Reihe: Telekommunikation und Mediendienste · Band 2

Herausgegeben von Norbert Szyperski, Udo Winand, Dietrich Seibt und Rainer Kuhlen

Martin Engelien/Kai Bender (Hrsg.)

GeNeMe98

Gemeinschaften in Neuen Medien

TU Dresden, 1./2.10.1998

JOSEF EUL VERLAG

Lohmar · Köln

Reihe: Telekommunikation und Mediendienste · Band 2

Herausgegeben von Prof. Dr. Dr. h. c. Norbert Szyperski, Köln, Prof. Dr. Udo Winand, Kassel, Prof. Dr. Dietrich Seibt, Köln, und Prof. Dr. Rainer Kuhlen, Konstanz

Doz. Dr.-Ing. habil. Martin Engelien
Dipl.-Inf. (FH) Kai Bender (Hrsg.)

GeNeMe98

Gemeinschaften in Neuen Medien

TU Dresden, 1./2.10.1998

JOSEF EUL VERLAG
Lohmar · Köln

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

GeNeMe <1998, Dresden>:
GeNeMe 98 : Gemeinschaften in neuen Medien / Technische Universität Dresden, Fakultät Informatik, Institut für Informationssysteme, Dozentur „Entwurfmethoden und Werkzeuge für Anwendugssysteme". Martin Engelien; Kai Bender (Hrsg.). – Lohmar ; Köln : Eul, 1998.
(Reihe: Telekommunikation und Mediendienste ; Bd. 2)
ISBN 3-89012-632-4

Josef Eul Verlag GmbH
Brandsberg 6
53797 Lohmar
Tel.: 0 22 05 / 91 08 91
Fax: 0 22 05 / 91 08 92
e-mail: eul.verlag.gmbh@t-online.de

Printed in Germany
Druck: Rosch-Buch, Scheßlitz

Gedruckt auf säurefreiem und 100% chlorfrei gebleichtem Papier

Technische Universität Dresden
Fakultät Informatik • Institut für Informationssysteme
Dozentur „Entwurfsmethoden und Werkzeuge für Anwendungssysteme"

Doz. Dr.-Ing. habil. Martin Engelien
Dipl.-Inf. (FH) Kai Bender
(Hrsg.)

Dresden, 1./2.10.1998

GeNeMe98

Gemeinschaften in Neuen Medien

Workshop zu Organisation, Kooperation und Kommunikation auf der Basis innovativer Technologien

Forum für den Dialog zwischen Wissenschaft und Praxis zur Inversion der Virtualität (Ubiquitous Computing)

unter der Schirmherrschaft von:

Dr. W. Vehse
Staatssekretär für Wirtschaft
des Landes Sachsen

Prof. Dr. A. Mehlhorn
Rektor der TU Dresden

sowie unter Mitwirkung der
GI-Regionalgruppe Dresden

und mit freundlicher Unterstützung folgender Partner:

IST priv. Institut für angewandte Software-Technologie GmbH, Dresden
eine Ausgründung der TU Dresden auf dem Gebiet der Technologien und Anwendungen in den Neuen Medien

Heyde AG,
Bad Nauheim/ Dresden
Beratung • Software • Integration

B.4. Komponentenarchitektur für verteilte Systeme

Prof. Dr. J. Raasch
Fachhochschule Hamburg

Abstract

An der Fachhochschule Hamburg führen wir in der Informatik seit einigen Semestern das Projekt „SEVERS – Software-Engineering für die Versicherungswirtschaft" im Ausbildungskontext durch. Nach Betrachtung der im Rahmen des GDV (Gesamtverband der Deutschen Versicherungswirtschaft) formulierten Versicherungs-Anwendungs-Architektur (VAA) entstand im Projekt eine eigene SEVERS-Anwendungsarchitektur, die einige softwaretechnische Probleme der VAA vermeidet. Folgende Hauptziele wurden verfolgt:

- Information-Hiding als zentrales Konstruktionsprinzip,
- Verifizierbare und zertifizierbare Komponenten,
- Entwurf einer Migrationsstrategie,
- Verwendung aktueller Entwicklungsmethoden und Technologien für neue Komponenten.

In diesem Beitrag wird die SEVERS-Architektur kurz vorgestellt und es werden einige Fragen diskutiert, die mit der Anwendungsentwicklung verteilter, komponentenbasierter Anwendungen und mit der Einbettung solcher Systeme in Informationsmärkte zu tun haben.

Die Versicherungswirtschaft liefert für die hier geschilderte Architekturentwicklung den Anlaß und den ersten Anwendungskontext. Die vorgestellte Architektur ist aber übertragbar auf andere, ähnlich gelagerte Anwendungsfelder und damit verallgemeinerbar.

1 Ausgangspunkt

Die Initiative einiger Versicherungsunternehmen innerhalb des GDV (Gesamtverband der Deutschen Versicherungswirtschaft e.V.) führte zur Formulierung der VAA - Versicherungs-Anwendungs-Architektur [4]. Diese beabsichtigt, einen softwaretechnischen Rahmen für die Versicherungsanwendungen der Zukunft zu setzen und dabei einen Komponentenmarkt zu etablieren.

VAA geht von folgenden Konstruktionsregeln aus:

- Es gibt eine Trennung von Daten, Funktionen und Kontrolle [7, p.20].

- Komponenten dürfen keine anderen Komponenten aufrufen [6, p.44]. Statt dessen ist es nur erlaubt, über einen Datenmanager direkt auf Datenbestände anderer Komponenten zuzugreifen.

Mit diesen Regeln wird das Geheimnisprinzip [18] unterlaufen. Systeme können entstehen, die genauso wartungs- und pflegeintensiv sind wie jene Altsysteme, die es mittelfristig abzulösen gilt. Die dringend erforderliche Flexibilisierung der Anwendungen wird nicht erreicht.

Aus der Beobachtung dieser informationstechnischen Defizite entstand die Motivation für die Formulierung einer eigenen komponentenbasierten Anwendungsarchitektur. Die hier beschriebene Architekturentwicklung wird von unserer Arbeitsgruppe im SEVERS-Projekt („Software-Engineering für die Versicherungswirtschaft", vgl. [9]) an der FH Hamburg getragen. Diese verfolgt das vorrangige Ziel, durch exemplarische Vertiefung des Projektstudiums in Kooperation mit Wirtschaftsunternehmen praxisrelevante Qualifikationsmöglichkeiten für Studierende zu eröffnen.

