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

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

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.[1] Für diese Arbeit ist vor allem die betriebswirtschaftliche Sicht auf Liquidität entscheidend. Die volkswirtschaftliche Sicht wird daher nicht näher erläutert.[2]

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.[3] Sie hängt demnach von der Nähe des Objektes zu Geld ab. Zahlungsmittel haben die höchste Objektliquidität, Immobilien eine geringe.[4] Die Liquidität von Subjekten bezeichnet die Fähigkeit eines Subjekts, z.B. einer Bank, alle Zahlungsverpflichtungen erfüllen zu können.[5]

Zeitlich kann Liquidität in kurz und langfristig unterschieden werden. Bei der kurzfristigen Liquidität steht der Zahlungsaspekt im Vordergrund, meist nur auf einen Tag bezogen.[6] Es muss zu jeder Zeit sichergestellt werden, dass alle fälligen Zah-

[1] vgl. [**?**, S.3] und [**?**, S.13]
[2] vgl. [**?**, S.10]
[3] vgl. [**?**, S.10]
[4] vgl. [**?**, S.3]
[5] vgl. [**?**, S.3] und [**?**, S.11]
[6] vgl. [**?**, S.3f]

lungen in der entsprechenden Höhe beglichen werden können. Diese Bedingung ist bei der Steuerung von Banken zu jedem Zeitpunkt streng einzuhalten. [7]. Synonym werden auch die Begriffe operative Liquidität sowie dispositive Liquidität verwendet.[8]

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.[9] Für Banken ist dies besonders wichtig, da es einen wichtigen Wettbewerbsvorteil gegenüber Mitbewerbern darstellt.[10] Zwischen der kurz und langfristigen Liquidität besteht eine beidseitige Wechselwirkung – eine schlechte kurzfristige Liquidität führt zu Problemen bei der langfristigen Liquidität.[11]

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.[12]

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.[13]

Das Liquiditätsrisiko ist das Risiko, gegenwärtige oder zukünftige Zahlungsverpflichtungen entweder nicht, nicht vollständig oder nicht zeitgerecht nachkommen zu kön-

[7] vgl. [?, S.13] und [?, S.12]
[8] vgl. [?, S.13]
[9] vgl. [?, S.4]
[10] vgl. [?, S.13]
[11] vgl. [?, S.15]
[12] vgl. [?, S.4] und [?, S.65]
[13] vgl. [?, S.5] und [?, S.37]

nen.[14]. 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.[15]. Im weitesten Sinne wird zu dem Liquiditätsrisiko auch die Opportunitätskosten hinzugezogen, die entstehen wenn eine gewinnbringende Transaktion aufgrund fehlender Zahlungsmittel nicht durchgeführt werden kann.[16]

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.[17]. 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 Leitzinses der Notenbank.[18] 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. [19]

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

[14] vgl. [?, S.467f] , [?, S.166f] und [?, S.6]
[15] vgl. [?, S.90] und [?, S.79]
[16] vgl. [?, S.79]
[17] vgl. [?, S.7]
[18] vgl. [?, S.7f]
[19] vgl. [?, S.9]
[20] vgl. [?, S.7f] und [?, S.12]
[21] vgl. [?, S.12] und [?, S.51]
[22] vgl. [?, S.10]
[23] vgl. [?, S.513f]

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.[24] Die Steuerung der Liquiditätsrisiken sollten in die Gesamtbanksteuerung eingebunden werden - nur so kann eine gesamtheitliche Sicht gewährleistet werden.[25]

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. [26]

Durch die hohen Verflechtungen und Abhängigkeiten der Banken untereinander besteht ein Systemrisiko.[27] 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[GL] zusammenbricht. Dadurch fehlt allen Banken eine Liquiditätsquelle – die Probleme einer einzelnen Bank haben den gesamten Bankensektor erreicht.[28]

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.[29] In Deutschland spielen die Mindestanforderungen an das Risikomanagement (MaRisk) und die Liquiditätsverordnung (LiqV) eine wichtige Rolle. In den MaRisk sind Anforderungen an das Liquiditätsrisikomanage-

[24] vgl. [?, S.256f]
[25] vgl. [?, S.20f]
[26] vgl. [?, S.457f]
[27] vgl. [?, S.233f]
[28] vgl. [?, S.265]
[29] vgl. [?, S.291] und [?, S.1f]

ment in Banken festgelegt und damit die Vorgaben von Basel II, dem Nachfolger von Basel I, in deutschem Recht umgesetzt.[30] In der LiqV ist vor allem die Öffnungsklausel wichtig. Durch die Öffnungsklausel ist es Banken möglich, interne Methoden zu verwenden, um rechtliche Vorgaben abzudecken. Dadurch kann das Liquiditätsrisiko über die rechtlichen Vorgaben hinaus gesteuert werden und die Integration in die Gesamtbanksteuerung wird erleichtert.[31]

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 zweiter Dimension unterteilt die Regelungen in nationale und internationale Regelungen. Die genannten Vorschriften sind in Tabelle 1 entsprechend eingeordnet. [32]

	International	National
Qualitativ	-	LiqV
Quantitativ	Basel II	MaRisk

Tabelle 1: Einordnung einiger Liquiditätsvorschriften

Die Maßnahmen, 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.[33]

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, Mindestliquiditätsquote (engl. Liqudiity Coverage Ration) (LCR) und Strukturelle Liquiditätsquote (engl. Net Stable Funding Ratio) (NSFR), die schrittweise eingeführt werden. Die LCR soll verhindern, dass kurzfristige Probleme auf dem Geldmarkt auf Banken übergreifen

[30] vgl. [**?**, S.52f und 58]
[31] vgl. [**?**, S.53] und [**?**, S.194]
[32] vgl. [**?**, S.15]
[33] vgl. [**?**, S.85ff]

und damit 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 LCR wird ab 2015 für alle Kreditinstitute binden eingeführt.[34]

Neben dem LCR existiert mit dem 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 NSFR soll erreicht werden, dass Banken längerfristige Refinanzierungsquellen nutzen und sich nicht nur auf kurzfristige Quellen beschränken. Die Vorgaben bezüglich der NSFR müssen erstmals ab 2018 eingehalten werden.[35]

2.5 Zusammenfassung

[34] vgl. [**?**, S.56]
[35] vgl. [**?**, S.56f]

3 SAP LRM und 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 LRM auch über mobile Endgeräte wie z.B. dem Apple iPad durch die Bereitstellung von entsprechenden Applikationen ermöglicht.[36]

Durch den Einsatz der LRM Lösung können Banken rechtliche Anforderungen, wie z.B. die Einhaltung des LCR und des NSFR im Rahmen von Basel III sicherstellen. Zusätzlich können durch die Bereitstellung einer Berechnungskomponente eigene Kennzahlen zur Steuerung und Überwachung der Bank definiert und umgesetzt werden. Durch den Einsatz von SAP High Performance Analytic Appliance (HANA) ist es möglich, eine zahlungsstromorientierte Herangehensweise trotz der großen Datenmengen zu ermöglichen und gleichzeitig in Echtzeit[37] 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.[38]

[36] vgl. [**?**, S.10 und S.23]

[37] Die Echtzeitanforderung wurde in diesem Kontext auf eine Reaktionszeit von einer Sekunde festgelegt.

[38] vgl. [**?**, S.9 und S.17f]

Das SAP LRM ist dazu in der Lage, selbst mit komplexen Marktsituationen umgehen zu können und somit niedrige Refinanzierungskosten zu erreichen. Als Datenquelle für 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 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.[39]

3.2.2 Architektur

3.2.2.1 NGAP

Die Next Generation ABAP Plattform (NGAP) stellt die Technologische Grundlage für SAP LRM dar. Sie ist eine Plattform für in ABAP geschriebene Business-Anwendungen und soll eine moderne Alternative zu dem SAP NetWeaver Application Server bieten. Entscheidende Merkmale ist die Nutzung von HANA im ABAP-Umfeld, eine Verschlankung des gesamten Technologiestacks, die Einführung einer auf Eclipse[40] basierenden Entwicklungsumgebung und die vereinfachte Unterstützung von REST-basierenden Webservices.[41]

Für viele Anwendungen der SAP stellt der SAP NetWeaver Application Server die Grundlage dar. Er 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 SAP NetWeaver Application Server sind Codeteile bis zu 20 Jahre alt.[42]

Diese Nachteile sind der Grund für die Entwicklung der NGAP. Der NGAP ist eine

[39] vgl. [**?**, S.18 und S.45]
[40] Eclipse IDE - http://www.eclipse.org/
[41] vgl. [**?**, S.1f]
[42] vgl. [**?**] und [**?**, S.1]

separate Codelinie, die ursprünglich auf dem SAP NetWeaver Application Server in der Version 7.2 basiert. Die wichtigsten Veränderungen sind das Entfernen des kompletten Java-Stacks, die Verschlankung 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.[43] 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.[44] Insgesamt konnten durch die Änderungen die Anzahl der Codelinien um 60% reduziert werden. Dadurch ergeben sich Vorteile in der Wartung da potentiell weniger Fehler enthalten sein können.[45]

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

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.[46] Das Ziel von HANA ist es aktuelle operationale Daten in Verbindung mit bestehenden historischen Daten in Echtzeit zu Analysieren und somit Informationen zu gewinnen.[47]

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.[48] Außerdem ist neben der zeilenbasierten Organi-

[43] vgl. [?]
[44] vgl. [?]
[45] vgl. [?, S.2]
[46] vgl. [?, S.14f]
[47] vgl. [?]
[48] vgl. [?, S.12f]

sation 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.[49]

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.[50]

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.[51] Erreicht wird dies durch die Erweiterung der Abfragesprache Structured Query Language (SQL) zu SQLScript[GL]. 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.[52] 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 benutzt. Es wurde ursprünglich als Oberfläche für SAP Business ByDesign[GL] entwickelt und wird jetzt auch für weitere Anwendungen unter Anderem für Anwendungen, die auf HANA und NGAP basieren, verwendet.[53] Ein Beispiel eines mit dem Oberon Framework programmierte Oberfläche ist in Abbildung 1 auf S. 12 zu sehen.

