Eine Fallstudie zur Spezifikation von Fachkomponenten eines Informationssystems für Virtuelle Finanzdienstleister – Beschreibung und Schlussfolgerungen

Peter Fettke, Peter Loos, Markus von der Tann

Technische Universität Chemnitz, Fakultät für Wirtschaftswissenschaften, Information Systems & Management (Professur Wirtschaftsinformatik II), D-09107 Chemnitz, Germany, Tel.: +49/371/531-4375, Fax: -4376, E-Mail: peter.fettke@isym.tu-chemnitz.de, loos@isym.tu-chemnitz.de, markus.vdtann@isym.tu-chemnitz.de, WWW: http://www.isym.tu-chemnitz.de/

Zusammenfassung. In dem Beitrag wird zunächst kurz die Funktionalität des Forschungsprototyps cofis.net, einem Informationssystem für Virtuelle Finanzdienstleister, vorgestellt und ein Einblick in seine Entwicklungsgeschichte gewährt. Anschließend wird ein Überblick über die Fachkomponenten des Informationssystems geben. Die Fachkomponenten von cofis.net wurden auf Basis des Memorandums zur Vereinheitlichung der Spezifikation von Fachkomponenten des Arbeitskreises 5.10.3: Komponentenorientierte betriebliche Anwendungssysteme der Gesellschaft für Informatik spezifiziert. Auszüge aus der erstellten Spezifikation werden vorgestellt. Darüber hinaus werden Erfahrungen, die bei der Spezifikation gemacht worden sind, sowie dabei identifizierte Problembereiche dargelegt. Abgerundet wird die Fallstudie durch Empfehlungen, die Hinweise zur Weiterentwicklung des Memorandums beschreiben.

Schlüsselworte: Standardisierung, Component-based Software Engineering, Virtualität, cofis.net, Virtuelle Anlage, Virtuelle Überweisung, Bankleitzahlen

1 Einführung in die Fallstudie

1.1 Charakterisierung des Systems

Zur Zeit sieht sich die Finanzdienstleistungswirtschaft vielfältigen technologischen sowie wirtschaftlichen Entwicklungen gegenüber.(Swoboda 2000) Virtuelle Finanzdienstleister sind ein Ansatz, um diesen Entwicklungen gerecht zu werden. In Anlehnung an (Stockmann 1998) wird unter einem Virtuellen Finanzdienstleister ein Finanzdienstleister verstanden, der mit Hilfe moderner Informations- und Kommunikationstechnologien die Dienstleistungen am Markt agierender Finanzdienstleistungsanbieter vermittelt, neu kombiniert und konfiguriert. An der Fakultät für Wirtschaftswissenschaften der Technischen Universität Chemnitz wurde von den Professuren *Wirtschaftsinformatik II* sowie *Finanzwirtschaft und Bankbetriebslehre* ein Modell eines Virtuellen Finanzdienstleisters konzipiert und mit der Entwicklung eines entsprechenden prototypischen Informationssystems begonnen.(Fettke et al. 2001)

1.2 Ziele der Entwicklung des Prototyps

Die mit der Prototypenentwicklung einhergehenden Forschungsaktivitäten sind mit folgender Zielstellung verbunden (Fettke et al. 2001, S. 5f.): Aus betriebswirtschaftlicher Sicht werden verschiedene Finanzprodukte unter Laborbedingungen konzipiert und modelliert, um diese im Rahmen verschiedener experimenteller Studien zu evaluieren. Die Ergebnisse sollen auf die Konzeption neuer Finanzprodukte übertragen werden. Aus wirtschaftsinformatischer Sicht ist die Zielstellung zweigeteilt. Auf der einen Seite werden im Sinne eines explorativen Prototypings die fachlichen Anforderungen an Informationssysteme für Finanzdienstleister in der elektronischen Finanzdienstleistungswirtschaft ermittelt und spezifiziert. Auf der anderen Seite sollen im Sinne eines experimentellen Prototypings verschiedene Architekturmodelle sowie technische Lösungsideen für Informationssysteme für Virtuelle Finanzdienstleister untersucht und bewertet werden.

1.3 Funktionaltität des Prototyps

Zur Zeit implementiert der Prototyp die beiden Finanzprodukte Virtuelle Überweisung sowie Virtuelle Anlage. Eine Virtuelle Überweisung ermöglicht es einem Kunden, einen Geldbetrag von einem Konto bei einer beliebigen Bank auf ein Konto bei einer beliebigen anderen Bank via Internet zu überweisen. Bei einer Virtuellen Anlage werden die Anlageprodukte verschiedener Banken vom Virtuellen Finanzdienstleister einheitlich aufbereitet und dem Kunden angeboten. Auf einer groben Ebene, besteht der Prototyp aus drei zentralen Komponenten (im allgemeinen Sinne):

- Der Virtual Helpdesk realisiert die Darstellung der Finanzprodukte zum Kunden. Er verwaltet die durchgeführten Überweisungen, getätigten Anlagen sowie Kundendaten. Dieser Systemteil ist vergleichbar mit einer universellen Finanzmanagement-Software, die nicht nur Kontobewegungen dokumentiert, sondern auch Analyse- und Beratungsfunktionen bereitstellt.
- Der Finanzdienstleistungs-Bus (FDL-Bus) ist die zentrale Komponente zum einheitlichen und standardisierten Datenaustausch zwischen dem Virtual Helpdesk sowie realen Finanzdienstleistern. Dieser Systembaustein sorgt dafür, dass der Virtual Helpdesk unabhängig von konkreten Datenformaten für Geschäftsdaten und weiteren Transaktionen gehalten werden kann. Der Austausch der Daten über den FDL-Bus basiert einheitlich auf XML.
- Die Anbindung realer Finanzdienstleister an den FDL-Bus wird über Schnittstellen-Komponenten realisiert. Diese Schnittstellen sind je nach angebundenem Finanzdienstleister unterschiedlich ausgelegt und automatisiert. Bei einer hochautomatisierten Variante ist es möglich, das Format der Stamm- sowie Transaktionsdaten, die von den Finanzdienstleistern geliefert werden, direkt auf das definierte XML-Format abzubilden. (Im Idealfall basieren die Datenformate des Virtuellen Finanzdienstleisters und der originären Anbieter der Finanzdienstleistungen auf dem selben XML-Format). Im anderen Extremfall existiert an der Schnittstelle ein Medienbruch, der dazu führt, dass die Schnittstellen-Komponente Fax-Seiten generiert und an den entsprechenden Finanzdienstleister übersendet, der den Auftrag ausführen soll.

Einen Überblick über Technologien, die bei der Implementierung des Prototypen eingesetzt wurden, gibt Tabelle 1.