2 Begriffe

Die folgenden Begriffsabgrenzungen fordern möglichst wenig einschränkende Eigenschaften. Deswegen sind unsere Schlußfolgerungen auch für jene relevant, die engere Begriffsdefinitionen bevorzugen.

2.1 System

Ein System ist eine Menge von Elementen zusammen mit ihren Beziehungen untereinander [1, p.11]. Ein System liefert eine Aufteilung der Realität in zwei Bereiche: das Systeminnere und die Systemumgebung. Aus der Realität wird ein Teil zur Untersuchung bzw. Gestaltung herausgegriffen. Die Abgrenzung zwischen System und Umgebung muß entscheidbar sein, was im Anwendungskontext normalerweise kein Problem ist.

2.2 Modul

Ein Modul ist ein System, das den Grundsatz des Information-Hiding [18] einhält: Dienstleistungen (services) des Systems sind für die Umgebung nur über explizite Schnittstellen verfügbar. Ein Nutzer des Moduls benötigt lediglich eine auf die Nutzung ausgerichtete Schnittstellen-Sicht [2], [14].

Schnittstellen von Moduln haben den Charakter von Verträgen: in der Vorbedingung (Precondition) einer Schnittstelle werden vollständig und eindeutig alle Bedingungen genannt, die der Nutzer erfüllen muß, um in den Genuß der Leistungen des Moduls zu

kommen. In der Nachbedingung (Postcondition) werden alle Bedingungen genannt, deren Erfülltsein der Modul garantiert, wenn vom Nutzer die Precondition eingehalten wurde [11], [13], [14].

2.3 Komponente

Eine Architektur ist gegeben durch eine Zerlegung eines Systems in Moduln, wobei diese Moduln miteinander ausschließlich explizit über die vorgesehenen Schnittstellen kommunizieren [9]. Diese Moduln bezeichnen wir als Architekturkomponenten.
Eine Architekturkomponente spielt in der Architektur eines Systems eine semantisch hervorgehobene Rolle. Daneben werden Entwurfskomponenten betrachtet. Eine Entwurfskomponente ist ein Modul, der mit dem Ziel entworfen wird, als Architekturkomponente genutzt zu werden. Meist ist einfach von Komponenten die Rede. Eine Komponente ist insbesondere ein Modul.
Von Komponenten werden Eigenschaften gefordert [17, p.34ff], die auch unmittelbare Bedeutung für die Gestaltung von Referenzarchitekturen haben. Ohne diese Eigenschaften von Komponenten ist die Vision eines Komponentenmarktes nicht realisierbar. Hier werden nur wenige besonders wichtige Eigenschaften aufgezählt:

Semantische Einheit
Eine Komponente muß ihre gesamte Semantik kapseln. Komponenten erzeugen semantisch vollständige Ergebnisse, keine Halbfertigprodukte.

Unabhängigkeit
Komponenten sind als Moduln voneinander unabhängig, d.h. austauschbar bei gleicher Schnittstelle.

Verifizierbarkeit
Verträge zur Spezifikation der Schnittstellen müssen so verbindlich und eindeutig formuliert sein, daß eine Verifikation von angebotenen Komponenten möglich ist. Die Schnittstellenspezifikation muß ausreichen, um die Korrektheit einer Komponentenimplementation im Hinblick auf diese Spezifikation zu beweisen.

Zertifizierbarkeit
Komponenten sind die Einheit für die Zertifizierung. Es muß eine Kontrollinstanz geben, die angebotene Komponenten prüft und inhaltlich und softwaretechnisch abnimmt. Eine solche Kontrollinstanz kann ihre Aufgabe nur wahrnehmen, wenn ihre Prüfobjekte auf dem Geheimnisprinzip basieren, also Komponenten sind.

3 Komponentenbasierte Anwendungsarchitektur

Ein wichtiger Ausgangspunkt für die Neufassung operativer Systeme ist die Rekonstruktion der für das Unternehmen relevanten Geschäftsprozesse. Die in der Geschäftsprozeßmodellierung erkannten Regeln werden als Steuerungswissen dem Workflowmanagement bereitgestellt. Jede Anwendungskomponente muß ihre gesamte Semantik kapseln, dazu gehört auch ihre eigene Vorgangssteuerung, in der ihr Anteil am Workflowmanagement implementiert wird. Also wird das Regelwerk des Workflowmanagements zu einem erheblichen Anteil auf die Komponenten verteilt.
Semantisch werden die Anwendungskomponenten durch die Geschäftsprozeßorientierung und durch die Rekonstruktion der Begriffswelt des Anwenders motiviert.
Technisch wird unsere Komponentenarchitektur durch den Aufbau der Anwendungskomponenten und einen Styleguide für die Schnittstellen dieser Anwendungskomponenten, vergleichbar den Styleguides für grafische Benutzungsschnittstellen (GUI), charakterisiert.
Die Entwicklung der SEVERS-Architektur erfolgt in den Schritten:

- Formulierung eines Styleguide für die Schnittstellen von Anwendungskomponenten,
- Spezifikation der für eine Versicherungsanwendung erforderlichen Anwendungskomponenten unter Beachtung des Styleguides,
- Prototypische Realisierung dieser Anwendungskomponenten.

Diese Schritte werden in mehreren Evolutionszyklen durchgeführt, um die Erfahrungen mit realisierten Prototypen im Styleguide berücksichtigen zu können.

4 Geschäftsprozesse

Operative Systeme reagieren auf Ereignisse, die in der Umgebung auftreten. An das System herangetragene Aufträge werden im Rahmen von Geschäftsprozessen bearbeitet. Ergebnisse gehen als Antworten des Systems an die Umgebung.
Geschäftsprozesse sind meist so kompliziert und vielgestaltig, daß eine elementare, flache Darstellung nur wenig sinnvoll ist. Geschäftsprozesse werden besser in einer Baumstruktur (Bild 1) beschrieben: Wurzelknoten ist der Geschäftsprozeß, Teilprozesse bilden die Knoten, die Blätter repräsentieren einzelne atomare Arbeitsschritte (Funktionen).

