Endbericht der Projektgruppe

Electronic Commerce in der Versicherungsbranche

Beispielhafte Unterstützung verteilter Geschäftsprozesse

Leitung:

Prof. Dr. Volker Gruhn
Dipl.-Math. Dirk Peters
Dipl.-Inform. Ursula Wellen

Teilnehmer:

Thomas Bühren
Muharrem Cakir
Ercan Can
Andreas Dombrowski
Gereon Geist
Mustafa Gürgen
Sven Handschumacher
Markus Heller
Christian Lüer
Guy Vollmer
Jörg von Werne

Universität Dortmund
Fachbereich Informatik
Lehrstuhl für Software-Technologie
Baroper Straße 301
44227 Dortmund

Inhaltsverzeichnis

Vorwort

Die Rahmenbedingungen für unternehmerisches Handeln wurden in den letzten Jahren einem rasanten Entwicklungsprozeß unterworfen. Wachsende Investitionen, zunehmender Wettbewerbsdruck durch fallende Grenzen in Europa sowie die europäische Währungsunion haben die Situation derart zugespitzt, daß es für Unternehmen unabdingbar geworden ist, ihre Effektivität zu steigern, um bei gleichen Kosten mehr, bzw. besseres zu leisten und zu produzieren. Die zunehmende Verfügbarkeit von Telekommunikationsinfrastruktur führt dabei zum Einsatz neuer Informations-Technologien, um den zukünftigen Anforderungen gewachsen zu sein.

Insbesondere die Versicherungsbranche wird durch die harte Konkurrenzsituation im europäischen Binnenmarkt mobilisiert. Die langfristige Perspektive der deutschen Versicherungsbranche in einem deregulierten Versicherungsmarkt hängt zu einem nicht unwesentlichen Teil von der Flexibilität und Kreativität ab, mit der die einzelnen Unternehmen eine engere Kundenbindung über elektronische Wege realisieren. Die Versicherungsunternehmen benötigen daher verstärkt elektronische Lösungen, die auf einer Telekommunikationsinfrastruktur aufsetzen, um diese Bindung sicherzustellen. Das Internet als weltweiter Marktplatz mit prognostizierten 150 Milliarden Dollar Umsatz in diesem Jahr ist dabei das Medium der Zukunft. Über das Internet werden mittel- bis langfristig verteilte Geschäftsprozesse verschiedenster Art abgewickelt, indem neben der Produktwerbung und -bestellung sowie der umfangreichen Kundenwerbung und -betreuung auch der tatsächliche Ablauf eines Geschäftsprozesses unterstützt werden wird.

Diese informationstechnologischen Lösungen bieten die Möglichkeit, das häufig komplizierte Produktgeflecht von Policen, Verträgen und Versicherungen mit den unterschiedlichsten Tarifstrukturen kundengerechter und transparenter zu gestalten sowie das verteilte Erstellen und Bearbeiten solcher Dokumente zu ermöglichen.

Die Projektgruppe ***eCCo*** hat sich aus diesem Grunde zum Ziel gesetzt, zwei verteilte Geschäftsprozesse der Versicherungsbranche unter Einbeziehung der VAA (Versicherungs-Anwendungs-Architektur) beispielhaft zu unterstützen und darüberhinaus einen graphischen Editor für verteilte Geschäftsprozesse zu entwickeln.

Ferner war es erklärtes Ziel der Projektgruppe ***eCCo***, durch Kontakt zu Versicherungsunternehmen den notwendigen Bezug zur realen Welt herzustellen und entsprechendes Feedback aus der Praxis in den eigenen Software-Entwicklungsprozeß einfließen zu lassen. Um dieser Zielsetzung gerecht zu werden, hielt die Projektgruppe ***eCCo*** begleitend zur eigentlichen Software-Entwicklung Vorträge bei mehreren Versicherungsunternehmen (u.a. Württembergische-, Barmenia- und Aachen-Münchner-Versicherung). So konnten die erzielten Ergebnisse mit der objektorientierten Weiterentwicklung der VAA vorgestellt werden und Einblicke in die Praxis der Software-Entwicklung gewonnen werden. Zudem wurde im Rahmen der sich an die Vorträge anschließenden Gespräche der gesamte Entwicklungsprozeß der Projektgruppe ***eCCo*** einer eingehenden und kritischen Prüfung unterzogen.

Die Projektgruppe ***eCCo*** hat ihre Arbeit im Februar 1998 aufgenommen und präsentiert nachfolgend die Ergebnisse ihrer einjährigen Software-Entwicklung.

Unserer besonderer Dank für die Bereitstellung von Hard- und Software für die Projektgruppe gilt

- Herrn Kay Mühlenbruch von der BOV Computersysteme GmbH in Essen sowie
- Frau Renate Wanke von der Microsoft GmbH in Neuss.

Zudem möchten wir uns für die freundliche Unterstützung bei

- Herrn Dr. Dieter Ackermann vom Volkswohlbund in Dortmund,
- Herrn Lutz Doblaski von der Württembergische Versicherung AG in Stuttgart,
- Herrn Joachim Kober von der Vereinigte Haftpflichtversicherung in Hannover,
- Herrn Michael Manz von der Aegon-Lebensversicherungs AG in Düsseldorf,
- Herrn Hans-Ulrich Moers von der Barmenia Versicherung in Wuppertal,
- Herrn Johannes Schlattmann von der LVM Versicherungen in Münster,
- Herrn Prof. Dr. Reinhold Schönefeld von der Inverso GmbH in München,
- Herrn Dr. Michael Timm Management- und Technologieberatung in Köln sowie
- Herrn Klaus Wolf von der Aachen Münchener Versicherungs AG in Aachen bedanken.

Dortmund, im März 1999 Guy Vollmer

1 Einleitung

Um eine erste Orientierung bezüglich der Arbeit der Projektgruppe ***eCCo*** und damit auch dieses Endberichts zu bekommen, werden in Kapitel 1 einleitend die Grundlagen erläutert. In Kapitel 2 wird zunächst die Weiterentwicklung der VAA zu einer stärker objektorientierten Architektur, der sogenannten VAA++ vorgestellt, um daran anschließend in Kapitel 3 ausführlich die Realisierung dieser Weiterentwicklung und die neuen Rollen der einzelnen Komponenten der VAA++ zu beschrieben. Die beiden folgenden Kapitel widmen sich den beispielhaft unterstützten Geschäftsprozessen „Lebensversicherungs-Antrag“ (Kapitel 4) und „Kraftfahrzeug-Schaden“ (Kapitel 5). Der entwickelte Editor für verteilte Geschäftsprozesse und seine Funktionalität wird in Kapitel 6 detailliert dargestellt. Kapitel 7 beschreibt die exemplarische Modellierung und Simulation des Anwendungsbeispiels „LebensversicherungsAntrag“ mit Hilfe des in Kapitel 6 vorgestellten Editors für verteilte Geschäftsprozesse. Kapitel 8 beinhaltet das Fazit der Projektgruppe ***eCCo*** und liefert darüberhinaus einen Ausblick auf mögliche Weiterentwicklungen. Abschließend ist ein Glossar (Kapitel 9), ein Abbildungsverzeichnis (Kapitel 10), sowie ein Literaturverzeichnis (Kapitel 11) in diesen Endbericht integriert.

1.1 Was ist eine Projektgruppe?

Die Universität Dortmund mit über 30.000 immatrikulierten Studenten, davon 2.500 im Fachbereich Informatik, stellt den größten Informatikfachbereich der Bundesrepublik Deutschland. Als Reaktion auf die vieldiskutierte Diskrepanz zwischen Hochschulausbildung und Praxis wurde an der Universität Dortmund die sogenannte Projektgruppe ins Leben gerufen, die zentraler Bestandteil des Informatikstudiums ist. Alle Informatikstudenten nehmen im Laufe ihres Hauptstudiums an einer einjährigen Projektgruppe teil, in deren Verlauf sie mit insgesamt acht bis zwölf Studenten ein praxisbezogenes Software-Projekt eigenständig und eigenverantwortlich bearbeiten. Zur Unterstützung bei auftretenden Frage- und Problemstellungen begleitet sie ein Betreuer-Team, das sich in der Regel aus einem Professor/einer Professorin und ein bis zwei Assistenten/Assistentinnen zusammensetzt.

Die Projektgruppe ***eCCo*** am Lehrstuhl für Software-Technologie des Fachbereichs Informatik der Universität Dortmund wurde von Herrn Prof. Dr. Volker Gruhn, Herrn Dipl.-Math. Dirk Peters und Frau Dipl.-Inform. Ursula Wellen betreut. Als Studenten nahmen Thomas Bühren, Muharrem Cakir, Ercan Can, Andreas Dombrowski, Gereon Geist, Mustafa Gürgen, Sven Handschumacher, Markus Heller, Christian Lüer, Guy Vollmer und Jörg von Werne teil.

1.2 Vorgehensmodell

Als Grundlage für die Software-Entwicklung innerhalb der Projektgruppe wurde der sogenannte rollenbasierte Ansatz gewählt. Dieser Ansatz sorgt durch die Definition eigenverantwortlicher Rollen für mehr Transparenz und Effizienz während des Entwicklungsprozesses.

Da die verschiedenen Rollen im Entwicklungsprozeß in Abhängigkeit zueinander stehen, wird ein Interessenausgleich zwischen den beteiligten Rollen notwendig. Die sich daraus ergebende Kooperation zwischen den beteiligten Personen erhöht das gegenseitige Verständnis für die rollenspezifischen Aufgaben, Auffassungen und Probleme und erleichtert es somit frühzeitig, entstehende Probleme zu identifizieren und zu lösen.

So wurden die eigenständigen Rollen Projektmanager, Qualitätsmanager, System-Administrator, Entwickler innerhalb des Projekts definiert und von jeweils einem oder mehreren Projektgruppenteilnehmer ausgeübt. Die Projektgruppe ***eCCo*** mit elf Teilnehmern wurde zudem in drei Workpackages unterteilt, um ein konzentriertes, zielorientiertes Arbeiten in kleinen Arbeitsgruppen zu erreichen und die Bearbeitung von klar definierten Teilproblemen an diese Workpackages delegieren zu können. Es gab somit zu jeder Phase des Entwicklungsprozesses eine fest definierte Rolle für jeden Projektgruppenteilnehmer.

Innerhalb der Projektgruppe wurde vom Qualitätsmanager ein Vorgehensmodell entwickelt, das den gesamten objektorientierten Prozeß zur Software-Entwicklung in sechs iterativ zu durchlaufende Phasen gliederte [Mey88]:

- Anforderungsdefinition
- Objektorientierte Analyse (OOA)
- Grober Systementwurf
- Objektorientiertes Design (OOD)
- Objektorientierte Programmierung (OOP)
- Test und Validierung

Ferner wurden vom Qualitätsmanager Richtlinien erarbeitet, nach denen jede Phase abzuschließen war. Alle Workpackages durchliefen die obengenannten Phasen und erstellten zum Abschluß jeder Phase ein Dokument, das die einzelnen Ergebnisse präzise beschreibt. Zur Fortsetzung der Workpackage-Tätigkeiten wurden anschließend die erarbeiteten Ergebnisdokumente von den Betreuern, dem Projektmanager sowie dem Qualitätsmanager in Reviews auf Korrektheit überprüft und abgenommen. Damit wurde ein hohes Maß an Transparenz erzielt, da zu jeder Zeit der jeweilige Stand der Entwicklung nachvollziehbar war. Außerdem wurde dem Projektmanager eine effiziente Projektfortschrittskontrolle ermöglicht, die die weitere Detailplanung des Projektes unterstützte [Rin94].

Im Zuge der Integration kam es zu einer engen Zusammenarbeit der Workpackages für die beiden Geschäftsprozesse, da durch die Realisierung der objektorientierten Weiterentwicklung der Versicherungs-Anwendungs-Architektur einige entwickelte Teilergebnisse von beiden Workpackages genutzt werden konnten [Oes98]. Darüberhinaus wurde der entwickelte Editor für verteilte Geschäftsprozesse in die Arbeit der Projektgruppe ***eCCo*** integriert, indem er zur Modellierung des Geschäftsprozesses „Lebensversicherungs-Antrag“ eingesetzt wurde (siehe Kap. 7).

In der abschließenden Testphase der Projektgruppe ***eCCo*** veranlasste der Qualitätsmanager zudem das Ausscheiden je eines Software-Entwicklers aus jedem Workpackage, damit dieser die entwickelte Software eines anderen Workpackages testet und somit ein effizienter Test realisiert werden konnte.

1.3 Technische Grundlagen

Zum Zwecke der Spezifikation der zu entwickelnden Software wurde die Unified Modeling Language (UML) verwendet [Oes98, Fow97]. Sie wird häufig zur objektorientierten Software-Entwicklung verwendet und dient der Modellierung von Objekten, Komponenten und Geschäftsprozessen.

Die Realisierung unserer Software wurde mit der objektorientierten Programmiersprache Java durchgeführt, wobei wir uns in der Projektgruppe für die Entwicklungsumgebung Microsoft Visual J++ 1.1 entschieden haben [Krü97]. Zudem wurde die Swing-Klassenbibliothek von Sun Microsystems in der Entwicklung verwendet.

Rechnerseitig betrieben wir ein eigenes, selbstverwaltetes Local Area Network mit sechs Personalcomputern unter dem Betriebssystem Microsoft Windows NT 4.0, das von unserem System-Administrator installiert und gewartet wurde.

1.4 Was ist Electronic Commerce?

Unter Electronic Commerce versteht man allgemein die Geschäftsabwicklung mit Hilfe von elektronischen Mitteln [Bef96]. Es ist die Zusammenfassung aller Prozesse zur:

- Produktinformation und Produktwerbung
- Kundenwerbung
- Bestellung von Produkten und Dienstleistungen
- Austausch von Handelsdaten

Electronic Commerce ist also der Austausch von Informationen, die für eine kommerzielle Transaktion über elektronische Medien notwendig sind. Der dabei nicht näher spezifizierte Begriff „elektronische Medien“ umfaßt sowohl die Nutzung von Telekommunikationsinfrastruktur, Radio- und Fernsehwerbung als auch Tele-Shopping und Tele-Marketing.

