ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Konzeption und Implementierung einer prozess-orientierten Finanztransaktions Verwaltung- und Analyse Anwendung für mobile Endgeräte

Masterarbeit an der Universität Ulm

Vorgelegt von:
Fabian Maier
fabian.maier@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Dr. Vera Künzle

Betreuer:
Johannes Schobel

2014

Fassung 20. Oktober 2014

Kurzfassung

Im Bereich des Online-Banking existieren viele verschiedene Protokolle zum Austausch von Informationen und Anwendungen, welche diese Protokolle implementieren und diese Informationen verarbeiten. Durch wiederkehrende Aufgaben, wie beispielsweise dem Abrufen neuer Transaktionen oder dem Tätigen von Überweisungen (beispielsweise Daueraufträge), empfiehlt sich der Einsatz eines Prozessmanagement-Systems.

Ziel der vorliegenden Arbeit ist es, ein System zu konzipieren, welches die Vorteile eines Prozessmanagement-Systems in einem Online-Banking System nutzen kann. Hierbei sollen Probleme im Zusammenspiel dieser Komponenten identifiziert und Lösungen entwickelt dafür werden.

Die vorliegende Arbeit erläutert etablierte Technologien in diesem Umfeld. Darüber hinaus werden wichtige Banking Standards sowie ausgewählte, bereits existierende Banking Anwendungen vorgestellt und miteinander verglichen.

Mit den Erkenntnissen aus den untersuchten Technologien, Banking Standards und Anwendungen konnten die Anforderungen an eine prozess-orientierte Finanztransaktions Verwaltung- und Analyse Anwendung definiert werden. Nach der Konzeption einer grundlegenden Architektur für ein solches System wurde mit EMBS, dem *Extended Mobile Banking System*, eine prototypische Implementierung dieses Konzeptes umgesetzt. Hierbei konnten Probleme in dieser Domäne entdeckt und Lösungen entwickelt werden.

Ebenfalls konnten einige Aspekte identifiziert werden, welche zu zusätzlichen Forschungsfragestellungen führten.

Danksagung

Vielen Dank an meine Gutachter, Prof. Dr. Manfred Reichert und Dr. Vera Künzle, für ihre Unterstützung bei meiner Masterarbeit. Weiteren Dank an meinen Betreuer Johannes Schobel, welcher mir bei jedem Problem hilfreich zur Seite stand.

Auch meiner Familie, welche mich immer unterstützte gilt mein Dank.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Ziele der Arbeit 2

1.2 Aufbau der Arbeit 3

2 Grundlagen **5**

2.1 Home Banking Computer Interface 5

2.2 Prozessmanagement und Geschäftsprozesse 6

2.3 Webservices 8

3 Related Work **11**

3.1 Etablierte Banking Standards 11

3.1.1 Banking Communication Standard (BCS) 12

3.1.2 Electronic Banking Internet Communication Standard (EBICS) . . 12

3.1.3 Multi Bank Standard (MBS) 12

3.1.4 Open Financial Exchange 13

3.1.5 Zusammenfassung 13

3.2 Bankingsoftware von Kreditinstituten und der Privatwirtschaft 14

3.2.1 Ausgewählte Produkte der Sparkassen 15

3.2.2 StarMoney 17

3.2.3 Wiso – Mein Geld 18

3.2.4 numbrs 19

3.3 Zusammenfassung 20

4 Anforderungen **21**
4.1 Funktionale Anforderungen . . . 21
4.2 Nichtfunktionale Anforderungen . . . 23
4.3 Technische Anforderungen . . . 23
4.4 Zusammenfassung . . . 24

5 Architektur **27**
5.1 Übersicht des Gesamtsystems . . . 27
5.2 EMBS Architektur . . . 28
5.2.1 Datenbank und ORM . . . 28
5.2.2 BPMS-Schnittstelle . . . 29
5.2.3 HBCI-Schnittstelle . . . 30
5.2.4 REST-Schnittstelle . . . 30
5.2.5 Graphical User Interface (GUI) . . . 30
5.2.6 Kernanwendung . . . 30
5.2.7 Überblick . . . 31

6 Umsetzung und Implementierung **33**
6.1 HBCI-Schnittstelle . . . 34
6.1.1 Aufbau einer HBCI-Nachricht . . . 35
6.1.2 HBCI-Dialog . . . 36
6.1.3 HBCI Bibliothek . . . 48
6.2 EMBS-Prozess . . . 50
6.2.1 Die REST-Aktivität in AristaFlow . . . 53
6.2.2 BeanShell-Skript Aktivität . . . 54
6.2.3 Aktivität zur Benachrichtigung des Benutzers . . . 55
6.3 Zusammenfassung der Implementierung . . . 58

7 Fazit **59**
7.1 Ausblick . . . 60

1

Einleitung

Im Bereich des Internetbanking gibt es für Banking Anwendungen viele wiederkehrende Aufgaben, wie beispielsweise das Abrufen von Umsätzen oder das Überweisen von Rechnungen. Zudem gibt es viele verschiedene Protokolle zur Kommunikation und Darstellung von Daten der Banksysteme.

Oft ist es kompliziert an solche Transaktionsdaten zu kommen und selbige in ein verwertbares Format zu bringen, da diese nur bei der Bank vorliegen. Dies hat zur Folge, dass die Entwicklung neuer Anwendungen in diesem Bereich sehr aufwendig ist. Um einfache Aktionen durchzuführen, wie etwa eine Benachrichtigung beim Eingehen neuer Umsätze, entsteht in keinem Verhältnis stehender Aufwand.

Aus der Sicht des Programmierers der Anwendung kommt hinzu, dass weitere Probleme im Bezug auf die Entwicklung auftreten. Ein Benutzer kann mehrere Konten haben, welche mitunter bei unterschiedlichen Kreditinstituten liegen können.

Hinzu kommt, das oft mehrere Anwendungen für die Weiterverarbeitung der Daten zuständig sind. Neben dem einfachen Bankingportal des Kreditinstitutes ist bei Unternehmen oft ein Verwaltungssystem, wie zum Beispiel SAP, im Einsatz und muss über solche Vorgänge informiert werden. Diese Anwendungen können in verschiedenen Programmiersprachen geschrieben, auf verschiedenen Rechnerarchitekturen kompiliert und für verschiedene Betriebssysteme entwickelt worden sein, was einen Austausch zwischen den unterschiedlichen Anwendungen zusätzlich erschwert.

Diese Probleme sind aus anderen Anwendungsbereichen wohl bekannt und es stehen etablierte Konzepte zur Verfügung, um sie zu lösen.

- Immer wiederkehrende Aufgaben lassen sich in Prozessmanagement-Systemen abbilden und kontrolliert ausführen.
- Durch die Entwicklung von erweiterbaren Systemen lassen sich Grundfunktionen, wie zum Beispiel das Kommunizieren mit Kreditinstituten, bei weiteren Entwicklungen wiederverwenden.
- Mithilfe von Webservices lassen sich von unterschiedliche Programmiersprachen, Betriebssysteme und Rechnerarchitekturen abstrahieren.

1.1 Ziele der Arbeit

Um die Masse an gebräuchlichen Bankingstandards in einer Prozessmanagements-Umgebung verwenden zu könne, müssen diese über eine Schnittstelle so gekapselt werden, das ein Zugriff aus dem Prozessmanagement-System möglich ist.

Hierzu soll eine Architektur entwickelt werden, welche sowohl mit Prozessmanagement-Systemen umgehen, als auch mit Kreditinstituten kommunizieren kann. Hierbei soll das Prozessmanagement-System zur Kontrollflusssteuerung verwendet werden. Aus Sicherheitsgründen darf dieses jedoch nur wenige, für die Steuerung des Ablaufs unerlässliche, Daten zur Verarbeitung erhalten.

Auf dieser Architektur sollen nun, durch den Einsatz von Webservices, Anwendungen entwickelt werden, welche nicht auf eine bestimmte Konfiguration der Umgebung ange-

wiesen sind.
Eine Umsetzung von neuen Funktionalitäten soll so leichter zu umzusetzen sein. Durch einen Abgleich der schon vorhandenen und neuen Umsätzen lässt sich eine Benachrichtigung des Benutzers implementieren. Ebenfalls soll es so möglich sein, eine Aufteilung des verfügbaren Kapitals für bestimmte Verwendungszwecke ermöglicht werden. Eine automatische Zuordnung von Umsätzen in bestimmte Töpfe soll ebenfalls möglich sein, sobald diese abgebucht oder überwiesen werden.
Durch die Unterstützung der Anwendung durch ein Prozessmanagement-System kann der Ablauf leicht verändert und adjustiert werden. Außerdem kann ein Endanwender des Systems seine eigenen Prozesse für Überweisungen, das Empfangen von neuen Transaktionen oder einem andereren Geschäftsprozess der Bank entwickeln und beliebig erweitern.

Dies wurde mit dem, für die vorliegende Arbeit entwickelten System, *Extended Mobile Banking System (EMBS)* umgesetzt. Dieses gibt Entwicklern die Möglichkeit, mit den Servern der Kreditinstitute sowie einem Prozessmanagement-System zu kommunizieren. Dem Prozessmanagement-System wurde ebenfalls die Möglichkeit gegeben, mit EMBS Informationen auszutauschen, und so die für den Kontrollfluss wichtigen Daten zu übermitteln.

Als weiteres Feature soll das Aufteilen des vorhanden Kapitals auf verschiedene Töpfe implementiert werden. So ist es möglich, Umsätze, welche Filterkriterien erfüllen, automatisch von zugehörigen Töpfen zu addieren oder zu subtrahieren. Diese Filterkriterien könnten etwa der Betreff oder der Absender der Transaktion sein. Wenn der Benutzer dies wünscht, wird er automatisch über diese neuen Umsätze informiert.

1.2 Aufbau der Arbeit

Der weitere Aufbau der Arbeit gliedert sich wie folgt: Nach dieser Einleitung werden in Kapitel 2 wichtige Technologien und Begriffe für das weitere Verständnis der Arbeit erläutert. Das Kapitel 3 stellt unterschiedliche etablierte Bankingprotokolle vor und vergleicht diese miteinander. Darüber hinaus werden verschiedene bereits existierende

Banking Anwendungen diskutiert. Darauf folgen in Kapitel 4 funktionale, nichtfunktionale sowie technische Anforderungen an das zu konzipierende Gesamtsystem. Diese Anforderungen werden in Kapitel 5 anschließend in eine Architektur überführt, deren einzelne Komponenten sowie deren Wirkungsweise diskutiert werden. Kapitel 6 erklärt am Beispiel eines Umsatzabrufs die technische Umsetzung und Implementierung der Komponenten, sowie die konkrete Verwendung des HBCI Protokolls. Das Kapitel 7 schließt die Arbeit mit einem Fazit und gibt einen Ausblick auf mögliche Erweiterungen des Systems.

2

Grundlagen

Zum besseren Verständnis der Arbeit werden in diesem Kapitel einige Grundlagen erläutert. Dazu wird in Kapitel 2.1 das *Home Banking Computer Interface (HBCI)*, ein Protokoll zur Kommunikation mit deutschen Banken erklärt. Danach beschreibt Kapitel 2.2 einige Grundlagen zu *Prozessmanagement* und *Geschäftsprozessen*. Zuletzt wird in Kapitel 2.3 ein Überblick über *Webservices* gegeben.

2.1 Home Banking Computer Interface

Das Home Banking Computer Interface (HBCI) ist ein vom zentralen Kreditausschuss entwickelter Online-Banking Standard [Zenc]. Dieser dient der Kommunikation zwischen Kreditinstituten und Kunden über das Internet. Der Kunde hat so die Möglichkeit, seine

Bankgeschäfte online abzuwickeln. Neben HBCI existieren noch andere Standards, von denen einige Kapitel 3.1 näher beleuchtet werden.

Die Version 1.0 dieses Standards wurde im Jahr 1997 veröffentlicht. Als erste funktionsfähige Version gilt jedoch erst Version 2.01 aus dem Jahre 1998, da hier neue Geschäftsvorfälle implementiert wurden und Fehler im Protokoll behoben wurden. Im Jahre 1999 folgte Version 2.1, im Jahr 2000 Version 2.2. Als Sicherheitsverfahren wurden in dieser Zeit Disketten oder Chipkarten Terminals eingesetzt.

Durch eine rasche Änderung des Nutzerverhaltens in dieser Zeit (das Internet war nun der breiten Masse zugänglich), waren hier schnell Limitierungen im Protokoll ersichtlich. Deshalb entstand eine inoffizielle Version 2.2+. Diese implementierte auch das *PIN/TAN Verfahren (Persönliche Identifikationsnummber/Transaktionsnummer)*. Hierdurch mussten Bankkunden nicht mehr Disketten oder Chipkarten Terminals einsetzen.

Seit der Version 3.0 (aus dem Jahr 2002) wurde das Protokoll in *Financial Transaction Service (FinTS)* umbenannt, gleichzeitig wurde das nun schon verbreitete PIN/TAN Verfahren auch in die offiziellen Versionen übernommen.

Wurde bisher noch zur Darstellung der Daten eine Trennungszeichensyntax verwendet (siehe dazu auch Kapitel 6.1.1) und zur Übertragung eine TCP-Verbindung genutzt, so kam mit FinTS 4.0 in 2004 eine neue Spezifikation des Standards heraus, welche nun die Übertragung der Daten per *XML* und *HTTP*/*HTTPs* unterstützt.

Mehr Informationen zum Aufbau und der Benutzung von HBCI werden in Kapitel 6.1 gegeben.

2.2 Prozessmanagement und Geschäftsprozesse

Ein Geschäftsprozess besteht aus Einzeltätigkeiten, sogenannten *Aktivitäten*, welche ausgeführt werden, um ein definiertes Ziel zu erreichen. Damit die Aktivitäten Informationen austauschen können, werden *Datenelementen*, in welchen beliebige Daten gespeichert werden können, verwendet. Die Aktivitäten werden mit *logischen Verknüp-*

fungen aneinandergereiht, um einen *Prozess* abzubilden. Ein einfaches Beispiel für einen solchen Geschäftsprozess ist in Abbildung 2.1 zu sehen.

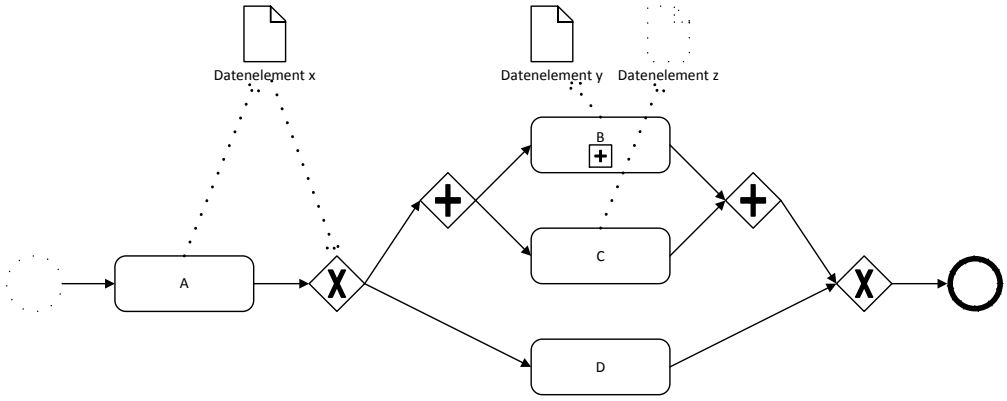

Abbildung 2.1: Beispiel für einen Geschäftsprozess

Jeder Prozess hat genau einen *Start-* und genau einen *Endpunkt*. Über Datenelemente erhält der Prozess Informationen. Sie können von Aktivitäten und Gateways gelesen oder geschrieben werden. In diesem Beispiel schreibt Aktivität A in Datenelement x einen Wert, welcher von dem nächsten Element, einem *XOR-Gateway*, gelesen wird.