Oberflächen werden mit dem Oberen Framework mit der Programmiersprache C#

[49] vgl. [?, S.13f]
[50] vgl. [?, S.109f]
[51] vgl. [?, S.155f]
[52] vgl. [?, S.9f]
[53] vgl. [?]

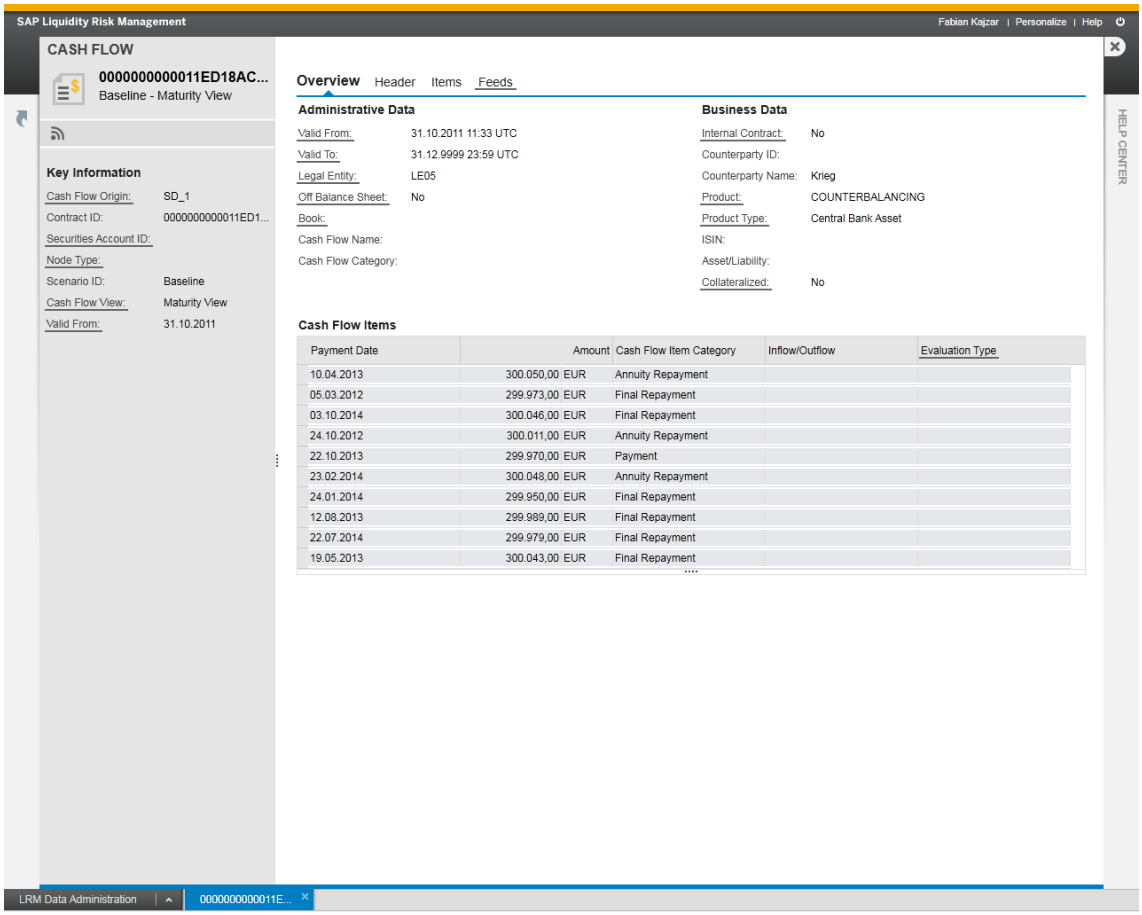

Abbildung 1: Darstellung eines CashFlows-Eintrags mit dem Oberon-Framework

und Microsoft Silverlight[54] entwickelt. Silverlight ist eine Technologie von Microsoft, die als Plugin für verschiedene moderne Browser verfügbar ist und die Entwicklung von Rich Internet Application (RIA)[GL] unterstützt. Die Rich Internet Application (RIA) stellt dabei die erste Schicht der drei Schichtenarchitektur von Oberon dar. Die weiteren Schichten, Die Anwendungslogik und die Persistenz, werden von NGAP und HANA übernommen.[55]

Das Oberon Framework ist als Standardframework ausgelegt und versucht, möglichst 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 der aktuellen Version des Oberon Frameworks abgedeckt werden können. In diesem Fall müssen eigene Lösungen für die Anforderungen

[54] Microsoft Silverlight - http://www.microsoft.com/silverlight/
[55] vgl. [?]

entwickelt werden.[56]

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

Es existieren genau zwei Arten von Liquiditätsgruppen, der Datenbankselektion und der Berechnung. Die Datenbankselektion bildet immer den Ursprung einer Berechnung. Dabei werden einzelne Zahlungsströme von der nach Selektionskriterien selektiert, und nach Bedingungen Zusammengefasst. Selektionskriterien können z.B. der hinterliegende Produkttyp des Zahlungsstroms wie z.B. verzinsliches Wertpapier, oder das entsprechende Finanzrating, z.B. 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.[59]

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, z.B. die nächste Woche, der entsprechende Wert an jedem einzelnen Tag entscheidend. Für den weiteren Horizont, z.B. den Zeitraum zwischen 3 und 5 Jahren, ist nicht jeder einzelne Tag entscheidend, es reichen aggregierte Werte, z.B. 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.[TODO beispiel][60]

[56] vgl. [**?**]
[57] Polyhierarchien sind hierarchische Strukturen, bei der ein Element mehrere übergeordnete Elemente haben kann.
[58] vgl. [**?**, S.38]
[59] vgl. [**?**, S.40]
[60] vgl. [**?**, S.44]

Neben der Datenbankselektion existiert die Liquiditätsgruppenart Berechnung. Dabei werden aggregierte Zahlungsströme nach verschiedenen Regeln verrechnet. Zur 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 z.B. erwartete Kreditausfälle simulieren. Über diese Variablen werden die Szenarien umgesetzt. Mehrere Variablenausprägungen legen ein Szenario fest.[61]

Zur Laufzeit wird die Berechnungskomponente mit 4 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.[62]

3.3 Xcelsius

3.3.1 Überblick und Funktionsumfang

Xcelsius ist ein Teil der SAP Business Objects 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-,

[61] vgl. [**?**, S.39 und S.45]
[62] vgl. [**?**, S.44 und S.47]

Balken- und Flächendiagramme stehen Schaltflächen zur Manipulation wie 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 an die Anforderungen angepasst werden kann.[63]

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 also sowohl Berechnungen, als auch Formatierungen mit Excel durchgeführt werden. Excel wird hierbei als Schnittstelle zwischen der Datenbeschaffung aus verschiedensten Quellen und der eigentlichen Visualisierung genutzt. Alle Daten, die visualisiert werden sollen, werden in das Tabellenkalkulationsblatt von Excel geschrieben und von dort aus ausgelesen. Die Daten können demnach vor der Darstellung in Excel beliebig bearbeitet werden.[64]

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 Bedienung kann als intuitiv beschrieben werden. Besondere Programmiersprachenkenntnisse sind für die Erstellung von Dashboards nicht erforderlich.[65]

Die mit Xcelsius erstellten Dashboards werden mit Hilfe der Adobe Flex Technologie dargestellt. Dadurch ist kein extra Server oder gar eine Datenbank 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.[66]

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 XML-Format verwendet. Außerdem existieren weitere Möglichkeiten, wodurch Dashboards z.B. untereinander Daten austauschen können.[67]

[63] vgl. [?, S.31f und S.229f]
[64] vgl. [?, S.32 und S.236]
[65] vgl. [?, S.32 und S.239]
[66] vgl. [?, S.31f und S.230]
[67] vgl. [?, S.283f]

3.3.2 Bedienungskonzept

Die Oberfläche von Xcelsius lässt sich in 4 Bereiche Aufteilen. Die Bereiche sind in Abbildung 2 dargestellt. Der erste Bereich zeigt die Zeichenfläche, also das eigentliche Dashboard an. Durch die Umsetzung des WYSIWYG-Prinzips kann schon während der Erstellung des Dashboards das spätere Ergebnis dargestellt werden. 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.