Es gibt hierbei bereits ausgestaltete Beziehungen zwischen den Transaktionspartnern [EECT]:

- **Business-to-Business** (u.a. Bestellsysteme, Auftragsbearbeitung, Lieferung, Bezahlung)
- **Business-to-Consumer** (u.a. Internet-Shops (z.B. Buch- und CD-Bestellung))

Der **Business-to-Consumer** Bereich hat durch die rasante Verbreitung des Internets in den letzten Jahren zunehmend an Bedeutung gewonnen und ist somit auch für die Versicherungsbranche interessant geworden. Durch diesen Bereich können Versicherungsunternehmen eine enge Bindung zu ihren Kunden realisieren und sich mittelfristig neue Absatzmärkte schaffen.

Die Projektgruppe ***eCCo*** bearbeitete Problemstellungen aus dem **Business-to-Business** Bereich, da es sich zum einen bei den zu unterstützenden Geschäftsprozessen um versicherungsinterne Abläufe sowie teilweise um Abläufe zwischen verschiedenen Versicherungsunternehmen handelt. Zum anderen realisiert der Editor die graphische Modellierung verteilter Geschäftsprozesse und der Kommunikationskanäle zwischen entfernten Geschäftsprozeß-

teilen. Dieser Editor für verteilte Geschäftsprozesse dient somit ebenfalls der Unterstützung versicherungsinterner Abläufe, sowie deren Modellierung und Simulation.

1.5 Was ist die Versicherungs-Anwendungs-Architektur ?

Eine Grundproblematik zahlreicher Versicherungsunternehmen sind alte, selbstentwickelte Software-Systeme, die ihre Geburtsstunde vielfach in den sechziger und siebziger Jahren hatten und somit nach mittlerweile überholten Methoden und Programmierkonzepten entwickelt wurden. Diese proprietären Software-Systeme sind in aller Regel schlecht zu warten und nur unflexibel an die heutigen Anforderungen anzupassen. Erschwerend kommt hinzu, daß eigene Neuentwicklungen nur parallel zu bestehenden Systemen durchgeführt werden können - schließlich ist jedes Unternehmen weiterhin auf die reibungslose Abwicklung aller Geschäftsprozesse angewiesen. Naturgemäß ist die Entwicklung von Software, die auf die speziellen Anforderungen und Wünsche eines Unternehmens abgestimmt ist, sehr zeit- und damit kostenintensiv.

Aus diesem Grund wurde vom Ausschuß Betriebswirtschaft des Gesamtverbandes der Deutschen Versicherungswirtschaft (GDV) 1993 beschlossen, eine Versicherungs-Anwendungs-Archi-tektur (VAA) zu erarbeiten. Die VAA soll als Basis für die Entwicklung von Versicherungsanwendungen dienen. Mit der Definition der VAA sollen die Rahmenbedingungen für einen offenen Markt für standardisierte Anwendungs- und Basis-Software-Bausteine geschaffen werden. Langfristiges Ziel ist dabei die Initiierung eines Software-Komponentenmarktes, so daß die Versicherungsunternehmen die jeweils benötigten Anwendungsbausteine erwerben und in ihr System integrieren können.

Es handelt sich bei der VAA um ein Konstruktionsprinzip für ein Software-System bestehend aus einem Daten-, Funktions- und Prozeßmodell zuzüglich der Anwendungsplattform mit Software-Bausteinen und einer Schnittstellenarchitektur. Für einige Software-Bausteine wurden mittlerweile die Anforderungen an diese Komponenten, ihre Funktionalität und ihre Schnittstellen im Detail definiert [GDV96].

In Abbildung 1 ist das Zusammenspiel der Komponenten innerhalb der VAA zu sehen. Deutlich werden hierbei die 3 Ebenen der VAA:

- die Steuerungsebene mit dem Workflow-Manager, dem Vorgangs-Manager und dem Dialog-Manager
- die Anwendungsebene mit den beispielhaft herausgegriffenen Anwendungsbausteinen „Leben“, „Sonderwagnisdatei“, „Vertrag“ und „Versicherungspartner“
- die Dienstebene mit dem Parameter-System, dem Daten-Manager und weiteren Diensten wie unter anderem „Fehlerbehandlung“, „Textverarbeitung“ und „Berechtigungssystem“

Die Anwendungsbausteine werden hierbei ausschließlich von der Steuerungsebene aufgerufen und können Funktionen anderer Anwendungsbausteine nur über die Steuerungsebene benutzen. Es gibt keine hierarchische Anordnung der Anwendungsbausteine.

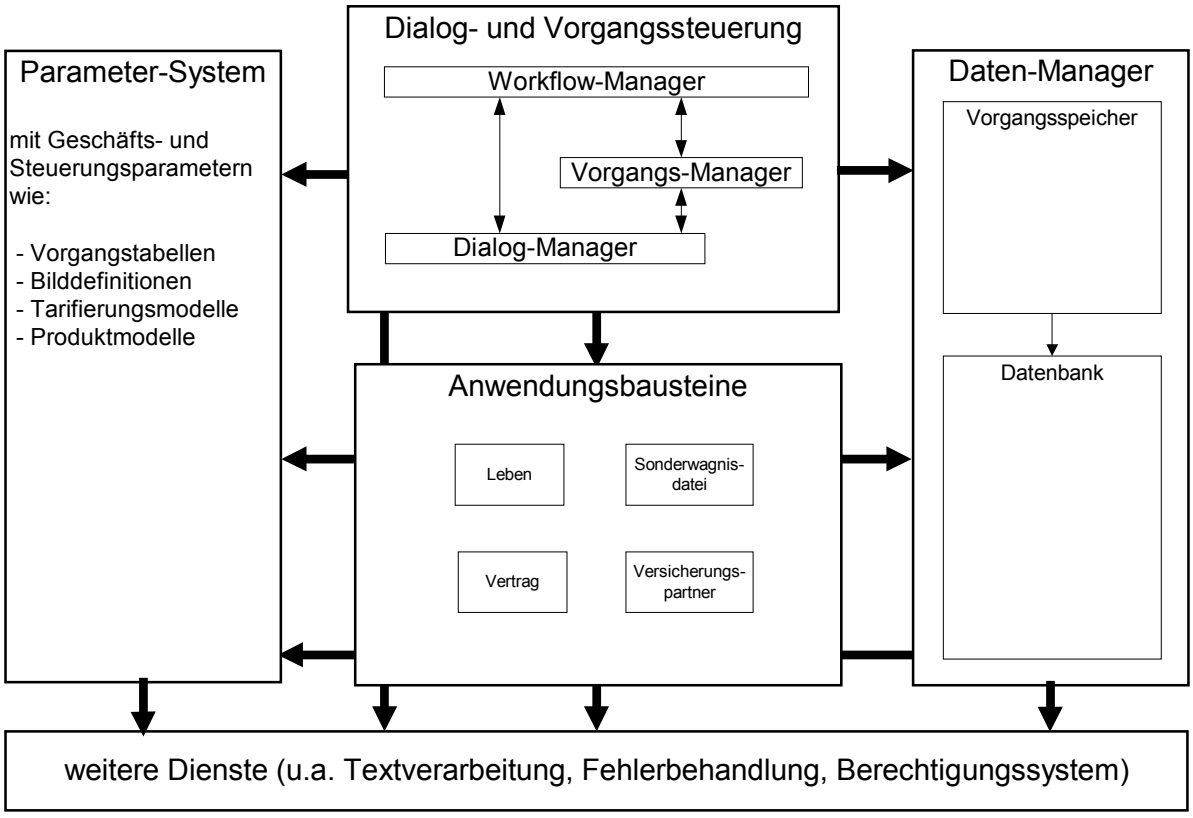

Abb. 1: Zusammenspiel der Komponenten in der VAA

Ferner gibt es keine Design-Vorschriften für die Anwendungsbausteine der VAA in Richtung Objektorientierung. Durch die Einschränkung, daß sich Anwendungsbausteine nicht gegenseitig aufrufen können, ist es erforderlich, daß die Funktionen der Anwendungsbausteine auf dem Datenmodell arbeiten. Es ist somit jederzeit möglich, daß verschiedene Anwendungsbausteine auf den gleichen Datensatz zugreifen.

1.6 *eCCo*-Kernergebnisse

Als Kernergebnisse der einjährigen Projektarbeit werden im folgenden vorgestellt:

- die Weiterentwicklung der VAA zu einer objektorientierten Architektur,
- der Workflow-Manager mit erweiterungsfähiger Klassenbibliothek für Workflow-Knoten,
- die verteilte Bearbeitung von Workflows: Teilnahme am Workflow überall im Internet,
- der Datenmanager zur Speicherung persistenter Objekte in relationalen Datenbanken (die Anwendung arbeitet stets mit Objekten und nicht mit Datenbank-Aufrufen),
- die Geschäftsprozesse „Lebensversicherungs-Antrag“ sowie „Kraftfahrzeug-Schaden“,
- der Editor für verteilte Geschäftsprozesse und
- die Modellierung des Anwendungsbeispiels „Lebensversicherungs-Antrag“ mit Hilfe des Editors.

2 Objektorientierte Weiterentwicklung der VAA

Einerseits sollte die Projektgruppe ***eCCo*** die Anwendungen für die verteilten Geschäftsprozesse „Lebensversicherungs-Antrag“ (s. Kap.4) und „Kraftfahrzeug-Schaden“ (s. Kap.5) unter Einbeziehung der VAA [GDV96] entwickeln. Andererseits wurde die Methode der objektorientierten Software-Entwicklung als zentrale Methode zur Realisierung des Software-Projekts definiert. Da die VAA nicht in allen Einzelheiten objektorientiert umgesetzt werden kann, bedurfte es einer eigenen, der Objektorientierung Rechnung tragenden Weiterentwicklung der VAA.

2.1 Kritikpunkte an der VAA

Hinsichtlich einer auf der VAA aufsetzenden und zugleich objektorientierten Software-Entwicklung sind zwei Punkte von zentraler Bedeutung:

- Die Kapselung des Vorgangswissens ist notwendig.
  In der VAA sind keine gegenseitigen Aufrufe der Anwendungsbausteine vorgesehen. Somit muß der Vorgangs-Manager in der VAA „allwissend“ sein. Unserer Ansicht nach ist eine Kapselung von Teilen dieses Wissens wünschenswert. Um dies zu realisieren, sollten sich die einzelnen Anwendungsbausteine gegenseitig aufrufen können. Durch die klare Definition der Schnittstellen wird weiterhin das Ziel eines sich entwickelnden Komponentenmarkts verfolgt.

- Die Kapselung der Daten ist notwendig.
  Die Kommunikation der Anwendungsbausteine erfolgt nach der VAA stets über die Daten. Daraus ergibt sich innerhalb der VAA die Möglichkeit, daß einzelne Anwendungsbausteine auf „fremde“ Daten zugreifen dürfen. Ein wesentliches Problem ist hierbei die Konsistenz der Daten. Kein Anwendungsbaustein kann wissen, wie ein konsistenter Zustand fremder Daten auszusehen hat. Somit wäre also eine Kapselung des Datenzugriffs durch den jeweiligen Baustein wünschenswert. Zu diesem Zwecke muß der oben dargelegte gegenseitige Aufruf der Anwendungsbausteine erlaubt sein.

2.2 Von der VAA zur VAA++

Die objektorientierte Weiterentwicklung der VAA hat zu ihrer ursprünglichen Version zwei wesentliche Pluspunkte aufzuweisen:

Zum einen haben in der VAA++ Daten einen eindeutigen Besitzer. Dies hat den Vorteil, daß die Konsistenz der Daten gewährleistet werden kann und alle Änderungen lokal bleiben. Die Idee der zentralen Datenhaltung wird aufrechterhalten, so daß im gesamten System nur eine Datenbank implementiert werden muß, was den Administrationsaufwand positiv beeinflußt.

Zudem werden so Datenbankabfragen auf einem zentralen Datenbestand besser realisierbar. Bei einem kompletten Austausch des Datenbanksystems muß außerdem nur die Daten-Komponente geeignet angepaßt werden.

Zum anderen ist in der VAA++ die gegenseitige Benutzung der Anwendungsbausteine möglich. Es bedarf hierzu einer exakten Definition der Import- und Exportschnittstellen. Dabei werden die Anwendungs-Bausteine hierarchisch angeordnet, da auf diesem Wege kreisfreie Benutzt-Graphen realisiert werden und das Prinzip „nothing works until everything works“ vermieden werden kann.

In Abbildung 2 wird die VAA++ in einer an [GDV96] angelehnten Symbolik dargestellt. Die Pfeile beschreiben dabei die „Benutzt-Beziehungen“ zwischen den einzelnen Systemteilen, und die Behälter im Parameter-System sowie im Daten-Manager symbolisieren die nun getrennten Bereiche innerhalb dieser Komponenten.

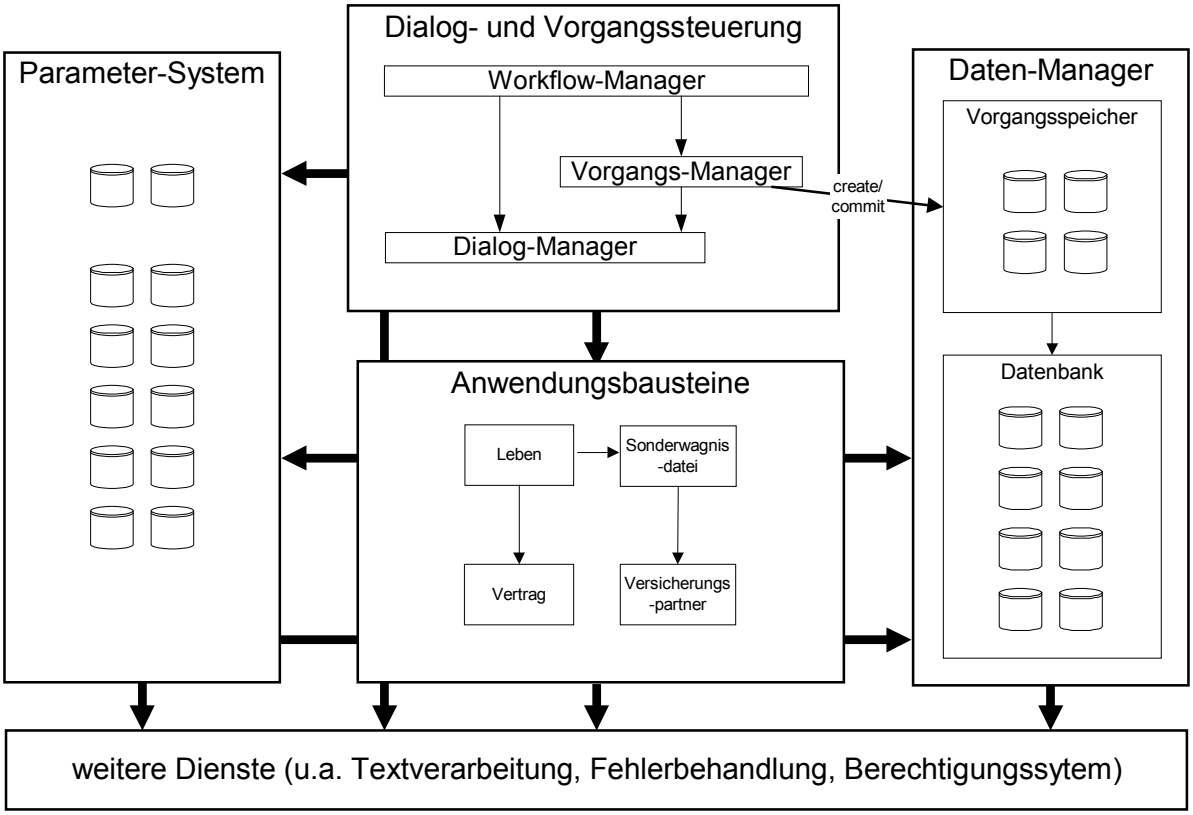

Abb.2: Die VAA++ in der Darstellungsweise der VAA

Die Änderungen der VAA++ (Abb. 2) im Vergleich zur VAA (Abb. 1) sind:

- eine hierarchische Anordnung der Anwendungsbausteine,
- der Daten-Manager wird vom Parameter-System zur Speicherung der Parameterdaten benutzt,
- getrennte Bereiche innerhalb des Daten-Managers sowie
- getrennte Bereiche innerhalb des Parameter-Systems.

