



Duale Hochschule Baden-Württemberg  
Mannheim

## **Bachelorarbeit**

**Entwicklung einer Zwischenschicht für die Nutzung weiterer Anwendungen in  
Verbindung mit der Berechnungskomponente des Liquidity Risk  
Managements**

## **Studiengang Wirtschaftsinformatik**

Vertiefungsrichtung Softwaremethodik

Verfasser:	Fabian Kajzar
Matrikelnummer:	428094
Firma:	SAP AG
Abteilung:	Application Strategic Innovation - HPA
Kurs:	WWI 09 SW B
Studiengangsleiter:	Prof. Dr.-Ing. Jörg Baumgart
Wissenschaftlicher Betreuer:	Prof. Dr. Hans-Henning Pagnia hans-henning.pagnia@dhbw-mannheim.de 0621 4105-1131
Firmenbetreuer:	Jens Mett jens.mett@sap.com 06227 7-61785
Bearbeitungszeitraum:	13. Februar 2012 bis 4. Mai 2012

# Kurzfassung

Verfasser: Fabian Kajzar

Kurs: WWI 09 SW B

Firma: SAP AG

Thema: Entwicklung einer Zwischenschicht für die Nutzung weiterer Anwendungen in Verbindung mit der Berechnungskomponente des Liquidity Risk Managements

# Inhaltsverzeichnis

<b>Verzeichnisse</b>	<b>V</b>
Abkürzungsverzeichnis . . . . .	V
Abbildungsverzeichnis . . . . .	VI
Tabellenverzeichnis . . . . .	VII
Listingverzeichnis . . . . .	VIII
<b>1 Einleitung</b>	<b>1</b>
<b>2 Grundlagen des Liquiditätsrisikomanagements</b>	<b>2</b>
2.1 Einleitung . . . . .	2
2.2 Liquidität . . . . .	2
2.3 Liquiditätsrisiko . . . . .	4
2.4 Liquiditätsrisikomanagement . . . . .	5
2.5 Zusammenfassung . . . . .	8
<b>3 SAP LRM und BusinessObjects Xcelsius</b>	<b>9</b>
3.1 Einleitung . . . . .	9
3.2 SAP LRM . . . . .	9
3.2.1 Funktionen . . . . .	9
3.2.2 Architektur . . . . .	10
3.2.2.1 NGAP . . . . .	10
3.2.2.2 HANA . . . . .	12
3.2.2.3 Oberon . . . . .	13
3.2.3 Berechnungskomponente . . . . .	14
3.3 BusinessObjects Xcelsius . . . . .	17
3.3.1 Überblick und Funktionsumfang . . . . .	17
3.3.2 Bedienungskonzept . . . . .	18
3.3.3 Architektur . . . . .	21
3.3.4 Erweiterungsmöglichkeiten . . . . .	22
3.4 Zusammenfassung . . . . .	24
<b>4 Spezifikation</b>	<b>25</b>
4.1 Einleitung . . . . .	25
4.2 Ausgangssituation . . . . .	25

4.3	Zielsetzung . . . . .	26
4.4	Anforderungen . . . . .	28
4.5	Anwendungsfälle . . . . .	30
4.6	Zusammenfassung . . . . .	32
<b>5</b>	<b>Umsetzungsmöglichkeiten</b>	<b>33</b>
5.1	Einleitung . . . . .	33
5.2	Rahmenbedingungen . . . . .	33
5.3	BusinessObjects Universum . . . . .	33
5.4	Webservice . . . . .	35
5.5	Gegenüberstellung und Auswahl der Umsetzungsmöglichkeiten . . . .	36
5.6	Zusammenfassung . . . . .	38
<b>6</b>	<b>Umsetzung</b>	<b>39</b>
6.1	Einleitung . . . . .	39
6.2	Analyse . . . . .	39
6.2.1	Überblick . . . . .	39
6.2.2	Grundlegende Architektur und Funktionsweise . . . . .	40
6.2.3	Statisches Modell . . . . .	42
6.2.4	Dynamisches Modell . . . . .	43
6.3	Entwurf . . . . .	45
6.3.1	Orchestrierung der Webservices . . . . .	45
6.3.2	Statisches Modell . . . . .	46
6.3.3	Dynamisches Modell . . . . .	48
6.3.4	Zielstruktur in BusinessObjects Xcelsius (Xcelsius) . . . . .	49
6.4	Implementierung . . . . .	51
6.4.1	Flex Sandbox . . . . .	51
6.4.2	Asynchroner Programmablauf . . . . .	53
6.4.3	WebserviceAblauf . . . . .	56
6.4.4	Xcelsius Binding . . . . .	57
6.5	Zusammenfassung . . . . .	59
<b>7</b>	<b>Evaluation</b>	<b>60</b>
7.1	Einleitung . . . . .	60
7.2	Erfüllung der Anforderungen . . . . .	60
7.3	Möglichkeiten . . . . .	60
7.4	Vergleich . . . . .	60
7.5	Performance . . . . .	60
7.6	Zusammenfassung . . . . .	60
<b>8</b>	<b>Zusammenfassung</b>	<b>61</b>

<b>A Anhang</b>	<b>VIII</b>
<b>Glossar</b>	<b>X</b>
<b>Literaturverzeichnis</b>	<b>XIII</b>
<b>Ehrenwörtliche Erklärung</b>	<b>XVII</b>

# Verzeichnisse

## Abkürzungsverzeichnis

CRM	Customer Relationship Management
DBMS	Datenbankmanagementsystem
ERP	Enterprise Resource Planning
HANA	SAP High Performance Analytic Appliance
HTTP	Hypertext Transfer Protocol
LiqV	Liquiditätsverordnung
MaRisk	Mindestanforderungen an das Risikomanagement
NetWeaver	SAP NetWeaver Application Server
NGAP	Next Generation ABAP Plattform
SQL	Structured Query Language
Xcelsius	BusinessObjects Xcelsius
XML	Extensible Markup Language

## Abbildungsverzeichnis

1:	Darstellung eines CashFlows-Eintrags mit dem Oberon-Framework . . .	14
2:	[TODO] WT: Aufbau der Benutzeroberfläche von Xcelsius . . . . .	19
3:	[TODO] WT: Beispiel eines Dashboards in Xcelsius . . . . .	20
4:	[TODO] WT Grundlegende Arbeitsweise mit Xcelsius . . . . .	21
5:	[TODO] WT Architekturüberblick von Xcelsius . . . . .	22
6:	[TODO] Plunginsystem von Xcelsius . . . . .	23
7:	Darstellung der Kennzahlenanalyse im SAP LRM . . . . .	26
8:	Darstellung der Finanzierungsmatrix im SAP LRM . . . . .	27
9:	Umsetzungsmöglichkeit der Erweiterung mittels BusinessObjects Uni- versum . . . . .	34
10:	Umsetzungsmöglichkeit der Erweiterung mittels Webservices . . . . .	35
11:	Überblick der grundlegenden Architektur der Erweiterung . . . . .	41
12:	Klassendiagramm der Analysephase . . . . .	42
13:	Sequenzdiagramm der Analysephase . . . . .	44
14:	Abhängigkeiten der Webservices . . . . .	46
15:	Klassendiagramm der Entwurfsphase . . . . .	47
16:	Sequenzdiagramm der Entwurfsphase . . . . .	49
17:	Interne Objektstruktur vor der Umwandlung durch den XcelsiusAdapter	50
18:	[TODO] [TODO Referenzieren!] . . . . .	52
19:	Umwandlung der XML-Antwort in die interne Objektstruktur . . . . .	57

## Tabellenverzeichnis

1:	Einordnung einiger Liquiditätsvorschriften . . . . .	7
2:	Gegenüberstellung der Vor- und Nachteile der Umsetzungsmöglichkeiten für die Erweiterung . . . . .	37
3:	Ergebnis der Umwandlung durch den XcelsiusAdapter . . . . .	51



## Listingverzeichnis

1:	Aufruf eines Webservice in der Erweiterung . . . . .	54
2:	Beispielhafter Ablauf der Orchestrierung von Webservices . . . . .	55
3:	Deserialisierung des XML-Format in die interne Objektstruktur . . . . .	58
4:	Proxy-Servlet zur Umgehung der Cross-Domin Request Beschränkungen	IX

# 1 Einleitung

# 2 Grundlagen des Liquiditätsrisikomanagements

## 2.1 Einleitung

## 2.2 Liquidität

Der Begriff der Liquidität ist weit verbreitet und im allgemeinen Sprachgebrauch festgesetzt. Allerdings ist eine eindeutige Definition des Begriffs schwierig, da Liquidität sehr vielschichtig ist, mehrere Dimensionen besitzt und die jeweilige Bedeutung von der Perspektive der Betrachtung abhängt.<sup>1</sup> Für diese Arbeit ist vor allem die betriebswirtschaftliche Sicht auf Liquidität entscheidend. Die volkswirtschaftliche Sicht wird daher nicht näher erläutert.<sup>2</sup>

In der betriebswirtschaftlichen Sicht wird zunächst die Liquidität von Objekten von der Liquidität von Subjekten unterschieden. Die Objektliquidität ist die Fähigkeit eines Vermögensgegenstandes in Zahlungsmittel umgewandelt werden zu können.<sup>3</sup> Sie hängt demnach von der Nähe des Objektes zu Geld ab. Zahlungsmittel haben die höchste Objektliquidität, Immobilien eine geringe.<sup>4</sup> Die Liquidität von Subjekten bezeichnet die Fähigkeit eines Subjekts, zum Beispiel einer Bank, alle Zahlungsver-

---

<sup>1</sup> vgl. [Dür11, S.3] und [Bar08, S.13]

<sup>2</sup> vgl. [Die10, S.10]

<sup>3</sup> vgl. [Moc07, S.10]

<sup>4</sup> vgl. [Dür11, S.3]

pflichtungen erfüllen zu können.<sup>5</sup>

Zeitlich kann Liquidität in kurz und langfristig unterschieden werden. Bei der kurzfristigen Liquidität steht der Zahlungsaspekt im Vordergrund, meist nur auf einen Tag bezogen.<sup>6</sup> Es muss zu jeder Zeit sichergestellt werden, dass alle fälligen Zahlungen in der entsprechenden Höhe beglichen werden können. Diese Bedingung ist bei der Steuerung von Banken zu jedem Zeitpunkt streng einzuhalten.<sup>7</sup> Synonym werden auch die Begriffe operative Liquidität sowie dispositive Liquidität verwendet.<sup>8</sup>

Die langfristige Liquidität bezeichnet die Fähigkeit langfristige Refinanzierungsmittel auf der Passiv-Seite der Bilanz aufzunehmen um dadurch die gewünschte Entwicklung auf der Aktiv-Seite der Bilanz ermöglichen zu können. Sie ist also mit den Zielen des Subjektes verknüpft.<sup>9</sup> Für Banken ist dies besonders wichtig, da es einen entscheidenden Wettbewerbsvorteil gegenüber Mitbewerbern darstellt.<sup>10</sup> Zwischen der kurz und langfristigen Liquidität besteht eine beidseitige Wechselwirkung – eine schlechte kurzfristige Liquidität führt zu Problemen bei der langfristigen Liquidität.<sup>11</sup>

Die Folgen von Liquiditätsproblemen können weitreichend sein. Probleme mit sowohl der kurzfristigen als auch der langfristigen Liquidität können zu einem Reputationsverlust führen. Gerade bei Banken hat dies schwere Auswirkungen, da Fremdkapitalgeber das Vertrauen in die Bank verlieren. Dies wiederum hat Auswirkungen auf die Passiv-Seite der Bilanz, viel Fremdkapital wird verloren gehen. Im schlimmsten Fall, wenn die Bank ihren Zahlungsverpflichtung nicht mehr nachkommen kann, muss sie Insolvenz anmelden.<sup>12</sup>

---

<sup>5</sup> vgl. [Dür11, S.3] und [Die10, S.11]

<sup>6</sup> vgl. [Dür11, S.3f]

<sup>7</sup> vgl. [Bar08, S.13] und [Die10, S.12]

<sup>8</sup> vgl. [Bar08, S.13]

<sup>9</sup> vgl. [Dür11, S.4]

<sup>10</sup> vgl. [Bar08, S.13]

<sup>11</sup> vgl. [Bar08, S.15]

<sup>12</sup> vgl. [Dür11, S.4] und [GR10, S.65]

## 2.3 Liquiditätsrisiko

Die Finanzinstitute haben in der Vergangenheit dem Liquiditätsrisiko keine besondere Bedeutung zugewandt. Ob ein Institut das Risiko gesondert behandelt hat oder nicht, konnte frei gewählt werden. Erst im Jahr 2007, als die Grundstückspreise in den USA zusammengebrochen sind und dadurch viele Banken in Liquiditätsschwierigkeiten gekommen sind, rückte die Behandlung des Liquiditätsrisikos in den Fokus - nicht zuletzt durch die Pleite der Lehman Brothers Bank.<sup>13</sup>

Das Liquiditätsrisiko ist das Risiko, gegenwärtige oder zukünftige Zahlungsverpflichtungen entweder nicht, nicht vollständig oder nicht zeitgerecht nachkommen zu können.<sup>14</sup> Grundsätzlich ist das Liquiditätsrisiko bei allen Unternehmen vorhanden. Bei Banken ist es allerdings besonders stark ausgeprägt, da hier sowohl die Ein- als auch die Auszahlungen in hohem Maße von dem Kundenverhalten abhängen.<sup>15</sup> Im weitesten Sinne wird zu dem Liquiditätsrisiko auch die Opportunitätskosten hinzugezogen, die entstehen wenn eine gewinnbringende Transaktion aufgrund fehlender Zahlungsmittel nicht durchgeführt werden kann.<sup>16</sup>

Analog zu der Unterteilung des Liquiditätsbegriffes kann auch das Liquiditätsrisiko weiter unterteilt werden. Zunächst unterscheidet man bei dem bankenbezogenen Liquiditätsrisiko das Liquiditätsspannungsrisiko und das Zahlungsmittelbedarfsrisiko.

Das Liquiditätsanpassungsrisiko beinhaltet grundsätzlich Risiken aufgrund von Zuflüssen und kann in das Refinanzierungsrisiko und das Marktliquiditätsrisiko unterteilt werden.<sup>17</sup> Wenn im Falle eines Engpass nicht genügend Mittel beschafft werden können, oder dies nur unter erhöhten Marktpreisen erreicht werden kann, wird von dem Refinanzierungsrisiko gesprochen. Das Vertrauen der Marktteilnehmer ist hier entscheidend. Beeinflusst werden kann es vor allem durch die Veränderung des Leit-

---

<sup>13</sup>vgl. [Bar08, S.5] und [Rom10, S.37]

<sup>14</sup>vgl. [Hul10, S.467f] , [Zer10, S.166f] und [Dür11, S.6]

<sup>15</sup>vgl. [Alb10, S.90] und [STW08, S.79]

<sup>16</sup>vgl. [STW08, S.79]

<sup>17</sup>vgl. [Dür11, S.7]

zinses der Notenbank.<sup>18</sup> Das Marktliquiditätsrisiko bezieht sich auf die Geldnähe von Aktiva und bezeichnet das Risiko, einen Aktivposten nur zu hohen Transaktionskosten liquidieren zu können. Es ist nur schwer beeinflussbar, da es von dem aktuellen Angebot und der Nachfrage auf dem jeweiligen Markt abhängt.<sup>19</sup>

Das Zahlungsmittelbedarfsrisiko, auch originäres Liquiditätsrisiko genannt, beruht im Gegensatz zu dem Liquiditätsspannungsrisiko auf den Abflüssen von Liquidität. Es wird hauptsächlich das Terminrisiko und das Abrufisiko unterschieden.<sup>20</sup> Das Terminrisiko resultiert aus verspäteten Zahlungseingängen, genauer gesagt aus außerplanmäßigen Prolongationen von Aktivgeschäften über die vereinbarte Kapitalbindungsdauer hinaus.<sup>21</sup> Ein Beispiel ist die Verlängerung eines Kredites, da der Kreditnehmer die Tilgung oder die Zinsen des Kredites nicht bezahlen kann.<sup>22</sup> Das Abrufisiko beruht auf einer unerwarteten Ausnutzung von zugesagten Kreditlinien. Hier findet ein Liquiditätsabfluss in unerwarteter Höhe statt.<sup>23</sup> Der bekannteste und zugleich extremste Fall des Abrufisikos ist eine Bankenpanik<sup>GL</sup>.

## 2.4 Liquiditätsrisikomanagement

Das Liquiditätsrisikomanagement bei Banken hat in den letzten Jahren zunehmend an Bedeutung gewonnen. Durch die Implementierung eines Liquiditätsrisikomanagements soll vor allem die Liquiditätssituation einer Bank jederzeit transparent dargestellt werden können. Durch die Darstellung kann wiederum die Situation überwacht und so die Zahlungsfähigkeit sichergestellt werden. Gleichzeitig können die Liquiditätskosten minimiert werden. Weitere Ziele sind die Erfüllung der Anforderungen von Ratingagenturen sowie rechtliche Anforderungen.<sup>24</sup> Die Steuerung der Liquiditätsrisiken sollten in die Gesamtbanksteuerung eingebunden werden - nur so kann

---

<sup>18</sup>vgl. [Dür11, S.7f]

<sup>19</sup>vgl. [Dür11, S.9]

<sup>20</sup>vgl. [Dür11, S.7f] und [Die10, S.12]

<sup>21</sup>vgl. [Poh08, S.12] und [Zer05, S.51]

<sup>22</sup>vgl. [Dür11, S.10]

<sup>23</sup>vgl. [SLK08, S.513f]

<sup>24</sup>vgl. [SS08, S.256f]

eine gesamtheitliche Sicht gewährleistet werden.<sup>25</sup>

Grundsätzlich existiert zum einen die Ansicht, dass für das Management des Liquiditätsrisikos keine gesetzlichen Regelungen und Vorschriften festgelegt werden müssen. Eine Regulierung findet in der freien Marktwirtschaft automatisch statt, indem Institutionen, die ein schlechtes Liquiditätsrisikomanagement haben, gegen über Institutionen mit einem besseren Liquiditätsrisikomanagement Wettbewerbsnachteile haben. Dass es im Liquiditätsrisikomanagement allerdings doch zahlreiche rechtliche Vorschriften gibt liegt an der besonderen Situation im Bankensektor.<sup>26</sup>

Durch die hohen Verflechtungen und Abhängigkeiten der Banken untereinander besteht ein Systemrisiko.<sup>27</sup> Ein starker Mittelabfluss bei einer einzelnen Bank, der bei dieser Bank zu Liquiditätsproblemen führt und schließlich zur Insolvenz kann dazu führen, dass das Vertrauen der Banken untereinander und damit der Interbankenhandel<sup>GL</sup> zusammenbricht. Dadurch fehlt allen Banken eine Liquiditätsquelle – die Probleme einer einzelnen Bank haben den gesamten Bankensektor erreicht.<sup>28</sup>

Um dieses Systemrisiko zu begrenzen existieren einige gesetzliche Vorgaben. Hier ist als Ursprung der erste Basler Akkord zu nennen. Er wurde 1988 von der Aufsichtsbehörde der G-10 Staaten in Verbindung mit der Basler Bank für internationalen Zahlungsausgleich erlassen. Der erste Basler Akkord wird allgemein hin als Basel I bezeichnet. In Basel I ist festgelegt, dass Banken Eigenkapital in Abhängigkeit zu ihren Risiken hinterlegen müssen.<sup>29</sup> In Deutschland spielen die Mindestanforderungen an das Risikomanagement (MaRisk) und die Liquiditätsverordnung (LiqV) eine wichtige Rolle.