Technologie	Produkt
Implementierungssprache	Java (JDK 1.1.8), HTML, Javascript, XML
Middleware	Brokat Twister 2.3.5
Webserver	XPresso mit SSL
Datenbank	Oracle 8i

Tabelle 1: Überblick über die Implementierungstechnologien des Systems cofis.net

1.4 Ziele der Fallstudie

Die in diesem Beitrag beschriebene Fallstudie verfolgt das Ziel, den derzeitigen Stand des Memorandums zur Spezifikation von Fachkomponenten des Arbeitskreises 5.10.3: Komponentenorientierte betriebliche Anwendungssysteme der Gesellschaft für Informatik zu evaluieren. Dafür wurden die bisher entwickelten Fachkomponenten des Systems cofis.net gemäß des aktuellen Standes (Stand vom 2001-07-16) des Memorandums (Ackermann et al. 2001) spezifiziert. Die dabei gewonnenen Ergebnisse und die gesammelten Erfahrungen werden in diesem Beitrag dargestellt. Darüber hinaus werden Empfehlungen formuliert, in welche Richtungen das Memorandum weiterentwickelt werden sollte.

2 Überblick über die Fachkomponenten

Das System cofis.net besteht aus den in Tabelle 2 genannten Fachkomponenten. Aus der Tabelle geht hervor, welche Funktionalitäten von einer Fachkomponente im Allgemeinen angeboten werden. Einige Komponenten sind nicht elementar, bestehen also aus weiteren Fachkomponenten. Die Bestandteile einer nicht elementaren Fachkomponente werden ebenso in der Tabelle dargelegt.

Bei der Systementwicklung von cofis.net wurde folgender Komponenten-Begriff unterstellt: Eine Komponente ist eine Menge von Java-Packages, deren bereitgestellte Dienste durch die Schnittstellenbeschreibungssprache der verwendeten Middleware spezifiziert werden. Darüber hinaus werden ebenso Java-Applets als eigenständige Komponenten verstanden. Es wird an dieser Stelle eingeräumt, dass nicht jede Komponente des cofis.net-Systems alle der in (Turowski 2001, S. 15-19) genannten Merkmale einer Fachkomponente uneingeschränkt erfüllt. Welche Merkmale gut bzw. weniger gut erfüllt werden, wird im Folgenden näher erläutert:

- Merkmal *Softwarebaustein*: Alle Komponenten sind durch entsprechende Java-Packages realisiert und können als Softwarebausteine verstanden werden. Insofern ist dieses Merkmal erfüllt.
- Merkmal *wohldefinierte Schnittstelle*: Alle Komponenten bieten ihre Dienste über Schnittstellen an. Diese Schnittstellen sind im Repository der Middleware explizit spezifiziert. Dieses Merkmal ist damit erfüllt.

Fachkomponente	Funktionalität	besteht aus
cofis.net (Frontend)	Grafische Nutzeroberfläche des Systems cofis.net	Virtual Financial Helpdesk, Applets der Nutzer- und Rechteverwaltung, Applets der Kontenverwaltung
cofis.net (Backend)	Gesamtfunktionalität des Systems cofis.net	Nutzer- und Rechte-verwaltung, Konten-verwaltung, Virtuelle Anlageverwaltung, Virtuelle Überweisung
Virtual Financial Helpdesk	Grafische Nutzeroberfläche für den Kunden, ermöglicht die Abwicklung aller Kundenaufträge	Elementare Komponente
Kontenverwaltung	Verwaltung aller Kundenkonten	Elementare Komponente
Virtuelle Anlage-verwaltung	Verwaltung aller virtuellen Anlagen bzw. Anlagenverträge	Elementare Komponente
Virtuelle Überwei-sung	Verwaltung aller Transaktionsdaten, die zur Abwicklung Virtueller Überweisungen benötigt werden	Elementare Komponente
Nutzer- und Rechteverwaltung	Verwaltung aller Nutzer und deren Rechte	Elementare Komponente
Business Log	Fachliche Logbuch-Funktionalitäten, Anzeige fachlicher Ereignisse, die im System ausgelöst werden	Elementare Komponente
Bankleitzahlen	Suche von Bankleitzahlen, Überprüfung der Konsistenz von Bankleitzahlen	Elementare Komponente
Finanzdienstleister (Backend)	Ausgewählte Funktionalität (Zahlungsverkehr und Anlage) eines Informationssystems für Finanzdienstleister (nachgebildet)	Elementare Komponente
Finanzdienstleister (Frontend)	Grafische Nutzeroberfläche für ausgewählte Funktionalität (Zahlungsverkehr und Anlage) eines Informationssysstems für Finanzdienstleister (nachgebildet)	Elementare Komponente
Finanzdienstleister-Bus	Kommunikationskomponente zur Interaktion der cofis.net- sowie Finanzdienstleister-Komponenten	Elementare Komponente
Komponenten-Anwendungs-Framework	Bereitstellung anwendungsbezogener Dienste, bspw. Handhabung von XML-Daten, Sicher-stellung der Persistenz von Benutzerdaten	Elementare Komponente

Tabelle 2: Überblick über die Fachkomponenten des Systems cofis.net

- Merkmal *wiederverwendbar*: Die Dienste der erstellten Komponenten können prinzipiell ohne Veränderungen von beliebigen anderen Komponenten genutzt werden. Beispielsweise können dieselben Geschäftsabläufe durch unterschiedliche Oberflächen genutzt werden. Auch wenn noch keine praktischen Erfahrungen zur Wiederverwendbarkeit der Komponenten vorliegen, ist trotzdem absehbar, dass die Wiederverwendbarkeit unterschiedlich gut ist. Während bspw. für die Fachkomponente Bankleitzahlen eine gute Wiederverwendbarkeit gegeben ist, dürfte die Wiederverwendbarkeit der Benutzerverwaltung geringer sein. Dieses Merkmal ist damit nur zum Teil erfüllt.
- Merkmal *einsetzbar in unvorhersehbaren Kombinationen mit anderen Komponenten*: Die Bindung der Komponenten erfolgt zur Laufzeit und nicht zur Übersetzungszeit. Daher ist es möglich, eine einzelne Komponente ohne Neuübersetzung des Gesamtsystems gegen eine andere Komponente mit identischer Schnittstelle zur Laufzeit auszutauschen. Dieses Merkmal ist damit erfüllt.
- Merkmal *abgeschlossen*: Das Datenmodell der Komponenten ist nur z. T. entkoppelt. Dies bedeutet, dass jede Komponente prinzipiell mit dem gesamten Datenbankmodell ausgeliefert werden muss, auch wenn diese Komponente nur einen Teilbereich des Datenbankmodells nutzt.
- Merkmal *vermarktbar*: Dieses Merkmal ist bei Forschungsprototypen prinzipiell nicht erfüllt.

Zusammenfassend ist festzustellen, dass die Komponenten des Systems das Merkmal *vermarktbar* gar nicht und die Merkmale *abgeschlossen* und *wiederverwendbar* nur eingeschränkt erfüllen. Die restlichen Merkmale werden uneingeschränkt erfüllt.

3 Ergebnisse der Fallstudie

In der Tabelle 3 werden die (bewerteten) Ergebnisse der Fallstudie dargestellt. Positive Aspekte werden mit dem Symbol „+“, negative mit „-“ und neutrale mit „o“ markiert. Ebenso werden für jede Spezifikationsebene entsprechende Empfehlungen zur weiteren Entwicklung ausgesprochen. Im Anhang dieses Beitrags ist die vollständige Spezifikation der Fachkomponenten „Bankleitzahlen“ sowie „Virtuelle Anlage“ angegeben.