2.3 Schichtenarchitektur der VAA++

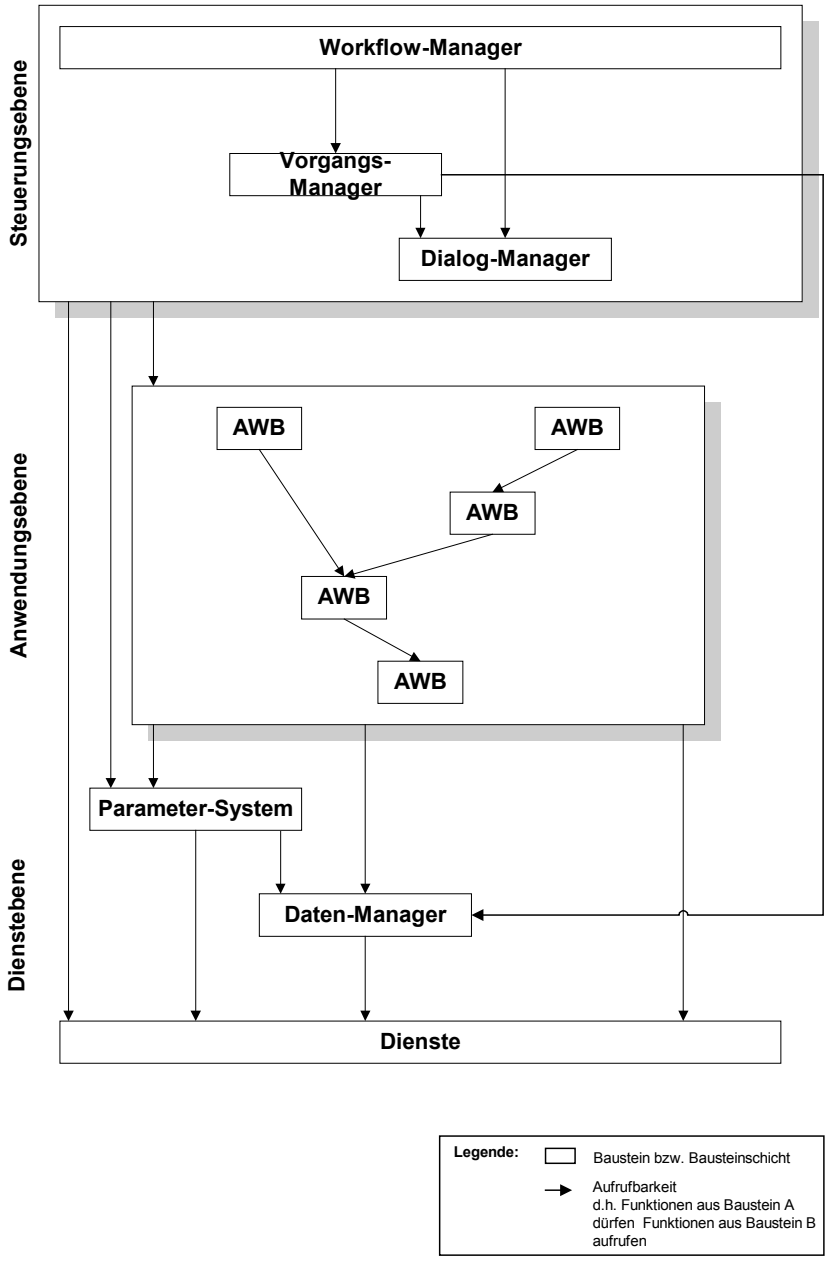

Abb.3: Darstellung der Schichtenarchitektur der VAA++

Die VAA++ ist aus drei Schichten aufgebaut, wie in Abbildung 3 zu sehen ist. In der obersten Schicht befindet sich die Steuerungsebene, gefolgt von der Anwendungs- und Dienstebene. Die oberste Ebene enthält den Workflow-Manager, den Vorgangs-Manager sowie den Dialog-Manager. Auf dieser Ebene kann der Workflow-Manager auf alle weiteren Manager zugreifen. Der Zugriff des Vorgangs-Manager ist hingegen auf den Dialog-Manager beschränkt. Unterhalb der Steuerungsebene befindet sich die Anwendungsebene. Hier sind alle Anwendungsbausteine hierarchisch angeordnet, so daß Anwendungsbausteine, die sich in der Hierarchie oben befinden, nur Bausteine aus der darunter liegenden Hierarchie aufrufen können.

Zur Dienstebene gehören das Parameter-System, der Daten-Manager sowie die Basisdienste. Hierbei können Aufrufe aus dem Parameter-System sowohl auf den Daten-Manager wie auch auf Basisdienste erfolgen. Der Daten-Manager kann sich ebenfalls der angebotenen Basisdienste bedienen.

Alle Komponenten aus der Steuerungsebene können sowohl auf die komplette Anwendungsebene wie auch auf das Parameter-System und die Basisdienste zugreifen. Der Vorgangs-Manager kann auf den Daten-Manager zum Starten und Beenden eines Vorgangs-Speichers zugreifen. Die Anwendungsbausteine können sowohl auf das Parameter-System, auf den Vorgangs-Speicher des Daten-Managers und die Basisdienste zugreifen.

2.4 Anpassung des Daten-Managers

Für den Daten-Manager hat sich die Aufgabe verändert. Er ist nun für die Persistentmachung von Objekten zuständig und realisiert damit die Existenz der Objekte über die Laufzeit der Anwendung hinaus. Die logischen Datensichten aus der VAA entfallen, und es existiert kein unternehmensweites Datenmodell mehr. Die einzelnen Anwendungsbausteine haben die Datenhoheit über ihre Objekte. Der Daten-Manager legt die Objekte in einer Datenbank ab. Der Vorgangsspeicher ist weiterhin vorhanden, enthält statt logischer Teilsichten von Daten jedoch stets vollständige Objekte. Im Unterkapitel 3.7.1 wird die hier skizzierte neue Rolle des Daten-Managers sowie seine Architektur detailliert beschrieben.

Nach der Vorstellung der objektorientierten Weiterentwicklung der Versicherungs-Anwendungs-Architektur wird in Kapitel 3 nun die praktische Umsetzung der beschriebenen Architektur VAA++ ausführlich dargestellt und erläutert.

3 Praktische Umsetzung der VAA++

Die in Kapitel 2 vorgestellte Architektur VAA++ wurde konkret in der Programmiersprache Java realisiert. Diese Implementierung bezeichnen wir als das System VAA++. Damit kann ein Unternehmen die Bearbeitung von Geschäftsprozessen elektronisch unterstützen.

Zunächst wird vorgestellt, auf welche Weise ein Anwender neue Geschäftsprozesse, die auf elektronische Unterstützung umgestellt werden sollen, in das VAA++-System integrieren kann. Dabei wird das System zunächst aus Anwendersicht vorgestellt. Dies dient dazu, einen Überblick über die Verwendung des Systems zu bekommen und die Begriffe zu beschreiben, auf die im weiteren Bezug genommen wird.

Danach wird die Struktur des VAA++-Systems im Detail beschrieben und die technische Umsetzung erläutert.

3.1 Unterstützung neuer Geschäftsprozesse mit der VAA++

Soll ein neuer Geschäftsprozeß integriert werden, so muß man ihn von den Begriffen der Alltagswelt auf technische Begriffe abbilden. In einem ersten Schritt beschreibt man den neuen Geschäftsprozeß umgangssprachlich, danach wird eine Menge von **Anwendungsbausteinen**, **Dialogmasken**, **Vorgängen** und **Workflows** definiert. Zuletzt werden diese Teile implementiert und der neue Geschäftsprozeß systemweit verfügbar gemacht.

Die technischen Begriffe werden in Kapitel 3.2 definiert. Um jedoch den Gesamtzusammenhang zwischen diesen Begriffen deutlich werden zu lassen, werden sie bereits an dieser Stelle benutzt und können zunächst mit ihrer intuitiven Bedeutung verwendet werden.

Die Schritte, die für die Integration eines neuen Geschäftsprozesses notwendig sind, werden nun im Einzelnen vorgestellt:

1. Beschreibung des Geschäftsprozesses

   Für den typischen Ablauf eines Geschäftsprozesses wird eine umgangssprachliche Beschreibung erstellt. Dabei werden Bearbeiter, benötigte Dokumente, die Wertberechnungen und die Entscheidungen zunächst für einzelne Teilschritte beschrieben und dann die Reihenfolge und Abhängigkeiten dieser Teilschritte festgelegt.

   Die Wertberechnungen werden später auf technischer Ebene durch Methoden von Anwendungsbaustein-Objekten durchgeführt, die Teilschritte im Fachkonzept werden technisch durch sogenannte Vorgänge realisiert, und die fachlichen Abhängigkeiten der Vorgänge werden in Workflows festgelegt.

2. Modellierung des Workflows

a) Zusammenstellen der Anwendungsbausteine

   Ein Anwendungsbaustein besteht aus logisch zusammenhängenden Anwendungsobjekten, die Attribute und Methoden besitzen. Sie berechnen Teilergebnisse innerhalb eines Teilschrittes, das sind meist einfache, grundlegende Funktionsberechnungen. Die Methoden des Anwendungsbausteines werden von einem Vorgang oder von anderen Anwendungsbausteinen aufgerufen. Die Objekte können durch den Daten-Manager gespeichert werden.

b) Festlegen der Dialogmasken, Vorgänge und Workflows für den Geschäftsprozeß

   Die Dialoge, Vorgänge und Workflows steuern den Ablauf eines Geschäftsprozesses. Dialogmasken werden für die Kommunikation des Systems mit dem Benutzer benötigt, wenn ihm Daten angezeigt oder von ihm angefordert werden.

   Ein Vorgang erhält zur Bearbeitung ein Anwendungsobjekt und kann Dialogmasken aufrufen, um Daten anzuzeigen oder Dateneingabe zu ermöglichen. Er kann das Anwendungsobjekt verändern und leitet dieses oder ein anderes innerhalb eines sogenannten Token an den nächsten Vorgang weiter.

   Die örtliche und zeitliche Reihenfolge der Vorgänge wird in einem Workflow definiert, der somit den Fluß der Vorgangsergebnisse durch die einzelnen Stationen eines Geschäftsprozesses beschreibt.

3.2 Definitionen

Zum besseren Verständnis werden nun die bereits benutzten Begriffe definiert und gegeneinander abgegrenzt.

Ein **Anwendungsbaustein** besteht aus Anwendungsobjekten und enthält logisch zusammenhängende Daten und Methoden aus einem Teil des Anwendungsgebietes. Die Methoden manipulieren die Daten oder berechnen daraus Werte. Die Objekte verschiedener Bausteine können dazu im Rahmen der vorgegebenen Hierarchie zusammenarbeiten. Die Funktionalität eines Anwendungsbausteines ist innerhalb des Anwendungsbereiches typisch und wird selten verändert.

Eine **Dialogmaske** dient dazu, dem Benutzer Daten anzuzeigen und ihm die Eingabe von Daten zu ermöglichen. Sie werden von Vorgängen zur Anzeige gebracht. Dialoge können die Eingaben und Knopfdrücke des Benutzers nicht selbständig verarbeiten, sondern melden diese dem Vorgang, von dem sie aufgerufen worden sind.

Ein **Vorgang** ist eine einzelne Teilaufgabe des Geschäftsprozesses, die von einem Bearbeiter in einer zusammenhängenden Arbeitssitzung bearbeitet werden kann. Alle Änderungen an Daten, die innerhalb eines Vorgangs anfallen, werden gleichzeitig gesichert oder verworfen („Alles-oder-nichts-Prinzip“, auch „Transaktionsprinzip“). In einem Vorgang wird aus einem Eingabeobjekt (Token) ein neues Ausgabeobjekt erzeugt.

Ein **Workflow** beschreibt die Reihenfolge der Ausführung von Vorgängen innerhalb eines Geschäftsprozesses und definiert den Fluß der Arbeitsergebnisse zwischen mehreren Bearbeitern an mehreren Orten zu verschiedenen Zeiten. Ein Workflow ist ein Graph mit Knoten, welche entweder die Ausführung eines Vorgangs starten oder zur Steuerung des Workflows dienen. Entlang der Graphkanten werden die Arbeitsergebnisse durch ein Token von einem Knoten zum nächsten transportiert.

Ein **Token** ist die Informationseinheit, die zwischen einzelnen Knoten eines Workflow-Graphen transportiert wird. Ein Token enthält ein numerisches Ergebnis, eine Referenz auf eine Menge von Anwendungsobjekten und Informationen darüber, wann es an den nächsten Knoten weitergeleitet werden soll. Dies kann zu einem festgesetzten Zeitpunkt oder bei Eintreffen einer angegebenen Nachricht erfolgen.