In den MaRisk sind Anforderungen an das Liquiditätsrisikomanagement in Banken festgelegt und damit die Vorgaben von Basel II, dem Nachfolger von Basel I, in deutschem Recht umgesetzt.<sup>30</sup> In der LiqV ist vor allem die Öffnungsklausel wichtig. Durch die Öffnungsklausel ist es Banken möglich, interne Methoden zu verwenden,

---

<sup>25</sup>vgl. [Bar08, S.20f]

<sup>26</sup>vgl. [Pau11a, S.457f]

<sup>27</sup>vgl. [Sch08, S.233f]

<sup>28</sup>vgl. [Hul10, S.265]

<sup>29</sup>vgl. [Hul10, S.291] und [Hue04, S.1f]

<sup>30</sup>vgl. [RM08, S.52f und 58]

um rechtliche Vorgaben abzudecken. Dadurch kann das Liquiditätsrisiko über die rechtlichen Vorgaben hinaus gesteuert werden und die Integration in die Gesamtbanksteuerung wird erleichtert.<sup>31</sup>

Die gesetzlichen Vorschriften können in zwei Dimensionen eingeteilt werden. Die erste Dimension ist die Unterscheidung in qualitative und quantitative Bedingungen. Die qualitativen Bedingungen legen Anforderungen an die Qualität von Strukturen und Prozessen fest, bei den quantitativen Bedingungen geht es um wertmäßige Einhaltung von bestimmten Kennzahlen. Die zweite Dimension unterteilt die Regelungen in nationale und internationale Regelungen. Die genannten Vorschriften sind in Tabelle 1 entsprechend eingeordnet.<sup>32</sup>

	International	National
Qualitativ	-	LiqV
Quantitativ	Basel II	MaRisk

Tabelle 1: Einordnung einiger Liquiditätsvorschriften

Die Maßnahmen, die mit Hilfe des Liquiditätsrisikomanagements getroffen werden sind meist entweder ursachen- oder wirkungsbezogen. Ursachenbezogene Maßnahmen zielen darauf ab, das Risiko bereits vor dem Eintritt zu begrenzen. Wirkungsbezogene Maßnahmen haben das Ziel, die Auswirkungen eines Risikos, nachdem es eingetreten ist, zu vermindern. Ein Beispiel für eine ursachenbezogene Maßnahme bei Banken ist die Risikominderung, in dem die Höhe von möglichen Liquiditätsrisiken im Vorraus über Frühwarnsysteme begrenzt wird. Eine wirkungsbezogene Maßnahme ist der Risikotransfer, bei dem die Folgen über eine Versicherung auf andere abgewälzt werden.<sup>33</sup>

In der aktuellen Entwicklung spielt Basel III eine wichtige Rolle. Durch Basel III soll das Liquiditätsrisikomanagement weiter homogenisiert werden. Ein zentraler Punkt ist die Einführung von zwei Liquiditätskennzahlen,  $LCR^{GL}$  und  $NSFR^{GL}$ , die schrittweise eingeführt werden. Die Liquidity Coverage Ratio (LCR) soll verhindern, dass kurzfristige Probleme auf dem Geldmarkt auf Banken übergreifen und damit

<sup>31</sup>vgl. [RM08, S.53] und [HSG08, S.194]

<sup>32</sup>vgl. [Dür11, S.15]

<sup>33</sup>vgl. [Dür11, S.85ff]



das Systemrisiko eindämmen. Banken müssen ausreichend Vermögenspositionen besitzen, die selbst in einer Stresssituation liquidiert werden können. Damit müssen unerwartete Liquiditätsabflüsse in einem Zeitraum von 30 Tagen abgefangen werden können. Der Liquidity Coverage Ratio (LCR) wird ab 2015 für alle Kreditinstitute binden eingeführt.<sup>34</sup>

Neben dem Liquidity Coverage Ratio (LCR) existiert mit dem Net Stable Funding Ratio (NSFR) eine weitere wichtige Kennzahl im Rahmen von Basel III. Er bezieht sich auf das Verhältnis von mittel und langfristigen Aktivpositionen im Vergleich zu langfristigen Refinanzierungsquellen. Mit dem Net Stable Funding Ratio (NSFR) soll erreicht werden, dass Banken längerfristige Refinanzierungsquellen nutzen und sich nicht nur auf kurzfristige Quellen beschränken. Die Vorgaben bezüglich der Net Stable Funding Ratio (NSFR) müssen erstmals ab 2018 eingehalten werden.<sup>35</sup>

## 2.5 Zusammenfassung

---

<sup>34</sup>vgl. [Pau11b, S.56]

<sup>35</sup>vgl. [Pau11b, S.56f]

## **3 SAP LRM und BusinessObjects Xcelsius**

### **3.1 Einleitung**

### **3.2 SAP LRM**

#### **3.2.1 Funktionen**

Das SAP LRM ist eine Anwendung für Banken, um Liquiditätsrisiken managen zu können. Es unterstützt den kompletten Risikomanagementprozess von der Identifikation der Risiken, der genaueren Analyse eines Risikos über die Findung von Maßnahmen zur Behandlung des Risikos bis zur Berichtserstellung und Überwachung. Der Schwerpunkt liegt dabei besonders auf dem letzten Teil, der Berichtserstellung und Überwachung. Neben einer Weboberfläche wird der Zugriff auf die Informationen des SAP LRM auch über mobile Endgeräte, wie zum Beispiel dem Apple iPad, durch die Bereitstellung von entsprechenden Applikationen ermöglicht.<sup>36</sup>

Durch den Einsatz der LRM Lösung können Banken rechtliche Anforderungen, wie zum Beispiel die Einhaltung des Liquidity Coverage Ratio (LCR) und des Net Stable Funding Ratio (NSFR) im Rahmen von Basel III sicherstellen. Zusätzlich können durch die Bereitstellung einer Berechnungskomponente eigene Kennzahlen zur

---

<sup>36</sup>vgl. [Fre12, S.10 und S.23]

Steuerung und Überwachung der Bank definiert und umgesetzt werden. Durch den Einsatz von der SAP High Performance Analytic Appliance (HANA) ist es möglich, eine zahlungsstromorientierte Herangehensweise trotz der großen Datenmengen zu ermöglichen und gleichzeitig in Echtzeit<sup>37</sup> mit den Daten zu arbeiten. Im Bereich des Liquiditätsrisikomanagement müssen mehrere Geschäftsbereiche involviert werden. Deshalb bietet das SAP LRM kollaborative Möglichkeiten um dies zu unterstützen.<sup>38</sup>

Als Datenquelle für die Zahlungsströme werden neben SAP-Systemen, meist ein Enterprise Resource Planning (ERP)-System, auch nicht-SAP-Systeme unterstützt. Somit kann sichergestellt werden, dass Analysen auf einer vollständigen Datenbasis durchgeführt werden. Für die Analysen können verschiedene Szenarien simuliert werden. Bei jedem Szenario werden dabei Annahmen über die zukünftige finanzielle Entwicklung getroffen. Meist handelt es sich dabei um die Herauf- oder Herabstufung der Kreditwürdigkeit von Kunden, die Entwicklung von Aktienkursen, das Verhalten der Marktteilnehmer oder Währungskursschwankungen. Durch diesen Spielraum ist es möglich, ein genaues Bild über die Liquiditätsrisiken der Bank zu erhalten.<sup>39</sup>

## **3.2.2 Architektur**

### **3.2.2.1 NGAP**

Die Next Generation ABAP Plattform (NGAP) stellt die technologische Grundlage für das SAP LRM dar. Sie ist eine Plattform für in ABAP geschriebene Unternehmensanwendungen und soll eine moderne Alternative zu dem SAP NetWeaver Application Server (NetWeaver) bieten. Dieser stellte bisher die Grundlage für fast alle Anwendungen der SAP dar. Entscheidende Merkmale ist die Nutzung von HANA im ABAP-Umfeld, eine Verschlankung des gesamten Technologiestacks, die Einführung einer auf Eclipse<sup>40</sup> basierenden Entwicklungsumgebung und die vereinfachte

---

<sup>37</sup>Die Echtzeitanforderung wurde in diesem Kontext auf eine Reaktionszeit von einer Sekunde festgelegt.

<sup>38</sup>vgl. [Fre12, S.9 und S.17f]

<sup>39</sup>vgl. [Fre12, S.18 und S.45]

<sup>40</sup>Eclipse IDE - <http://www.eclipse.org/>

Bereitstellung von REST<sup>GL</sup>-basierenden Webservices.<sup>41</sup>

Der NetWeaver hat sich als ausgereifte und verlässliche Plattform erweisen, die sowohl einen Java- als auch einen ABAP Stack besitzt und in Verbindung mit vielen Betriebssystemen und Datenbanken genutzt werden kann. Durch die Unterstützung der beiden Programmiersprachen, der Betriebssysteme und der verschiedenen Datenbanken ist das Thema Kompatibilität bei der Weiterentwicklung ein entscheidendes Merkmal. Dadurch wird die Komplexität von Erweiterungen deutlich erhöht und durch die lange Entwicklungszeit des NetWeaver sind Codeteile bis zu 20 Jahre alt.<sup>42</sup>

Diese Nachteile sind der Grund für die Entwicklung der NGAP. Die NGAP basiert auf dem Quellcode des NetWeaver in der Version 7.2. Die wichtigsten Veränderungen sind das Entfernen des Java-Stacks, die Verschlinkung des zugrundeliegenden Kernels und des ABAP-Stacks. Der Kernel wurde komplett restrukturiert, weiter modularisiert und die Unterstützung auf wenige Betriebssysteme und die Datenbank von HANA als einzige Datenbank reduziert. Durch den Wegfall der Kompatibilitätsanforderungen konnten große Teile des ABAP-Stacks entfernt oder vereinfacht werden.<sup>43</sup> Des Weiteren wurde zum ersten mal ein durchgehendes API für die Programmierung von Anwendungen auf Basis der NGAP eingeführt. Dadurch sind interne Änderungen an der NGAP möglich, ohne die Anwendungen anpassen zu müssen.<sup>44</sup> Insgesamt konnten durch die Änderungen die Anzahl der Quellcodezeilen um 60% reduziert werden. Dadurch ergeben sich Vorteile in der Wartung da potentiell weniger Fehler enthalten sein können.<sup>45</sup>

Zusammengefasst bietet die NGAP eine einfacherer Möglichkeit für die Entwicklung von Anwendungen in Verbindung von aktuellen Innovationen wie zum Beispiel HANA oder dem Anschluss von mobilen Clients auf Basis von REST-Webservices.

---

<sup>41</sup>vgl. [SAP12a, S.1f]

<sup>42</sup>vgl. [BMH10] und [SAP12a, S.1]

<sup>43</sup>vgl. [Har10]

<sup>44</sup>vgl. [BMH10]

<sup>45</sup>vgl. [SAP12a, S.2]

### 3.2.2.2 HANA

Die HANA ist ein Produkt der SAP und besteht aus Softwarekomponenten, die in Kombination mit zertifizierter Hardware verkauft werden. Es ist die Reaktion auf den Bedarf nach der schnellen Auswertung von großen Datenmengen. Dies soll durch die Ausnutzung der Leistungssteigerung von modernen Computern erreicht werden. Hier ist zum einen die Entwicklung von Einkernprozessoren zu Mehrkernprozessoren zu nennen und zum anderen die Verfügbarkeit von schnellem Hauptspeicher in der benötigten Größe zu vertretbaren Kosten.<sup>46</sup> Das Ziel von HANA ist es aktuelle operationale Daten in Verbindung mit bestehenden historischen Daten in Echtzeit zu Analysieren und somit Informationen zu gewinnen.<sup>47</sup>

Der Kern der HANA bildet dabei ein Hauptspeicherbasiertes Datenbankmanagementsystem (DBMS). Dabei werden alle Daten nicht wie bei traditionellen DBMS auf Festplatten gespeichert, sondern im Hauptspeicher gehalten um höhere Zugriffsgeschwindigkeiten zu erreichen.<sup>48</sup> Außerdem ist neben der zeilenbasierten Organisation der Daten im Speicher auch die spaltenbasierte Organisation möglich. Die zeilenbasierte Organisation ist von Vorteil, wenn auf einzelne Datensätze komplett zugegriffen werden soll, die spaltenbasierte Organisation ist bei Tabellen mit einer hohen Anzahl an Spalten und bei spaltenbasierten Operationen wie der Aggregation oder der Suche überlegen. Durch die Unterstützung von beiden Organisationsformen kann die jeweils beste Form gewählt werden.<sup>49</sup>

Veränderungen in einem Datensatz einer Tabelle können auf Wunsch nicht in dem Eintrag der Tabelle direkt geändert, sondern nur die Differenzen an die Tabelle angefügt werden. Dadurch bleibt die Information, wie sich der Datensatz im Laufe der Zeit verändert hat, erhalten und kann in späteren Auswertungen als weitere Information hinzugezogen werden. Zusätzlich ist das Anfügen der Veränderung schneller durchzuführen wie die Veränderung des bestehenden Datensatzes.<sup>50</sup>

---

<sup>46</sup>vgl. [PZ11, S.14f]

<sup>47</sup>vgl. [SAP12c]

<sup>48</sup>vgl. [Kle10, S.12f]

<sup>49</sup>vgl. [Kle10, S.13f]

<sup>50</sup>vgl. [PZ11, S.109f]

Zu den genannten Veränderungen wird in Anwendungen, die auf Basis von HANA entwickelt werden, versucht, ein Teil der Anwendungslogik schon auf der Datenbank selbst zu berechnen.<sup>51</sup> Erreicht wird dies durch die Erweiterung der Abfragesprache Structured Query Language (SQL) zu SQLScript<sup>GL</sup>. Mit SQLScript ist es unter anderem durch das Hinzufügen von Datentypen, Prozeduren und Operationen möglich, Anwendungslogik abzubilden. Diese Berechnungen können von der Datenbank durch Parallelisierung sehr schnell durchgeführt werden.<sup>52</sup> Als Resultat kann die Datenübertragung zwischen dem DBMS und der Anwendung verringert werden. Es muss nur noch das Ergebnis und nicht die Datensätze, auf denen das Ergebnis basiert, übertragen werden. Zusätzlich wird die Komplexität der Anwendung verringert, da ein Teil der Logik von dem DBMS übernommen wird.

### 3.2.2.3 Oberon

Als Oberflächentechnologie wird für das SAP LRM das Oberon Framework genutzt. Es wurde ursprünglich als Oberfläche für SAP Business ByDesign<sup>GL</sup> entwickelt und wird jetzt auch für weitere Anwendungen unter anderem für Anwendungen, die auf HANA und NGAP basieren, verwendet.<sup>53</sup> Ein Beispiel eines mit dem Oberon Framework umgesetzte Oberfläche ist in Abbildung 1 auf S. 14 zu sehen.

Oberflächen werden mit dem Oberon Framework mit der Programmiersprache C# und Microsoft Silverlight<sup>54</sup> entwickelt. Silverlight ist eine Technologie von Microsoft, die als Plugin für verschiedene moderne Browser verfügbar ist und die Entwicklung von Rich Internet Application (RIA)<sup>GL</sup> unterstützt. Die Rich Internet Application (RIA) stellt dabei die erste Schicht der drei Schichtenarchitektur von Anwendungen dar. Die weiteren Schichten, die Anwendungslogik und die Persistenz, werden von NGAP und HANA übernommen.<sup>55</sup>

Das Oberon Framework ist als Standardframework ausgelegt und versucht, möglichst

---

<sup>51</sup>vgl. [PZ11, S.155f]

<sup>52</sup>vgl. [SAP11, S.9f]

<sup>53</sup>vgl. [SAP12b]

<sup>54</sup>Microsoft Silverlight - <http://www.microsoft.com/silverlight/>

<sup>55</sup>vgl. [SAP12b]

**CASH FLOW**

000000000011ED18AC...  
Baseline - Maturity View

**Key Information**

Cash Flow Origin: SD\_1  
Contract ID: 000000000011ED1...  
Securities Account ID:  
Node Type:  
Scenario ID: Baseline  
Cash Flow View: Maturity View  
Valid From: 31.10.2011

**Overview** Header Items Feeds

**Administrative Data**

Valid From: 31.10.2011 11:33 UTC  
Valid To: 31.12.9999 23:59 UTC  
Legal Entity: LE05  
Off Balance Sheet: No  
Book:  
Cash Flow Name:  
Cash Flow Category:

**Business Data**

Internal Contract: No  
Counterparty ID:  
Counterparty Name: Krieg  
Product: COUNTERBALANCING  
Product Type: Central Bank Asset  
ISIN:  
Asset/Liability:  
Collateralized: No

**Cash Flow Items**

Payment Date	Amount	Cash Flow Item Category	Inflow/Outflow	Evaluation Type
10.04.2013	300.050,00 EUR	Annuity Repayment		
05.03.2012	299.973,00 EUR	Final Repayment		
03.10.2014	300.046,00 EUR	Final Repayment		
24.10.2012	300.011,00 EUR	Annuity Repayment		
22.10.2013	299.970,00 EUR	Payment		
23.02.2014	300.048,00 EUR	Annuity Repayment		
24.01.2014	299.950,00 EUR	Final Repayment		
12.08.2013	299.989,00 EUR	Final Repayment		
22.07.2014	299.979,00 EUR	Final Repayment		
19.05.2013	300.043,00 EUR	Final Repayment		

Abbildung 1: Darstellung eines CashFlows-Eintrags mit dem Oberon-Framework

viele Anforderungen abzudecken um so in vielen Produkten verwendet werden zu können. Allerdings ist es sehr Wahrscheinlich, dass spezielle Anforderungen eines bestimmten Bereichs nicht mit dem Oberon Framework abgedeckt werden können. In diesem Fall müssen eigene Lösungen für die Anforderungen entwickelt werden.<sup>56</sup>

### 3.2.3 Berechnungskomponente

Die Berechnungskomponente stellt den wichtigsten Teil des SAP LRM dar. Mit ihr werden alle Berechnungen im Umfeld des Liquiditätsrisikomanagements zentral durchgeführt. Die Berechnung findet auf Basis von Liquiditätsgruppen statt. Mit

<sup>56</sup>vgl. [SAP12b]

Hilfe von Liquiditätsgruppen werden sowohl aggregierter Zahlungsstrom als auch Kennzahlen einheitlich berechnet. Liquiditätsgruppen können in Polyhierarchien<sup>57</sup> angeordnet werden. Dadurch können Berechnungen modular aufgebaut, wiederverwendet und einfach nachvollzogen werden. Das Abbilden von komplexen Berechnungsvorschriften wird erleichtert.<sup>58</sup>

Es existieren zwei Arten von Liquiditätsgruppen: Die Datenbankselektion und die Berechnung. Die Datenbankselektion bildet immer den Ursprung einer Berechnung. Dabei werden einzelne Zahlungsströme nach festgelegten Selektionskriterien ausgewählt, und nach Bedingungen zusammengefasst. Selektionskriterien können zum Beispiel der hinter liegende Produkttyp des Zahlungsstroms wie zum Beispiel verzinsliches Wertpapier, oder das entsprechende Finanzrating, zum Beispiel AAA sein. Gleichzeitig findet eine Währungsumwandlung in eine einheitliche Zielwährung statt. Mehrere Zahlungsströme werden in Abhängigkeit ihrer Fälligkeit in Behältern zusammengefasst. Diese Behälter werden von dem festgelegten Fälligkeitsband definiert.<sup>59</sup>

Ein Fälligkeitsband ist eine spezielle Einteilung der Zeitachse. Der Grund für die Einführung von Fälligkeitsbändern liegt in der späteren Auswertung der berechneten Werte. Dabei ist für naheliegende Zeiträume, zum Beispiel die nächste Woche, der entsprechende Wert an jedem einzelnen Tag entscheidend. Für den weiteren Horizont, zum Beispiel den Zeitraum zwischen drei und fünf Jahren, ist nicht jeder einzelne Tag entscheidend, es reichen aggregierte Werte, zum Beispiel nach Monat oder Jahr. Somit kann in einer Auswertung sowohl die aktuelle Situation begutachtet werden, als auch die langfristige Auswirkungen ohne, dass die Auswertung unübersichtlich wird. Jeder Eintrag entspricht dabei einem Behälter. Die Anzahl und Größe der Behälter ist in einem Fälligkeitsband definiert.<sup>60</sup>

Neben der Datenbankselektion existiert die Liquiditätsgruppenart Berechnung. Dabei werden aggregierte Zahlungsströme nach verschiedenen Regeln verrechnet. Zur

---

<sup>57</sup>Polyhierarchien sind hierarchische Strukturen, bei der ein Element mehrere übergeordnete Elemente haben kann.