Ebene	Ergebnisse
Administrations-Ebene	+ Falls zur Spezifikation der Ebene ein entsprechendes XML-Werkzeug eingesetzt wird, ist eine gute Lesbarkeit und leichte Erstellbarkeit der Spezifikation möglich. - Teile der Spezifikation sind noch zu restriktiv vereinbart, so besteht lediglich die Möglichkeit, jeweils ein Komponenten-System- bzw. –Anwendungs-Framework zu spezifizieren. Empfehlung: Die Spezifikationsvorschrift (DTD) sollte entsprechend der angefallenen Probleme erweitert bzw. weniger restriktiv gestaltet werden.
Syntax-Ebene	+ Die Spezifikation der Schnittstelle kann auf Basis der OMG IDL leicht vorgenommen werden. + Durch die Verwendung standardisierter Datentypen wird eine Komponente unabhängig von einer Implementierungssprache und die Spezifikati-

	on besser nachvollziehbar. - Die in der Fallstudie eingesetzte Middleware unterstützt keine IDL-konformen Datentypen bzw. Signaturen. Dadurch war es notwendig, sämtliche Schnittstellen-Definitionen aller Fachkomponenten manuell zu konvertieren. Dieses Vorgehen erscheint wenig praktikabel. Empfehlung: Es sollte nicht Aufgabe des Spezifikationsvorschlages sein, entsprechende Mapping-Anweisungen zwischen verschiedenen Schnittstellenbeschreibungssprachen zu normieren. Allerdings sollte darüber nachgedacht werden, wie eine einheitliche Dokumentation eines derartigen Vorgehens aussehen könnte. Ferner ist zu klären, wie bei der Spezifikation vorzugehen ist, wenn die verwendete IDL mächtiger als die OMG IDL ist. Evtl. sollten an dieser Stelle ebenso alternative Spezifikationsnotationen berücksichtigt werden.
Verhaltens-Ebene	+ Die Notation zwingt, Vor- und Nachbedingungen bzw. Invarianten von Diensten zu explizieren. Diese Technik unterstützt übliche Prinzipien des Software Engineering. - Die Formulierung des Verhaltens einer Fachkomponente mit OCL war prinzipiell schwierig durchzuführen, obgleich das Verhalten z. T. trivial war. Ursache hierfür ist zunächst ein fehlendes UML-Modell. Ferner erscheint es kaum sinnvoll, für bestimmte Aussagen entsprechende Methoden einzuführen, um diese ausdrücken zu können. Dies wäre bspw. bei der Komponente Bankleitzahlen notwendig, um die Aussage „name ist ungueltig“ durch eine entsprechende Methode zu ersetzen. Dieses Vorgehen erscheint umständlich. So wurde im Ergebnis das Verhalten der Komponente in „Pseudo-OCL“ formuliert. - Bereits einfache Verhaltensmerkmale einer Fachkomponente werden schwer spezifizierbar und auch von anderen Begutachtern schwer nachvollziehbar. - Der hohe Aufwand für eine vollständig formale Spezifikation wurde bspw. bei der Komponente „Nutzer- und Rechteverwaltung“ deutlich, so dass diese Komponente bei der Spezifikation zunächst ausgespart wurde. Empfehlung: Es sollte eine sekundäre allgemeinsprachliche Notation eingeführt werden, welche die angegebenen Verhaltensregeln zusätzlich erläutert. Ferner sollten Aussagen getroffen werden, auf welche Art und Weise ein UML-Modell einzuführen ist, um OCL ordnungsgemäß verwenden zu können. Dabei sollten ebenso die Beziehungen zwischen diesem UML-Modell und der Syntax-Ebene von Fachkomponenten dargelegt werden.
Abstimmungs-Ebene	+ Die vollständige Spezifikation dieser Ebene offenbart Nutzungsmöglichkeiten der Dienste einer bzw. mehrerer Fachkomponenten. Dabei werden insbesondere Abhängigkeiten zwischen verschiedenen Komponenten deutlich. - Im Wesentlichen treten hierbei ähnliche Probleme wie bei der Spezifikation auf der Verhaltensebene auf: Die Spezifikation wird im Verhältnis

	zum Gehalt der getroffenen Aussagen einerseits komplex und andererseits schwer verständlich. Siehe hierzu bspw. die Komponentenabstimmungs-Ebenen der Komponente Virtuelle Anlage. Es sei darauf hingewiesen, dass in diesem Beispiel die Abhängigkeiten zum Komponenten-System-Framework sowie zum Komponenten-Anwendungs-Framework nicht dargestellt worden sind. Empfehlung: Zunächst sollte eine Literaturquelle angeführt werden, die einen guten Zugang zu Temporalen Logiken ermöglicht. Ferner sollten ebenso Konzepte zur systematischen natürlichsprachlichen Ergänzung der formalen Spezifikation in Betracht gezogen werden.
Qualitäts-Ebene	- Es ist unklar, welche Größen letztendlich spezifiziert werden sollen, wie diese verglichen werden sollen und mit welchen Methoden sie erhoben werden sollen. Die von uns vorgenommene Spezifikation hat einen „konstruierten“ Charakter. Empfehlung: Konkretere Ausarbeitung, in welchem Umfang Qualitäts-Maße zu spezifizieren sind und welche Verfahren zur Anwendung kommen sollen. Ebenso sollte eine konkrete Notation für die Spezifikation eingeführt werden (bspw. XML). Hierbei ist insbesondere zu berücksichtigen, welche Vorarbeiten bereits in der Literatur existieren. Ferner ist zu überlegen, inwieweit der Gedanke von Referenzimplementierungen zielführend eingesetzt werden kann.
Domänen-Ebene	Vorbemerkung: Die Definition der betrieblichen Aufgabe erfolgte nicht auf dieser Ebene, sondern im Rahmen der Administrations-Ebene. Damit nimmt diese Ebene ausschließlich die Rolle einer Terminologie-Ebene ein. + Auf Basis der in Anlehnung an das Thesaurus-Konzept vorgeschlagenen Beziehungen, die zwischen Begriffen bestehen können, konnte leicht ein Begriffssystem aufgebaut werden. + Es erwies sich als nützlich, die Definition verschiedener Begriffe nicht auf allen Spezifikations-Ebenen zu verteilen, sondern an einer Stelle zentralisiert vorzunehmen. - Die im Memorandum vorgeschlagene Normsprache konnte aufgrund fehlender Erläuterungen der Technik zur genauen Verwendung nicht benutzt werden. - Die Spezifikation wurde zunächst in einer einzigen Datei gepflegt, die für alle Komponenten relevant war. Dadurch wurden einerseits Unklarheiten bei der Zuordnung von Begriffen zu Komponenten vermieden und andererseits konnten keine redundanten Definitionen erstellt werden. Allerdings sieht der bisherige Standardisierungsansatz keine Komponenten-übergreifende Terminologie-Ebene vor. Empfehlung: Erstens. Es sollte die Nützlichkeit von Fachnormsprachen an einem konkreten Beispiel einer Fachkomponentenspezifikation aufgezeigt werden. Ferner sollte eine leicht beschaffbare Literaturquelle zur Einführung in Fachnormsprachen angegeben werden. Falls keine klaren Vorzüge der Fachnormsprachen zu erkennen sind, sollten diese durch den Thesau-