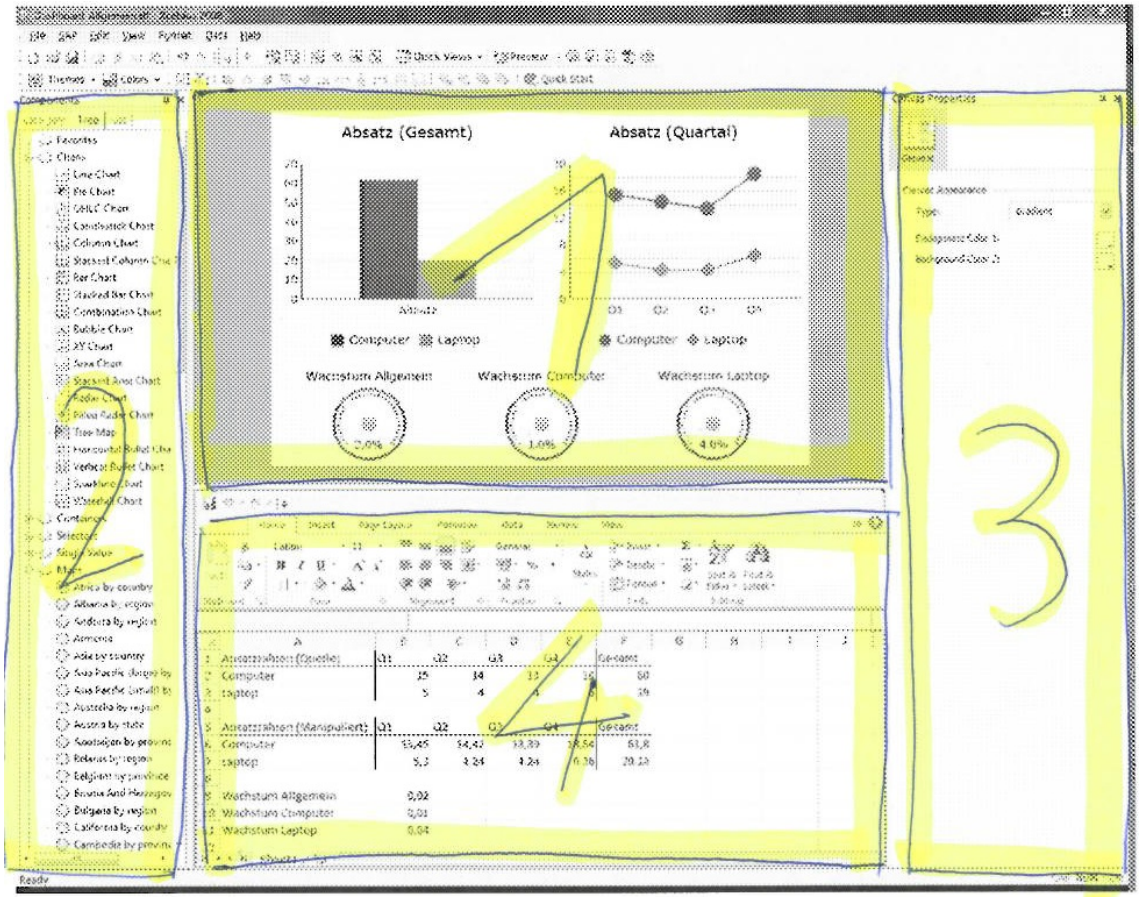

Abbildung 2: [TODO] WT: Aufbau der Benutzeroberfläche von Xcelsius

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 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. Dafür steht der volle Funktionsumfang von Microsoft Excel zur Verfügung. 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).[68]

Der generelle Ablauf des Erstellens von Dashboards soll an dem Beispiel der Absatzzahlen von Produkten in Abbildung 3 auf S. 18 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 eine Datenquelle, z.B. 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.[69]

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. Dreht der Nutzer an einem Drehknopf, wird der aktualisierte Wert in die Zelle in Bereich 3 geschrieben. Da die Datenquelle der Graphen mit Bezug auf die Wachstumswerte berechnet werden, verändert sich dynamisch die Graphen. Das Prinzip ist schematisch noch einmal in Abbildung 4 auf S. 19 dargestellt. So ist es möglich, mit einfachen Mitteln interaktive Dashboards zu erstellen.[70]

3.3.3 Architektur

Xcelsius basiert auf dem .NET Framework[GL] von Microsoft und ist in weiten Teilen in C++ geschrieben. Neben C++ wird Action Script 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. 20 dargestellt.

[68] vgl. [?, S.229]
[69] vgl. [?, S.229f]
[70] vgl. [?, S.237]

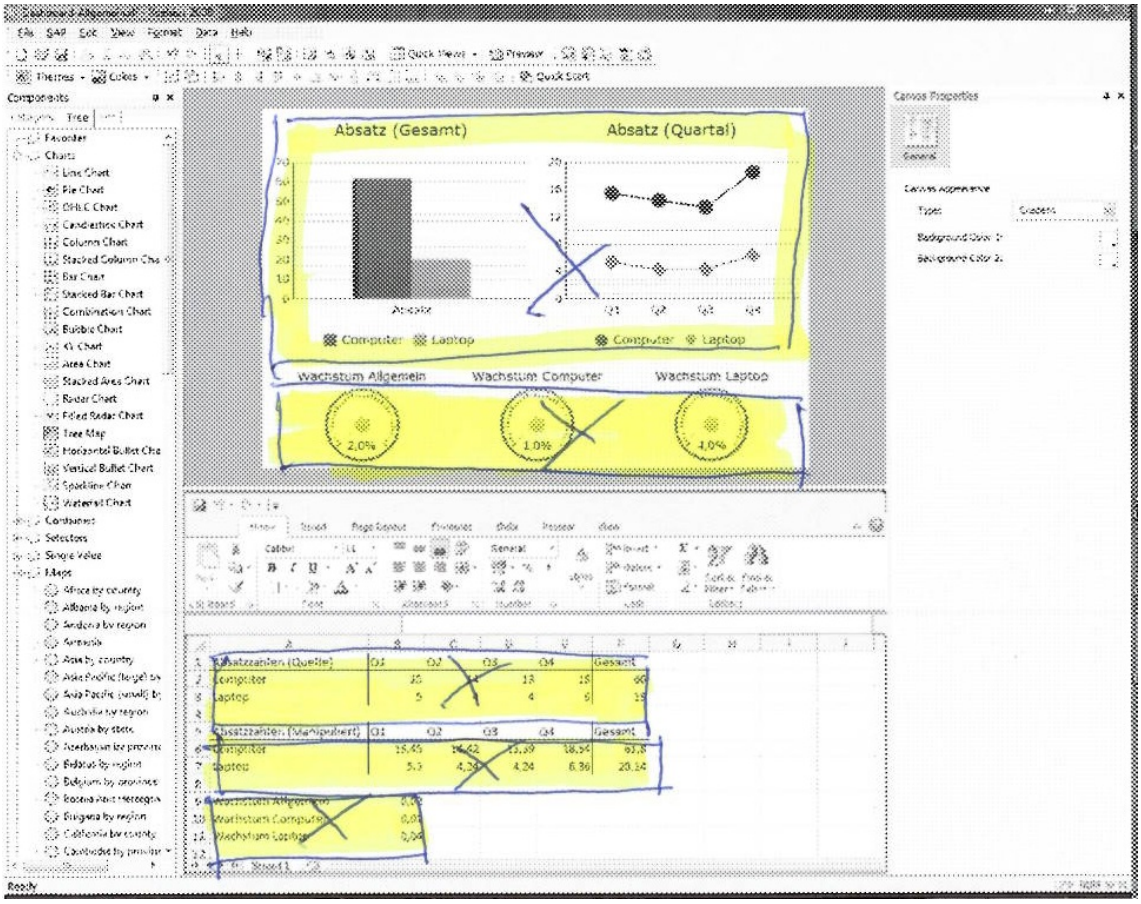

Abbildung 3: [TODO] WT: Beispiel eines Dashboards in Xcelsius

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 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,

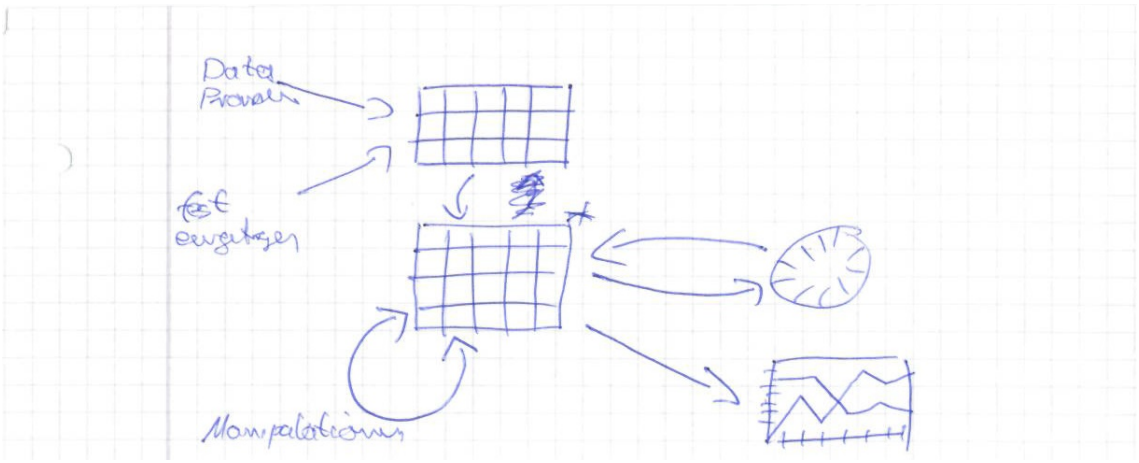

Abbildung 4: [TODO] WT Grundlegende Arbeitsweise mit Xcelsius

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.[71]

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 [TODO Glos?] mit dem komplett eigene Komponenten erstellt werden können. Dadurch ist es möglich, Xcelsius um nahezu beliebige 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 Action Script [TODO Glos]. 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. [72]

[71] vgl. [?]
[72] vgl. [?, S.229] und [?, S.3f]

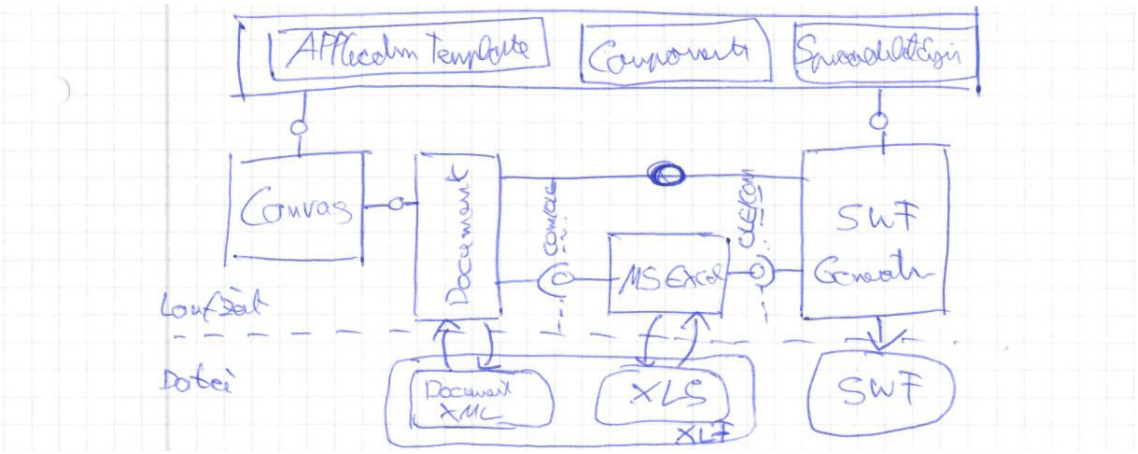

Abbildung 5: [TODO] WT Architekturüberblick von Xcelsius

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 z.B. 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.[73] Den Vorgang zeigt Abbildung 6 auf S. 21.