<sup>58</sup>vgl. [Fre12, S.38]

<sup>59</sup>vgl. [Fre12, S.40]

<sup>60</sup>vgl. [Fre12, S.44]



Verfügung stehende unter anderem Addition, Subtraktion, Multiplikation und Division.

Generell hat eine Liquiditätsgruppe genau einen Output-Parameter, eine Berechnungsregel und kann zusätzlich mehrere Input-Parameter besitzen. Die Berechnung innerhalb einer Liquiditätsgruppe kann von szenarioabhängigen Variablen beeinflusst werden. Damit lassen sich zum Beispiel erwartete Kreditausfälle simulieren. Über diese Variablen werden die Szenarien umgesetzt. Mehrere Variablenausprägungen legen ein Szenario fest.<sup>61</sup>

Zur Laufzeit wird die Berechnungskomponente mit vier Parametern aufgerufen. In dem ersten Parameter werden eine oder mehrere Liquiditätsgruppen festgelegt, von denen später das Ergebnis zurückgeliefert wird. Ein weiterer Parameter legt den Stichtag fest, nach dem die Zahlungsströme ausgewählt werden. Jede Berechnung wird immer auf Basis von einem Fälligkeitsband durchgeführt, welches die Behälter festlegt, in dem die Ergebnisse zusammengefasst werden. Schließlich wird noch eine Zielwährung festgelegt, in die alle Zahlungsströme umgerechnet werden.

Das Resultat der Berechnung ist jeweils der Output der festgelegten Liquiditätsgruppen. Zur Berechnung der Gruppen müssen in der Regel im Hintergrund weitere Liquiditätsgruppen berechnet werden, die allerdings nicht als Ergebnis zurückgeliefert werden. Während einer Berechnung werden Ergebnisse von Gruppen, die schon einmal berechnet wurden, zwischengespeichert, um Rechenaufwand zu sparen. Die Gruppen, die berechnet werden müssen, können aus der Hierarchie der entsprechenden Liquiditätsgruppe abgelesen werden.<sup>62</sup>

---

<sup>61</sup>vgl. [Fre12, S.39 und S.45]

<sup>62</sup>vgl. [Fre12, S.44 und S.47]

## 3.3 BusinessObjects Xcelsius

### 3.3.1 Überblick und Funktionsumfang

Xcelsius ist ein Teil der SAP BusinessObjects-Portfolio. Dabei handelt es sich um Anwendungen, mit denen Daten in einem Unternehmen analysiert und ausgewertet werden können. Xcelsius ist davon ein Tool zur Visualisierung von Daten durch die Erstellung von interaktiven Dashboards.

Dazu bietet es viele Möglichkeiten um Daten visualisieren und interaktiv beeinflussen zu können. Neben 20 Diagrammtypen, darunter unter anderem Linien-, Kreis-, Balken- und Flächendiagramme, stehen Schaltflächen zur Manipulation, wie zum Beispiel Schieberegler, Optionsfelder und Drehknöpfe, zur Verfügung. Zusätzlich können Kartenelemente zur Darstellung von Geoinformationen genutzt werden und Bilder und Texte unterstützend in ein Dashboard eingebunden werden. Viele Komponenten bieten zusätzlich Einstellungsmöglichkeiten, durch die das Aussehen oder das Verhalten angepasst werden kann.<sup>63</sup>

Die Daten, auf denen die Visualisierungen in Xcelsius basieren, werden durch die Integration von Microsoft Excel verwaltet. Excel steht dabei in vollem Umfang zur Verfügung. Es können sowohl Berechnungen, als auch Formatierungen mit Excel durchgeführt werden. In Xcelsius wird Excel als Schnittstelle zwischen der Datenbeschaffung aus verschiedensten Quellen und der eigentlichen Visualisierung genutzt. Alle Daten, die visualisiert werden sollen, werden in ein Tabellenkalkulationsblatt von Excel geschrieben und von dort aus ausgelesen. Die Daten können demnach vor der Darstellung in Excel beliebig bearbeitet werden.<sup>64</sup>

Die gesamte Erstellung eines Dashboards findet nach dem „What you see is what you get“-Prinzip statt. Der Anwender sieht schon während der Erstellung, wie das Ergebnis aussieht. Durch die Verwendung von Excel und dem Umsetzen des Wysiwyg-Prinzip benötigen Nutzer meist nur eine geringe Einarbeitungszeit und die Bedie-

---

<sup>63</sup>vgl. [Egg09, S.31f und S.229f]

<sup>64</sup>vgl. [Egg09, S.32 und S.236]

nung kann als intuitiv beschrieben werden. Besondere Programmiersprachenkenntnisse sind für die Erstellung von Dashboards nicht erforderlich.<sup>65</sup>

Die mit Xcelsius erstellten Dashboards werden mit Hilfe der Adobe Flex Technologie dargestellt. Dadurch ist kein spezieller Server oder ein DBMS notwendig. Es wird nur das Flash-Plugin benötigt. Außerdem muss dafür für die Darstellung keine Netzwerkverbindung zu einem Server oder dem Internet vorhanden sein. Zusätzlich können Dashboards in verschiedene Formate exportiert werden um sie so in Microsoft PowerPoint-Präsentationen oder auf Internetseiten einzubinden.<sup>66</sup>

Als Datenquellen für ein Dashboard stehen mehrere Möglichkeiten zur Verfügung. Zum einen können Daten einfach in das Excel kopiert werden. Es ist auch möglich, ein SAP Business Warehouse System direkt anzuschließen, oder Daten über Webservices zu beziehen. Hierzu wird das Extensible Markup Language (XML)-Format verwendet. Außerdem existieren weitere Möglichkeiten, mit dem Dashboards zum Beispiel untereinander Daten austauschen können.<sup>67</sup>

### 3.3.2 Bedienungskonzept

Die Oberfläche von Xcelsius lässt sich in vier Bereiche aufteilen. Die Bereiche sind in Abbildung 2 auf S. 19 dargestellt. Der erste Bereich zeigt die Zeichenfläche, also das eigentliche Dashboard an. Auf der Zeichenfläche können Komponenten beliebig platziert werden. Alle verfügbaren Komponenten werden in dem Bereich 2 auf der linken Seite nach Kategorien geordnet dargestellt. Um eine Komponente auf der Zeichenfläche zu platzieren, wählt der Nutzer die Komponente aus und platziert sie per Drag& Drop auf der Zeichenfläche.

Viele Komponenten bieten Einstellungsmöglichkeiten an, mit denen das Verhalten und das Aussehen der Komponente festgelegt werden kann. Wird eine Komponente auf der Zeichenfläche ausgewählt, werden alle verfügbaren Eigenschaften in Bereich

---

<sup>65</sup>vgl. [Egg09, S.32 und S.239]

<sup>66</sup>vgl. [Egg09, S.31f und S.230]

<sup>67</sup>vgl. [Egg09, S.283f]

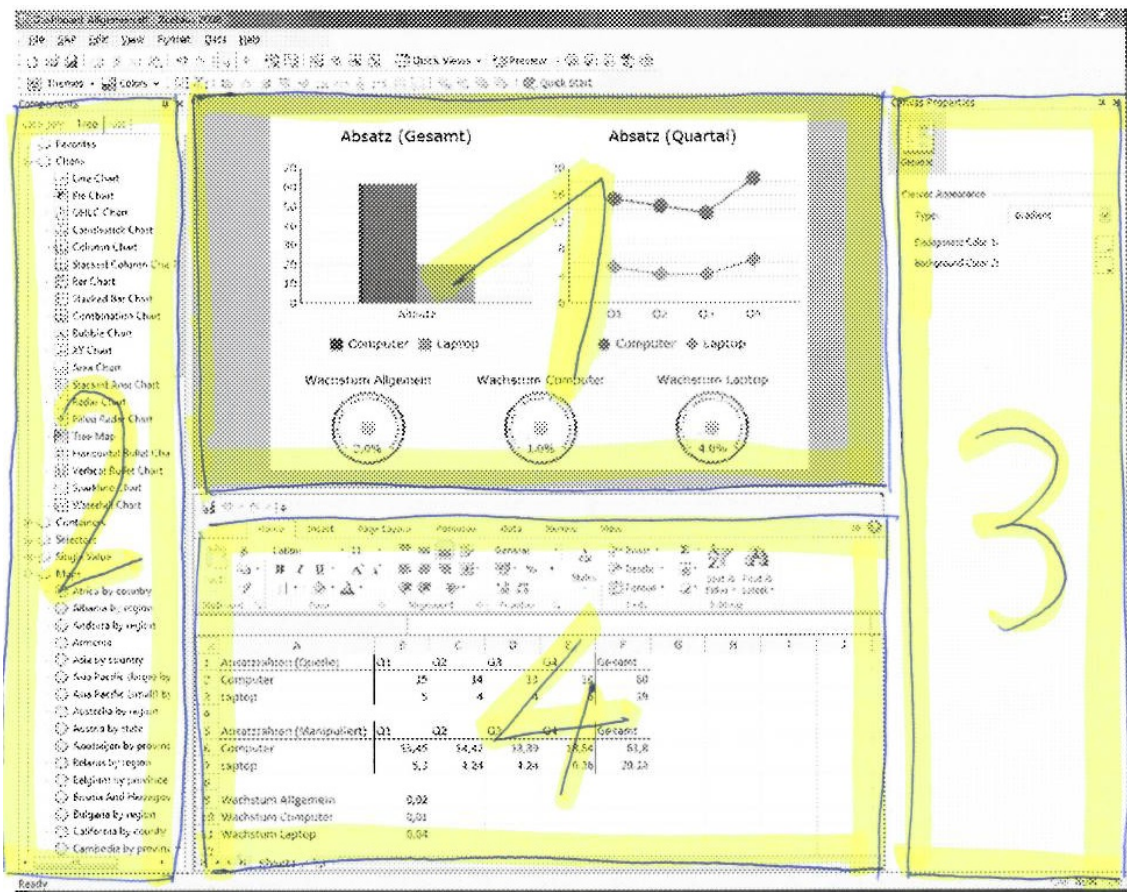


Abbildung 2: [TODO] WT: Aufbau der Benutzeroberfläche von Xcelsius

3 auf der rechten Seite dargestellt. Wichtige Eigenschaften einer Komponente stellen oft die Daten dar, die die Komponenten anzeigen soll. Alle Daten werden in Bereich 4 festgelegt. Die Verbindung zwischen der Datenhalten (Microsoft Excel in Bereich 4) und der Darstellung (Zeichenfläche in Bereich 1) von Xcelsius erfolgt durch das Referenzieren von Zellen oder Zellbereichen über die Eigenschaften einer Komponente (Bereich 3).<sup>68</sup>

Der generelle Ablauf des Erstellens von Dashboards soll an dem Beispiel der Absatzzahlen von Produkten in Abbildung 3 auf S. 20 verdeutlicht werden: Zunächst werden Daten in das Excel in einen festgelegten Bereich eingetragen. Dies Daten können entweder fest eingegeben werden, meistens werden sie aber dynamisch über

<sup>68</sup>vgl. [Egg09, S.229]

eine Datenquelle, zum Beispiel einen Webservice in den Bereich geschrieben. In dem Beispiel wurden die Absatzzahlen händisch in den Zellbereich 1 eingetragen. Als nächstes können in Excel Berechnungen durchgeführt werden. In dem Beispiel wird die Summe der Absatzzahlen pro Jahr durch eine Excel-Formel berechnet. Um die Interaktivität von Dashboards zu zeigen, soll der Nutzer die Möglichkeit haben, die Auswirkungen von verschiedenen Wachstumsraten interaktiv zu sehen. Dazu werden die veränderten Absatzzahlen (Bereich 2) auf Basis der Quelldaten in Bereich 1 und den Wachstumsdaten in Bereich 3 berechnet.<sup>69</sup>

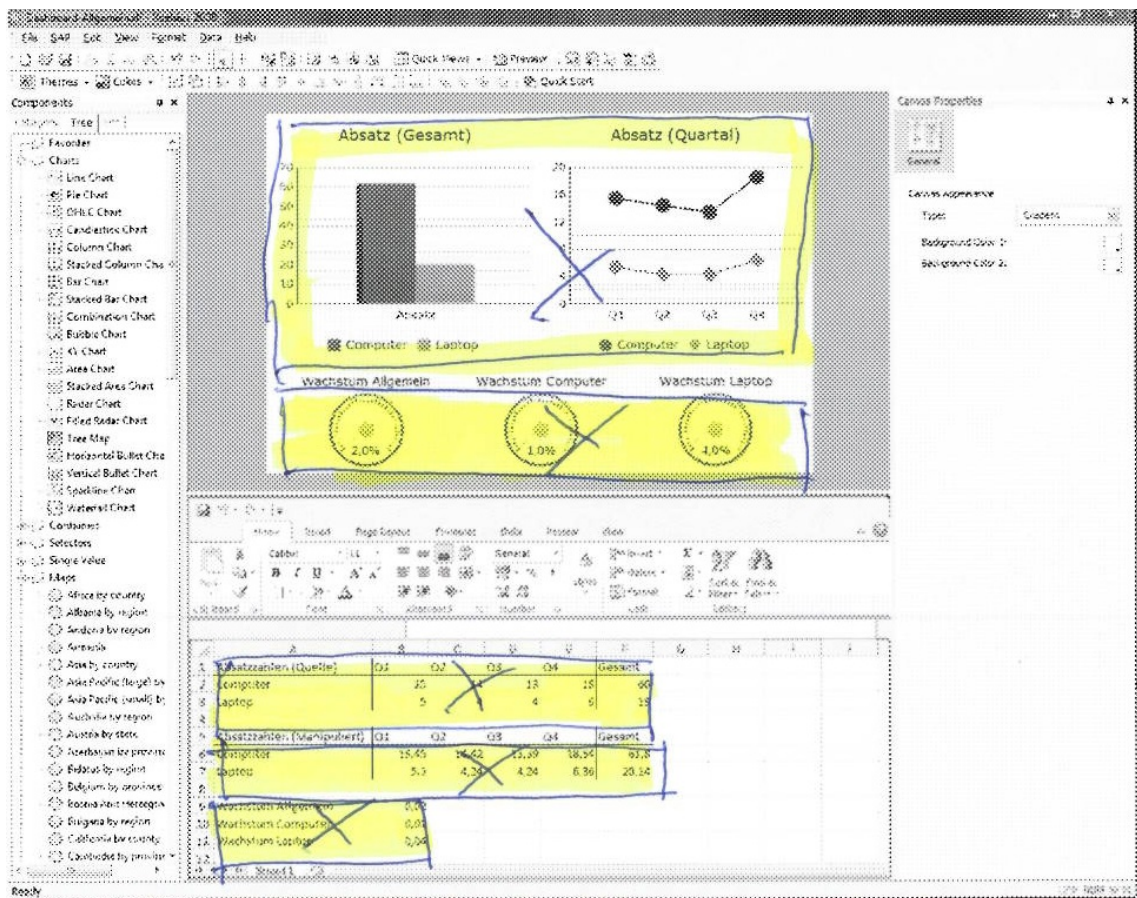


Abbildung 3: [TODO] WT: Beispiel eines Dashboards in Xcelsius

Schließlich werden auf der Zeichenfläche zwei Diagramme festgelegt, die ihre Daten aus Bereich 2 auslesen. Die Wachstumsraten können mit drei Drehknöpfe festgelegt werden. Die Werte der Drehknöpfe sind mit den Zellen in Bereich 3 verknüpft.

<sup>69</sup>vgl. [Egg09, S.229f]



Dreht der Nutzer an einem Drehknopf, wird der aktualisierte Wert in die Zelle in Bereich 3 geschrieben. Da die Datenquelle der Diagramme mit Bezug auf die Wachstumswerte berechnet werden, verändert sich dynamisch die Diagramme. Das Prinzip ist schematisch noch einmal in Abbildung 4 dargestellt. So ist es möglich, mit einfachen Mitteln, interaktive Dashboards zu erstellen.<sup>70</sup>

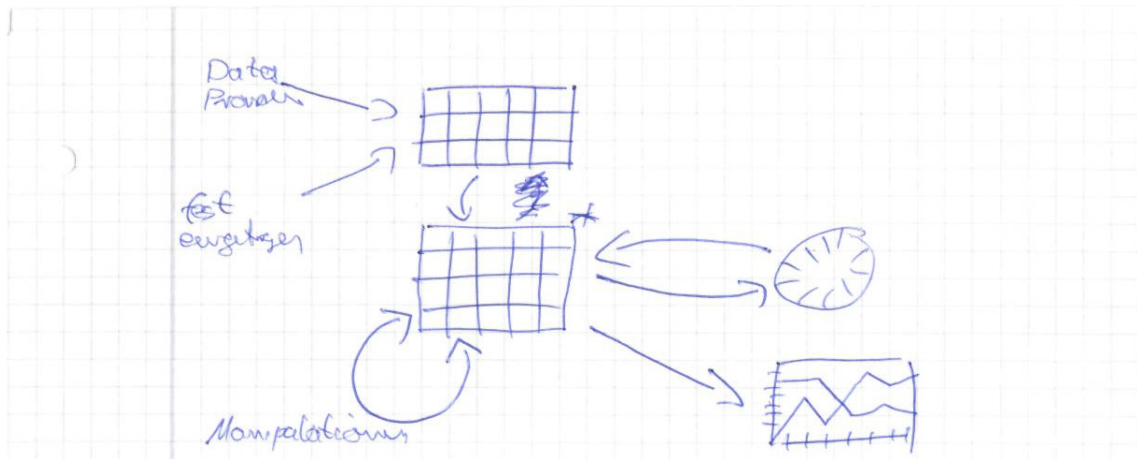


Abbildung 4: [TODO] WT Grundlegende Arbeitsweise mit Xcelsius

### 3.3.3 Architektur

Xcelsius basiert auf dem .NET Framework<sup>GL</sup> von Microsoft und ist in weiten Teilen in C++ geschrieben. Neben C++ wird ActionScript<sup>GL</sup> und das Adobe Flex Framework für die Benutzeroberfläche eingesetzt. Die Architektur von Xcelsius ist modular aufgebaut und besteht aus mehreren Teilen. Ein Überblick ist in Abbildung 5 auf S. 22 dargestellt.

Das zentrale Element ist das Document. Dabei handelt es sich um ein Objektmodell, das den aktuellen Zustand des Dashboards repräsentiert. Für die Darstellung wird das Canvas verwendet, welches in ActionScript geschrieben ist. Die Canvas-Komponente nutzt dabei mehrere Hilfskomponenten. Dazu gehört das ApplicationTemplate, welches das Grundgerüst von alle Dashboards darstellt und mehreren

<sup>70</sup>vgl. [Egg09, S.237]

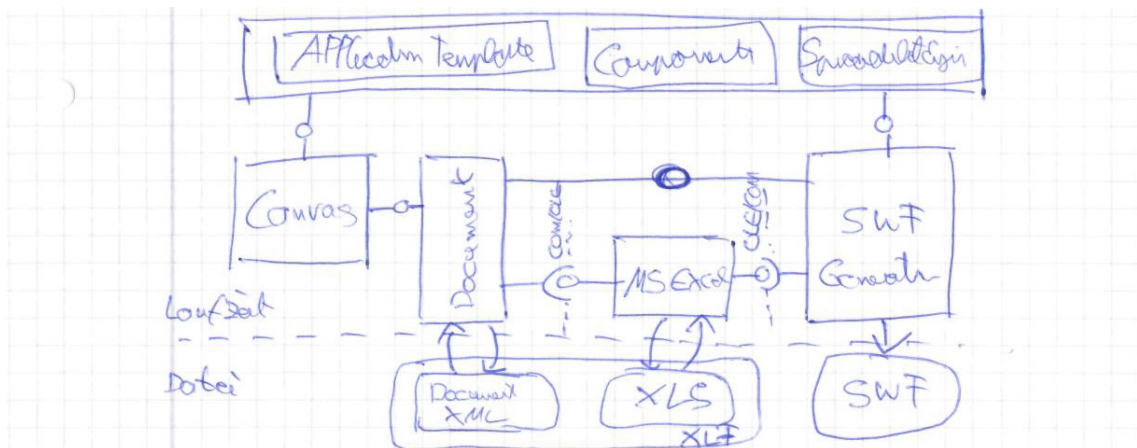


Abbildung 5: [TODO] WT Architekturüberblick von Xcelsius

Components, die die einzelnen Elemente eines Dashboards darstellen. Dies ist auch der Ansatzpunkt für spätere Erweiterungen.

