilung in diese beiden Kernkompetenzen vorgenommen.

Im Zentrum der Aktivitäten von Wincor Nixdorf steht dabei die Optimierung von kundenbezogenen Prozessen, insbesondere von Unternehmen mit ausgeprägtem Filialgeschäft. Diese Prozessoptimierung wird durch ein breites Angebot an Automatisierungs- und Selbstbedienungsprodukten, Kassenhardware sowie den zugehörigen Software-, Beratungs- und Serviceleistungen unterstützt und umgesetzt. Ziel ist dabei die Verbesserung der Servicequalität bei gleichzeitiger Kostenreduktion im Filialbereich.[14]

3.3 WN Retail Store Solutions

Während das Bankengeschäft mit Geldautomaten knapp 64% des Gesamtumsatzes ausmacht, stellt der Geschäftsbereich Handel (Retail Division) das wesentlich kleinere Kerngeschäft dar.[15] Zur Retail Division werden verschiedene Organisationseinheiten innerhalb von Wincor Nixdorf gezählt, unter anderem auch die „WN Retail Store Solutions“ in Berlin. Diese Abteilung bietet Prozess- und IT-Kostenoptimierung durch Filialautomatisierung und Technologie-Standardisierung im Bereich Self-Checkout an. Das Lösungs- und Dienstleistungsportfolio umfasst sowohl Software zur Filialautomatisierung als auch Entwicklungs- und Beratungs-Dienstleistungen. Im Bereich Self-Checkout bietet Retail Store Solutions mit der TPiSCAN Software eine Systemsoftware zur Steuerung der hauseigenen Self-Checkout Terminals an. Eine einheitliche Schnittstelle stellt die Integration in eine vorhandene POS-Lösung sicher, welche auch von einem Drittanbieter stammen kann.[16]

---

[13] vgl. WN Website Unternehmenskultur 2007

[14] vgl. WN Website Strategie 2007

[15] vgl. WN Halbjahresbericht 06/07

[16] vgl. WN Website TPiSCAN 2007

4 Anwendungsszenario

4.1 Self-Checkout im Filialgeschäft

4.1.1 Marktsituation

"*With more than $178 Billion in projected sales through self-checkout in 2005, this study looks at arguably the hottest store-level retail technology of today.*"[17]

Mit diesem Statement beginnt die im Jahr 2006 von der IHL Consulting Group veröffentliche Studie, welche sich mit der Kundenakzeptanz bezüglich Self-Checkout-Systemen im nordamerikanischen Markt beschäftigt. Das Zitat zeigt, dass das Thema Self-Checkout in den letzten Jahren besonders vom amerikanischen Markt sehr positiv aufgenommen und bewertet wurde.

„*More than 90 percent of consumers surveyed said they had used a self-checkout lane in the previous 12 months.*"[18]

Während im amerikanischen Raum Self-Checkout schon seit vielen Jahren erfolgreich eingesetzt wird, reagierte der europäische Markt bei der Einführung eher zögerlich auf die neue Technologie. Das mag einerseits an der mangelnden Erfahrung der europäischen Retailer mit dem Thema Filialautomatisierung liegen. Andererseits lässt sich die vorhandene Skepsis des europäischen Marktes vielleicht auch mit dem sensibleren Umgang mit der Thematik der Arbeitsplatzrationalisierung erklären.

Nichtsdestotrotz haben auch die europäischen Handelsketten die Chancen erkannt, welche die Self-Checkout-Technologie mit sich bringt. So befindet sich Wincor Nixdorf mit seiner iScan-Produktfamilie und der Softwarelösung TPiSCAN augenblicklich bei drei Kunden in einem Rollout und in verschiedenen Projekten bei mehreren großen europäischen Handelsketten im Piloteinsatz. Die erste Hardware-Produktfamilie befindet sich in der Serien-Produktion und für die zweite Produktgeneration ist eine Serienproduktion der Hardware noch für das Jahr 2007 geplant.

[17] IHL Consulting Group 2006

[18] Self Service World 2007

4.1.2 Aufbau einer Filiale

Unter einer Filiale versteht man einen „*räumlich getrennten Zweigbetrieb einer Unternehmung, der rechtlich und wirtschaftlich von einer Zentrale (Hauptniederlassung) abhängig ist*“[19]. Gerade im Einzelhandel ist der Filialbetrieb sehr verbreitet. Typischerweise findet die Einführung der Self-Checkout Terminals innerhalb einer Filiale im Rahmen eines Piloteinsatzes statt. Mit diesem Testeinsatz soll gewährleistet werden, dass vor dem eigentlichen Rollout ausgiebig mit der neuen Hard- und Software getestet werden kann. Außerdem wird diese Phase benötigt, um Mitarbeiter und Kunden mit der neuen Technik vertraut zu machen und auf diese Weise auch die Akzeptanz für die Geräte zu überprüfen.

Schließlich wird in der Pilotierungsphase auch analysiert, ob sich die Erwartungen an die Anschaffung der Self-Checkout-Terminals, beispielsweise in Hinblick auf Sicherheit und Wirtschaftlichkeit, erfüllen. Für diese Testphase werden in der Filiale meist ein oder zwei Self-Checkout-Terminals installiert. Sollte sich für einen Rollout entschieden werden, kommen je nach Filialgröße dann üblicherweise zwei bis sechs Terminals zum Einsatz.

Es ist nicht üblich, nur Self-Checkout-Terminals innerhalb einer Filiale zu betreiben. Es werden stets auch weiterhin herkömmliche elektronische Kassensysteme eingesetzt, meist macht deren Anzahl deutlich mehr als 50% der angebotenen Checkout-Zonen aus.

[19] Brockhaus 2007 Filiale

Die folgende Abbildung stellt den typischen Aufbau einer Filiale mit Self-Checkout-Einsatz schematisch dar.

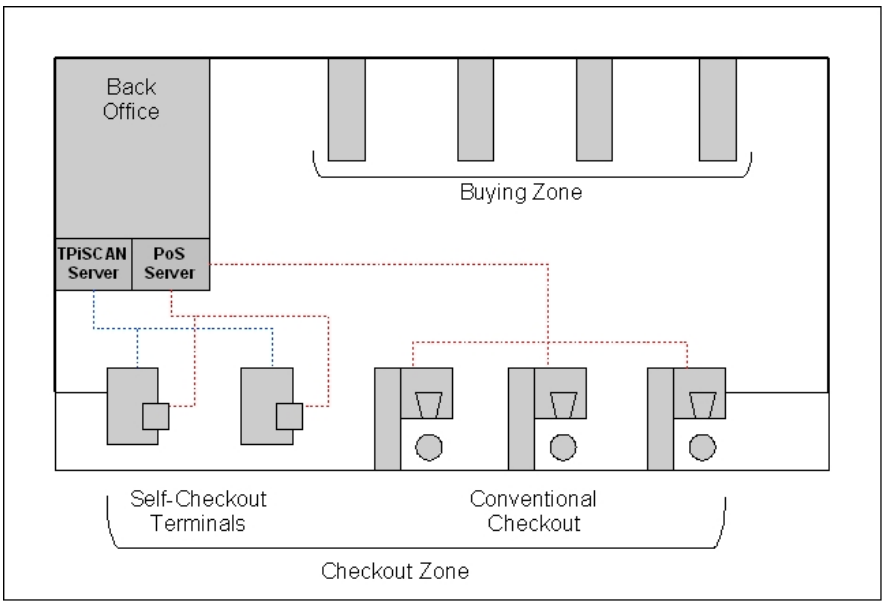

Abb. 2: Übersicht zum Aufbau einer Filiale (Quelle: eigene Darstellung)

Üblicherweise werden die eingesetzten Self-Checkout-Terminals innerhalb der Checkout Zone, also des Bezahlbereichs der Filiale, neben den herkömmlichen Kassen gruppiert aufgebaut. Während jeder herkömmliche Kassenarbeitsplatz von jeweils einem Mitarbeiter besetzt werden muss, werden in der Regel mehrere Self-Checkout-Terminals von nur einem Mitarbeiter betreut. Dieser so genannte Attendant hat die Aufgabe, den Betrieb der Terminals zu überwachen und bei Bedarf den Kunden bei der Benutzung zu unterstützen.

Technische Ausstattung

Sowohl die herkömmlichen elektronischen Computerkassen als auch die Self-Checkout Terminals arbeiten auf Basis eines modularen Rechnersystems, welches in verschiedensten Ausführungen angeboten wird. So sind Systeme als Thin- oder Fat-Clients verfügbar, werden als text- und grafikorientierte Varianten angeboten und unterstützen gängige Betriebssysteme wie MS-DOS, Linux-Derivate oder MS Windows Versionen.

An diese Rechnersysteme werden sämtliche Peripheriegeräte angeschlossen, wobei durch zahlreiche Anschlussmöglichkeiten der Forderung nach höchster Konfigurationsmodularität und Skalierbarkeit nachgekommen wird. Wincor Nixdorf

bietet hier sowohl Eigenentwicklungen als auch OEM-Peripherie an und übernimmt bei Bedarf auch die Systemintegration von geeigneten Fremdkomponenten.[20]

Während die herkömmlichen Kassensysteme mit handelsspezifischer Peripherie wie Kassendrucker, Barcode-Scanner, Geldlade, Flachbildschirm, Kartenleser und Bondrucker ausgestattet sind, kommen bei den Self-Checkout-Terminals spezielle Komponenten wie Touchscreen-Monitor, Münz- und Notenprüfer, Münz- und Notenausgabe sowie Sicherheitsfeatures wie Kontrollwaage und Statusampel zum Einsatz.

Sämtliche Rechnersysteme kommunizieren innerhalb einer Filiale über ein standardisiertes Netzwerk wie LAN oder WAN mit dem sich im Backoffice der Filiale befindlichen Server. Bei dem Netzwerk kann es sich sowohl um ein drahtloses oder auch ein verkabeltes Netzwerk handeln, als Protokolle werden TCP/IP oder IPX/SPX verwendet. [21]

Der Begriff Backoffice bezeichnet hier einen separaten Raum innerhalb der Filiale, zu dem die Kunden keinen Zutritt haben. In diesem Raum befindet sich die zentrale EDV der Filiale, also meist ein oder auch mehrere Server, welche für den Betrieb der Kassen innerhalb der Checkout Zone notwendig sind.

4.2 Vorhandene Informationssysteme und Systemsoftware

4.2.1 POS-Applikation

Im Backoffice wird zwischen dem so genannten POS-System und dem TPiSCAN-Server unterschieden. Während das POS-System mit allen Kassensystemen der Checkout Zone verbunden ist, sind lediglich die Self-Checkout Terminals mit dem TPiSCAN Server verbunden.

Das POS-System und der TPiSCAN-Server sind zwei voneinander vollständig unabhängige Anwendungen. Das POS-System ist für die Kommunikation sämtlicher Kassen mit dem Filial-Server zuständig und bietet ansonsten vor allem Funktionen für die Komponenten-Konfiguration, Preisgestaltung und Verkaufsanalyse an.[22] Dabei beruht das POS-System auf einer „UPOS“ genannten standardisierten Geräte-

[20] vgl. WN Datenblatt Kassenfamilie BEETLE 2007, Seiten 2-4

[21] vgl. WN Datenblatt iSCAN Basic Line 2007, Seite 2

[22] vgl. Stahlknecht 2005, S. 370-372

schnittstelle, welche je nach verwendetem Betriebssystem in verschiedenen Implementierungen (wie Beispielsweise JavaPOS oder RDI) existiert.

Das POS-System verwaltet auch artikelspezifische Stammdaten, welche in einer Datenbank gepflegt werden. Diese Daten stehen allerdings nur dem POS-System zur Verfügung und können von der TPiSCAN Anwendung nicht abgefragt werden. An dieser Stelle wird nicht näher auf das POS-System eingegangen, da es für diese Arbeit nicht relevant ist. Stattdessen widmet sich das nächste Kapitel ausführlicher der TPiSCAN Anwendung.

4.2.2 TPiSCAN Applikation

Die TPiSCAN Anwendung ist eine vollständig in Java entwickelte Lösung zum Betreiben der Self-Checkout-Terminals von Wincor Nixdorf. Da Self-Checkout spezifische Prozesse und Funktionen wie beispielsweise die Kontrolle der Sicherheitswaage und das Cash-Management, also die Verwaltung des ein- und ausgezahlten Geldes, nicht vom POS-System unterstützt werden, musste eine eigene Anwendung geschaffen werden.

Dies hat den Vorteil, dass bei der Integration von Self-Checkout in eine Filiale nicht das vorhandene POS-System ausgetauscht werden muss, sondern nur ein Adapter zur TPiSCAN Applikation implementiert werden muss. Für die Entwicklung dieses Adapters stehen neben umfangreichen Interface Beschreibungen auch Bibliotheken für die Programmiersprachen Java, C und .NET zur Verfügung.[23]

[23] vgl. WN Website TPiSCAN 2007

Die folgende Abbildung zeigt den Zusammenhang von POS-System und TPiSCAN Applikation.

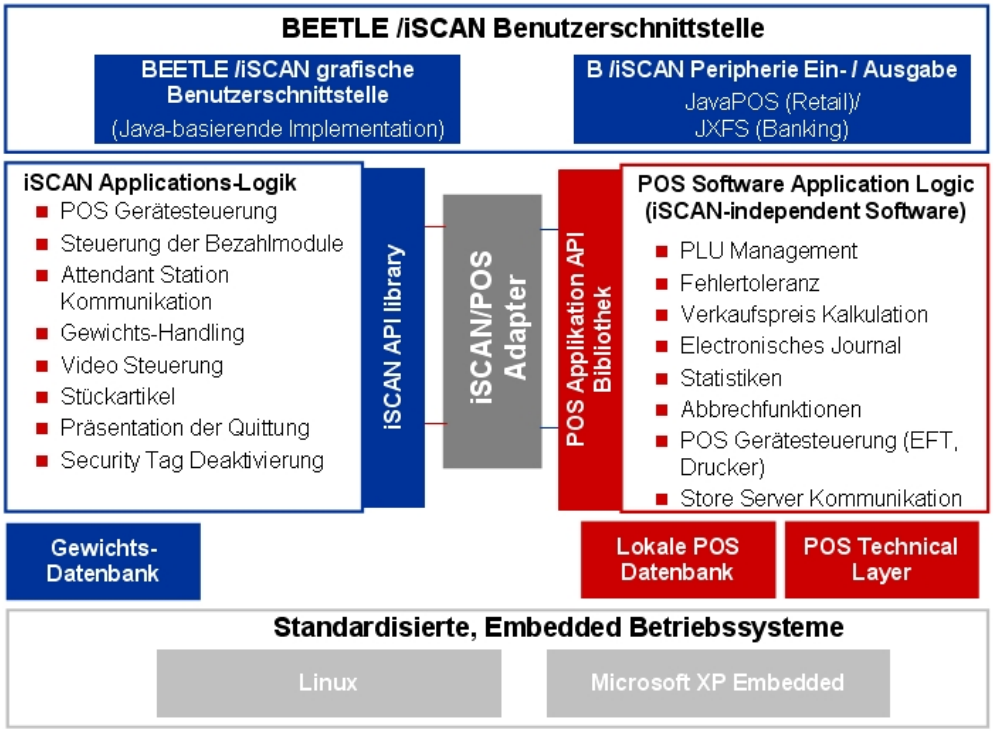

Abb. 3: Zusammenhang zwischen TPiSCAN- und POS-System (Quelle: WN SCO Solutions 2005, S. 9)

Sämtliche Kommunikation zwischen TPiSCAN Software und POS-System findet über den Adapter statt. Mit so genannten POS Calls – also Anfragen an das POS-System – kann die TPiSCAN Applikation Teilaufträge zur Durchführung des Verkaufsvorgangs anordnen und als Reaktion darauf Status-Informationen vom POS-System erhalten.

Die TPiSCAN Applikation ist eine Client-Server Lösung. Dabei werden sämtliche Self-Checkout Terminals innerhalb einer Filiale als Clients – innerhalb von TPiSCAN als „Lane“ bezeichnet – betrieben, welche mit einem TPiSCAN Server im Backoffice kommunizieren. Während der TPiSCAN Server lediglich die Artikelgewichts-Datenbank verwaltet und die Benutzer-Authentifizierung überwacht, werden die einzelnen Terminals als Fat-Clients betrieben, auf denen jeweils die gesamte Business-Logik abläuft. Da es Self-Checkout Terminals in den unterschiedlichsten Ausstattungen gibt, wird der speziell für eine Ausstattung gebaute Software-Client auch als „Run Composite“ bezeichnet.

Die in der TPiSCAN Server-Software integrierte Verwaltung der Artikelgewichts-datenbank bedient sich des Datenbankmanagement-Systems MySQL. Die Datenbank wird lokal aus dem TPiSCAN Server aus Java heraus über einen JDBC-Treiber

angesprochen und gelesen. Des Weiteren verfügt TPiSCAN über den Webserver Apache Tomcat, welcher vollständig in die TPiSCAN Anwendung integriert ist und experimentell zur Bereitstellung von Webservices genutzt wird.

Benutzungsoberfläche

„*Alle für den Self-Checkout relevanten Geschäftsprozesse werden von der TPiSCAN Applikation genutzt und mit Hilfe einer ergonomisch optimal gestalteten graphischen Benutzeroberfläche abgebildet.*“[24]

Die Benutzungsoberfläche stellt die Schnittstelle der Applikation zum Kunden dar, welcher über einen Touchscreen Monitor das Self-Checkout-Terminal bedient. Gleichzeitig kann der Attendant über die Bedienoberfläche am Touchscreen Monitor spezielle, für den Kunden nicht erreichbare, Funktionen aufrufen, indem er sich mit einem so genannten Attendant Key direkt am Terminal identifiziert. Alternativ steht eine so genannte Attendant Station zur Verfügung. Dies ist ein separater Rechner, mit welchem der Attendant sämtliche von ihm betreute Terminals beobachten und remote auf sie zugreifen kann.

Die Oberfläche der TPiSCAN Software ist in Java Swing programmiert. Dabei wird jede Bildschirmseite einer Aktivität zugeordnet und mehrere Aktivitäten zu einem kundenspezifischen Geschäftsprozess zusammengestellt. Ein besonderes Feature der Oberfläche ist die individuelle Konfigurierbarkeit und Mehrsprachigkeit, welche konsequent umgesetzt wird. Auf diese Weise wird der Kundenwunsch nach individuellen Benutzoberflächen befriedigt.

Der TPiSCAN Server

Im Backoffice laufen auf dem TPiSCAN Server die Informationen der einzelnen Self-Checkout-Terminals zusammen. Neben der Verwaltung von Kommunikation und Datenbeständen bietet der TPiSCAN Server auch eine Reihe von Tools an, mit Hilfe derer die Konfiguration der TPiSCAN Software und Hardware vorgenommen werden kann. Sämtliche Konfigurationsdateien innerhalb der TPiSCAN Software werden prinzipiell im XML Format gehalten, so dass ein einheitlicher Umgang per Konfigurationstool möglich wird.

[24] WN Website TPiSCAN 2007

Entwicklung und Installation

Die Software ist modular aufgebaut und in Pakete unterteilt. Entwickelt wird mit Eclipse in Java, wobei zur Unterstützung der Arbeit Applikationen zur Versionsverwaltung (SVN) und zur Projektorganisation verwendet werden.

Für ein Release werden aus dem Java Code per Build-Prozess mehrere installierbare Softwarepakete zusammen gebaut. Neben dem TPiSCAN Runtime-Paket sind dies ein Administrator-Paket und ein Toolkit. Dazu kommt ein kundenspezifisches Paket mit Workflow- und GUI-Resourcen. Auf dem Server-Rechner wird mit dem Administrator-Paket der Admin-Server installiert, während alle Terminals mit dem Administrator-Paket eine Admin-Client Installation erhalten. Über diese Client-Server Struktur können dann die restlichen kundenspezifischen Software-Pakete vom Server-Rechner auf die Terminals verteilt werden.

Nach der Installation der Pakete lassen sich Server oder das entsprechende Run-Composite (zum Beispiel per Startmenü) über Skripte starten. Bei der Installation wird streng auf die Trennung von Benutzerdaten und Core-Daten geachtet. Während die Benutzerdaten den vom Kunden voll einsehbaren und konfigurierbaren Teil der Anwendung ausmachen, dürfen die Core-Daten, also der eigentliche Java-Code, nicht verändert werden. Dazu wird Java-Code aus dem Core-Bereich während des Build-Prozesses obfuskiert, um Decompilation massiv zu erschweren.[25]

[25] vgl. Java Code Obfuscation

5 Anforderungsanalyse

5.1 Vorgehensweise

Auf Grundlage des vorangegangenen Einstiegs in die Thematik Self-Checkout im Einzelhandelsbereich wird in diesem Kapitel der Ist-Zustand des Statistikmoduls von TPiSCAN analysiert. Anschließend wird eine Anforderungserhebung durchgeführt, welche sämtliche Anforderungen an die zu entwickelnde Applikation zur Visualisierung der Statistikdaten aufzeigt und dokumentiert.

Die Definition von Anforderungen und die Analyse des Ist-Zustandes bilden dabei die Grundlage für die Entwicklung der Software. Mit der Qualität der Analyse steht und fällt auch die Qualität des zu entwickelnden Produktes. Oberflächlich oder falsch ermittelte Anforderungen würden zu Unzufriedenheit und mangelnder Akzeptanz für die Software führen und somit eine spätere Nachbearbeitung mit sich bringen. [26] [27]

Die Situations-Analyse und das Aufstellen der Anforderungen erfolgen stets in Zusammenarbeit beziehungsweise der Befragung des Auftraggebers – also Wincor Nixdorf. Dabei stellen Interviews mit den Entwicklern und vorhandene Dokumentationen die primäre Quelle für Informationen dar. Daneben ergibt sich aber auch durch die Begleitung am Arbeitsplatz und die Analyse ähnlicher Systeme ein Verständnis für die Problematik.