Änderungen im Ablauf eines Geschäftsprozesses werden in den Workflows definiert. Demgegenüber werden Anwendungsbausteine nicht bzw. nur selten verändert, denn sie kapseln lediglich die Art und Weise, wie Berechnungen durchgeführt werden und nicht, wie diese Werte zu verknüpfen sind.

Man unterscheidet die Begriffe der fachlichen Ebene und der technischen Ebene. Während Geschäftsprozesse, Teilschritte und Teilaufgaben im Anwendungsgebiet definiert sind, gehören Dialoge, Token, Vorgänge und Workflows zur technischen Welt der VAA++.

3.3 Überblick VAA++

Die Architektur des VAA++-Systems ist, wie in Abbildung 4 zu sehen, von oben nach unten in die drei Ebenen Steuerungsebene, Anwendungsebene und Dienstebene unterteilt, die später im Detail beschrieben werden. Zusätzlich gibt es das Paket „Systemstart“.

In der Steuerungsebene sind alle Funktionen zu finden, die zur Steuerung des Ablaufs von Geschäftsprozessen nötig sind, so z.B. die Dialoge, Vorgänge und Workflows der Geschäftsprozesse. Die Anwendungsebene enthält alle Anwendungsbausteine, die auf fachlicher Ebene für die Berechnung gebraucht werden. In der Dienstebene liegen alle Dienste, die im gesamten System in Anspruch genommen werden können (z.B. Datenspeicherung, Drucken, Fax, Mail).

Im Paket „Systemstart“ sind alle Funktionen untergebracht, mit denen in der Zentrale oder auf den Client-Rechner das Hauptprogramm und alle notwendigen Dienste-Server gestartet werden.

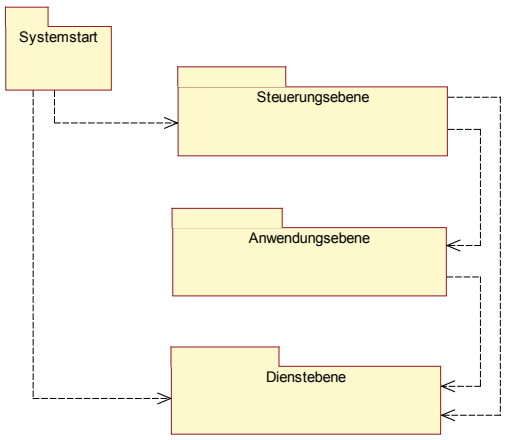

Abb. 4: Überblick VAA++

Steuerungs-, Anwendungs- und Dienstebene sind hierarchisch geordnet, höherliegende Schichten dürfen in den darunterliegenden Schichten alle Dienstleistungen aufrufen. So kann die Steuerungsebene sowohl die Anwendungsebene als auch die Dienstebene benutzen. Die Anwendungsebene benutzt ebenfalls die Dienstebene, diese kann höherliegende Schichten nicht ansprechen.

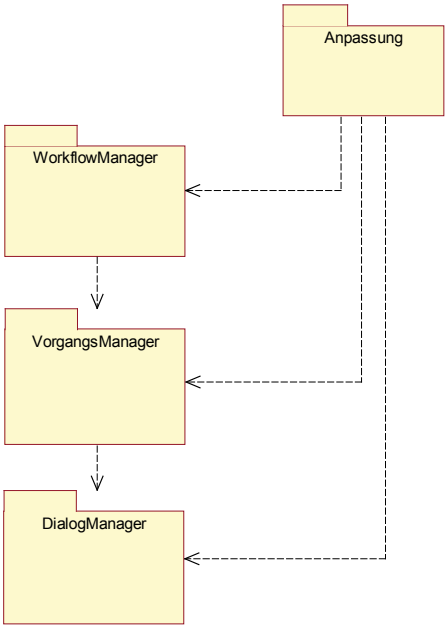

Abb. 5: Steuerungsebene

3.4 Steuerungsebene

Die in Abbildung 5 dargestellte Steuerungsebene enthält alle Objekte, die zur Steuerung der Abläufe von Geschäftsprozessen zusammenarbeiten.

Hier finden sich der Workflow-Manager, der Vorgangs-Manager, der Dialog-Manager und das Paket „Anpassungen“, in dem die Dialoge, Vorgänge und Workflows für alle Geschäftsprozesse enthalten sind.

3.4.1 Dialog-Manager

Der Dialog-Manager ist für die Anzeige des Hauptfensters, des Logfensters mit Meldungen für den Benutzer und der Dialogmasken verantwortlich. Dialoge werden von Vorgängen aufgerufen, wenn diese Daten anzeigen und Dateneingaben anfordern möchten.

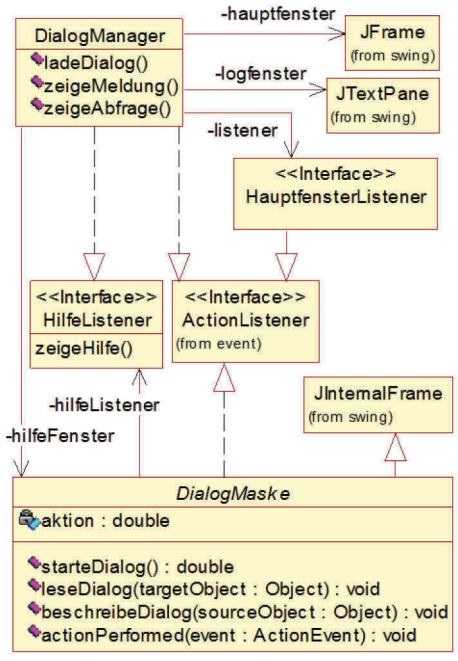

Abb. 6: Dialog-Manager

Wie im UML-Klassendiagramm [Oes98] in Abbildung 6 zu sehen, erben alle Dialogmasken von der Klasse “DialogMaske”. Sie sorgen im wesentlichen dafür, daß alle Benutzeraktionen, z.B. Knopfdrücke und Tastatureingaben, für die Weiterverarbeitung registriert werden. Dialogmasken können mit Daten beschrieben werden (diese sind dann in der Maske sichtbar) und dort eingegebene Daten können ausgelesen werden.

Berechnungen und Entscheidungen aufgrund eingegebener Daten dürfen nicht in den Masken geschehen, sondern nur in den aufrufenden Vorgängen. Dialoge sind frei von Anwendungswissen und Steuerinformationen, sie können daher einfach gegen andere Dialoge ausgetauscht werden. Dies ist z.B. bei der Verwendung anderer Programmiersprachen von Vorteil, die nicht wie Java eine betriebssystemunabhängige Oberflächenbibliothek bieten.

Dialogmasken können selbständig oder als Teil von anderen Dialogmasken benutzt werden. So wird z.B. die Eingabemaske, mit der eine Adresse erfaßt wird, innerhalb der Eingabemaske für einen Versicherungspartner wiederverwendet. Auf diese Weise kann die Anzahl der Bestandteile, die zu implementieren sind, erheblich verringert werden.

3.4.2 Vorgangs-Manager

Der Vorgangs-Manager dient dazu, auf dem lokalen Rechner die Ausführung einzelner Vorgänge zu ermöglichen. Er verwaltet eine Menge von Vorgängen.

Vorgänge besitzen einen eigenen Vorgangsspeicher, in welchem sie die gerade benötigten Daten ablegen können. Änderungen an diesen Daten werden erst bei erfolgreicher Beendigung eines Vorgangs vom Vorgangsspeicher in die Datenbank übertragen. Andernfalls werden alle Daten des Vorgangsspeichers verworfen. Vorgänge können Dialogmasken vom Dialog-Manager ausführen lassen und deren Ergebnisse und Aktionen auswerten.

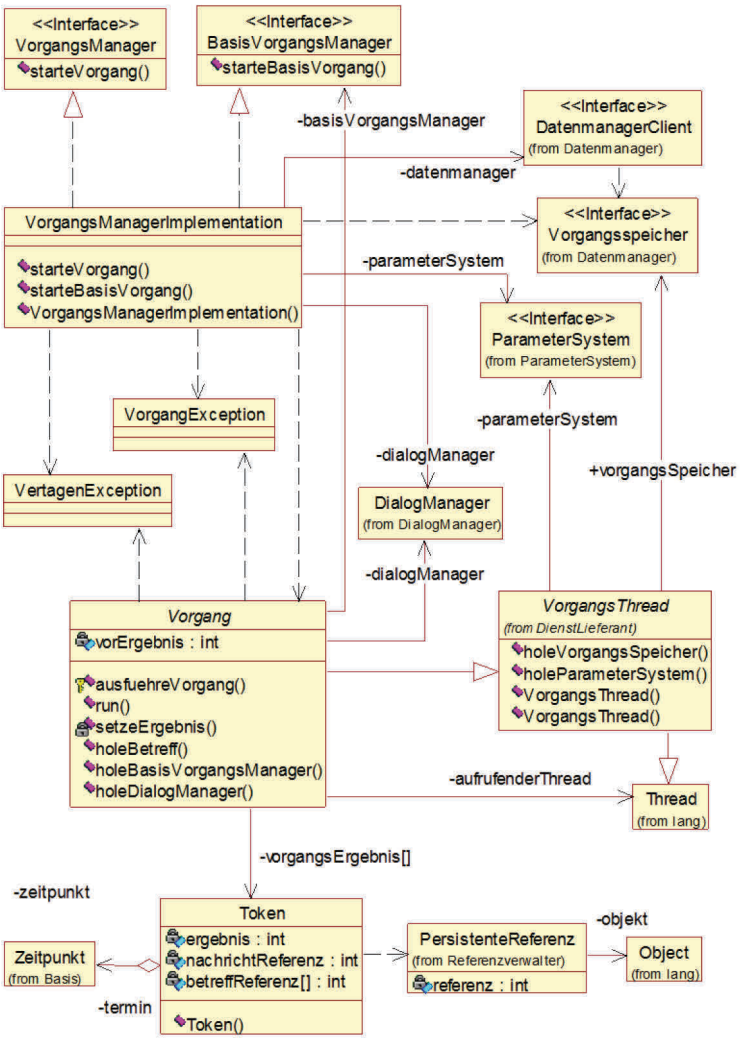

Abb. 7: Vorgangs-Manager

Es ist möglich, daß ein Vorgang andere Vorgänge aufruft. Diese besitzen als „Basisvorgänge“ keinen eigenen Vorgangsspeicher und arbeiten innerhalb des bereits vorhandenen Vorgangsspeichers des Aufrufers. So können Vorgänge in kleinere Einheiten zerlegt werden, sie werden zum einen übersichtlicher und zum anderen können so wiederverwendbare Teile eines Vorgangs gekapselt werden.

Die Klasse „VorgangsManagerImplementation“ aus Abbildung 7 stellt den Vorgangs-Manager bereit. Es gibt ihn auf jedem Client-Rechner genau einmal sowie auf einem zentralen Vorgangs-Server, der automatisch ablaufende Vorgänge ausführt.

Der Vorgangs-Manager startet auf Anfrage des zentralen Workflow-Managers lokal Vorgänge (mit Vorgangsspeicher) und Basisvorgänge (ohne eigenen Vorgangsspeicher). Er richtet für jeden ablaufenden Vorgang einen Vorgangsspeicher ein und übergibt diesen dem Vorgang, wie auch Referenzen auf Parameter-System und Dialog-Manager.

Ein „VorgangsThread“ erbt vom Java-Thread. Er speichert die Referenzen auf den eigenen Vorgangs-Speicher und das Parameter-System, so daß diese Dienste während der Ausführung eines Vorgangs auch von den aufgerufenen Anwendungsbausteinen angesprochen werden können. Man kann einen VorgangsThread als eine Art „kleine Laufzeitumgebung“ für einen Vorgang betrachten.

Die Klasse „Vorgang“ erbt von VorgangsThread und beinhaltet alle weiteren Daten, die für die Ausführung eines Vorgang notwendig sind. Ein Vorgang erhält als Eingabe ein Betreff-Token und liefert ein Ergebnis-Token zusammen mit einem numerischen Vorgangsergebnis zurück. Als technische Daten speichert er neben Referenzen auf Vorgangs-Speicher und Parameter-System auch eine auf den eigenen Dialog-Manager.

3.4.3 Workflow-Manager

Der zentrale Workflow-Manager ist für die Steuerung aller Workflows verantwortlich, die im System aktiv sind oder auf Fortsetzung warten. Mit der Workflow-Engine verwaltet er die Liste der Benutzer des Systems und steuert den Verlauf der Workflows. Ein Ausschnitt des Klassendiagramms der Workflow-Engine ist in Abbildung 8 zu sehen.

Der Workflow-Manager verwaltet für jeden Workflowtyp einen Referenzgraphen, der nur ein einziges Mal im Zentralrechner existiert und den sich alle Workflows desselben Typs als Vorlage teilen. Wenn ein Benutzer einen Workflow neu startet, wird der zugrundeliegende Workflow-Graph betrachtet und für den ersten Knoten ein eigener KnotenThread gestartet, der seinen Knoten “in Ausführung” darstellt – ähnlich dem zuvor beschriebenen Verhältnis zwischen Vorgang und VorgangsThread. Der KnotenThread startet die Ausführung der spezifischen Aktion des ersten Knotens. Der Knoten teilt der WorkflowEngine schließlich mit, welche Token als Ergebnis produziert wurden. Diese Token werden von der WorkflowEngine dann an den oder die nachfolgenden Knoten im Workflow-Graphen weitergereicht.

Wenn ein Knoten als Rückgabewert Token produziert, in denen eine spätere Weiterverarbeitung festgelegt ist, so wird die entsprechende Graph-Kante als “WorkflowZweig” in die in Abbildung 8 gezeigte Liste “wartendeWorkflowZweige” aufgenommen. Diese Liste wird stets bei Eintreffen einer Nachricht auf Token durchsucht, die auf eben diese Nachricht warten. Periodisch wird ebenso nach Token gesucht, die zu einem angegeben Zeitpunkt weitergereicht werden sollen. Gefundene WorkflowZweige werden anschließend wie oben beschrieben fortgesetzt.