Anhand der Informationen die aus dem Datenelement gelesen werden, kann der Gateway entscheiden, ob der „obere“ oder der „untere“ Pfad durch den Prozess genommen wird. Da es sich bei dem XOR-Gateway um ein *exklusives Oder* handelt, ist es nicht möglich beide Pfade gleichzeitig zu wählen.

Wenn der obere Pfad gewählt wird, wird anschließend ein *AND-Gateway* ausgeführt. Dies bewirkt, das sowohl der Pfad zu *Subprozess* B als auch der Pfad zu Aktivität C ausgeführt wird.

Innerhalb des eingebundenen Subprozess B muss sich ein valider Geschäftsprozess befinden. Das heißt, der Subprozess besitzt ebenfalls genau einen Start- und Endknoten. Ein Subprozess bietet den Vorteil, dass er auch in anderen Prozessen wiederverwendet werden kann. Darüber hinaus abstrahiert ein Subprozess und bietet so eine übersichtlichere Darstellung. Sobald sowohl Subprozess B als auch Aktivität C ausgeführt und

abgeschlossen wurden, kann die Ausführung nach dem *AND-Gateway-Join* weitergehen. Nach dem *XOR-Gateway-Join* ist der Prozess beendet.

Dieses Prozess-Modell ist die Vorlage für eine *Prozessinstanz*. Eine Prozessinstanz ist dabei die konkrete Ausführung des Modells zu einem bestimmten Zeitpunkt mit bestimmten Daten.

In dieser Arbeit wurde AristaFlow [DRRM+09] benutzt um mit dem ADEPT Prozessmodell [Rei00] die vorliegende Aufgaben zu modellieren und auszuführen. Mehr hierzu findet sich in Kapitel 6.2.

2.3 Webservices

Ein Webservice bietet Anwendungen die Möglichkeit, automatisiert auf einen entfernten Dienst über das Internet zuzugreifen. Dieser kann sowohl Daten lesen, als auch schreiben.

Ob Webservice und Client auf verschiedenen Betriebssystemen, Programmiersprachen oder Rechnerarchitekturen betrieben werden, spielt dabei keine Rolle. Es gibt verschiedene Möglichkeiten Webservices zu implementieren. Beispiele dafür sind *SOAP (Simple Object Access Protocoll)* [W3C], *RPC (Remote Procedure Call)* [Whi] oder *REST (Representational State Transfer)* [Fie00].

In dieser Arbeit wurde ein Webservice implementiert, welcher auf der REST-Architektur basiert. Hierbei sind einige Regeln zu beachten.

- Jede Ressource muss eine eindeutige *URL (Uniform Resource Locator)* besitzen, über welche sie von Anwendungen angesprochen werden kann. Hierbei ist eine Ressource ein Objekt, auf welches eine Anwendung zugreifen kann.
- Es darf unterschiedliche Repräsentationen der Ressource geben. So kann diese, zum Beispiel, je nach Anfrage entweder als XML- oder als JSON-Objekt zurückgegeben werden. Auch bei der Anfrage können unterschiedliche Repräsentationen für die Daten verwendet werden, sofern diese vom Webservice akzeptiert werden. Ob dies möglich ist, wird üblicherweise im Header der Nachricht übermittelt.

- Durch das Verzichten auf Sessions ist ein Webservice zustandslos. Daher muss jede Anfrage an den Service sämtliche Informationen beinhalten, welche zu dessen Ausführung notwendig sind. Dies ist ein Vorteil, da Webservices hierdurch sehr leichtgewichtig implementiert werden können und weniger Daten vorhalten müssen.
- Die vier wichtigsten Operationen sind: **GET**, **PUT**, **POST** und **DELETE**.
  - GET beschafft Informationen und verändert in der Persistenzschicht keine Daten. GET ist somit ein rein lesender Zugriff.
  - POST erstellt eine neue Ressource. Da die neue Ressource noch nicht adressierbar ist, verweist die URL auf die übergeordnete Ressource. Erst wenn die Ressource gespeichert wurde, bekommt sie ihre eindeutige, adressierbare Identifikation.
  - PUT ändert eine bestehende Ressource oder legt diese, falls sie noch nicht vorhanden ist, an.
  - Mit DELETE wird die angegebene Ressource gelöscht.
  - Seit HTTP 1.1 steht zusätzlich eine PATCH Methode zur Verfügung [IET]. Diese bietet, im Gegensatz zu PUT, welches die Ressource überschreibt, die Möglichkeit, nur bestimmte Teile einer Ressource zu ändern.

  Wird GET, PUT, PATCH und DELETE mit der gleichen URL mehrfach ausgeführt, so muss das Ergebnis auf der Persistenzschicht dasselbe sein. Das heißt nach der ersten Anfrage ändert sich hier nichts mehr.

Nachdem *HBCI*, *Prozessmanagement* und Webservices erklärt wurden, kann in Kapitel 3 auf andere Standards im Banking-Bereich eingegangen werden und ausgesuchte Online-Banking Anwendungen vorgestellt werden.

3

Related Work

In diesem Kapitel werden andere Online-Banking Standards aus dem europäischen und amerikanischen Raum, welche sich in diesem Bereich durchgesetzt und etabliert haben, näher betrachtet. Auch andere, schon existierende Online-Banking Lösungen werden in Kapitel 3.2 verglichen, um eine Übersicht zu gewinnen. Die hieraus gewonnen Erkenntnisse werden in Kapitel 3.3 zusammengefasst.

3.1 Etablierte Banking Standards

Weltweit gibt es viele verschiedene Standards im Online-Banking Bereich. In Deutschland werden BCS (siehe Kapitel 3.1.1), HBCI und EBICS (siehe Kapitel 3.1.2) genutzt. Österreich hat mit MBS (siehe Kapitel 3.1.3) einen eigenen Standard geschaffen. In

den Vereinigten Staaten von Amerika hat sich der OFX Standard (siehe Kapitel 3.1.4) durchgesetzt hat.

3.1.1 Banking Communication Standard (BCS)

Im DFÜ-Abkommen von 1995 wurde der *Banking Communication Standard (BCS)* [Zenb] zur Übertragung von Daten zwischen Kunden und Kreditinstituten vom *Zentralen Kreditauschuss* beschlossen [Zenc]. Dabei werden *File Transfer, Access and Management (FTAM)*-Dateien zwischen Kreditinstitut und Kunde übertragen. Dies funktioniert jedoch nur über eine vollwertige ISDN-Verbindung. So entstehen für den Zugang zum BCS-System für den Kunden Kosten. Eine asymetrische Verschlüsselung per *Public / Private Key* ist möglich. Bis 2010 war es für Banken in Deutschland verpflichtend, neben Standards ihrer Wahl den BCS Standard anzubieten.

3.1.2 Electronic Banking Internet Communication Standard (EBICS)

Ab dem 1. Januar 2008 sind Banken in Deutschland verpflichtet den *Electronic Banking Internet Communication Standard (EBICS)* anzubieten [Zena]. Auch gibt es Kooperationen mit Frankreich zur Verwendung von EBICS [Die].

Das als Nachfolger für BCS ebenfalls vom zentralen Kreditausschuss verabschiedete Protokoll verschickt über eine mittels TLS verschlüsselte HTTP-Verbindung XML-Container. In diesem befinden sich die Daten zur Transaktion. Neben der Kommunikation zwischen Kunde und Kreditinstitut bietet EBICS die Möglichkeit für die Kommunikation zwischen Banken. Der Einsatz von EBICS findet vorallem in Firmen und Behörden statt.

3.1.3 Multi Bank Standard (MBS)

Multi Bank Standard (MBS) ist ein von allen österreichischen Banken verwendetes Bankingprotokoll, welches 1997 von der *Studiengesellschaft für die Zusammenarbeit im Zahlungsverkehr (STUZZA)* eingeführt wurde [RAC]. Über eine SSL verschlüsselte Verbindungen werden im *United Nations Electronic Data Interchange For Administration,*

Commerce and Transport (EDIFACT) [Uni] Format vorliegende Daten über *Multi-purpose Business Security over IP (MBS/IP)* ausgetauscht.
EDIFACT ist ein internationaler Standard für den Austausch strukturierter Geschäftsdaten zwischen Computersystemen. Verantwortlich für die Verwaltung und Erweiterung des Standards ist die *United Nations Economic Commission for Europe (UNECE)*.

3.1.4 Open Financial Exchange

Open Financial Exchange (OFX) wurde 1997 veröffentlicht. Federführend waren hier Microsoft und Intuit beteiligt. Vor allem in den Vereinigten Staaten von Amerika ist OFX weit verbreitet. Die Übertragung der Daten wird im Standard nicht ausdrücklich festgelegt, jedoch haben sich SSL und HTTP als etablierte Protokolle durchgesetzt. In der aktuellen Version (2.2.1) wird zum Aufbau und zur Strukturierung der Daten XML verwendet.

3.1.5 Zusammenfassung

Durch verschiedene nationale und internationale Ansprüche an die Online-Banking Kommunikation konnte sich bisher kein Standard international durchsetzen. Zur besseren Übersicht sind in Tabelle 3.1.5 die einzelnen Banking Standards zusammengefasst.

Da kein einheitlicher Banking Protokoll Standard existiert, musste aus den Standards der passendste ausgewählt werden. Bei der Wahl von HBCI (siehe dazu Kapitel 2.1) spielte neben dem kostenfreien Zugang zum Service der Banken auch die weite Verbreitung in Deutschland eine wichtige Rolle.

Im folgenden Kapitel werden nun ausgewählte existierende Bankinganwendungen näher betrachtet.

Tabelle 3.1: Übersicht über die Online-Banking Protokolle

Protokoll	Version	Datenformat	Übertragung	Pflichstandard
BCS	2.0	FTAM	ISDN	In Deutschland bis zum 31.12.2010
EBICS	2.5	XML	HTTPS	In Deutschland ab dem 01.01.2008
HBCI	2.2+	Trennzeichensyntax, angelehnt an UN/EDIFACT	TCP	Nein
FinTS	4.0	XML	Webservice / SOAP	Nein
OFX	2.1.1	XML	Webservice / SOAP	Nein
MBS	2.0	EDIFACT	MBS/IP	In Österreich

3.2 Bankingsoftware von Kreditinstituten und der Privatwirtschaft

Heutzutage existieren bereits zahlreiche Möglichkeiten zum Online-Banking. Diese stammen zum Teil von den jeweiligen Kreditinstituten, als auch aus der Privatwirtschaft. Kreditinstitute bieten meist die Möglichkeit, direkt über ihre Website als auch über Anwendungen für aktuelle mobile Betriebssysteme Online-Banking zu betreiben. Dies geht jedoch nur mit den Konten der jeweiligen Kreditinstitute. Um diesem Problem zu entgehen, gibt es sowohl für mobile als auch für klassische Betriebssystemen eine Fülle an Anwendungen, welche von anderen Firmen angeboten werden und teilweise mit mehreren Kreditinstituten gleichzeitig funktionieren.

Um einen Überblick der Leistungen der Angebote der Kreditinstitute zu bekommen, werden in Kapitel 3.2.1 verschiedenen ausgewählte Angebote der Sparkasse als Fallbeispiel betrachtet. Anschließend werden zwei seit längerem existieren Lösungen für Desktopsysteme betrachtet, *StarMoney* (siehe Kapitel 3.2.2) und *Wiso – Mein Geld* (siehe Kapitel 3.2.3).

Abschließend wird die mobile Anwendung numbrs für iOs vorgestellt, welche neben einer gelungen grafischen Oberfläche einfach zu handhaben ist.

3.2.1 Ausgewählte Produkte der Sparkassen

Die Sparkassen bieten sowohl Online-Banking über ihre Homepage an, als auch über mobile Anwendungen für Android, iOs [Spaa] und Windows Phone. Für klassische Desktop-Betriebssysteme bieten sie keine direkte Lösung an, empfehlen jedoch StarMoney von StarFinanz (siehe Kapitel 3.2.2).

Homepage der Sparkasse

Auf ihrer Homepage [Spad] bietet die Sparkasse zahlreiche Services an. Diese können jedoch vom Funktionsumfang je nach Gemeinde stark variieren.

Über die bereitgestellte Plattform gibt es die Möglichkeit den aktuellen Finanzstatus des Nutzers einzusehen und Umsätze für die einzelnen Konten bei den Sparkassen abzurufen. Überweisungen, Überträge und Daueraufträge können ebenfalls getätigt werden. Daneben informiert die Seite über aktuelle Angebote der Sparkasse und bietet die Möglichkeit direkt über ein abgeschlossenes System mit der Bank beziehungsweise dem zuständigen Kundenbetreuer zu kommunizieren.

Einige Sparkassen bieten darüber hinaus die Möglichkeit eines Kontoweckers, welcher über Umsätze informiert. Dieser arbeitet in 3 verschiedenen Modi: *Limitwecker*, *Umsatzwecker* und *Kontostandswecker*. Diese Modi lassen sich getrennt voneinander nutzen. Der Nutzer muss beim Einrichten des Dienstes festlegen, ob er per E-Mail, Push-Nachricht oder SMS benachrichtigt werden möchte.

- **Limitwecker:** Der Benutzer kann einen beliebigen Kontostand definieren, bei dem er benachrichtigt wird, wenn dieser unter- oder überschritten wird.
- **Umsatzwecker:** Hier kann der Benutzer festlegen, ob er bei bestimmten Umsätzen benachrichtigt werden möchte. Diese Kriterien sind vom Anwender frei definierbar, beispielsweise die Höhe des Umsatzes oder Stichworte im Verwendungszweck.

- **Kontostandswecker:** Beim Kontostandswecker wird der Nutzer einmal täglich über seine aktuellen Umsätze informiert. So ist er täglich über seine neusten Kontobewegungen informiert.

Eine Benachrichtigung per E-Mail ist dabei für den Benutzer kostenlos, für Push-Nachrichten auf das Mobiltelefon werden 4 Cent berechnet. Eine SMS kostet den Anwender 9 Cent.

Mobile Anwendung für Android

Aktuell existieren zwei Varianten der mobilen Anwendung der Sparkasse, *Sparkasse* und *Sparkasse+* [Spab]. *Sparkasse+* bietet die Möglichkeit, Konten von mehreren Sparkassen oder sogar anderen Kreditinstituten zu verwalten. Für die Anwendung, welche im App Store des jeweilige Betriebssystems verhanden ist, muss der Benutzer einmalig 89 Cent zahlen.