5.2 Ist-Analyse zum Statistikmodul von TPiSCAN

5.2.1 Scope und Modi

Handelsübliche POS-Systeme verfügen in der Regel über ausgefeilte Berichts- und Statistikmodule. Denn ein POS-System soll nicht nur den reibungslosen Betrieb der Kassen gewährleisten, sondern möglichst auch Probleme und Fehler aufdecken und Aussagen über das Einkaufsverhalten der Kunden treffen können. Obwohl die TPiSCAN Software keine stammdaten-spezifischen Informationen behandelt, gibt es dennoch eine Reihe geschäftsprozess-relevanter Informationen, deren Analyse und

[26] vgl. Balzert 2005, S. 9-12

[27] vgl. Balzert 2001, S. 144--145

Auswertung sehr sinnvoll ist. Dazu zählen im Wesentlichen die drei Gruppen „Attendant Requests“, „POS Calls“ und „Tender“. Die Attendant Requests stehen für sämtliche Aktionen, bei denen der Kunde oder das System einen Attendant anfordern. Die Gruppe POS Calls deckt dagegen sämtliche Anfragen des TPiSCAN Systems an das POS-System und umgekehrt ab. Tender beinhaltet sämtliche Informationen, welche die Ein- und Auszahlung von Geldmitteln betreffen.

Zur Auswertung wurde für die TPiSCAN Software ein Statistikmodul entwickelt, welches Informationen über die Systembenutzung erfasst. Das Statistikmodul zeichnet die Daten direkt auf der jeweiligen TPiSCAN Lane auf und verfügt dabei über die zwei Modi „Session based“ und „Periodical“. Bei der Session-Statistik werden die Daten zwischen dem Anmelden (Sign-In) und dem folgenden Abmelden (Sign-Off) der Kasse an einer Lane statistisch erfasst. Bei der periodischen Statistik werden die erfassten Daten innerhalb aufeinander folgender Perioden konfigurierbarer Länge in eine Statistik geschrieben.[28]

5.2.2 Architektur und Ist-Prozess

Das Statistikmodul der TPiSCAN Systemsoftware ist direkter Bestandteil einer Installation – eines Run Composites – eines Self-Checkout Terminals. Es ist als eigenes TPiSCAN Modul angelegt und liest seine Konfiguration aus einer Konfigurationsdatei im Konfigurationsordner der TPiSCAN Installation auf der Lane aus (siehe Anhang B3). Je nach Konfiguration werden mehr oder weniger detaillierte Einträge in die Statistik geschrieben.

Das Statistikmodul überwacht die Aufrufe von bestimmten Methoden in TPiSCAN und wertet diese aus. Methodenaufrufe, die demselben Muster (engl. „Pattern“) entsprechen, werden gruppiert und Eigenschaften wie die Anzahl der Aufrufe, die Dauer eines Aufrufes oder ähnliche Attribute werden kumuliert. Sämtliche Durchschnitts-Berechnungen beziehen sich dabei auf die vordefinierte Aufzeichnungsdauer einer Statistik, also eine Session oder eine Periode. Deshalb wird die Statistik auch immer erst mit dem Ende der Aufzeichnungsdauer erstellt.[29]

Die folgende Abbildung stellt die vorhandene Architektur schematisch dar:

[28] vgl. WN TPiSCAN Statistikmodul 2007, S. 4

[29] vgl. WN TPiSCAN Statistikmodul 2007, S. 5-8

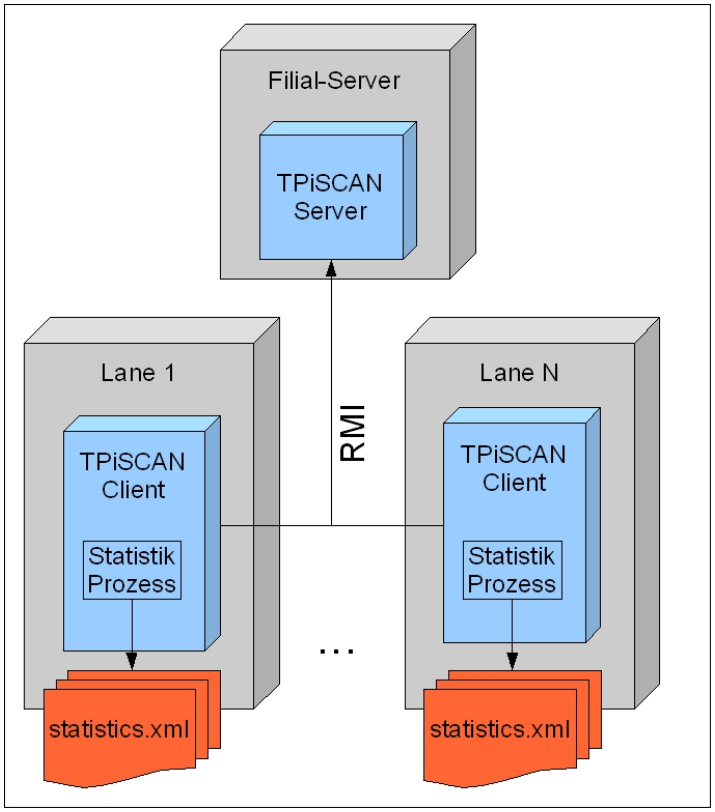

Abb. 4: Architektur des Statistikmoduls (Quelle: eigene Darstellung)

Sämtliche Lanes innerhalb der Filiale kommunizieren mit dem TPiSCAN Server über Remote Method Invocations (RMI), einer Java eigenen Art des Remote Procedure Calls (RPC), um entfernte Java Objekte aufzurufen. Entfernt bedeutet dabei, dass sich das aufzurufende Java Objekt innerhalb einer fremden Java Virtual Machine (JVM) - nämlich in der des Servers - befindet.[30]

Das Statistikmodul erzeugt also je nach Konfiguration auf jeder Lane der Filiale endlich viele Statistik-Dateien. Der Prozess von der Konfiguration des Statistikmoduls bis zur Auswertung der produzierten Statistik-Dateien lässt sich als „erweiterte Ereignis-gesteuerte Prozesskette“ folgendermaßen abbilden:[31]

[30] vgl. Bell Parr 2003, S. 589

[31] vgl. Stahlknecht 2005, S. 236-237

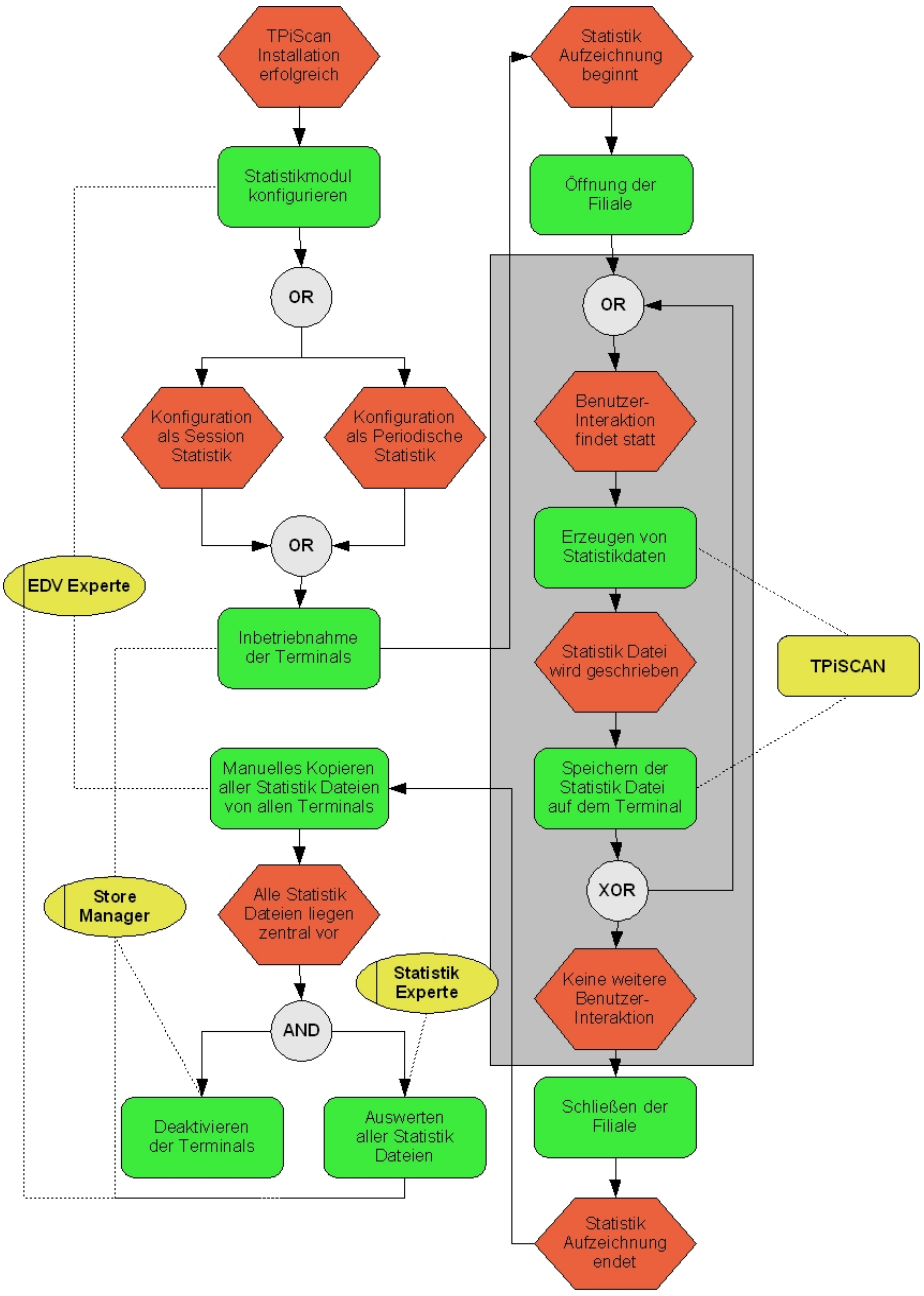

Abb. 5: eEPK des Ist-Prozesses (Quelle: eigene Darstellung)

Nach der Konfiguration des Statistikmoduls durch den EDV-Experten der Filiale, können die Self-Checkout-Terminals vom Filialleiter, dem so genannten Store Manager, in Betrieb genommen werden. Das Aufzeichnen der Statistik-Dateien – innerhalb der eEPK als grauer Kasten dargestellt – findet parallel auf jedem Terminal der Filiale statt.

Um an die Statistik-Dateien auf den Terminals zu gelangen, muss die Filiale geschlossen sein, da ein manueller Zugriff auf die Terminals erfolgen muss. Der EDV Experte kopiert die Daten händisch von den Terminals und stellt sie auf einem separaten Rechner zentral zur Weiterverarbeitung zur Verfügung. Dort kann der

Statistik-Experte die einzelnen Dateien zusammenführen und für den Filialleiter aufbereiten. Dabei muss die Rolle des Statistik-Experten nicht zwangsläufig eine eigenständige Person sein. Es kann sich bei ihm auch um den Filialleiter selbst, oder den EDV-Experten der Filiale handeln.

5.2.3 XML-Ausgabedateien

Eine Statistik wird im XML-Format abgelegt und gespeichert. Die XML-Dateien erhalten durch einen Timestamp eine eindeutige Bezeichnung und werden in einem speziellen Verzeichnis auf der Lane abgelegt.

Jede XML-Datei enthält einen statischen Header und einen dynamischen Teil, in den Informationen zu den drei oben beschriebenen Statistikgruppen POS Calls, Attendant Requests und Tender geschrieben werden.[32] Die Schemadefinition der XML-Datei in Form einer DTD findet sich im Anhang A wieder.

Statischer Header

Der statische Header weist bei jeder geschriebenen XML-Datei die gleiche Struktur auf. Er dient dazu, die geschriebene Statistik eindeutig zu identifizieren, indem Informationen über die Filiale, das Terminal sowie Start- und Endzeit der Aufzeichnung gespeichert werden. Außerdem werden Informationen über die innerhalb der Aufzeichnungszeit stattgefunden Transaktionen am Terminal gespeichert. Als Transaktion wird der Prozess bezeichnet, welcher zwischen der ersten Berührung des Touchscreen Monitors durch den Kunden und der Bezahlung (inklusive Wechselgeld- und Quittungs-Entnahme) stattfindet.

Der statische Header ist innerhalb der XML-Datei nach folgendem Muster aufgebaut:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<statistics storenumber="NNN" terminalnumber="TTT" start="SSS"
startverbose="yyyy/MM/dd hh:mm:ss" end="EEE"
endverbose="yyyy/MM/dd hh:mm:ss" operationmode="OOO"
version="VVV" >
<transactions count="CCC" durationaverage="DDD"
durationaverageverbose="hh:mm:ss.mmm" total="TTT" totalnet="NNN"
assisted="AAA" />
```

Die erste Zeile stellt den Eintrag für den XML Prozessor zum Lesen des XML Dokuments dar. Der Knoten <statistics> beinhaltet Attribute zur Identifizierung der

[32] vgl. WN TPiSCAN Statistikmodul 2007, S. 10-11

Statistik-Datei. Die Filialkennung (storenumber) und die Terminalkennung (terminalnumber) werden ebenso wie der Betriebsmodus (operationmode) und die Versionsnummer (version) des Statistikmoduls als Zeichenketten hinterlegt.

Entscheidend sind hier die Attribute start, end, startverbose und endverbose. Sie ordnen der aufgezeichneten Statistik einen exakten Zeitrahmen zu, auf dessen Basis sämtliche Kumulierungs- und Durchschnitts-Berechnungen stattfinden. Während die Attribute start und end die Zeit in Form eines Unix-Timestamps aufzeichen (Anzahl der Millisekunden seit dem Fixpunkt 01.01.1970 00:00Uhr UTC), beinhalten die Attribute mit dem „verbose"-Zusatz (engl. für „wortreich") einen Zeitstempel in Form eines Datum-Formats (yyyy/MM/dd hh:mm:ss).

Das Element <transactions> zählt die Anzahl (count) der in dem Aufzeichnungszeitraum stattgefundenen Transaktionen sowie deren durchschnittliche Dauer (durationaverage und durationaverageverbose). Der Zähler „assisted" gibt als Untermenge die Anzahl der Transaktionen wieder, welche vom Kunden nicht selbstständig, sondern mit Attendant-Unterstützung durchgeführt wurden. Die Attribute „total" und „totalnet" gehören eigentlich in den Bereich der Geldmittel (Tender). Sie geben den Gesamtumsatz des Terminals innerhalb der Aufzeichnungszeit der Statistik als Brutto- (total) und Nettowert (totalnet) wieder.[33]

POS Calls

Ein Eintrag für einen aufgezeichneten POS Call erfolgt innerhalb der XML-Datei nach dem folgenden Muster:

```
<method name="methodname(attId,param1,param2)" count="CCC"
successful="SSS" meanDuration="TTT" />
```

Neben der Bezeichnung der Interface-Methode (methodname) beinhaltet der Eintrag auch optionale Details (attId, param1 und param2). Die beiden Counter „count" und „successful" beschreiben die Anzahl der POS Calls, welche innerhalb der Aufzeichnunsperiode dem Pattern „methodname(attId,param1,param2)" entsprechen (counter) und, als Untermenge davon, die Anzahl der erfolgreichen Methodenaufrufe (successful). Das Attribut „meanDuration" kennzeichnet die durchschnittliche Laufzeit der gezählten, erfolgreichen POS Calls.[34]

[33] vgl. Anhang A

[34] vgl. WN TPiSCAN Statistikmodul 2007, S. 5-6

Attendant Requests

Analog zu dem Aufzeichnen von POS Calls erfolgt auch das Aufzeichnen von Attendant Requests. Attendant Requests werden jedoch mit zwei Einträgen aufgelistet, da einerseits die aufgetretenen Attendant Requests und andererseits die später tatsächlich bestätigten (und somit bearbeiteten) Attendant Requests separat erfasst werden. Zu den bestätigten Attendant Requests liegen mehr Informationen vor:

```
<attendantrequest id="attreqtype" count="CCC" />
<confirmed-attendantrequest id="attreqtype(attId,param1,param2)"
count="CCC" meanLifeTime="TTT" meanBlockingTime="BBB" />
```

Der erste Eintrag zählt das Auftreten von Attendant Requests mit einer bestimmten Bezeichnung (attreqtype). Der zweite Eintrag hingegen zählt das Auftreten (count) von bestätigten Attendant Requests welche dem Pattern “attreqtype(attId,param1,param2)“ entsprechen. Für diese Einträge werden auch die durchschnittliche Lebenszeit eines Attendant Requests (meanLifeTime) sowie seine durchschnittliche Sperrzeit (meanBlockingTime) eingetragen. Als Sperrzeit wird hier der Zeitraum bezeichnet, in dem ein Attendant Request einen anderen überschneidet und aufgrund seiner höheren Priorität die Bearbeitung des anderen Requests blockiert.

Die Anzahl von bestätigten Attendant Requests (zweiter Eintrag) mit demselben Bezeichner (attreqtype) aber unterschiedlichen Details wird kumuliert im ersten Eintrag dargestellt. Der Zähler (count) im ersten Eintrag kann aber entsprechend höher sein als die Summe der bestätigten Attendant Requests, wenn Requests stattfanden, aber nicht bestätigt wurden.[35]

Tender

Die Gruppe Tender umfasst sämtliche Einträge, welche sich mit der Ein- und Auszahlung von Geldmitteln beschäftigen. Dabei findet innerhalb dieser Gruppe eine weitere Einteilung in Untergruppen statt:[36]

[35] vgl. WN TPiSCAN Statistikmodul 2007, S. 6-8

[36] vgl. Anhang A

<table>
<tr><td colspan="6"><tender></td></tr>
<tr><td colspan="5"><cash></td><td><noncash></td></tr>
<tr><td colspan="5"><u>Attribute:</u>
amountpayed=“PPP“
amountcollected=”CCC”</td><td><u>Attribute:</u>
description=“tendertype “
oder
description=“tendertype(overpayAmount)“
count=“CCC“</td></tr>
<tr><td><coindispenser></td><td><coinacceptor></td><td><notedispenser></td><td><noteacceptor></td><td><cashoutbycustomercancel></td><td><cashoutcausedbyvoid></td></tr>
<tr><td colspan="4"><u>Attribute:</u>
amount=”AAA”
count=”CCC”
denomination=”DDD”
type=”TTT”</td><td colspan="2"><u>Attribute:</u>
count=”CCC”
amount=”AAA”</td></tr>
</table>

Das Element <tender> beinhaltet zwei Kind-Elemente, <cash> und <noncash>. <cash> beschreibt sämtliche Zahlungsvorgänge, bei denen Bargeld zum Einsatz kommt. Innerhalb des Elementes <cash> wird zwischen Einzahlungs-Vorgängen (acceptor) und Auszahlungsvorgängen (dispenser) unterschieden. Da gleichzeitig eine Aufteilung nach Geldmittel-Art (Münze und Schein) stattfindet, ergeben sich insgesamt vier Kind-Elemente von <cash>. Zu diesen vier Elementen werden als Attribute ein Zähler (count), der Betrag (amount), die Geldstückelung (denomination) und der Typ der Geldmittel-Hardware (type) aufgezeichnet. Die Geldstückelung bezeichnet den Wert eines Geldmittels. Eine 2 Euro Münze hat demnach eine Denomination von 200.

Die beiden Elemente <cashoutbycustomercancel> und <cashoutcausedbyvoid> stellen zwei Sonderfälle von <cash> dar. <cashoutbycustomercancel> bezeichnet das Ereignis das eintritt, wenn ein Kunde einen begonnenen Bezahlvorgang mit Bargeld abbricht und das System das eingezahlte Geld wieder auszahlen muss. <cashoutcausedbyvoid> ist eine Barauszahlung, welche stattfindet, weil ein Attendant die Transaktion oder eine bereits verbuchte Bargeld-Teilzahlung storniert.

Das zweite Kindelement von <tender> stellt das Element <noncash> dar. Unter diesem Element werden sämtliche Bezahlungen mit Zahlmitteln außer Bargeld, zum Beispiel per Geldkarten aufgelistet. Das Attribut „description“ beinhaltet eine Bezeichnung des Zahlungsmittels und als optionales Detail den so genannten „Overpay Amount“. Dieser gibt an, ob ein Kunde nicht nur bezahlt, sondern auch noch zusätzlich einen bestimmten Betrag an Bargeld überbezahlt hat und somit ein erhöhtes Wechselgeld ausgezahlt bekommt. Außerdem wird die Anzahl der Bezahlungen eines entsprechenden Zahlungsmittels erfasst. Der Zähler (count) bezieht sich dabei auf Bezahlungen, welche dem Pattern „tendertype(overpayAmount)“ entsprechen. Aus Gründen des „Separation of concerns“ werden Geldbeträge beim Bezahlvorgang mit einer Geldkarte nicht in der Statistik erfasst.

5.2.4 Konfiguration des Statistikmoduls

Im Konfigurationsordner der jeder TPiSCAN Anwendung befindet sich eine Datei mit der Bezeichnung StatisticsConfig_default.xml (siehe dazu Anhang B2). Diese XML-Datei dient zur Konfiguration des Statistikmoduls. So kann pro Kunde individuell bestimmt werden, wie und vor allem was in der Statistik erfasst werden soll. Die Konfiguration erfolgt anhand folgender fünf Parameter:[37]