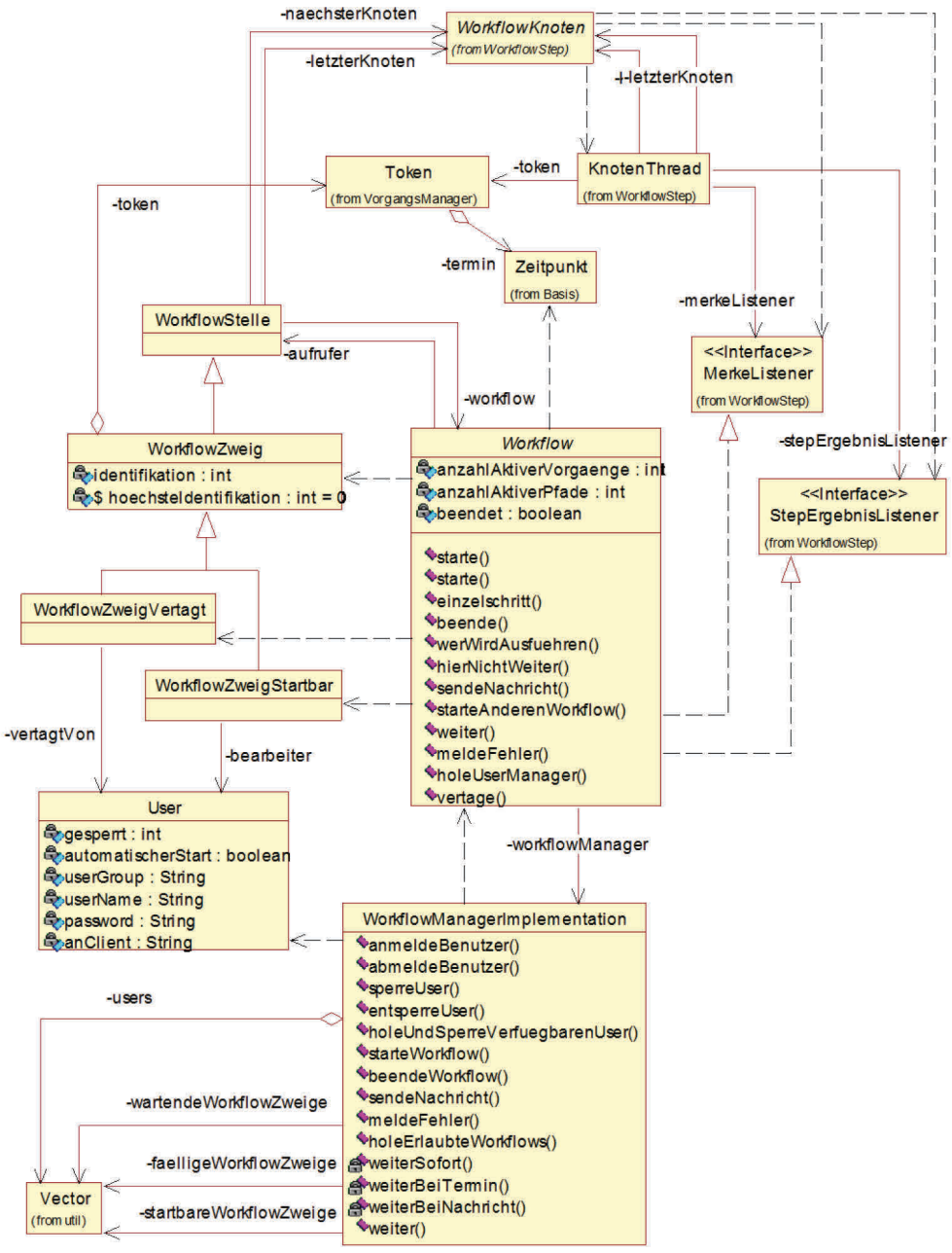

Abb. 8: Workflow-Engine

Das in der Übersicht zum gesamten Workflow-Manager in Abbildung 9 sichtbare Paket “WorkflowStep” enthält unter anderem alle Knoten, die für den Aufbau von Workflow-Graphen verwendet werden können. Diese Knotenbibliothek ist beliebig erweiterbar. Dadurch können neue Arten von Workflows unterstützt werden.

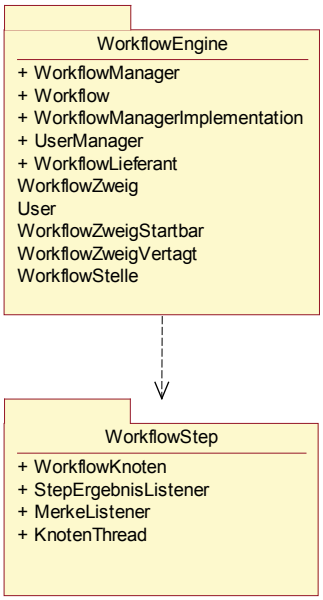

Abb. 9: Übersicht Workflow-Manager

Ein Knoten kann einen oder mehrere Eingänge und Ausgänge enthalten, auf denen Token weitergeleitet werden. Jeder Knotentyp kann eine andere Aufgabe erfüllen. Im folgenden werden einige der Knoten genauer beschrieben. Die beschriebenen Knoten sind als Ausschnitt des Klassendiagramms in Abbildung 10 dargestellt.

Ein **Synchronisationsknoten** wartet, bis an allen Eingängen einmal ein Token angekommen ist und merkt sich dessen "Betreff". Dann werden entweder alle Betreffs im Array eines Token weitergegeben oder nur der Betreff eines auswählbaren Eingangs. Das Merken der Betreffs geschieht über den MerkeListener, der vom Workflow implementiert wird. Dort können von jedem Knoten Daten abgelegt werden.

Der **Sammelknoten** ist ein “variabler” Synchronisationsknoten, denn man kann die Anzahl der Token, die er an seinem Eingang sammeln soll, frei wählen. Wenn am Eingang ein Token mit "Vorergebnis" 1 ankommt, wird ein Zähler erhöht und der Betreff gemerkt. Hat ein eintreffendes Token als “Vorergebnis” eine 2, dann wird der Zähler erniedrigt. Wenn der Zähler auf Null fällt, werden alle gemerkten Betreffs im Betreff-Array des Token weitergegeben. Dieses Weiterreichen kann auch manuell ausgelöst werden durch ein Token mit Vorergebnis 3.

Bei **Entscheidungsknoten** werden den Elementen des Arrays “verzweigungsBedingungen” kanonisch die numerierten Ausgänge zugeordnet. Das Token wird auf allen Ausgängen weitergereicht, deren zugeordenete Arraywerte mit dem Vorergebnis übereinstimmen.

Von dem **AndererWorkflowKnoten** wird ein angegebener Workflow mit dem mitgelieferten Token als Startwert gestartet und auf dessen Beendigung gewartet. Der aufgerufene Workflow liefert ein Token als Ergebnis, das dann an den nächsten Knoten weitergereicht wird.

NachrichtenKnoten: Der Betreff im erhaltenen Token wird als Nachricht an den WorkflowManager gesendet. Vorgänge können im Token festlegen, daß der Workflow erst nach Erreichen der angegebenen Nachricht fortgesetzt wird. Alle Zweige im Workflow, die auf diese gesendete Nachricht warten, werden bei der Ausführung des NachrichtenKnotens fortgesetzt.

Der **NachrichtWarteKnoten** dient zur einfachen Modellierung des Wartens auf eine Nachricht mit einem bestimmten Timeout. Bei Eintreffen der Nachricht wird der Workflow am ersten Ausgang des NachrichtWarteKnotens fortgesetzt, nach einer angegebenen

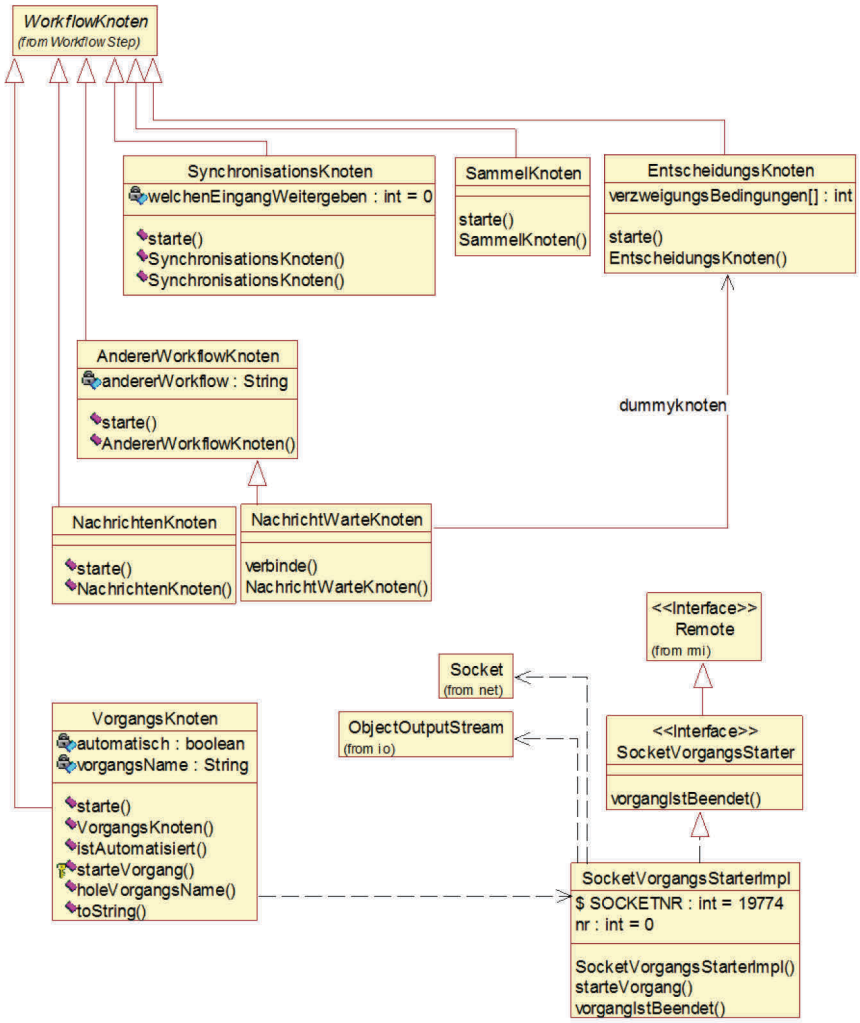

Abb. 10: Auswahl einiger Workflow-Knoten

Wartezeit ohne eingetroffene Nachricht am zweiten Ausgang. Intern startet der NachrichtWarteKnoten einen kleinen Workflow, der nach einer bestimmten Zeit oder bei Eintreffen der Nachricht mit einem von zwei "Vorergebnissen" abgebrochen wird. Dem NachrichtWarteKnoten wird für den Benutzer unsichtbar automatisch ein Entscheidungsknoten nachgeschaltet, der die zwei Ausgänge anhand dieses Vorergebnisses realisiert. An diesem Beispiel läßt sich erkennen, daß die Workflow-Idee auch dazu benutzt werden kann, um interne Teile des Systems VAA++ zu realisieren.

Der **VorgangsKnoten** dient dazu, die Ausführung eines Vorgangs aus einem Workflow heraus anzustoßen. Vor dem Aufruf des VorgangsKnotens legt der Workflow fest, von welchem Bearbeiter an welchem Rechner der Vorgang ausgeführt werden soll. Diese Angaben werden dem VorgangsKnoten mitgeteilt. Dieser baut dann eine Socket-Verbindung zum entsprechenden Client-Rechner auf und läßt dort den Vorgang ausführen. Das zurückgelieferte Token des Vorgangs wird dem VorgangsKnoten per RMI mitgeteilt.

Abbildung 11 zeigt das Zusammenspiel von zentralem Workflow-Manager und dem Vorgangs-Manager auf einem entfernten Client am Beispiel des Vorgangs “Freigabe durch Vorgesetzten”.

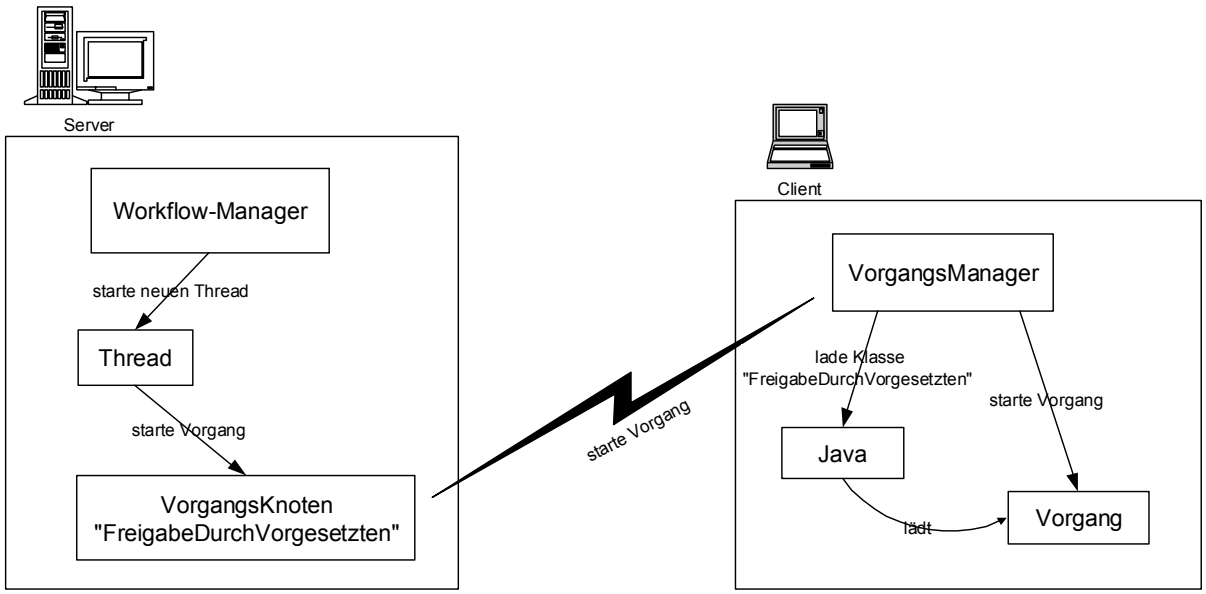

Abb. 11: Wie werden Vorgänge auf den Clients gestartet?