	rus-Ansatz ersetzt werden. Zweitens: Es sollte überlegt werden, wie eine Komponenten-übergreifende Terminologie zu pflegen ist. Hierbei sollte ebenso berücksichtigt werden, wie Komponenten-spezifische Definitionen vorgenommen werden sollten.
Ebenen-unabhängige Aspekte	- Es zeigte sich, dass eine rein formale Spezifikation der Verhaltens- und Abstimmungsebene schwer verständlich ist. Ebenso hatte der Ersteller der Spezifikation Schwierigkeiten, nach einem gewissen Zeitraum die eigens erstellte Spezifikation erneut zu verstehen. Daher wurden die Spezifikationen durch kurze, allgemein verständliche Erläuterungen ergänzt. - Es bleibt unklar, auf welche Art und Weise evtl. vorhandene Komponenten-System- bzw. Komponenten-Anwendungs-Frameworks zu spezifizieren sind. - Es wäre hilfreich, Angaben zur Installation von Komponenten einführen zu können. Hierbei ist insbesondere daran zu denken, welche Dienste von einer Fachkomponente vorausgesetzt werden, die von dem Betriebssystem zur Verfügung gestellt werden müssen (bspw. Netzwerkanbindung oder Datei-Verzeichnisdienste). - Es besteht bisher keine Möglichkeit, Angaben zur Konfiguration einer Komponente bzw. zu möglichen Konfigurationen zu spezifizieren. Mit anderen Worten sollte überlegt werden, wie ein Customizing einer Fachkomponente gehandhabt werden kann. o Es wurden keine Anstrengungen unternommen, Spezifikationsaussagen bzgl. der motivierten High Level Architecture (HLA) zu formulieren. Empfehlung: Erstens: Es sollte überlegt werden, in wieweit Möglichkeiten zur systematischen Integration von natürlichsprachlichen Aussagen innerhalb der Formalsprache bestehen. Zweitens: Ferner sollten Lösungen für die skizzierten Problembereiche entwickelt werden. Drittens: Es sollten Layout-Richtlinien zur Formatierung der Spezifikation entwickelt werden.

Tabelle 3: Ergebnisse der Fallstudie

4 Zusammenfassung und Ausblick

Zusammenfassend ist festzustellen, dass die Spezifikationen auf der Administrations-, der Syntax- sowie auf der Domänenebene (relativ) unproblematisch erstellt werden konnten. Die Spezifikationen auf der Verhaltens-Ebene sowie den Abstimmungs-Ebenen stehen im bekannten Zielkonflikt zwischen Präzision der getroffenen Aussagen einerseits und Verständlichkeit sowie Wirtschaftlichkeit andererseits. Die Qualitäts-Ebene hinterlässt insgesamt einen ausbaufähigen Eindruck.

Aussagen zur Beurteilung der Nutzenpotentiale der erstellten Spezifikationen im Software-Entwicklungsprozess können zur Zeit nicht getroffen werden. Um hierbei zu fundierten Aussagen zu kommen, sollten experimentelle Ansätze in konstruierten Entwicklungsszenarien herangezogen werden, um nicht nur formal-wissenschaftliche, sondern auch personelle sowie subjektive Faktoren wie Verständlichkeit, Lernfähigkeit u. ä. zu berücksichtigen.

Ergänzend sei bemerkt, dass die bisherigen Bemühungen zur Vereinheitlichung der Spezifikation von Fachkomponenten ausschließlich auf einer sprachlichen Ebene anzusiedeln sind. Eine sprachliche Basis zur Formulierung der Anforderungen, die von einer Fachkomponente ausgehen, ist zwar eine notwendige Voraussetzung, um die Standardisierung von Fachkomponenten voranzutreiben. Indes wäre es ebenso wünschenswert, methodische Hilfestellungen in der Art zu entwickeln, dass ein Vorgehensmodell zur Verfügung gestellt wird, aus dem hervorgeht, in welcher Reihenfolge konkrete Spezifikationsschritte durchzuführen sind. Ein solches Vorgehensmodell sollte darüber hinaus ebenso inhaltliche Aussagen treffen, nach welchen Kriterien die gesamte Funktionalität eines Anwendungssystems im Hinblick auf einzelne Komponenten aufgeteilt werden könnte. Insofern ist das vorhandene Memorandum zur Vereinheitlichung der Spezifikation von Fachkomponenten nur ein Baustein einer umfassenden Konstruktionslehre für komponentenbasierte betriebliche Informationssysteme.

Anhang

Der Anhang enthält die vollständige Spezifikation der Fachkomponenten Bankleitzahlen sowie Virtuelle Anlage.

A Spezifikation der Fachkomponente Bankleitzahlen

A.1 Spezifikation der Administrations-Ebene

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Peter Fettke
(TUC) -->
<!DOCTYPE ADMINISTRATION_LAYER SYSTEM "ADMINISTRATION_LAYER.dtd">
<ADMINISTRATION_LAYER>
      <TECHNICAL_ATTRIBUTES>
            <SIGNATURE>Bankleitzahlen</SIGNATURE>
            <VERSION>V 1.0</VERSION>
            <BILL_OF_MATERIALS>
                  <ARTIFACT NAME="Komponentenspezifikation.pdf"
DESCRIPTION="Spezifikationsdatei der Komponente."/>
                  <ARTIFACT NAME="bankcodes.jar" DESCRIPTION="Java-Class-
Dateien der Implementierung"/>
                  <ARTIFACT NAME="bankcodes.tws" DESCRIPTION="Komponenten-
IDL für Middlware Twister von Brokat"/>
                  <ARTIFACT NAME="create_db.sql" DESCRIPTION="SQL-Skript
zur Erstellung der Oracle-Datenbank zum Speichern der Datenbasis"/>
                  <ARTIFACT NAME="blz0010pc.txt"
DESCRIPTION="Originaldatenquelle der Bankleitzahlen der Deutschen Bundes-
bank (Quelle: http://www.bundesbank.de/)"/>
                  <ARTIFACT NAME="SATZ188.doc" DESCRIPTION="Beschreibung
der Originaldatenquelle blz0010pc.txt (Quelle:
http://www.bundesbank.de/)"/>
                  <ARTIFACT NAME="script_sampledata.pl" DESCRIPTION="Perl-
Skript zum Konvertieren des Datenfiles der Deutschen Bundesbank in entspre-
chende SQL-Anweisungen"/>
                  <ARTIFACT NAME="bankcodestests.jar"
DESCRIPTION="Beispiel-Client-Implementierungen zur Überprüfung der Kompo-
nentenfunktionalität."/>
            </BILL_OF_MATERIALS>
            <SYSTEMCONFIGURATION>
                  <PROCESSOR VALUE="x86"/>
                  <MEMORY_SIZE>512</MEMORY_SIZE>
```