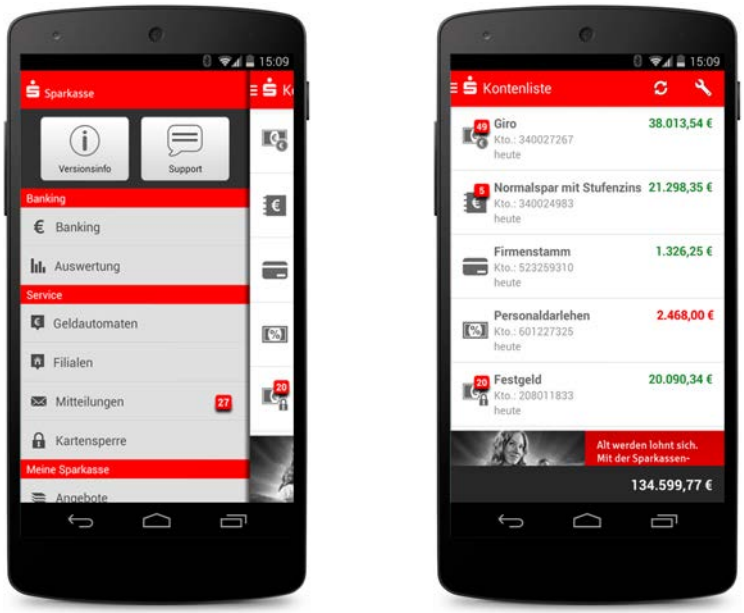

Abbildung 3.1: Screenshots der Anwendung *Sparkasse* [Spac]

Die Anwendung *Sparkasse* ist kostenlos, kann dafür nur Konten der Sparkasse einer Gemeinde verwalten. Der Funktionsumfang der beiden Anwendungen ist dabei identisch zu bereits beschriebenen Plattformen. Dies bedeutet der Benutzer kann sich seine Umsätze anzeigen lassen sowie Überweisungen und Überträge tätigen. Ebenfalls ist es möglich, die direkten Nachrichten der Bank zu Empfangen und Nachrichten an diese zu senden.

Sollte der Benutzer das smsTAN Verfahren nutzen, wird ihm aus Sicherheitsgründen jedoch nicht erlaubt, Aufträge für beispielsweise Überträge oder Überweisungen einzureichen. Hierdurch wird verhindert, das der Tan auf dem gleichen Gerät empfangen und benutzt wird.

3.2.2 StarMoney

StarFinanz [Sta] bietet drei Versionen seiner *StarMoney Software* an. StarMoney 9.0 für Privatkunden, StarMoney Business 6.0 sowie StarMoney Business 6.0 Plus für Geschäftskunden. Die Anwendungen existieren jeweils für Windows und Mac OS. Sie bezeichnen sich selbst als Marktführer für Online-Banking Software in Deutschland.

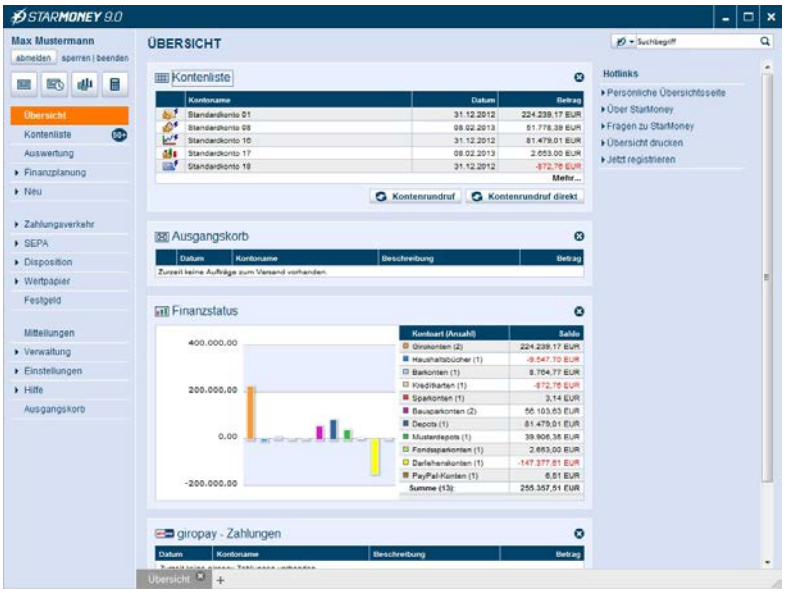

Abbildung 3.2: Screenshot von StarMoney [Sta]

StarMoney Business bietet als Mehrwert unter anderem Mehrbenutzerfähigkeit sowie Fremdwährungskonten und Auslandsüberweisungen. Um den EBICS-Standard (siehe Kapitel 3.1.2) nutzen zu können wird jedoch StarMoney Business Plus vorausgesetzt. EBICS richtet sich hauptsächlich an Firmenkunden und viele Vorgänge aus dem Privatkunden Bereich werden nicht unterstützt. Beispiele für nicht unterstützte Vorgänge sind Dauer- und Terminaufträge.

Standard Banking-Funktionen wie Überweisungen, Überträge sowie Einblick in die Umsätze bietet auch schon die Basisversion. Ein interessantes Feature abseits der Basisfunktionalität ist die Möglichkeit, Umsätze zu kategorisieren. Dies geschieht entweder per Hand für jeden Umsatz einzeln oder über eine automatische Funktion. Diese versucht, anhand der händisch zugewiesenen Umsätze Muster festzustellen. Diese Muster werden auf zukünftige Umsätze angewendet. Zur genaueren Funktionsweise der Mustererkennung äußert sich die Herstellerfirma nicht.

Über diese Kategorisierung lässt sich für einen einstellbaren Zeitraum anzeigen, wie sich die einzelnen Kategorien in absoluten und relativen Werten verhalten haben.

3.2.3 Wiso – Mein Geld

Wiso – Mein Geld von buhl-data [buh] liegt aktuell in der Version 2014 vor. Neben den üblichen Banking Vorgängen bietet es die Möglichkeit, besser mit Kategorien zu arbeiten. So lernt die Anwendung selbstständig über getätigte Kategorisierungen, Absender einer bestimmten Kategorie zuzuordnen. Auch hier können Konten bei mehreren Banken hinzugefügt werden, sofern diese den HBCI Standard anbieten. Andere Protokolle werden von *Wiso – Mein Geld* nicht unterstützt.

Mit der Anwendung ist es möglich, neben aktuellen Umsätzen auch einen Blick in die Zukunft zu werfen. Hierbei wird bewertet, welche Umsätze in anderen Monaten regelmäßig auftraten und so ein Ausblick für den nächsten Monat erstellt.

3.2.4 numbrs

Die Online-Banking Anwendung *numbrs* [Cen], wurde von der Schweizer Firma Centralway Switzerland AG entwickelt.

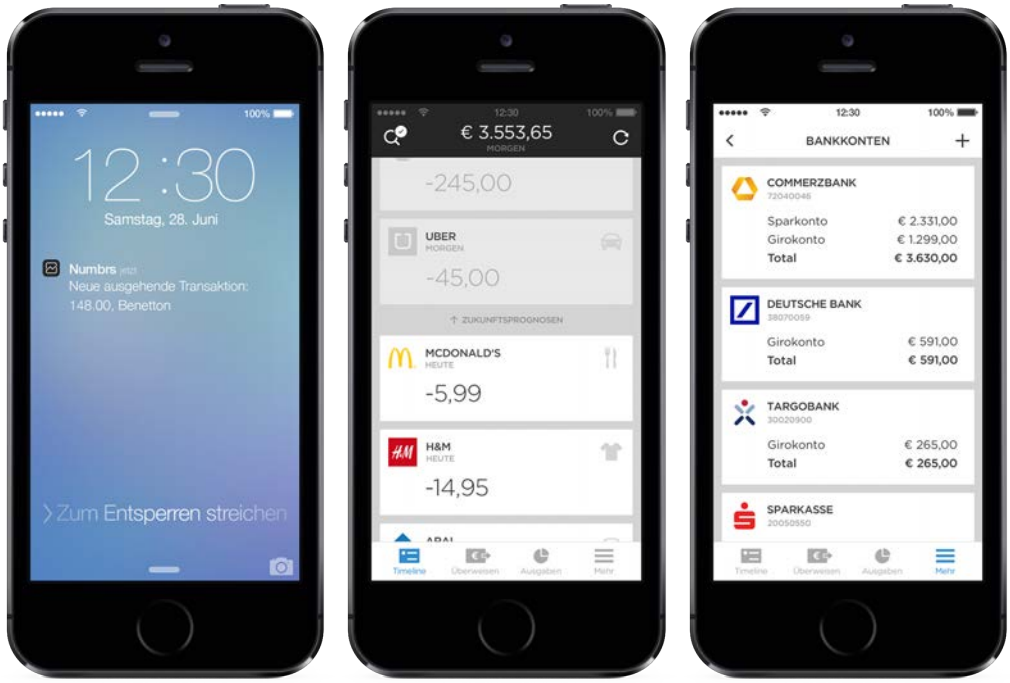

Abbildung 3.3: Screenshots von *numbrs* [Cen]

Mehrere Konten bei verschiedenen Kreditinstituten werden unterstützt und können in einer gemeinsamen Zeitleiste dargestellt werden. Eine Kategorisierung und Benachrichtigung über neue Umsätze ist ebenfalls möglich. Ähnlich wie bei *Wiso - Mein Geld* wird versucht, über Algorithmen zukünftige Umsätze zu berechnen. Über die Kategorisierung gibt es die Möglichkeit, sich Statistiken über Einnahmen und Ausgaben anzeigen zu lassen.

Numbrs für iOs wurde ansprechend gestaltet (siehe Abbildung 3.3), ist allerdings bisher nur für iOs erschienen. Es setzt ebenfalls auf den HBCI Standard.

3.3 Zusammenfassung

Die eigenen Lösungen der Kreditinstitute beschränken sich meist auf die Funktionalität ihrer Kernkompetenzen und Services aus dem eigenen Kreditinstitut. Daneben gibt es in der Privatwirtschaft für Endbenutzer Lösungen, welche über diese Grundleistungen hinausgehen. Sie bieten die Möglichkeit, die vergangen Umsätze unabhängig von den Instituten und deren interne Aufteilung der Konten aufzuteilen und ermöglichen dem Nutzer so eine wesentlich detailliertere Analyse.

Zum Aufteilen des Geldes für die Zukunft bieten allerdings keine etablierte Anwendungen entsprechende Funktionalität. Soll etwa für eine größere Investition Geld im voraus gespart werden, so erfährt der Benutzer hier keine Unterstützung durch seine Bankinganwendung.
Auch wenn der Benutzer eine Erweiterung der Anwendung benötigt wird er nicht unterstützt. Wenn etwa weitere Systeme, wie zum Beispiel ein SAP-System, über neue Transaktionen informiert werden sollen, kann er nicht den Prozess der bestehenden Anwendung ändern.

Nachdem in diesem Kapitel nun bestehende Lösungen betrachtet werden konnten, widmet sich das nächste Kapitel an die hieraus ableitbaren Anforderungen an ein eigenes System.

4

Anforderungen

In diesem Kapitel werden funktionale und nicht funktionale Anforderungen, die an EMBS gestellt werden erklärt und zusammengefasst. Darüber hinaus werden die technischen Aspekte, welche für die Entwicklung wichtig sind, beleuchtet.

4.1 Funktionale Anforderungen

Im Folgenden wird beschrieben, welche Funktionen das System haben soll.

Periodische Anforderung von Transaktionen: Das System soll periodisch, bei den ihm anvertrauten Konten, über die HBCI-Schnittstelle der Bank abfragen, ob neue Geschäftsvorfälle vorhanden sind.
Benachrichtigungen über neue Transaktionen: Sobald eine neue Transaktion vom

System erkannt wird, soll, wenn der Nutzer dies eingestellt hat, eine Benachrichtigung verschickt werden.

Der Benutzer soll mit Hilfe von *Töpfen* in der Lage sein, seine Finanzen unabhängig von der Kontenzuordnung bei der Bank organisieren zu können. Dies hilft ihm sein Geld bestimmten Nutzungszwecken zuzuordnen.

- **Töpfe erstellen:** Der Benutzer soll in der Lage sein seine Finanzen über verschiedene Töpfe zu organisieren.

- **Hierarchische Organisation der Töpfe:** Diese sollen hierarchisch angeordnet werden und miteinander verbunden werden können.

- **Periodischer Geldfluss:** Bei miteinander verbundenen Töpfen soll einstellbar sein, in welchem wiederkehrenden Zeitraum, welcher Betrag von einem in den anderen Topf fließt.

- **Manueller Geldfluss:** Ebenfalls soll es möglich sein, einmalig Geld zwischen Töpfen zu verschieben.

- **Geldfluss aufgrund von Transaktionseigenschaften:** Als letzte Möglichkeit zur Aufteilung soll es möglich sein, Transaktionen anhand ihrer Eigenschaften einem gewissen Topf zuzuweisen. Ein Beispiel hierfür wäre, dass jede Transaktion, welche von einem bestimmten Konto (beispielsweise dem Konto des Arbeitgebers) überwiesen wird, in einen vorher definierten Topf („Gehaltstopf“) kommen soll.

Schnittstellen: Um eine schnelle Entwicklung von weiteren grafischen Oberflächen für das System zu ermöglichen, sollen über REST-Schnittstellen alle relevanten Informationen bereitgestellt werden.

Vorteile eines Prozessmanagement-Systems nutzen: Es wird in dem System einige wiederkehrende Abläufe geben, wie zum Beispiel das Abholen und Verarbeiten der neuen Transaktionen, das Verschieben von Geld zwischen den Töpfen oder das Tätigen

von Überweisungen. Diese wiederkehrenden Abläufe können sich, durch Weiterentwicklung des Systems oder das Hinzufügen neuer Funktionen, im Laufe der Zeit verändern. Hierfür soll ein Prozessmanagement-System verwendet werden. Dies ermöglicht eine einfache spätere Anpassung und Bearbeitung der Abläufe.

4.2 Nichtfunktionale Anforderungen

Dieses Kapitel beschreibt allgemeine Eigenschaften des Systems.

Erweiterbarkeit: Da neben den bereits oben aufgeführten funktionalen Anforderungen einige weitere Funktionen denkbar und wünschenswert sind (zum Beispiel das Ausführen einer Überweisung), muss das System erweiterbar sein. Dies soll durch einen modularen Aufbau gewährleistet werden.

Persistenter Speicher: Da der Abruf und die Verarbeitung der Bank Transaktionen einige Zeit in Anspruch nimmt, ist eine Datenbank für die persistente Speicherung der bereits abgerufenen Transaktionen unerlässlich.

Benutzerfreundliche Oberfläche: Die Oberfläche soll benutzerfreundlich sein und mit möglichst wenig Lernaufwand bedienbar sein.

4.3 Technische Anforderungen

Daneben existieren auch technische Anforderungen an das System, welche in diesem Kapitel beschrieben werden.

Sicherheitsaspekte: Durch das Arbeiten mit hoch sensiblen Daten darf die Sicherheit nicht vernachlässigt werden. Da dies den Umfang dieser Arbeit übersteigen würde, müsste die Implementierung dieser Aspekte für einen produktiven Betrieb des Systems nachgeholt werden.

Skalierbarkeit: Lediglich die Akzeptanz der Nutzer, nicht die technische Umsetzung soll das Wachstum des Systems bestimmen. Daher ist schon während der Implementierung auf eine gute Skalierbarkeit des Systems Wert zu legen.