Die WorkflowEngine erzeugt und startet zunächst wie für jeden Knoten einen KnotenThread, der die Angabe des Zielrechners für die entfernte Ausführung des Vorgangs beinhaltet. Der VorgangsKnoten kann mit diesen Daten eine Socket-Verbindung zum Vorgangs-Manager des Clients aufbauen und beauftragt diesen mit der Ausführung des Vorgangs. Die Ausführung von Vorgängen wurde bereits beschrieben. In der Graphik sieht man zusätzlich, daß zunächst durch Java die Klasse “Freigabe durch Vorgesetzten” geladen wird. Diese Klasse enthält die Vorgangsdefinition, die nun abgearbeitet wird. Nach Beendigung des Vorgangs werden die Vorgangsergebnisse per RMI dem VorgangsKnoten zurückgemeldet.

RMI verwaltet jedoch nur lokale oder fest angegebene Rechnernamen. Darum konnte nur die Richtung vom Client zum Server über RMI realisiert werden, denn hier kann der Rechnername des Zentralrechners beim Programmstart vorgegeben werden, während ein Client im weltweiten Internet eine bei jedem Start wechselnde Adresse erhalten kann. Die Richtung vom Server zu den Clients wurde daher durch Benutzung von Sockets realisiert.

3.4.4 Anpassungen

Das in Abbildung 12 in Ausschnitten gezeigte Paket „Anpassungen“ stellt einen zentralen Ort für alle Anpassungen dar, die innerhalb der Steuerungsebene des VAA++-Systems benutzt werden können. Man unterscheidet zwischen Systemanpassungen, allgemeinen und spezifischen Anpassungen für die Versicherungsbranche.

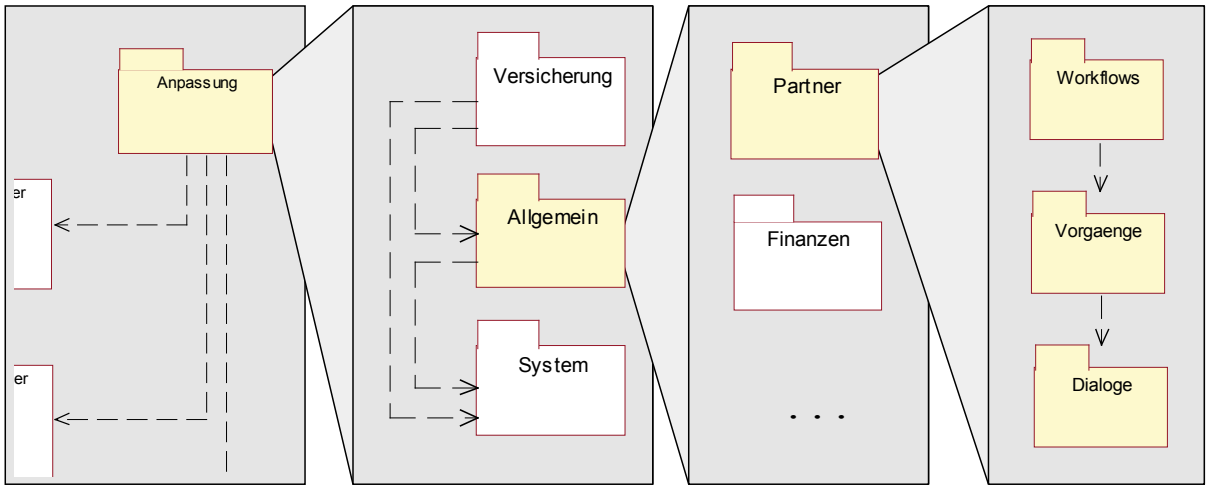

Abb. 12: Das Package Anpassung und darin enthaltene Packages

Systemanpassungen betreffen Anpassungen an internen Teilen des Systems VAA++. Dies können z.B. das Login- oder Hilfefenster sein, aber auch das Grundlayout aller Dialoge im System. Dadurch ist VAA++ leicht konfigurierbar. Zu den allgemeinen Anpassungen zählen die allgemein verfügbaren Dialogmasken, Vorgänge und Workflows etwa für die Anwendungsbausteine „Partner“, „Kommunikation“, „Historie“ und „Finanzen“.

Die Anpassungen für die Branche „Versicherungen“ enthalten Dialoge, Vorgänge und Workflows zu den benutzten Versicherungs-Anwendungsbausteinen und den durchzuführenden Geschäftsprozessen. Die Projektgruppe ***eCCo*** hat beispielhaft zwei Geschäftsprozesse der Versicherungsbranche mit dem System VAA++ implementiert. Details zu diesen Geschäftsprozessen folgen in den Kapiteln 4 und 5.

3.5 Systemstart

Im Paket „Systemstart“ sind alle Funktionen untergebracht, mit denen in der Zentrale oder auf den Client-Rechner das Hauptprogramm und alle notwendigen Dienste-Server gestartet werden.

Dies sind in der Zentrale der Workflow-Manager, der zentrale Vorgangs-Manager, der zentrale Daten-Manager und das zentrale Parameter-System. Darüberhinaus wird die RMI-Registierungsdatenbank gestartet, mit der entfernte Methodenaufrufe über Rechnergrenzen hinweg verwaltet werden.

Auf Client-Rechnern werden dagegen nur ein lokaler Vorgangs-Manager, ein lokaler Dialog-Manager und der lokale Teil des Daten-Managers gestartet. Nach dem Start ist auf dem Client das Hauptfenster des VAA++-Systems aus Abbildung 13 zu sehen.

Das Bild zeigt den Bildschirm des Clients nach dem Anmelden des Benutzers. Man erkennt das Fenster, in dem vom Benutzer neue Workflows gestartet werden können. Der „Arbeitskorb“ beinhaltet die Vorgänge, die von diesem Benutzer aktuell zu bearbeiten sind, während im Fenster „Vertagte Vorgänge“ alle vom Benutzer auf einen bestimmten Termin vertagten Vorgänge angezeigt werden, die dann manuell auch sofort durchgeführt werden können.

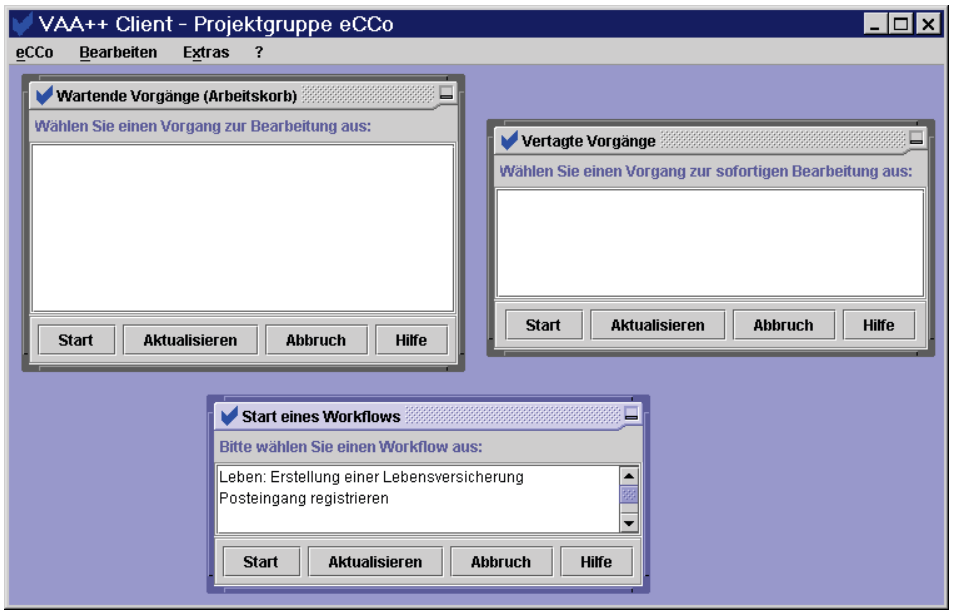

Abb. 13: Das Hauptfenster des Clients

Weiterhin gibt es zur Ausführung des gesamten Systems auf einem einzigen Rechner ein Standalone-Programm, das Client und Server vereint. Die Kommunikation zwischen Client- und Server-Seite auf einem Rechner läuft dabei wie gehabt über die Schnittstellen von RMI und Sockets.

Zur Bearbeitung von Vorgängen durch Benutzer, deren Stationen nicht im Netzwerk der Hauptverwaltung, sondern beliebig im Internet verteilt sind, gibt es zusätzlich die Möglichkeit, die gesamte Client-Seite des Systems VAA++ in einem WWW-Browser als Applet zu starten.

3.6 Anwendungsebene

Die Anwendungsebene enthält alle Anwendungsbausteine von VAA++. Es gibt allgemeine Bausteine (siehe Abbildung 14) und branchenspezifische Bausteine wie z.B. für die Versicherungsbranche. Beispiele für Versicherungs-Anwendungsbausteine werden in Kapitel 4 beschrieben. Die Anwendungsbausteine sind hierarchisch angeordnet und können sich daher nicht zyklisch gegenseitig aufrufen.

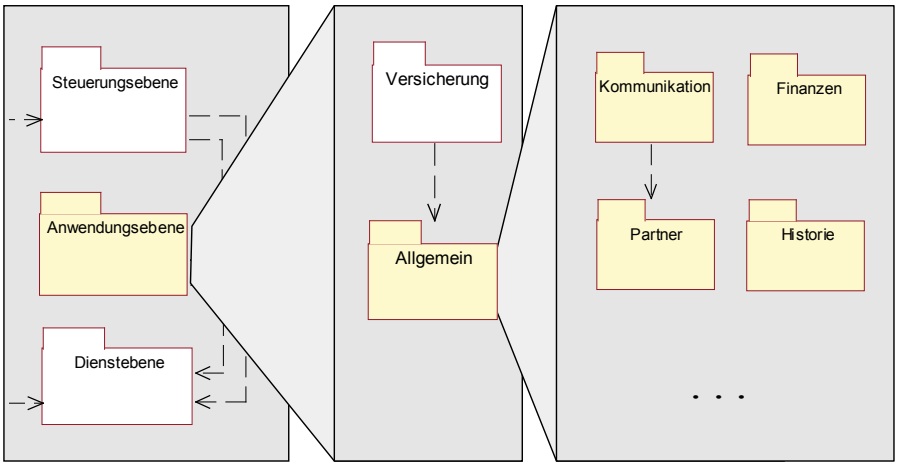

Abb. 14: Die Schichtung der Anwendungsebene und Packages des allgemeinen Teils

Allgemeine Bausteine wie zum Beispiel „Partner“, „Kommunikation“, „Finanzen“ und „Historie“ werden von allen Workflows in Anspruch genommen. Demgegenüber werden Objekte in Versicherungsbausteinen nur von Versicherungs-Workflows aufgerufen.

In den folgenden beiden Unterkapiteln werden die interessanten Bausteine „Partner“ und „Kommunikation" genauer beschrieben.

3.6.1 Anwendungsbaustein Partner

Der Anwendungsbaustein „Partner“ kapselt die Objekte, die für die Bereitstellung von Partnerdaten zuständig sind.

Zentrale Klasse in Abbildung 15 ist „Partner“, von der sowohl „NatürlicherPartner“ als auch „NichtNatürlicherPartner“ erben. Interessant ist die Klasse „GeldPartner“. Sie stellt einen Partner dar, der eine Bankverbindung besitzt. GeldPartner erbt nicht von Partner, sondern speichert lediglich einen Verweis auf einen Partner. Ein Kunde stellt eine Erweiterung eines GeldPartner dar und erbt nicht direkt von Partner, wie man annehmen könnte. Dies geschieht deshalb, weil ein Partner für das Unternehmen in verschiedenen Rollen präsent sein kann. Zum Beispiel kann ein Kunde auch gleichzeitig ein bei derselben Versicherung geführter Arzt sein.

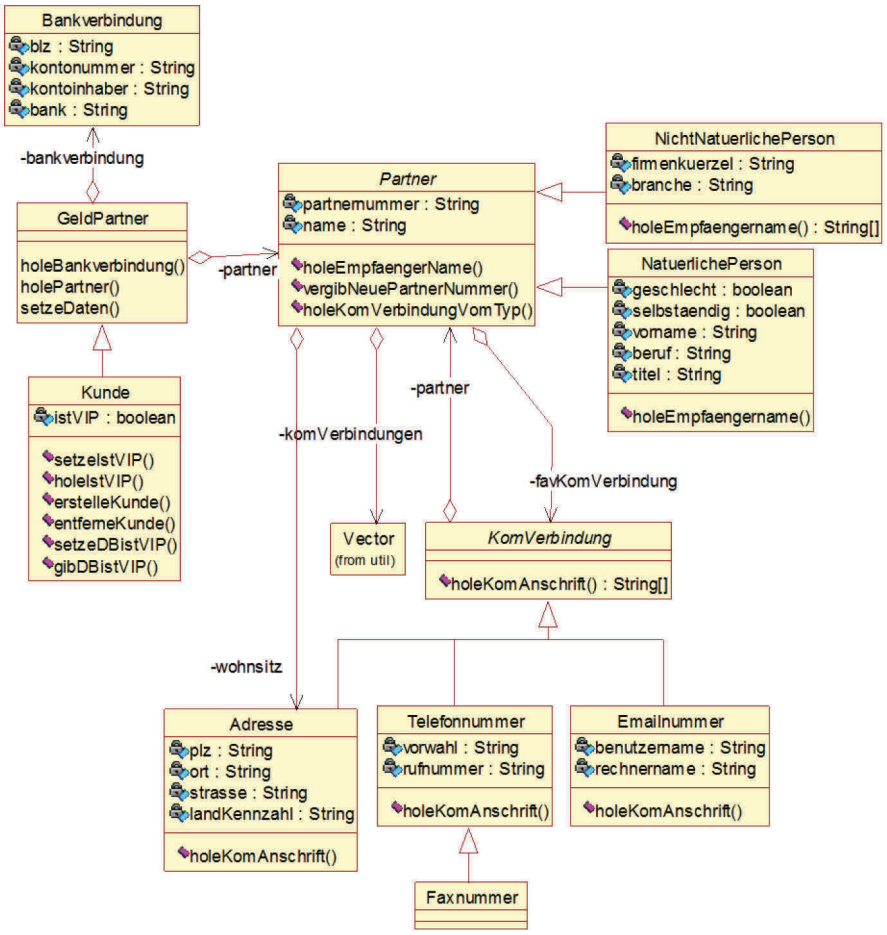

Abb. 15: Anwendungsbaustein Partner

Ein Partner kann eine oder mehrere Kommunikationsverbindungen haben, dies können Emailnummern, Telefonnummern oder Postadressen sein. Durch die Vererbungsbeziehung kann jederzeit eine neue Art von Kommunikationsbeziehungen integriert werden. Faxnummer ist ein Beispiel, wie aufbauend auf einer gegebenen Verbindung eine weitere definiert wird.