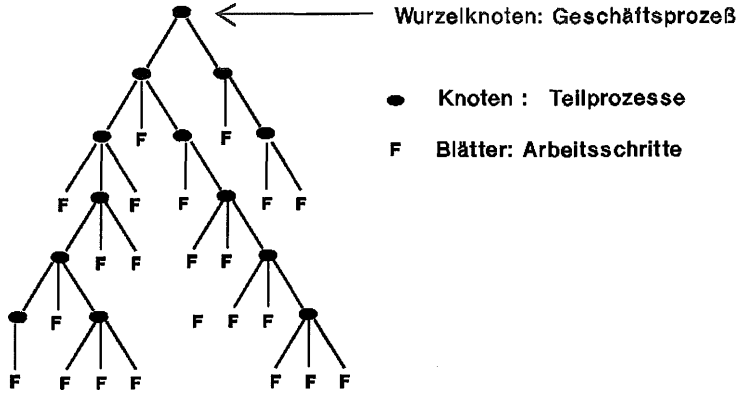

Bild 1: Baumstruktur von Geschäftsprozessen

Für die Teilprozesse und Arbeitsschritte innerhalb eines Geschäftsprozesses gelten zahlreiche Randbedingungen (Constraints) logischer und zeitlicher Art. Zum Beispiel erfolgt die Vertragsausfertigung immer logisch und zeitlich erst nach der Risikoprüfung, nur existierende Verträge darf man kündigen, eine Mahnung wird vielleicht erst drei Wochen nach Rechnungsstellung versandt.

Das Workflowmanagement hat die Aufgabe, im konkreten Ablaufgeschehen die Einhaltung der Constraints zu überwachen, die auch eine Interpretation als Konsistenzregeln besitzen. Es verfügt daher über die Repräsentation der Geschäftsprozesse sowie über alle für die Geschäftsprozesse relevanten Constraints.

5 Definition der Anwendungsarchitektur

Die Bausteine der Anwendungsarchitektur sind die Anwendungskomponenten. Diese unterstützen das Geheimnisprinzip [18].

5.1 Aufbau der Anwendungskomponenten

Gegenüber Nutzern von Anwendungskomponenten ist nur die Komponentenschnittstelle sichtbar, die einem strengen Styleguide folgt. Eine Anwendungskomponente verbirgt auch die zur Persistenz der eigenen Objekte erforderliche Datenbank, mit der Komponentenarchitektur gibt es keine globale Datenbank mehr. Die Anwendungskomponenten sind intern aus mehreren Subkomponenten zusammengesetzt.

Orientiert an den für Informationssysteme üblichen Architekturmustern [10, p.56f], [16, p.117ff] wurde die im Bild 2 gezeigte Aufbaustruktur abgeleitet.

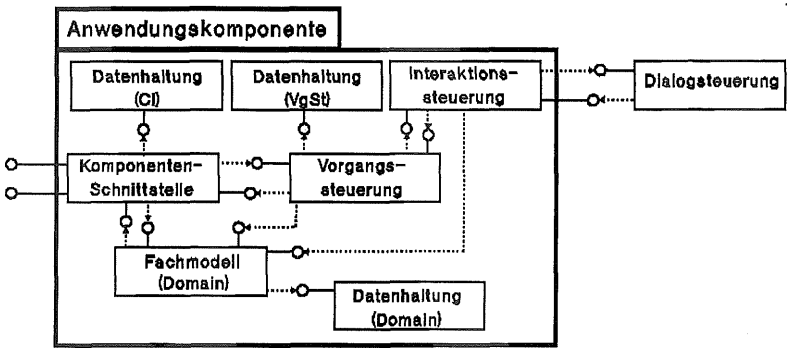

Bild 2: Aufbau von Anwendungskomponenten

Die Subkomponenten übernehmen jeweils bestimmte Aufgaben, nämlich

Fachmodell (Domain)
Die Objekte des Anwenders werden im Fachmodell realisiert. Die Systementwicklung geht von einem genauen Verständnis des Aufgabenbereichs aus. Um dieses zu erreichen, ist eine „menschenbezogene Rekonstruktion der Begriffswelt des Anwenders" [12] erforderlich.

Komponentenschnittstelle
Die Komponentenschnittstelle enthält sämtliche Export- und Importschnittstellen der Anwendungskomponente und abstrahiert damit sowohl die Leistungen für andere als auch die Leistungen, die von anderen erbracht werden müssen.

Vorgangssteuerung
Die Vorgangssteuerung enthält den Teil des Workflowmanagements, der für die Anwendungskomponente relevant ist, sowohl die Repräsentation der Geschäftsprozesse, als auch die logischen und zeitlichen Constraints.

Interaktionssteuerung
Die Interaktionssteuerung realisiert die Ablaufsteuerung für die Dialoge mit Benutzern.

Dialogsteuerung
Die Dialogsteuerung enthält alle Darstellungsformate der Benutzungsschnittstelle. Die Anwendungskomponente stellt neben einer Standardversion der Dialogsteuerung eine

öffentliche Schnittstelle zur Dialogsteuerung bereit, so daß etwa firmenspezifische Anpassungen möglich werden.

Datenhaltungen
Anwendungskomponenten kapseln ihre Datenbank. Darüberhinaus sind auch die Datenhaltungen der Teilkomponenten der Anwendungskomponente voneinander getrennt. Die Zugriffskompetenz wird auch innerhalb der Anwendungskomponenten genau eingehalten.

5.2 Komponentenobjekte

Eine Anwendungskomponente hat die Aufgabe, einen semantisch eng zusammenhängenden Ausschnitt des fachlichen Unternehmensmodells, also mehrere eng kollaborierende Klassen, zu kapseln. Der Zusammengehörigkeit der Elemente einer Anwendungskomponente wird meistens dadurch konkretisiert, daß bestimmte Objekte innerhalb der Anwendungskomponente verwaltet werden, deren Details verborgen, deren Existenz und Integrität jedoch nach außen hin vertraglich garantiert wird und daher referenziert werden kann.
Zum Beispiel ist es Aufgabe einer Partnerkomponente, Personen mit ihren sämtlichen Rollen, die sie für das Versicherungsunternehmen spielen können, mit allen zugehörigen Informationen und Verarbeitungsformen zu kapseln und für Nutzer über Schnittstellen referenzierbar zu machen. Entsprechendes gilt für die Vertragskomponente, die den Vertragsbegriff kapselt oder etwa für die Schaden/Leistung-Komponente, die einzelne Schadensfälle mit allen Details kapselt.
Daher resultiert der Begriff des Komponentenobjekts. Komponentenobjekte gehören einer Klasse innerhalb der Komponente an und sind nach außen hin persistent referenzierbar. Dabei muß die globale Konsistenz aller lokalen Datenbanken sichergestellt werden. Zur extern sichtbaren Existenz und zur referentiellen Integrität von Komponentenobjekten gibt es einige Grundmuster, nach denen die Erzeugung, Änderung und Löschung erfolgt.