```
<parameter name="CREATE_PERIODIC_STATISTICS" value="true"/>
<parameter name="PERIOD_LENGTH" value="15"/>
<parameter name="PERIOD_FILENAME_PREFIX" value="statistics"/>
<parameter name="PERIOD_SUFFIX_MODE" value="TIME"/>
<parameter name="SESSION_STATISTICS_ADAPTION" value="false"/>
```

Der erste Parameter legt fest, ob Periodische Statistiken erzeugt werden. Der zweite Parameter legt dann fest, wie groß die einzelnen Aufzeichnungs-Perioden sein sollen. Der Wert wird in Minuten angegeben.

[37] vgl. WN TPiSCAN Statistikmodul 2007, S. 8-9

Mit dem dritten und vierten Parameter kann die Benennung der zu speichernden XML Statistik-Dateien gesteuert werden. Möglich ist die Angabe eines beliebigen Prefix sowie eines Suffix aus der Menge {TIME, DATE, PERIOD}. Der letzte Parameter legt fest, ob eine Synchronisation zwischen der Session-Statistik und der Perioden-Statistik stattfinden soll.

Neben diesen fünf Parametern zur Konfiguration der periodischen Statistik, existieren in der Konfigurationsdatei noch zwei weitere Einträge. Diese legen fest, mit welcher Detailgenauigkeit POS Calls und Attendant Requests aufgezeichnet werden sollen: [38]