An das Document ist mittels COM/OLE [TODO] Microsoft Excel angebunden. Speichern und Laden eines Dashboards übernimmt die XLF-Komponente. Diese erstellt oder lädt eine Xcelsius File (.xlf), welches intern aus dem Document im XML-Format und einer Excel-Datei besteht.

Für den Export des fertigen Dashboards existiert der SWF Generator. Aus dem Document und den Daten und Formeln, die in Microsoft Excel erstellt wurden, wird eine .swf Datei generiert. In dieser werden nun alle Berechnungen, die bis jetzt von Microsoft Excel übernommen wurden, von einer weiteren Hilfsklasse, der Spreadsheet Engine berechnet.<sup>71</sup>

### 3.3.4 Erweiterungsmöglichkeiten

Durch die Einstellungsmöglichkeiten der einzelnen Komponenten und die Einbindung des vollen Funktionsumfang von Microsoft Excel ist Xcelsius von sich aus sehr flexibel. Zusätzlich dazu existiert noch ein SDK<sup>GL</sup> mit dem komplett eigene Komponenten erstellt werden können. Dadurch ist es möglich, Xcelsius um nahezu beliebige

<sup>71</sup>vgl. [KHZC09]

Funktionen zu erweitern.

Die Erweiterungen, die mit dem SDK programmiert werden können, müssen auf dem Adobe Flex SDK 2.0.1 basieren. Die Programmiersprache ist ActionScript. Es können drei verschiedene Typen von Erweiterungen mit dem SDK entwickelt werden. Dabei handelt es sich um visuelle Komponenten, die später auf dem erstellten Dashboard dargestellt werden, um Datenverbindungen, die nicht auf dem Dashboard angezeigt werden, sondern nur im Hintergrund aktiv sind und in der Regel Daten in das Excel eintragen. Die dritte Möglichkeit ist die Erweiterung der Steuerungslogik von Excel, spielt aber im Rahmen dieser Bachelorarbeit keine Rolle.<sup>72</sup>

Die visuelle Komponente wiederum besteht aus zwei Teilen, die separat von einander entwickelt werden können. Der erste Teil ist für die Darstellung der Komponente auf dem Zeichenblatt des Dashboards verantwortlich. Der zweite Teil übernimmt die Darstellung der Eigenschaften, die die Komponente bieten soll. Es ist gleichzeitig die Verbindung zwischen Xcelsius und der entwickelten Komponente. Die Logik der Komponente kann je nach Bedarf auf beide Teile verteilt werden. Das Ergebnis der Entwicklung der beiden Teilkomponenten sind zwei kompilierte .swf Dateien [TODO !?]. Diese Dateien, in Verbindung mit Informationen zu der Komponente, wie zum Beispiel Name, Version, Typ und Entwickler werden mit dem Xcelsius SDK zu einem Xcelsius Plugin zusammengefügt. Dieses kann dann in Xcelsius importiert und die Komponenten so genutzt werden.<sup>73</sup> Den Vorgang zeigt Abbildung 6.

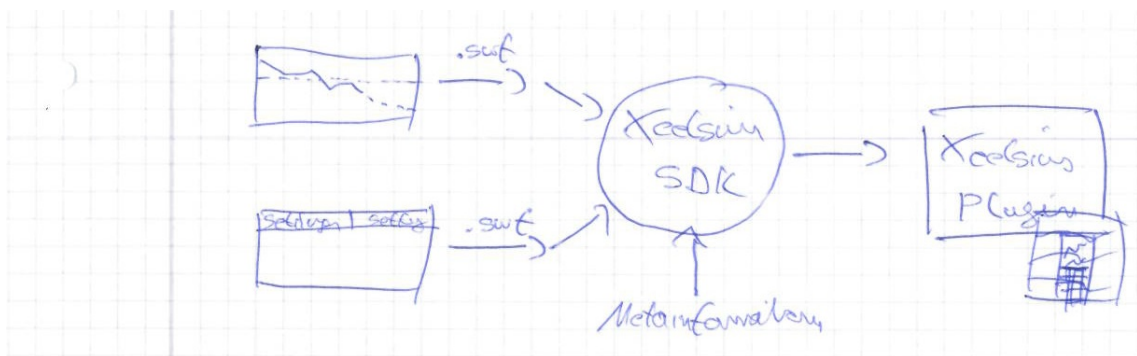


Abbildung 6: [TODO] Plunginsystem von Xcelsius

<sup>72</sup>vgl. [Egg09, S.229] und [Mac08, S.3f]

<sup>73</sup>vgl. [Egg09, S.283f] und [Mac08, S.6]



## **3.4 Zusammenfassung**

# 4 Spezifikation

## 4.1 Einleitung

## 4.2 Ausgangssituation

In dem derzeitigen Stand des SAP LRM gibt es für die Analyse der aktuellen Liquiditätssituation einer Bank zwei Einstiegsmöglichkeiten. Es existiert die Kennzahlenanalyse und die Finanzierungsmatrix. Diese unterscheiden sich in ihrer Nutzung.

Der Zweck der Kennzahlenanalyse ist die Darstellung von Kennzahlen, die nach einer festen Vorschrift berechnet werden können. Neben internen Kennzahlen, die eine Bank für sich selbst definiert hat, sind hier vor allem offizielle Kennzahlen von Bedeutung. Dazu gehören zum Beispiel im Rahmen von Basel III der Liquidity Coverage Ratio (LCR) und der Net Stable Funding Ratio (NSFR). Somit kann die Kennzahlenanalyse genutzt werden, um offizielle regulatorische Vorschriften zu überwachen. Die Darstellung der Kennzahlen ist ausschließlich in tabellarischer Form möglich. Ein Beispiel zeigt Abbildung 7 auf S. 26.

Neben der Kennzahlenanalyse existiert die Finanzierungsmatrix. Hier sollen nicht feste Kennzahlen dargestellt werden. Vielmehr wird die Möglichkeit geboten, die Liquiditätssituation der Bank frei zu analysieren. Sowohl die Finanzierungsmatrix, als auch die Kennzahlenanalyse, nutzen für die Berechnung der Werte die Berechnungskomponente. Bei der Finanzierungsmatrix hat der Nutzer die Möglichkeit, die Selektionskriterien in Form der nötigen Parameter für den Aufruf der Berechnungs-

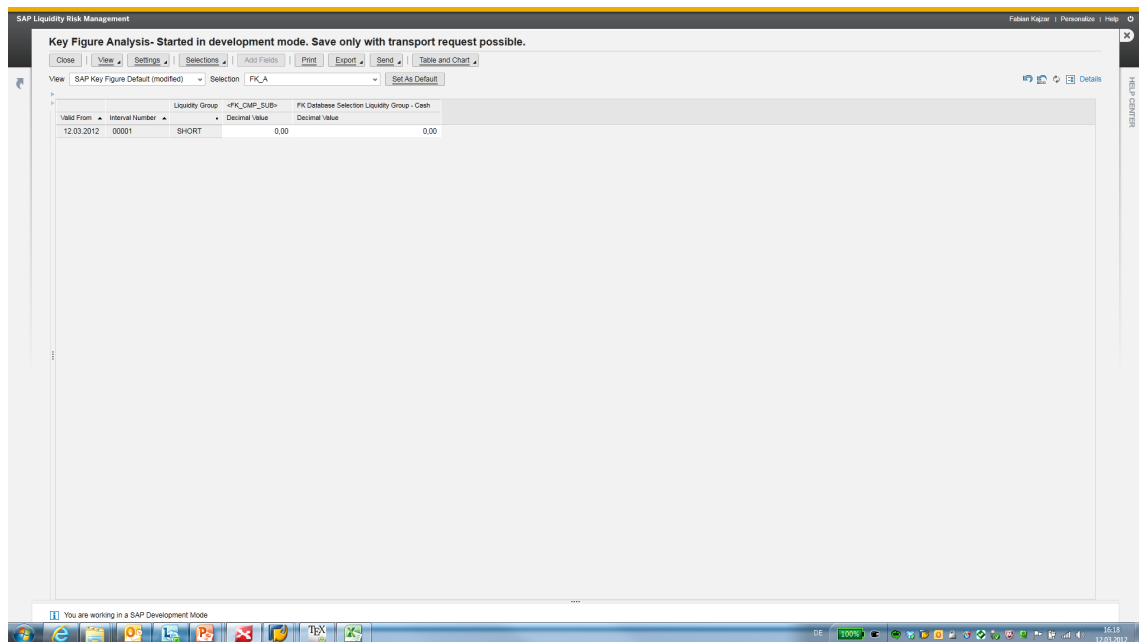


Abbildung 7: Darstellung der Kennzahlenanalyse im SAP LRM

komponente zur Laufzeit festlegen. Die Darstellung ist hier auf ein Liniendiagramm festgelegt. Abbildung 8 auf S. 27 zeigt die Finanzierungsmatrix.

Allgemein ist festzustellen, dass in der aktuellen Version keine Möglichkeit besteht, die Ergebnisse der Berechnungen in irgendeiner Form in anderen Auswertungen, zum Beispiel Auswertungen, die die gesamte Situation der Bank beschrieben, einzubinden. Des Weiteren besteht bei der Darstellung der Ergebnisse keine Formatierungsmöglichkeit. Somit sind zum Beispiel Farbe, Form und Größe des Diagramms fest vorgegeben und nicht durch den Nutzer änderbar.

## 4.3 Zielsetzung

Im Rahmen dieser Bachelorarbeit sollen einige Limitierungen, die in der aktuellen Version des SAP LRM vorhanden sind, aufgehoben werden. Hier ist vor allem die Bereitstellung von weiteren Alternativen zu nennen. Neben der Sicht auf die Liquiditätssituation der Bank durch die beiden Einstiegspunkte Kennzahlenanalyse und

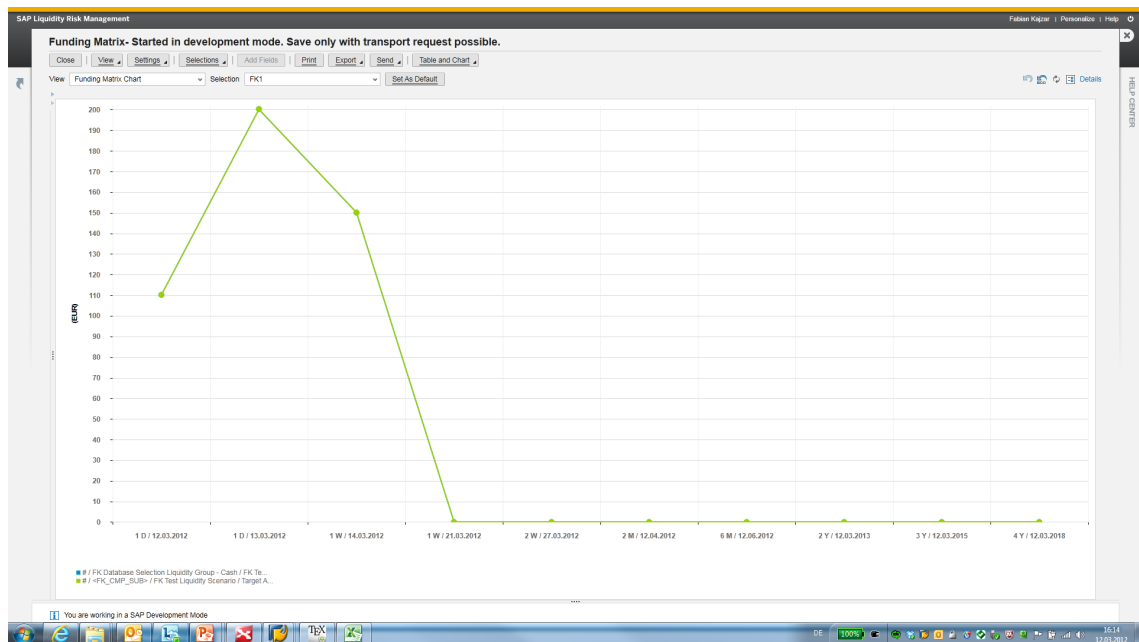


Abbildung 8: Darstellung der Finanzierungsmatrix im SAP LRM

der Finanzierungsmatrix im SAP LRM soll diese Auswertung mit Hilfe von weiteren Anwendungen durchgeführt werden können.

Durch die Anbindung von weiteren Anwendungen bietet sich die Möglichkeit, die Liquiditätssteuerung in die Gesamtsteuerung der Bank einzubinden. Ein Ziel ist es, in einen Bericht, der Daten zu der allgemeinen Situation der Bank anzeigt, auch Informationen über das Liquiditätsrisikomanagement aufzunehmen. Dadurch kann die Situation der Bank gesamtheitlich erfasst werden und Entscheidungen unter der Berücksichtigung von weiteren Informationen getroffen werden.

Simultan dazu können durch eine Anbindung des SAP LRM an weitere Anwendungen die jeweiligen Vorteile der angebundenen Anwendung ausgenutzt werden. Hier ist zum Beispiel die Flexibilität in der Darstellung der Daten zu nennen. Wenn die Anwendung weitere grafische Visualisierungsmöglichkeiten bietet, können diese auf die Daten des SAP LRM angewendet werden.

Insgesamt soll die Anbindung des SAP LRM mit möglichst wenig erforderlichen Einstellungen konfigurierbar sein. Gleichzeitig ist es erstrebenswert, dass das Kon-

zept der Anbindung und die Implementierung, die im Rahmen der Bachelorarbeit entwickelt wird, in der grundsätzlichen Idee und Architektur auch auf weitere Anwendungen übertragen werden kann.

## 4.4 Anforderungen

Für die Erweiterung werden im Rahmen der Spezifikation einige Anforderungen definiert. Die Anforderungen werden zunächst in funktionale und nicht-funktionale Anforderungen unterteilt.

Bei den funktionalen Anforderungen existieren die zwei Folgenden Hauptanforderungen:

(A1)     **Einbindung von Kennzahlen**

Die Erweiterung soll es ermöglichen, dass Kennzahlen, die im SAP LRM mit Hilfe der Berechnungskomponente ermittelt werden und im Bereich der Kennzahlenanalyse zur Verfügung stehen, in weitere Anwendungen integriert werden und dort weiterverarbeitet werden können.

(A2)     **Freie Analyse der Liquiditätssituation**

Mit der Erweiterung soll die Einbindung der Ergebnisse von beliebigen Liquiditätsgruppen ermöglicht werden. Die Liquiditätsgruppen sollen in Anlehnung an die Finanzierungsmatrix im SAP LRM eingestellt werden und auf der Berechnungskomponente basieren. Neben den erforderlichen Parametern, wie zum Beispiel dem genutzte Laufzeitband und der Zielwährung sollen weitere Einschränkungsmöglichkeiten geboten werden, welche die Berechnungskomponente unterstützt.

Durch die Umsetzung dieser beiden Anforderungen können die Ziele, die im Kapitel 4.3 auf S. 26 festgelegt sind, erreicht werden. Dadurch bieten sich für den Nutzer bei der Auswertung und Analyse der Liquiditätssituation einer Bank zusätzliche Möglichkeiten.

Neben den funktionalen Anforderungen wurden auch nicht-funktionale Anforderungen identifiziert. Diese sollen im Folgenden näher erläutert werden:

(A3) **Übertragbarkeit**

Im Rahmen der Bachelorarbeit wird exemplarisch die Anbindung einer weiteren Anwendung zur Auswertung an das SAP LRM implementiert werden. Das Konzept der Umsetzung soll dennoch auf weitere Anwendungen übertragen werden können.

(A4) **Korrektheit der Ergebnisse**

Die Daten, die mit Hilfe der Erweiterung in zusätzlichen Anwendungen verfügbar gemacht werden, müssen mit den Ergebnissen des SAP LRM übereinstimmen. Nur wenn das sichergestellt ist, kann die Erweiterung sinnvoll genutzt werden.

(A5) **Erweiterbarkeit**

Das SAP LRM befindet sich aktuell noch in einer ersten Version in Entwicklung. Weitere Versionen sind schon jetzt in Planung. Zusätzliche Funktionen und Möglichkeiten, die in einer Weiterentwicklung des SAP LRM eingeführt werden, sollen mit möglichst geringem Aufwand auch auf die Erweiterung übertragen werden können.

(A6) **Wartbarkeit**

Um die Folgekosten der Erweiterung, zum Beispiel im Hinblick auf die Beseitigung von Fehlern, möglichst gering zu halten, soll eine gute Wartbarkeit von Anfang an bei der Entwicklung beachtet werden.

(A7) **Sicherheitsanforderungen**

Die Erweiterungen wird Zugang zu den Daten des SAP LRM bieten. Es handelt sich dabei um vertrauliche Daten, die Zugangsbeschränkungen unterliegen. Es muss sichergestellt sein, dass die Zugangsbeschränkungen eingehalten werden und Daten nicht von unberechtigten Personen ausgelesen werden können.

(A8) **Leistung**

Ein wichtiges Unterscheidungsmerkmal des SAP LRM durch die Nutzung von HANA ist die schnelle Analyse von Daten. Die Erweiterung soll

die Geschwindigkeitsvorteile von HANA auch in anderen Anwendungen verfügbar machen.

Die Reihenfolge der Anforderungen spiegelt gleichzeitig auch die Priorität der Umsetzung wieder. Die wichtigsten Anforderungen sind Anforderung 3 (A3 auf S. 29) und Anforderung 4 (A4 auf S. 29). Anforderung 8 (A8 auf S. 29) steht bei der Entwicklung nicht im Mittelpunkt.

## 4.5 Anwendungsfälle

Zum besseren Verständnis des Ziels der Erweiterung werden die Hauptanwendungsfälle im Folgenden näher beschrieben. Der Anwendungsfall 1 konkretisiert die Anforderung 1 (A1 auf S. 28) und behandelt die Einsicht in wichtige Liquiditätskennzahlen einer Bank.

### Anwendungsfall 1: Liquiditätskennzahl einsehen

**Ziel**

Liquiditätskennzahl der Bank einsehen

**Vorbedingung**

Die Liquiditätskennzahl ist im SAP LRM angelegt und verfügbar

**Nachbedingung: Erfolg**

Die Liquiditätskennzahl kann eingesehen werden

**Nachbedingung: Fehlschlag**

Anzeige einer Fehlermeldung

**Aktuere**

Controller, Manager

**Auslösendes Ereignis**

Der Wert der Liquiditätskennzahl soll überprüft werden

**Beschreibung**

- 1 Die einzusehende Liquiditätskennzahl wird ausgewählt
- 2 Die Daten werden aus dem SAP LRM ermittelt
- 3 Die Daten werden der Anwendung zur Verfügung gestellt

Der Anwendungsfall 2 bezieht sich auf die Anforderung 2 (A2 auf S. 28). Hierbei geht es um die freie Analyse der Liquiditätssituation einer Bank.