3.4 Zusammenfassung

[73] vgl. [?, S.283f] und [?, S.6]

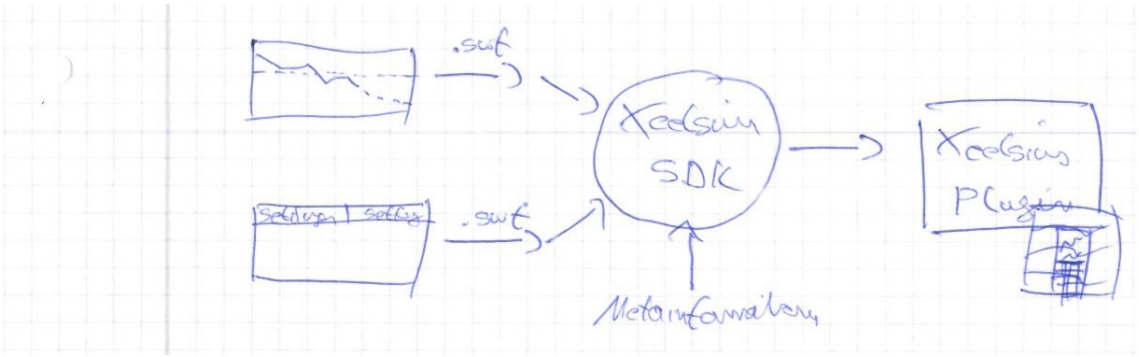

Abbildung 6: [TODO] Plunginsystem von Xcelsius

4 Spezifikation

4.1 Einleitung

4.2 Ausgangssituation

In der aktuellen Entwicklungsversion des SAP LRM gibt es für die Analyse der aktuellen Liquiditätssituation der Bank zwei Einstiegsmöglichkeiten. Es existiert die Funding Matrix und die Key Figure Analysis. Diese unterschieden sich in ihrer Nutzung.

Der Zweck der Key Figure Analysis 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 gemeint. Dazu gehören z.B. im Rahmen von Basel III der LCR und der NSFR. Die Key Figure Analysis kann 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. 23.

Neben der Key Figure Analysis existiert die Funding Matrix. Hier sollen nicht feste Kennzahlen dargestellt werden. Vielmehr wird die Möglichkeit geboten, die Liquiditätssituation der Bank frei zu analysieren. Sowohl die Funding Matrix, als auch die Key Figure Analysis, nutzen für die Ermittlung der Werte die Berechnungskomponente. Bei der Funding Matrix hat der Nutzer die Möglichkeit,die Selektionskriterien in Form der nötigen Parameter für den Aufruf der Berechnungskomponente zur Laufzeit festlegen. Die Darstellung ist hier auf ein Liniendiagramm festgelegt, ein Beispiel zeigt Abbildung 8 auf S. 24 [TODO].

Allgemein ist festzustellen, dass in der aktuellen Version keine Möglichkeit besteht, die Ergebnisse der Berechnungen in irgendeiner Form in anderen Auswertungen, z.B. Auswertungen die die gesamte Situation der Bank beschrieben, einzubinden. Des weiteren besteht bei der Darstellung der Ergebnisse keine Formatierungsmöglichkeit.

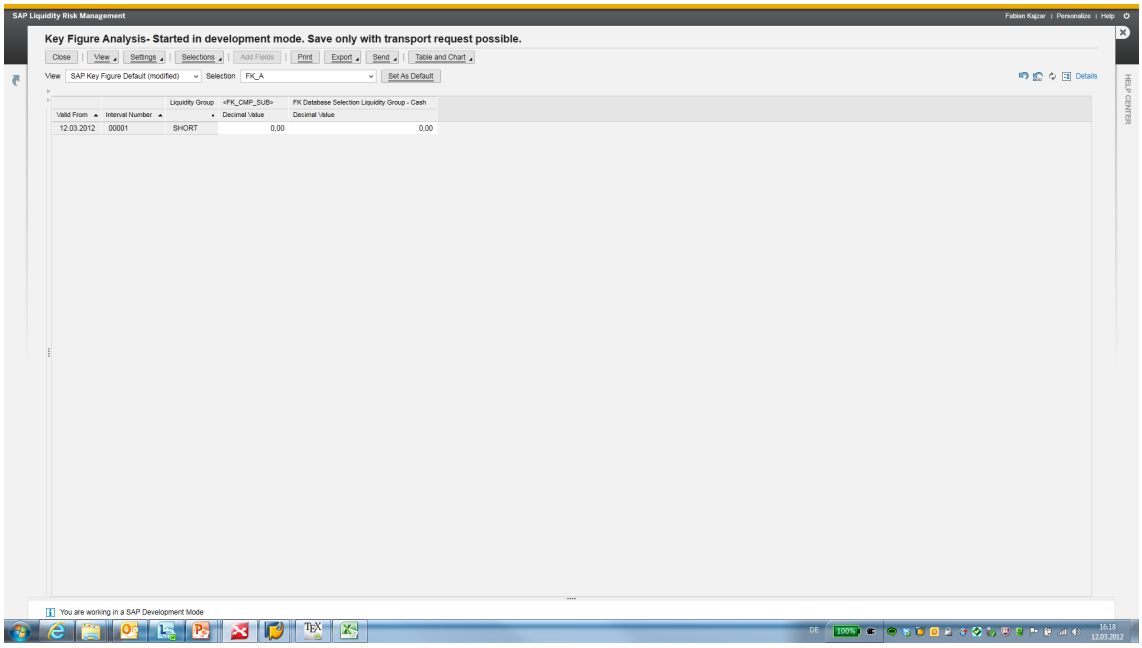

Abbildung 7: [TODO] KeyFigureAnalysis im SAP LRM

Somit sind z.B. Farbe, Form und Größe des Diagramms fest vorgegeben.

4.3 Zielsetzung

Im Rahmen dieser Bachelorarbeit sollen die größten Limitierungen, die sich aus der aktuellen Situation ergeben, 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 Key Figure Analysis und der Funding Matrix im SAP LRM soll diese Auswertung mit weiteren Anwendungen durchgeführt werden können.

Durch die Anbindung von weiteren Anwendungen bietet sich die Möglichkeit, die Liquiditätssteuerung in die Gesamtstererung 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 LRM an weitere Anwendungen

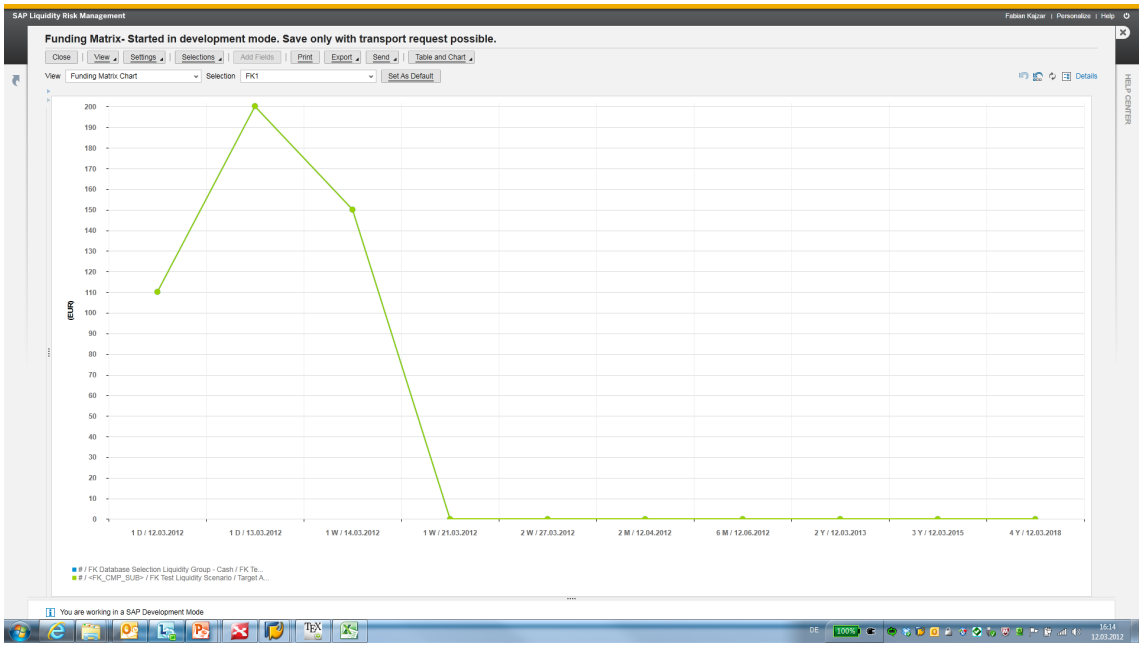

Abbildung 8: [TODO] WT FundingMatrix im SAP LRM

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 graphische Visualisierungsmöglichkeiten bietet, können diese auf die Daten des SAP LRM angewendet werde.

Insgesamt soll die Anbindung des LRM mit möglichst wenig Einstellungsmöglichkeiten konfigurierbar sein. Gleichzeitig ist es erstrebenswert, dass das Konzept der Anbindung und die Implementierung, die im Rahmen der Bachelorarbeit entwickelt werden soll, in der grundsätzlichen Idee und Architektur auch auf weitere Anwendungen übertragen werden kann.

4.4 Anforderungen

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

Bei den funktionalen Anforderungen wurden die zwei folgenden Hauptanforderungen ermittelt:

(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 Key Figure Analysis zur Verfügung stehen, in weitere Anwendungen integriert 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 Funding Matrix im SAP LRM eingestellt werden und auf der Berechnungskomponente basieren. Neben den erforderlichen Parametern wie 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 Abschnitt Zielsituation [TODO] festgelegt sind erreicht werden. Dadurch bieten sich für den Nutzer für die Auswertung und Analyse der Liquiditätssituation einer Bank zusätzliche Möglichkeiten.