5.3 Styleguide für Komponentenschnittstellen

Im Styleguide für die Anwendungskomponenten-Schnittstellen wird festgelegt, welche Dienste die einzelnen Komponenten anbieten dürfen. Diese Schnittstellen haben eine vordefinierte Semantik. Umgekehrt dürfen keine Schnittstellen eingerichtet werden, die nicht durch den Styleguide legitimiert sind. Durch Metaprogrammierung wird der Styleguide frei von der Semantik der Anwendungskomponenten formuliert.
Im Folgenden werden einige Interfaces beispielhaft angedeutet.

5.3.1 Interface: Erzeugen und Löschen von Komponentenobjekten

Die Existenz von Komponentenobjekten wird extern gesteuert und die referentielle Integrität wird über Grenzen von Anwendungskomponenten hinweg sichergestellt. Aufgrund externer Aufträge werden Komponentenobjekte eingerichtet, wobei eine persistente Referenz dem Auftraggeber zurückgegeben wird. Je nach Anwendungssemantik sind folgende Situationen zu unterscheiden:

Autonom einrichten
Autonome Komponentenobjekte werden eingerichtet, ohne mit Komponentenobjekten anderer Anwendungskomponenten verbunden sein zu müssen. Beispiel ist etwa ein Gutachter für Schäden in der KFZ-Versicherung. Er ist zu speichern, auch ohne daß er bereits Schäden begutachtet hat. Ein weiteres Beispiel ist das Versicherungsprodukt, das durch einen Geschäftsprozeß eingerichtet wird, wobei seine Existenz nicht von der Existenz anderer Komponentenobjekte abhängig ist.
Ein Löschen autonomer Komponentenobjekte ist nur zulässig, wenn das Komponentenobjekt nicht mehr referenziert wird (restriktiv).

Existenzabhängig einrichten
Existenzabhängige Komponentenobjekte werden auf Veranlassung von Komponentenobjekten anderer Anwendungskomponenten eingerichtet und sind von diesen existentiell abhängig. Beispiel ist der Versicherungsnehmer, der bei Einrichtung eines Vertrages etabliert wird und in seiner Existenz abhängig ist von dem Vertrag, in dem er referenziert wird.
Ein Löschen existenzabhängiger Komponentenobjekte ist nur durch den Eigentümer des Komponentenobjektes möglich, der es selber in der Vergangenheit eingerichtet hat (kaskadierend).

Verbinden mit existierendem Komponentenobjekt
Zu autonom existierenden Komponentenobjekten kann eine Verbindung (Referenz) hergestellt werden. Die Verbindung kann gelöscht werden, ohne daß das referenzierte, autonom existierende Komponentenobjekt gelöscht wird. Beispiel ist ein (autonom eingerichteter) Gutachter, der einen Schadensfall zu begutachten hat und deswegen durch das einzurichtende SchadenLeistung-Objekt referenziert wird. Eine Verbindung kann nur durch den Eigentümer gelöscht werden (akzeptierend).

5.3.2 Interface: Ausgabe von Komponentenobjekten

Eine Anwendungskomponente darf nicht die Kompetenz besitzen, in den Details einer anderen Komponente zu navigieren und sich eventuell benötigte Informationen selber zu suchen. Statt dessen werden entsprechende Aufträge an die andere Komponente gegeben, die jedoch keine Objektreferenzen herausgibt, sondern nur String-Objekte, die das jeweilige Komponentenobjekt extern beschreiben. Je nach inhaltlichen Erfordernissen dürfen hier zum Beispiel Formate für Adreßaufkleber oder Menüzeilen vereinbart werden.

5.3.3 Interface: Aufträge vom Workflowmanagement oder von anderen Komponenten

Zu jedem Auftrag, den eine Anwendungskomponente gemäß ihrer semantischen Spezifikation erfüllen können muß, gibt es eine entsprechende Schnittstelle der Anwendungskomponente. Die Vertragskomponente bietet zum Beispiel Dienste an, um Verträge einzurichten, zu ändern, aufzulösen. Die entsprechenden Aufträge beinhalten zumeist weitere Aufträge an andere Komponenten, etwa an die Partnerkomponente, einen Versicherungsnehmer einzurichten, oder an die Produktkomponente, eine Verbindung zu einem Tarif zu etablieren.

5.3.4 Import-Interface: von anderen Komponenten benötigte Leistungen

Auch Importschnittstellen (Zugriffe auf andere Anwendungskomponenten) werden explizit in der eigenen Schnittstellenklasse definiert. Wenn also etwa im Zuge einer Vertragseinrichtung ein Versicherungsnehmer benötigt wird, der vom Vertrag referenziert werden muß, dann ist ein Auftrag an die Partnerkomponente zur Einrichtung eines Versicherungsnehmers zu erteilen. Dies erfolgt jedoch über das Interface der Vertragskomponente und nicht etwa direkt durch Objektmethoden innerhalb der Vertragskomponente. Dadurch wird jede Anwendungskomponente vollständig durch ihr Interface repräsentiert, sowohl durch die Exportdienste (Leistungen für andere Anwendungskomponenten) als auch durch die Importdienste (Leistungen, die von anderen für die gegebene Anwendungskomponente erbracht werden müssen).

5.4 Geschäftsprozesse und Anwendungskomponenten

Durch die standardisierten Schnittstellen von Anwendungskomponenten, in denen unmittelbar ihre Semantik zum Ausdruck kommt, liefert unsere Komponentenarchitektur gleichzeitig einen Ausgangspunkt zur Formulierung einer Spezifikationssprache für Geschäftsprozesse: diese werden in der Terminologie der durch Anwendungskomponenten angebotenen Dienste spezifiziert. Im Rahmen der Geschäftsprozeßmodellierung ist es erforderlich, zu identifizieren, welche Dienste welcher Anwendungskomponente zu benutzen sind und wie der gesamte Geschäftsprozeß aus diesen Bausteinen zusammengesetzt ist.

Aufbauend auf dem Komponentenbegriff und dem Styleguide für Anwendungskomponenten werden die in Versicherungen benötigten Anwendungskomponenten spezifiziert und realisiert. Einige dieser Anwendungskomponenten werden im VAA-Konzept als Kernentities bezeichnet [5, p.16ff]: Partner, Vertrag, VersicherungsObjekt, Produkt, SchadenLeistung, InkassoExkasso, Provision,..