Atomare Transaktionen: Auf technischer Seite muss sichergestellt werden, das alle Transaktionen atomar sind und nicht etwa eine Überweisung abgebrochen wird, bevor sie komplett durchgeführt wurde.

4.4 Zusammenfassung

Da nun alle funktionale, nichtfunktionale und technischen Anforderungen definiert wurden, wird nun zur besseren Übersicht die Punkte der vorangegangen Unterkapitel in Tabelle 4.1 zusammengefasst.

Tabelle 4.1: Anforderungen an das Gesamtsystem

Anforderung	**Art**
Allgemeine Anforderungen an das System	
Periodische Anforderung von Transaktionen	funktional
Benachrichtigungen über neue Transaktionen	funktional
Erweiterbarkeit	nicht funktional
Benutzerfreundliche Oberfläche	nicht funktional
Persistenter Speicher	nicht funktional
Sicherheitsaspekte	technisch
Skalierbarkeit	technisch
Töpfe	
Töpfe erstellen	funktional
Hierarchische Organsation der Töpfe	funktional
Periodischer Geldfluss	funktional
Manueller Geldfluss	funktional
Geldfluss aufgrund von Transaktionseigenschaften	funktional
Schnittstelle des Systems	
REST-Schnittstelle	funktional
Prozessmanagement-System	
Vorteile eines Prozessmanagement-Systems nutzen	funktional

5

Architektur

Nachdem im vorherigen Kapitel wichtige Anforderungen des Gesamtsystems beschrieben wurden, sollen diese nun in eine Architektur übersetzt werden. In Kapitel 5.1 wird die Einordnung im Gesamtsystem dargestellt, in Kapitel 5.2 wird die Architektur innerhalb von EMBS aufgezeigt.

5.1 Übersicht des Gesamtsystems

Um eine bessere Übersicht über das Gesamtsystem zu bekommen ist dieses in Abbildung 5.1 dargestellt. Hier ist ebenfalls zu sehen, dass viele Menschen nicht mehr nur ein einziges Konto haben, oft verteilen sie ihre Konten gar auf mehrere Banken.
Die Verwaltung dieser Konten bei unterschiedlichen Banken soll durch EMBS vereinfacht werden. Durch die Fähigkeit mit unterschiedlichen Banken zu kommunizieren und

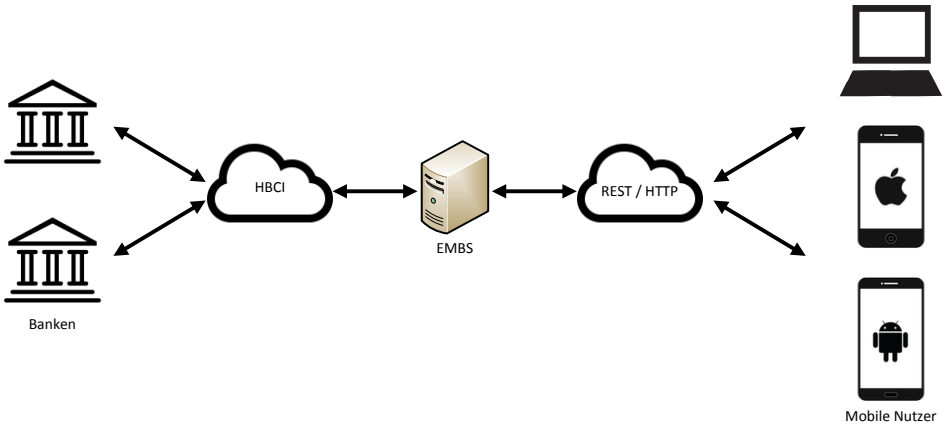

Abbildung 5.1: Übersicht des Gesamtsystems

die Daten zu aggregieren, kann das System (mithilfe von etablierten Standards in der Kommunikation) eine Schnittstelle bieten, um diese den Nutzern bereitzustellen.
Hierdurch bekommt der Benutzer die Möglichkeit EMBS auf einer Vielzahl mobiler Endgeräte zu nutzen.

5.2 EMBS Architektur

Dieses Kapitel befasst sich mit dem detaillierten Aufbau der Anwendung, welche in verschiedene Schichten aufgebaut ist. Abbildung 5.2 verdeutlicht diesen Aufbau.

5.2.1 Datenbank und ORM

Der Abruf der Transaktionen von den verschiedenen Bankinstituten und das anschließende Verarbeiten der Daten beansprucht Rechenzeit. Außerdem muss die Konfiguration, welcher der Nutzer anlegt, persistent sein.
Da all diese Daten strukturiert vorliegen ist die Nutzung eines Datenbanksystems ein Muss.

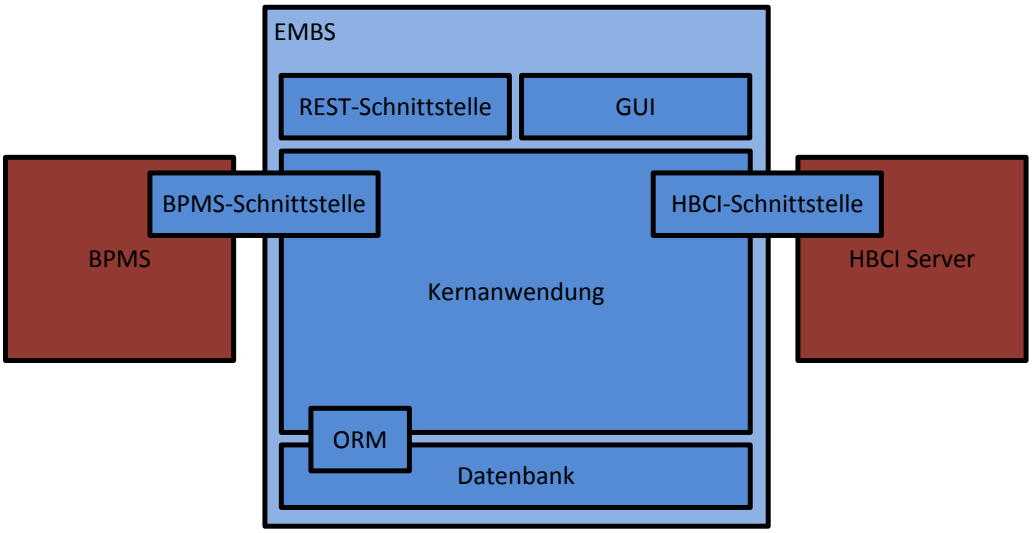

Abbildung 5.2: Architektur von EMBS

Allerdings ist die manuelle Pflege eines Datenbanksystems mühsam. Daher erfolgt die Persistierung aller Daten über Objektrelationale Abbildung (englisch object-relational mapping, ORM). Dies bietet viele Vorteile, einige davon werden nachfolgend erwähnt. Die zugrunde liegende Datenbank ist dabei völlig irrelevant, solange das genutzte ORM-Framework diese abstrahiert. Außerdem ermöglicht es das Deklarieren der zu persistierenden Daten direkt über die Klassendefinition. Zuletzt wird dem Framework überlassen die Daten in die Datenbank zu speichern sowie die Objekte bei Bedarf aus der Datenbank zu laden.

Gerade im Implementierungsprozess ermöglicht dies eine schnelle und effiziente Umsetzung von Änderungen am Datenbankmodell.

5.2.2 BPMS-Schnittstelle

Zur Verbindung der Kernanwendung vom EMBS mit dem Prozessmanagement-System ist eine Schnittstelle notwendig. Über diese können zuvor im BPMS definierte Prozesse-Modelle gestartet und so instanziiert werden.

Aufgrund der Auslagerung der Kommunikation mit diesem externen System in eine eigene Schicht ist es einfach möglich, die Konnektivität zu ändern oder gar das verwendete Prozessmanagement-System auszutauschen, ohne das im gesamten EMBS Anpassungen vorgenommen werden müssen.

5.2.3 HBCI-Schnittstelle

Die HBCI-Schnittstelle übernimmt die Kommunikation mit den Kreditinstituten. Müssen Daten an die Kernanwendung zurückgegeben werden, so werden diese vorher entsprechend aufbereitet oder transformiert.

5.2.4 REST-Schnittstelle

Über die REST-Schnittstelle kommunizieren die Benutzer oder fremde Anwendungen mit dem EMBS. Auch hier bietet es sich an, wie schon bei der BPMS-Schnittstelle, diese in eine eigene Schicht auszulagern, somit ist der Realisierungsaufwand deutlich geringer, falls eine alternative oder zusätzliche Schnittstelle (beispielsweise SOAP) zur Verfügung gestellt werden soll.

5.2.5 Graphical User Interface (GUI)

Über diese REST-Schnittstelle greift nun die grafische Oberfläche auf die vom EMBS bereitgestellten Inhalte zu. Sofern das mobile Endgerät die Möglichkeit hat, HTTP-Anfragen zu stellen, ist es möglich, dem Benutzer den Zugriff auf EMBS zu ermöglichen. Diese Möglichkeit ist bei allen weit verbreiteten mobilen Betriebssystemen (zum Beispiel Android, iOs oder WindowsPhone) gegeben.

5.2.6 Kernanwendung

Bisher sind die einzelnen Komponenten jedoch noch nicht miteinander verbunden. Außerdem fehlt es an der eigentlichen Logik, welche die beschafften Informationen

miteinander verknüpft. Dies geschieht in der Kernanwendung, welche die Komponenten verbindet und somit das Herzstück des Systems ist.

5.2.7 Überblick

In Abbildung 5.2 und den nachfolgenden Erklärungen wurde gezeigt, wie durch *ORM* das Design und der Zugriff auf die Datenbank deutlich komfortabler wird. Die *Datenbank* ermöglicht uns die Daten zu persistieren und somit von den anderen Komponenten verarbeitet zu werden. Über die *BPMS-* sowie die *HBCI-Schnittstelle* wird der Zugriff auf die beiden Fremdkomponenten im System realisiert. Die *REST-Schnittstelle* bietet der *Grafischen Oberfläche* die Möglichkeit auf die nun aggregierten und aufbereiteten Daten zuzugreifen.

6

Umsetzung und Implementierung

In diesem Kapitel werden detailliertere Einblicke in die Implementierung eines Systems zur prozess-orientierten Finanztransaktions-Verwaltung und -Analyse gegeben. In Kapitel 6.1 wird dabei näher auf die Implementierung und Umsetzung des HBCI-Protokolls über eine externe Bibliothek eingegangen. Das Kapitel 6.2 erklärt die Nutzung des Prozessmanagement-Systems im Kontext dieser Arbeit und stellt wichtige Aktivitäten im Prozessmodell vor.

Bevor diese Aspekte diskutiert werden, zeigt Abbildung 6.1 einen Gesamtüberblick über die erwähnten Komponenten und die Kommunikation dieser untereinander.

Über einen Webservice Aufruf ist das EMBS in der Lage, einen Prozess im Prozessmanagement-System zu instanziieren. Dieser Aufruf geschieht periodisch für alle Benutzer gemeinsam. Sobald dieser Prozess Informationen zur Verarbeitung benötigt, kann er diese über einen REST Aufruf vom EMBS erhalten. Sollten die gewünschten Informationen nicht bereits

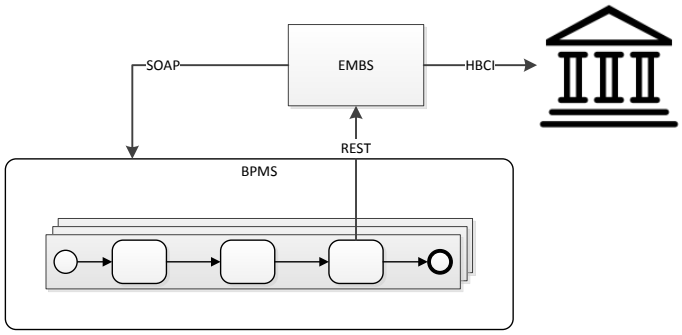

Abbildung 6.1: Kommunikation zwischen EMBS, BPMS und Kreditinstitut

in der Datenbank des EMBS vorhanden sein, so werden diese direkt über das HBCI Protokoll von dem entsprechenden Kreditinstitut angefordert, aufbereitet und in der Datenbank abgespeichert. Anschließend werden sie an die entsprechende Instanz des Prozessmanagements-Systems zurückgegeben.

6.1 HBCI-Schnittstelle

Um die Kommunikation mit verschiedenen Banken zu realisieren, wird das HBCI-Protokoll verwendet. Da die meisten deutschen Banken den HBCI Standard umsetzen (obwohl sie nicht verpflichtet sind den HBCI Standard anzubieten), ermöglicht dies dem System die Kommunikation.

Um einen Überblick über das Protokoll zu geben wird in Kapitel 6.1.1 exemplarisch der Aufbau einer solchen HBCI-Nachricht vorgestellt. Das Kapitel 6.1.2 erklärt den genauen Ablauf der einzelnen Nachrichten in einer solchen Kommunikation. Auf die Implementierung der Schnittstelle im entworfenen System wird in Kapitel 6.1.3 eingegangen.

Die hier gezeigten Nachrichten sind alle in der HBCI-Version 2.2+ [Bun] dargestellt. Dies liegt darin begründet, dass mit der Sparkasse Ulm ein Live-System vorlag, welches nur HBCI 2.2+ spricht.

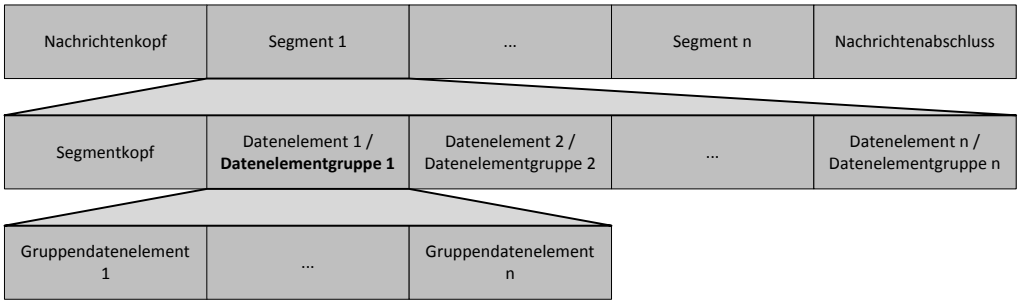

Abbildung 6.2: Übersicht über den Aufbau einer Nachricht

FinTS 4.0 arbeitet im Gegensatz zu HBCI 2.2+ nicht mehr mit einer Trennzeichensyntax sondern baut auf XML auf. Daher ist der Aufbau in aktuellen Versionen des Standards anders.

6.1.1 Aufbau einer HBCI-Nachricht

Für jede HBCI-Nachricht ist der Aufbau ganz genau vorgeschrieben (siehe Abbildung 6.2). Nach der Entschlüsselung der Nachricht liegen alle darin enthaltene Daten in lesbarer Form vor. Die verschiedene Segmente werden durch den *Nachrichtenkopf* (HNHBK) und den *Nachrichtenabschluss* (HNHBS) eingeschlossen. Diese (und andere) Felder werden in dem folgenden Kapitel 6.1.2 näher betrachtet. Als Trennzeichen zwischen dem Nachrichtenkopf, den einzelnen Segmenten, welche die Nutzdaten enthalten, und dem Nachrichtenabschluss werden Apostrophe **'** verwendet.