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

(A3) **Übertragbarkeit**
Im Rahmen der Bachelorarbeit soll 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 für zusätzliche 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 ein Weiterentwicklungen des SAP LRM eingeführt werden, sollen mit möglichst geringem Aufwand in die Erweiterung eingeführt werden können.

(A6) **Wartbarkeit**
Um die Folgekosten der Erweiterung, z.B. 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 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 auf anderen Anwendungen verfügbar machen.

Die Reihenfolge der Anforderungen spiegelt gleichzeitig auch die Priorität der Umsetzung wieder. Die wichtigste Anforderungen sind Anforderung 3 auf S. 25 und Anforderung 4 auf S. 25, Anforderung 8 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 Kapitel 4.5 konkretisiert die Anforderung 1 auf S. 25 und behandelt die Einsicht in wichtige Liquiditätskennzahlen einer Bank.

Anwendungsfall 1: Liquiditätskennzahl einsehen

Ziel

Liquiditätskennzahl der Bank einsehen

Vorbedingung

Die Liquiditätskennzahl ist im System vorhanden

Nachbedingung: Erfolg

Die Liquiditätskennzahl kann eingesehen werden

Nachbedingung: Fehlschlag

Fehleranzeige

Aktuere

Controller, Management

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 Kapitel 4.5 bezieht sich auf die Anforderung 2 auf S. 25. 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 System vorhanden

Nachbedingung: Erfolg

Die Liquiditätssituation kann eingesehen werden

Nachbedingung: Fehlschlag

Fehleranzeige

Aktuere

Controller, Management

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 analysierende 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, z.B. 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, 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 die Daten des LRM. Gleichzeitig soll aber bei der Analyse und dem Entwurf die Anforderung an die Übertragbarkeit (A[TODO]) 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 semantsiche Ebene, die es ermöglicht Daten zu konsumieren, ohne die technischen Aspekte der Datenbank kennen zu müssen.[74]

Es können Daten aus relationalen Datenbanksystemen, die z.B. in normalisierter Form vorliegen [TODO Glossar?], und aus Data Warehouse-Systeme, die Daten z.B. im Sternschema [TODO ?] ablegen, konsumiert werden. Das Universum abstrahiert diese Daten auf Objekte, die dem Nutzer bereitgestellt werden.[75]

[74] vgl. [**?**, S.48f]
[75] vgl. [**?**, S.49]

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 wiederum an Xcelsius angeboten wird. Ein Überblick ist in Abbildung 9 dargestellt.

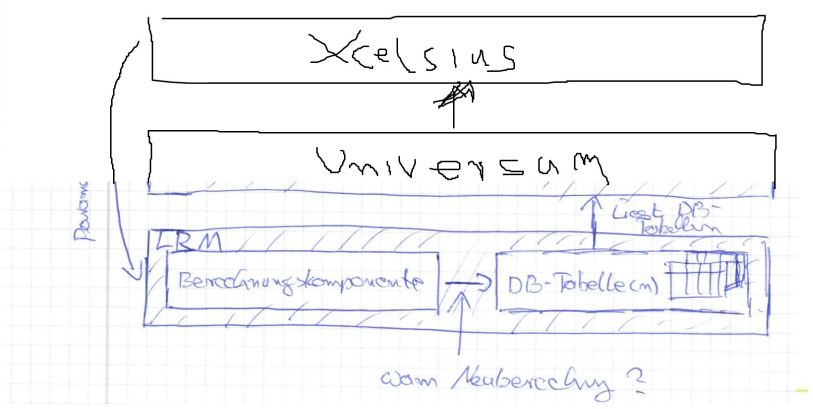

Abbildung 9: [TODO] Umsetzungsmöglichkeit mit BO-Universen

Da ein Universum allerdings nur lesenden Zugriff auf die Daten bietet, stellt sich die Frage, wie eventuelle Selektionsparameter von Xcelsius an die Berechnugnskomponente übergeben werden können. Zusätzlich muss die Berechnung durch die Berechnungskomponente abgestoßen werden, um die Tabelle mit den aktuellen Daten zu füllen. Eine automatische, periodische Berechnung (z.B. durch einen eingeplanten Hintergrundprozess) würde stets veraltete Daten liefern.

Schließlich sind Universen nur mit der Verwendung von BusinessObjects Anwendungen ausgelegt. Das Kriteritum der Übertragbarkeit kann somit nicht garantiert werden.

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 über den Webservice Daten konsumiert. Ein Überblick stellt Abbildung 10 auf S. 31 dar, welche im Folgenden näher erläutert wird.

Dadurch, dass der Aufruf direkt im SAP LRM entgegen genommen wird und nicht indirekt z.B. über ein Universum erfolgt, können eventuelle Selektionsparameter

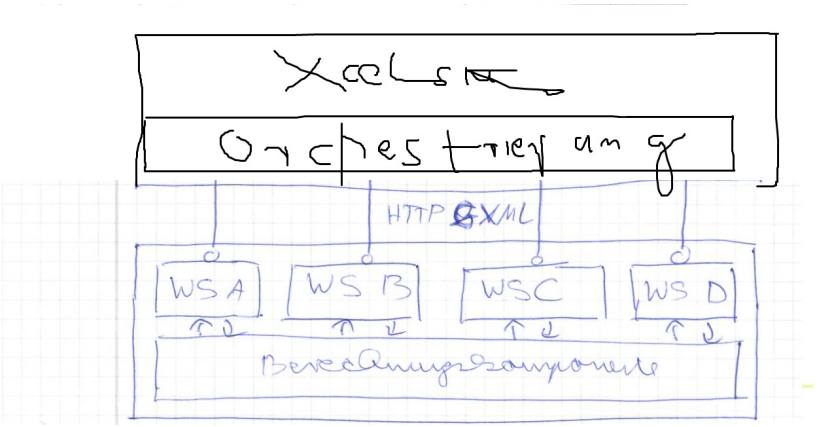

Abbildung 10: [TODO] Umsetzungsmöglichkeit mit WebServices

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 z.B. HTTP für die Übertragung oder XML als Beschreibungssprache eingesetzt, können die Webservices auch von anderen Anwendungen verwendet werden. Dies ist für das Kriterium der Übertragbarkeit wichtig.

Außerdem kommt hinzu, dass eine gute Modularisierung und Wiederverwendbarkeit erreicht werden kann. Die anzubietende 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 internen Frameworks umgesetzt. Dieses bietet die Möglichkeit, festgelegte Klassen ohne zusätzlichen Programmieraufwand automatisch über einen Webservice bereitzustellen. Dazu zählen unter anderem die Liquiditätsszenarien und die Laufzeitbänder. Somit könnten Teile der Webserviceumsetzung ohne großen Aufwand umgesetzt werden.

Ein mögliches Risiko dieser Umsetzungsmöglichkeit stellt die Performance 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-WebServices zur Funktionsunfähigkeit des gesamten Systems.

Xcelsius bietet zwar eine Möglichkeit zur Einbindung von einzelnen WebServices, allerdings besteht standardmäßig keine Möglichkeit, mehrere Webservices zusam-

menzusetzen 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 angewendenten Umsetzungsmöglichkeit wurden die Vor- und Nachteile von beiden Alternativen ermittelt und in Tabelle X [TODO] gegenübergestellt.

	Vorteile	Nachteile
BO Universe	• Standardmäßige Anbindung von Universen	• zusätzliche Infrastruktur notwendig • Übertragbarkeit auf andere Anwendungen • Problem der dynamsichen Übergabe von Parametern • Problem der Aktualisierung der berechneten Daten
WebServices	• Übergabe von Selektionsparametern • keine zusätzliche Infrastruktur erforderlich • gute Modularisierung und Übertragbarkeit • teilweise Bereitstellung von WebServices durch Framework	• Komplexe Orchestrierung erforderlich • Performanceprobleme bei WebService-Aufrufen • Ausfall eines Teil-WebService führt zum Ausfall des Gesamtsystems

Tabelle 2: [TODO]

Dabei fällt auf, dass die Umsetzung mit Hilfe von Universen im Vergleich zu der

Umsetzung über WebServices deutlich weniger Vorteile bietet bietet. Gleichzeitig bieten die Universen zusätzlich not 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.

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 [TODO] 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 5 Teilbereiche unterteilt werden. Sie werden für die Strukturierung der Analyse genutzt. Bei den Bereichen handelt es sich um folgende Teile:

1 WebService-Bereitstellung im SAP LRM
Das SAP LRM System stellt über WebServices alle benötigten Daten bereit.

2 WebService-Konsumierung in Xcelsius
Die WebServices werden konsumiert und zunächst in ein internes Objektformat umgewandelt.

3 WebService-Orchestrierung 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 Excel von Xcelsius geschrieben und Selektionsparameter aus dem Excel gelesen werden.

6.2.2 Grundlegende Architektur und Funktionsweise

In diesem Abschnitt wird ein Überblick (vgl. dazu **??** auf S. ??) die grundlegende Architektur der Lösung gegeben. Dabei werden Erweiterungen in zwei Systemen (LRM und Xcelsius) umgesetzt. Gleichzeitig wird die grundlegende Funktionsweise gerade im Hinblick auf die Interaktion mit dem Nutzer nähergebracht.