```
<posmethod-counting-rule match="methodname"
counterId="formattingpattern" />

<attendant-request-counting-rule match="attendantrequesttype"
counterId="formattingpattern" />
```

Mit diesen Einträgen kann explizit für jeden beliebigen POS Call (methodname) und Attendant Request (attendantrequesttype) definiert werden, mit welchem Detailgrad die Aufzeichnung erfolgen soll. Mit einem * wird die Regel auf alle POS Calls beziehungsweise Attendant Request angewendet.

Die Detailgenauigkeit wird über das Attribut „counterId“ festgelegt. Das Feld „formattingpattern“ entspricht dem Pattern einer java.text.Message, welche sich aus einer Reihung von Platzhaltern in geschweiften Klammern zusammensetzt. Möglich ist zum Beispiel {0}({1},{2},{3}).[39] Dabei stehen die vier numerischen Platzhalter 0,1,2,3 für:

{0} - Methodenname

{1} - Benutzerkennung des Attendants

{2} und **{3}** - zwei generische Parameter des Methodenaufrufs

[38] vgl. WN TPiSCAN Statistikmodul 2007, S. 9-10

[39] vgl. WN TPiSCAN Statistikmodul 2007, S. 10

5.2.5 Schwachstellen

Zur Auswertung der aufgezeichneten Statistikdaten müssen die XML-Dateien direkt auf der jeweiligen Lane geöffnet und in ihrer Rohform als Baumstruktur gelesen werden. Ein Vergleich verschiedener XML-Dateien ist nur eingeschränkt und händisch per Editor oder Browser möglich, ein Vergleich von XML-Dateien verschiedener Lanes wird nicht unterstützt. Deshalb werden die produzierten XML-Dateien meist mit Hilfe von MS Excel oder professionellen Statistik- und Reporting-Programmen von Drittanbietern (wie beispielsweise SPSS oder Crystal Reports) ausgewertet. Dazu ist es jedoch notwendig, manuell von jedem Terminal die XML-Statistik-Dateien zu kopieren und anschließend auf einem Rechner mit entsprechenden Analysemöglichkeiten wieder zusammenzuführen. Dazu müssen dann manuell sämtliche XML-Dateien eingelesen und entsprechend ihrer Aufzeichnungszeit und ihres Entstehungsortes (Terminals) vereinigt werden. Erst dann ist es möglich, per Analyse Grafiken oder Tabellen zur genaueren Interpretation zu erzeugen.

Dieses Vorgehen ist nicht nur sehr zeit- und arbeitsintensiv, sondern birgt auch viele Möglichkeiten für Fehlerquellen, da der Prozess weder standardisiert noch automatisiert ist. Letztendlich ist der Aufwand für den Filialleiter, um an eine aussagefähige Statistik zu kommen, sehr groß. Die Akzeptanz für den Einsatz und die Auswertung der produzierten Statistik-Dateien ist gering, da für den Prozess viele Fachkenntnisse im Bereich EDV und Statistik notwendig sind.

Gesucht wird also nach einer Lösung, welche den Prozess optimiert, automatisiert und vor allem vereinfacht!

5.3 Anforderungserhebung

5.3.1 Einleitung

Zweck

Die Erhebung von Anforderungen im Rahmen des Requirement Engineering ist ein entscheidender und zum Teil sehr komplexer Prozess, da der Entwurf einer Software direkt vom Ergebnis der Erhebung abhängig ist.[40] Deshalb wurde der Prozess vom Institute of Electrical and Electronic Engineers (IEEE) in der Software Requirements Specification (SRS) standardisiert.[41] Die SRS unterteilt sich in die beiden Bereiche Customer-Requirements, welche eine betriebswirtschaftliche Sicht darstellen und Development-Requirements, welche eine technische Sicht auf die Software beschreiben. Die folgende Anforderungserhebung versucht den empfohlenen Standard weitestgehend zu befolgen.

Ziel

Die zu entwickelnde Anwendung soll dazu dienen, die vom bestehenden TPiSCAN Statistikmodul produzierten Statistikdaten zu visualisieren. Dazu soll der Benutzer über Abfragebedingungen die Art der Darstellung beeinflussen können.

5.3.2 Allgemeine Beschreibung

Produkt-Perspektive

Die zu entwickelnde Anwendung wird im Rahmen der TPiSCAN Systemsoftware für Self-Checkout-Terminals von Wincor Nixdorf entwickelt. Das bedeutet, dass die Anwendung in die bestehende, zuvor beschriebene Systemlandschaft integriert werden muss. Dabei darf und kann die Anwendung mit der bestehenden TPiSCAN Applikation kommunizieren, da sie als Modul selbiger entwickelt wird. Sämtliche bereits bestehende Mechanismen, beispielsweise für das Auslesen von Konfigurationsdateien, können verwendet und erweitert werden. Im Rahmen der TPiSCAN Software-Umgebung stehen eine MySQL-Datenbank und ein Apache Tomcat Embedded Webserver zur Verfügung, welche verwendet werden können.

[40] vgl. Alpar 2002, S. 283-287

[41] vgl. IEEE SRS 1998

Produktfunktionen

Die Anwendung hat das Ziel, den bestehenden Ist-Prozess zur Erfassung und Auswertung der produzierten Statistikdaten zu verbessern. Damit ist neben einer Automatisierung des Prozesses vor allem eine Verbesserung der Prozess-Sicherheit und Prozess-Geschwindigkeit gemeint. Es soll für den Benutzer mit möglichst geringem Aufwand und ohne Fachkenntnisse im Bereich der Informatik oder Statistik möglich sein, zu einer übersichtlichen und aussagekräftigen Statistik zu gelangen.

Für den Benutzer muss es möglich sein, ohne die Installation zusätzlicher Software, von jedem beliebigen Computer im Filialnetzwerk eine aktuelle Statistik für jedes Self-Checkout Terminal innerhalb der Filiale zu erhalten. Haupteinsatz der Applikation soll jedoch nicht die Darstellung aktuell produzierter Statistikdaten sein, sondern vielmehr die eher seltene aber regelmäßige Visualisierung historischer Statistiken.

Benutzermerkmale

Die Zielgruppe für die Benutzung der Applikation setzt sich im Wesentlichen aus den Mitarbeitern der Filiale zusammen. Dabei handelt es sich entweder um einen Filial-Mitarbeiter oder den Filial-Leiter. In der Regel wird die statistische Auswertung eines historischen Zeitraums aber vom Filial-Leiter vorgenommen. Es ist davon auszugehen, dass weder die Mitarbeiter der Filiale, noch der Filial-Leiter selbst über Fachkenntnisse im Bereich der Statistik oder der EDV verfügen. Deshalb muss die Visualisierung der Statistikdaten vor allem einfach und verständlich sein.

Einschränkungen

Die Anwendung baut direkt auf dem bereits vorhandenen TPiSCAN Statistikmodul auf. Dadurch ergeben sich automatisch Einschränkungen, da nur solche Statistikdaten ausgewertet und visualisiert werden können, die auch zuvor vom TPiSCAN Statistikmodul erfasst wurden.

Die wichtigsten Einschränkungen bestehen darin, dass die Erfassung der Statistikdaten nicht in Form einer Einzeldatenerfassung stattfindet, sondern vom TPiSCAN Statistikmodul eine Statistik produziert wird, in der die Daten zu Perioden zusammengefasst werden.[42] Außerdem basiert die Statistik auf einer technischen Sicht auf die stattgefundene Benutzerinteraktion mit dem Self-Checkout Terminal. Es werden keinerlei Artikel spezifische Daten (also Stammdaten) in der Statistik erfasst.

[42] vgl. WN TPiSCAN Statistikmodul 2007, S. 4

Annahmen und Abhängigkeiten

Die Applikation wird nicht entwickelt, um ein vollwertiges Programm zur Statistikdaten-Auswertung zu ersetzen. Es soll vermieden werden, den Benutzer mit für ihn unverständlichen statistischen Kennzahlen zu konfrontieren. Vielmehr ist das Ziel, mit Hilfe der Applikation das Auslesen der vom TPiSCAN Statistikmodul produzierten XML-Statistik-Dateien zu ersetzen und eine bessere Interpretierbarkeit der Daten zu erreichen.

Für spätere Versionen ist die Implementierung von darüber hinausgehenden Funktionen durchaus sinnvoll. Dazu könnten beispielsweise der Datenexport, eine konfigurierbare Reporting-Funktion oder Mitarbeiter spezifische Abfragen gehören. Letztere sind aus rechtlichen Gründen derzeit nicht vorgesehen. Die Anwendung soll so entworfen und implementiert werden, dass die angesprochenen Erweiterungen einfach durchführbar sind.

5.3.3 Spezifizierte Anforderungen

funktionale Anforderungen

Die funktionalen Anforderungen beschreiben, **was** die Applikation macht. Die nachfolgende Liste dokumentiert die an die Applikation gestellten funktionalen Anforderungen und entstand durch Gespräche mit den Entwicklern von Wincor Nixdorf:

- Die Anwendung soll den vorhandenen Ist-Prozess der Statistikdaten-Auswertung ersetzen.
- Die Anwendung soll die Statistikdaten in einer visuell leicht erfassbaren Form aufbereiten.
- Die Anwendung soll von jedem Rechner im Filialnetzwerk aus erreichbar sein.
- Es soll für den Benutzer möglich sein, mit Hilfe der Anwendung Statistikdaten eines bestimmten Zeitraumes abzufragen.
- Es soll für den Benutzer möglich sein, mit Hilfe der Anwendung die Statistikdaten eines bestimmten oder aller Self-Checkout Terminals in der Filiale abzufragen.
- Die Anwendung soll ohne Einarbeitungszeit benutzbar und leicht verständlich sein.
- Die Anwendung soll Mehrsprachigkeit unterstützen.
- Das Aussehen der Anwendung muss vom Benutzer voll anpassbar sein.

nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen einer Anwendung zeigen, **wie** etwas innerhalb der Applikation gemacht werden soll. Die nachfolgende Liste dokumentiert die an die Applikation gestellten nicht-funktionalen Anforderungen und entstand ebenfalls durch Gespräche mit den Entwicklern von Wincor Nixdorf:

- Die Anwendung baut auf dem vorhandenen TPiSCAN Statistikmodul auf.
- Statistikdaten werden visuell in Form von Tabellen und Diagrammen dargestellt.
- Benutzerabfragen werden über ein Formular entgegengenommen.
- Die Anwendung soll Benutzeranfragen ausreichend schnell (<1min.) beantworten.
- Die Anwendung muss ohne eine Installation von zusätzlicher Software ausführbar sein und wird als Teil der TPiSCAN Software ausgeliefert.
- Die Anwendung muss in Java implementiert werden.
- Der Benutzer darf beim Anpassen des Aussehens keinen Java Code verändern.

externe Schnittstellen

Durch die Integration der Applikation in die TPiSCAN Software von Wincor Nixdorf, kann die Anwendung über Schnittstellen auf sämtliche innerhalb von TPiSCAN vorhandene Module zugreifen. Dies betrifft vor allem den Zugriff auf den eingebetteten Apache Tomcat Webserver und globalen Konfigurationsdateien. Des Weiteren besteht über einen JDBC Treiber eine Schnittstelle zu dem extern vorhandenen MySQL-Datenbanksystem.

Design Constraints

Da die Applikation in die bestehende TPiSCAN Systemlandschaft integriert werden soll, besteht die Anforderung, die Entwicklung der Anwendung in Java durchzuführen. Dabei muss immer die in TPiSCAN verwendete Version von Java verwendet werden. Sämtliche Konfigurationsdateien müssen im Format XML vorliegen, damit der vorhandene Konfigurations-Mechanismus von TPiSCAN auf diese Dateien zugreifen kann. Des Weiteren muss die Architektur die innerhalb von TPiSCAN bestehende Trennung von benutzer-spezifischen Daten und Core-Daten unterstützen.

Das bedeutet, dass sämtliche Konfigurationsmöglichkeiten, welche die Erscheinung – also die Benutzungsoberfläche – der Anwendung betreffen, aus dem Java Code

ausgelagert sein müssen. In der Konsequenz bedeutet diese Vorgabe eine strikte Einhaltung des Model-View-Controller Patterns (MVC).[43]

Anforderungen an die Performance

Die Anwendung ist nicht dazu gedacht, die produzierten Statistikdaten in Echtzeit zu Visualisieren. Das wird allein schon dadurch verhindert, dass die Statistikdaten periodisch erfasst werden. Stattdessen ist das Haupteinsatzgebiet die seltenere Abfrage von historischen Statistikdaten zur Analyse von Fehlern oder Unregelmäßigkeiten. Dazu ist es unter Umständen notwendig, dass sehr großen Datenmengen durchsucht und zusammengefasst werden müssen. Die Performance ist dann als positiv anzusehen, wenn die Darstellung der Statistik schneller erfolgt, als der zuvor vorhandene Ist-Prozess benötigte.

Tatsächlich sollte aber aus Gründen der Gebrauchstauglichkeit und Ergonomie eine Antwortzeit der Anwendung von wenigen Sekunden bis hin zu maximal einer Minute pro Benutzerabfrage gewährleistet werden.[44]

Qualitätsanforderungen

Neben der Qualität der eigentlichen Visualisierung, welche die Oberfläche der Applikation als Schnittstelle zum Benutzer ausmacht, gilt es genauso, eine gewisse Qualität des Java-Codes einzuhalten. Darunter lassen sich alle Aspekte eines guten Programmier-Stils einschließlich der Einhaltung gängiger Konventionen und umfassender Dokumentation zusammenfassen.

Speziell sollen die in der Norm DIN 66272 spezifizierten dynamischen Qualitätsmerkmale zur Bewertung von Softwareprodukten (Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit) sowie statische Qualitätsmerkmale wie Programmstruktur, Programmkomplexität und Kommentarumfang beachtet werden.[45]

[43] vgl. Bell Parr 2003, S. 419-421

[44] vgl. Stahlknecht 2005, S. 311-312

[45] vgl. Stahlknecht 2005, S. 310-311

6 Aspekte des Entwurfs

6.1 Soll-Prozess zur Erfassung von Statistikdaten

Die Problematik bei der Auswertung der vom TPiSCAN Statistikmodul produzierten Statistik-Dateien besteht in der Interpretation der Statistikdaten. Damit eine solche möglich ist, müssen die Daten mit einer externen Software korrekt zusammengeführt und abgeglichen werden. Dieser Arbeitsschritt und auch die folgende Interpretation des vereinigten Datenbestandes erfordern explizite Fachkenntnisse im Bereich der EDV und der Statistik. Der notwendige Prozess ist zeitlich viel zu aufwendig und birgt sehr viele mögliche Fehler, welche durch falsches Zusammenführen der Einzeldaten entstehen können.

Zur Lösung des Problems soll eine eigenständige Applikation entworfen werden, welche die Interpretation der Daten aus TPiSCAN heraus ermöglicht. Dazu muss als notwendige Vorarbeit die automatische Zentralisierung des Datenbestandes realisiert werden. Bisher müssen die Statistik-Dateien per Hand kopiert und manuell wieder physisch zusammengeführt werden. Dieses Vorgehen ist zeitlich weder praktikabel, noch sicherheitstechnisch empfehlenswert, da der Prozess zu viele mögliche Fehlerquellen mit sich bringt. Die automatische Zentralisierung des Datenbestandes, als erster Schritt zur Realisierung einer Applikation zur Auswertung der Statistikdaten, soll durch einen neuen Prozess implementiert werden.

Zu beachten ist, dass der neue Prozess den bisher bestehenden Ist-Prozess nicht ersetzt, sondern stattdessen parallel dazu implementiert wird. Dazu können Teile des bestehenden Ist-Prozesses einfach übernommen werden. Der neue Prozess erfordert jedoch eine separate Konfiguration für das Statistikmodul (siehe Anhang B3) und eine getrennte Datenhaltung für die XML Dateien. Dies berücksichtigt die Tatsache, dass einige Kunden ihre Geschäftsprozesse bereits an dem bestehenden Ist-Prozess ausgerichtet haben. Eine Änderung an dem Ist-Prozess würde automatisch einen zusätzlichen Aufwand für den Kunden bedeuten. Durch die Parallel-Implementierung kann der Kunde seine bestehenden Geschäftsprozesse behalten und bei Interesse zu einem beliebigen Zeitpunkt eine Migration zum neu angebotenen Prozess vornehmen.

Die folgende Abbildung stellt den Soll-Prozess bei der Erzeugung von Statistik-Dateien auf den Self-Checkout Terminals dar.

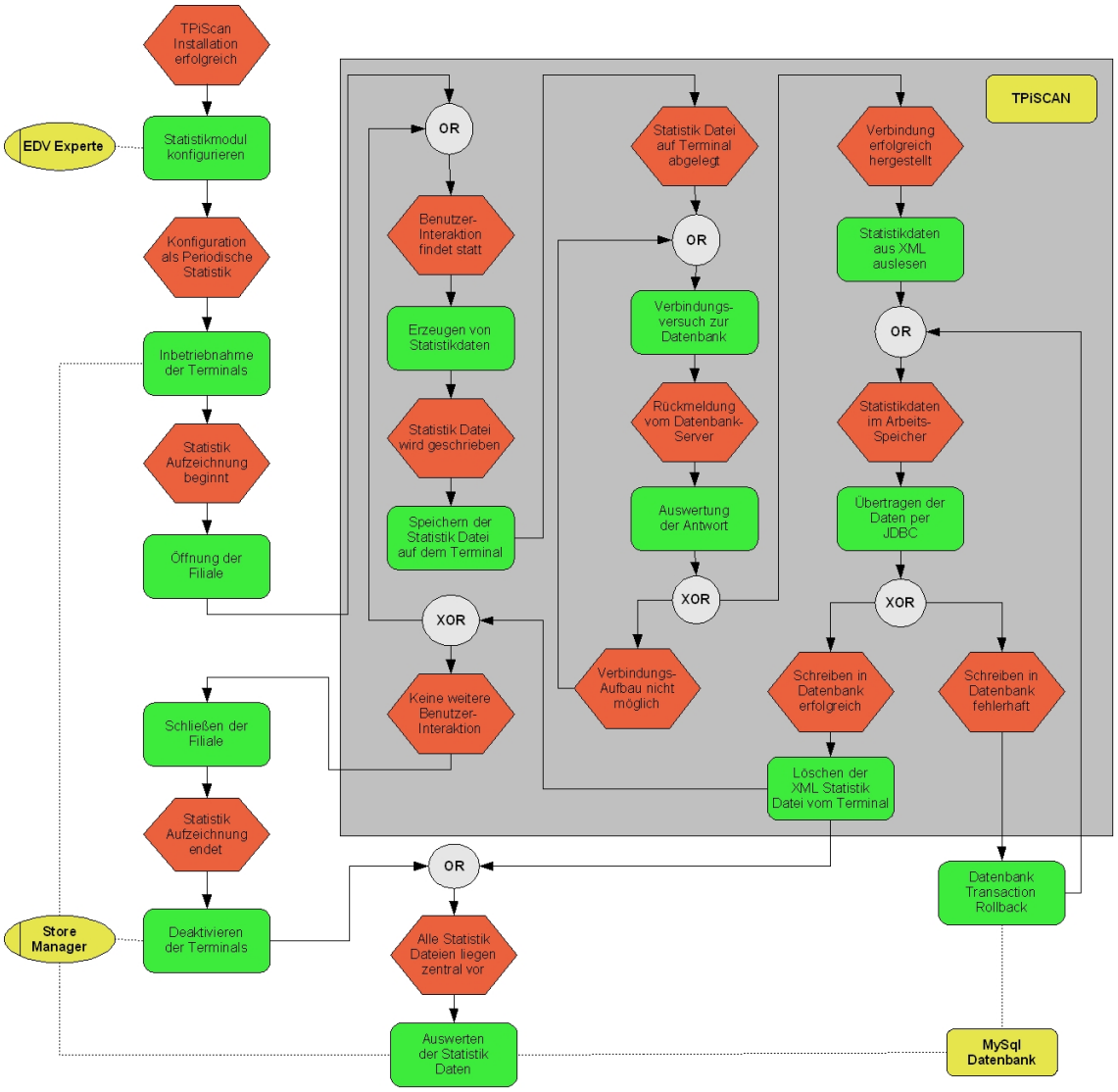

Abb. 6: eEPK des Soll-Prozesses (Quelle: eigene Darstellung)

Die grundlegende Änderung besteht darin, dass die erzeugten XML-Statistik-Dateien nicht mehr auf dem Terminal gespeichert werden, bis sie irgendwann zu einem beliebigen Zeitraum wieder manuell gelöscht werden. Stattdessen werden die Statistikdaten nur noch vorübergehend in der XML-Datei hinterlegt. Sobald eine XML-Datei geschrieben wurde, werden die Daten per JDBC in eine MySQL-Datenbank geschrieben.

Tritt bei dieser Transaktion[46] ein Fehler auf, wird die Transaktion rückgängig gemacht (Transaction Rollback) und der Schreibvorgang auf die Datenbank wird erneut gestartet. Erst wenn die Daten erfolgreich in der Datenbank hinterlegt wurden, wird die

[46] vgl. Heuer Saake 2000, S. 497

XML-Statistik-Datei auf dem Terminal gelöscht. Durch dieses Vorgehen ist eine Transaktionssicherheit beim Scheiben auf die Datenbank gewährleistet. Der grau hinterlegte Teil der eEPK kennzeichnet den Teil der Prozesskette, welcher innerhalb der TPiSCAN Anwendung auf jedem Self-Checkout Terminal abläuft. Die Datenbank wird also parallel von sämtlichen Lanes aus mit Daten gefüllt.

Ein weiterer positiver Aspekt dieser Vorgehensweise ergibt sich aus der Tatsache, dass nun jedes Mal, nachdem eine Statistik-Datei auf einem Terminal geschrieben wurde, die enthaltenen Daten umgehend über die Datenbank zugänglich sind. Es ist also nicht mehr notwenig, die Daten manuell von den Terminals zu kopieren und dazu die Terminals für den Kunden zu blockieren oder gar bis zum Filialschluss zu warten. Die Daten können sofort aus der Datenbank ausgelesen werden.

Als weitere wesentliche Änderung für den neuen Prozess, wird anfangs bei der Konfiguration des Statistikmoduls auf die Erzeugung von Session-Statistiken verzichtet. Um die Datenbank mit aussagekräftigen Statistikdaten zu füllen, ist es sinnvoll, auf periodische Statistiken mit äquidistanten Aufzeichnungszeiträumen und maximaler Detailgenauigkeit zurück zu greifen. Diese lassen sich später einfacher miteinander vergleichen und verknüpfen, als es mit Session-Statistiken der Fall wäre. Die Konfiguration des Statistikmoduls ist in Anhang B3 abgebildet und wurde im Kapitel 5.2.4 beschrieben.

Für die Auswertung der in die Datenbank geschriebenen Daten ist eine Applikation notwendig, deren Architektur im folgenden Kapitel erläutert wird.

6.2 Architektur

Die Applikation wird als Web-Anwendung realisiert, um der Anforderung gerecht zu werden, von einem beliebigen Client Rechner aus ohne die Installation zusätzlicher Software mit der Anwendung arbeiten zu können. Unter einer Web-Anwendung versteht man eine Applikation, bei der die Benutzer-Interaktion ausschließlich über einen Webbrowser stattfindet und die Anwendung selbst auf einem Webserver ausgeführt wird.

Webbrowser gehören mittlerweile zur Standard-Software eines jeden Betriebssystems und müssen deshalb in der Regel nicht extra installiert werden. Somit ist es für einen Benutzer sehr leicht möglich, ohne zusätzlichen Aufwand mit der Webanwendung zu arbeiten. Dazu ist es lediglich notwendig, die URL der Anwendung auf dem Webserver zu kennen.

Die folgende Abbildung stellt die Client-Server-Kommunikation einer Webanwendung schematisch dar:

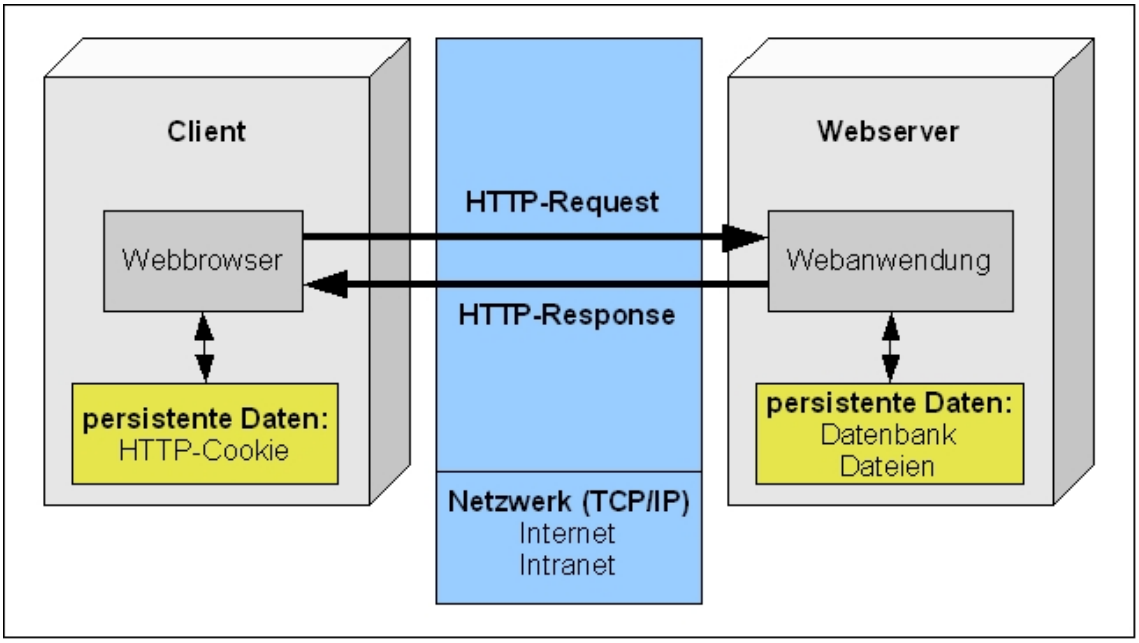

Abb. 7: Webanwendung schematisch
(Quelle: eigene Darstellung, in Anlehnung an Wiki Web 2007)

Der Anwender ruft dabei per URL im Browser ein Programm auf dem Webserver auf. Diese Anfrage wird als HTTP-Request über das Netzwerk an den Server gesendet. Dieser verarbeitet die Anfrage indem er auf persistente Daten wie eine Datenbank oder Dateien zurückgreift. Das Ergebnis der Client-Anfrage wird als HTTP-Response vom Server über das Netzwerk zurück an den Client gesendet. Dort wird das gesendete Ergebnis innerhalb des Webbrowsers dargestellt. Daten können clientseitig in einem HTTP-Cookie oder in einer HTTP-Session persistent gespeichert werden. Die Kommunikation zwischen Client und Server findet über das Netzwerkprotokoll TCP/IP statt.[47]

[47] vgl. Gourley Totty HTTP 2002, S. 4-14

Adaption des Konzeptes

Das bewährte Konzept der Webanwendung lässt sich ohne Probleme auf das eingangs skizzierte Szenario im Bereich Self-Checkout anwenden. Dazu wird die im Kapitel 5.2.2 beschriebene vorhandene Architektur erweitert.

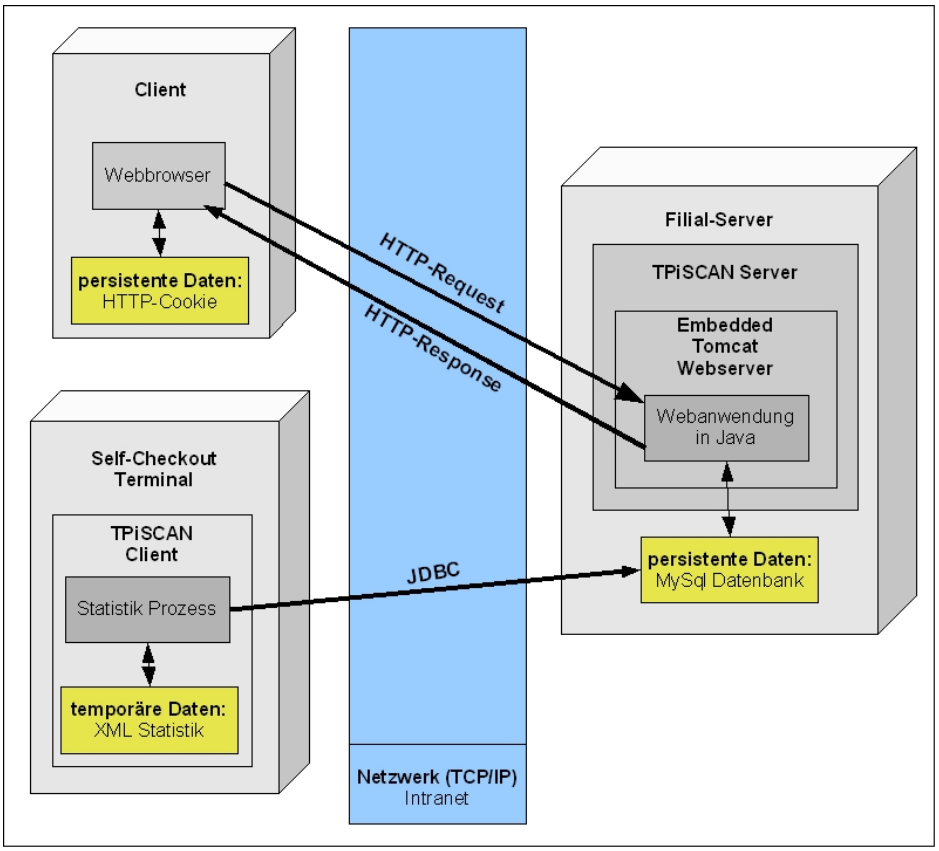

Abb. 8: Anwendungsarchitektur (Quelle: eigene Darstellung)

Auf dem MySQL-Datenbankserver wird eine neue Datenbank für die Statistikdaten angelegt. Auf die Datenbank kann vom TPiSCAN Server aus lokal und von allen Lanes aus über das vorhandene Intranet zugegriffen werden. Während die Lanes die anfallenden Statistikdaten periodisch in die Datenbank schreiben, nutzt der Webserver die Datenbank als persistenten Speicher, mit Hilfe dessen er auf Benutzeranfragen antworten kann.