```
                  <DISK_SIZE>10</DISK_SIZE>
                  <OPERATING_SYSTEM VALUE="WinNT">

      <OPERATING_SYSTEM_VERSION>4.0</OPERATING_SYSTEM_VERSION>
                  </OPERATING_SYSTEM>
                  <DBMS>Oracle 8i</DBMS>
                  <COMPONENT_FRAMEWORK>Brokat Twister
2.3.5</COMPONENT_FRAMEWORK>
            </SYSTEMCONFIGURATION>
      </TECHNICAL_ATTRIBUTES>
      <BUSINESS_ATTRIBUTES>
            <BUSINESS_TASK>Suche Bankleitzahl</BUSINESS_TASK>
            <BUSINESS_TASK>Suche Bank</BUSINESS_TASK>
            <BUSINESS_TASK>Überprüfe Bankleitzahl</BUSINESS_TASK>
            <BUSINESS_TASK>Überprüfe Bank</BUSINESS_TASK>
            <BRANCH_OF_ECONOMY>Kreditgewerbe (65)</BRANCH_OF_ECONOMY>
            <FUNCTIONAL_AREA>Rechnungswesen</FUNCTIONAL_AREA>
            <USER_DEFINED_DESCRIPTION>Die Komponente ermöglicht die Suche
nach Banken anhand verschiedener Kriterien wie Ort der Niederlassung, Bank-
leitzahl oder Name der Bank. Darüber hinaus kann eine gegebene Bankverbin-
dung hinsichtlich ihrer Korrektheit überprüft wer-
den.</USER_DEFINED_DESCRIPTION>
      </BUSINESS_ATTRIBUTES>
      <MISC_ATTRIBUTES>
            <MANUFACTURER>Technische Universität Chemnitz, Professur Wirt-
schaftsinformatik II, D-09107 Chemnitz</MANUFACTURER>
            <CONTACT>Peter Fettke, +49/371/531-4362, peter.fettke@isym.tu-
chemnitz.de</CONTACT>
            <TERMS_OF_LICENCE>Allgemeine Lizenzbedingungen liegen nicht
vor, sondern müssen jeweils im Einzelfall ausgehandelt wer-
den.</TERMS_OF_LICENCE>
            <TARGET_GROUP>Forschungseinrichtungen oder forschungsinteres-
sierte Industrieunternehmen.</TARGET_GROUP>
      </MISC_ATTRIBUTES>
</ADMINISTRATION_LAYER>
```

A.2 Spezifikation der Syntax-Ebene

```
Interface Bankleitzahlen
{
  typedef string Bankleitzahl;

  struct Bank
  {
    string bankleitzahl;
    string bankname;
    string ort;
  }

  sequence <Bank> BankenListe;

  exception UndefinierteBank();

  boolean istGueltigeBankleitzahl(in Bankleitzahl bankleitzahl);
  boolean istGueltigerBankname(in string bankname);

  boolean istGueltigeBank(in Bankleitzahl bankleitzahl,
                          in string bankname);
  string sucheBankleitzahl(in string bankname) raises (UndefinierteBank);
  string  sucheBanknamen(in Bankleitzahl bankleitzahl) raises
```

```
                        (UndefinierteBank);
  BankenListe sucheBankenMitBanknamenMuster(in string banknamenMuster);
  BankenListe sucheBankenMitBankenMusterUndStadtMuster(
                          in string banknamenMuster,
                          in string stadtMuster);
};
```

A.3 Spezifikation der Verhaltens-Ebene

Vorbemerkung: Hierbei handelt es sich z. Z. um „Pseudo-OCL“.

```
Bankleitzahlen::sucheBankleitzahl(name : bankname) : blz
  pre : name ist ein gültiger Bankname

Bankleitzahlen::sucheBanknamen(blz : bankleitzahl) : bankname
  pre : blz ist eine gültige Bankleitzahl

Bankleitzahlen::sucheBankenMitBanknamenMuster(name : banknamenMuster) :
bankListe
  post : result = leere Menge, wenn name ungueltig ist (Suche ist link-
strunkierend)