**Anwendungsfall 2: Liquiditätssituation analysieren****Ziel**

Liquiditätssituation der Bank analysieren

**Vorbedingung**

Erforderliche Zahlungsströme, Liquiditätsszenarien, Liquiditätsgruppen und Laufzeitbänder sind im SAP LRM angelegt und verfügbar

**Nachbedingung: Erfolg**

Die Liquiditätssituation kann eingesehen werden

**Nachbedingung: Fehlschlag**

Anzeige einer Fehlermeldung

**Aktuere**

Controller, Manager

**Auslösendes Ereignis**

Es besteht Unklarheit über die Liquiditätssituation der Bank

**Beschreibung**

- 1 Der zu analysierende Zeitraum wird mit dem Laufzeitband festgelegt



- 2** Die Zielwährung der Analyse wird festgelegt
- 3** Das Liquiditätsszenario wird festgelegt
- 4** Die zu analysierenden Liquiditätsgruppen werden festgelegt
- 5** Die Daten werden aus dem SAP LRM ermittelt
- 6** Die Daten werden der Anwendung zur Verfügung gestellt

**Erweiterungen**

- 4a** Weitere Selektionsmerkmale, zum Beispiel nach Produkttyp oder Organisationseinheit werden festgelegt
- 6a** Die Daten werden in der Anwendung weiterverarbeitet
- 6b** Die Daten werden in der Anwendung visualisiert

## 4.6 Zusammenfassung

# 5 Umsetzungsmöglichkeiten

## 5.1 Einleitung

## 5.2 Rahmenbedingungen

Bis zu diesem Zeitpunkt ist die Arbeit, mit Ausnahme der Vorstellung von Xcelsius in Kapitel 3.3 auf S. 17, allgemein gehalten. Die Spezifikation beschreibt ein allgemeines Konzept für die Einbindung von Daten des SAP LRM zur weiteren Analyse. Für das weitere Vorgehen soll das Konzept an dem konkreten Beispiel von Xcelsius umgesetzt werden. Der folgende Teil behandelt demnach die konkrete Anbindung von Xcelsius an das SAP LRM. Gleichzeitig soll aber bei der Analyse und dem Entwurf die Anforderung 3 an die Übertragbarkeit (A3 auf S. 29) wenn möglich weiterhin beachtet werden.

## 5.3 BusinessObjects Universum

Eine Möglichkeit, externe Daten in Xcelsius zu integrieren, stellt BusinessObjects Universum dar. Dabei handelt es sich um eine Zwischenschicht, die zwischen dem Bereitsteller der Daten und dem Konsumenten geschaltet wird. Ein Universum ist dabei eine semantische Ebene, die es ermöglicht Daten zu konsumieren, ohne die technischen Aspekte der Datenbank kennen zu müssen.<sup>74</sup>

---

<sup>74</sup>vgl. [SAP06, S.48f]

Es können Daten aus relationalen DBMS, die zum Beispiel in normalisierter Form vorliegen, und aus Data-Warehouse-Systeme, die Daten zum Beispiel im Sternschema ablegen, konsumiert werden.<sup>75</sup> Das Universum abstrahiert diese Daten auf Objekte, die dem Nutzer bereitgestellt werden.<sup>76</sup>

Die Einbindung von Universen wird in Xcelsius standardmäßig unterstützt. Es könnte somit im SAP LRM eine oder mehrere Datenbanktabellen erstellt werden, die mit den Ergebnissen der Berechnungskomponente gefüllt werden. Für diese Datenbanktabellen wird ein Universum erstellt, welches dann schließlich an Xcelsius angebunden wird. Ein Überblick ist in Abbildung 9 dargestellt.

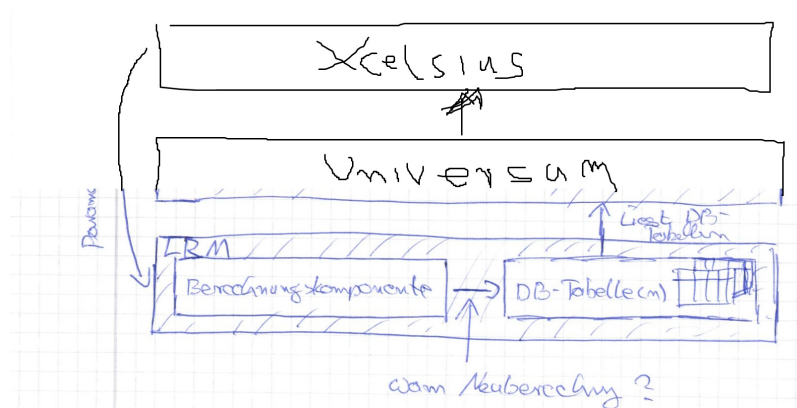


Abbildung 9: Umsetzungsmöglichkeit der Erweiterung mittels BusinessObjects Universum

Da ein Universum allerdings nur lesenden Zugriff auf die Daten bietet, stellt sich die Frage, wie eventuelle Selektionsparameter von Xcelsius an die Berechnungskomponente übergeben werden können. Zusätzlich muss die Berechnung durch die Berechnungskomponente manuell angestoßen werden, um die Tabelle mit den aktuellen Daten zu füllen. Eine automatische, periodische Berechnung, zum Beispiel durch einen eingeplannten Hintergrundprozess, würde stets veraltete Daten liefern.

Schließlich sind Universen nur für die Verwendung mit BusinessObjects Anwendungen ausgelegt. Das Kriterium der Übertragbarkeit in Anforderung 3 (A3 auf S. 29) kann somit nicht garantiert werden.

<sup>75</sup>[TODO]Erklärung normalisiert und Sternschema

<sup>76</sup>vgl. [SAP06, S.49]

## 5.4 Webservice

Eine weitere Möglichkeit stellen Webservices dar. Für die Umsetzung muss im SAP LRM der Webservice implementiert werden. In Xcelsius wird dieser Webservice aufgerufen und somit über den Webservice Daten konsumiert. Ein Überblick stellt Abbildung 10 dar, welche im Folgenden näher erläutert wird.

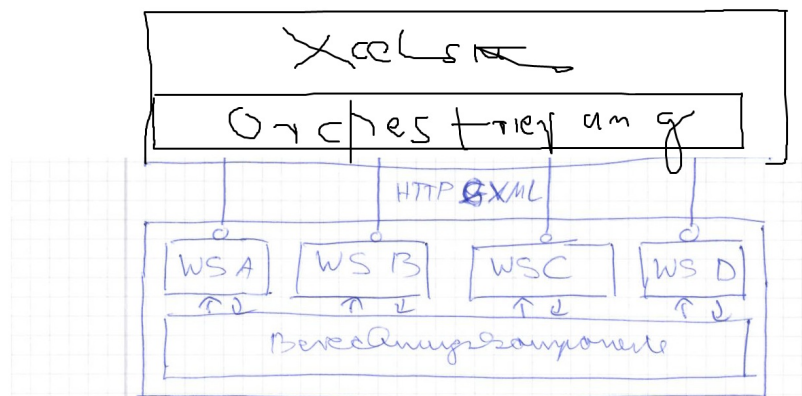


Abbildung 10: Umsetzungsmöglichkeit der Erweiterung mittels Webservices

Dadurch, dass der Aufruf direkt im SAP LRM entgegen genommen wird und nicht indirekt, zum Beispiel über ein Universum erfolgt, können eventuelle Selektionsparameter dynamisch übergeben werden. Auch das Problem mit veralteten Daten besteht nicht, da die Berechnung bei jedem Aufruf nach Bedarf durchgeführt werden kann.

Werden für die Umsetzung der Webservices offene Standards wie zum Beispiel das Hypertext Transfer Protocol (HTTP) für die Übertragung oder die XML als Beschreibungssprache eingesetzt, können die Webservices auch von anderen Anwendungen verwendet werden. Dies ist für das Kriterium der Übertragbarkeit in Anforderung 3 (A3 auf S. 29) wichtig.

Außerdem kommt hinzu, dass bei dieser Umsetzungsmöglichkeit eine gute Modularisierung und Wiederverwendbarkeit erreicht werden kann. Die benötigte Funktionalität kann auf mehrere kleine Webservices aufgeteilt werden. Es entstehen voneinander unabhängige Teile, die leichter gewartet und unabhängig voneinander verwendet

werden können. In Xcelsius müssen diese Teile dann orchestriert werden.

Teile des Backends des SAP LRM wurden mit Hilfe eines Frameworks umgesetzt. Dieses bietet die Möglichkeit, bestimmte Klassen ohne zusätzlichen Programmieraufwand automatisch über einen Webservice bereitzustellen. Dazu zählen unter anderem die Liquiditätsszenarien und die Laufzeitbänder. Somit könnten Teile der Webservices ohne großen Aufwand umgesetzt werden.

Ein mögliches Risiko dieser Umsetzungsmöglichkeit stellt die Anforderung 8 an die Performance (A8 auf S. 29) dar. Der zusätzliche Aufwand für das Aufrufen von mehreren Webservices, in festgelegten Reihenfolgen und Abhängigkeiten, kann zu einer längeren Gesamtlaufzeit führen. Zusätzlich führt der Ausfall eines Teil-Webservice zur Funktionsunfähigkeit der gesamten Erweiterung.

Xcelsius bietet zwar eine Möglichkeit zur Einbindung von einzelnen Webservices, allerdings besteht standardmäßig keine Möglichkeit, mehrere Webservices zusammenzusetzen und zu orchestrieren. Dafür müsste eine eigene Erweiterung auf Basis des Xcelsius SDK entwickelt werden.

## **5.5 Gegenüberstellung und Auswahl der Umsetzungsmöglichkeiten**

Für die Auswahl der im Rahmen der Bachelorarbeit angewendeten Umsetzungsmöglichkeit wurden die Vor- und Nachteile der Alternativen ermittelt und in Tabelle 2 auf S. 37 gegenübergestellt.

Bei der Gegenüberstellung fällt auf, dass die Umsetzung mit Hilfe von BusinessObjects Universe im Vergleich zu der Umsetzung über Webservices deutlich weniger Vorteile bietet. Gleichzeitig bieten die Möglichkeit mit BusinessObjects Universe zusätzlich noch mehr Nachteile. Unter diesen Nachteilen findet sich auch die ungelöste Frage der Parameterübergabe und die fehlende Übertragbarkeit auf andere Anwendungen, welche als wichtige Anforderung definiert ist.

	Vorteile	Nachteile
BO Universe	<ul style="list-style-type: none"> <li>• Standardmäßige Anbindung von Universen</li> </ul>	<ul style="list-style-type: none"> <li>• zusätzliche Infrastruktur notwendig</li> <li>• Übertragbarkeit auf andere Anwendungen</li> <li>• Problem der dynamischen Übergabe von Parametern</li> <li>• Problem der Aktualisierung der berechneten Daten</li> </ul>
Webservices	<ul style="list-style-type: none"> <li>• Übergabe von Selektionsparametern einfach möglich</li> <li>• keine zusätzliche Infrastruktur erforderlich</li> <li>• gute Modularisierung und Übertragbarkeit</li> <li>• teilweise Bereitstellung von Webservices durch Framework</li> </ul>	<ul style="list-style-type: none"> <li>• Komplexe Orchestrierung erforderlich</li> <li>• Performanceprobleme bei Webservice-Aufrufen</li> <li>• Ausfall eines Teil-Webservice führt zum Ausfall des Gesamtsystems</li> </ul>

Tabelle 2: Gegenüberstellung der Vor- und Nachteile der Umsetzungsmöglichkeiten für die Erweiterung

Aus diesen Gründen wurde sich für die weitere Vorgehensweise für die Umsetzung mit Hilfe von Webservices entschieden.

## **5.6 Zusammenfassung**

# 6 Umsetzung

## 6.1 Einleitung

## 6.2 Analyse

### 6.2.1 Überblick

In diesem Abschnitt wird der Analyseprozess der Entwicklung näher betrachtet. Die Ermittlung der Anforderungen an die Anwendung, welches ein Teil der Analysephase ist, wurde in diesem Fall schon in dem allgemeinen Teil der Bachelorarbeit in Kapitel 4.4 auf S. 28 durchgeführt. Die dort ermittelten Anforderungen werden ohne Änderungen übernommen und in dem Folgenden nicht noch einmal näher erläutert.

Die Entwicklung der Erweiterung für Xcelsius kann in fünf Teilbereiche unterteilt werden. Sie werden für die Strukturierung der Analyse genutzt. Bei den Bereichen handelt es sich um folgende Teile:

- 1     Bereitstellung der Webservice im SAP LRM  
      Das SAP LRM System stellt über Webservices alle benötigten Daten bereit.
- 2     Konsumierung der Webservice in Xcelsius  
      Die Webservices werden konsumiert und zunächst in ein internes Objektformat umgewandelt.



- 3     Orchestrierung der Webservice in Xcelsius  
Die Ergebnisse der Konsumierung der Webservices werden miteinander verbunden und in eine interne Zielobjektstruktur umgewandelt.
- 4     Dateiaustausch zwischen der Erweiterung und Xcelsius  
Die Ergebnisse der Orchestrierung müssen in das Tabellenblatt von Xcelsius geschrieben und Selektionsparameter aus dem Tabellenblatt gelesen werden.

### 6.2.2 Grundlegende Architektur und Funktionsweise

In diesem Abschnitt wird ein Überblick (siehe dazu Abbildung 11 auf S. 41) über die grundlegende Architektur der Lösung gegeben. Dabei werden Erweiterungen in zwei Systemen, dem SAP LRM und Xcelsius, umgesetzt. Gleichzeitig wird die grundlegende Funktionsweise, gerade im Hinblick auf die Interaktion mit dem Nutzer, näher gebracht.

Im SAP LRM System muss für jeden Webservice ein Webservice-Provider bereitgestellt werden. Der Webservice-Provider kann über einfache Funktionsaufrufe entweder Berechnungen über die Berechnungskomponente durchführen lassen, oder auf Objekte des SAP LRM, wie zum Beispiel einem Laufzeitband oder eine Liquiditätsgruppe zugreifen. Das Ergebnis liefert er über das HTTP im XML-Format.

Die drei restlichen Teile, die im Überblick eingeführt wurden, werden in einem Plugin für Xcelsius implementiert. Dazu zählt die Konsumierung der bereitgestellten Services und anschließend die Orchestrierung.

Der Xcelsius-Adapter ist über ein Tabellenblatt die Schnittstelle für den Nutzer. Das Tabellenblatt wird dabei in zwei Bereiche unterteilt. Dies sind ein Daten- und ein Einstellungs-Teil. Die Ergebnisse der Erweiterung werden von dem Adapter in den Daten-Teil geschrieben. Damit endet die Zuständigkeit der Erweiterung. Der Nutzer kann die Daten beliebig weiterverarbeiten, in der Regel wird er sie mit Diagrammen visualisieren.

Die Selektionsparameter, die zum Beispiel für die Berechnungskomponente im SAP

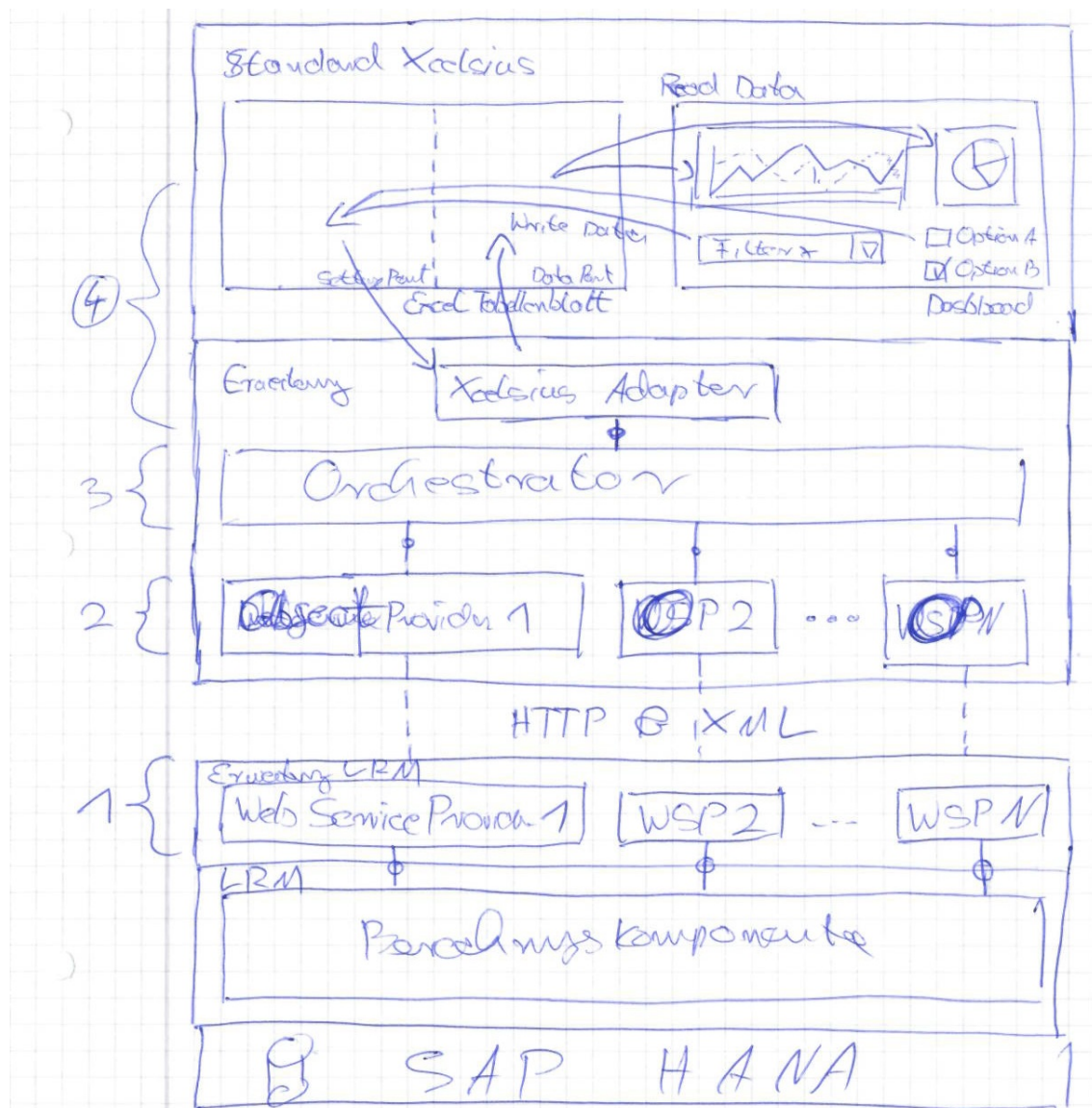


Abbildung 11: Überblick der grundlegenden Architektur der Erweiterung

LRM gebraucht werden (siehe dazu Kapitel 3.2.3 auf S. 16), werden aus dem Einstellungs-Teil des Tabellenblatts ausgelesen. Wie die Daten von dem Nutzen in den Bereich geschrieben werden, ist wiederum komplett dem Nutzer überlassen. Eine Möglichkeit sind Kombinationsfelder, die Xcelsius standardmäßig bereitstellt.

Die Erweiterung stellt nur die Möglichkeit zur Verwendung von Daten aus dem SAP LRM in Xcelsius bereit. In welcher Form die Daten weiter verarbeitet werden, wird

bewusst komplett offengelassen.

### 6.2.3 Statisches Modell

Für diese Bereiche wurde in einem ersten Schritt die grundlegenden Klassen und ihre Assoziationen zueinander identifiziert. Das entsprechende Klassendiagramm der Analysephase ist in Abbildung 12 dargestellt.