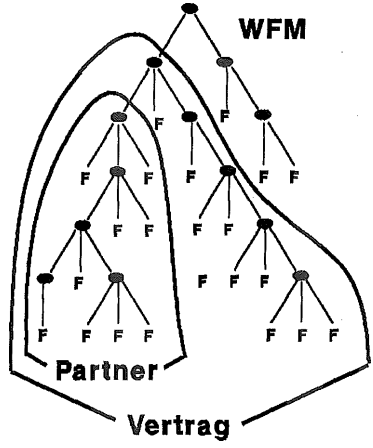

Bild 3: Anwendungskomponenten realisieren Äste des Geschäftsprozeß-Baumes

Geschäftsprozesse werden über den Workflowmanager, der die globale Ablaufsteuerung kapselt, gestartet. Die Bearbeitung erfolgt dann in der semantisch zuständigen Anwendungskomponente und wird entsprechend delegiert. Die Anwendungskomponente kapselt das gesamte Wissen, das zur Bearbeitung des Auftrags erforderlich ist. Daher ist es erforderlich, das Regelwerk des Workflowmanagers zum größeren Teil auf die Anwendungskomponenten zu verteilen und dort im Rahmen der sog. Vorgangssteuerung zu realisieren.

Teilprozesse eines Geschäftsprozesses können an andere Anwendungskomponenten *delegiert* werden. Diese sind dann für alle Aufgaben innerhalb des Teilprozesses verantwortlich und geben die durch die Schnittstelle garantierte Leistung zurück. Dabei werden aber immer nur ganze „Äste" der Geschäftsprozeß-Baumstruktur delegiert (Bild 3).

Wie zum Beispiel ein Vertrag eingerichtet wird, das ist in der Vertragskomponente zu kapseln. Die Vorgangssteuerung der Vertragskomponente verfügt über das entsprechende Wissen, welche Aufgaben an andere Anwendungskomponenten zu delegieren sind. Diese erledigen ihre Teilaufgabe vollständig, kommunizieren dabei eventuell mit dem Anwender. So erteilt die Vertragskomponente gegebenenfalls der Partnerkomponente den Auftrag, eine Versicherungsnehmer zu etablieren.

Das bis dato von VAA propagierte Vorgehen, nach dem Komponenten sich nicht gegenseitig aufrufen dürfen [6, p.44], würde dazu führen, daß keine Komponenten entstehen, die auf dem Geheimnisprinzip basieren. Denn das gesamte Ablaufwissen wäre außerhalb jeder Anwendungskomponente lokalisiert und damit von Daten und Funktionen getrennt. Änderungen wären nicht auf eine Komponente beschränkt, sondern würden auch Änderungen in der Komponente „Workflowmanager" erfordern, die getrennt von der Anwendungskomponente ist. Die Ablaufregeln wären ohne Daten und Funktionen unverständlich, Daten und auch Funktionen wären ohne die Ablaufregeln und Steuerungsbedingungen unverständlich.

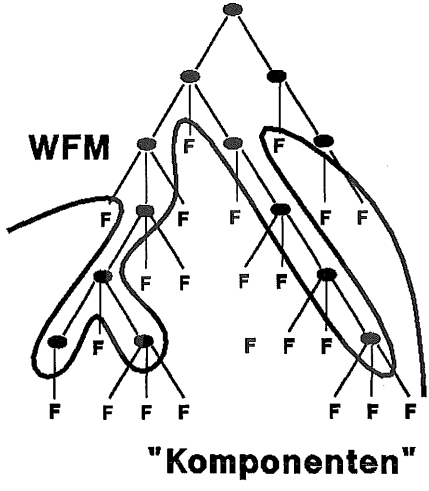

Bild 4: Trennung von Workflowmanager und Komponenten (VAA)

Das VAA-Konzept ist daher nur realisierbar mit einem Workflowmanager, der über das gesamte Ablaufwissen im Detail verfügt und der von den Komponenten nur die elementaren RUDI-Funktionen (read-update-delete-insert) aufruft (Bild 4). In VAA gibt es daher offenbar gar keine Komponenten, sondern nur eine globale Datenbank, deren physikalischer Aufbau durch einen Datenmanager [6] abstrahiert wird.

6 Transaktionsverarbeitung

Der Verzicht auf eine globale Datenbank bedeutet auch, daß die in Datenbanksystemen verfügbaren Mechanismen zur Einhaltung der globalen Konsistenz der Datenbasis nicht genutzt werden können. Die Kompliziertheit der Geschäftsprozesse und die Aufteilung auf Komponenten bringen es mit sich, daß das in Datenbanksystemen elementar verfügbare Konzept der einfachen (pessimistischen) Transaktionen nur lokal genutzt werden kann.

Geschäftsprozesse sind abzubilden auf Transaktionen. Für Geschäftsprozesse müssen nämlich die ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability) zugesichert werden. Reale Geschäftsprozesse sind jedoch meistens so kompliziert, daß sie nicht als einfache, flache Transaktionen abgewickelt werden können. Auch die Aufteilung des Gesamtsystems in Anwendungskomponenten mit jeweils lokalen, von außen unsichtbaren Datenbanken fordert eine Aufteilung der Geschäftsprozeß-Transaktionen.

Daher sind die Konzepte der langen und der geschachtelten Transaktionen zu implementieren. Dabei wird es jeder Anwendungskomponente überlassen, in ihrem Innern ein Transaktionsmanagement zu realisieren, das die Aufträge von außen (über das Interface: „Aufträge vom Workflowmanagement oder von anderen Komponenten") als Transaktionen behandelt.

Jede Anwendungskomponente, die einen Auftrag erteilt, also eine Transaktion veranlaßt, muß darüberhinaus alle Informationen zur Auftragserteilung aufzeichnen (Logfile), die gegebenenfalls ein geschachteltes Zurücksetzen ermöglichen. Damit kann auch in verteilten Systemen die ACID-Eigenschaft für Geschäftsprozesse zugesichert werden.

Insgesamt sind also folgende Regeln einzuhalten, um Transaktionsverarbeitung zu sichern:

- Jede Komponente behandelt sämtliche an sie gestellten Aufträge als Transaktionen.
- Jede Komponente protokolliert die Aufträge, die im Rahmen einer Transaktion an andere Komponenten gerichtet wurden.
- Der globale Workflowmanager ist auch eine Komponente. Er besitzt einen Transaktionsmonitor mit Logfile.