Bankleitzahlen::sucheBankenMitBanknamenMusterUndStadtMuster(name : bankna-
menMuster, stadt : stadtMuster) : bankListe
  post : result = leere Menge, wenn
          (name  ungueltig ist) oder
          (stadt ungueltig ist) oder
          (keine Bank name in der Stadt stadt existiert (Suche ist link-
strunkierend))
```

A.4 Spezifikation der Intra-Komponenten-Abstimmungs-Ebene

Bei der Benutzung der Dienste der Fachkomponente sind keine Restriktionen zu beachten.

A.5 Spezifikation der Inter-Komponenten-Abstimmungs-Ebene

Bei der Benutzung der Dienste der Fachkomponente sind keine Restriktionen zu beachten.

A.6 Spezifikation der Qualitäts-Ebene

Ausgangspunkt für die Messung der Qualitätseigenschaften ist folgende Systemumgebung:

- Prozessorarchitektur: Intel Pentium III 866 MHz
- Hauptspeicher: 1 GB RAM
- Windows NT 4.0
- Oracle 8i
- Brokat Twister 2.3.5

Unter dieser Systemumgebung können die Dienste der Komponente folgendermaßen spezifiziert werden. Es wird der Dienst „istGueltigeBankleitzahl“ spezifiziert. Als Eingabedaten werden zufällig ermittelte, gleichverteilte Werte verwendet.

Qualitätseigenschaft	Spezifikation
Durchsatzzeit	Der Durchsatz beträgt 1,8 s bei einer Arbeitslast von 1000 Dienstanforderungen.
Antwortzeit	Die Antwortzeit beträgt 18 ms.
Antwortzeitverhalten	Das Antwortzeitverhalten beträgt 0,0324 ms.
Verfügbarkeit	keine Angabe
Wiederanlaufzeit	keine Angabe

A.7 Spezifikation der Terminologie-Ebene

Wort	Definition
Bankleitzahl	Eine Bankleitzahl ist eine achtstellige Nummer, die als Kurzzeichen für ein Kreditinstitut mit eigenem Zentralbankgirokonto steht. Die Bankleitzahl dient ebenso als Kontonummer im Giroverkehr mit der Bundesbank. Synonyme: BLZ, Bankennummerierung
Bankleitzahl, gültig	Eine Bankleitzahl ist gültig, wenn ein entsprechender Datensatz in der Datei „blz0010pc.txt“ (Quelle: Deutsche Bundesbank, http://www.bundesbank.de/) mit demselben Wert im Datenfeld „Bankleitzahl“ existiert.
Bankname	Bezeichnung für ein Kreditinstitut gemäß der Felder „Verkürzte Bezeichnung der Kreditinstitutsniederlassung“ oder „Kurzbezeichnung der Kreditinstitutsniederlassung“ in der Datei SATZ188.doc. (Quelle: Deutsche Bundesbank, http://www.bundesbank.de/)
Bankname, gültig	Ein Bankname ist gültig, wenn ein entsprechender Datensatz in der Datei „blz0010pc.txt“ (Quelle: Deutsche Bundesbank, http://www.bundesbank.de/) mit demselben Wert im Datenfeld „Verkürzte Bezeichnung der Kreditinstitutsniederlassung“ oder „Kurzbezeichnung der Kreditinstitusniederlassung“ existiert.

B Spezifikation der Fachkomponente Virtuelle Anlage

B.1 Spezifikation der Administrations-Ebene

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- specification of administration layer for component Virtual Investment
-->
<!DOCTYPE ADMINISTRATION_LAYER SYSTEM "dtd/administration_layer.dtd">

<ADMINISTRATION_LAYER>
    <TECHNICAL_ATTRIBUTES>
        <SIGNATURE>Komponente Virtuelles Anlageprodukt</SIGNATURE>
        <VERSION>V 1.0</VERSION>
        <BILL_OF_MATERIALS>
          <ARTIFACT NAME="Komponentenspezifikation_Virtual_Investment.pdf"
DESCRIPTION="Spezifikationsdatei der Komponente Investment."/>
```

```
          <ARTIFACT NAME="virtual_investment.jar" DESCRIPTION="Java-Class-
Dateien der Implementierung"/>
          <ARTIFACT NAME="virtual_investment.tws" DESCRIPTION="Komponenten-
IDL für Middlware Twister von Brokat"/>
          <ARTIFACT NAME="create_db.sql" DESCRIPTION="SQL-Script zur Er-
stellung der benötigten Datenbanktabellen."/>
          <ARTIFACT NAME="virtual_investment_tests.jar"
DESCRIPTION="Beispiel-Client-Implementierungen zur Überprüfung der Kompo-
nentenfunktionalität."/>

        </BILL_OF_MATERIALS>

      <SYSTEMCONFIGURATION>
          <PROCESSOR VALUE="x86"/>
              <MEMORY_SIZE>512</MEMORY_SIZE>
            <DISK_SIZE>10</DISK_SIZE>
            <OPERATING_SYSTEM VALUE="WinNT">
                <OPERATING_SYSTEM_VERSION>4.0</OPERATING_SYSTEM_VERSION>
            </OPERATING_SYSTEM>
            <DBMS>Oracle 8i</DBMS>
            <BUSINESS_COMPONENT_FRAMEWORK>Framework des COFIS-
Systems</BUSINESS_COMPONENT_FRAMEWORK>
            <COMPONENT_FRAMEWORK>Brokat Twister 2.3.5</COMPONENT_FRAMEWORK>
      </SYSTEMCONFIGURATION>

    </TECHNICAL_ATTRIBUTES>

    <BUSINESS_ATTRIBUTES>
            <BUSINESS_TASK>Gib alle derzeit vorhandenen virtuellen Anlagen
aus</BUSINESS_TASK>
            <BUSINESS_TASK>Gib alle neu getätigten virtuellen Anlagen
aus</BUSINESS_TASK>
            <BUSINESS_TASK>Gib alle aktiven virtuellen Anlagen
aus</BUSINESS_TASK>
            <BUSINESS_TASK>Gib alle abgelaufenen virtuellen Anlagen
aus</BUSINESS_TASK>
            <BUSINESS_TASK>Gib für jeden Kunden die bisher gezahlte Quel-
lensteuer für alle virtuellen Anlagen aus</BUSINESS_TASK>

            <BUSINESS_TASK>Zahle Zinsen für eine virtuelle Anla-
ge</BUSINESS_TASK>
            <BUSINESS_TASK>Setze neues Auslaufdatum für eine virtuelle An-
lage</BUSINESS_TASK>
            <BUSINESS_TASK>Beende eine aktive virtuelle Anla-
ge</BUSINESS_TASK>
            <BUSINESS_TASK>Verlängere eine aktive virtuelle Anla-
ge</BUSINESS_TASK>
            <BUSINESS_TASK>Bestätige den Vertrag für eine virtuelle Anla-
ge</BUSINESS_TASK>
            <BUSINESS_TASK>Lehne den Vertrag für eine virtuelle Anlage
ab</BUSINESS_TASK>

            <BUSINESS_TASK>Führe den Produkt Management Zyklus für virtuel-
le Anlagen einer Bank aus</BUSINESS_TASK>
            <BUSINESS_TASK>Nimm XML-Daten für virtuelle Anlagen entge-
gen</BUSINESS_TASK>

            <BRANCH_OF_ECONOMY>Kreditgewerbe (65)</BRANCH_OF_ECONOMY>
            <FUNCTIONAL_AREA>Finanzen</FUNCTIONAL_AREA>
            <USER_DEFINED_DESCRIPTION>Die Komponente stellt Funktionen ein-
facher virtueller Finanzprodukte zur Verfügung. Es werden alle benötigten
Funktionalitäten, die bei Vermittlung einfacher Anlageprodukte notwendig
```

```
sind (z.B. Kauf, Zinszahlung, usw.) bereitgestellt.
</USER_DEFINED_DESCRIPTION>

    </BUSINESS_ATTRIBUTES>

    <MISC_ATTRIBUTES>
            <MANUFACTURER>Technische Universität Chemnitz, Professur Wirt-
schaftsinformatik II, D-09107 Chemnitz</MANUFACTURER>
            <CONTACT>Peter Fettke, +49/371/531-4362, peter.fettke@isym.tu-
chemnitz.de</CONTACT>
            <TERMS_OF_LICENCE>Allgemeine Lizenzbedingungen liegen nicht
vor, sondern müssen jeweils im Einzelfall ausgehandelt wer-
den.</TERMS_OF_LICENCE>
            <TARGET_GROUP>Forschungseinrichtungen oder forschungsinteres-
sierte Industrieunternehmen.</TARGET_GROUP>

              <MISC>
                    <ATTRIBUTE_NAME>zusätzlich benötigte Funktionalität für
den Dienst 'execPMC'</ATTRIBUTE_NAME>
                <ATTRIBUTE_VALUE>IBM AFS-Client 3.5<ATTRIBUTE_VALUE>
            </MISC>
            <MISC>
                <ATTRIBUTE_NAME>Konfigurationshinweis</ATTRIBUTE_NAME>
                <ATTRIBUTE_VALUE>Verbinden eines Pfades: "/afs/tu-
chemnitz.de"<ATTRIBUTE_VALUE>
            </MISC>
    </MISC_ATTRIBUTES>
</ADMINISTRATION_LAYER>
```

B.2 Spezifikation der Syntax-Ebene

```
Interface VirtualInvestment
{
    typedef string XMLDataLocation;
    typedef string XMLData;

    struct Date
    {
        int day;
        int month;
        int year;
    }

    struct Customer
    {
        int customer-id;
        string name;
        string firstname;
    }

    struct Investment
    {
        int vi-Id;
        int vi-variation-id;
        int vi-contract-id;
        Customer customer;
        Date vi-begindate;
        Date vi-enddate;
        string state-of-investment;
        double amount;
```

```
        double withholding_tax;
        double interest;
    }

    sequence <Investment> Investments;

    exception writeError();
    exception readError();

    Investments getInvestments          () raises (readError);
    Investments getNewInvestments       () raises (readError);
    Investments getActiveInvestments    () raises (readError);
    Investments getAmortizedInvestments () raises (readError);

    double listWithholdingTaxes () raises (readError);

    void payInterest (in Investment: selectedInvestment) raises
                     (writeError);

    void setExpirationDate (in Investment: selectedInvestment, Date
                            expDate) raises (writeError);

    void cancelInvestment  (in Investment: selectedInvestment)
                                          raises (writeError);
    void prolongInvestment (in Investment: selectedInvestment)
                                          raises (writeError);
    void confirmInvestment (in Investment: selectedInvestment);
    void rejectInvestment  (in Investment: selectedInvestment);

    void executePMC (in XMLDataLocation: dataLocation)
                                          raises (writeError);

    boolean receiveXMLData (in XMLData: investmentXMLData);
};
```