Ein *Segment* besteht dabei aus einem Segmentkopf sowie Datenelementen und Datenelementgruppen. Diese werden durch ein Pluszeichen **+** voneinander getrennt.

Datenelementgruppen werden aus Gruppendatenelementen zusammengesetzt. Hier wird der Doppelpunkt **:** als Trennzeichen verwendet.

Der Unterschied zwischen einem Datenelement und einer Datenelementgruppe ist, das die Datenelementgruppe eine beliebig Größe haben kann und so Daten in Listenform übertragen kann.

6.1.2 HBCI-Dialog

Anhand des Umsatzabrufes für mehrere Konten wird nun auf Zusammensetzung, Reihenfolge und Aufbau dieser Nachrichten detaillierter eingegangen. Dieser Ablauf wird in Abbildung 6.3 schematisch dargestellt. Zur besseren Übersicht wird auf den Nachrichtenkopf sowie den Nachrichtenabschluss verzichtet und nur die einzelnen Segmente dargestellt. Diese werden im weiteren Verlauf der Arbeit zeitlich linear durchgegangen und beim erstmaligen Auftauchen genauer betrachtet.

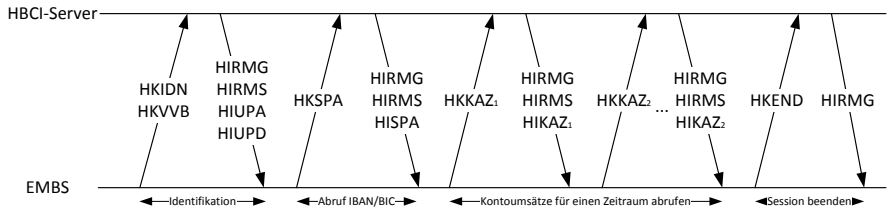

Abbildung 6.3: Übersicht über die Abfolge der Nachrichten bei einem Umsatzabruf

Identifikation (HKIDN) und Verarbeitsungsvorbereitung (HKVVB)

Als erstes sendet der HBCI-Client eine HKIDN + HKVVB Nachricht zur Identifikation und Verarbeitungsvorbereitung. Diese Nachricht ist in Abbildung 6.4 abgebildet.

HNHBK	HNSHK	HKIDN	HKVVB	HNSHA	HNHBS

Abbildung 6.4: Aufbau einer HKIDN + HKVVB Nachricht

HNHBK (Nachrichtenkopf): Jedes Segment beginnt mit einem `Segmentkopf`, welcher in Abbildung 6.5 genauer abgebildet ist.

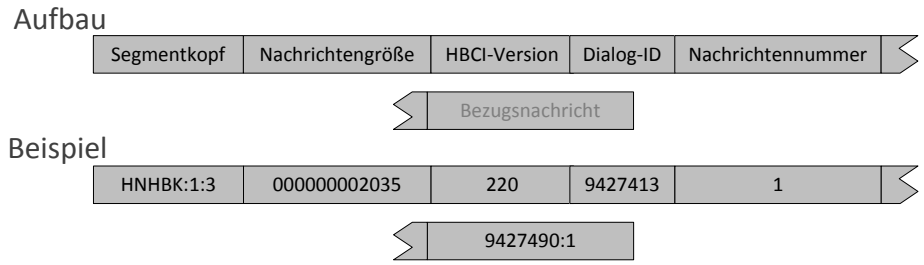

Abbildung 6.5: Aufbau eines HNHBK-Segments. Optionale Felder sind ausgegraut.

Dieser enthält eine Datenelementgruppe, welche aus der `Segmentkennung`, der `Segmentnummer` und der `Segmentversion` besteht. Optional darf hier auch auf ein `Bezugssegment` verwiesen werden.

Die `Nachrichtengröße` teilt dem Empfänger die Größe der Nachricht in Byte nach Komprimierung und Verschlüsselung mit und wird mit führenden Nullen auf zwölf Zeichen aufgefüllt. Im Feld `HBCI-Version` wird die Version codiert, sodass beim Parsen der Nachricht die genaue Spezifikation angewendet werden kann. Um die Nachricht einem spezifischen Dialog zuordnen zu können, enthält die Nachricht zusätzlich eine `Dialog-ID`. Damit der HBCI-Server der Bank weiß, dass ein neuer Dialog initiiert wurde, ist die Dialog-ID in der ersten Nachricht 0. In der Antwort wird nun eine, zu diesem Zeitpunkt und bei dieser Bank eindeutige, Dialog-ID vergeben. Diese Dialog-ID muss in allen darauf folgenden Nachrichten verwendet werden, um auf den richtigen Dialog zu verweisen. Hierdurch kann Bezug auf frühere Nachrichten genommen werden. Damit auch die Reihenfolge der Nachrichten des Dialoges eindeutig bestimmt sind, wird die `Nachrichtennummer` verwendet. Auch diese muss innerhalb eines Dialoges eindeutig sein. Damit Bezug auf eine vorherige Nachricht genommen werden kann, können in der `Bezugsnachricht` Dialog-ID und Nachrichtennummer der entsprechenden Nachricht hinterlegt werden. Jedoch ist dies nur bei Nachrichten des HBCI-Servers zulässig, der HBCI-Client muss dieses Feld frei lassen.

HNSHK (Signaturkopf): Der Signaturkopf (siehe Abbildung 6.6) beschreibt die Verschlüsselung sowie den Absender der Nachricht genauer.

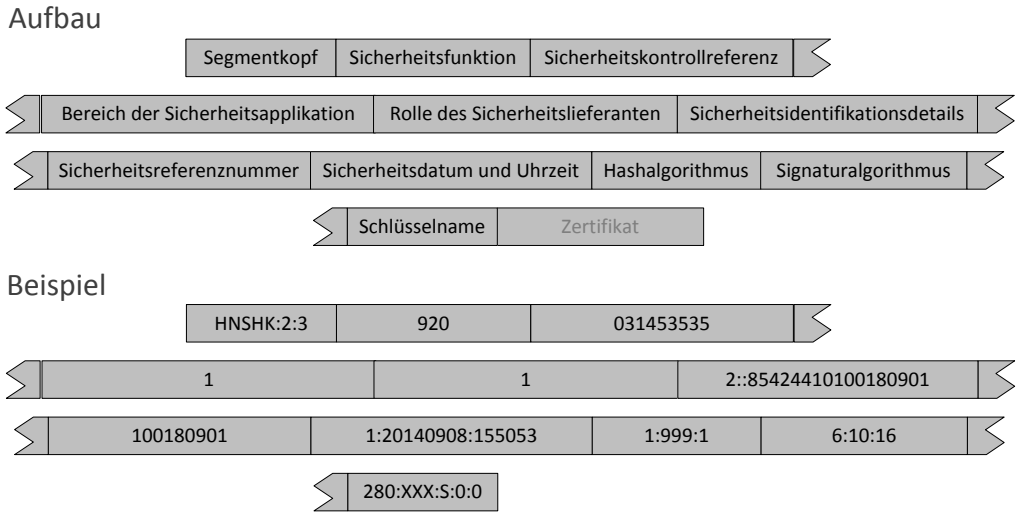

Abbildung 6.6: Aufbau eines HNSHK-Segments. Optionale Felder sind ausgegraut.

Die `Sicherheitsfunktion` gibt an, welche Funktion für die Sicherheit in dieser Nachricht verwendet wird. Dies kann entweder das symmetrische *DES-DES-Verfahren* (DDV) oder das asymmetrische *RSA-DES-Hybridverfahren* (RDH) sein.

Ein Verweis auf den Signaturabschluss befindet sich in der `Sicherheitskontrollreferenz`. Diese Referenz darf bis zu 14 Zeichen lang sein. Das Feld `Bereich der Sicherheitsapplikation` beschreibt, welche Bereiche der Nachricht verschlüsselt werden. Mögliche Werte hier wären **1** (dabei würde der Signaturkopf und die HBCI-Nutzdaten verschlüsselt) oder **2** (dies würde bedeuteten es werden von Signaturkopf bis Signaturabschluss alle Segmente verschlüsselt). Wer für die Verschlüsselung zuständig ist, wird mit der `Rolle des Sicherheitslieferanten` angegeben. Eine **1** kennzeichnet den Verfasser der Nachricht, eine **3** einen Unterstützer (zum Beispiel bei einer Zweitsignatur) und eine **4** wäre ein reiner Übermittler, welcher für den Inhalt keine Verantwortung trägt. Die **2** ist nicht vergeben. Durch das Einbetten von signierten Nach-

richten in weitere Signaturköpfe und Signaturabschlüsse ist das mehrfache Signieren von Nachrichten möglich. In den `Sicherheitsidentifikationsdetails`, einer Datenelementgruppe, werden alle Parteien, welche im Sicherheitsprozess involviert sind, explizit aufgeführt. Diese Datenelementgruppe besteht aus `Bezeichnern für die Sicherheitspartei` (hier steht eine **1** für eine Nachricht vom Kunden an die Bank, eine **2** für die Gegenrichtung) und entweder aus der `Identifikation eines Schlüssels` (bei DDV als Verschlüsselungsfunktion) oder aus der `Identifikation einer Partei` (dies geschieht beim RDH-Verfahren über die Kundensystem-ID, welche im nächsten Segment erklärt wird). Damit eine Transaktion nicht zweimal ausgeführt wird, gibt es die `Sicherheitsreferenznummer`. Dies wird dadurch erreicht, dass in jedem Dialog die Sicherheitsreferenznummer nur einmalig auftreten darf. `Sicherheitsdatum und Uhrzeit` beinhaltet den lokalen Zeitstempel des Systems, welches die Nachricht signiert hat. `Hashalgorithmus` und `Signaturalgorithmus` definieren die jeweils verwendeten Algorithmen. Der `Schlüsselname` identifiziert den verwendeten Schlüssel eindeutig. Optional kann im Feld `Zertifikat` ein verwendetes Zertifikat angegeben werden, welches zur Authentifizierung des Benutzers beiträgt.

HKIDN (Identifikation): Im Identifikationssegment wird sowohl der Absender geprüft, als auch wichtige Umgebungsinformationen übermittelt. Der Aufbau ist in Abbildung 6.7 dargestellt.

Abbildung 6.7: Aufbau eines HKIDN-Segments

Die `Kreditinstitutskennung` identifiziert dabei die Bank eindeutig und besteht aus einem Länderkennzeichen sowie der Bankleitzahl. Ebenfalls eindeutig ist die `Kunden-ID`. Die `Kundensystem-ID` ist eindeutig für die verwendete Software auf der Hardware des Kunden. Das bedeutet, wenn ein Kunde auf einer Workstation mehrere Anwendungen zur Kommunikation mit der Bank hat, so hat jede der Anwendungen

eine eigene Kundensystem-ID. Mit Hilfe des `Kundensystemstatus` gibt der Kunde an, ob die Kundensystem-ID erforderlich ist.

HKVVB (Verarbeitungsvorbereitung): In diesem Segment, welches in Abbildung 6.8 dargestellt ist, werden dem HBCI-Server nähere Informationen über das Kundensystem angegeben.

Abbildung 6.8: Aufbau eines HKVVB-Segments

In der `BPD-Version` werden die Bank-Parameterdaten übertragen. Sollten diese dem HBCI-Client noch nicht bekannt sein, so sendet er hier eine **0**. So kann der HBCI-Server prüfen, ob die Paramater bekannt und aktuell sind. Genau umgekehrt funktioniert dies auch bei der `UPD-Version`, den User-Parameterdaten. In welcher Sprache der Client die Antworten in diesem Dialog empfangen will, gibt er in der `Dialogsprache` an. HBCI 2.2 definiert hier Deutsch (**1**), Englisch (**2**) und Französisch (**3**). In den Feldern `Produktbezeichnung` und `Produktversion` wird der Name sowie die Version des verwendeten Softwareclients angegeben.

HNSHA (Signaturabschluss): Hier wird, wie in Abbildung 6.9 dargestellt, eine Referenz zum Signaturkopf gezogen. Damit sind die Nutzdaten umschlossen und es ist klar, was von dieser Signatur signiert wird.

Abbildung 6.9: Aufbau eines HNSHA-Segments

Die `Sicherheitskontrollreferenz` bezieht sich auf das gleichnamige Feld im Signaturkopf. Die eigentliche Signatur der Nachricht wird als Binärcode im `Validierungsresultat` verwendet.

HNHBS (Nachrichtenabschluss): Jede Nachricht muss mit einem Nachrichtenabschluss (siehe Abbildung 6.10) beendet werden. Neben seinem `Segmentkopf` gibt er zusätzlich die Nachrichtennummer aus dem Nachrichtenkopf an.

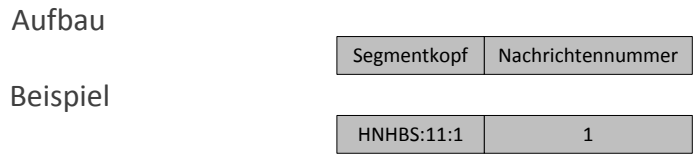

Abbildung 6.10: Aufbau eines HNHBS-Segments

Rückmeldung zur Gesamtnachricht (HIRMG), Rückmeldung zu Segmenten (HIRMS), Userparameter Allgemein (HIUPA) und Kontoinformationen (HIUPD)

In dieser Nachricht (siehe Abbildung 6.11) muss eine Rückmeldung zur Gesamtnachricht (HIRMG) sein, eine Rückmeldung zu jedem Segment (HIRMS), allgemeine Userparameter (HIUPA) und zu jedem Konto die Kontoinformation (HIUPD).

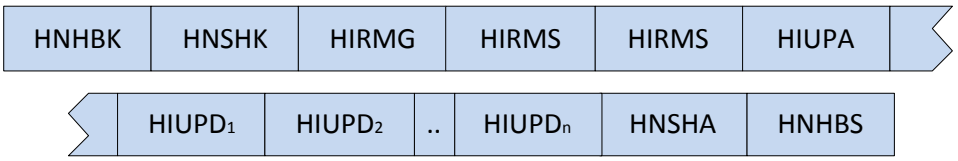

Abbildung 6.11: Aufbau eines *HIRMG + HIRMS + HIUPA + HIUPD* Nachricht

HIRMG (Rückmeldung zur Gesamtnachricht): Das in Abbildung 6.12 dargestellte Segment steht an Beginn jeder Antwort des HBCI-Servers. Es gibt Rückmeldeinformationen zur gesamten Nachricht zurück.

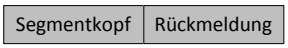

Abbildung 6.12: Aufbau eines HIRMG-Segments

Die `Rückmeldung` ist eine Datenelementgruppe. Sie besteht aus einem vierstelligen `Rückmeldungscode`, welcher den Status angibt. Ist die erste Ziffer eine **0**, so war die Ausführung ein Erfolg. Eine **3** beschreibt eine Warnung, eine **9** einen Fehler. Der Rückmeldungscode darf nur mit einer **0** beginnen, wenn alle weiteren Segmente ebenfalls einen Erfolg vermelden. Das zweite Gruppendatenelement, das `Bezugsdatenelement`, darf nur gesetzt werden wenn es sich um eine Rückmeldung auf ein bestimmtes Segment handelt. Der `Rückmeldungstext` enthält einen frei wählbaren Text des HBCI-Servers zur Bearbeitung der Anfrage oder zur Fehlermeldung. Abschließend können optional `Rückmeldungsparameter` angegeben werden, die beispielsweise Vorschläge zum weiteren Vorgehen enthalten.