Im Falle etwa des Zurücksetzens einer Vertragseinrichtung wird also vom globalen Workflowmanager an die Vertragskomponente der Auftrag zum Zurücksetzen propagiert. In der Vertragskomponente kann dann lokal ein Zurücksetzen vorgenommen werden. Dabei werden alle Aufträge bekannt, die bei Einrichtung des Vertrags an andere Komponenten gerichtet wurden, z.B. der Auftrag an die Partnerkomponente zur Etablierung eines Versicherungsnehmers. Weil dieser Auftrag bekannt ist, kann die Partnerkomponente die Aufgabe propagiert bekommen, die entsprechende Transaktion zurückzusetzen. Bei lokalen Informationen geschieht dies in eigener Verantwortung. Werden Aufträge erkannt, die beim Einrichten an andere Komponenten erteilt wurden, so wird der Auftrag zum Zurücksetzen weiter propagiert.

7 Verteilung

Die aus der semantischen Architektur der Anwendungskomponenten abgeleitete technische Architektur soll beliebige Verteilungsmodelle in Mehrschichtenarchitekturen unterstützen. Daher werden Mechanismen bereitgestellt, um trotz Verteilung einer Anwendungskomponente auf mehrere Rechnerebenen (Client/Server oder multi-tier) den semantischen Zusammenhalt der Komponente und das Geheimnisprinzip sicherzustellen.

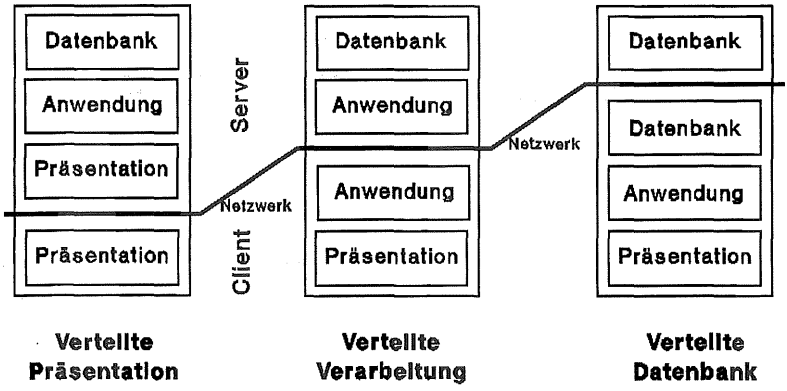

Bild 5: einige Verteilungsformen

Der Inhalt der Anwendungskomponenten ist semantisch motiviert: Jede Anwendungskomponente kapselt einen eng zusammengehörenden Teil des Unternehmensgesamtmodells, ergänzt um Vorgangssteuerung, Komponentenschnittstelle und Benutzungsschnittstelle (Interaktionssteuerung und Dialogsteuerung) und ist von tehnischen Verteilungskriterien unabhängig.

Zur Nutzung in einem verteilten System muß die Anwendungskomponente in einen Server-Teil und einen Client-Teil aufgespalten werden (eine Mehrschichten-Architektur erfordert natürlich entsprechend weitere Aufspaltungen). Durch die *semantische Architektur* wird zunächst als Verteilungsform (vgl. z.B. [15]) die verteilte Präsentation nahegelegt (vgl. Bild 2). Für praktisch einsetzbare Systeme sind dagegen meist andere Verteilungsformen günstiger (Bild 5).

Bestimmend für die *technische Architektur* der Anwendungskomponenten ist die Netzbelastung. Die Trennungslinie zwischen Aufgaben des Server- und des Client-Teils (Bild 6) der Anwendungskomponente muß also frei von semantischen Gesichtspunkten allein aufgrund technischer Kriterien festgelegt werden können.

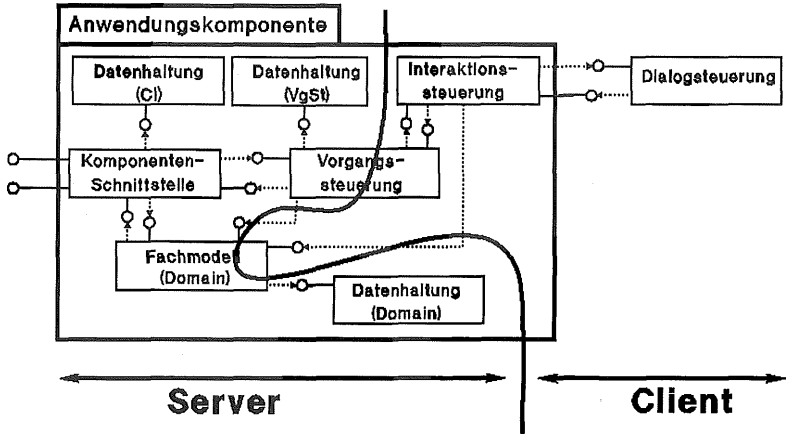

Bild 6: Trennung einer Anwendungskomponente in Server- und Client-Anteil

Dabei darf aber das Geheimnisprinzip auf Ebene der Anwendungskomponenten nicht unterlaufen werden. Also werden innerhalb der Komponenten weitere Schnittstellen entwickelt, die eine Kommunikation zwischen Client- und Server-Teil der Anwendungskomponente ermöglichen, aber jeden Seiten-Einstieg verbieten. Durch Vordefinition mehrerer solcher Schnittstellen wird zugleich eine freie Wahl der Lokalisierung von Subkomponenten der Anwendungskomponente auf Rechner in einer Mehrschichtenarchitektur ermöglicht.

Damit liegt eine Abgrenzung vor, die exemplarisch aufzeigt, welche Maßnahmen der Anwendungsarchitektur erforderlich werden, wenn man Frontends von größeren operativen Anwendungssystemen kundenfreundlich in Informationsmärkten anbieten will. Der Client-Anteil der Anwendung muß im Gesamtkontext entwickelt werden und

in eventuell mehreren konkurrierenden Architekturmustern implementiert und parallel unterschiedlichen Nutzerkreisen angeboten werden.

8 Anwendungsentwicklung für Komponenten

Es wird davon ausgegangen, daß Anwendungen im Rahmen der Architektur evolutionär entwickelt werden. Durch Kommunikation mit Benutzern werden die wesentlichen Geschäftsprozesse rekonstruiert, bevor Systemvisionen entwickelt und in Prototypen vergegenständlicht werden. Diese Prototypen, mit denen explorativ und experimentell die Tauglichkeit des Anwendungskonzeptes hergestellt und abgesichert wird, müssen nicht in verteilten Systemen ablaufen. Die Verteilung der Komponenten erfolgt später im Rahmen der technischen Realisierung. Die semantische Architektur soll von Verteilungsgesichtspunkten, die technische Architektur soll von semantischen Gesichtspunkten unabhängig sein.