B.3 Spezifikation der Verhaltens-Ebene

Vorbemerkung: Hierbei handelt es sich z. Z. um „Pseudo-OCL“.

```
VirtualInvestment::getInvestments () : Investments
    pre: VirtualInvestment::receiveXMLData (XMLData:
         newVirtInvestmentContract) wurde nicht ausgeführt
    post: Investments = empty
```

Wurde der Komponente bisher kein Vertrag über eine virtuelle Anlage via XML übermittelt, liefert getInvestments die leere Menge als Ergebnis zurück.

```
VirtualInvestment::getNewInvestments () : Investments
    pre: VirtualInvestment::receiveXMLData (XMLData:
         newVirtInvestmentContract) wurde erfolgreich ausgeführt
           and
           (VirtualInvestment::confirmInvestment (Investment:
            selectedInvestment)
            xor
            VirtualInvestment::rejectInvestment (Investment:
            selectedInvestment) )
            wurde noch nicht ausgeführt
    post: Investments contains selectedInvestment
```

Wurde ein Vertrag für eine Virtuelle Anlage übermittelt, jedoch noch nicht bestätigt oder abgelehnt, so ist dieser Vertrag in der Menge der zurückgelieferten Anlagen enthalten.

```
VirtualInvestment::getActiveInvestments () : Investments
    pre: VirtualInvestment::confirmInvestment (Investment:
          selectedInvestment) wurde ausgeführt
           and
               VirtualInvestment::cancelInvestment (Investment:
               selectedInvestment) wurde noch nicht ausgeführt
           or
               VirtualInvestment::setExpirationDate (Investment:
               selectedInvestment, Date expDate) wurde erfolgreich
               ausgeführt
               and
                   expDate >= actualDate
    post: Investments contains selectedInvestment
```

Wurde ein Vertrag bestätigt und noch nicht von Systemseite aus beendet (automatisch durch Beendigung am Laufzeitende oder manuell durch Setzen des Auslaufdatums auf einen Vergangenheitswert), so ist dieser Vertrag in der Menge der zurückgelieferten Anlagen enthalten.

```
VirtualInvestment::getAmortizedInvestments () : Investments
   pre: VirtualInvestment::cancelInvestment (Investment:
        selectedInvestment) wurde erfolgreich ausgeführt
          or
          VirtualInvestment::setExpirationDate (Investment:
          selectedInvestment, Date: expDate)
          and
              expDate < actualDate
   post: Investment contains selectedInvestment
```

Wurde ein Vertrag am Laufzeitende beendet oder das Auslaufdatum auf ein vergangenes Datum gesetzt, ist dieser Vertrag in der zurückgelieferten Menge enthalten.

```
VirtualInvestment::listWithholdingTaxes () : TaxAmount
   pre: VirtualInvestment::payInterest (Investment: selectedInvestment)
        wurde ausgeführt
          and
              Zinszahlung für den Kunden war zu versteuern
   post: TaxAmount > 0 für den Kunde
```

Wurde für einen Vertrag eine Zinszahlung durchgeführt, die zu versteuern war (Freibetrag überschritten), so wurde die entsprechende Quellensteuer abgeführt.

```
VirtualInvestment::payInterest (Investment: selectedInvestment)
    pre: Freibetrag des Kunden ist erreicht/überschritten
    post: VirtualInvestment::listWithholdingTaxes () : TaxAmount > 0
```

War der Freibetrag des Kunden vor der Zinszahlung erreicht bzw. überschritten, wird bei der Zinszahlung die entsprechende Quellensteuer abgeführt.

```
VirtualInvestment::setExpirationDate (Investment: selectedInvestment, Date:
expDate)
    pre: selectedInvestment.state-of-investment = active
           and
           expDate < actualDate
    post: selectedInvestment.state-of-investment = amortized
```

Wird ein vergangenes Auslaufdatum für eine aktive Anlage gesetzt, gilt die Anlage als abgelaufen.

```
VirtualInvestment::cancelInvestment (Investment: selectedInvestment)
    pre: selectedInvestment.state-of-investment = active
    post: selectedInvestment.state-of-investment = amortized
```

Durch den Aufruf des Dienstes ändert sich der Status einer Anlage von „aktiv“ auf „abgelaufen“.

```
VirtualInvestment:: prolongInvestment (Investment: selectedInvestment)
    pre: selectedInvestment.state-of-investment = active
           and
           selectedInvestment ist verlängerbar
    post: selectedInvestment.endDate = selectedInvestment.endDate +
            weitere Laufzeit
           and
           selectedInvestment.state-of-investment = active
```

Darf eine Anlage verlängert werden und ist sie noch nicht abgelaufen, wird ein neues Ablaufdatum für die Anlage gesetzt. Die Anlage bleibt weiter aktiv.

```
VirtualInvestment:: confirmInvestment (Investment: selectedInvestment)
    pre: selectedInvestment.state-of-investment = prefaced
    post: selectedInvestment.state-of-investment = active
```

Möchte der Kunde eine Anlage erwerben, wird der Vertrag nach der Bestätigung durch die Komponente geschlossen und die Anlage aktiv.

```
VirtualInvestment:: rejectInvestment (Investment: selectedInvestment)
    pre: selectedInvestment.state-of-investment = prefaced
    post: selectedInvestment.state-of-investment = rejected
```

Möchte der Kunde eine Anlage erwerben und wird der Vertrag durch den realen Finanzdienstleister abgelehnt, wird dies der Komponente mitgeteilt.

```
VirtualInvestment::executePMC (XMLDataLocation: dataLocation)
    pre: Investments: Investments_before_PMC =
            VirtualInvestment::getInvestments ()
    post: Investments: Investments_after_PMC =
            VirtualInvestment::getInvestments ()
            and
                Investments_before_PMC = Investments_after_PMC
```

Ausführen des Produkt Management Zyklus. Es wird lediglich ein Update der von der Komponente angebotenen Anlageprodukte durchgeführt, bestehende Verträge bleiben davon unberührt.

```
VirtualInvestment::receiveXMLData (XMLData: investmentXMLData) : IsSuccess-
ful
    post: IsSuccessful = true, wenn das erhaltene XML-Dokument erfolgreich
          verarbeitet werden konnte
```

B.4 Spezifikation der Intra-Komponenten-Abstimmungs-Ebene

```
VirtualInvestment

    intially executePMC (XMLDataLocation)
```

Zu Beginn der Verwendung der Komponente ist der Produkt Management Zyklus auszuführen, um die Konsistenz der Daten zu gewährleisten.

```
always (receiveXMLData (XMLInvestmentData)
    and
        XMLInvestmentData enthält den Vertrag zum Erwerb der Anlage
        selectedVirtInvestment)
    before
        (confirmInvestment (selectedVirtInvestment)
         xor
             rejectInvestment(selectedVirtInvestment))
```

Bevor ein Vertrag für eine Anlage von der Komponente entweder abgelehnt oder bestätigt werden kann, muß zuvor der entsprechende Kaufantrag des Kunden eingegangen sein.

```
always confirmInvestment (selectedVirtInvestment)
    before
        (cancelInvestment(selectedVirtInvestment) or
         setExpirationDate(selectedVirtInvestment, expDate) or
         payInterest(selectedVirtInvestment) or
         prologInvestment(selectedVirtInvestment))
```

Eine Anlage muß bestätigt und aktiv werden, bevor sie verlängert oder beendet werden kann, bzw. Zinsen für sie gezahlt werden kann.