Die Abbildungen 16 und 17 zeigen, wie ein Objekt der Klasse „Partner“ in den Dialogmasken von VAA++ repräsentiert wird. In der zweiten Abbildung erkennt man insbesondere, daß ein Partner eine beliebige Anzahl von Kommunikationsverbindungen besitzen kann.

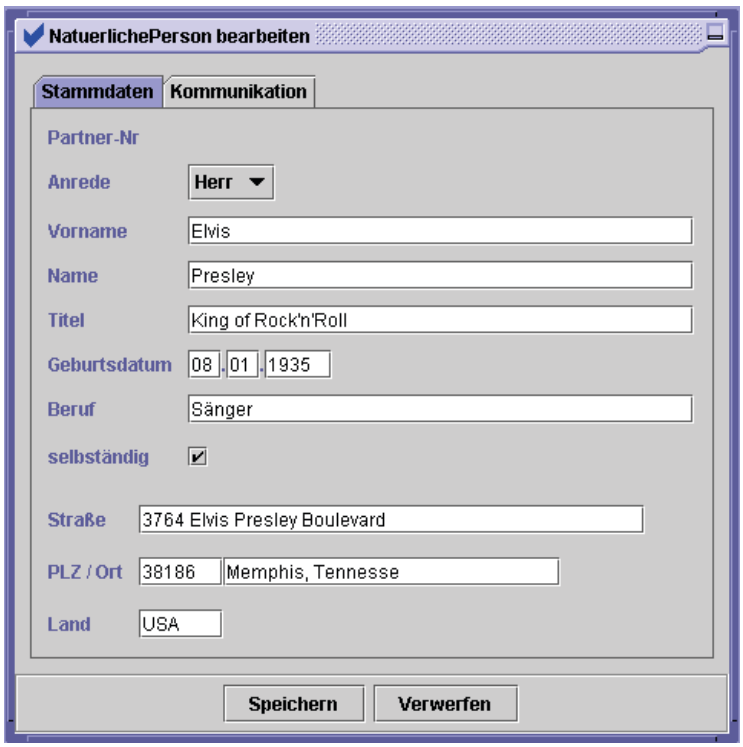

Abb. 16: Die erste Seite der Dialogmaske einer natürlichen Person

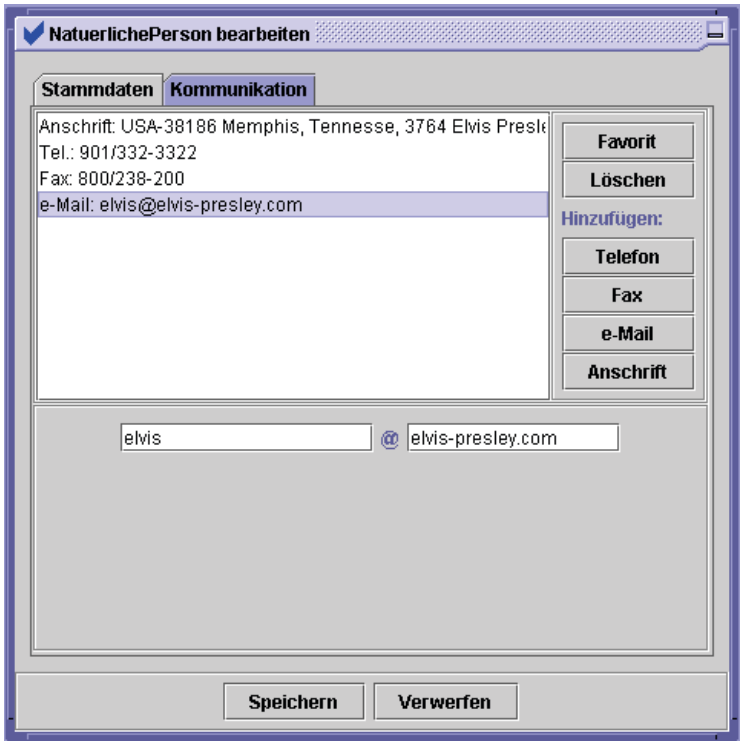

Abb. 17: Die Kommunikations-Seite der Dialogmaske eines Partners

Abbildung 18 zeigt noch einmal die verschiedenen Rollen eines Partners. Man sieht, daß sowohl Vererbung als auch die Assoziation benutzt wurden, um zu ermöglichen, daß ein Partner in mehreren Rollen bei einer Versicherung auftreten kann. So ist erkennbar, daß ein Lebensversicherter (der im Paket „Leben“ definiert wird), nicht etwa von NatürlichePerson erbt, sondern lediglich einen Verweis speichert d.h. ein Partner kann die Rolle eines Lebensversicherten spielen.

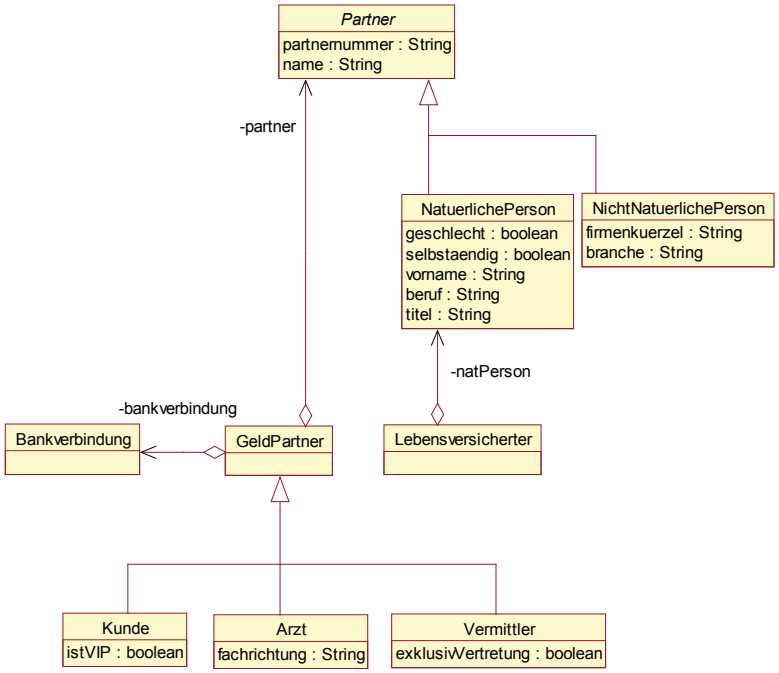

Abb. 18: Die verschiedenen Rollen eines Partners

3.6.2 Anwendungsbaustein Kommunikation

Der allgemeine Anwendungsbaustein „Kommunikation“ in Abbildung 19 stellt die Grundfunktionen für das Erstellen und Versenden von Briefen zur Verfügung.

Die Klasse „Brief“ speichert dazu eine eindeutige Brief-Identifizierungsnummer, einen Vorlagenamen, eine AusgangsRepräsentation und eine EingangsRepräsentation sowie einen Verweis auf einen Adressaten. Ein Brief wird über den Briefversender erstellt. Er wählt anhand der Art der übergebenen Zieladresse aus, ob eine Email, ein Fax oder ein Papierausdruck erzeugt werden soll. Entsprechend wird die Repräsentation als Graphik oder HTML erfolgen und gespeichert. Ein Brief muß nicht sofort nach Erstellen versendet werden. Beim Versand, der ebenfalls über den Briefversender abgewickelt wird, kann nun der geeignete Dienst (Maildienst, Faxdienst oder Druckdienst) ausgewählt werden.

Es ist möglich, daß ausgehende Briefe eine Rückantwort erwarten. Diese Rückantwort wird später als EingangsRepräsentation gespeichert.

Interessant ist die Erzeugung einer TextRepräsentation gelöst. Zusammen mit einem Brief wird der Name einer Textvorlage mit noch einzusetzenden Parametern gespeichert. Diese wird geladen und darin bei der Erstellung mit Hilfe eines im Brief gespeicherten Wörterbuchs alle Parameter ersetzt. Der fertige HTML-Text kann dann als TextRepräsentation verwendet werden.

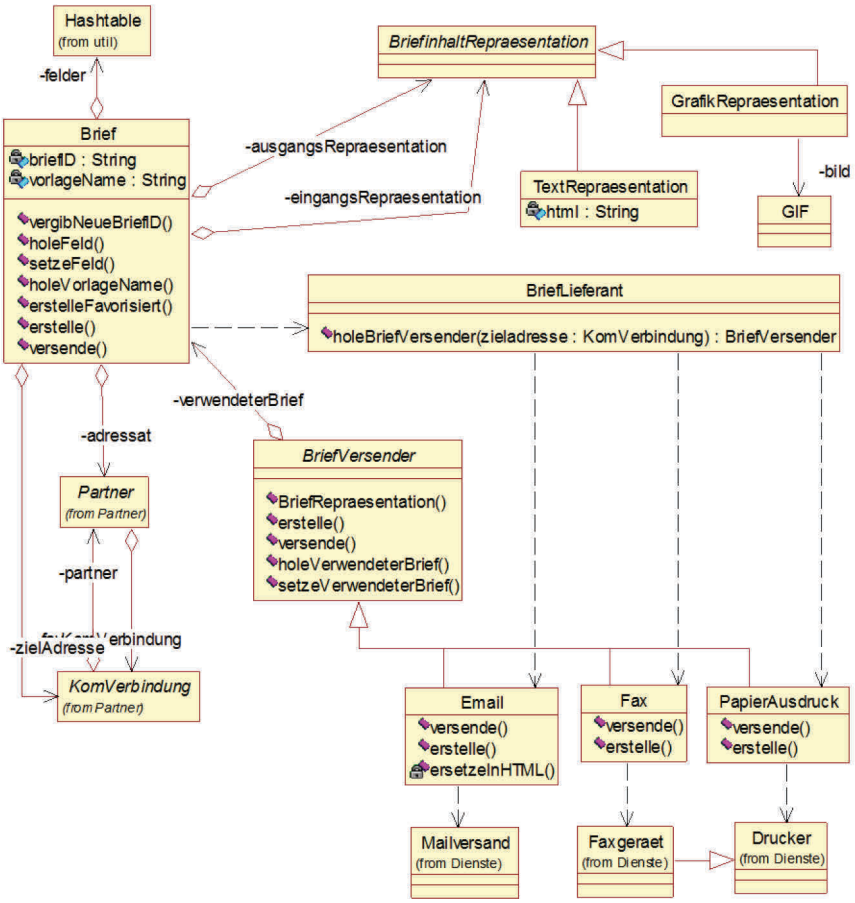

Abb. 19: Anwendungsbaustein Kommunikation

3.7 Dienstebene

Die Dienstebene enthält Bausteine für grundlegende Dienste eines EDV-Systems, die von den übergeordneten Ebenen benutzt werden können. Neben einfachen Diensten wie Mail-Versand, Fax- und Druckdienst sind hier das Parameter-System und der Daten-Manager zu finden, wie in Abbildung 20 zu sehen.

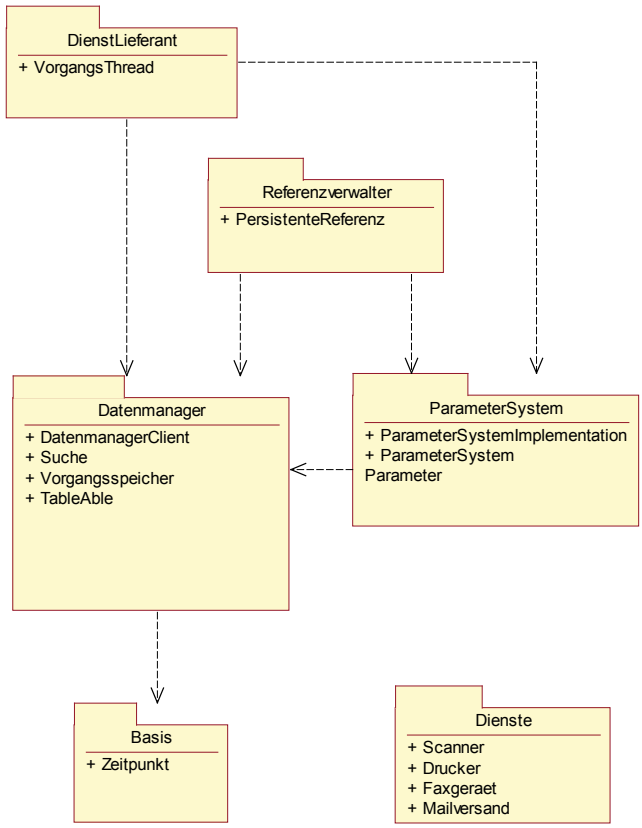

Abb. 20: Übersicht zur Dienstebene

Das Parameter-System kann benutzt werden, um Hilfsdaten zu speichern, die nicht zu den operativen Daten einer Versicherung gehören. Parameter sind Daten, die nur benötigt werden, um fachlich relevante Daten zu generieren oder zu bearbeiten. Dies können z.B. die Dialoge des Systems sein. Auch die höchste bisher vergebene Vertragsnummer wird dort abgelegt, weil sie nur zur Organisation benötigt wird und als solche keine Information für Versicherungen darstellen. Diese Daten stellt das Parameter-System im direkten Zugriff bereit. Dadurch können z.B. Dialogmasken auch von externen Editoren ausgelesen werden und nach Bearbeitung dort erneut eingespielt werden.

Der Daten-Manager verwaltet alle Daten, die entweder fachlich relevante Informationen darstellen oder aus diesen als Arbeitsergebnisse berechnet wurden. Er verbietet den direkten Zugriff auf die Daten. Stattdessen stellt er sie einem anfordernden Vorgang in dessen Vorgangsspeicher zur Verfügung, diese besitzen eine Transaktionssemantik: der Vorgang kann dort Daten ablegen und verändern und sie nach dem „Alles-Oder-Nichts-Prinzip“ gleichzeitig dem Daten-Manager für die Speicherung übergeben. Der Daten-Manager wird im folgenden Unterkapitel ausführlicher dargestellt.

In der Dienstebene ist auch der Referenzverwalter zu finden. Er sorgt dafür, daß alle Objekte, die vom Daten-Manager abgespeichert werden können, eine Identifizierungsnummer („PersistenteReferenz“) bekommen, die systemweit eindeutig ist. Diese Nummer wird in den Token von Vorgang zu Vorgang gereicht und durch den Referenzverwalter wieder auf die Objekte abgebildet.