Der Anwender denkt in Begriffen, die meistens sehr komplex sind und nur durch Zuhilfenahme zahlreicher Hilfskonstruktionen angemessen informationstechnisch repräsentiert werden können. Diese Hilfskonstruktionen haben aber für den Anwender keine besondere Bedeutung und sind ihm meistens prinzipiell unverständlich.

Zum Beispiel erfolgt die objektorientierte Modellierung eines Vertrages und seiner unterschiedlichen Zustände sinnvollerweise mit Hilfe des State-Patterns [3], das jedoch in seinen Einzelheiten dem Anwender meist nicht nahegebracht werden kann. Ihm reicht es, wenn das typische Zustandsverhalten realisiert wird. Noch deutlicher ist die Modellierung von Angeboten, Aufträgen, Verträgen und Rechnungen in einem Relationenmodell, das wegen Normalisierung eine Vielzahl einfacher Tabellen anbietet, die einzeln und in ihrem Zusammenwirken einem Anwender nur schwer erklärt werden können.

Die eher technisch motivierten Beschreibungssprachen zerlegen die komplexen Begriffe des Anwenders regelmäßig in einfachere Bestandteile, die technisch besser beherrschbar sind und modellieren dann die Beziehungen zwischen diesen Begriffen. Für die angemessene direkte Modellierung der Begriffe des Anwenders liefert jedoch unser Begriff der Anwendungskomponente eine Klammer: Den komplexen Domain-Begriffen wird nicht eine Klasse oder ein Entity zugeordnet, sondern eine Anwendungskomponente. Diese liefert dann wegen Lokalitäts- und Geheimnisprinzip einen begrenzten und überschaubaren Kontext, in dem, nach wie vor mit den Mitteln der objektorientierten Analyse, die Begriffe in allen Details analysiert und repräsentiert werden können. Für die Modellierung bringt dies den willkommenen Nebeneffekt, daß der Kontext der Repräsentation eines Fachbegriffs lokal und explizit ist.

9 Migration

Der Übergang auf eine neue Architektur kann eine Neufassung der gesamten Anwendungswelt erfordern. Nach unserem Architekturkonzept kann der ggf. dramatische Übergang entzerrt, zeitlich gestreckt und gemildert werden.

Durch die Regeln zum Aufbau von Anwendungskomponenten ist zunächst nicht festgelegt, welche Technologie im Inneren dieser Komponenten benutzt wird. Selbstverständlich wird man bei neu zu entwickelnden Komponenten aktuelle Technologie anwenden (Objektorientierung). Parallel zu neuen Anwendungskomponenten können ältere Systemteile weiterbenutzt werden, wenn sie durch geeignete Maßnahmen verborgen werden können (Wrapper-Techniken), so daß sie einem Nutzer gegenüber als Anwendungskomponente im hier präzisierten Sinn erscheinen.

Damit ergibt sich ein *evolutionärer* Weg der Migration: Versicherungsunternehmen sind nicht gezwungen, in einem Schritt auf eine neue Referenzarchitektur umzustellen. Statt dessen wird die real vorhandene Anwendungslandschaft intern schrittweise strukturell verbessert.

Ein Anwendungsunternehmen sollte zunächst seine Anwendungen überarbeiten und dabei die bedeutsamen Teilsysteme logisch separieren. Wenn dann die Teilsysteme mit Schnittstellen gemäß unserem Styleguide ausgestattet werden, dann entsteht für das Unternehmen einige Vorteile:

- Die Anwendungslandschaft wird besser überschaubar und damit planbar.
- Es entsteht die strategische Option zur Ersetzung veralteter Anwendungskomponenten durch neugestaltete, die etwa leistungsfähiger in Richtung Ergonomie oder Vernetzung sind, bei gleichbleibender logischer Schnittstelle und bei gleichbleibender Anwendungssemantik.
- Bewährte Teile der Altanwendungen können mit neuen Schnittstellen (Wrapper) weiterbenutzt werden, wobei sie einem Nutzer als neue Komponenten erscheinen.

10 Operative Verfahren und Electronic Commerce

Für Versicherungsunternehmen ist es wie für alle Betreiber großer, kundenorientierter, operativer Verfahren attraktiv, auf elektronischen Märkten vertreten zu sein. Dies setzt voraus, daß besonders klare Schnittstellen gegenüber dem Markt definiert werden. Im Folgenden wird dargestellt, welche Möglichkeiten sich mit unserer Komponentenarchitektur ergeben.

In Versicherungsunternehmen geht man in Ermangelung global verfügbarer Netzwerke bisher davon aus, daß Kunden ihre Anträge, Schadensmeldungen usw. in jedem Fall auf Dokumenten dem Versicherungsunternehmen zuleiten und als Antwort neben einer tele-

fonischen Auskunft auch wieder Dokumente erhalten. Durch die Verbreitung des Internet wird ein neuer Informationsweg nutzbar. Anwendungsbeispiele im Kontext der Versicherungen wären:

- Antragstellung
- Schadensmeldung
- Information über den aktuellen individuellen Versicherungsschutz
- Tarifüberschneidungen, Über- bzw. Unterversicherung
- Versicherungsberatung
- Verfolgung von Geschäftsprozessen mit Eingriffsmöglichkeiten durch den Kunden (Tracking)

Die ersten beiden Punkte bilden im Wesentlichen die (Papier-) Dokumentformate auf Internet-Formate ab und sind daher sicher als erster erforderlicher Schritt zu betrachten. Die anderen weiterführenden Dienste stellen dagegen für viele Versicherungsunternehmen noch eine Herausforderung dar.

Um derartige Dienste überhaupt extern sinnvoll anbieten zu können, müssen interne Voraussetzungen geschaffen werden. Auch ohne Blick auf internationale Vernetzung werden diese Voraussetzungen seit einigen Jahren in Versicherungsunternehmen diskutiert unter den Problemstellungen

- Übergang von der Spartenorientierung zur Kundenorientierung
- Neuere Anwendungen wie zentrale Auskunfterteilung (Telefon-Helpdesk)

Eine zentrale und vollständige Auskunfterteilung setzt voraus, daß die gesamte informationstechnische Infrastruktur kundenorientiert ausgelegt ist. Kundenorientierung erfordert, daß die einzigen spartenbezogenen Informationen bei der Produktdefinition (Tarif) gespeichert werden. Alle anderen Anwendungskomponenten müssen frei von Spartenwissen implementiert sein. Weiterhin ist ein integriertes Partnersystem erforderlich, das einen Zugriffspfad von der Person über ihre Rollen zu sämtlichen Geschäftsprozeß-Ergebnissen ermöglicht.