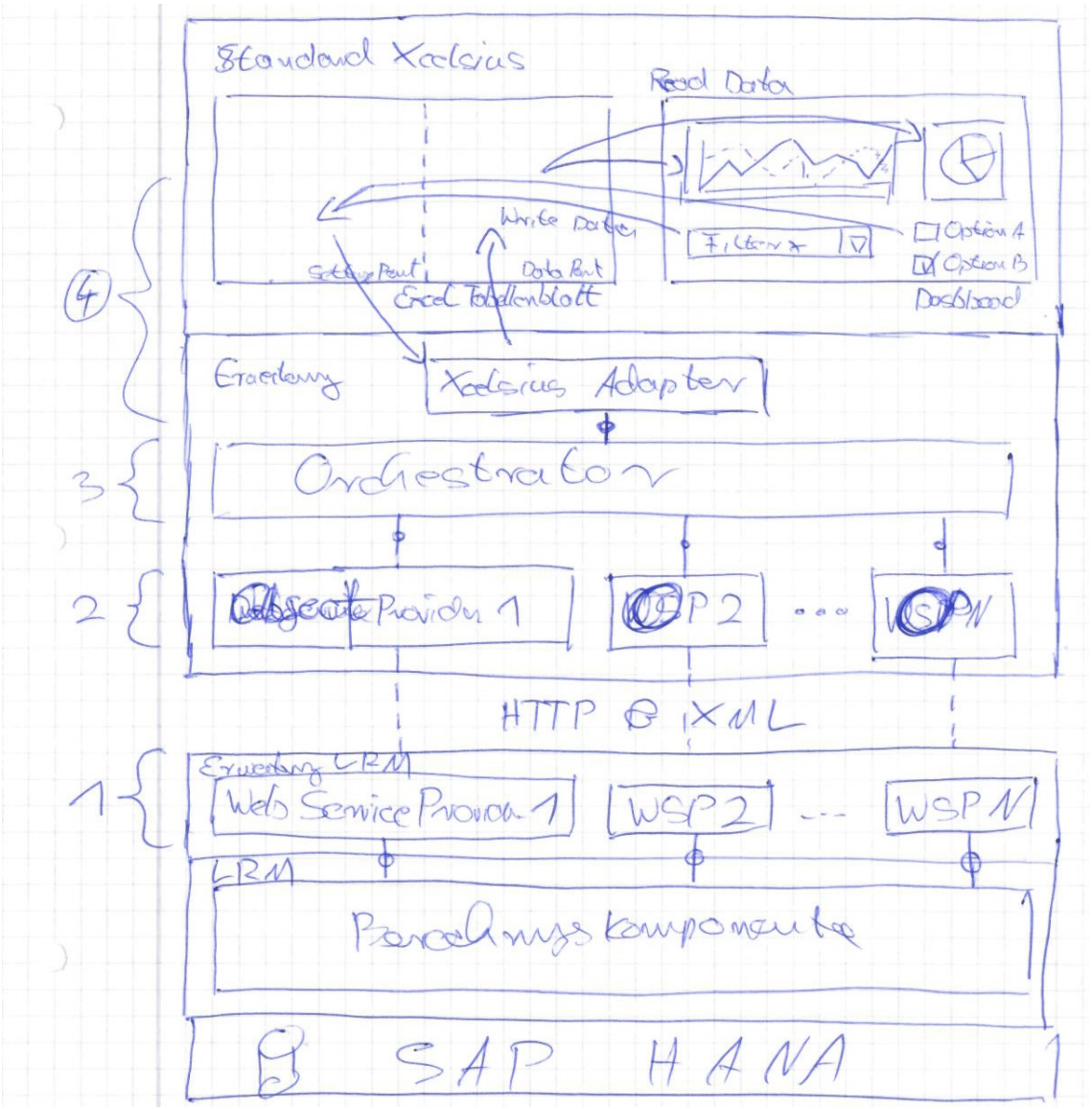

Abbildung 11: [TODO] WT Architekturüberblick der Erweiterung

Im SAP LRM System muss für jeden WebService ein WebServiceProvider bereitgestellt werden. Der WebServiceProvider kann über einfache Funktionsaufrufe ent-

weder Berechnungne über die Berechnungskomponente durchführen lassen, oder auf die BusinessObjekte, wie z.B. einem Laufzeitband oder eine Liquiditätsgruppe zugreiffen. Das Ergebniss liefert er über das HTTP-Protokoll im XML-Format.

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

Der Xcelsius Adapter ist über ein Tabellenblatt die Schnitstelle für den User. Das Tabellenblatt wird dabei in zwei Bereiche unterteilt. 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 z.B. für die Berechnungskomponente im LRM gebraucht werden (vgl. [TODO verweis zu Parametern], werden aus dem Settings-Part 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 LRM in Xcelsius bereit. In welcher Form die Daten verwendet 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 auf S. 36 dargestellt.

Das Kernstück der Erweiterung ist die Klasse Orchestrator. Hier ist die komplette Orchestrierungslogik für die Kennzahlen und die Funding Matrix hinterlegt, d.h. die Ergebnisse der einzelnen WebServices werden kombiniert. Dazu existieren die entsprechenden Methoden updateKPI() und updateFundingMatrix(). Für den Datenaustausch mit dem Excel von Xcelsius existiert zwischen der Klasse Orchestrator und der Klasse XcelsiusAdapter eine 1:1 Assoziation.

Die Klasse XcelsiusAdapter stellt die Verbindung zwischen der Erweiterung und dem Excel Tabellenblatt von Xcelsius dar. Mit der Methode writeToXcelsius() können

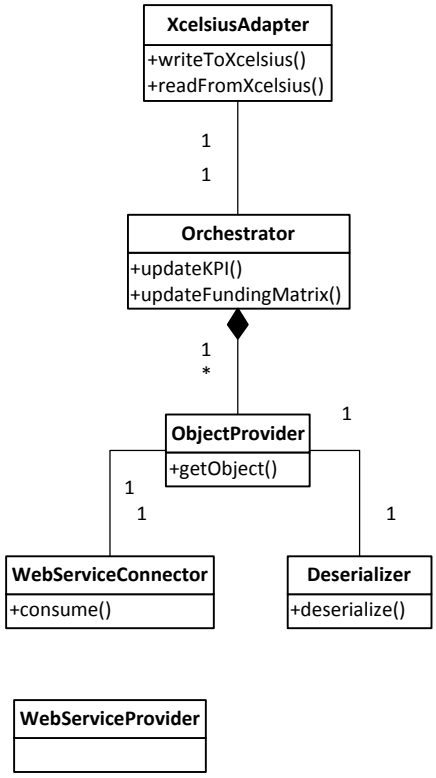

Abbildung 12: Klassendiagramm der Analysephase

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 ObjectProviders 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 jeweilse eine 1:1 Assotiation. Der WebServiceConnector übernimmmt die Kommunikation zwischen der Erweiterung und dem SAP LRM System über das HTTP Protokoll. 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-Protokoll 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 User in einem Sequenzdiagramm in Abbildung 13 dargestellt.

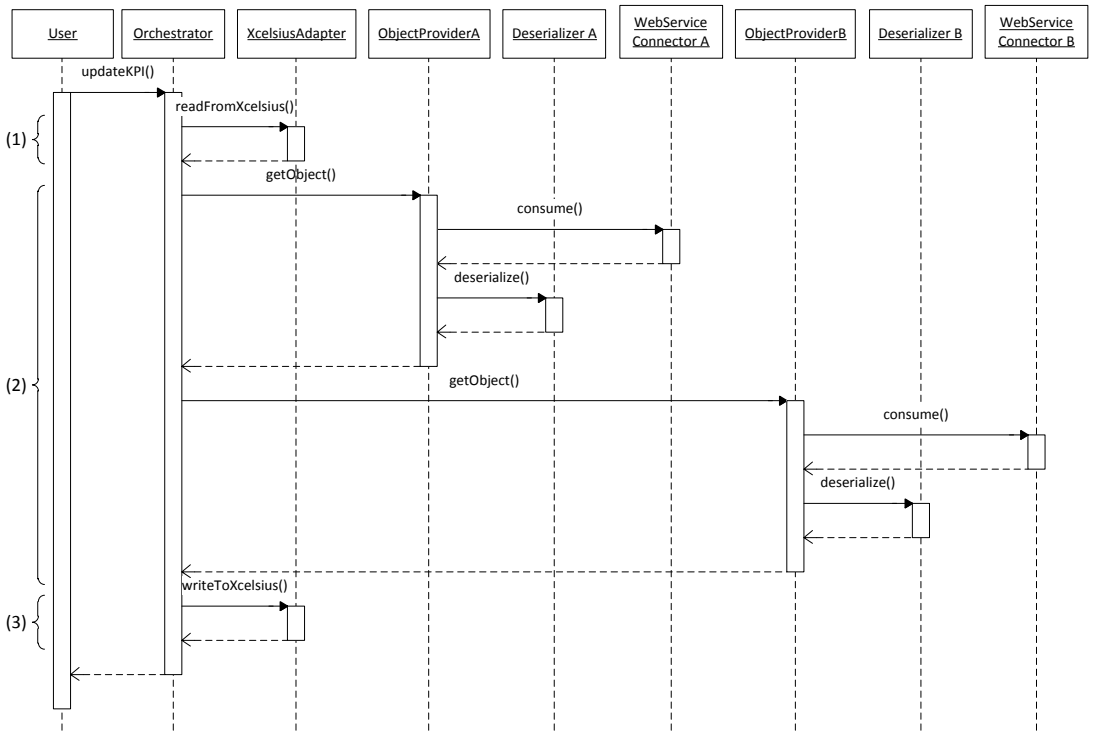

Abbildung 13: Klassendiagramm der Analysephase

In dem Beispiel möchte der User den Wert einer Kennzahl abfragen. Dazu ruft er in der Orchestrator-Instanz die Mehtode 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 Users ermittelt. Dazu gehört die Kennzahl, die abgerufen werden soll und evtl. weitere Einstellungen.

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

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

Der Aufruf der WebServiceProvider durch den Orchestrator kann asynchron erfolgen. Hat der Orchestrator alle Ergebnisse der Prvider 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 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-XML-Format [TODO] zugegriffen werden kann. Diese Services werden in der Erweiterung genutzt werden. Zusätzlich befindet sich eine iPad-App in Entwicklung. Für diese wurden zwei weitere WebServices entwickelt, die dem Zugriff auf KPIs und die FundingMatrix bieten.