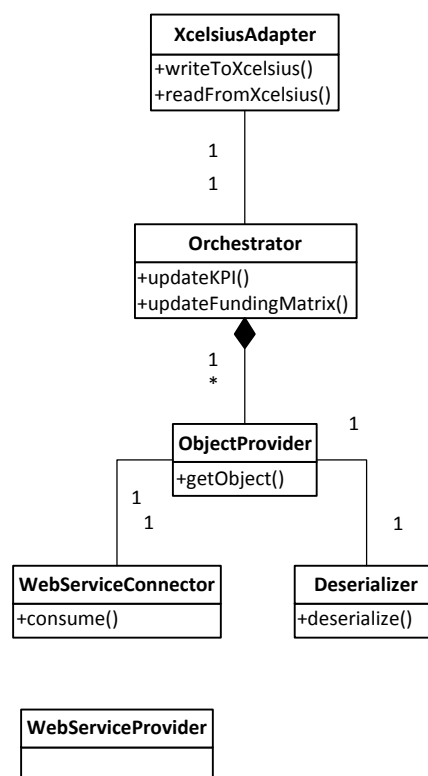


Abbildung 12: Klassendiagramm der Analysephase

Das Kernstück der Erweiterung ist die Klasse *Orchestrator*. Hier ist die komplette Logik der Orchestrierung für die Kennzahlenanalyse und die Finanzierungsmatrix hinterlegt, das heißt die Ergebnisse der einzelnen Webservices werden kombiniert. Dazu existieren die entsprechenden Methoden *updateKPI()* und *updateFundingMatrix()*. Für den Datenaustausch mit dem Tabellenblatt von Xcelsius existiert zwi-

schen der Klasse *Orchestrator* und der Klasse *XcelsiusAdapter* eine 1:1 Assoziation.

Die Klasse *XcelsiusAdapter* stellt die Verbindung zwischen der Erweiterung und dem Tabellenblatt von Xcelsius dar. Mit der Methode *writeToXcelsius()* können Daten in das Tabellenblatt geschrieben werden. Zum Lesen von Daten existiert die Methode *readFromXcelsius()*.

Eine weitere wichtige Klasse stellt der *ObjectProvider* dar. Ein *ObjectProvider* ist eine abstrahierte Schicht über einem Webservice. Die Aufgabe eines *ObjectProvider* ist der einfache Zugriff auf Objekte, die über einen Webservice bezogen werden. Dieser Zugriff findet über die Methode *getObject()* statt. Mehrere *ObjectProvider* sind Teil der Klasse *Orchestrator*, zwischen diesen beiden Klassen existiert eine Komposition.

Um die Webservices abstrahiert darstellen zu können, nutzt ein *ObjectProvider* zwei weiteren Klassen, den *WebServiceConnector* und den *Deserializer*. Hier besteht jeweils eine 1:1 Assoziation. Der *WebServiceConnector* übernimmt die Kommunikation zwischen der Erweiterung und dem SAP LRM System über das HTTP. Der *Deserializer* wandelt das Resultat der Kommunikation über den Webservice in ein internes Objektformat um.

Die Klasse *WebServiceProvider* muss im SAP LRM implementiert werden. Sie nimmt die Anfragen der Erweiterung über das HTTP entgegen, liest die entsprechenden Objekte aus der Datenbank und serialisiert sie im XML-Format.

## 6.2.4 Dynamisches Modell

Um eine bessere Vorstellung von den Vorgängen zu erlangen, wird in dem Folgenden exemplarisch der Ablauf einer Anfrage durch einen Nutzer erläutert. Das entsprechende Sequenzdiagramm ist in Abbildung 13 auf S. 44 dargestellt.

In dem Beispiel möchte der Nutzer den Wert einer Kennzahl abfragen. Dazu ruft er

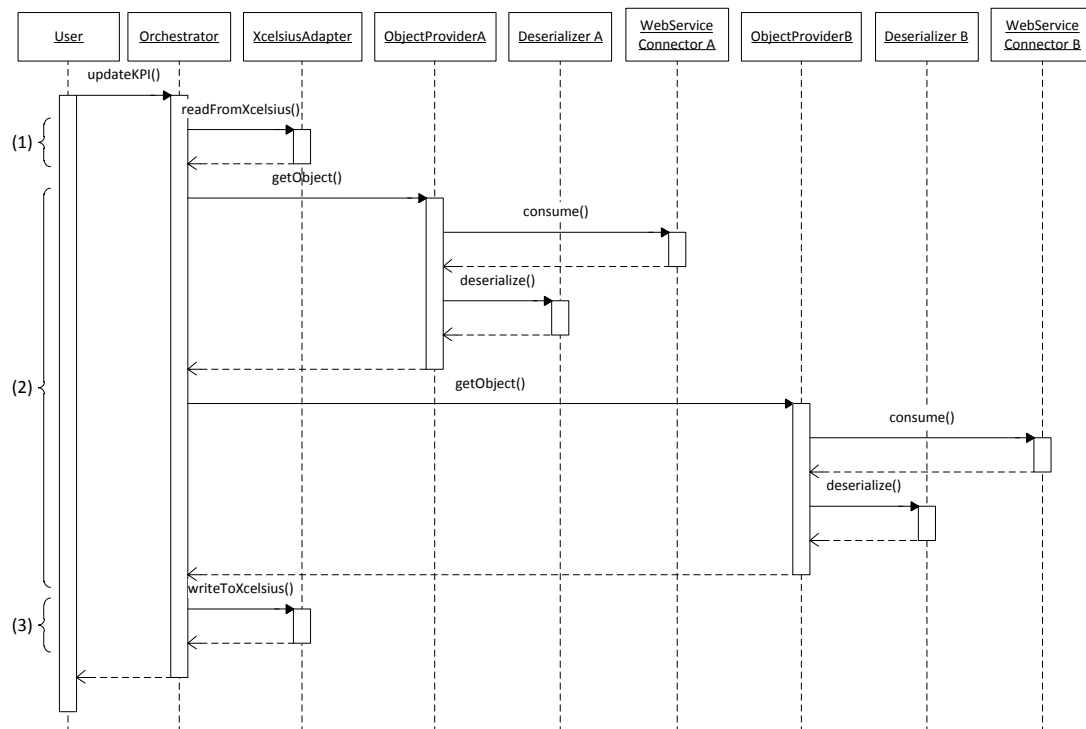


Abbildung 13: Sequenzdiagramm der Analysephase

in der *Orchestrator*-Instanz die Methode *updateKPI()* auf.

Daraufhin werden drei Schritte durchlaufen, die in dem Diagramm auf der linken Seite markiert sind. Zunächst wird über den *XcelsiusAdapter* die aktuellen Selektionseinstellungen des Nutzers ermittelt. Dazu gehört die Kennzahl, die abgerufen werden soll und eventuell weitere Einstellungen.

Anschließend wird in einem zweiten Schritt ein oder mehrere Webservices über die entsprechenden *WebServiceProvider* konsumiert. Die Objekte, die die *WebServiceProvider* zurückliefern werden durch den *Orchestrator* verbunden.

Die *WebServiceProvider* rufen über einen *WebServiceConnector* den XML-Inhalt ab. Anschließend wird das Ergebnis an den entsprechenden *Deserializer* weitergegeben. Dieser wandelt es in eine interne Objektstruktur um und gibt die Instanz an den *ObjectProvider* zurück. Dieser leitet es an den *Orchestrator* weiter.

Der Aufruf der *WebServiceProvider* durch den *Orchestrator* kann asynchron erfolgen. Hat der *Orchestrator* alle Ergebnisse der *ObjectProvider* erhalten, erfolgt die Umwandlung in das Zielobjektformat und schließlich in dem dritten Schritt das Übergeben des Ergebnis an den *XcelsiusAdapter*. Dieser schreibt die Daten in das Tabellenblatt von Xcelsius.

Dadurch stehen die Daten dem Nutzer in Xcelsius zur Verfügung. Der Aufruf von *updateKPI()* ist fertig.

## 6.3 Entwurf

### 6.3.1 Orchestrierung der Webservices

Durch die Verwendung eines ABAP-Frameworks zur Implementierung von wichtigen Klassen im SAP LRM, existieren schon Webservices, die für die Erweiterungen genutzt werden können. Bei den Klassen handelt es sich um:

- Laufzeitband
- Liquiditätsszenario
- Liquiditätsgruppe

Hierfür bietet das Framework eine Schnittstelle, mit der auf die Objekte über das Atom Syndication Format<sup>GL</sup> zugegriffen werden kann. Diese Webservices können in der Erweiterung genutzt werden. Zusätzlich befindet sich eine iPad-App für das SAP LRM in Entwicklung. Für diese wurden zwei weitere Webservices entwickelt, die dem Zugriff auf Kennzahlen und die Finanzierungsmatrix bieten.

Es ergeben sich fünf nutzbare Webservices, die orchestriert werden müssen. Das Ziel ist immer der Zugriff auf den Webservice der Finanzierungsmatrix oder den Webservice für die Kennzahlen. Die Abhängigkeiten der jeweiligen Webservices sind in Abbildung 14 auf S. 46 dargestellt und sollen im Folgenden erläutert werden:

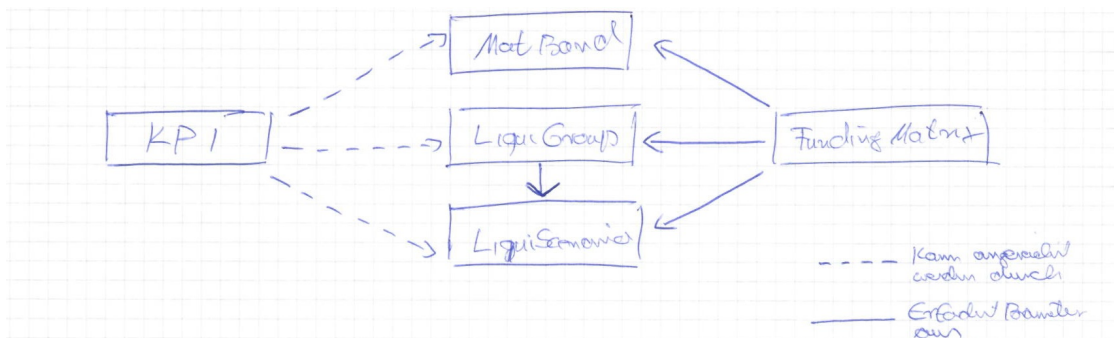


Abbildung 14: Abhängigkeiten der Webservices

Der Webservice der Finanzierungsmatrix ruft bei einem Aufruf die Berechnungskomponente des SAP LRM auf und muss demnach mit den vier erforderlichen Parametern versorgt werden (siehe dazu Kapitel 3.2.3 auf S. 16). Es besteht eine Abhängigkeit zu den drei Webservices, die automatisch durch das Framework bereit stehen. Der Webservice der Liquiditätsgruppen ist so umgesetzt, dass er ein Liquiditätsszenario als Parameter erwartet und darauf alle möglichen Liquiditätsgruppen zurückliefert – auch hier besteht eine Abhängigkeit.

Der Webservice der Kennzahlen erfordert keine Parameter und kann eigenständig aufgerufen werden. Er liefert alle Kennzahlen, die im SAP LRM angelegt und verfügbar sind. Das Resultat beinhaltet allerdings nicht alle Informationen, zum Beispiel zu dem verwendeten Liquiditätsszenario für die Berechnung oder der hinter liegenden Liquiditätsgruppe. Das Ergebnis kann also durch weitere Aufrufe von Webservices angereichert werden.

### 6.3.2 Statisches Modell

Im Gegensatz zu der Analyse wird der Entwurf auf einer höheren Detailstufe durchgeführt und das Resultat der Analyse auch im Hinblick auf die Performance untersucht und optimiert. Das sich daraus ergebene Klassendiagramm ist in Abbildung 15 auf S. 47 zu sehen.

Objekte, wie zum Beispiel ein Liquiditätsszenario oder ein Laufzeitband, werden

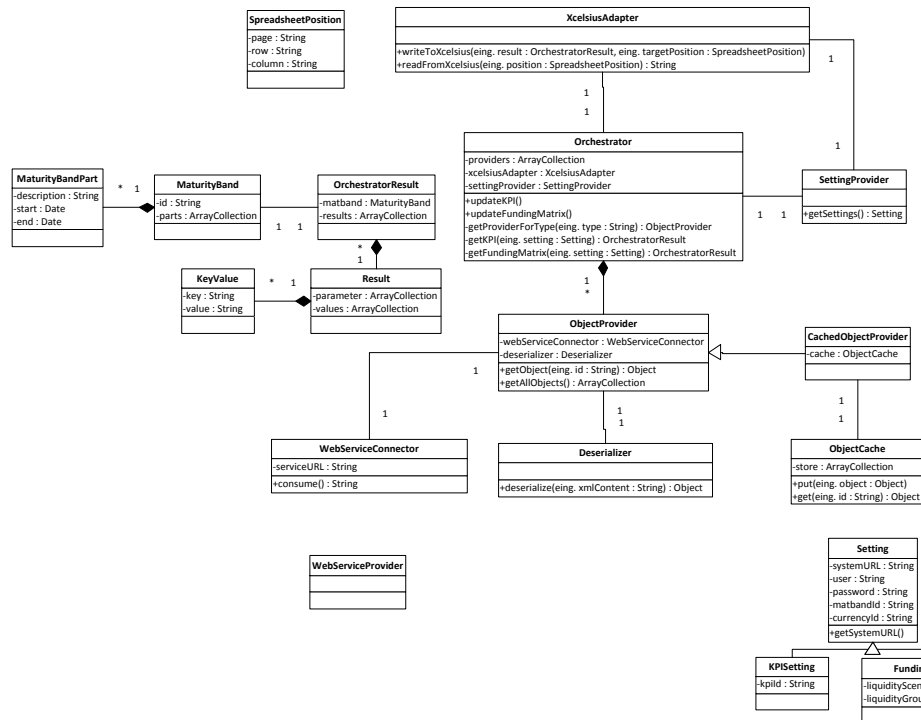


Abbildung 15: Klassendiagramm der Entwurfsphase

nur einmal angelegt und danach in der Regel nicht mehr geändert. Bei der Orchestrierung der Webservices müsste allerdings nach dem Ergebnis der Analyse jedes Mal eine neue Abfrage gestartet werden. Um das zu verhindern wird die Klasse *ObjectProvider* durch die neu hinzugefügte Klasse *CachedObjectProvider* erweitert. Ein *CachedObjectProvider* hält alle Objekte, die schon einmal angefragt wurden, in einem internen Cache und liefert bei einer erneuten Nachfrage das Objekt aus dem Cache zurück, ohne einen neuen Webservice aufrufen zu müssen.

Für das Ermitteln der aktuellen Einstellungen, die der Nutzer in dem Tabellenblatt von Xcelsius hinterlegt hat, wurde die Klasse *SettingProvider* hinzugefügt. Der *SettingProvider* nutzt den *XcelsiusAdapter* zum Auslesen des Tabellenblattes und liefert über die Methode *getSetting()* alle erforderlichen Einstellungen für die Abfragen der Webservices zurück. Dadurch kann der *XcelsiusAdapter* sich rein auf das funktionale Lesen und Schreiben des Tabellenblattes beschränken und von der



Logik, welche Zelle wie ausgelesen werden muss, befreit werden.

Für den einfachen Zugriff auf die einzelnen *ObjectProvider* wurde der Klasse *Orchestrator* die Hilfsmethode *getProviderForType()* hinzugefügt. Die später von dem Nutzer aufgerufenen Methoden *updateKPI()* und *updateFundingMatrix()* nutzen die neuen Hilfsmethoden *getKIP()* und *getFundingMatrix()*. Durch die neue Signatur kann der *XcelsiusAdapter* und der *SettingProvider* einfacher angebunden werden.

Zudem legt mit der Klasse *OrchestratorResult* die Zielobjektstruktur der Orchestrierung fest. Eine Instanz der Klasse *OrchestratorResult* besteht dabei aus beliebig vielen *Result*-Instanzen. Genauer gesagt wird für jede abgefragte Liquiditätsgruppe eine *Result*-Instanz erzeugt.

Eine *Result*-Instanz hat zum einen mehrere Parameter. Diese sind, um eine gute Erweiterbarkeit zu erreichen, in einfachen Key-Value Paaren verwaltet. Beispiele für Parameter können die Liquiditätsgruppe und das genutzte Liquiditätsszenario sein. Zum anderen werden in dem Attribut *values* die einzelnen Werte gespeichert.

Die Werte beziehen sich in ihrer Reihenfolge auf das zugrundeliegende Laufzeitband. Dieses ist für alle *Result*-Instanzen das gleiche und ist deshalb nur einmal als Attribut des *OrchestratorResult* vorhanden. Die einzelnen Abschnitte des Laufzeitbandes sind in dem Attribut *parts* abgelegt. Ein Laufzeitbandabschnitt hat eine Beschreibung, ein Start und ein End-Datum.

Generell wurden alle Klassen um typisierte Attribute erweitert. Den Methoden wurde die Signatur hinzugefügt.

### 6.3.3 Dynamisches Modell

An dem Dynamischen Modell müssen in der Entwurfsphase nur kleinere Änderungen durch die neu hinzugekommenen Klassen durchgeführt werden. Das angepasste Sequenzdiagramm ist in Abbildung 16 auf S. 49 dargestellt.

In Abschnitt (1) ist die Einbindung des *SettingProvider* zu sehen. Außerdem lässt

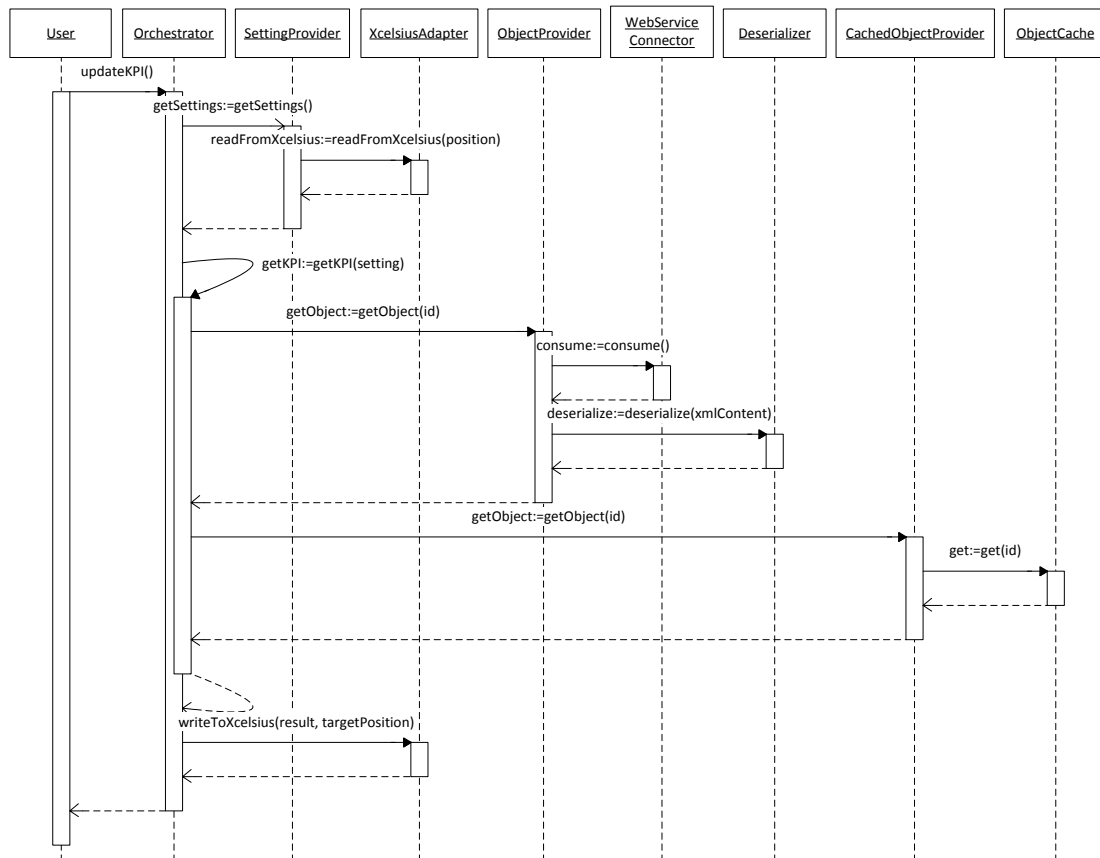


Abbildung 16: Sequenzdiagramm der Entwurfsphase

sich das Zusammenspiel zwischen der öffentlichen, parameterfreien Methode *updateKPI()* und der privaten Methode *getKPI()* sehen. In der Orchestrierung wird nun auch ein *CachedObjectProvider* verwendet. Dieser hatte das angeforderte Objekt schon einmal über seinen Webservice abgefragt und in seinem Cache zwischengespeichert. Dieser Vorgang ist in dem Sequenzdiagramm aus Platzgründen nicht abgebildet. Die erneute Anfrage kann er ohne einen weiteren Aufruf eines Webservices direkt beantworten.