HIRMS (Rückmeldung zu Segmenten): Dieses Segment ist eine Rückmeldung zu einem Segment in der Anfrage des Clients. Ist keine Rückmeldung für ein Segment erforderlich, so darf diese entfallen.

Der Aufbau der Rückmeldung zu Segmenten ist identisch zum Aufbau der Rückmeldung zur Gesamtnachricht.

HIUPA (Userparameter Allgemein): Dieses Segment, siehe Abbildung 6.13, gibt die `Allgemeinen Userparameter` des HBCI-Servers an den Client zurück.

Aufbau

Segmentkopf	Benutzerkennung	UPD-Version	UPD-Verwendung

Beispiel

HIUPA:5:2:4	6744068818	0	0

Abbildung 6.13: Aufbau eines HIUPA-Segments

Die `Benutzerkennung` gibt an, auf welchen Benutzer sich die Parameter Beziehen. Über diese Kennung kann zum Beispiel erkannt werden, in welcher Rolle (und somit auch die zugeteilten Rechte) der Benutzer eingeloggt ist. Über die `UPD-Version` erkennt die Gegenseite (in diesem Fall der HBCI-Client), ob sich an den Parametern etwas geändert hat, da diese vom HBCI-Server bei jeder Änderung inkrementiert wird.

Die `UPD-Verwendung` gibt an, wie mit Geschäftsvorfällen, welche in den `Konto-informationen` nicht erwähnt werden, umgegangen wird. Eine **0** gibt an, dass diese Geschäftsvorfälle gesperrt sind, eine **1** gibt an, das der Client diese Information online vom HBCI-Server holen muss, sollte er mehr Informationen zu diesem Geschäftsvorfall benötigen.

HIUPD (Kontoinformationen): In dem in Abbildung 6.14 abgebildeten Segment werden zu jedem Konto des Nutzers die verfügbaren Informationen zurückgegeben.

Segmentkopf	Kontoverbindung	Kunden-ID	Kontowährung	Name1	Name 2

Kontoproduktbezeichnung	Kontolimit	Erlaubte Geschäftsvorfälle

Abbildung 6.14: Aufbau eines HIUPD-Segments. Optionale Felder sind ausgegraut.

Die `Kontoverbindung` besteht aus der `Kontonummer`, wenn vorhanden einem `Unter-kontomerkmal` (zum Beispiel ein Konto mit einer anderen Währung), der `Länder-kennung` sowie dem `Kreditinstitutscode` (was der Bankleitzahl entspricht).

Über die `Kunden-ID` lässt sich die Rolle, welchem dem Nutzer für dieses Konto zugeordnet ist, abrufen. Sie wird dem Kunden von seinem Kreditinstitut zugeordnet und muss innerhalb des Instituts eindeutig sein. Optional kann die `Kontowährung` spezifiziert werden. Sie muss in ihrem ISO-Währungscode, beispielsweise **EUR**, ausgeschrieben werden. In `Name 1` und `Name 2` wird, in bis zu 27 Zeichen pro Feld, der Name des Kontoinhabers angegeben. In der `Kontoproduktbezeichnung` kann, in bis zu 30 Zeichen, eine vom Kreditinstitut frei wählbare Bezeichnung eingetragen sein.

Abbildung 6.15: Aufbau einer Limitgruppe. Optionale Felder sind ausgegraut.

Wenn für das Konto ein Limit besteht, welches angibt, wie viel der Kunde abheben darf, kann dieses im `Kontolimit` angegeben werden (siehe Abbildung 6.15). An erster Stelle steht die `Limitart`. Hier steht ein **E** für ein `Einzelauftragslimit`, ein **T** für ein `Tageslimit`, ein **W** für ein `Wochenlimit`, ein **M** für ein `Monatslimit` und ein **Z** für ein `Zeitlimit`. Wurde ein Zeitlimit eingestellt, so muss im Feld `Limit-Tage` ein dazugehöriger Wert stehen.

Als Letztes kommt eine Liste von bis zu 999 `erlaubten Geschäftsvorfällen`, deren Aufbau in Abbildung 6.16 abgebildet ist.

Abbildung 6.16: Aufbau eines erlaubten Geschäftsvorfalles. Optionale Felder sind ausgegraut.

Nach dem Namen des `Geschäftsvorfall`s muss die `Anzahl der benötigten Signaturen` (gültige Werte sind hier Zahlen zwischen Eins und Vier) angegeben werden. Für die Felder Limitart, Limitbetrag und Limit-Tage gelten die gleichen Regeln

wie beim Kontolimit. Dies darf jedoch nur dann angegeben sein, wenn für das Konto kein allgemeines Limit festgesetzt wurde.

Nachricht mit Kontoumsätzen (HKKAZ)

Aus den aus der HIUPD Nachricht gewonnen Kontoinformationen kann nun der Umsatz abgefragt werden. Der Aufbau einer solchen Nachricht ist in 6.17 abgebildet.

Abbildung 6.17: Aufbau einer HKKAZ-Nachricht

HKKAZ (Kontoumsätze/Zeitraum): HKKAZ ist die Anfrage für die Kontoumsätze in einen bestimmten Zeitraum. Der Aufbau des Segmentes ist in Abbildung 6.18 dargestellt.

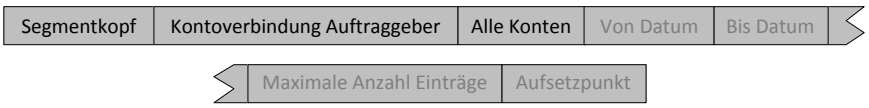

Abbildung 6.18: Aufbau eines HKKAZ-Segments. Optionale Felder sind ausgegraut.

In der `Kontoverbindung` wird das Konto, für welches die Umsätze benötigt werden, angegeben. Sollten die Umsätze für alle Konten des Kunden gewünscht sein, so muss die `Alle Konten` Option explizit gewählt werden. Der Zeitraum für den die Umsätze abgerufen werden lässt sich durch die Felder `Von Datum` sowie `Bis Datum` wählen. Mit Hilfe des Feldes `Maximale Anzahl` lässt sich die Anzahl der Einträge, welche zurückgegeben werden sollen, begrenzen. Muss die Antwort des Kreditinstitutes in mehreren Segmenten übertragen werden, so kann mittels des `Aufsetzpunktes` eine vorangegangene Anfrage wiederaufgenommen werden.

Rückmeldung zur Gesamtnachricht, den Segmenten und Kontoumsätzen (HIKAZ)

Stellt ein Client eine Anfrage zu aktuellen Umsätzen, antwortet der Server mit der in Abbildung 6.19 dargestellten Nachricht.

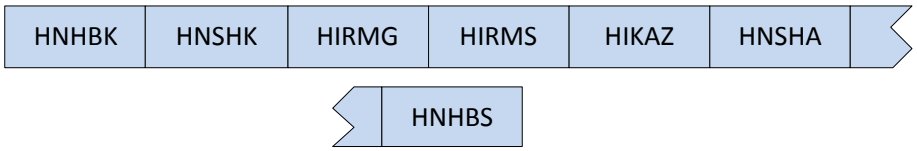

Abbildung 6.19: Aufbau einer HIRMG + HIRMS + HIKAZ Nachricht

HIKAZ (Rückmeldung Kontoumsätze/Zeitraum): Als Antwortnacht auf eine HKKAZ Anfrage werden die angeforderten Umsätze zurückgegeben. Dies geschieht per HIKAZ-Segmenten, welche in Abbildung 6.20 dargestellt sind.

Abbildung 6.20: Aufbau eines HIKAZ-Segments. Optionale Felder sind ausgegraut.

Die `Gebuchten Umsätze` sind abgeschlossene Buchungen (aus dem vom Client angeforderten Zeitraum). Sie werden als Binärdaten im S.W.I.F.T. MT 940 Format [Soc] zurückgegeben. Ein Beispiel für einen solchen Umsatz wird in Listing 6.1 vorgestellt.

Listing 6.1: MT 940 codierter Umsatz

```
1 :20:STARTUMSE
2 :25:63050000/1234567890
3 :28C:00000/001
4 :60F:C140902EUR450,00
```

```
:61:1409050905DR50,00N027NONREF
:86:096?00SEPA UEBERTRAG SOLL?109310?20SVWZ+DATUM 05.09.2014,?21
13.17 UHR, 1.TAN 852685?30SOLADES1ULM?31DE71630500000000XXXXXXX?3
2FABIAN MAIER?34997
:62F:C140905EUR400,00
```

Die Datenfelder sind dabei durch Doppelpunkte voneinander getrennt.

In Zeile 1 ist mit der **20** das Feld als `Referenz` ausgezeichnet. Dies ist eine vom HBCI-Client vergebene ID für den Geschäftsvorfall. In der nächsten Zeile folgt mit dem Wert **25** die `Kontobezeichnung`, welche aus Bankleitzahl und Kontonummer besteht. Der Typ **28C** in Zeile 3 bezeichnet die `Auszugsnummer`, welche die Kontoauszüge sortiert. Im Feld **60F** wird der `Anfangssaldo` dargestellt. Das führende **C** weist den Saldo als „Habensoll“ aus, ein **D** steht für „Sollsaldo“. Hierauf folgt das Datum des Saldos sowie der aktuelle Betrag.

In Zeile 5 folgt nun der eigentliche `Umsatz`. Nach dem Datum der Buchung steht ebenfalls ein **D** für „Soll“. Das darauf folgende **R** steht für den dritten Buchstabe der Währung (zum Beispiel EU**R**), gefolgt von dem Betrag des Umsatzes. Der nun folgende Buchstabe **N** ist eine Konstante, gefolgt von einer Zahlenfolge welche für den Buchungsschlüssel steht. Der letzte Eintrag der Zeile 5 gibt dem HBCI-Server die Möglichkeit, eine Referenz zu setzen. Dies ist hier durch den Wert **NONREF** nicht geschehen.

Das `Mehrzwecksfeld` erstreckt sich über die Zeilen 6-8. Hier ist der Verwendungszweck sowie eine Referenz zum Empfänger codiert. Analog zum Anfangssaldo wird der Umsatz in Zeile 9 vom `Schlusssaldo` beendet.

Freiwillig ist für die Bank die Rückgabe von noch `nicht gebuchten Umsätzen`. Diese werden im S.W.I.F.T. MT 942 Format zurückgegeben.

Beenden des Dialoges (HKEND)

Durch diese Nachricht wird der Dialog beendet. Der Aufbau ist in Abbildung 6.21 dargestellt.

Abbildung 6.21: Aufbau einer HKEND Nachricht

HKEND (Dialogende): Durch das in Abbildung 6.22 dargestellte Segment wird der laufende Dialog beendet.

Abbildung 6.22: Aufbau eines HKEND-Segments

Durch die vom HBCI-Client versendete Nachricht kann der in der `Dialog-ID` angegebene Dialog beendet werden. Als Antwort kommt von dem HBCI-Server eine abschließende HIRMG-Nachricht. Nun kann die Datenverbindung geschlossen werden, da der Dialog beendet ist.

6.1.3 HBCI Bibliothek

Wie im vorangegangenen Kapitel gezeigt ist der Aufbau der Nachrichten sowie der Dialogablauf komplex und daher fehleranfällig.

Zur besseren Verwendung des HBCI-Protokolls sollte eine HBCI-Bibliothek verwendet werden, welche den Umgang mit dem Protokoll erleichtert.

Da HBCI nur in Deutschland Verwendung findet, ist die Auswahl an passenden Bibliotheken stark beschränkt. Dennoch finden sich entsprechende Vertreter, wie beispielsweise *AqBanking* [Pre] oder *HBCI4Java* [Pal].

Aqbanking wird unter der „modifizierten BSD Lizenz“ veröffentlicht. Zur Verwendung wird entweder über einen Kommandozeilenbefehl die Software gestartet, welche die Resultate der Anfrage zurückliefert. Optional lässt sich die Anwendung direkt in C++ einbinden und so als Bibliothek verwenden.

HBCI4Java ist eine Java Bibliothek, welche sich direkt in eine Java Applikation einbinden oder auch über die Kommandozeile aufrufen lässt. Sie ist unter der GPL Lizenz lizensiert.

Für die Verwendung von HBCI4java spricht eine natürliche Integration in Java, ein großes Manko war jedoch die letzte Veröffentlichung einer neuen Version im Oktober 2009. Gerade mit der Einführung von SEPA war dies nicht tragbar. Aqbanking wird regelmäßig aktualisiert, erfordert jedoch den Umweg über die Kommandozeile oder C++.

Schlussendlich fiel die Wahl auf einen Fork [Wil] von HBCI4Java. An der privat weiterentwickelten Version wurdne Erweiterungen für chipTAN, smsTAN sowie SEPA implementiert.

Die verwendete Bibliothek erleichtert die Verwendung des komplexen HBCI-Protokolls erheblich. So spart sich der Entwickler das manuelle Zusammenbauen der Nachrichten und kann unabhängig von der HBCI-Version der Server des Kreditinstitutes ein Standardisiertes Interface zur Kommunikation mit den HBCI-Servern nutzen. In Listing 6.2 wird auszugsweise demonstriert, wie der in Kapitel 6.1.2 gezeigte Abruf von Umsätzen vonstatten geht.

Listing 6.2: Funktion zum Abrufen der Umsätze eines Benutzers

```
1     job = handler.newJob("KUmsAll");
2
3     job.setParam("my.number", ktoNr);
4     job.setParam("my.blz", blz);
5     job.setParam("startdate",
6         dt1.format(today.minusDays(7).toDate()));
7     job.setParam("enddate", dt1.format(today.toDate()));
8
9     String job-ID = "statementOfAccount;
10
```

```
11      jobs.put(job-ID, job);
12      // put resultmode to job
13      jobs.put(job-ID + "_resultMode", MY_PROPS);
14
15      GVRKUms res = (GVRKUms) execute(handler, jobs);
16
17      List<UmsLine> umsaetze = res.getFlatData();
18      List<UmsLine> unbookedUmsaetze = res.getFlatDataUnbooked();
```

In Zeile 1 wird ein neuer HBCI-Job vom Typ **KUmsAll** erstellt. `KUmsAll` bedeutet, dass alle Umsätze für einen bestimmten Zeitraum abgerufen werden sollen.

Der Job wird in den Zeilen 3-7 mit der Kontonummer und der Bankleitzahl des entsprechenden Kontos sowie einem Datumsbereich (exemplarisch hier 7 Tage) initialisiert. In der Zeile 9 wird zur Identifikation eine Job-ID vergeben. In den Zeilen 11 bis 13 wird der Auftrag sowie der `ResultMode` zu einer Liste von auszuführenden Aufträgen hinzugefügt. Der ResultMode dient der Bibliothek als Hinweis an sich selbst, in welcher Form die Ausgabe gewünscht ist.