Es ergeben sich fünf nutzbare WebServices, die orchestriert werden müssen. Das Ziel ist immer der Zugriff auf den FundingMatrix Service oder den KPI Service. Die Abhängigkeiten der jeweiligen Services sind in Abbildung 14 dargestellt und sollen im Folgenden erläutert werden:

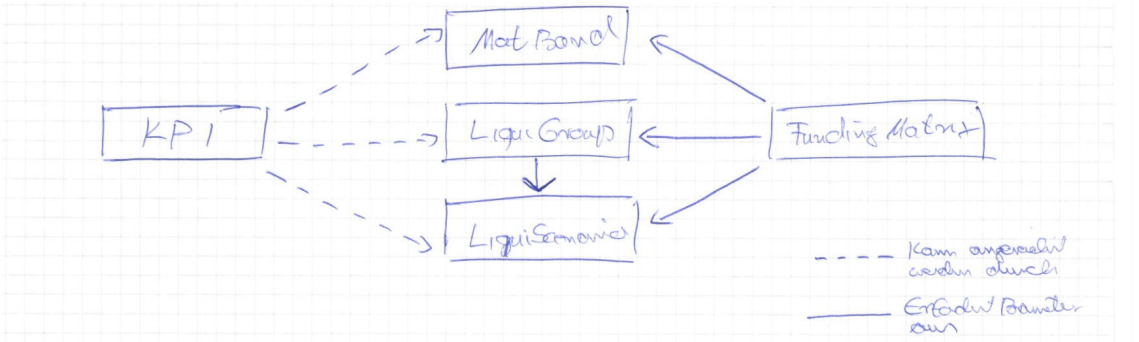

Abbildung 14: Abhängigkeiten der WebServices

Der FundingMatrix WebService ruft bei einem Aufruf die Berechnungskomponente des SAP LRM auf und muss demnach mit den vier Parametern versorgt werden (TODO siehe X). Es besteht eine Abhängigkeit zu dem drei WebServices, die automatisch durch das Framework bereit stehen. Der LiquiGroup Service ist so umgesetzt, dass er ein LiquiditätsSzenario als Parameter erwartet und darauf alle möglichen Liquiditätsgruppen zurückliefert – auch hier besteht eine Abhängigkeit.

Der KPI WebService erfordert keine Parameter und kann eigenständig aufgerufen werden. Er liefert alle fest definierten KPIs im System zurück. Das Resultat beinhaltet allerdings nicht alle Informationen, z.B. zu dem verwendeten Liquiditätsszenario für die Berechnung oder der hinterliegenden Liquiditätsgruppe. Das Ergebnis kann also durch weitere WebService Aufrufe 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. 40 zu sehen.

Objekte, wie z.B. ein LiquiditätsSzenario oder ein Laufzeitband, werden 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

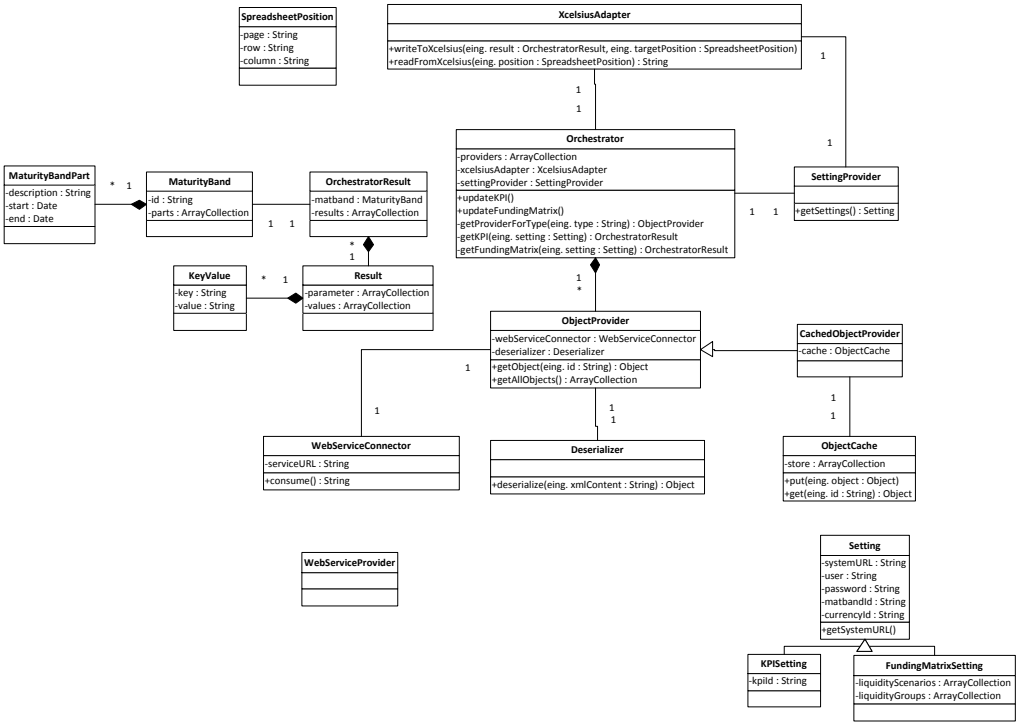

Abbildung 15: Klassendiagramm der Entwurfsphase

Abfrage gestartet werden. Dafür 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 aufruf zu starten.

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() udn 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 ResultInstanz 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 Szenario sein. Zum Anderen werden in dem Attribut values die einzelnen Werte gespeichert.

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

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.

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

6.3.3 Dynamisches Modell

An dem Dynamischen Modell sind in der Entwurfsphase nur kleinere Änderungen durch die neu hinzugekommenen Klassen durchgeführt worden. Das angepasste Sequenzdiagramm ist in Abbildung 16 auf S. 42 dargestellt.

In Abschnitt (1) ist die Einbindung des SettingProvider zu sehen. Außerdem lässt 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 dem ObjectCache zwischengespeichert. Dieser Vorgang ist in dem Sequenzdiagramm aus Platzgründen nicht abgebildet. Die erneute Anfrage kann er ohne einen weiteren WebService Aufruf direkt beantworten.

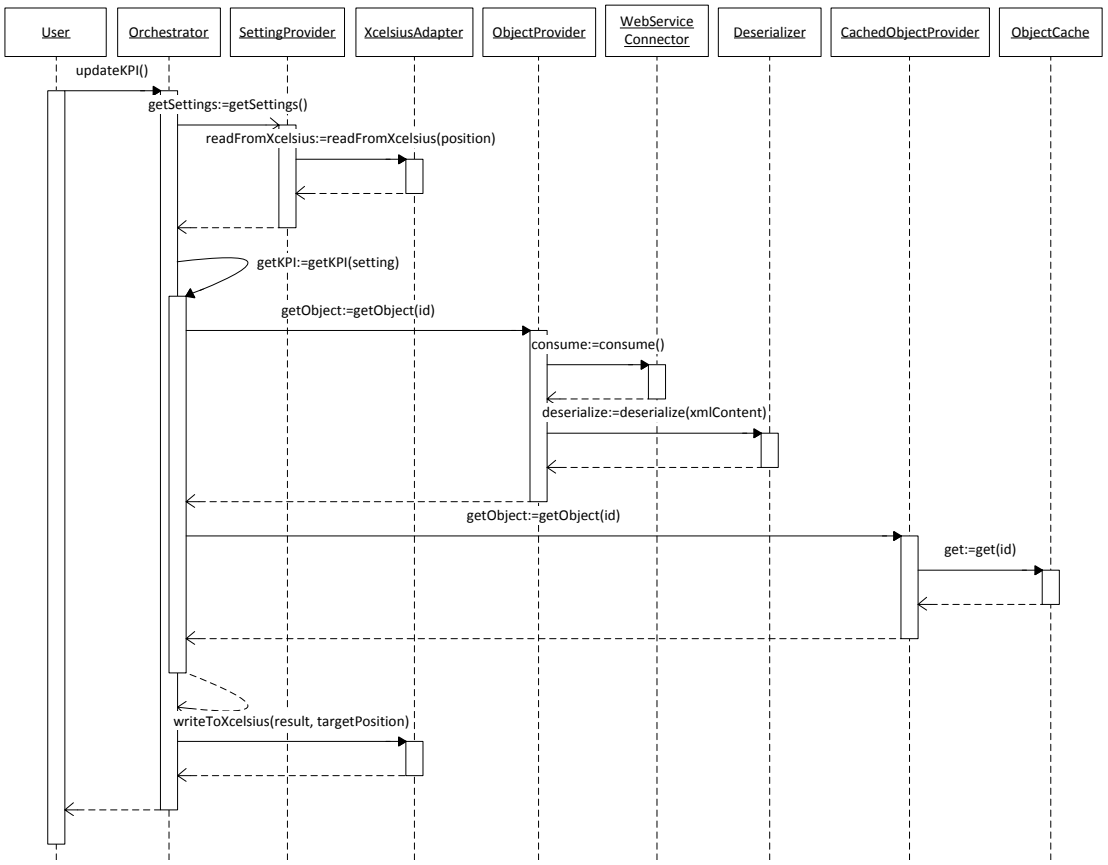

Abbildung 16: Sequemzdiagramm der Entwurfsphase

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 Abschnitt [TODO] **??** auf S. ?? dargestellt. Diese Objektstruktur muss in eine Tabellenstruktur in Xcelsius umgewandelt werden. Ein konkretes Beispiel für die Objektstruktur zeigt Abbildung 17 auf S. 43 [TODO], welches in Tabelle 3 auf S. 44 [TODO] entsprechend umgesetzt ist.