Apache Tomcat Webserver

Für das Ausführen einer Webanwendung wird ein Webserver benötigt. Dazu wird ein Apache Tomcat Webserver der Apache Software Foundation eingesetzt. Apache Tomcat stellt einen so genannten Servlet-Container zur Verfügung, also eine

Umgebung, in der Java-Code innerhalb eines Webservers ausgeführt werden kann.[48] Der Tomcat Webserver wird als Modul in den TPiSCAN Server integriert und dort als Embedded Webserver ausgeführt. Die Bezeichnung „embedded“ (dt. „eingebettet“) beschreibt die Tatsache, dass der Webserver nicht als allein stehende Anwendung ausgeführt und betrieben wird, sondern innerhalb einer anderen Anwendung verborgen gestartet wird.

Mit Hilfe des Apache Tomcat Webservers können HTTP-Anfragen nicht mehr nur statisch, sondern durch die Java-Integration auch dynamisch beantwortet werden. Das bedeutet, dass auf der Serverseite ein Java Programm der HTTP-Request auswertet und anschließend dynamisch einen HTTP-Response zusammenbaut. Dieses Java-Programm muss innerhalb des Apache Tomcat Webservers deployed (dt. „ausgeliefert“) werden, also dort explizit als vorhandene Webanwendung angemeldet und konfiguriert werden, um dann per URL von außen erreichbar zu sein. Das Deployment der Webanwendung nimmt der Apache Tomcat Webserver selbstständig vor.[49]

Der Apache Tomcat Server wird innerhalb einer anderen Applikation über die Klasse org.apache.catalina.startup.Embedded gestartet. Um den Webserver korrekt zu starten, müssen die Klassenmethoden in einer vorgeschrieben Reihenfolge aufgerufen werden. Auf diesem Weg wird auch die Information übergeben, welche Webanwendungen innerhalb des Webservers zur Verfügung stehen. Dazu wird einerseits der Kontextpfad – also die URL – der Anwendung übergeben und andererseits dem Webserver das Verzeichnis mitgeteilt, mit dem die Webanwendung deployed werden soll.[50]

Üblicherweise befinden sich bei einem Tomcat Webserver die HTML-Dateien und Java-Klassen innerhalb einer gemeinsamen Verzeichnis-Struktur.[51] Da TPiSCAN aber auf die Trennung von Kunden- und Core-Daten setzt, wird auch im Webserver der Deployment-Pfad aus verschiedenen physischen Verzeichnissen zusammengesetzt. Dies wird in der Konfigurationsdatei EmbeddedTomcatConfig_default.xml vorgenommen (siehe dazu Anhang B1).

---

[48] vgl. Apache Tomcat 2007

[49] vgl. Apache Tomcat 5.0 Deployment

[50] vgl. Apache Tomcat API 5.0 Embedded

[51] vgl. Apache Tomcat 5.0 Deployment

Die eigentlichen Informationen, wie der Webserver mit der Webapplikation umgehen soll, finden sich im „Web Application Deployment Descriptor“ wieder. Dies ist eine XML Konfigurationsdatei mit der Bezeichnung web.xml, welche innerhalb des Deployment-Pfades liegt.[52] Im folgenden Kapitel werden die Technologien vorgestellt, mit deren Hilfe die Webanwendung innerhalb des Apache Tomcat Webservers realisiert werden soll.

6.3 Technologien

6.3.1 MySQL und Apache Derby

Wie bereits beschrieben, benötigen der neu entworfene Soll-Prozess und die zu entwickelnde Web-Applikation eine Datenbank, in welcher die Statistikdaten zentralisiert gespeichert werden. Dazu bietet sich die Nutzung des bereits im TPiSCAN Umfeld vorhandenen relationalen DBMS **MySQL** an.

Laut Aussage des Herstellers MySQL AB, ist das gleichnamige Produkt MySQL „*The world's most popular open source database*“[53], also die populärste Open-Source-Datenbank der Welt. Ohne Zweifel hat sich die Datenbank seit ihrer ersten Freigabe im Jahr 1994 aber zu einem Quasi-Standard im Bereich der relationalen Datenbanken im Web-Umfeld entwickelt. Das liegt einerseits daran, dass mittlerweile in der Version 5 für ein Open-Source-Produkt erstaunlich viel Funktionalität und Performance geboten wird und andererseits durch die stetig wachsende Community auch der Support des Produktes sehr gut ist.

Richtig bekannt geworden ist MySQL dann durch den Einsatz im so genannten LAMP Paket – also der Kombination von Linux, Apache, MySQL und PHP.[54] Es sollte klar sein, dass es MySQL nicht mit der Performance großer kommerzieller DBMS wie Oracle oder DB2 aufnehmen kann. Nichtsdestotrotz eignet es sich sehr gut für den Einsatz in kleinen bis mittelgroßen Webanwendungen.

Ein großer Nachteil der MySQL-Datenbank besteht darin, dass sie unter der GPL – der GNU General Public License – lizenziert wird. Das bedeutet, dass sämtliche Software, welche mit der MySQL-DB arbeitet und diese kostenlos nutzen will, ebenfalls mit der

[52] vgl. Apache Tomcat 5.0 Deployment

[53] vgl. MySQL Website 2007

[54] vgl. O’Reilly OnLAMP 2001

GPL lizenziert werden muss. Dies ist bei kommerziellen Produkten natürlich nicht praktikabel. Deshalb bietet der Hersteller MySQL AB ein duales Lizenzsystem an, bei dem für den kommerziellen Einsatz von MySQL bezahlt werden muss.[55] Damit geht aber der wesentliche Vorteil der Kostenersparnis beim Einsatz von Open-Source Software verloren.

Ein weiterer zu beachtender Nachteil ergibt sich aus der verteilten Entwicklung, wie sie bei Wincor Nixdorf praktiziert wird. Jeder Software Entwickler soll im Entwicklungsprozess der TPiSCAN Software vollen Zugriff auf sämtliche Resourcen haben, um selbstständig entwickeln und testen zu können. Das bedeutet, dass jeder Entwickler entweder eine lokale Installation von MySQL benötigt oder alle Entwickler auf ein und derselben MySQL-Datenbank arbeiten müssten. Da beides jedoch nicht besonders praktisch ist, wurde nach einer Lösung gesucht, die es ermöglicht, ohne Installation ein komplettes DBMS über den SVN-Software-Verteilungsprozess von Wincor Nixdorf für alle Entwickler zur Verfügung zu stellen.

Die Lösung heißt **Apache Derby**. Dabei handelt es sich um ein Subprojekt der Apache DB, welche von der Apache Software Foundation betrieben wird. Die Datenbank wird unter der Apache License in der Version 2.0 vertrieben und ist daher auch Open-Source Software. Wesentliche Features der relationalen Datenbank sind die vollständige Implementierung in Java, die äußerst geringe Paketgröße (2MB für Engine und Treiber) und die Bereitstellung eines „Embedded JDBC“-Treibers. Dieser Embedded-Treiber ermöglicht es, die Apache Derby Datenbank ohne Installation, sondern lediglich durch die Integration der Derby-Programmbibliotheken, voll funktionsfähig in die eigentliche Java-Anwendung einzubinden.[56] Somit ist es möglich, die Datenbank ohne Installation an alle Entwickler zu verteilen.

Da MySQL und Apache Derby in Sachen Syntax und Funktionalität weitestgehend kompatibel sind, wird eine zweigleisige Datenbank-Lösung implementiert. Beim Start des Apache Tomcat Webservers überprüft TPiSCAN automatisch, ob die Anwendung auf einem Entwicklungsrechner oder im Realeinsatz gestartet wird. Wird die Anwendung im Entwicklermodus – dem so genannten „devStation Modus“ – gestartet, wird die integrierte Apache Derby Datenbank initialisiert und ist dann per embedded JDBC-Treiber ansprechbar. Findet der Start jedoch im Realeinsatz statt, wird die

[55] vgl. MySQL Licensing 2007

[56] vgl. Apache Derby 2007

Derby-Datenbank nicht initialisiert, sondern eine Verbindung zur MySQL-Datenbank auf dem Server aufgebaut.

Die Konfiguration der beiden Datenbanken wird in den Konfigurationsdateien StatisticDBConfig_default.xml und StatisticDBConfig_devStation.xml hinterlegt, welche beim Deployment der Webanwendung ausgelesen werden. Die beiden Dateien enthalten Informationen über den zu verwendenden JDBC-Treiber, die Datenbank-URL und die Benutzer-Authentifizierung (siehe dazu Anhang B2).

6.3.2 Apache Tapestry

Eine Webanwendung kann in Java entwickelt werden, indem dazu die beiden zur Verfügung stehenden Ansätze des Java Servlets oder der Java Server Pages (JSP) benutzt. Beide Ansätze unterscheiden sich grundsätzlich voneinander, da in einer JSP-Seite Java-Code in eine HTML-Seite eingebettet wird, während ein Servlet eine komplett in Java geschriebene Klasse ist, welche lediglich HTML-Code in Form eines HTTP-Responses ausgibt. Beide Ansätze haben das gemeinsame Ziel, Client-Anfragen (HTTP-Requests) entgegenzunehmen und dynamisch zu beantworten.[57]

Der Aufbau einer komplexen Webapplikation nur unter Verwendung dieser beiden Techniken hat allerdings einen entscheidenden Nachteil. Anstatt sich voll und ganz der Implementierung der Business-Logik widmen zu können, muss der Entwickler sehr viel Zeit in das so genannte „Request Handling“ investieren. Damit ist bei einer Webanwendung der Prozess gemeint, welcher steuert, welche Klassen in einem bestimmten Zusammenhang zueinander stehen und interagieren, um eine Benutzeranfrage zu beantworten. Gerade bei großen Webanwendungen kann dieser Prozess sehr komplex werden und die entsprechende Implementierung sehr viel Zeit in Anspruch nehmen.

Deshalb bedient man sich heutzutage bei der Entwicklung von Webanwendungen häufig so genannter Web-Frameworks. Ein Framework stellt eine Reihe von abhängigen Klassen samt API zur Verfügung, welche eine Anwendungsarchitektur bei der Entwicklung vorgeben.[58] Insbesondere definiert das Framework Schnittstellen der Klassen und behandelt selbstständig den Kontrollfluss innerhalb der Anwendung. So

[57] vgl. Ship Tapestry 2004, S. 5-6

[58] vgl. Ship Tapestry 2004, S. 16-17

wird die konsequente Wiederverwendung einer einmal erdachten Anwendungsarchitektur gewährleistet.[59]

Für die konkrete Implementierung der Webanwendung in TPiSCAN wird ein Web-Framework benötigt, das einerseits selbstständig die Benutzerinteraktion bei der Client-Server-Kommunikation steuert und andererseits den Anforderungen nach Mehrsprachigkeit und ausgereifter Kundenkonfiguration nachkommt. Diese Anforderungen werden von **Apache Tapestry** erfüllt.

Apache Tapestry ist ein Open-Source Framework für die Entwicklung von dynamischen Webanwendungen in Java und wird unter der Apache Software License vertrieben. Dabei baut Tapestry auf dem Standard-Java-Servlet-API auf und ist deshalb in beliebigen Servlet-Containern, wie dem Apache Tomcat Webserver, lauffähig. Tapestry wird zu den komponentenorientierten[60] Frameworks gezählt, da es eine Webanwendung intern in ein Set von Webseiten aufteilt, welche aus wieder verwendbaren Komponenten zusammengebaut werden.[61] Mit diesem Konzept ist es Tapestry möglich, selbstständig so komplexe Dinge wie das URL-Handling, das persistente Speichern von Informationen auf Client- und Server-Seite oder die Benutzereingabe-Validierung und Internationalisierung zu behandeln.[62]

Tapestry bedient sich HTML-Templates, welche über XML Konfigurationsdateien mit Komponenten ausgestattet werden. Die HTML-Templates bestehen aus reinem HTML-Code und können dementsprechend mit beliebigen Editoren und minimalen Kenntnissen verändert und angepasst werden. Die Besonderheit besteht aber in der Zuweisung der Komponenten durch die XML-Konfigurationsdateien. Das Tapestry Framework beinhaltet mehr als fünfzig fertige Komponenten, welche von einfachsten HTML Ein- und Ausgabe-Elementen bis hin zu komplexen Elementen wie Navigationsbäumen reichen.[63] Durch das Zuweisen einer Komponente zu einem HMTL-Template, wird die Komponente dynamisch zur Laufzeit in die HTML-Seite eingebunden. Besonders interessant für die Entwicklung von Anwendungen ist Tapestry durch die sehr konsequente Umsetzung eines objektorientierten

[59] vgl. Stahlknecht 2005, S. 324

[60] vgl. Mertens 2004, S. 159-160

[61] vgl. Ship Tapestry 2004, S. 17-18

[62] vgl. Apache Tapestry 2007

[63] vgl. Apache Tapestry 4.0 Components

Entwicklungsansatzes. Damit die Komponenten dynamisch aufgebaut und mit Inhalt gefüllt werden können, lässt sich über die XML-Konfigurationsdatei auch ein Java-Objekt zuweisen, welches Methoden zur Konstruktion der Komponenten beinhalten kann.[64]

Dieses Konzept – also die strikte Trennung von Layout und Inhalt im Sinne des MVC Patterns – unterstützt auch die von Wincor Nixdorf in TPiSCAN geforderte Trennung von kundenspezifischen Daten und Core-Daten.[65]

Tapestry ist ein sehr komplexes Framework, welches den Anspruch hat, dem Entwickler möglichst viel Arbeit abzunehmen. Daher wurden sehr viele Bestandteile von Drittanbietern integriert, was es notwendig macht, sehr viele Bibliotheken in die Webanwendung einzubinden zu müssen. Dies umfasst unter anderem die eigenständigen Projekte HiveMind, Apache Commons, JUnit und OGNL, um nur die Wichtigsten zu nennen.[66] Zusammen mit der Tatsache, dass das Tapestry Framework nur unzureichend dokumentiert ist und wenige Beispielanwendungen existieren, können sich so bei der Einarbeitung einige Probleme ergeben.

6.3.3 JFreeChart

Ziel der Webanwendung soll es sein, die in der Datenbank zentralisiert vorliegenden Statistikdaten aufzuarbeiten und letztendlich zu visualisieren. Eine Visualisierung lässt sich innerhalb einer HTML-Seite realisieren, indem man verschiedene HTML-Standard-Elemente wie Tabellen, Hintergrundfarben und Linien miteinander kombiniert, so dass im Ergebnis eine Art Diagramm entsteht. Diese Vorgehensweise ist aber nicht nur optisch unschön, sondern auch in ihrer Anwendung höchst unpraktikabel, da jedes Diagramm individuell entworfen werden müsste.

Für den professionellen Umgang mit Diagrammen existieren mittlerweile eine ganze Reihe freier Java-Bibliotheken wie zum Beispiel OpenChart2, PtPlot oder MagPlot, welche neben der Unterstützung verschiedenster Diagramm-Typen auch eine sehr saubere Programmierung und ein gut dokumentiertes API zur Verfügung stellen.[67] Ich habe mich hier für die Bibliothek **JFreeChart** des Herstellers Object Refinery Limited

[64] vgl. Tong 2005, S 21-24

[65] vgl. Ship Tapestry 2004, S. 29-30

[66] vgl. Tong 2005, S. 14

[67] vgl. JFreeChart Alternatives 2007

entschieden, da diese Bibliothek eine Open-Source-Software unter der GNU Lesser General Public License (LGPL) ist, womit also auch der Einsatz innerhalb kommerzieller Software gestattet wird.[68] Außerdem hat die JFreeChart-Bibliothek gegenüber anderen Bibliotheken den Vorteil, dass sowohl das client- als auch serverseitige Generieren von Diagrammen und die Ausgabe in verschiedensten Grafikformaten wie beispielsweise Java Swing, PNG, JPEG, PDF oder SVG unterstützt werden.[69]

Beim Rendern der Grafiken greift JFreeChart auf die ebenfalls von Object Refinery Limited entwickelte JCommon-Bibliothek zurück, welche dementsprechend ebenfalls im Projekt eingebunden werden muss. Diese Bibliothek beinhaltet verschiedene nützliche Klassen, welche beispielsweise ein Logging Framework, einen XML-Parser, Serialisierungs-Tools und verschiedene Layout-Manager bereitstellen.[70]

Obwohl JFreeChart im Jahr 2000 ursprünglich als Bibliothek für normale Java Applikationen entwickelt wurde, erkannten die Entwickler schnell die Einsatzmöglichkeiten in Webanwendungen. In den Kapiteln 18 und 19 des kostenpflichtigen und 566 Seiten starken „The JFreeChart Class Library Developer Guide“ widmen sich 14 Seiten dem Einsatz der Bibliothek innerhalb von Java-Applets und -Servlets.[71] In Anlehnung an die dort skizzierte Implementierung kann auch innerhalb der TPiSCAN Webanwendung ein Java-Servlet zur serverseitigen Generierung der Diagramm-Grafiken verwendet werden.

6.4 Anwendungs-Logik

6.4.1 Funktionsumfang der Webanwendung

Aus den im Kapiteln 5.3 definierten Anforderungen ergibt sich der Funktionsumfang für die zu entwickelnde Webanwendung.

So soll es für den Benutzer möglich sein, sich sämtliche in der Statistik erfassten Daten per Webbrowser in Form von Tabellen und Diagrammen anzeigen zu lassen. Dazu wählt der Benutzer zuerst eine ihn interessierende **Abfrage** aus. Dies könnte

[68] vgl. LGPL License 2007

[69] vgl. JFreeChart 2007

[70] vgl. JCommon 2007

[71] vgl. JFreeChart Developer Guide 2006, S. 113-127

beispielsweise die Anzahl von getätigten Transaktionen an den Self-Checkout-Terminals sein. Daraufhin hat der Benutzer die Möglichkeit, sich für einen **Abfragezeitraum** zu entscheiden. Er wählt also den Zeitraum aus, für den innerhalb des zentralisierten Statistikdatenbestandes die Anzahl der getätigten Transaktionen ermittelt werden soll. Dies kann einen beliebigen Abfragezeitraum umfassen, welcher sich in einer Größenordnung von minimal einer Stunde und maximal n Jahren bewegt.

Außerdem hat der Benutzer die Möglichkeit, den Abfragezeitraum in eine beliebige Anzahl äquidistanter **Darstellungsperioden** einzuteilen, für die jeweils einzeln die Berechnung der Anzahl der Transaktionen stattfindet. Dies dient der Übersichtlichkeit und Interpretierbarkeit der statistischen Daten. So ist die Darstellung eines einzelnen Transaktionszählers für einen Anfragezeitraum von einer Woche weniger aussagekräftig als eine Reihe von sieben Zählern, welche die Anzahl der Transaktionen für jeden einzelnen Wochentag darstellen. Da die Darstellungsperioden die Detailgenauigkeit beziehungsweise die Feinkörnigkeit der Visualisierung beschreiben, werden sie nachfolgend innerhalb der Webanwendung als Granularität bezeichnet.

Ein weiteres Abgrenzungsmerkmal für die Abfrage stellt die Auswahl der innerhalb der Filiale vorhandenen Self-Checkout Terminals dar. Durch die Auswahl einer bestimmten oder aller **Lanes** gleichzeitig, können die erfassten Statistikdaten der einzelnen Lanes miteinander verglichen werden. So ist es möglich, Unregelmäßigkeiten oder Fehler, welche sich in den erfassten Statistikdaten ausdrücken, ganz konkret einer bestimmten Zeit und einem bestimmten Objekt zuzuordnen.

Nach der Formulierung der oben genannten vier **Abfragebedingungen** wird eine Datenbankabfrage gestartet, welche als Ergebnis die Statistikdaten für die gewählte Abfrage, den konkreten Abfragezeitraum und die gewählte Lane beinhaltet. Mit dem Ergebnis der Datenbankabfrage wird anschließend in Java die Berechnung der Zähler je nach gewählter Granularität durchgeführt. Das Ergebnis dieser Berechnung wird dann schließlich auf einer Webseite in tabellarischer Form und als Diagramm präsentiert.

6.4.2 Aggregation von Statistikdaten

Unter einer **Aggregation** versteht man eine Zusammenfassung oder Anhäufung von Daten. Der Begriff wird in der Informatik vor allem im Bereich der Datendefinitionssprache SQL verwendet[72], wo mit Aggregation eine Verdichtung von Daten gemeint ist, wobei aus einer Menge von Fakten ein aussagekräftiger Einzelfakt abgeleitet wird. In SQL sind Aggregationen mittels der GROUP BY Klausel realisierbar.[73] Im Sinne der Statistik ist beispielsweise die Bildung der Summe eine Aggregation.

In den Statistikdaten, welche innerhalb der TPiSCAN Applikation produziert werden, existieren insgesamt vier Typen von häufbaren Zählern:

1) Summenzähler vom Java-Typ Integer
2) Betragszähler vom Java-Typ Double
3) Zähler über die Gesamtdauer vom Java-Typ Long
4) Zähler über die Durchschnittsdauer vom Java-Typ Long

Diese Zähler werden in der Datenbank natürlich mit ihren entsprechenden SQL-Datentypen abgebildet. Der vierte Zähler-Typ stellt eine Besonderheit dar, da hier nicht Summen, sondern Mittelwerte aggregiert werden.

Beim Aggregieren von Mittelwerten muss darauf geachtet werden, dass der aggregierte Ergebniswert keine Summe, sondern wiederum einen Mittelwert darstellt.[74] Die einfache Berechnung des arithmetischen Mittels der Mittelwerte hilft hier allerdings nicht weiter. Vielmehr muss das gewogene arithmetische Mittel (GAM) berechnet werden, welches neben den zu aggregierenden Mittelwerten auch deren zu Grunde liegende Häufigkeiten berücksichtigt.[75]

---

[72] vgl. Heuer Saake 2000, S. 268-269

[73] vgl. Heuer Saake 2000, S. 361-366