### 6.3.4 Zielstruktur in Xcelsius

Eine zentrale Rolle in der Erweiterung spielt die Übergabe der Ergebnisse der Orchestrierung an das Tabellenblatt von Xcelsius. Dies muss in einer Art und Weise

geschehen, sodass der Nutzer die Daten später einfach weiterverarbeiten oder visualisieren kann.

Diese Aufgabe wird von dem *XcelsiusAdapter* übernommen. Ausgangspunkt ist immer eine *OrchestratorResult*-Instanz, welche das Ergebnis der Orchestrierung darstellt. Den Aufbau der Instanz ist in Kapitel 6.3.2 auf S. 48 dargestellt. Diese Objektstruktur muss in eine Tabellenstruktur in Xcelsius umgewandelt werden. Ein konkretes Beispiel für die Objektstruktur zeigt Abbildung 17, welches in Tabelle 3 auf S. 51 entsprechend umgesetzt ist.

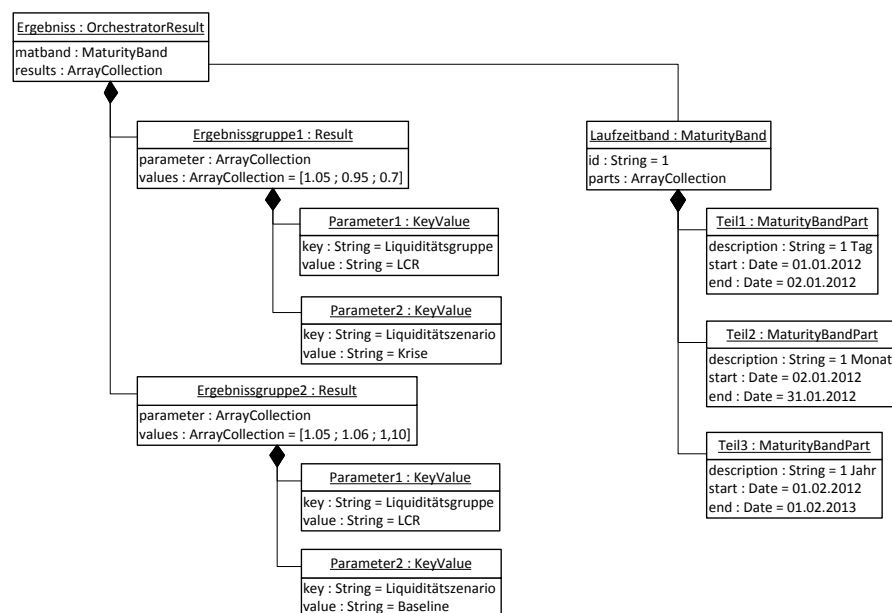


Abbildung 17: Interne Objektstruktur vor der Umwandlung durch den XcelsiusAdapter

Die Tabellenstruktur soll folgendermaßen aufgebaut werden: Eine Ergebnisreihe wird immer in einer Spalte dargestellt. Dabei handelt es sich um alle Ergebniswerte, die für eine Liquiditätsgruppe ermittelt wurden. Die Ergebnisreihen sind in der Objektstruktur die *Ergebnis*-Instanzen. Die Werte sind in der ArrayCollection *values* gespeichert. Sie beziehen sich auf die einzelnen Abschnitte des zugrundeliegenden Laufzeitbandes.

In der ersten Spalte werden die Abschnitte des Laufzeitbandes abgetragen. Diese

<b>Liquiditätsgruppe: Liquiditätszenario:</b>	<b>FLE Krise</b>	<b>FLE Baseline</b>
1 Tag	1,05	1,05
1 Monat	0,95	1,06
1 Jahr	0,70	1,10

Tabelle 3: Ergebnis der Umwandlung durch den XcelsiusAdapter

können zum Beispiel später in einer Visualisierung als X-Achsenbeschriftung verwendet werden. In der Objektstruktur sind das die *Laufzeitbandabschnitt*-Instanzen. In einer Zeile in der Tabellenstruktur sind demnach die Werte aller *Ergebnis*-Instanzen für ein Laufzeitbandabschnitt eingetragen.

Über den eigentlichen Werten werden noch die Parameter eingetragen, die als Key-Value-Paar in der *Ergebnis*-Instanz vorhanden sind. Sie können bei einer Visualisierung als Beschriftung für eine Ergebnissreihe verwendet werden.

So entsteht die Tabellenstruktur in Tabelle 3, welche durch den *XcelsiusAdapter* in das Tabellenblatt geschrieben wird. Dort können die Daten durch den Nutzer weiterverarbeitet werden.

## 6.4 Implementierung

### 6.4.1 Flex Sandbox

Alle Anwendungen, die mit Hilfe von Adobe Flex programmiert sind, unterliegen bestimmten Sicherheitsbeschränkungen. Darunter fällt auch die Erweiterung, die im Rahmen dieser Bachelorarbeit entwickelt wird. Die Ausführung von Flex-Anwendungen erfolgt in einer Sandbox. Dadurch sollen ungewollte Änderungen auf Client-Seite verhindert werden – die Anwendung darf sich nur in einem klar definierten Raum bewegen.

Eine dieser Beschränkungen ist die Verhinderung von Cross-Domain Requests. Dabei

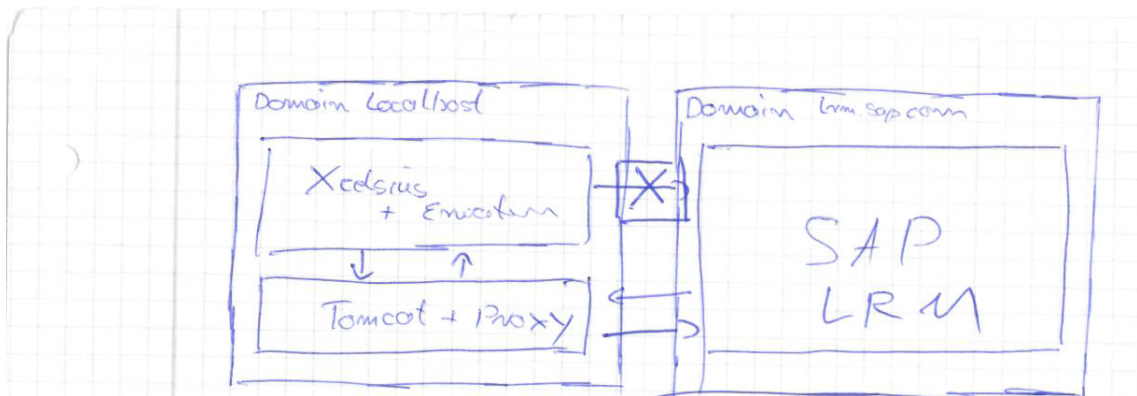


Abbildung 18: [TODO] [TODO Referenzieren!]

handelt es sich um einen Aufruf eines Webservice, dessen Ziel eine andere Domain ist. In dem konkreten Beispiel der Erweiterung für Xcelsius ist die Domain der lokale Computer (localhost). Es ist nur möglich, Webservices aufzurufen, die der Computer selbst bereitstellt. Das SAP LRM selbst läuft nicht auf dem Computer, sondern auf einem Server in einer eigenen Domain. Dadurch ist das Konsumieren von Daten aus dem SAP LRM erstmal nicht möglich, da es durch die Sandbox blockiert wird.

Für dieses Problem kommen zwei Lösungsansätze in Frage. Zum einen ist es Möglich, dass der Anbieter des Webservice explizit festlegen kann, dass der Webservice auch über verschiedene Domains hinweg konsumiert werden kann. Dies geschieht über eine Datei crossdomain.xml, welche in dem Hauptverzeichnis der Domain abgelegt werden muss. Bei der Erweiterung wäre das der Anwendungsserver des SAP LRM. Änderungen an diesem zu veranlassen würde einen hohen Aufwand benötigen.

Um diesen Aufwand zu umgehen, wurde der zweite Lösungsansatz umgesetzt. Hierbei wird auf dem Client ein lokaler Webserver installiert. Dieser Webserver ist über die Domain localhost erreichbar. Alle Anfragen der Erweiterung gehen gegen den lokalen Webserver. Dadurch, dass die Flex-Anwendung auch unter der Domain localhost ausgeführt wird, ist der Zugriff erlaubt. Der Webserver stellt die Anfrage an den richtigen Webservice im SAP LRM und leitet die Antwort an die Erweiterung weiter.

Für die Umsetzung wurde lokal ein Tomcat<sup>GL</sup>-Webserver installiert. Für diesen wur-

de ein einfaches Java-HttpServlet entwickelt, welches die geforderte Funktionalität umsetzt. Das Coding des Servlets ist in Listing 4 auf S. IX zu sehen.

Für die Zukunft sollte allerdings versucht werden, die Lösung über die `crossdomain.xml`-Datei umzusetzen. Dadurch entfällt der zusätzlich notwendige lokale Webserver.

### 6.4.2 Asynchroner Programmablauf

Im Vergleich zu dem Entwurf und der Analyse, bei dem alle Aufrufe synchron durchgeführt wurden, wird dies aus zwei Gründen in der Umsetzung auf hauptsächlich asynchrone Aufrufe umgestellt. Dabei handelt es sich um folgende Gründe:

Auf der einen Seite kann sich dadurch ein Geschwindigkeitsvorteil ergeben. Falls sich bei der Orchestrierung die Antwort eines Aufruf eines Webservice verzögert, können trotzdem alle anderen Anfragen schon verarbeitet werden. Der dadurch gewonnene Vorteil ist aufgrund technischer Limitierungen leider nicht allzu groß, da die Anwendung später nur auf einem Thread ausgeführt wird und somit eine echt parallele Verarbeitung von zwei simultan erhaltenen Antworten nicht möglich ist.

Auf der anderen Seite wird für den Aufruf der Webservices in der Klasse *WebServiceConnector* die Flex-Klasse *HTTPService* verwendet. Diese bietet nur die Möglichkeit über `EventListener` auf zwei Events, nämlich dem erfolgreichen Aufruf des Webservice und dem fehlgeschlagen Aufruf zu reagieren. Dabei werden zwei Callback-Methoden angegeben, die dann asynchron aufgerufen werden. Ab diesem Zeitpunkt endet somit der bis dahin ausschließlich synchrone Aufruf. Das entsprechende Coding ist in Listing 1 auf S. 54 dargestellt.

Durch die Asynchronität ändert sich entsprechend die Signatur der Methoden. Der Rückgabewert entfällt, stattdessen wird ein zusätzlicher Parameter mitgegeben. Bei `ActionScript` kann dies entweder eine Methode eines Objektes sein, siehe dazu Listing 1 auf S. 54. Alternativ kann auch eine Objekt-Instanz übergeben werden. Auf dieser Instanz wird dann zu dem entsprechenden Zeitpunkt eine festgelegte Methode aufgerufen.

```
public function consume():void{
    // HTTPService-Instanz erstellen
    webService = new HTTPService();
    webService.url = serviceURL;
    webService.method = "GET";
    // Event Listener hinzufügen
    webService.addEventListener(ResultEvent.RESULT, onHttpRequestSuccess);
    webService.addEventListener(FaultEvent.FAULT, onHttpRequestFailure);
    // Anfrage senden
    webService.send();
}

public function onHttpRequestSuccess(event:ResultEvent){
    // Asynchroner Aufruf bei Erfolg
    // ...
}

public function onHttpRequestFailure(event:FaultEvent){
    //Asynchroner Aufruf bei Fehlschlag
    // ...
}
```

Listing 1: Aufruf eines Webservice in der Erweiterung

Anfang und Ende eines Aufrufes von einem Webservice über den *ObjectProvider* und den *Deserializer* ist der *Orchestrator*. Dieser startet die Aufrufe in der Methode *getKPI()* oder *getFundingMatrix()* und erhält die Antworten durch die Asynchronität in beliebiger Reihenfolge in der Methode *onObjectProviderSuccess()*. Um die Aufrufe unterscheiden und identifizieren zu können, wird bei dem Aufruf ein *RequestToken* mitgegeben. Dieser wird weitergereicht und schließlich auch der Methode *onObjectProviderSuccess()* als Parameter zurückgegeben.

Um festzustellen, ob alle Anfragen erfolgreich zurückgekommen sind und somit mit der finalen Orchestrierung begonnen werden kann, werden die *RequestToken* in einer internen Liste verwaltet. Diese Liste enthält alle Anfragen, von denen die Antwort noch aussteht. Wird die Antwort erhalten, wird der Token aus der Liste entfernt, in einen *ResultToken* umgewandelt, indem das eigentliche Ergebnis angehängt wird, und in einer weiteren Liste zwischengespeichert. Wurden alle Anfragen erfolgreich beantwortet, kann die Orchestrierung durchgeführt werden. Die einzelnen Ergebnisse können dann aus der Liste mit den *ResultToken*-Instanzen bezogen werden.

Die Methode *onObjectProviderSuccess()* wird asynchron aufgerufen und verändert unter anderem die beiden Listen zur Verwaltung der Token. Demnach handelt es sich hierbei normalerweise um einen kritischen Abschnitt, in dem kritische Daten

verändert werden. Der Zugriff auf diese Methode müsste synchronisiert werden. Da aber die Anwendung später auf nur einem Thread ausgeführt wird, kann auf eine Synchronisation verzichtet werden.

In Listing 2 ist in allgemeiner Form der erläuterte Ablauf implementiert. Dabei werden zwei beliebige *ObjectProvider* abgefragt. Die jeweiligen Ergebnisse können dann in gewünschter Weise orchestriert werden. Die in dem Beispiel verwendeten Klassen wurden teilweise vereinfacht und entsprechen nicht den Klassen in dem statischen Modell der Erweiterung.

```
private function orchestrateWebServices():void{
    // Verwaltung der Token
    pendingRequestToken = new ArrayCollection();
    finishedRequestToken = new ArrayCollection();

    // Beispielhafte Anfrage an ein ObjectProvider
    var objectProviderA:ObjectProvider = new ObjectProvider();
    var token:RequestToken = new RequestToken(RequestToken.REQUEST_A);
    pendingRequestToken.addItem(token);
    objectProviderA.getObject(token, this);

    // beliebige weitere Anfragen an ObjectProvider ...
}
// Callback-Methode fuer die ObjectProvider
public function onObjectProviderSuccess(result:Object, token:RequestToken):void{
    // Ergebnis der Anfrage vermerken
    pendingRequestToken.removeItemAt(pendingRequestToken.getItemIndex(token));
    finishedRequestToken.addItem(new ResultToken(token, result));

    if(pendingRequestToken.length == 0){ // alle Ergebnisse liegen vor
        // Zugriff auf Ergebnisse
        var ergA:Object = getResultForRequest(RequestToken.REQUEST_A);
        // Ergebnisse Orchestrieren ...
    }
}
// Hilfsmethode fuer den Zugriff auf ein Resultat
private function getResultForRequest(request:String):Object{
    for each(var token:ResultToken in finishedRequestToken){
        if(token.attribute == request)
            return token.result;
    }
    throw "no result";
}
```

Listing 2: Beispielhafter Ablauf der Orchestrierung von Webservices



### 6.4.3 WebserviceAblauf

In diesem Abschnitt soll an dem Beispiel des Finanzierungsmatrix-Webservice exemplarisch gezeigt werden, wie der Aufruf und die Umwandlung in ein internes Objektformat realisiert wird.

Der Ausgangspunkt ist dazu der *Orchestrator*, der die *getObject*-Methode des *FundingMatrixObjectProviders* aufruft. Zunächst wird hier die passende URL für den Webservice-Aufruf auf Grundlage der Einstellungen generiert. In diesem Fall werden über die URL die Parameter Liquiditätsgruppen, Liquiditätsszenarien, Laufzeitband und Zielwährung übergeben. Es wird eine *WebserviceConnector*-Instanz erzeugt und über die *consume()*-Methode des *WebserviceConnectors* der eigentliche Aufruf gestartet. Entsprechend dem asynchronen Programmablauf, siehe dazu Kapitel 6.4.2 auf S. 53, wird die eigene Instanz als Callback und der *RequestToken* mitgegeben.

Tritt bei dem Aufruf ein Fehler auf, wird asynchron die Methode *onWebServiceFail()* aufgerufen. Der Fehler muss nur dem *Orchestrator* weitergeleitet werden. Dazu wurde auch hier bei dem *getObject*-Aufruf eine Referenz auf den *Orchestrator* als Callback mitgegeben.

Wenn kein Fehler auftritt, wird statt der Methode zur Behandlung des Fehlers die *onWebServiceSuccess()*-Methode aufgerufen. Der Parameter *ResultEvent* enthält die Antwort des Webservices als Zeichenkette. Diese muss nun in die interne Objektstruktur umgewandelt werden. Dazu wird der *FundingMatrixObjectDeserializer* verwendet. Die XML-Eingabe für den *Deserializer* und die daraus generierte Objektstruktur ist in Abbildung 19 auf S. 57 dargestellt.

Bei der Deserialisierung kann wiederum auf Hilfsklassen von Flex zugegriffen werden. Über die Klasse *SimpleXMLDecoder* kann einfach ein Proxy-Objekt erzeugt werden. Dadurch kann auf die einzelnen Elemente in der XML-Struktur wie auf Attribute eines Objektes zugegriffen werden. Der Quellcode, der Umwandlung, wie sie in Abbildung 19 auf S. 57 dargestellt ist, zeigt, findet sich in Listing 3 auf S. 58.

Die *ObjectProvider* und *Deserializer* für die weiteren Webservices sind analog dazu

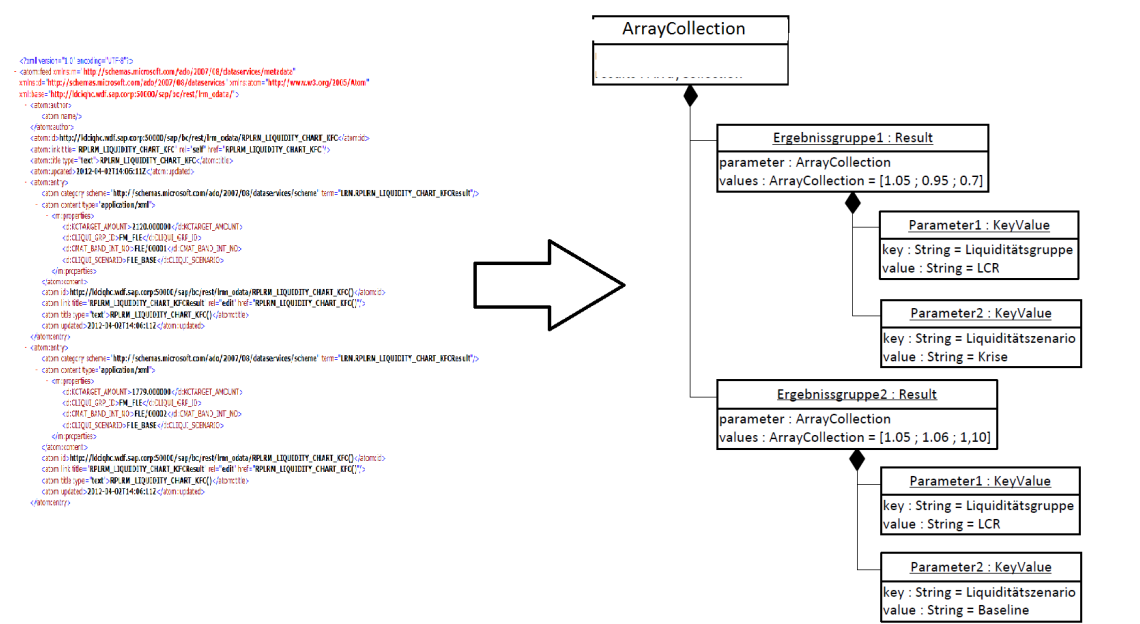


Abbildung 19: Umwandlung der XML-Antwort in die interne Objektstruktur

programmiert. Bei einem *CachedObjectProvider* wird in der *getObject*-Methode zunächst ermittelt, ob sich das angeforderte Objekt schon im Cache befindet und dann direkt zurückgegeben. Ist es noch nicht vorhanden, wird der Webservice-Aufruf gestartet und nach der Deserialisierung wird das Objekt im Cache für weitere Anfragen gehalten.