Mit dem Aufruf der execute()-Methode in Zeile 15 wird der Auftrag an den entsprechenden Server geschickt und dort ausgeführt. Das Ergebnis wird in den Zeilen 17 und 18 bearbeitet, indem auf gebuchte und ungebuchte Umsätze zugegriffen wird.

Als Vorbereitung für diesen Auftrag muss der Kunde einmalig seine Bankdaten eingeben. Hierdurch kann die Verschlüsselung und Kommunikation mit dem HBCI-Server stattfinden.

6.2 EMBS-Prozess

Zur Koordination der verschiedenen Aktivitäten sowie zur leichten Anpassbarkeit bei Änderungen wurde ein Prozessmodell erstellt. Anhand der Umsatzabfrage wird in diesem Kapitel die Zusammenarbeit zwischen dem Prozessmanagement-System und EMBS dargestellt.

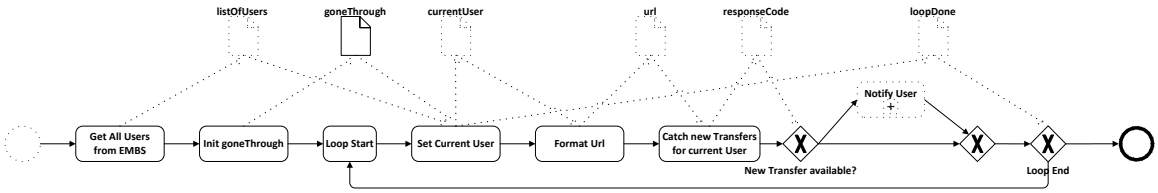

Abbildung 6.23: Überblick über den Prozess

In regelmäßigen Intervallen ruft das EMBS über das Webservice Interface des verwendeten Prozessmanagement-Systems den modellierten Prozess auf. Durch die hohe Sensibilität der zu verarbeitenden Daten werden dem Prozessmanagement-System jedoch nur Stammdaten des Anwenders (wie User-ID, Emailadresse) sowie ein Flag ob neue Umsätze vorhanden sind, zur Verfügung gestellt. Dies minimiert einerseits den Austausch sensibler Daten über das Netzwerk und stellt für einen potentiellen Angreifer auf Seite des Prozessmanagement-System bei einem Serverzugriff nur ein Minimalset an interessanten Daten zur Verfügung. Andererseits müssen aber aufgrund dieses Minimalsets an Daten alle Entscheidungen getroffen werden können. Hier gilt es, genau auszuwählen, welche Daten wirklich nötig sind und übermittelt werden müssen.

In der ersten Aktivität, *Get all Users from EMBS*, werden alle Benutzer des EMBS abgefragt. Dies geschieht über eine REST-Schnittstelle, deren Funktionsweise in Kapitel 6.2.1 genauer betrachtet wird. Das Prozessmanagement-System bekommt eine JSON formatierte Antwort, in welcher die Benutzer aufgeführt werden. Anschließend wird in der Aktivität *Init goneThrough* die Variable *goneThrough* initialisiert. Dies geschieht mit Hilfe eines BeanShell-Skriptes, welches in die Variable eine leere Liste speichert. Mit dieser Liste kann nun die Liste der Benutzer in der Aktivität *listOfUsers* durchgegangen werden.

In einer Schleife, welche alle Nutzer durchläuft, werden nun die eigentlichen Nutzerdaten verarbeitet. In der Aktivität *Format Url* wird die Adresse der REST-Schnittstelle des EMBS zum Abrufen neuer Umsätze definiert. Daraufhin werden durch die Aktivität *Catch New Transfers for current User* sämtliche Transaktionsdaten des aktuellen Benutzers abgerufen. Erneut wird hier die REST-Schnittstelle des Prozessmanagement-Systems

verwendet. Über den HTTP-Statuscode, welcher vom EMBS zurückgeliefert wird, kann entschieden werden, ob es neue Überweisungen gab (dies wird durch den HTTP-Statuscode `201 - CREATED` angegeben) oder ob die Datenbank des EMBS schon auf dem neuesten Stand war (hier wird der HTTP-Statuscode `200 - OK` verwendet).

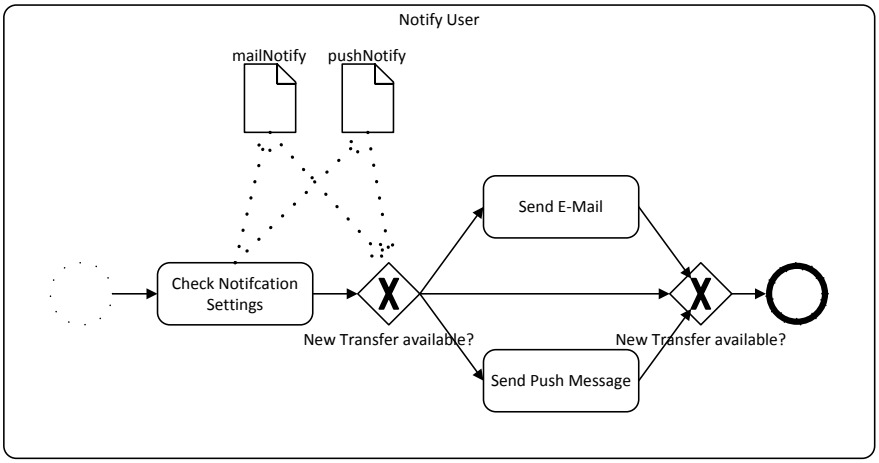

Abbildung 6.24: Subprozess zur Benachrichtigung des Benutzers

Falls neue Transaktionen dieses Benutzers vorhanden sind, wird der Subprozess (siehe Abbildung 6.24) gestartet, welcher den Benutzer über neue Kontobewegungen informiert. Dazu wird über die REST-Schnittstelle die vom Benutzer gewünschte Benachrichtigungsart abgerufen. Dies könnte das Versenden einer E-Mail (diese Aktivität wird in Kapitel 6.2.3 erläutert), das Senden einer Push-Nachricht auf ein Smartphone oder auch der Wunsch keine Benachrichtigung zu erhalten, sein. Je nach gespeicherter Benachrichtigungsart wird nun die passende Aktion ausgeführt.

Die Schleife wird nun für jeden Benutzer durchgeführt, sodass sämtliche Transaktionen verarbeitet werden. Wie beschrieben kommuniziert das BPMS und das EMBS ausschließlich über leichtgewichtige REST Aufrufe. Im folgenden Kapitel wird diese Aktivität näher beschrieben.

6.2.1 Die REST-Aktivität in AristaFlow

Um mit AristaFlow auf die vom EMBS bereitgestellte Funktionalität zugreifen zu können, muss eine Aktivität implementiert werden, welche es erlaubt, REST-Aufrufe abzusetzen. Diese Aktivität soll die Möglichkeit bieten, verschiedene HTTP-Anfragen wie *GET, PUT, POST* und *DELETE* zu stellen. Jede dieser Anfragen muss eine URL zugewiesen werden, unter der diese Aktion durchgeführt werden kann. Weiter müssen im Header der Nachricht Variablen gesetzt sowie bei PUT und POST eine Nachricht verschickt werden können. Als Antwort soll dabei der entsprechende HTTP-Code, die vom Server übergebenen Header sowie der Nachrichtenrumpf mit Nutzdaten empfangen werden können.

Um das Manipulieren der Header zu vereinfachen, erwartet die REST-Schnittstelle diese als valides JSON Objekt. Auch das Analysieren der Header Daten beim Empfangen der Antwortnachricht wird vereinfacht, indem diese als JSON zurückgegeben werden.

Der Nachrichtenrumpf wird immer in einfachem Text gesetzt, um so dem Benutzer der Schnittstelle größtmögliche Flexibilität zu erlauben.

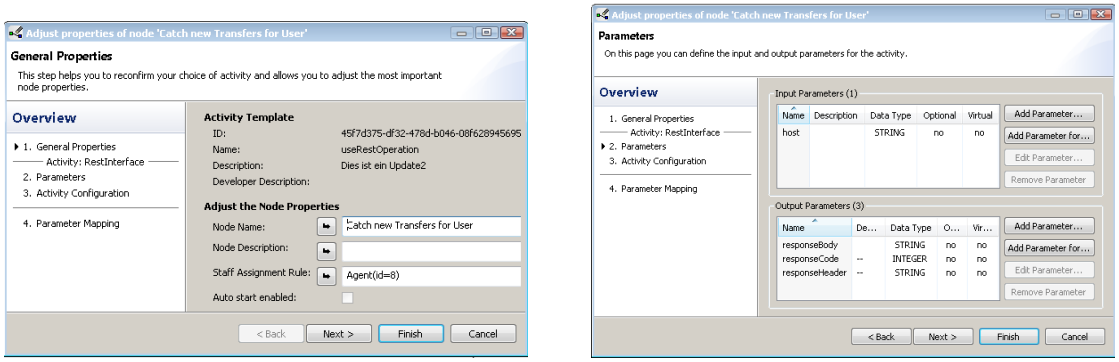

Abbildung 6.25: Konfigurieren des Prozess-Schrittes

Die Benutzung der bereitgestellten REST Aktivität gestaltet sich denkbar einfach. Nachdem der Benutzer die Aktivität auf einen entsprechenden Knoten im Prozess Modell gezogen hat, startet sich ein Dialog zur Konfiguration der Schnittstelle (siehe Abbildung 6.25). Hier können grundlegende Einstellungen für den auszuführenden Prozess-Schritt wie Name, Beschreibung sowie wer zur Ausführung berechtigt ist, definiert werden.

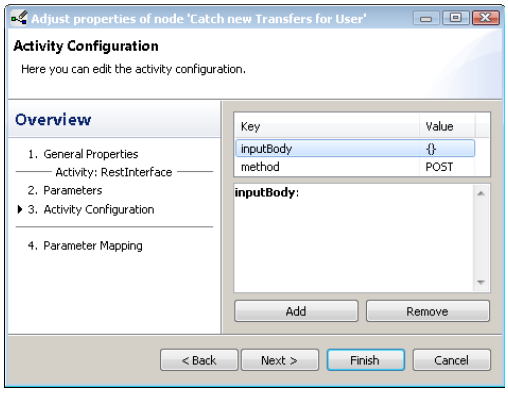

Abbildung 6.26: Konfigurieren des Prozess-Schrittes

Weiter müssen die Ein- sowie Ausgabe-Parameter vom Modellierer definiert werden. `responseCode`, `responseHeader` sowie `responseBody` sind bereits als Standard-parameter für die Ausgabe definiert, gültige Eingabe-Parameter sind `host`, `inputBody`, `inputHeader` sowie `method`.

Der Benutzer kann sich nun entscheiden, welche er als Parameter (siehe Abbildung 6.25) und welche er als Konfiguration (siehe Abbildung 6.26) übergibt. Parameter werden dabei über Variablen innerhalb der Prozessinstanz befüllt, können also dynamisch zur Laufzeit festgelegt werden. Die Konfigurationsdaten bestehen aus fest zugewiesenen Werten und sind für alle Instanzen dieses Prozess-Modells gleich. Es bleibt der Entscheidung des Modellierers überlassen, welche Eingaben er als Parameter übergibt und welche mit der Konfiguration übergeben werden. Die Variablen `host` und `method` müssen unbedingt belegt werden, da ein REST-Aufruf diese benötigt. `inputHeader` und `inputBody` hingegen sind optionale Angaben. Fehlt einer der Pflichtparameter, so erscheint eine Fehlermeldung.

6.2.2 BeanShell-Skript Aktivität

Um die von der REST-Aktivität zurückgelieferten Informationen verarbeiten zu können, bietet AristaFlow bereits die Möglichkeit BeanShell-Skripte zu nutzen. BeanShell [Nie]

ist dabei eine Skriptsprache für Java. Sie erlaubt es, Skripte in Java ähnlicher Syntax zu schreiben und auszuführen.

Anhand des Knotens *Format Url* lässt sich die Funktionsweise von BeanShell-Skripten verstehen.

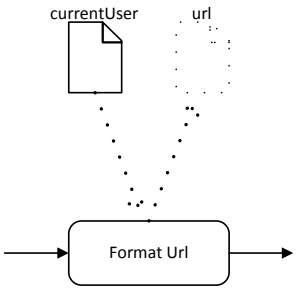

Abbildung 6.27: Datenversorgung des Knoten mit der entsprechenden BeanShell Aktivität

Die Aktivität bekommt als Eingabe den aktuellen Benutzer (`currentUser`) und gibt als Ausgabe die `url` zum Abrufen des Umsatzes für diesen speziellen Benutzer, zurück. Das BeanShell-Skript dieser Aktivität ist in Listing 6.3 abgebildet. Auf die Ein- und Ausgabe-Parameter kann dabei zugegriffen werden, als ob sie unmittelbar davor im Quelltext definiert worden wären.

Listing 6.3: BeanShell-Skript der *Format Url Aktivität*

```
1 url = "http://embs.stoerfaktor.net:6543/" +
2     currentUser + "/umsatz";
```

6.2.3 Aktivität zur Benachrichtigung des Benutzers

AristaFlow bietet bereits eine vorgefertigte Aktivität zum Versenden von E-Mails. Der entsprechende Konfigurationsdialog ist in Abbildung 6.28 dargestellt. In diesem Fall sind die Parameter hart konfiguriert, über Parameter und Platzhalter bei den Feldern lässt sich dies jedoch auch dynamisch umsetzen.

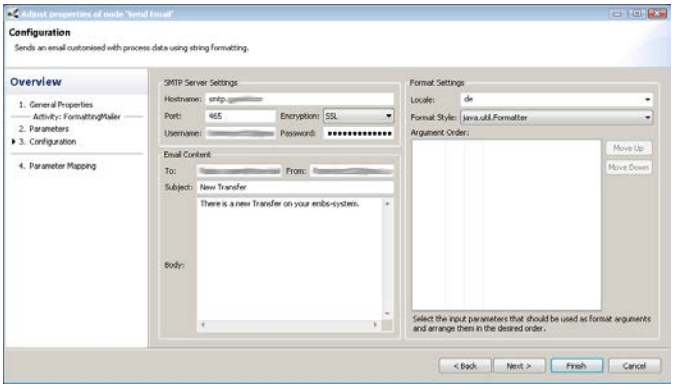

Abbildung 6.28: Konfiguration der Email-Aktivität

Eine andere Möglichkeit der Benachrichtigung wäre das Senden einer Push Nachricht auf das Smartphone des Benutzers. Hierfür existieren unterschiedliche Lösungen der Plattform-Hersteller. Für Android bietet Google seinen *Google Cloud Messaging (GCM)* [Goo] Service an. Für iOs bietet Apple *Push Notifications* [App] an, Microsoft für Windows Phone seinen gleichnamigen Dienst *Push Notifications* [Mic]. Diese Benachrichtigungsdienste bieten eine zuverlässige Methode der Nachrichtenübermittlung an mobile Endgeräte ohne den Stromverbrauch durch aktives Polling der Geräte zu erhöhen.

Die Funktionsweise der Lösungen der unterschiedlichen Anbieter ist dabei sehr ähnlich. Aus diesem Grund beschränkt sich die nähere Beschreibung hier auf den von Google angebotenen Service, dessen grundlegende Funktionsweise in Abbildung 6.29 dargestellt ist.