[74] vgl. Schweizer Aggregation 1990, S. 36-38

[75] vgl. Eckstein 2003, S. 42

$$GAM = \sum_{j=1}^{k} n_j \bar{x}_j \Big/ \sum_{j=1}^{k} n_j$$

Wobei hier $\bar{\mathbf{x}}_j$ das Klassenmittel und $\mathbf{n_j}$ die absolute Klassenhäufigkeit bezeichnet.[76]

Beim Entwurf der Webanwendung gilt es zu berücksichtigen, dass je nach Benutzerabfrage sehr große Datenbestände aggregiert werden müssen. Damit bei der Aggregation eine gute Performance erreicht wird, sollte möglichst auf speicherintensive Operationen sowie überflüssige Objekt-Initialisierungen verzichtet werden.

6.4.3 Periodenproblematik

Um einen geeigneten Algorithmus für die Aggregation der Statistikdaten zu entwickeln, ist es notwendig, sich zuvor mit den Problemen zu beschäftigen, welche beim Umgang mit Zeitperioden auftreten können. Der Begriff der Periode wird hier nicht im mathematischen Sinne einer regelmäßig wiederkehrenden Ziffernfolge gebraucht, sondern ausschließlich im umgangssprachlichen Sinne. Gemeint ist also ein Zeitabschnitt von t_1 bis t_2, welcher sich auf einem Zeitstrahl t_0-t_N befindet. und „*durch bestimmte Abläufe, Ereignisse oder Entwicklungen geprägt ist*“[77]. Die Zeit, welche innerhalb einer Periode von t_1 bis t_2 verstreicht, wird als Δt (gelesen: „delta t“) dargestellt und als Periodenlänge bezeichnet.

Auf dem gesamten Zeitstrahl verteilt, finden verschiedenste, statistisch relevante Ereignisse statt. Jedem dieser Einzelereignisse ist ein Zeitstempel zugeordnet, welcher das Ereignis genau einem Punkt auf dem Zeitstrahl zuordnet. Eine Periode umfasst als Teilabschnitt des gesamten Zeitstrahls also nur eine Untermenge der Gesamtmenge an Einzelereignissen, welche sich über den Zeitstrahl verteilen.

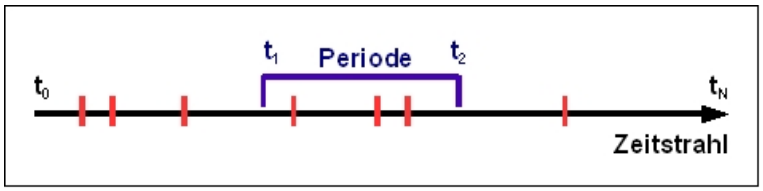

Abb. 9: Eine Periode auf dem Zeitstrahl (Quelle: eigene Darstellung)

[76] vgl. Bartsch 2004, S. 674

[77] Brockhaus 2007 Periode

Aufzeichnungs-Perioden

Das TPiSCAN Statistikmodul produziert im definierten Soll-Prozess zur Erzeugung von Statistikdaten in fest definierten Zeitabständen periodisch Statistik-Dateien. Diese beinhalten Informationen über sämtliche statistisch relevanten Ereignisse innerhalb des verstrichenen Zeitabschnitts (im Folgenden als „Aufzeichnungs-Periode“ bezeichnet). Allerdings beinhalten die produzierten Statistik-Dateien keine Informationen mehr über die stattgefundenen Einzelereignisse. Durch die vorgenommene Aggregation der Werte durch das Statistikmodul, werden gleiche Ereignisse zusammengefasst und aggregiert. Dadurch geht die Detailinformation über den exakten Zeitpunkt des Auftretens von Einzelereignissen verloren. Stattdessen lassen sich nur noch Aussagen über die Häufigkeit des Auftretens eines Ereignistyps innerhalb der Aufzeichnungs-Periode treffen.

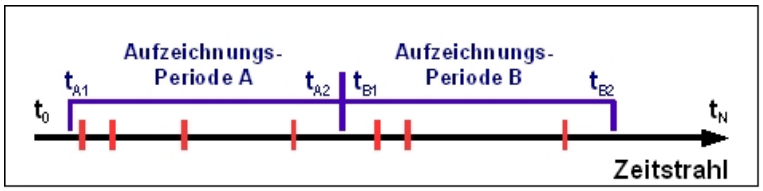

Abb. 10: Äquidistante Aufzeichnungsperioden (Quelle: eigene Darstellung)

Neben der Häufigkeit des Auftretens von Ereignissen, wird auch deren Dauer erfasst. So wäre es beispielsweise interessant zu wissen, wie lange eine bestimmte Transaktion innerhalb der Aufzeichnungs-Periode gedauert hat. Genau diese Aussage lässt sich mit der verwendeten Art der periodischen Statistiken allerdings *nicht* treffen. Stattdessen aggregiert das Statistikmodul alle innerhalb der Aufzeichnungs-Periode stattgefunden Transaktionsdauern zu einer Gesamtdauer. Mit Hilfe der Information über die Häufigkeit des Auftretens von Transaktionen, wird dann für die Aufzeichnungs-Periode auch eine durchschnittliche Transaktionsdauer berechnet. Die Information über die einzelnen Transaktionsdauern geht aber durch die Aggregation verloren.

Im Fall der TPiSCAN Webanwendung haben die Aufzeichnungs-Perioden eine äquidistante Länge von 15 Minuten (siehe dazu Anhang B2). Eine Aufzeichnungs-Periode beginnt und endet immer genau an vier möglichen Zeitpunkten innerhalb einer Stunde; bei 0, 15, 30 oder 45 Minuten. Zu dieser Regel existiert eine Ausnahme. Wird das TPiSCAN Statistikmodul aktiviert, beginnt die erste Aufzeichnungs-Periode im Moment der Aktivierung und läuft dann bis zum nächstfolgenden der vier genannten möglichen Endzeitpunkte. Somit ist die erste Aufzeichnungs-Periode nach der Aktivierung des TPiSCAN Statistikmoduls immer kleiner oder gleich einer

Viertelstunde. Bei der Beendung der Statistikaufzeichnung (Abbruch der Applikation) wird die letzte angebrochene Aufzeichnungs-Periode nicht erfasst, da sie nicht vollständig abgeschlossen wurde.

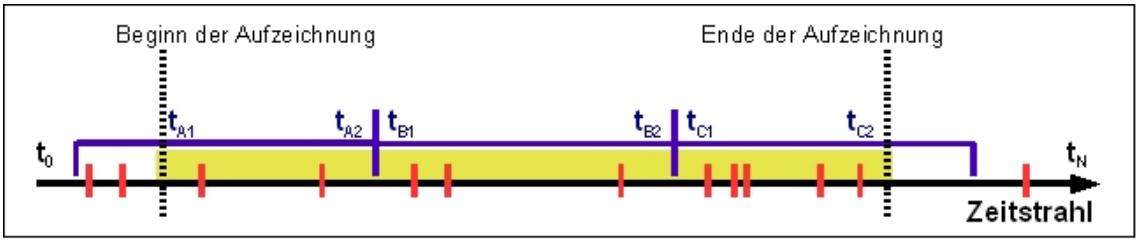

Abb. 11: Sonderfall erste und letzte Aufzeichnungs-Periode (Quelle: eigene Darstellung)

Perioden-Überschneidungen

Da ein Einzelereignis bei genauer Betrachtung nicht wie eingangs der Einfachheit halber erwähnt einen Punkt auf dem Zeitstrahl darstellt, sondern eine Ereignisdauer aufweist, ergeben sich bei der Zuordnung von Einzelereignissen zu einer Aufzeichnungs-Periode einige Probleme. De facto ist ein Einzelereignis also nichts anderes als eine Periode, welche auch einen Start- und Endzeitpunkt auf dem Zeitstrahl besitzt. Da eine Ereignisperiode nicht zwangsweise innerhalb einer Aufzeichnungs-Periode liegen muss, stellt sich die Frage, nach welchem Prinzip die Zuordnung bei sich überschneidenden Perioden stattfindet.

Dabei geht das TPiSCAN Statistikmodul sehr pragmatisch vor; ein Ereignis wird erst dann in die Statistik aufgenommen, wenn es vollständig stattgefunden hat. Es ergeben sich also zwei denkbare Fälle, welche eintreten können:

1) Die Ereignisperiode $E(t_{E1};t_{E2})$ liegt vollständig innerhalb der Aufzeichnungs-Periode $(t_{A1};t_{A2})$. Es gilt also $t_{A1} \leq t_{E1} \leq t_{A2}$ und $t_{A1} \leq t_{E2} \leq t_{A2}$.

2) Die Ereignisperiode $E(t_{E1};t_{E2})$ überschneidet sich mit zwei aufeinander folgenden Aufzeichnungs-Perioden $A(t_{A1};t_{A2})$ und $B(t_{B1};t_{B2})$, wobei $t_{A1} \leq t_{A2} = t_{B1} \leq t_{B2}$. Es gilt also $t_{A1} \leq t_{E1} \leq t_{A2}$ und $t_{B1} \leq t_{E2} \leq t_{B2}$.

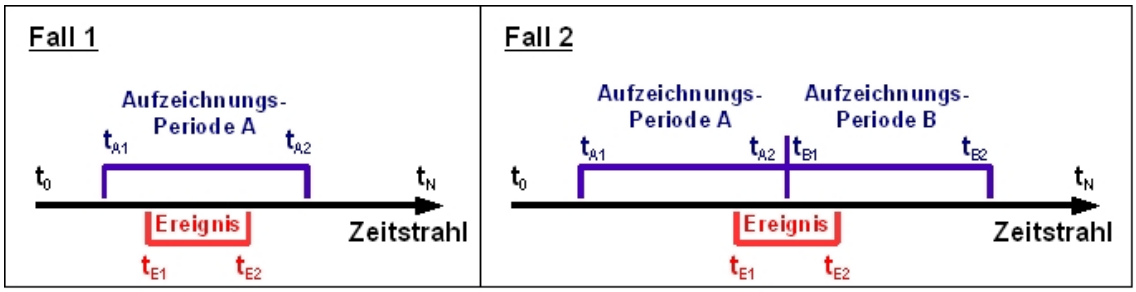

Abb. 12: Betrachtung von Ereignissen als Perioden (Quelle: eigene Darstellung)

Im ersten Fall wird das Ereignis E der Aufzeichnungs-Periode A zugeordnet. Handelt es sich bei dem Einzelereignis beispielsweise um eine stattgefundene Transaktion, so wird für die Aufzeichnungs-Periode A der Zähler für stattgefundene Transaktionen um eins erhöht und die Dauer der Transaktion zu der Gesamtdauer aller bisher stattgefundenen Transaktionen addiert.

Im zweiten Fall findet ein Teil des Ereignisses E in der Aufzeichnungs-Periode A und ein anderer Teil in der Aufzeichnungs-Periode B statt. Da das Ereignis E aber erst innerhalb der Aufzeichnungs-Periode B beendet ist, wird das ganze Einzelereignis E der Aufzeichnungs-Periode B zugeordnet. Somit enthält die Aufzeichnungs-Periode A keine Informationen über das zum Teil stattgefundene Ereignis E, während die Aufzeichnungs-Periode B sämtliche Informationen zum Einzelereignis E beinhaltet. Im Falle einer stattgefundenen Transaktion würden sich also die Zähler der Aufzeichnungs-Periode B verändern.

Das beschriebene Problem tritt bei sämtlichen Ereignissen auf, welche während des Übergangs von einer zur folgenden Aufzeichnungs-Periode stattfinden. Da das Problem technisch bedingt ist, wird es zur Kenntnis genommen, aber toleriert.

Darstellungs-Perioden

Wie in Kapitel 6.4.1 beschrieben, hat der Benutzer die Möglichkeit, sich innerhalb des gewählten Abfragezeitraums die Ergebnisse mit einer beliebigen Granularität darstellen zu lassen. Diese Granularität ist nichts anderes, als eine Darstellungs-Periode, innerhalb derer n Aufzeichnungs-Perioden zusammengefasst werden. Beinhalten beispielsweise zehn aufeinander folgende Aufzeichnungs-Perioden jeweils einen Transaktionszähler, so wird für die Darstellungs-Periode, welche die zehn Aufzeichnungs-Perioden umfasst, ein neuer Transaktionszähler als Summe der zehn Einzelzähler gebildet. Bei dieser Aggregation muss auf die in Kapitel 6.4.2 angesprochene Besonderheit bei der Mittelwertberechnung geachtet werden.

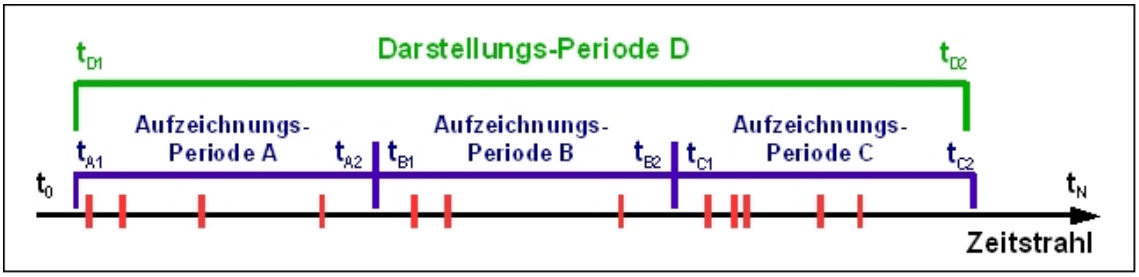

Abb. 13: Zusammenfassen mehrerer Aufzeichnungs-Perioden zu einer Darstellungs-Periode (Quelle: eigene Darstellung)

Damit sich beim Zusammenfassen mehrerer Aufzeichnungs-Perioden zu einer Darstellungs-Periode nicht auch die zuvor beschriebenen Perioden-Überschneidungsprobleme ergeben, ist darauf zu achten, dass die Länge der Darstellungs-Periode immer ein ganzzahliges Vielfaches der Länge der äquidistanten Aufzeichnungs-Perioden ist. Somit bieten sich Darstellungs-Perioden an, welche ein Vielfaches einer Viertelstunde sind (1 Stunde, 1 Tag, 1 Woche, etc.).

Perioden-Synchronität

Eine wesentliche Funktion der Webanwendung soll der Vergleich der Statistiken verschiedener Self-Checkout Terminals sein. Dazu schreiben die einzelnen Lanes ihre Statistikdaten parallel zueinander in die zentrale Statistik Datenbank. Damit eine Aggregation der Daten von verschiedenen Lanes durchgeführt werden kann, muss gewährleistet sein, dass die Aufzeichnungs-Perioden aller Lanes äquidistant sind.

Ein weiteres zu beachtendes Problem ist, dass die Lanes technisch gesehen eigenständige Rechner mit eigener JVM und einem OS sind. Dass heißt, dass die produzierten Statistik-Dateien einen Zeitstempel erhalten, welcher direkt von der Systemzeit auf dem jeweiligen Rechner abhängig ist. Die Aufzeichnungs-Perioden einzelner Lanes sind nur dann aggregierbar, wenn gewährleistet werden kann, dass die Systemzeit auf allen Lanes gleich ist. Nur so beschreiben zwei gleiche Zeitstempel verschiedener Lanes exakt denselben Zeitpunkt. Hier bietet sich der Einsatz eines Timeservers zur Gewährleistung der Zeit-Synchronität an.

Außerdem besteht die Möglichkeit, beim Einlesen der XML Dateien Ungenauigkeiten an den Periodenrändern künstlich zu „glätten“, also mathematisch zu runden.

6.4.4 Datumsarithmetik

Die Aggregation der Statistikdaten erfolgt auf Grundlage der äquidistanten, periodisch aufeinander folgenden Aufzeichnungs-Perioden. Zähler, wie sie in Kapitel 6.4.2 beschrieben wurden, beziehen sich immer genau auf eine bestimmte Aufzeichnungs-Periode, wobei eine Aufzeichnungs-Periode ein unikaler Abschnitt auf dem Zeitstrahl ist, welcher durch seinen Start- und Endzeitpunkt eindeutig identifiziert wird. Diese beiden Periodengrenzen werden beim Schreiben der XML-Statistik-Dateien durch das TPiSCAN Statistikmodul gesetzt und liegen beim Erzeugen der Datei in Form eines Unix-Timestamps (Anzahl der Millisekunden seit dem 01.01.1970, 00:00:00) vor.

Werden die Statistikdaten anschließend in die Datenbank geschrieben, so werden der Start- und der Endzeitpunkt jeweils durch einen Eintrag vom SQL-Datentyp TIMESTAMP repräsentiert. Im Gegensatz zu den SQL-Datentypen DATE und DATETIME, werden TIMESTAMP Werte MySQL-intern in die Zeitzone UTC konvertiert und beim Auslesen wieder zur aktuellen Zeitzone (der des MySQL-Servers) zurück konvertiert. Somit wird der repräsentierte Zeitpunkt wirklich exakt und vor allem vergleichbar hinterlegt.[78]

In Java wird ein Start- oder Endzeitpunkt einer Periode aus der Datenbank als Java-Typ java.sql.Timestamp ausgelesen. Diese Klasse ist lediglich eine Wrapper-Klasse um den Java-Datentyp java.util.Date, mit dem Ziel, der JDBC-API das Auslesen von SQL TIMESTAMP-Werten zu ermöglichen. Zum Rechnen in Java wird der Typ java.sql.Timestamp über die Methode getTime() in den Datentyp java.util.Date konvertiert.[79]

Das Rechnen mit Daten wird allgemein als Datumsarithmetik bezeichnet. Dabei geht es zum Beispiel darum, die Zeitdifferenz zwischen zwei Zeitpunkten exakt zu ermitteln oder ausgehend von einem beliebigen Zeitpunkt einen anderen Zeitpunkt zu ermitteln. Dass die Datumsarithmetik kein triviales Thema ist, liegt daran, dass Zeit keine konstante Größe ist.

[78] vgl. MySQL 4.1 Timestamp

[79] vgl. Sun Java API 1.5 Timestamp

Folgende einfache Beispiele verdeutlichen das Problem:

- „*Eine Minute hat nicht immer 60 Sekunden (z.B. wenn vom Internationalen Erd-Rotations-Service (IERS) angeordnete Schaltsekunden eingefügt werden, was 2006-01-01 zum 23. Mal seit 1972 passierte).*
- *Ein Tag hat nicht immer 24 Stunden (z.B. bei Sommer-/Winterzeitumstellung).*
- *Eine Woche hat nicht immer 7 Tage (z.B. die erste Woche im Jahr).*
- *Ein Jahr hat nicht immer 365 Tage (z.B. in Schaltjahren und bei Kalenderanpassungen).*
- *Die Berechnung, ob ein Jahr ein Schaltjahr ist, ist nicht immer fehlerfrei (alle 4 Jahre, außer alle 100, wenn nicht durch 400 teilbar).*“[80]

Es ist also nicht ohne weiteres möglich, ausgehend von einem Zeitpunkt t_1 einen Zeitpunkt t_2 exakt zu berechnen, indem man einfach zu t_1 eine Anzahl von Millisekunden addiert oder subtrahiert. Vielmehr ist ein detailliertes Wissen über die Bedingungen und Regeln erforderlich, denen die Zeit gehorcht. Diese Regeln erscheinen auf den ersten Blick unverständlich, beruhen aber weitestgehend auf der Berücksichtigung astronomischer Naturgesetze, wie der Erdrotation und den Mondbewegungen.[81]

In der Konsequenz bedeutet das für das Rechnen mit Daten in Java, dass die Klasse java.util.Date lediglich als Datumsspeicher (64-Bit-Long-Variable) verwendet werden kann. Zur Berechnung von Datums- und Zeitwerten muss auf eine intelligentere Java Klasse zurückgegriffen werden – den java.util.Calendar. Dieser wird meist in Form eines Gregorianischen Kalenders vom Typ java.util.GregorianCalendar benutzt. Diese Klasse kapselt Wissen über die angesprochenen Regeln und Bedingungen beim Rechnen mit der Zeit und eignet sich daher, um innerhalb der Webanwendung die Aggregation von Aufzeichnungs-Perioden zu Darstellungs-Perioden zu unterstützen.[82]

[80] T. Horn Datumsarithmetik

[81] vgl. Kapel Western Calendars 2006, S.11-18

[82] vgl. Sun Java API 1.5 GregorianCalendar

6.4.5 Mehrsprachigkeit

Das Apache Tapestry Webframework bringt eine Implementierung für die Unterstützung von Mehrsprachigkeit beziehungsweise Internationalisierung einer Webanwendung mit sich. Die Konfiguration dieser Implementierung ist aber nicht mit dem innerhalb von TPiSCAN verwendeten Konfigurationsmechanismus kompatibel. Deshalb wird die Mehrsprachigkeit innerhalb der Webanwendung über den für Java standardmäßigen Weg durch Property-Dateien realisiert.

Eine Java Property-Datei ist eine Textdatei mit der Endung „.properties“, welche eine durch Zeilenumbrüche separierte Liste von Schlüssel-Wert-Paaren beinhaltet. Über den Schlüssel kann aus der Property-Datei ein Wert ausgelesen werden, welcher üblicherweise einen lokalisierten Text vom Typ String enthält. Indem für jede Sprache eine eigene Property-Datei angelegt wird, können einem Schlüssel unterschiedlich lokalisierte Texte zugewiesen werden.

In Java wird statt des lokalisierten Textes nur noch der Schlüssel hinterlegt, nach dem in den Property-Dateien gesucht werden soll. Dies geschieht mit Hilfe der Klasse java.util.ResourceBundle. Mit der Methode getBundle() wird die Property-Datei zu einer bestimmten Landessprache ausgewählt. Dazu ist als Argument ein Objekt vom Typ java.util.Locale zu übergeben, welches eine Landessprache eindeutig beschreibt.[83] Der gesuchte Schlüssel wird aus der selektierten Property-Datei mit der Methode getString() ausgelesen. Der Rückgabewert ist der hinterlegte lokalisierte Text als Java-Datentyp String.[84]

Mit diesem einfachen Mechanismus lassen sich sämtliche in der Webanwendung dargestellten Texte in verschiedenen Sprachen hinterlegen, und durch das Setzen des Locale-Objektes kann die angezeigte Sprache dynamisch geändert werden.

Neben normalen Texten müssen aber im Rahmen einer Internationalisierung auch landesspezifische Datum- und Währungsformate berücksichtigt werden.[85] Dabei helfen die Java-Klassen java.text.DateFormat[86] und java.text.NumberFormat[87], welche durch

[83] vgl. Sun Java API 1.5 Locale

[84] vgl. Sun Java API 1.5 ResourceBundle

[85] vgl. Bell Parr 2003, S. 587-588

[86] vgl. Sun Java API 1.5 DateFormat

[87] vgl. Sun Java API 1.5 NumberFormat

die Übergabe des aktuellen Locales landesspezifische Formatierungen für Datum, Zahlen und Währungen bereitstellen können.

6.4.6 Visualisierung

Die Visualisierung der aggregierten Statistikdaten stellt den für den Benutzer interessanten Teil der Webanwendung dar. Die Darstellung der berechneten Ergebnisse erfolgt innerhalb einer HTML-Seite im Browser und wird durch das Webframework Tapestry gesteuert. Für das serverseitige Generieren der dargestellten Diagramme wird auf die Bibliothek JFreeChart zurückgegriffen.

Die einfachste und in fast allen Situationen sinnvolle Art der Präsentation der Statistikdaten erfolgt durch die Darstellung innerhalb einer HTML-Tabelle. Dabei enthält jede Zeile der Tabelle genau eine Darstellungsperiode. Innerhalb der Tabellenspalten werden dann pro Darstellungsperiode die gewünschten aggregierten Zähler dargestellt. Da die Tabelle je nach gewählten Abfragebedingungen mehrere hundert Zeilen umfassen kann, ist es notwendig, virtuelle Tabellenseiten einzuführen, mit denen die Tabelle durchblättert werden kann.

Zur einfachen und schnellen Ermittlung von Minimal- und Maximalwerten, ist es sinnvoll, der Tabelle eine Sortierfunktion zur Verfügung zu stellen, welche es ermöglicht, die Zeilen der Tabelle nach jeder beliebigen Spalte auf- oder absteigend zu sortieren. Die gewünschte Funktionalität wird von der Tapestry Komponente „Contrib:Table“ bereitgestellt.[88] Diese Komponente wird innerhalb der Webanwendung als Standard-Ausgabeformat für die aggregierten Statistikdaten verwendet.

Begin	End	Transactions
22 January 2003 00:00	23 January 2003 00:00	2,835
23 January 2003 00:00	24 January 2003 00:00	3,007
24 January 2003 00:00	25 January 2003 00:00	3,020
25 January 2003 00:00	26 January 2003 00:00	2,685
26 January 2003 00:00	27 January 2003 00:00	2,926
27 January 2003 00:00	28 January 2003 00:00	2,806
28 January 2003 00:00	29 January 2003 00:00	2,927

Abb. 14: Darstellung von Statistikdaten mit einer Contrib:Table Komponente (Quelle: eigene Darstellung)

[88] vgl. Apache Tapesty 4.0 Contrib:Table

Neben der tabellarischen Form bietet sich natürlich zur Visualisierung die Darstellung in Form von Diagrammen an. Hier stellt sich das Problem, dass es nicht das eine bestimmte Diagramm zur Darstellung beliebiger Inhalte gibt. Deshalb ist es schwierig, Diagramme zur Präsentation von Statistikdaten generisch zu erzeugen. Stattdessen muss je nach darzustellendem Inhalt individuell entscheiden werden, welcher Diagrammtyp als „Template“ benutzt werden soll, beziehungsweise, ob der Einsatz einer grafischen Visualisierung überhaupt sinnvoll ist.[89]

Bei der Wahl eines geeigneten Diagramm-Typs ist darauf zu achten, dass dieser der Anforderung der intuitiven Verständlichkeit genügt. Es sollten also sinnvoller Weise nur bekannte und leicht interpretierbare Diagramm-Typen verwendet werden. Generell lässt sich der Einsatz von Diagrammen folgendermaßen klassifizieren:

1) Zur Darstellung der Entwicklung von Zählerständen oder Ereignisdauern, bieten sich Liniendiagramme an, welche auf der Ordinate den Zeitstrahl und auf der Abszisse den Zähler- oder Ereignisdauer-Stand abbilden.

2) Zur Darstellung von Proportionen zweier Zählerstände oder Ereignisdauern eignen sich besonders gut Balkendiagramme, welche auf der Ordinate den Zeitstrahl und auf der Abszisse den Zähler- oder Ereignisdauer-Stand abbilden.

3) Zur Darstellung von Anteilen mehrerer Ereignisse an einer Gesamtmenge, eignet sich das Kreisdiagramm. Dabei muss drauf geachtet werden, dass die Anzahl der Ereignisse nicht zu groß wird, da sonst die Lesbarkeit des Diagramms nicht mehr gewährleistet ist.[90]