#### 6.4.4 Xcelsius Binding

Für den Datenaustausch zwischen der entwickelten Erweiterung und dem Tabellenblatt von Xcelsius muss auf die bereitgestellten Funktionen des Xcelsius SDK zurückgegriffen werden. Dabei wird das Konzept des Bindings genutzt.

Bei diesem Konzept wird ein Attribut einer Instanz fest mit einem Zellenbereich auf dem Tabellenblatt verbunden. Dabei kann es sich entweder um ein flaches Attribut, wie zum Beispiel eine Zeichenkette oder eine Zahl, handeln, die dann mit einer einzelnen Zelle auf dem Tabellenblatt verbunden wird. Es kann allerdings auch auf Tabellenblatt-Seite ein Bereich festgelegt werden, der dann zum Beispiel mit einem

```

public override function deserialize(content:String):Object{
    var decoder:SimpleXMLDecoder = new SimpleXMLDecoder();
    var proxyObject:Object = decoder.decodeXML(new XMLDocument(content));

    var result:ArrayCollection = new ArrayCollection();
    var resultGroup:Result = null;

    for each(var entry:Object in proxyObject.feed.entry as Array){
        var liquiGroup:String = entry.content.properties.CLIQUI_GRP_ID as
            String;
        var liquiScenario:String = entry.content.properties.CLIQUI_SCENARIO
            as String;
        if(resultGroup == null || resultGroup.getParameter(Result.
            LIQUIDITY_GROUP) != liquiGroup || resultGroup.getParameter(
            Result.LIQUIDITY_SCENARIO) != liquiScenario){
            if(resultGroup != null) // dont add null at first time
                result.addItem(resultGroup);
            resultGroup = new Result();
            resultGroup.addParameter(new KeyValue(Result.
                LIQUIDITY_GROUP,liquiGroup));
            resultGroup.addParameter(new KeyValue(Result.
                LIQUIDITY_SCENARIO,liquiScenario));
        }
        var value:Number = entry.content.properties.KCTARGET_AMOUNT as
            Number;
        resultGroup.addValue(value);
    }
    result.addItem(resultGroup);
    return result;
}

```

Listing 3: Deserialisierung des XML-Format in die interne Objektstruktur

zweidimensionalen Array verbunden wird.

Für solch eine Verbindung muss zusätzlich noch eine Richtung festgelegt werden. Das Xcelsius SDK bietet dabei drei Möglichkeiten an:

### 1. Input-Binding

Bei einem Input-Binding findet der Datenfluss nur von dem Tabellenblatt zu der Erweiterung statt. Er dient damit ausschließlich dafür, auf das Tabellenblatt lesend zuzugreifen. Sobald sich der Wert im Tabellenblatt ändert, wird diese Änderung auf das Attribut übertragen.

### 2. Output-Binding

Das Output-Binding bietet ausschließlich schreibenden Zugriff auf das Tabellenblatt. Wird also der Attributswert verändert, wird die dazugehörige Zelle oder Zellbereich entsprechend angepasst. Der Datenfluss findet nur von der Erweiterung zu dem Tabellenblatt statt.

### 3. **Input-Output-Binding**

Bei dieser Verbindungsart findet zwischen dem Tabellenblatt ein beidseitiger Austausch statt. Von beiden Seiten her werden Veränderungen auf die jeweils andere Seite übertragen. Der Wert der Zelle wird mit dem Attributswert synchronisiert.

Für die Umsetzung der Erweiterung spielen Input-Output-Bindings keine Rolle, es werden nur Input- oder Output-Bindings verwendet. Das Auslesen der gewählten Parameter findet über mehrere Input-Bindings statt. Für das Laufzeitband wird dabei eine Zelle mit dem Attribut *selectedMaturityBand* des *XcelsiusAdapter* verknüpft. Bei den Liquiditätszenarien und den Liquiditätsgruppen können jeweils mehrere Ausgewählt werden. Hier bezieht sich das Binding demnach nicht auf eine einzelne Zelle, sondern auf einen Zellbereich.

## 6.5 Zusammenfassung

# **7 Evaluation**

## **7.1 Einleitung**

## **7.2 Erfüllung der Anforderungen**

## **7.3 Möglichkeiten**

## **7.4 Vergleich**

## **7.5 Performance**

## **7.6 Zusammenfassung**

## **8 Zusammenfassung**

# A Anhang

```

import java.io.*;
import java.net.*;
import javax.servlet.http.*;
import javax.servlet.*;

@SuppressWarnings("serial")
public class ProxyServlet extends HttpServlet {

    public void doGet (HttpServletRequest req, HttpServletResponse res) {
        // build request url
        String requestedUrl = req.getServletPath();
        String params = req.getQueryString();
        String urlString = "http://ldciqhc.wdf.sap.corp:5000";
        if(params != null)
            urlString += "?" + params;

        // build connection
        URL url = new URL(urlString);
        HttpURLConnection conn
= (HttpURLConnection) url.openConnection();

        // set password
        String encoding = new sun.misc.BASE64Encoder().encode("ldciqhc");
        conn.setRequestProperty ("Authorization", "Basic " + encoding);

        // start request
        conn.setRequestMethod("GET");
        conn.connect();

        // get response
        PrintWriter out = res.getWriter();
        BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream()));
        String content = null;
        while((content = reader.readLine()) != null){
            out.print(content);
        }
        out.flush();
        out.close();
    }
}

```

Listing 4: Proxy-Servlet zur Umgehung der Cross-Domin Request Beschränkungen



# Glossar

## **.NET Framework**

Das .NET Framework ist eine Plattform von Microsoft, mit der Anwendungen für das Betriebssystem Microsoft Windows erstellt und ausgeführt werden können. Die wichtigsten Komponenten sind Klassenbibliotheken, z.B. für die Entwicklung der Oberflächen, und die Common Language Runtime. Die Anwendungen können in verschiedenen Programmiersprachen geschrieben werden. Zu den unterstützten Sprachen zählen unter Anderem C++ und C#. Der Quellcode wird dann in die programmiersprachenunabhängige Common Intermediate Language compiliert. Diese Zwischensprache kann dann von der Common Language Runtime ausgeführt werden.<sup>77</sup>

## **ActionScript**

...

## **Atom Syndication Format**

...

## **Bankenpanik**

Eine Bankenpanik ist ein Ereignis, bei dem eine große Anzahl von Anlegern versucht, ihre Einlagen bei einer Bank abzuziehen. Der Grund kann zum Einen in der Veröffentlichung von schlechten Ergebnissen der Bank und damit einem Vertrauensverlust begründet sein, zum Anderen aber auch rein spekulativ sein. Für die Bank besteht die Gefahr der Insolvenz. Im Englischen spricht man von einem Bank Run.<sup>78</sup>

## **Interbankenhandel**

Interbankenhandel ist der Handel von Wertpapieren, Anlagen oder ähnlichem zwischen Banken. Synonym wird auch oft der Begriff Interbankenmarkt verwendet. Für die Zinssätze, mit denen Banken untereinander handeln, existieren anerkannte Referenzen, wie z.B. der LIBOR. (London Inter Bank Offered

---

<sup>77</sup>vgl. [Lou10, S.382ff]

<sup>78</sup>vgl. [Sch11, S.1f]

Rate). In Liquiditätsengpässen kann der Interbankenhandel eine wichtige Refinanzierungsrolle darstellen. Der Handel zwischen Banken hängt sehr stark von dem gegenseitigen Vertrauen ab.<sup>79</sup>

**Liquidity Coverage Ratio (LCR)**

...

**Net Stable Funding Ratio (NSFR)**

...

**REST**

...

**Rich Internet Application (RIA)**

Unter dem Begriff Rich Internet Application werden Webanwendungen bezeichnet, die in ihrer Funktionalität und ihrem Aussehen Desktopanwendungen ähneln. Erstmals eingeführt wurde der Begriff von Macromedia. Zwischen normalen Webanwendungen und RIA kann keine klare Grenze gezogen werden. Wichtiges Indiz für eine RIA ist der Einsatz von Technologien wie z.B. Adobe Flash, Adobe Air oder Microsoft Silverlight<sup>80</sup>

**SAP Business ByDesign**

SAP Business ByDesign ist eine Anwendung von SAP für mittelständige Unternehmen. Zu dem Funktionsumfang gehört sowohl ein ERP- als auch eine Customer Relationship Management (CRM)-Lösung. Die Besonderheit von SAP Business ByDesign ist, dass es auf Servern bei SAP betrieben wird und Kunden die Anwendung mieten und über das Internet konsumieren.

**SDK**

...

**SQLScript**

SQLScript ist eine Erweiterung der Abfragesprache SQL und wird in der Datenbank von SAP HANA verwendet. Mit Hilfe von SQLScript lässt sich Anwendungslogik in die Datenbank auslagern. Dazu wurden unter Anderem Datentypen, Prozeduren und Kontrollstrukturen hinzugefügt.<sup>81</sup>

---

<sup>79</sup>vgl. [Wil10, S.145f]

<sup>80</sup>vgl. [DD08, S.32f] und [Pfe09, S.3f]

Adobe Flash - <http://www.adobe.com/products/flashplayer.html>

Adobe Air - <http://www.adobe.com/products/air.html>

Microsoft Silverlight - <http://www.microsoft.com/silverlight/>

<sup>81</sup>vgl. [SAP11, S.9f]

## **Tomcat**

...

# Literaturverzeichnis

- [Alb10] ANJA ALBERT: *Bankenaufsichtliche Regulierung des Liquiditätsrisikomanagements*. In: STEFAN ZERANSKI (Herausgeber): *Ertragsorientiertes Liquiditätsrisikomanagement*, Seiten 85–199. Finanz-Colloquium, Heidelberg, 2010. ISBN: 978-3-936974-99-7.
- [Bar08] PETER BARTETZKY: *Liquiditätsrisikomanagement: Status quo*. In: PETER BARTETZKY, WALTER GRUBER und WEHN CARSTEN (Herausgeber): *Handbuch Liquiditätsrisiko*, Seiten 1–27. Schäffer-Poeschel, Stuttgart, 2008. ISBN: 978-3-7910-2747-0.
- [BMH10] STEFAN BATZDORF, BJÖRN MIELENHAUSEN und THEA HILLENBRAND: *NGAP 2.0 at a Glance*, 2010.
- [DD08] PAUL DEITEL und HARVEY DEITEL: *Ajax, rich Internet applications, and web development for programmers*. Prentice Hall, Upper Saddle River and NJ, 2008. ISBN: 978-0131587380.
- [Die10] THOMAS DIETZ: *Liquiditätsrisikomanagement in Banken und die Finanzkrise aus Sicht der Bankenaufsicht*. In: STEFAN ZERANSKI (Herausgeber): *Ertragsorientiertes Liquiditätsrisikomanagement*, Seiten 7–81. Finanz-Colloquium, Heidelberg, 2010. ISBN: 978-3-936974-99-7.
- [Dür11] DÜRRNAGEL: *Management des Liquiditätsrisikos in Banken: Analyse und Beurteilung der Methoden zur Liquiditätsrisikomessung unter Berücksichtigung bankaufsichtlicher Richtlinien*. Diplomica Verlag, Hamburg, 2011. ISBN: 978-3-8428-6186-2.
- [Egg09] NORBERT EGGER: *Reporting und Analyse mit SAP BusinessObjects*. Galileo Press, Bonn, 2009. ISBN: 978-3-8362-1380-6.
- [Fre12] PHILIPP FREUDENBERGER: *Liquidity Risk Management 1.0: powered by SAP HANA*, 2012.

- [GR10] WERNER GLEISSNER und FRANK ROMEIKE: *Risikoblindheit und Methodikschwächen im Risikomanagement*. In: FRANK ROMEIKE (Herausgeber): *Die Bankenkrise*, Seiten 59–88. Bank-Verlag Medien, Köln, 2010. ISBN: 978-3-86556-230-2.
- [Har10] MARTIN HARTIG: *NGAP 1.0 at a Glance*, 2010.
- [HSG08] HENNIG HEUTER, CHRISTIAN SCHÄFFER und WALTER GRUBER: *Einbettung der Liquiditätssteuerung in die Gesamtbanksteuerung*. In: PETER BARTETZKY, WALTER GRUBER und WEHN CARSTEN (Herausgeber): *Handbuch Liquiditätsrisiko*, Seiten 193–229. Schäffer-Poeschel, Stuttgart, 2008. ISBN: 978-3-7910-2747-0.
- [Hue04] FRANK HUELMANN: *Baseler Eigenkapitalvereinbarung: Basel I/II*. Books on Demand GmbH, Norderstedt, 2004. ISBN: 978-3833403989.
- [Hul10] JOHN HULL: *Risikomanagement in Banken und Finanzinstituten*. Pearson Studium, München, 2. Auflage, 2010. ISBN: 978-3-86894-043-5.
- [KHZC09] RAINER KERTH, TIM HUANG, GUO ZHU ZHENG und JESSE CALDERON: *Xcelsius: High Level Architecture*, 2009.
- [Kle10] WOLFRAM KLEIS: *SAP In-Memory Computing Engine: SAP Architecture Bluebook*. Walldorf, 2010.
- [Lou10] DIRK LOUIS: *Visual C++ 2010: Das umfassende Handbuch für Programmierer*. Addison-Wesley, München, 2010. ISBN: 978-3827329011.
- [Mac08] DONALD MACCORMICK: *An introduction to the power of the Xcelsius SDK*, 2008.
- [Moc07] NILS MOCH: *Liquiditätsrisikomanagement in Kreditinstituten: Eine kritische Analyse des Status quo in kleineren Kreditinstituten unter Berücksichtigung regulatorischer und betriebswirtschaftlicher Anforderungen*. Eul, Lohmar and and Köln, 2007. ISBN: 3899366352.
- [Pau11a] STEPHAN PAUL: *Qualitative Bankenaufsicht in der Marktwirtschaft: Theoretische Einordnung und empirische Befunde*. In: GERHARD HOFMANN (Herausgeber): *Basel III und MaRisk*, Seiten 455–485. Frankfurt School Verlag, Frankfurt am Main, 2011. ISBN: 978-3-940913-23-4.

- [Pau11b] STEPHAN PAUL: *Umbruch der Bankenregulierung: Die Entwicklung des Basler Regelwerks im Überblick*. In: GERHARD HOFMANN (Herausgeber): *Basel III und MaRisk*, Seiten 9–63. Frankfurt School Verlag, Frankfurt am Main, 2011. ISBN: 978-3-940913-23-4.
- [Pfe09] CHRISTIAN PFEIL: *Adobe AIR: RIAs für den Desktop entwickeln*. Addison-Wesley, München, 2009. ISBN: 978-3827327376.
- [Poh08] MICHAEL POHL: *Das Liquiditätsrisiko in Banken: Ansätze zur Messung und ertragsorientierten Steuerung*. Knapp, Frankfurt am Main, 2008. ISBN: 978-3831408283.
- [PZ11] HASSO PLATTNER und ALEXANDER ZEIER: *In-memory data management: An inflection point for enterprise applications*. Springer, Heidelberg, 2011. ISBN: 978-3-642-19363-7.
- [RM08] STEFAN REHSMANN und MARCUS MARTIN: *Neuerungen in der aufsichtsrechtlichen Behandlung des Liquiditätsrisikos*. In: PETER BARTETZKY, WALTER GRUBER und WEHN CARSTEN (Herausgeber): *Handbuch Liquiditätsrisiko*, Seiten 51–75. Schäffer-Poeschel, Stuttgart, 2008. ISBN: 978-3-7910-2747-0.
- [Rom10] FRANK ROMEIKE: *Chronologie der Subprime-Krise*. In: FRANK ROMEIKE (Herausgeber): *Die Bankenkrise*, Seiten 13–57. Bank-Verlag Medien, Köln, 2010. ISBN: 978-3-86556-230-2.
- [SAP06] SAP AG: *BusinessObjects: Universumserstellung*, 2006.
- [SAP11] SAP AG: *SAP HANA Database: SQLScript Guide*, 2011.
- [SAP12a] SAP AG: *Next-Generation ABAP Platform: Product Information Sheet*. 2012.
- [SAP12b] SAP AG: *Oberon: Internal Wiki*, 2012.
- [SAP12c] SAP AG: *SAP HANA Overview*, 2012.
- [Sch08] STEPHAN SCHÖNIG: *Liquiditätsrisikomanagement in Kreditinstituten vor dem Hintergrund geänderter aufsichtlicher Anforderungen*. In: OLIVER EVERLING (Herausgeber): *Bankrisikomanagement*, Seiten 232–249. Gabler, Wiesbaden, 2008. ISBN: 978-3-8349-0512-3.

- [Sch11] TIMO SCHRAND: *Die Finanzmarktkrise- Bank-Run und Regulierung des Bankensystems*. GRIN Verlag GmbH, München, 2011. ISBN: 978-3640920037.
- [SLK08] HENNER SCHIERENBECK, MICHAEL LISTER und STEFAN KIRMSSE: *Risiko-controlling und integrierte Rendite-, Risikosteuerung*. Gabler, Wiesbaden, 9. Auflage, 2008. ISBN: 978-3834904478.
- [SS08] DIRK SCHRÖTER und OLIVER SCHWARZ: *Optimale Strukturen und Prozesse für das Liquiditätsrisikomanagement*. In: PETER BARTETZKY, WALTER GRUBER und WEHN CARSTEN (Herausgeber): *Handbuch Liquiditätsrisiko*, Seiten 247–278. Schäffer-Poeschel, Stuttgart, 2008. ISBN: 978-3-7910-2747-0.
- [STW08] PETER SAUERBIER, HOLGER THOMAE und CARSTEN WEHN: *Praktische Aspekte der Abbildung von Finanzprodukten im Rahmen des Liquiditätsrisikos*. In: PETER BARTETZKY, WALTER GRUBER und WEHN CARSTEN (Herausgeber): *Handbuch Liquiditätsrisiko*. Schäffer-Poeschel, Stuttgart, 2008. ISBN: 978-3-7910-2747-0.
- [Wil10] LOTHAR WILDMANN: *Makroökonomie, Geld und Währung*. Oldenbourg, München, 2. Auflage, 2010. ISBN: 978-3486702408.
- [Zer05] STEFAN ZERANSKI: *Liquidity at risk zur Steuerung des liquiditätsmässig-finanziellen Bereiches von Kreditinstituten*. GUC, Verl. der Ges. für Unternehmensrechnung und Controlling, Chemnitz, 2005. ISBN: 978-3934235359.
- [Zer10] STEFAN ZERANSKI: *Implikationen auf die Weiterentwicklung des Liquiditätsrisikocontrollings*. In: FRANK ROMEIKE (Herausgeber): *Die Bankenkrise*, Seiten 163–195. Bank-Verlag Medien, Köln, 2010. ISBN: 978-3-86556-230-2.

# Ehrenwörtliche Erklärung

„Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Bachelorarbeit mit dem Thema

**Entwicklung einer Zwischenschicht für die Nutzung weiterer  
Anwendungen in Verbindung mit der Berechnungskomponente des  
Liquidity Risk Managements**

ohne fremde Hilfe angefertigt habe;

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Bachelorarbeit gekennzeichnet habe;
3. dass ich meine Bachelorarbeit bei keiner anderen Prüfung vorgelegt habe;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.“

Ort, Datum

Unterschrift