Die Tabellenstruktur soll folgendermaßen aufgebaut werden: Eine Ergebnissreihe

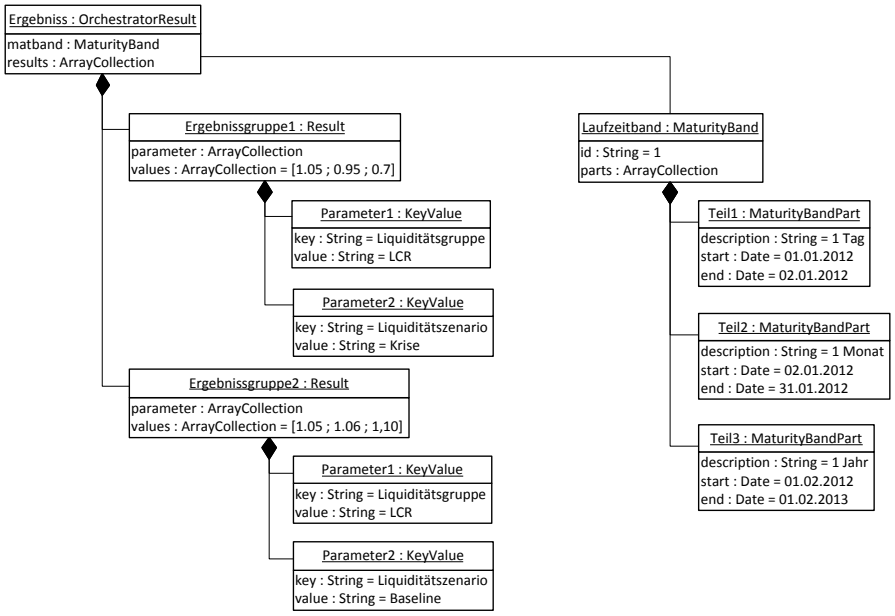

Abbildung 17: [TODO] WT Objektstruktur

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 Result-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 können z.B. später in einer Visualisierung als X-Achenbeschriftung verwendet werden. In der Objektstruktur sind das die MaturityBandPart-Instanzen. In einer Zeile in der Tabellenstruktur sind demnach die Werte aller Ergebnisinstanzen für ein Laufzeitbandabschnitt eingetragen.

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

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

Tabelle 3: [TODO]

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 ein einem klar definierten Raum bewegen.

Eine dieser Beschränkungen ist die Verhinderung von CrossDomainRequests [TODO]. Dabei handelt es sich um einen WebService Aufruf, 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. Da das SAP LRM nicht auf dem Computer läuft, sondern auf einem Server in einer eigenen Domain, ist das Konsumieren von Daten aus dem LRM erstmal nicht möglich, da es durch die Sanbox blockiert wird.

Für dieses Problem kommen zwei Lösungsansätze in Frage. Zum einen ist es Möglich, dass der Anbieter des WebService explizit feslegen kann, dass der WebService auch von über verschiedene Domänen hinweg konsumiert werden kann. Dies geschieht über eine Datei crossdomain.xml, welche in dem Hauptverzeichnis der Domain abgelegt werden muss. Dies wäre 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 er auch unter der domain localhost befindet, wird der Zugriff von der Sandbox zugelassen. Der WebServer stellt die Anfrage an den

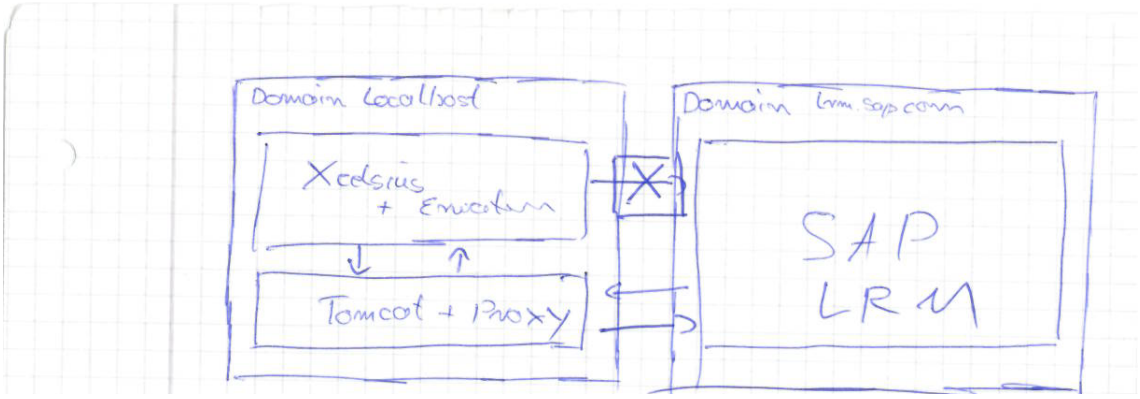

Abbildung 18: [TODO] [TODO Referenziereun!]

richtigen WebService im SAP LRM und leitet die Antwort an die Erweiterung weiter.

Für die Umsetzung wurde lokal ein Tomcat-Webserver installiert. Für diesen wurde ein einfaches Java-HttpServlet entwickelt, welches die geforderte Funktionalität umsetzt. Das Coding des Servlets ist in [TODO] 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 WebService-Aufrufes 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 WebSerivces in der Klasse WebServiceConnector die Flex-Klasse HTTPService verwendet. Diese bietet nur die Möglichkeit über EventListener auf zwei Events, nähmlich 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 Abbildung x dargestellt.

```
public function consume():void{
        // HTTPService-Instanz erstellen
        webService = new HTTPService();
        webService.url = serviceURL;
        webService.method = "GET";
        // Event Listener hinzufuegen
        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: [TODO]

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 oben [TODO]. Alternativ kann auch eine Objekt-Instanz übergeben werden. Auf dieser Instannz wird dann zu dem entsprechenden Zeitpunkt eine festgelegte Methode aufgerufen.

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 erhällt die Antworten durch die asynchronität in beliebiger Reihenfolge in der Methode onObjectProviderSuccess(). Um die Aufrufe unterschieden 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. [TODO was bei Fehler?!] 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 [TODO] ist in allgemeiner Form der erläuterte Ablauf implementiert. Dabei werden zwei beliebige ObjectProvider abgefragt und 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);
        objectProvider.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: [TODO]

6.4.3 WebServiceAblauf

In diesem Abschnitt soll an dem Beispiel des FundingMatrix 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 dann die passende URL für den WebService-Aufruf auf Grundlage der Settings 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 Connectors der eigentliche Aufruf gestartet. Entsprechend dem asynchronen Programmablauf [TODO Kapitelverweis] wird die eigene Instanz als Callback und der Request-Token mitgegeben.

Tritt bei dem Aufruf ein Fehler auf, wird asynchron die Methode onWebServiceFail aufgerufen. Der Fehler muss nur dem Orchestrator weitergeletet werden. Dazu wurde auch hier bei dem getObject-Aufruf eine Referenz auf den Orchestratro 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 WebService 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 [TODO] dargestellt.

Bei der Deserializierung 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 [TODO] dargestellt ist, zeigt findet sich in Listing [TODO].

Die ObjectProvider und Deserializer für die weiteren Webservices sind analog dazu programmiert. Bei einem CachedObjectProvider wird in der getObject-methode zunächst ermittelt, ob sich das angeforderte Objekt schon im Cach befindet und dann direkt zurückgegeben. Ist es nocht nicht vorhanden, wird der WebServiceAufruf gestartet und nach der Deserializerung wird das Objekt im Cache für weitere Anfragen gehalten.

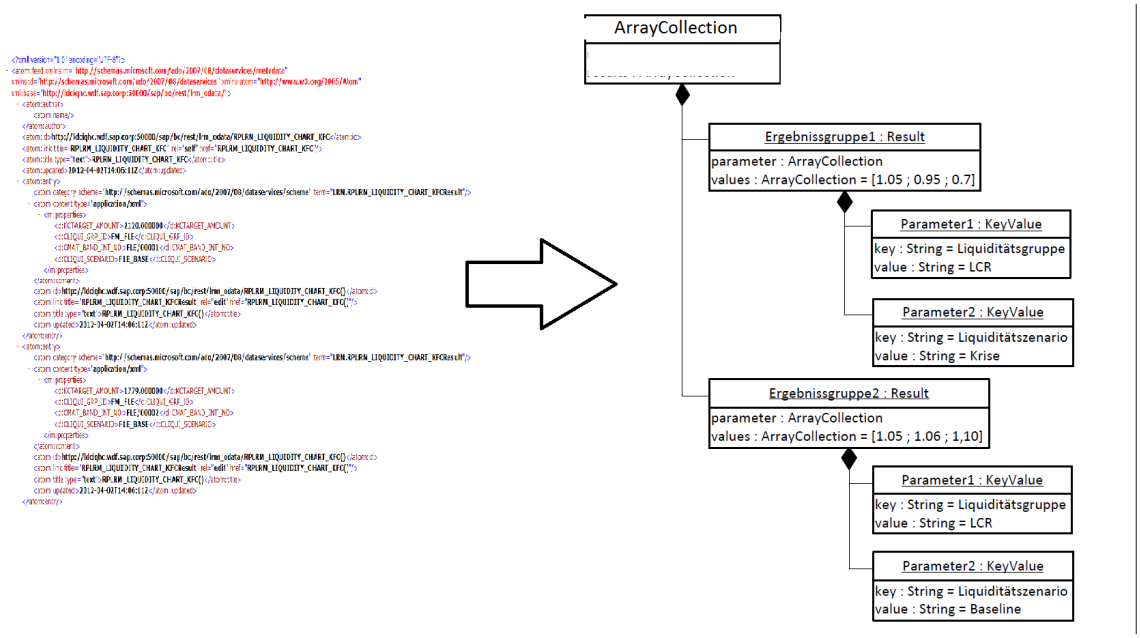

Abbildung 19: [TODO]

6.4.4 Xcelsius Binding

Für den Datenaustausch zwischen der entwickelten Erweiterung und dem Tabellenblatt von Xcelsius muss auf die bereitgestellten Funktionen des Xcelsius SDKs 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 z.B. 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 z.B. mit einem 2-dimensionalen 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.

```
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: [TODO]

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 [TODO] des Xcelsius-Adapters 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 Möglichkeiten

7.3 Vergleich

7.4 Performance

7.5 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, HttpServletRespor
                // build request url
                String requestedUrl = req.getServletPath();
                String params = req.getQueryString();
                String urlString = "http://ldciqhc.wdf.sap.corp:500
                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().enco
                conn.setRequestProperty ("Authorization", "Basic " +

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

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

Listing 4: [TODO]

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