4) Zur Darstellung von sehr vielen Ereignissen mit einer großen Anzahl von Attributen ist kein Diagrammtyp wirklich gut geeignet. Hier bietet sich die alleinige Darstellung innerhalb einer Tabelle an.

Die Skalierung und Beschriftung der Diagramme erfolgt dann individuell je nach Benutzerabfrage. Die Diagramme werden serverseitig als Pixelgrafiken generiert, indem das Ergebnis der Benutzeranfrage als Parameter an ein Servlet übergeben werden. Dieses Servlet implementiert die Grafik-Bibliothek JFreeChart und generiert mit deren Hilfe aus den übergebenen Parametern ein Diagramm.[91]

---

[89] vgl. Kamps Diagram Design 1999, S. 18

[90] vgl. Kamps Diagram Design 1999, S. 7

[91] vgl. JFreeChart Developer Guide 2006, S. 118-119

Dieses Diagramm wird im PNG-Format über den HTTP-Response innerhalb eines java.io.OutputStream ausgegeben.[92] Im Browser des Clients wird also ein auf dem Server erzeugtes Bild geladen. Die folgende Abbildung zeigt ein von JFreeChart generiertes Diagramm:

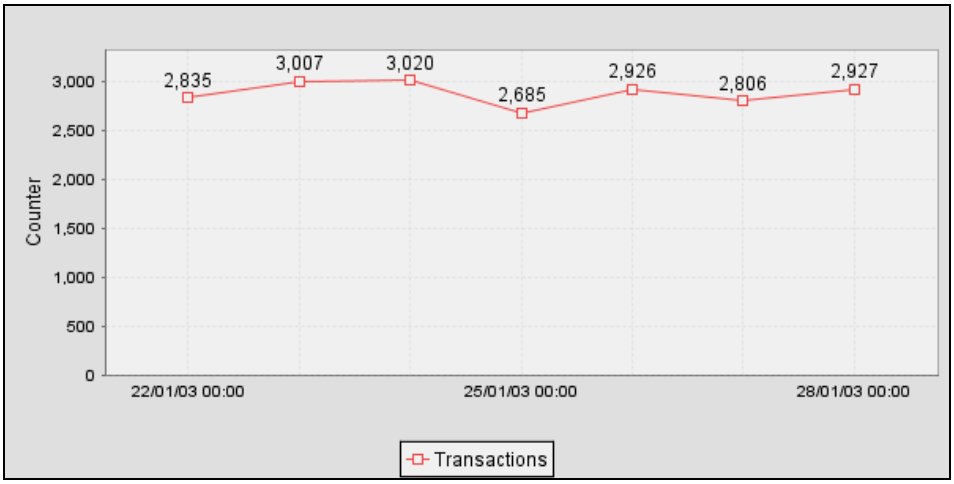

Abb. 15: Liniendiagramm zur Visualisierung der Anzahl der Transaktionen (Quelle: eigene Darstellung)

Auf die Berechnung und Darstellung von statistischen Kennzahlen (wie der Schiefe oder der Streuung) wird bewusst verzichtet, da die Anforderungsanalyse ergab, dass dafür kein Bedarf besteht.

6.5 Benutzungsoberfläche

Eine wesentliche Anforderung an die Applikation zur Visualisierung von Statistikdaten ist, dass die Bedienung der Anwendung intuitiv und ohne Einarbeitungszeit erfolgen soll. Deshalb bietet sich beim Entwurf der Benutzungsoberfläche die Verwendung eines typischen Layouts für eine Webanwendung an. Dieses besteht in der Regel aus einem Frameset, welches den sichtbaren Bildschirmbereich in drei Frames einteilt. Diese Frames übernehmen dabei dann üblicherweise spezielle Funktionen, wie die Navigation durch die Anwendung oder die Darstellung von Informationen.[93] Diese Aufteilung der Benutzungsoberfläche wird zwar von vielen als veraltet und optisch unschön bezeichnet, hat aber gerade aufgrund ihrer intuitiven Bedienbarkeit durchaus ihre Daseinsberechtigung.

---

[92] vgl. Sun Java API 1.5 OutputStream

[93] vgl. SelfHTML 2007 Frames

Im Falle der Webanwendung zur Statistikdaten Visualisierung lässt sich das beschriebene Konzept gut anwenden, wie die folgende Abbildung zeigt:

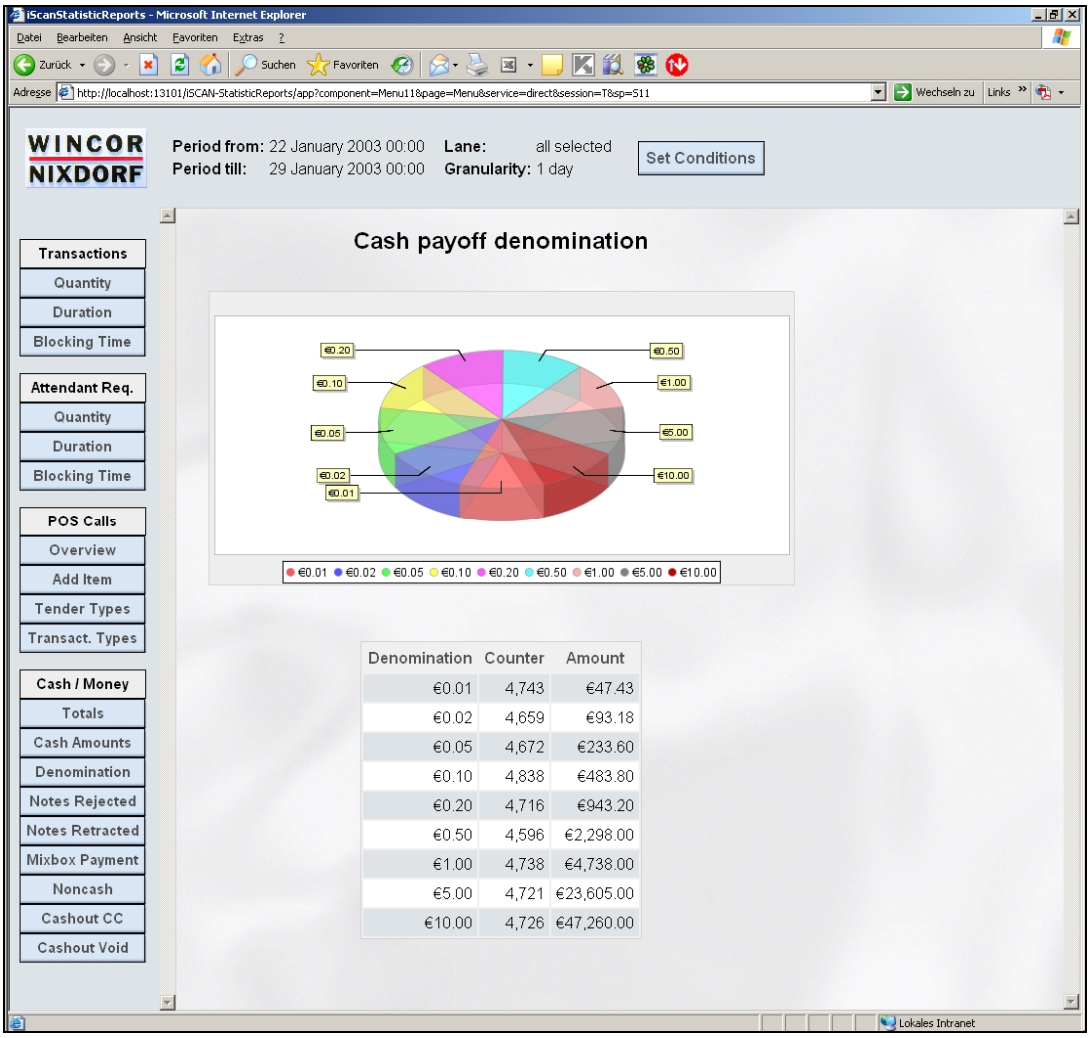

Abb. 16: Benutzungsoberfläche der Webanwendung (Quelle: eigene Darstellung)

In der Navigationsleiste am linken Bildschirmrand werden die verfügbaren statistischen Abfragen gruppiert als Links aufgelistet. Die Statusleiste am oberen Bildschirmrand zeigt die vom Benutzer gewählten Abfragebedingungen an und beinhaltet einen Link zu der HTML-Seite, wo diese Bedingungen gesetzt werden können. Den Rest des sichtbaren Bildschirmbereiches füllt das Anzeigefenster aus, welches die Visualisierung der aggregierten Statistikdaten beinhaltet. Dies geschieht, wie zuvor beschrieben, durch die Darstellung eines Diagramms und einer sortierbaren Tabelle.

Über den Link in der Statusleiste erreicht man die Eingabemaske für die Abfragebedingungen einer Statistik. Das Formular muss die Möglichkeit zur Eingabe einer Abfrageperiode, die Auswahl einer Lane und einer Granularität bieten. Die folgende Abbildung zeigt einen möglichen Aufbau:

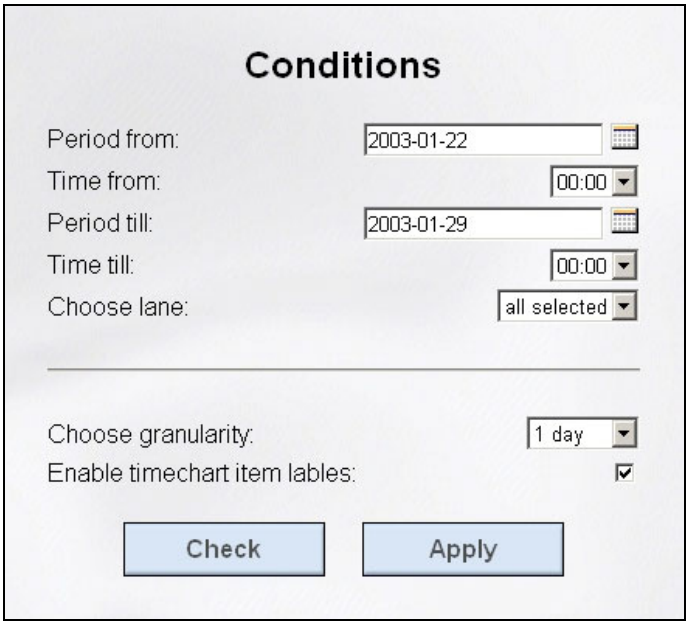

Abb. 17: Eingabeformular für Abfragen (Quelle: eigene Darstellung)

Das Formular unterstützt den Anwender bei der Eingabe des Datums durch die Verwendung eines Kalenders, welcher über das Kalendersymbol rechts neben dem jeweiligen Eingabefeld geöffnet werden kann. Der Kalender kann mit Hilfe der „DatePicker"-Komponente des Tapestry-Frameworks realisiert werden.[94] Eine Validierung der eingegebenen Daten findet über den Check-Button statt, die Anwendung der Abfragebedingungen für die zuvor gewählte statistische Abfrage findet über den Apply-Button statt.

Die gesamte Benutzungsoberfläche innerhalb des Browsers ist reines HTML, wobei die Tapestry-Komponenten zur Laufzeit generiert und in die HTML-Seite eingebettet werden. Dynamische Komponenten, wie der Kalender, werden in Tapestry intern mit JavaScript realisiert. Da die gesamte Benutzungsoberfläche aus HTML aufgebaut ist, lässt sie sich sehr einfach über den Standard-Layout-Mechanismus für Webseiten, per CSS-Datei, verändern und individuell anpassen. Dabei wird das Layout sämtlicher HTML-Seiten in einer zentralen Beschreibungsdatei abgelegt.

[94] vgl. Apache Tapestry 4.0 DatePicker

Durch die Änderung oder den Austausch dieser CSS-Datei kann das Layout sämtlicher HTML-Seiten der Anwendung verändert werden.[95] Diese Layoutbeschreibungen lassen sich auch auf die von Tapestry generierten Komponenten anwenden.

Das Layout der Oberfläche kann vom Kunden individuell angepasst werden. Dazu ist keine Änderung im Java-Code notwendig, sondern lediglich eine Änderung der zentralen CSS-Datei und der HTML-Templates, welche von Tapestry verwendet werden.

Durch den zuvor beschriebenen Mechanismus zur Unterstützung von mehreren Sprachen über Java Properties-Dateien, lässt sich die Oberfläche leicht landesspezifisch anpassen. Dies ist hier am Beispiel der Statusleiste dargestellt:

Abb. 18: Mehrsprachigkeit der Benutzungsoberfläche (Quelle: eigene Darstellung)

[95] vgl. SelfHTML 2007 CSS

7 Zusammenfassung und Ausblick

Arbeitsergebnisse

Die Auseinandersetzung mit den theoretischen Grundlagen des Bereichs Self-Checkout im Einzelhandel zu Beginn dieser Arbeit machte deutlich, welche Besonderheiten und Charakteristika innerhalb dieser Branche wichtig sind und was die wesentlichen Bestandteile von Self-Checkout ausmacht. Im Anwendungsszenario wurde aufgezeigt, welches Marktpotential sich durch die neue Technologie ergibt und worin der Tätigkeitsschwerpunkt der Wincor Nixdorf International GmbH innerhalb des Geschäftsbereichs Retail besteht. Diese Betrachtung war die Grundlage für das Verständnis der Anforderungsanalyse und des Entwurfs einer Anwendung für die spezielle Aufgabe der Visualisierung von Statistikdaten innerhalb der Wincor Nixdorf Systemsoftware TPiSCAN.

Durch die Anforderungsanalyse im fünften Kapitel konnten erhebliche Schwachstellen im bestehenden Ist-Prozess der Statistikdatenerfassung und -Auswertung aufgezeigt werden. Basierend auf der detaillierten Situationsanalyse, wurden konkrete betriebliche Anforderungen an die neu zu entwickelnde Applikation erhoben. Diese Anforderungen flossen im nächsten Schritt direkt in die Entwurfsphase ein, wo ein verbesserter Soll-Prozess zur Statistikdatenerfassung vorgeschlagen wurde. Darauf aufbauend wurde eine konkrete Anwendungs-Architektur zur Visualisierung der Statistikdaten entworfen, welche sich besonders gut in die bestehende Infrastruktur der analysierten TPiSCAN Anwendung integriert.

Anhand der in der Anforderungsanalyse definierten betriebswirtschaftlichen und technischen Anforderungen an die Anwendung war es möglich, durch die Auswahl geeigneter Open-Source-Technologien eine kostengünstige und dennoch effiziente technische Umsetzung zu konzipieren. Die im Kapitel 6.4 angestellten Überlegungen zur Anwendungs-Logik machen einen wesentlichen Teil der Entwurfsphase aus, da sie bereits konkrete Vorschläge und Ideen für die Implementierung der Anwendung beinhalten.

Mit der entworfenen Lösung zur Visualisierung von Statistikdaten konnten die gestellten Anforderungen an die Anwendung komplett erfüllt werden. Wie sich in der nachfolgenden Implementierungsphase zeigte, deckte die Konzeption alle wichtigen Punkte für eine erfolgreiche Implementierung ab.

Abschließende Bemerkung

Das entwickelte Konzept leistet nicht nur die gedankliche Vorarbeit für die erfolgreiche Implementierung einer Applikation zur Visualisierung von Statistikdaten im Retail Bereich Self-Checkout, sondern ergänzt auch den bestehenden Prozess der Erfassung von Statistikdaten. Die in dieser konzeptionellen Arbeit getroffenen Überlegungen sollen Wincor Nixdorf dabei unterstützen, die Kundenakzeptanz für die Statistik-datenauswertung der TPiSCAN Systemsoftware zu verbessern. Durch den Einsatz von zeitgemäßen Technologien und der vollständigen Integration in TPiSCAN, ist eine gute Wartbarkeit und Erweiterbarkeit der Applikation gewährleistet.

Auf Basis der in dieser Bachelorthesis präsentierten Konzeption war es mir möglich, eine Applikation zu implementieren, welche heute bereits als Bestandteil der TPiSCAN Systemsoftware ausgeliefert wird. Für die Zukunft kommen als mögliche optionale Erweiterungen unter anderem der Export der Statistikdaten-Visualisierung, eine Reporting-Funktion oder die Integration von Planzahlen in Betracht.

Literaturverzeichnis

Alpar 2002

Paul Alpar et al.: „Anwendungsorientierte Wirtschaftsinformatik“, Vieweg Verlag, 3. überarbeitete und erweiterte Auflage Oktober 2002, Braunschweig Wiesbaden 2002

Apache Derby 2007

Apache Software Foundation: „Apache Derby”, http://db.apache.org/derby/, Version vom 01.08.2007

Apache Tapestry 2007

Apache Software Foundation: „Tapestry – Welcome to Tapestry”, http://tapestry.apache.org/, Version vom 01.08.2007

Apache Tapestry 4.0 Components

Apache Software Foundation: „Framework Component Reference”, http://tapestry.apache.org/tapestry4/tapestry/ComponentReference/, Version vom 01.08.2007

Apache Tapesty 4.0 Contrib:Table

Apache Software Foundation: „Table”, http://tapestry.apache.org/tapestry4/tapestry-contrib/ComponentReference/Table.html, Version vom 01.08.2007

Apache Tapestry 4.0 DatePicker

Apache Software Foundation: „DatePicker”, http://tapestry.apache.org/ tapestry4/tapestry/ComponentReference/DatePicker.html, Version vom 01.08.2007

Apache Tomcat 2007

Apache Software Foundation: „Apache Tomcat“, http://tomcat.apache.org/, Version vom 01.08.2007

Apache Tomcat API 5.0 Embedded

Apache Software Foundation: „Embedded (Tomcat API Documentation)”, http://tomcat.apache.org/tomcat-5.0-doc/catalina/docs/api/org/apache/ catalina/startup/Embedded.html, Version vom 01.08.2007

Apache Tomcat 5.0 Deployment

Apache Software Foundation: „Application Developer's Guide – Deployment", http://tomcat.apache.org/tomcat-5.0-doc/appdev/deployment.html, Version vom 01.08.2007

Balzert 2001

Heide Balzert: „Objektorientierung in 7 Tagen", Spektrum Akademischer Verlag GmbH, 1. Nachdruck 2001, Heidelberg-Berlin 2001

Balzert 2005

Heide Balzert: „Lehrbuch der Objektmodellierung", Spektrum Akademischer Verlag GmbH, 2. Auflage 2005, München 2005

Bartsch 2004

Hans-Jochen Bartsch: „Taschenbuch Mathematischer Formeln", Fachbuchverlag Leipzig im Carl Hanser Verlag, 20. Auflage, München-Wien 2004

Bell Parr 2003

Douglas Bell, Mile Parr: „Java für Studenten", Pearson Studium Verlag, 3. überarbeitete Auflage, München 2003

Brockhaus 2007 Filiale

F.A. Brockhaus: „Der Brockhaus: in 15 Bänden", Permanent aktualisierte Online-Auflage, http://www.brockhaus.de/katalog/, Leipzig Mannheim 2002-2007

Brockhaus 2007 Periode

F.A. Brockhaus: „Der Brockhaus: in 15 Bänden", Permanent aktualisierte Online-Auflage, http://www.brockhaus.de/katalog/, Leipzig Mannheim 2002-2007

Brockhaus 2007 Visualisierung

F.A. Brockhaus: „Der Brockhaus: in 15 Bänden", Permanent aktualisierte Online-Auflage, http://www.brockhaus.de/katalog/, Leipzig Mannheim 2002-2007

Eckstein 2003

Peter P. Eckstein: "Repetitorium Statistik", Betriebswirtschaftlicher Verlag Dr. Th. Gabler GmbH, 5. vollständig überarbeitete und erweiterte Auflage April 2003, Wiesbaden 2003

Gourley Totty HTTP 2002

David Gourley, Brian Totty et al.: “HTTP - The Definite Guide”, O’Reilly & Associates Inc., First Edition, Sebastopol September 2002

Kamps Diagram Design 1999

Thomas Kamps: „Diagram Design - A Constructive Theory“, Springer-Verlag, Berlin Heidelberg 1999

Hartung 1986

Joachim Hartung: „Statistik: Lehr- und Handbuch der angewandten Statistik“, R. Oldenbourg Verlag GmbH, 5. durchges. Auflage, München 1986

Heuer Saake 2000

Andreas Heuer, Gunter Saake: „Datenbanken: Konzepte und Sprachen“, mitp-Verlag, 2. Auflage 2000, Landsberg 2000

IEEE SRS 1998

IEEE: “IEEE recommended practice for software requirements specifications”, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&isnumber=15571&arnumber=720574&punumber=5841, Version vom 11.08.2007

IHL Consulting Group 2006

IHL Consulting Group: „North American Retail Self-Checkout Systems Market Study“, http://www.ihlservices.com/ihl/product_detail.cfm?page=&ProductID=4, Version vom 19.07.2007

Java Code Obfuscation

University of Auckland, Douglas Low: „Protecting Java Code Via Code Obfuscation”, http://www.cs.arizona.edu/~collberg/Research/Students/DouglasLow/obfuscation.html, Version vom 01.08.2007

JCommon 2007

Object Refinery Limited: “JCommon – www.jfree.org”, http://www.jfree.org/jcommon/index.php, Version vom 12.08.2007

JFreeChart 2007

Object Refinery Limited: “JFreeChart”, http://www.jfree.org/jfreechart/, Version vom 01.08.2007

JFreeChart Alternatives 2007

Object Refinery Limited: “JFreeChart: FAQ”, http://www.jfree.org/jfreechart/faq.html#FAQ13, Version vom 12.08.2007

JFreeChart Developer Guide 2006

David Gilbert: “The JFreeChart Class Library Version 1.0.2 Developer Guide”, Object Refinery Limited, August 2006

Kapel Western Calendars 2006

Martin Kapel: “The structure and mathematics of the principal calendars of the western world”, The Edwin Mellen Press Ltd., Lampeter 2006

LGPL License 2007

Free Software Foundation (FSF): “GNU Lesser General Public License”, http://www.gnu.org/licenses/lgpl.html, Version vom 01.08.2007

Mertens 2004

Peter Mertens et al.: „Grundzüge der Wirtschaftsinformatik“, Springer-Verlag, 8. AuflageBerlin-Heidelberg-New York 2004

MySQL 4.1 Timestamp

MySQL AB: “MySQL 3.23, 4.0, 4.1 Reference Manual :: 11.3.1.2 TIMESTAMP Properties as of MySQL 4.1”, http://dev.mysql.com/doc/refman/4.1/en/timestamp.html, Version vom 11.08.2007

MySQL Licensing 2007

MySQL AB: „MySQL AB: MySQL Licensing Policy“, http://www.mysql.com/company/legal/licensing/, Version vom 01.08.2007

MySQL Website 2007

MySQL AB: “MySQL AB: The world’s most popular open source database”, http://www.mysql.com/, Version vom 01.08.2007

O’Reilly OnLAMP 2001

O’Reilly Media Inc.: “INLamp.com – LAMP: The Open Source Web Platform”, http://www.onlamp.com/pub/a/onlamp/2001/01/25/lamp.html, Version vom 11.08.2007

Schweizer Aggregation 1990

Karl Schweizer: “Der Aggregationseffekt”, Verlag Peter Lang GmbH, Frankfurt am Main 1990

SelfHTML 2007 Frames

Swen Wacker, SELFHTML e.V.: „SELFHTML: Framesets und Frames definieren“, http://de.selfhtml.org/html/frames/definieren.htm, Version vom 12.08.2007

SelfHTML 2007 CSS

Swen Wacker, SELFHTML e.V.: „SELFHTML: Stylesheets“, http://de.selfhtml.org/css/intro.htm, Version vom 01.08.2007

Self Service World 2007

NetWorld Alliance LLC: „Kiosk and Self-Service Stats and Facts | Self Service World“, http://www.selfserviceworld.com/research.php?rc_id=363, Version vom 19.07.2007

Ship Tapestry 2004

Howard M. Lewis Ship: “Tapestry in action”, Manning Publications Co., Greenwich 2004

Stahlknecht 2005

Stahlknecht, Hasenkamp: “Einführung in die Wirtschaftsinformatik”, Springer Verlag, 11. Auflage, Heidelberg 2005

Sun Java API 1.5 DateFormat

Sun Microsystems Inc.: “DateFormat (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/text/DateFormat.html, Version vom 13.08.2007

Sun Java API 1.5 GregorianCalendar

Sun Microsystems Inc.: “GregorianCalendar (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/util/GregorianCalendar.html, Version vom 03.08.2007

Sun Java API 1.5 Locale

Sun Microsystems Inc.: “Locale (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html, Version vom 03.08.2007

Sun Java API 1.5 NumberFormat

Sun Microsystems Inc.: “NumberFormat (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/text/NumberFormat.html, Version vom 13.08.2007

Sun Java API 1.5 OutputStream

Sun Microsystems Inc.: “OutputStream (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/io/OutputStream.html, Version vom 01.08.2007

Sun Java API 1.5 ResourceBundle

Sun Microsystems Inc.: “ResourceBundle (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/util/ResourceBundle.html, Version vom 03.08.2007

Sun Java API 1.5 Timestamp

Sun Microsystems Inc.: “Timestamp (Java 2 Platform SE 5.0)”, http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html, Version vom 03.08.2007

T. Horn Datumsarithmetik

Torsten Horn: „Java Date und Calendar“, http://www.torsten-horn.de/techdocs/java-date.htm#DatumsArithmetik, Version vom 03.08.2007

Tong 2005

Ka Iok ‘Kent’ Tong: „Enjoying Web Development with Tapestry“, TipTec Development, Second edition 2005, Printed in the United States

Wiki Web 2007

Wikimedia Foundation Inc.: „Wikipedia – Bild:Webanwendung”, http://de.wikipedia.org/wiki/Bild:Webanwendung_client_server_01.png, Urheber: Gerd Franke, September 2006, Version vom 10.08.2007

WN Datenblatt iSCAN Basic Line 2007

Wincor Nixdorf International GmbH: „Wincor-Nixdorf Basic Line “, http://www.wincor-nixdorf.com/internet/de/Products/Storeautomation/iSCAN/SolutionPortfolio/BasicLine/Main,templateId=blob.jsp,property=DetailPaper.pdf, Germany February 2007

WN Datenblatt Kassenfamilie BEETLE 2006

Wincor Nixdorf International GmbH: „Wincor Nixdorf - Kassenfamilie BEETLE“, http://www.wincor-nixdorf.com/internet/de/Products/PosSys/BeetleFamily/Main,templateId=blob.jsp,property=DetailPaper.pdf, Germany Januar 2006

WN Halbjahresbericht 06/07

Wincor Nixdorf International GmbH: „Wincor-Nixdorf Halbjahresbericht 2006/2007“, http://www.wincor-nixdorf.com/static/finanzberichte/2006-2007/q2/de/kennzahlen.html, Version vom 17.07.2007

WN SCO Solutions 2005

Wincor Nixdorf International GmbH, Alexandra Böddeker: “Self Checkout Solutions”, iSCAN 122005.ppt, Berlin Dezember 2005, Version vom 10.08.2007

WN TPiSCAN Statistikmodul 2007

Wincor Nixdorf International GmbH, Dr. Sönke Kannapinn: “TPiSCAN Statistics and Journal”, Statistics and Journal.pdf, Berlin Januar 2006, zuletzt bearbeitet am 25. Juni 2007, Version vom 01.08.2007

WN Website Company 2007

Wincor Nixdorf International GmbH: „Wincor Nixdorf - Unternehmen“, http://www.wincor-nixdorf.com/internet/de/Company/index.html, Version vom 17.07.2007

WN Website Historie 2007

Wincor Nixdorf International GmbH: „Wincor Nixdorf - Historie“, http://www.wincor-nixdorf.com/internet/de/Company/History/index.html, Version vom 10.08.2007

WN Website S&B 2007

Wincor Nixdorf International GmbH: „BEETLE/iSCAN – Tower Line, POS Tower 100 Scan&Bag”, http://www.wincor-nixdorf.com/internet/de/Products/Storeautomation/iSCAN/SolutionPortfolio/TowerLine/ScanBag/Main,templateId=blob.jsp,property=DetailPaper.pdf, Version vom 10.08.2007

WN Website SCO 2007

Wincor Nixdorf International GmbH: „Wincor Nixdorf - Self-Checkout Familie“, http://www.wincor-nixdorf.com/internet/de/Products/Storeautomation/iSCAN/Detail.html, Version vom 17.07.2007

WN Website Strategie 2007

Wincor Nixdorf International GmbH: „Wincor Nixdorf - Strategie“, http://www.wincor-nixdorf.com/internet/de/Company/Strategy/index.html, Version vom 10.08.2007

WN Website Unternehmenskultur 2007

Wincor Nixdorf International GmbH: „Wincor Nixdorf - Unternehmenskulltur“, http://www.wincor-nixdorf.com/internet/de/Company/PeopleWithSpirit/index.html, Version vorm 10.08.2007

WN Website TPiSCAN 2007

Wincor Nixdorf International GmbH: „Wincor Nixdorf - TPiSCAN“, http://www.wincor-nixdorf.com/internet/de/Products/Storeautomation/iSCAN/SolutionPortfolio/TPiScan/index.html, Version vom 21.07.2007

Anhang

Anhang A – TPiSCAN Statistikmodul: XML-DTD Datei