Wenn diese Voraussetzungen alle erfüllt sind, dann ist eine telefonische Auskunfterteilung möglich, aber auch die Entwicklung von Internet-basierten Dialogen.

Die Verfolgung des Bearbeitungsstandes von Geschäftsprozesssen schließlich setzt einen Workflowmanager voraus, der Zustandsinformation zu jedem in Bearbeitung befindlichen Geschäftsprozeß repräsentiert. Wenn diese Voraussetzung gegeben ist, dann ist es leicht, einen Internet-basierten Dialog bereitzustellen, mit dem sich ein Kunde die ihn selber betreffende Information beschaffen und womöglich sogar mit dem gerade zuständigen Sachbearbeiter in Kommunikation treten oder sonstwie Beiträge zur Erledigung des Geschäftsprozesses liefern kann.

Diese Beispiele verdeutlichen, daß die interne Systemarchitektur eine wesentliche Voraussetzung für ein Konzept der Informationsvernetzung darstellt, das über den Austausch von Papierdokumenten hinausgeht. Die hier vorgestellte Architektur erleichtert die Beteiligung eines Unternehmens an globalen Informationsvernetzungen.

11 Verallgemeinerung

Für unser Hochschulprojekt sind Versicherungsanwendungen Anlaß und Motivation für Fragestellungen und vorläufig wichtigster Anwendungsbereich. Es wird jedoch nicht angestrebt, die Architektur auf eine bestimmte Branche zu beschränken.
Unsere Architekturdefinition enthält keine versicherungsspezifischen Besonderheiten. Versicherungsdetails kommen nur in Motivationen und Beispielen vor. Daher ist die geschilderte Architektur übertragbar auf andere, ähnlich gelagerte Anwendungsfelder.

12 Zusammenfassung

Ein *Komponentenmarkt* setzt verifizierbare und zertifizierbare Komponenten und damit *Information-Hiding* voraus. Die *Schnittstellengestaltung* ist primäre Design-Aufgabe für die Architekturentwicklung.
Ausgangspunkt einer evolutionären *Migrationsstrategie* ist eine schrittweise Verbesserung der installierten und historisch gewachsenen Anwendungen, wodurch die strategische Option zur Einführung neuer Komponenten entsteht.
Neugefaßte Anwendungskomponenten werden selbstverständlich unter Nutzung *aktueller Entwicklungsmethoden und Technologien* entwickelt, z.B. Objektorientierung.
Der Auftritt eines Unternehmens auf *elektronischen Märkten* erfordert eine Einbindung der Kundeninteraktion in interne Abläufe. Dies kann zwar als erster Schritt mit einer Insellösung realisiert werden, jedoch ist mittelfristig eine Integration des Internet-Auftritts mit der internen Architektur anzustreben.
In diesen Punkten ist unser Konzept für eine komponentenbasierte Architektur von Informationssystemen hilfreich.

12.1 Weiteres Vorgehen

Die Entwicklung experimenteller und explorativer Prototypen wird eine Hauptaufgabe des Teilprojektes „Anwendungsarchitektur" des SEVERS-Projektes an der FH Hamburg bleiben. Daneben werden im Projekt weitere Themenbereiche untersucht, zum Beispiel die Datenauswertung (Data Warehouse, vgl. [8]) und die Verteilung und Vernetzung von Anwendungen.

12.2 Danksagung

An der Architekturentwicklung haben zahlreiche Personen mitgewirkt. Allen Beteiligten sei herzlich gedankt.
Mein besonderer Dank gilt Robert Aldrup (agens Consulting), Petra Becker, Prof. Helga Carls, Frank Dünnleder (PDV Unternehmensberatung), Martin Ersche, Prof. Dr. Erhard Fähnders, Prof. Jürgen Freytag, Prof. Dr. Wolfgang Gerken, Stefan Gertsobbe, Oliver Grampp, Gilles Iachelini, Andreas Kopka, Hartmut Kubesch (IDUNA-NOVA), Thorsten Lüders, Michael Mathé, Claus-Jürgen Moessinger (BERATA), Prof. Dr. Guido Pfeiffer, Volker Schrödter.

Literatur

[1] ed. Walter F. Daenzer: Systems Engineering - Leitfaden zur methodischen Durchführung umfangreicher Planungsvorhaben. Zürich: Verlag Industrielle Organisation, 1988 (6. Auflage).

[2] Ernst Denert: Software-Engineering. Berlin;.: Springer, 1991.

[3] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[4] Gesamtverband der Deutschen Versicherungswirtschaft: VAA - Die Anwendungsarchitektur der Versicherungswirtschaft. Dezember 1996 (2.Auflage).

[5] Fachliche Beschreibung, in [4].

[6] Technische Beschreibung, in [4].

[7] Fragen und Antworten, in [4].

[8] Wolfgang Gerken: Data Warehouse - Datengrab oder Informationspool? Versicherungswirtschaft 8/1997, 506-511

[9] Wolfgang Gerken, Jörg Raasch: VAA im Lichte der Objektorientierung. Versicherungswirtschaft, 8/1996, 492-498

[10] Erika Horn, Wolfgang Schubert: Objektorientierte Software-Konstruktion. Grundlagen - Modelle, Methoden - Beispiele. München; Wien: Hanser, 1993.

[11] Cliff B. Jones: Systematic Software Development Using VDM. Prentice-Hall, 1986.

[12] Klaus Kilberth, Guido Gryczan, Heinz Züllighoven: Objektorientierte Anwendungsentwicklung. Konzepte, Strategien, Erfahrungen. Braunschweig; Wiesbaden: Vieweg, 1994, 2. Auflage.

[13] Bertrand Meyer: Objektorientierte Softwareentwicklung. München; Wien: Hanser; London: Prentice-Hall, 1990. Prentice-Hall, 1988.

[14] M. Nagl: Softwaretechnik: Methodisches Programmieren im Großen. Springer, 1990.

[15] Klaus D. Niemann: Client/Server Architektur, Organisation und Methodik der Anwendungsentwicklung. Braunschweig; Wiesbaden: Vieweg, 1995.

[16] Bernd Oestereich: Objektorientierte Softwareentwicklung mit der Unified Method Language. München; Wien: Oldenbourg, 1997, 3. Auflage.

[17] Robert Orfali, Dan Harkey, Jeri Edwards: The Essential Distributed Objects Survival Guide. New York;..: John Wiley & Sons, 1996.

[18] David L. Parnas: On the Criteria to Be Used in Decomposing Systems into Modules. Communications of the ACM, vol. 5, no. 12, 1053-1058, 1972.