B.5 Spezifikation der Inter-Komponenten-Abstimmungs-Ebene

VirtualInvestment

```
executePMC (XMLDataLocation) and receiveXMLData (XMLData)
implies
    Framework : parseInvestmentData (InvestmentXMLData)
```

Beim Ausführen des Produkt Management Zyklus und bei Erhalt von XML-Daten anderer Finanzdienstleister werden immer die Dienste des Frameworks zum Parsen der XML-Dokumente benötigt.

```
receiveXMLData (XMLInvestmentData)
always_past
    FDL-Bus : transmitInvestmentData (InvestmentXMLData)
```

Der jeweilige Dienst receiveXMLData wird durch die Komponente FDL-Bus entsprechend aufgerufen.

```
confirmInvestment (selectedVirtInvestment) or
rejectInvestment  (selectedVirtInvestment) or
prologInvestment  (selectedVirtInvestment) or
cancelInvestment  (selectedVirtInvestment) or
payInterest       (selectedVirtInvestment) or
setExpirationDate (selectedVirtInvestment, expDate)
implies
    Framework : createXMLDocument (InvestmentData)
    before
        FDL-Bus : transmitInvestmentData (InvestmentXMLData)
```

Die Dienste zum Beenden, Verlängern, Bestätigen, Ablehnen einer virtuellen Anlage bzw. der Zinszahlung benötigen jeweils die Dienste des Frameworks zum Erzeugen der XML-Dokumente und den FDL-Bus zum anschließenden Übertragen dieser.

B.6 Spezifikation der Qualtitäts-Ebene

Zur Spezifikation der Qualitäts-Ebene wird die in Abschnitt A.6 beschriebene Systemumgebung unterstellt. Es wird der Dienst „payInterest“ spezifiziert.

Qualitätseigenschaft	**Spezifikation**
Durchsatzzeit	Der Durchsatz beträgt 2,0 s bei einer Arbeitslast von 1000 Dienstanforderungen.
Antwortzeit	Die Antwortzeit beträgt 20 ms.
Antwortzeitverhalten	Das Antwortzeitverhalten beträgt 0,0438 ms.
Verfügbarkeit	k.A.
Wiederanlaufzeit	k.A.

B.7 Spezifikation der Terminologie-Ebene

Wort	**Definition**
Dienst, erfolgreich ausgeführt	Ein Dienst wird erfolgreich ausgeführt, wenn während der Ausführung keine Exception/Ausnahme auftritt.
DTD	Eine Definition eines XML-Dokumententyps, die festlegt, in welcher Reihenfolge Elemente auftreten dürfen, wie sie verschachtelt sind und wie oft sie auftreten dürfen. Alle gültigen Dokumententypdefinitionen für cofis.net sind unter der URL http://www.isym.tu-chemnitz.de/cofis.net/xml/ verfügbar. Synonyme: Dokumententypdefinition
Finanz-dienstleister-Bus	Software-Bus, der als zentrale Kommunikationskomponente die Interaktion zwischen beliebigen Finanzdienstleistern gewährleistet. Als Datenaustauschformat wird XML verwendet. Synonyme: FDL-Bus
Freibetrag	Betrag, bis zu dem steuerfrei Zinsen ausgezahlt werden. Zinszahlungen werden versteuert, wenn der Freibetrag des Kunden überschritten wurde.
Produkt-Management-Zyklus	Tätigkeiten, die im Rahmen der Wartung, Pflege und Inbetriebnahme von Produkten nötig sind. Konkret umfaßt dies die Auswahl des Angebotes von virtuellen Anlagen durch einen Finanzdienstleister, deren Repräsentation in XML-Dokumenten und das Verfügbarmachen der Anlageprodukte durch Einstellen in das System. Synonyme: Product Management Cycle, PMC
Quellensteuer	Betrag, der bei Versteuerung von Zinszahlungen anfällt.
Virtuelles Anlage-produkt	Ein virtuelles Anlageprodukt ist ein einfaches Finanzprodukt, das durch das System cofis.net vermarktet wird. Es besteht aus den Angaben: Anlagenummer, Variantennummer, Vertragsnummer, Startdatum, Enddatum, Anlagenhöhe, Anlagenwährung, Anlagenstatus, Kundennummer, Kundenname.

	Synonyme: Virtuelle Anlage, Investment, virtual Investment
Virtuelles Anlageprodukt, Status	Eine virtuelle Anlage besitzt den Status *prefaced*, wenn sie vom Kunden erworben wurde, von der Bank jedoch noch nicht bestätigt oder abgelehnt wurde. Eine virtuelle Anlage besitzt den Status *active*, wenn die vom Kunden erworbene Anlage von der Bank bestätigt wurde und sie nicht abgelaufen ist. Eine virtuelle Anlage besitzt den Status *rejected*, wenn die vom Kunden erworbene Anlage von der Bank abgelehnt wurde. Eine virtuelle Anlage besitzt den Status *amortized*, wenn sie von der Bank beendet wurde, wenn das Ablaufdatum überschritten wurde und die Anlage nicht verlängerbar ist, wenn das Ablaufdatum überschritten wurde und die Anlage verlängerbar ist, vom Kunden oder der Bank aber nicht verlängert wurde.
XML-Dokument	Ein XML-Dokument ist ein Dokument, welches gemäß der Spezifikation der Extensible Markup Language (XML) der W3C erstellt wurde. Synonyme: XML-File, XML-Daten
XML-Dokument, gültig	Ein gültiges XML-Dokument ist ein XML-Dokument, das anhand einer DTD erfolgreich auf seine syntaktische Richtigkeit geprüft werden kann.

Literatur

Ackermann, J.; Brinkop, F.; Conrad, S.; Fettke, P.; Frick, A.; Klein, U.; Kotlar, O.; Loos, P.; Ortner, E.; Sahm, S.; Schmietendorf, A.; Turowski, K.: Memorandum zur Vereinheitlichung der Spezifikation von Fachkomponenten (Version vom 16.7.2001). München 2001.

Fettke, P.; Loos, P.; Thießen, F.; Zwicker, J.: Modell eines virtuellen Finanzdienstleisters: Der Forschungsprototyp cofis.net 1. Chemnitz 2001.

Stockmann, C.: Die virtuelle Bank - eine Begriffserklärung. In: C. Weinhardt; H. Meyer zu Selhausen; M. Morlock (Hrsg.): Informationssysteme in der Finanzwirtschaft. Berlin et al. 1998, S. 91-104.

Swoboda, U.: Quantensprung in der deutschen Kreditwirtschaft. In: U. Swoboda (Hrsg.): Direct Banking - Wie virtuelle Institute das Bankgeschäft revolutionieren. Wiesbaden 2000, S. 43-52.

Turowski, K.: Fachkomponenten - Komponentenbasierte betriebliche Anwendungssysteme. Habil.-Schr., Magdeburg 2001.