```
<!ELEMENT statistics (transactions, tender, poscalls, assistance)>
<!ATTLIST statistics
    storenumber       CDATA #REQUIRED
    terminalnumber    CDATA #REQUIRED
    start             CDATA #REQUIRED
    startverbose      CDATA #IMPLIED
    end               CDATA #REQUIRED
    endverbose        CDATA #IMPLIED
    operationmode     (normal|training|weightlearning) #REQUIRED
    version           CDATA #REQUIRED>

<!ELEMENT transactions EMPTY>
<!ATTLIST transactions
    count             CDATA #REQUIRED
    durationaverage   CDATA #REQUIRED
    durationaverageverbose   CDATA #IMPLIED
    total             CDATA #REQUIRED
    totalnet          CDATA #REQUIRED>

<!ELEMENT tender (cash, noncash*)>
<!ELEMENT cash (coindispenser*, coinacceptor*, notedispenser*, noteacceptor*,
                cashoutcausedbycustomercancel, cashoutcausedbyvoid)>
<!ATTLIST cash
    amountpayed       CDATA #IMPLIED
    amountcollected   CDATA #IMPLIED>
<!ELEMENT coindispenser                 EMPTY>
<!ATTLIST coindispenser
    amount            CDATA #IMPLIED
    count             CDATA #REQUIRED
    denomination      CDATA #IMPLIED
    type              (normal) #REQUIRED>
<!ELEMENT coinacceptor                  EMPTY>
<!ATTLIST coinacceptor
    amount            CDATA #IMPLIED
    count             CDATA #REQUIRED
    denomination      CDATA #IMPLIED
    type              (normal|mixbox) #REQUIRED>
<!ELEMENT notedispenser                 EMPTY>
<!ATTLIST notedispenser
    amount            CDATA #IMPLIED
    count             CDATA #REQUIRED
    denomination      CDATA #IMPLIED
    type              (normal|reject|retract) #REQUIRED>
<!ELEMENT noteacceptor                  EMPTY>
<!ATTLIST noteacceptor
    amount            CDATA #IMPLIED
    count             CDATA #REQUIRED
    denomination      CDATA #IMPLIED
    type              (normal|mixbox) #REQUIRED>
<!ELEMENT cashoutcausedbycustomercancel EMPTY>
<!ATTLIST cashoutcausedbycustomercancel
    count             CDATA #REQUIRED
    amount            CDATA #IMPLIED>
<!ELEMENT cashoutcausedbyvoid           EMPTY>
<!ATTLIST cashoutcausedbyvoid
    count             CDATA #REQUIRED
    amount            CDATA #IMPLIED>
<!ELEMENT noncash                       EMPTY>
<!ATTLIST noncash
    description       CDATA #REQUIRED
    count             CDATA #REQUIRED>
```

```
62
63 <!ELEMENT poscalls (group*)>
64 <!ELEMENT group (method*)>
65 <!ATTLIST group
66     name              CDATA #REQUIRED>
67 <!ELEMENT method EMPTY>
68 <!ATTLIST method
69     name              CDATA #REQUIRED
70     count             CDATA #REQUIRED
71     successful        CDATA #REQUIRED
72     notExecuted       CDATA #REQUIRED
73     meanDuration      CDATA #IMPLIED>
74
75 <!ELEMENT assistance (attendantrequest|confirmed-attendantrequest)*>
76
77 <!ELEMENT attendantrequest EMPTY>
78 <!ATTLIST attendantrequest
79     id                CDATA #REQUIRED
80     count             CDATA #REQUIRED>
81
82 <!ELEMENT confirmed-attendantrequest EMPTY>
83 <!ATTLIST confirmed-attendantrequest
84     id                CDATA #REQUIRED
85     count             CDATA #REQUIRED
86     meanLifeTime      CDATA #IMPLIED
87     meanBlockingTime  CDATA #IMPLIED>
```

Abb. 19: DTD zu den vom Statistikmodul erzeugten XML-Dateien (Quelle: WN TPiSCAN Statistikmodul 2007, S. 10-11)

Anhang B – Konfigurationsdateien

Anhang B1 – Embedded-Tomcat-Konfiguration

```
 1 <config>
 2   <tomcat>
 3     <classpath>
 4
 5       <pathelement location="${TPiSCAN_runtime}/core/projects/iScanWebServing/bin"/>
 6       <pathelement location="${TPiSCAN_runtime}/core/lib/iScanWebserving.jar"/>
 7
 8       <pathelement location="${TPiSCAN_runtime}/core/projects/iScanStatisticReports/bin"/>
 9       <pathelement location="${TPiSCAN_runtime}/core/lib/iScanStatisticReports.jar"/>
10       <pathelement location="${TPiSCAN_runtime}/core/lib/commons-codec-1.3.jar"/>
11
12       <pathelement location="${TPiSCAN_base}/resources/iSCAN-StatisticReports/WEB-INF"/>
13
14       <fileset dir="${TPiSCAN_runtime}/core/apache-tomcat-5.5.20-embed/lib">
15         <include pattern=".*.jar"/>
16       </fileset>
17
18       <fileset dir="${TPiSCAN_runtime}/core/lib/tapestry">
19         <include pattern=".*.jar"/>
20       </fileset>
21
22     </classpath>
23   </tomcat>
24 </config>
```

Abb. 20: EmbeddedTomcatConfig_default.xml (Quelle: eigene Darstellung)

Anhang B2 – Datenbank-Konfiguration

```
<config>
    <!-- jdbc driver name -->
    <driver driverName="com.mysql.jdbc.Driver"/>
    <!-- database name for database -->
    <!-- database url for database -->
    <!-- database port number for database (mysql:3306, derby:1527) -->
    <db dbName="statistics" dbUrl="jdbc:mysql:" dbPort="3306"/>
    <!-- user id for database usage -->
    <!-- password for database usage -->
    <user userId="iscan_user" password="checkout"/>
</config>
```

Abb. 21: StatisticDBConfig_default.xml (Quelle: eigene Darstellung)

```
<config>
    <!-- jdbc driver name -->
    <driver driverName="org.apache.derby.jdbc.ClientDriver"/>
    <!-- database name for database -->
    <!-- database url for database -->
    <!-- database port number for database (mysql:3306, derby:1527) -->
    <db dbName="Statistics" dbUrl="jdbc:derby:" dbPort="1527"/>
    <!-- user id for database usage -->
    <!-- password for database usage -->
    <user userId="iscan_user" password="checkout"/>
</config>
```

Abb. 22: StatisticDBConfig_devStation.xml (Quelle: eigene Darstellung)

Anhang B3 – TPiSCAN Statistikmodul-Konfiguration

```
<?xml version="1.0"?>
<config>

  <parameter name="CREATE_PERIODIC_STATISTICS"  value="true" />
  <parameter name="PERIOD_LENGTH"               value="15" />
  <parameter name="PERIOD_FILENAME_PREFIX"      value="statisticReports-" />
  <parameter name="PERIOD_SUFFIX_MODE"          value="TIME" />
  <parameter name="SESSION_STATISTICS_ADAPTION" value="false" />

  <posmethod-counting-rule match="*" counterId="{0}({1},{2},{3})" />
  <attendant-request-counting-rule match="*"  counterId="{0}({1},{2},{3})" />

  <period_parameters exportStatistics="true" />
  <session_parameters adaption="false" />
</config>
```

Abb. 23: StatisticReportsConfig_default.xml (Quelle: eigene Darstellung)

Anhang C – Implementierung der MySQL-Statistik-Datenbank

Die Webanwendung zur Visualisierung von Statistikdaten baut direkt darauf auf, dass die auf den einzelnen Self-Checkout-Terminals produzierten Statistikdaten zentralisiert in einer Datenbank vorliegen. Die Implementierung dieser Datenbank ist nicht Inhalt dieser Bachelorthesis. Trotzdem soll an dieser Stelle die implementierte Datenbank vorgestellt werden, da sie eine wichtige Voraussetzung für die Webanwendung darstellt.

Die Datenbank wird genutzt um die auf den Lanes produzierten Statistik-Daten zentral zu speichern. Dazu werden die temporären XML-Statistik-Dateien ausgelesen und deren Inhalt per JDBC in die Datenbank geschrieben. Die Datenbank muss also den Inhalt der in Kapitel 5.2.3 beschriebenen XML-Ausgabedateien mindestens 1:1 abbilden, oder im Sinne einer Optimierung erweitern.

Die implementierte Datenbank umfasst neun Tabellen, welche die Informationen aus den XML-Dateien aufnehmen. Neben den Informationen aus den XML-Dateien, werden aber auch noch zusätzliche Werte in die Datenbank geschrieben, welche von der Webanwendung ausgewertet werden können. Dazu zählen vor allem Informationen, welche die verwendete Währung (engl. „currency“) der Geldmittel betreffen.

Auf Grund der kompakten Datenbankgröße wurde weitestgehend auf die Verwendung von Surrogatschlüsseln, also künstlichen Primärschlüsseln, verzichtet. Es werden zusammengesetzte, dafür aber sprechende Primärschlüssel eingesetzt, was die Lesbarkeit und Handhabbarkeit der Datenbank deutlich vereinfacht. Nachteile sind natürlich die schwierigere Referenzierbarkeit der Datenelemente und die Größe des zusammengesetzten Schlüssels. Diese Nachteile wurden aber auf Grund der kompakten und übersichtlichen Datenbankstruktur als nicht kritisch eingestuft.

Die nachfolgende Abbildung zeigt das Datenmodell der Statistikdatenbank:

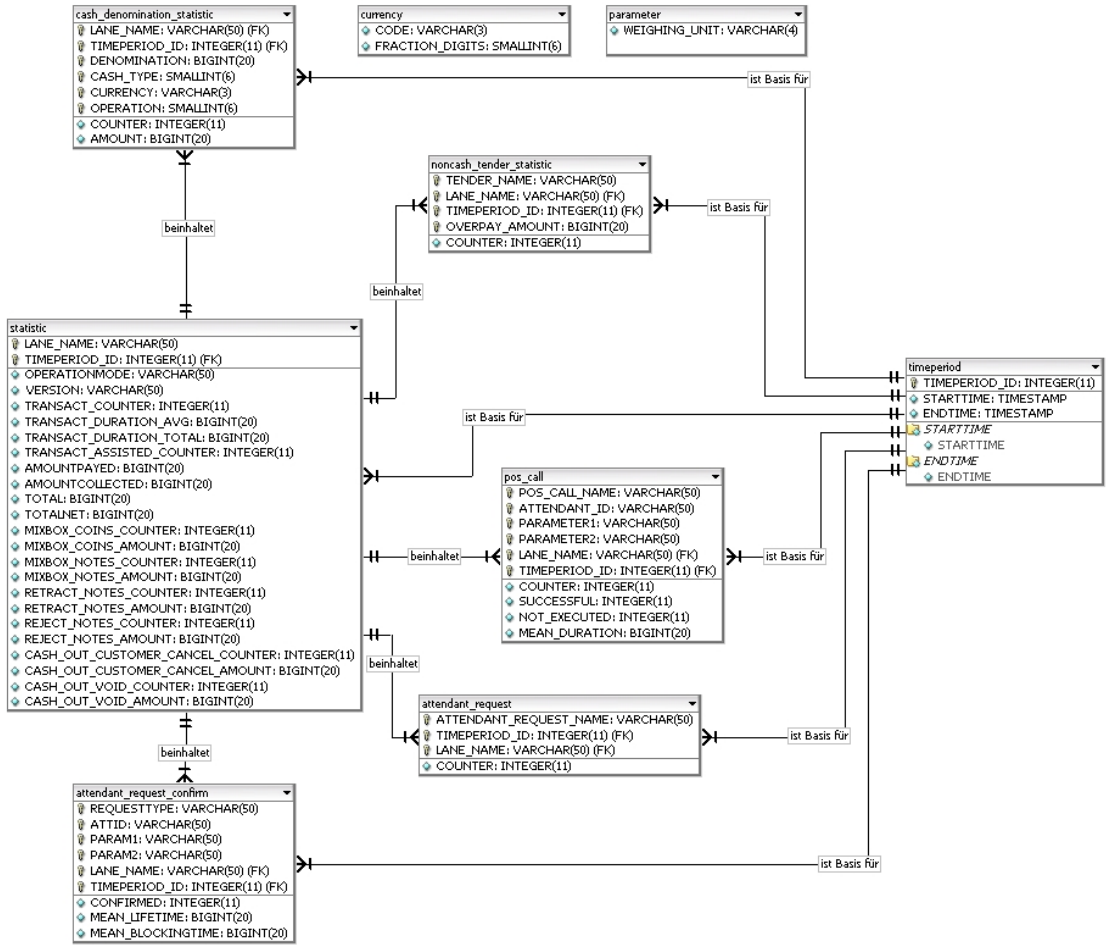

Abb. 24: Entity-Relationsship-Modell der Statistik-Datenbank (Quelle: eigene Darstellung)

Das Datenmodell lässt sich einfach lesen. Die beiden Periodengrenzen einer Aufzeichnungs-Periode werden in der Tabelle „timeperiod“ hinterlegt und erhalten eine ID als Primärschlüssel. Einer Aufzeichnungs-Periode werden über die Tabelle „statistic“ die Statistiken der verschiedenen Self-Checkout-Terminals zugeordnet. Jede dieser Statistiken enthält wiederum beliebig viele Ereignisse wie Geldauszahlungen POS Calls, oder Attendant Requests. Diese Ereignisse werden in eigenen Tabellen hinterlegt, welche über 1:N Relationen mit der Tabelle „statistic“ verbunden sind.

Somit enthält eine Ereignistabelle für die Zuweisung zu einer bestimmten Lane innerhalb einer bestimmten Aufzeichnungs-Periode als Fremdschlüssel die Attribute „TIMEPERIOD_ID“ und „LANE_NAME“. Die Primärschlüssel einer Ereignistabelle setzen sich aus den zur Identifizierung notwendigen Einzelschlüsseln zusammen. Zu beachten ist hier, dass die in den XML-Dateien verwendeten verschachtelten Identifier wie beispielsweise „methodname(attId,param1,param2)“ in der Datenbank als

Einzelschlüssel abgelegt werden, was die Größe der zusammengesetzten Primärschlüssel erklärt. Die als Alternative eingesetzte Apache Derby Datenbank besitzt dieselbe Datenbankstruktur, wie die hier beschriebene MySQL-Datenbank. Lediglich die verwendeten Datentypen unterscheiden sich an einigen Stellen.

Anhang D – Implementierung einer Apache Tapestry Webseite

Die Implementierung einer Apache Tapestry Webseite setzt sich aus drei Bestandteilen zusammen. Das Tapestry HTML-Template besteht aus reinem HTML-Code und teilt über den „jwcid“-Schlüssel dem Framework mit, an welchen Stellen Komponenten dynamisch eingefügt werden sollen.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>TransactionQuantity</title>
<link rel="stylesheet" type="text/css" href="page.css" />

</head>

<body jwcid="@Body" id="Standard" link="black" alink="black"
  vlink="black">

<table>
  <tr>
    <td>    </td>
    <td align="center">
    <h2><span jwcid="transactionQuantity_title">Number of Transactions</span></h2>
    <br />
     <img jwcid="chartObject" src="" border='1' style="border-color:lightgrey" width="600" height="300" />
 <br />
    <br />
    <br />
    <table border="0" jwcid="table" id="SingleBorder" cellpadding="5">
      <tr>
        <th>Start</th>
        <th>End</th>
        <th>Counter</th>
      </tr>
      <tr>
        <td>2007-08-01 00:00:00</td>
        <td>2007-08-08 00:00:00</td>
        <td>1234</td>
      </tr>
    </table>
    </td>
  </tr>
</table>
</body>
</html>
```

Abb. 25: HTML-Template von Apache Tapestry (Quelle: eigene Darstellung)

Damit das Framework weiß, welche Art der Komponente gemeint ist, werden die einzufügenden Komponenten innerhalb einer XML-Konfigurationsdatei mit der Dateiendung .page spezifiziert. Dabei werden der Typ der Komponente angegeben und entsprechend des Tapestry API die benötigten Attribute beschrieben. So gibt das Attribut „source“ innerhalb der Spezifikation der „Contib:Table“-Komponente beispielsweise die Datenquelle (Getter-Methode) an, mit der die dynamisch erzeugte Tabelle gefüllt werden soll.

```
<?xml version="1.0"?>
<!DOCTYPE page-specification PUBLIC
"-//Apache Software Foundation//Tapestry Specification 4.0//EN"
"http://jakarta.apache.org/tapestry/dtd/Tapestry_4_0.dtd">

<page-specification class="com.wn.retail.iscan.base.statisticreports.transactions.TransactionQuantity">

  <bean name="evenOdd" class="org.apache.tapestry.bean.EvenOdd">
    <set name="even" value="false"/>
  </bean>

  <component id="table" type="Contrib:Table">
    <binding name="source"  value="tableRows"/>
    <binding name="columns" value="columnNames"/>
    <binding name="rowsClass" value="beans.evenOdd.next"/>
    <binding name="pagesDisplayed" value="10"/>
  </component>

  <component id="chartObject" type="Image">
    <binding name="image" value="imageAsset"/>
  </component>

  <component id="transactionQuantity_title" type="Insert">
    <binding name="value" value="message:transactionQuantity" />
  </component>

</page-specification>
```

Abb. 26: XML-Konfigurationsdatei (.page) von Apache Tapestry (Quelle: eigene Darstellung)

Dazu wird mit dem Element <page-specification> auf ein Java-Objekt verwiesen, welches die zur Konstruktion der Tapestry Komponenten notwendigen Getter-Methoden zur Verfügung stellt. In diesem Fall ist das die Klasse TransactionQuantity. Diese Klasse wird bei der Anfrage der Webseite von Tapestry automatisch als Objekt initialisiert oder falls dieses bereits existiert neu geladen. Die implementierten Methoden getTableRows(), getColumnNames() und getImageAsset() dienen dazu, die in der XML-Datei spezifizierten Tapestry Komponenten mit einem dynamisch erzeugten Inhalt zu befüllen. Dabei kann die Klasse TransactionQuantity auf andere Java-Klassen zurückgreifen um beispielsweise auf die Datenbank zuzugreifen oder die Aggregation der Statistikdaten durchzuführen.

```java
/*
 * File: TransactionQuantity.java
 *
 * Copyright (c) 2004, 2005 Wincor Nixdorf International GmbH,
 * Heinz-Nixdorf-Ring 1, 33106 Paderborn, Germany
 * All Rights Reserved.
 *
 * This software is the confidential and proprietary information
 * of Wincor Nixdorf ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered
 * into with Wincor Nixdorf.
 */

package com.wn.retail.iscan.base.statisticreports.transactions;

import java.util.ArrayList;
import java.util.Collection;

import org.apache.tapestry.IAsset;
import org.apache.tapestry.asset.ExternalAsset;
import org.apache.tapestry.event.PageBeginRenderListener;
import org.apache.tapestry.event.PageEvent;
import org.jfree.data.general.Dataset;

import com.wn.retail.iscan.base.statisticreports.core.BasePageInheritance;
import com.wn.retail.iscan.base.statisticreports.core.PropertiesReader;
import com.wn.retail.util.build.NoObfuscate;

/**
 * @author Johannes W., Wincor-Nixdorf International GmbH
 *
 * The abstract class TransactionQuantity extends the class BasePageInheritance and represents the
 * Java logic part of the Tapestry view elements TransactionQuantity.html and TransactionQuantity.page.
 * TransactionQuantity implements a PageBeginRenderListener which calls the method PageBeginRender()
 * when TransactionQuantity.html is requested.
 * The methods getTableRows() and getColumnNames() provide the data for the HTML table.
 * The method getImageAsset() commits an image URL used to generate a chart object with the
 * ChartGeneratorServlet class.
 */
public abstract class TransactionQuantity extends BasePageInheritance implements PageBeginRenderListener
{

    @NoObfuscate
    public Collection getTableRows()
    {
        TransactionQuantityLogic logic = new TransactionQuantityLogic(getDb(), getConditions(), getLocales());
        Collection rowslist = null;
        rowslist = logic.getBusinessObjects();
        return rowslist;
    }

    @NoObfuscate
    public String getColumnNames()
    {
        String header = null;
        PropertiesReader reader = new PropertiesReader(getLocales());

        String begin = reader.getProperty("Begin");
        String end = reader.getProperty("End");
        String transactions = reader.getProperty("Transactions");

        header = new String(begin + ":Begin, " + end + ":End, " + transactions + ":Transactions");
        return header;
    }

    @NoObfuscate
    public IAsset getImageAsset()
    {
        return new ExternalAsset("chart?type=transactionQuantity", null);
    }

    @NoObfuscate
    public void pageBeginRender(PageEvent event)
    {
        TransactionQuantityLogic logic = new TransactionQuantityLogic(getDb(), getConditions(), getLocales());
        getChartData().setChartDataArray(new ArrayList<Dataset>());
        getChartData().getChartDataArray().add(logic.getChartDataset());
    }
}
```

Abb. 27: Java-Klasse von Apache Tapestry (Quelle: eigene Darstellung)

Markenerklärung

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenzeichen usw. können auch ohne besondere Kennzeichnung geschützte Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Fachhochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Abschließende Erklärung

Ich versichere hiermit, dass ich die vorliegende wissenschaftliche Arbeit selbstständig und ohne fremde Hilfe angefertigt und keine andere als die angegebene Literatur benutzt habe. Alle von anderen Autoren wörtlich übernommene Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit sind besonders gekennzeichnet. Diese Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

_____ _____

(Datum) (Unterschrift)