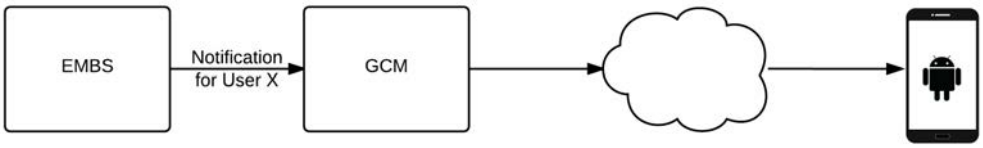

Abbildung 6.29: Übersicht über die Funktionsweise des Google Cloud Messaging System

Damit der Service genutzt werden kann, muss der Entwickler einen Entwickleraccount bei Google erworben haben. Dieser kostet einmalig 25 USD. Der Benutzer der Anwendung, welche GCM verwendet, muss auf seinem Smartphone mit seinem kostenlosen Google Konto angemeldet sein.

Die Kommunikation erfolgt über HTTP oder das *Extensible Messaging and Presence Protocol (XMPP)*. Bei HTTP muss der Nachrichtenkörper entweder im Form Encoding oder als JSON verschickt werden. Bei XMMP wird JSON in eine XML eingebettet.
GCM erlaubt bei XMPP sowohl das Senden von Nachrichten von einem Server an das Mobile Endgerät, als auch die das Senden von einem Mobilen Endgerät über GCM an den Server. Bei HTTP ist nur das Senden von Nachrichten des Servers an das Mobile Endgerät möglich. Bei XMMP werden die Nachrichten Asynchron verschickt, HTTP greift hier auf synchrone Verbindungen zu. Je nach Anforderungen der Anwendung kann aus diesen Informationen eine Entscheidung für eines der Systeme getroffen werden.

Zur Identifikation der Teilnehmer gibt es mehrere IDs. Über die *Application ID* kann sichergestellt werden, das nur die Nutzer einer Anwendung auch Benachrichtigungen über den GCM Service bekommt. Diese ID besteht aus der Bezeichnung der Anwendung, welche im Manifest angegeben wird.
Die *Registration ID* bezieht sich auf eine spezifische Anwendung auf einem spezifischen Smartphone. Nach der Installation der Anwendung auf dem Smartphone wird sie von GCM vergeben. Diese muss die Anwendung nun an den Benachrichtigungsserver des Entwicklers übermitteln. Dies bedeutet das, sollte der Benutzer der Anwendung mehrere mobile Endgeräte besitzen, diese jeweils eine eigene *Registration ID* besitzen.
Damit nicht jeder Server Nachrichten an GCM schicken kann, werden diese über eine *Sender ID* autorisiert. Diese dient dazu die Server zu identifizieren, welche Nachrichten über GCM schicken dürfen. Gegenüber GCM können sie sich mit einem *Sender Auth Token* identifizieren, welche bei jeder Nachricht an einen GCM Server im Header vorhanden sein muss.

Um die strikte Vermeidung sensibler Daten im Prozessmanagement-System fortzuführen, müsste hier ebenfalls ein Aufruf an das EBMS erfolgen. Dieses könnte so mit GCM oder einem System anderer Hersteller kommunizieren.

6.3 Zusammenfassung der Implementierung

In Kapitel 6.1 wurde der grundlegende Aufbau einer HBCI-Nachricht diskutiert, darüber hinaus wurde der Ablauf eines HBCI-Dialoges inklusive der entsprechenden Nachrichten vorgestellt. Abschließend wurde mit *HBCI4JAVA-Schnittstelle* eine Bibliothek vorgestellt, welche die Kommunikation mit HBCI-Servern vereinfacht. In Kapitel 6.2 wurde die Nutzung von AristaFlow als Prozessmanagement-System beleuchtet. Dafür wurde die im Kontext dieser Arbeit entworfene REST-Aktivität und deren Funktionalität vorgestellt sowie das Nutzen von BeanShell-Skripten zur Verarbeitung der Antworten. Weiter wurde das Versenden von Emails durch AristaFlow vorgestellt. Mit GCM, dem Google Cloud Messaging Service, wurde eine mögliche Implementierung vorgestellt, um Push Nachrichten an mobile Endgeräte zu versenden.

Zu einer abschließenden Einschätzung der Arbeit folgt nun Kapitel 7 mit einer Zusammenfassung der Arbeit und einem Ausblick über weitere offene Punkte und interessante Erweiterungen.

7

Fazit

In dieser Arbeit wurde ein Konzept entwickelt, um die Kommunikation zwischen Anwendungen, welche auf die Daten von Kreditinstituten und Banken zugreifen, zu vereinfachen. Darüber hinaus konnte der praktische Einsatz eines Prozessmanagement-Systems in dieser Domäne gezeigt werden.
Anhand dieses Konzeptes wurde eine Architektur entworfen und eine Anwendung umgesetzt.

Hierfür wurden die Grundlagen von HBCI, einem Standard zur Kommunikation von Kunden mit Kreditinstituten, Prozessmanagement und Web-Services erläutert. Nach einem Blick auf verschiedene andere Banking Standards sowie existierenden Online-Banking Lösungen, konnten funktionale, nichtfunktionale und technische Anforderungen abgeleitet und festgehalten werden.
Dies half dabei, die Architektur der Gesamtanwendung, die hierfür nötigen Komponenten

sowie das Zusammenspiel dieser einzelnen Komponenten zu entwerfen.
Nach den Phasen der Vorbereitung konnte die Implementierung durchgeführt werden. Um eine Weiterentwicklung zu erleichtern, wurde auf eine klare Abgrenzung der in der Architektur erdachten Komponenten geachtet. Neben der Logik in der Kernanwendung wurde ein ORM als Schnittstelle zur persistierenden Schicht verwendet. Die Kommunikation mit den Kreditinstituten wurde in eine HBCI-Schnittstelle ausgelagert, sodass durch einen Austausch dieser Komponente die Implementierung eines anderen Banking-Protokolls leicht umzusetzen wäre. Auch die Kommunikation mit dem Prozessmangement-System erfolgt über eine eigene Schnittstelle. Um eine Vielzahl an Endgeräten unterstützen zu können, wurde neben einer GUI eine REST-Schnittstelle implementiert, welche die Entwicklung von Anwendungen auf anderen Systemen und Geräten unterstützt.

In dieser Arbeit wurden einige interessante Aspekte der Implementierung, wie die HBCI-Schnittstelle sowie den EMBS-Prozess näher erläutert. Bei der HBCI-Schnittstelle spielte hier der grundlegende Aufbau einer Nachricht eine wichtige Rolle. Danach konnte ein HBCI-Dialog näher betrachtet werden, um am Schluss die Auswahl sowie die Verwendung der HBCI-Bibliothek erläutern zu können. Bei dem EMBS-Prozess konnte nach einem allgemeinen Überblick nun die eigens für diese Arbeit entwickelte REST-Aktivität erklärt werden. Auch auf der Aktivität zur Benachrichtigung des Benutzers, sowie verschiedene möglicher Implementierungen dieser, wurde ein Fokus gelegt.

7.1 Ausblick

Die in Kapitel 4 definierten Anforderungen konnten während der Konzeption und Implementierung umgesetzt werden. Dennoch bleiben einige Aspekte offen, an denen zusätzliche Forschungsfragestellungen abgeleitet werden können.
Das Thema Sicherheit wurde in dieser prototypischen Implementierung komplett außen vor gelassen und müsste für ein Produktivsystem nachgeholt werden. Hier sind die Authentifizierung und Authorisierung wichtige Aspekte, da unberechtigter Zugriff auf die Funktionen einer Bankinganwendung fatale Auswirkungen haben können. Ebenfalls

müsste eine zuverlässige Verschlüsselung für die Kommunikation zwischen der Anwendung und dem Kreditinstitut implementiert werden, um die sensiblen Informationen vor den Zugriffen Dritter zu schützen.
Daneben muss die Möglichkeit Überweisungen und Überträge zu tätigen implementiert werden.
Die in dieser Arbeit angesprochene Möglichkeiten der Implementierung auf Smartphones wäre ebenfalls ein interessanter Punkt, der noch umgesetzt werden kann.

Durch die Fülle an vorliegenden Daten wäre der Einsatz von Business Intelligence zur Auswertung der Transaktionen ebenfalls interessant. Durch die Fülle an vorliegenden Daten könnte dies für den Benutzer einige interessante Vorteile bieten, wie zum Beispiel Statistiken wie sich die Aufteilung des Geldes auf die Töpfe in den letzten Monaten entwickelt hat. Hieraus könnten nun auch Schlüsse auf zukünftige Entwicklung der Aufteilung gezogen werden.

Literaturverzeichnis

[App] APPLE INC: *Push Notifications for Developers*. https://developer.apple.com/notifications/, . – zuletzt besucht am 20. Oktober 2014

[buh] BUHL: *WISO Mein Geld 2014 Professional*. https://www.buhl.de/produkte/alle/wiso-meingeld-professional/product.html, . – zuletzt besucht am 20. Oktober 2014

[Bun] BUNDESVERBAND DEUTSCHER BANKEN E.V., DEUTSCHER SPARKASSEN- UND GIROVERBAND E.V., BUNDESVERBAND DER DEUTSCHEN VOLKSBANKEN UND RAIFFEISENBANKEN E.V., BUNDESVERBAND ÖFFENTLICHER BANKEN DEUTSCHLANDS E.V.: *HBCI Homebanking-Computer-Interface - Schnittstellenspezifikation*. http://www.hbci-zka.de/dokumente/spezifikation_deutsch/Gesamtdok_HBCI22.pdf,

[Cen] CENTRALWAY: *Numbr*. https://www.centralway.com/de/, . – zuletzt besucht am 20. Oktober 2014

[Die] DIE DEUTSCHE KREDITWIRTSCHAFT: *Neue deutsch-französische Kooperation im Zahlungsverkehr - weiterer Schritt auf dem Weg zum einheitlichen Zahlungsverkehrsraum (SEPA)*. http://www.die-deutsche-kreditwirtschaft.de/uploads/media/081114_ZKA_PI-EBICS-en.pdf, . – zuletzt besucht am 20. Oktober 2014

[DRRM+09] DADAM, Peter ; REICHERT, Manfred ; RINDERLE-MA, Stefanie ; LANZ, Andreas ; PRYSS, Rüdiger ; PREDESCHLY, Michael ; KOLB, Jens ; LY,

Linh T. ; JURISCH, Martin ; KREHER, Ulrich ; GOESER, Kevin: From ADEPT to AristaFlow BPM Suite: A Research Vision has become Reality. In: *Proceedings Business Process Management (BPM'09) Workshops, 1st Int'l. Workshop on Empirical Research in Business Process Management (ER-BPM '09)*, Springer, September 2009 (LNBIP 43), S. 529–531

[Fie00] FIELDING, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, Diss., 2000

[Goo] GOOGLE INC: *Google Cloud Messaging*. `https://developer.android.com/google/gcm/gcm.html`, . – zuletzt besucht am 20. Oktober 2014

[IET] IETF: *RFC 5789 - PATCH Method for HTTP*. `http://tools.ietf.org/html/rfc5789`, . – zuletzt besucht am 20. Oktober 2014

[Mic] MICROSOFT INC: *Push notifications for Windows Phone 8*. `http://msdn.microsoft.com/en-us/library/windows/apps/ff402558(v=vs.105).aspx`, . – zuletzt besucht am 20. Oktober 2014

[Nie] NIEMEYER, Pat: *BeanShell - Lightweight Scripting for Java*. `http://www.beanshell.org/`, . – zuletzt besucht am 20. Oktober 2014

[Pal] PALME, Stefan: *HBCI4Java - Open-Source HBCI-Client- und Server-Bibliothek für Java*. `http://hbci4java.kapott.org/index.html`, . – zuletzt besucht am 20. Oktober 2014

[Pre] PREUSS, Martin: *Aquamaniac Banking Interface*. `http://www.aquamaniac.de/sites/aqbanking/index.php`, . – zuletzt besucht am 20. Oktober 2014

[RAC] RACON SOFTWARE GMBH: *GRZ IT Gruppe - RACON Software GmbH*. `http://www.racon-linz.at/eBusiness/rlbooe_template2/559425961133575325-NA-675168162526050604-NA-25-NA.html`, . – zuletzt besucht am 20. Oktober 2014

[Rei00] REICHERT, Manfred: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. Juli 2000

[Soc] SOCIETY FOR WORLDWIDE INTERBANK FINANCIAL TELECOMMUNICATION: *Standards - MT-940*. https://www2.swift.com/uhbonline/books/public/en_uk/us9m_20140725/index.htm?subpage=ajg.htm, . – zuletzt besucht am 20. Oktober 2014

[Spaa] SPARKASSE: *„Sparkasse - Ihre mobile Filiale“ für iPhone, iPod-touch und iPad im App-Store von iTunes*. https://itunes.apple.com/de/app/id320599923?mt=8, . – zuletzt besucht am 20. Oktober 2014

[Spab] SPARKASSE: *Sparkasse+ - Android-Apps auf Google Play*. https://play.google.com/store/apps/details?id=com.starfinanz.smob.android.sbanking, . – zuletzt besucht am 20. Oktober 2014

[Spac] SPARKASSE: *Sparkasse - Android-Apps auf Google Play*. https://play.google.com/store/apps/details?id=com.starfinanz.smob.android.sfinanzstatus, . – zuletzt besucht am 20. Oktober 2014

[Spad] SPARKASSE: *Sparkasse Ulm (63050000)*. https://www.sparkasse-ulm.de/, . – zuletzt besucht am 20. Oktober 2014

[Sta] STARMONEY: *Die marktführende Online-Banking Software StarMoney*. http://www.starmoney.de/, . – zuletzt besucht am 20. Oktober 2014

[Uni] UNITED NATIONS ECONOMIC COMMISSION FOR EUROPE: *Introducing UN/EDIFACT*. http://www.unece.org/cefact/edifact/welcome.html, . – zuletzt besucht am 20. Oktober 2014

[W3C] W3C: *SOAP Specifications*. http://www.w3.org/TR/soap/, . – zuletzt besucht am 20. Oktober 2014

[Whi] WHITE, James E.: *RFC 707 - High-level framework for network-based resource sharing*. https://tools.ietf.org/html/rfc707, . – zuletzt besucht am 20. Oktober 2014

[Wil] WILLUHN, Olaf: *willuhn/hbci4java · GitHub*. https://github.com/willuhn/hbci4java, . – zuletzt besucht am 20. Oktober 2014

[Zena] ZENTRALER KREDITAUSSCHUSS: *Anlage 1 der Schnittstellenspezifikation für die Datenfernübertragung zwischen Kunde und Kreditinstitut gemäß DFÜ-Abkommen*

[Zenb] ZENTRALER KREDITAUSSCHUSS: *Anlage 2 der Schnittstellenspezifikation für die Datenfernübertragung zwischen Kunde und Kreditinstitut gemäß DFÜ-Abkommen*

[Zenc] ZENTRALER KREDITAUSSCHUSS: *Homepage des Zentraler Kreditausschuss*. `http://www.zentraler-kreditausschuss.de/,` . – zuletzt besucht am 20. Oktober 2014

Name: Fabian Maier Matrikelnummer: 675672

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Fabian Maier