Im Package Dienstlieferant befindet sich die Klasse „VorgangsThread“, die allen Vorgängen den Zugriff auf Parameter-System und Vorgangsspeicher ermöglicht und somit als eine Art „Laufzeitumgebung“ zum Starten von Vorgängen dient.

3.7.1 Daten-Manager

Entsprechend der Datenhaltung in der VAA ist in der VAA++ eine zentrale Datenhaltung vorgesehen. Diese Aufgabe übernimmt in der VAA++ eine in die Architektur eingebettete Daten-Manager-Komponente.

Die Entwurfsentscheidungen aus der Analyse- und Designphase dieser Daten-Manager-Komponente unterscheiden sich grundlegend von der VAA-Architektur. Drei wesentliche Besonderheiten des Daten-Managers der VAA++ seien hier kurz erläutert.

Die objektorientierte VAA++ gibt vor, daß die zu bearbeitenden Daten, z.B. Kunden und Verträge, Objektgestalt haben müssen. Daraus ergibt sich das Problem, in einem verteilten System für die Eindeutigkeit der Objektidentitäten zu sorgen. Dieses ist eine Aufgabe des hier vorgestellten Daten-Manager-Systems. Der Daten-Manager hält als einziges System in der VAA++ die systemweit eindeutigen Identitäten der Datenobjekte vor. Im Daten-Manager existieren einmalig die eigentlichen Datenobjekte des Systems, und die Komponente sorgt selbständig für Datenbanktransaktionen wie Speichern, Löschen und Laden, Persistenz und Transienz der Objekte. Das Anwendungssystem ist somit von der eigentlichen Datenhaltung entkoppelt. Es greift scheinbar auf immerwährend existierende Objekte zu, unabhängig von darunterliegenden Datenbanktransaktionen.

Um die Stabilität der Daten innerhalb der Datenobjekte und die Stabilität der Relationen zwischen den Datenobjekten zu gewährleisten, erhält das gesamte Anwendungssystem ausschließlich Kopien der tatsächlich vorgehaltenen Datenobjekte. Alle Vorgänge des Anwendungssystems erhalten zur Laufzeit jeweils die Datenobjektkopien, die sie zur Bearbeitung anfordern. Bei Beendigung eines Vorgangs werden alle bearbeiteten Objektkopien des Vorgangsspeichers vom Daten-Manager zum gleichen Zeitpunkt aktualisiert. So wird ein Konflikt zwischen den Transaktionen verschiedener Vorgänge vermieden und Datenkonsistenz des Systems ermöglicht.

Innerhalb des Daten-Managers werden alle Objekte mit einer eindeutigen Nummer, einem Surrogat, versehen. Dies ermöglicht, daß nach dem Laden eines Objektes aus der Datenbank die eigentliche Objektidentität wiederhergestellt werden kann. Das Objekt kann nun wie zuvor eingesetzt werden, und Objektbeziehungen wie Referenzen oder Vererbungen bleiben zur Laufzeit schlüssig.

Architektur

Im folgenden wird die Architektur des VAA++-Daten-Manager-Systems erläutert. Der Daten-Manager integriert sich in die Client-Server-Struktur der VAA++ dabei seinerseits ebenfalls als eigenständiges Client-Server-System.

Abbildung 21 zeigt eine Übersicht über die Klassen des Daten-Manager-Clients. Der clientseitige Daten-Manager besteht aus der Klasse „DatenmanagerClientImplementation“, einer Sucheinheit „SucheImplementation“ und der Klasse „VorgangsspeicherImplementation“.

Vorgangsspeicher und Sucheinheit stellen die eigentliche Schnittstelle für einen Anwendungsbaustein dar. Der Anwendung werden diese Funktionseinheiten dabei in Form von Interfaces zur Verfügung gestellt und ihre Implementierung in der jeweiligen Implementationsklasse vorgehalten.

Die tatsächliche Datenbearbeitung durch einen Anwendungsbaustein findet im Vorgangsspeicher statt. Hier werden alle benötigten Datenobjekte, genauer die Kopien der Datenobjekte, eines Vorgangs dem jeweiligen Anwendungsbaustein vorgehalten und Methoden zur Bearbeitung der Objekte bereitgestellt. Der Anwendungsbaustein greift hier somit direkt auf seine

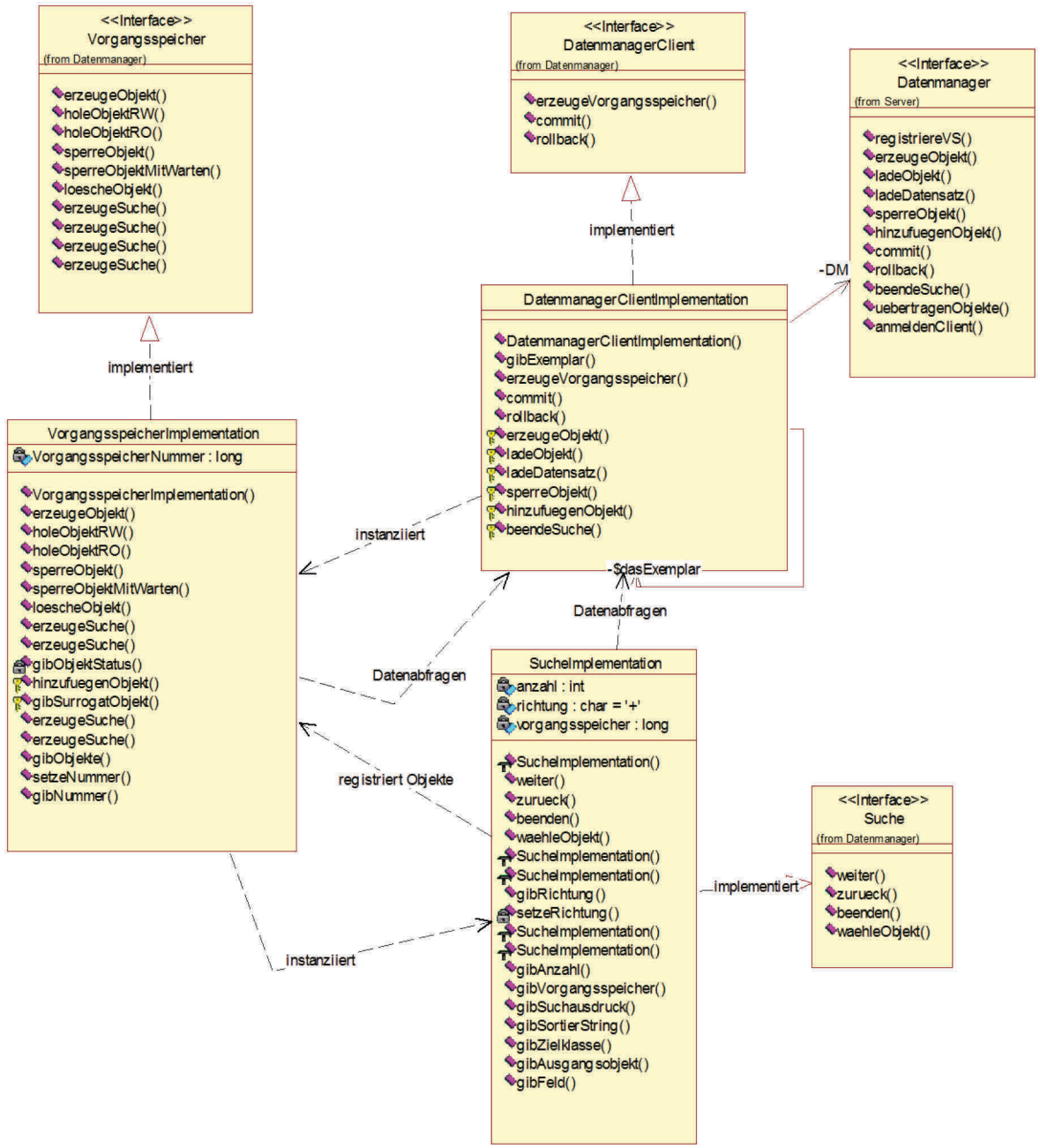

Abb. 21: Daten-Manager-Client

Datenobjekte zu, verändert sie und gibt bei Beendigung des Vorgangs alle bearbeiteten Objekte gleichzeitig zur Transaktion frei. Alle Objekte eines Vorgangsspeichers werden somit bei der Freigabe gleichzeitig dem Daten-Manager-Server zur Transaktion übergeben. Die DatenmanagerClientImplementation bildet dazu die Verbindung von der Clientseite des Systems zum Server mit der Klasse DatenmanagerImplementation.

Abbildung 22 gibt dazu einen Überblick über die Klassenstruktur des Daten-Manager-Server-Systems. Hauptbestandteil neben dem „Surrogatverwalter“ und der „DBTransaktion“ ist die „DatenmanagerImplementation“ mit der „Objekttabelle“. Neben der Schnittstellenfunktion des Daten-Managers zu den Clientsystemen werden hier alle Objekte, die im System bearbeitet werden, in Form von „Surrogatobjekten“ im Hauptspeicher bereitgehalten. Bei Bedarf erhalten die Anwendungssysteme die benötigten Datenobjekte als Kopien der in der Objekttabelle unter Verschluß gehaltenen Surrogatobjekte. Sobald ein Datenobjekt von keinem Clientsystem mehr verwendet wird, wird es der Datenbankschnittstelle übergeben, die in der Klasse DBTransaktion implementiert ist. Sobald andererseits ein Daten-Manager-Client ein Objekt anfordert, das noch kein weiterer Client geöffnet hält (und das somit nicht transient in der Objekttabelle, sondern persistent in der Datenbank gehalten liegt), wird es aus dem Datenbanksystem gelesen, zu dem benötigten Objekt instanziiert und der Clientseite über den Vorgangsspeicher zur Verfügung gestellt.

Um Kollisionen bei der Bearbeitung von Objekten durch mehrere Clients zu verhindern, verwaltet die Objekttabelle für ihre Objekte Zugriffsrechte. Außerdem benötigt das System ein-

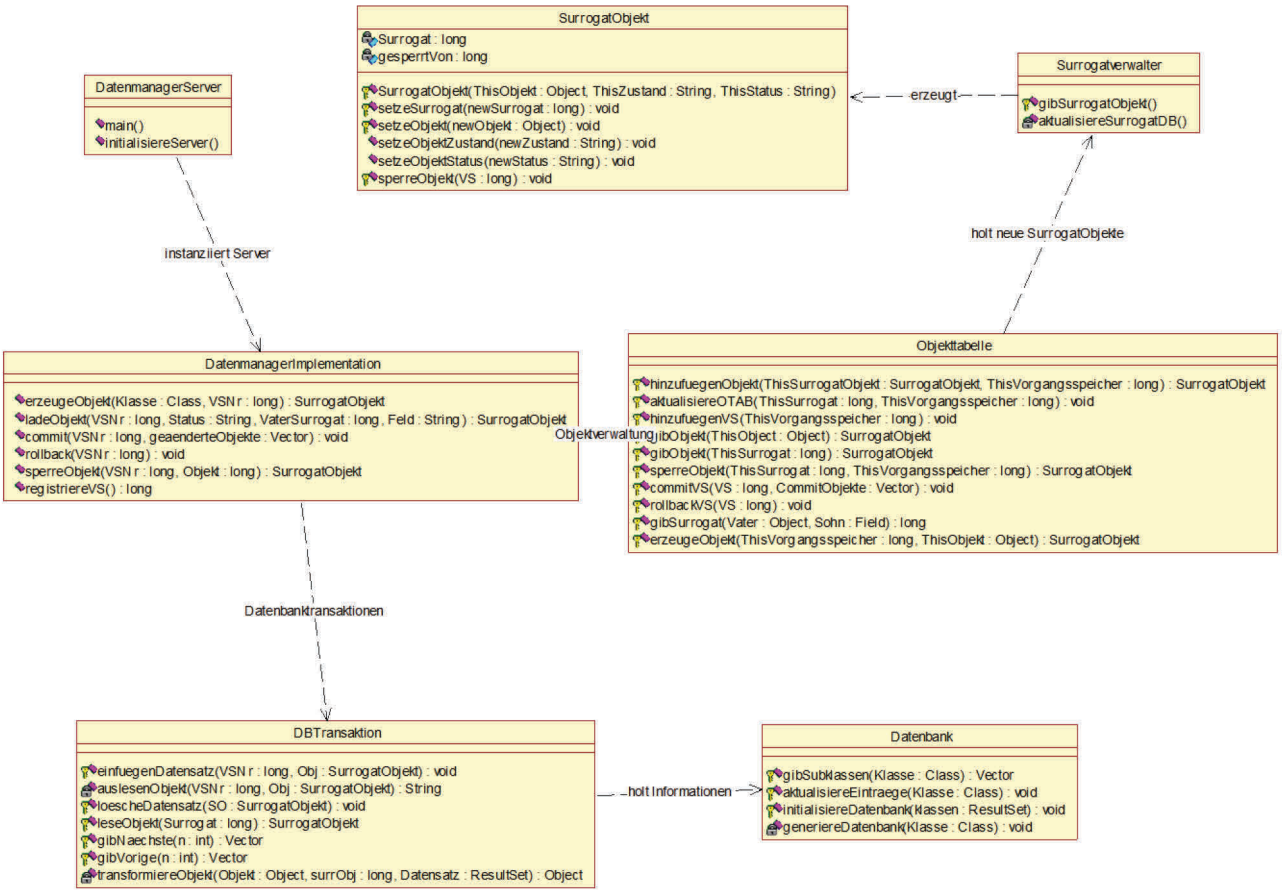

Abb. 22: Daten-Manager-Server

deutige Identifikatoren, die die Identität von Objektinstanzen sicherstellen. Die Klasse Surrogatverwalter vergibt zu diesem Zweck jedem Datenobjekt ein systemweit eindeutiges Surrogat.

Die Transaktionen zwischen dem Daten-Manager und den Datenbanksystemen schließlich bearbeiten die Klassen DBTransaktion und Datenbank. Diese Komponenten regeln eigenständig die Speicherung der Objekte in der Datenbank. Bei der Speicherung von Objekten werden die benötigten Tabellen und Spalten dynamisch ausgewählt und ggf. neu erzeugt. Dies ermöglicht sogar das Speichern von bisher unbekannten Objekten in der Datenbank, ohne manuelles Generieren neuer Datenmodelle oder manuelle Wartung der Datenbanktabellen. Die DBTransaktion verwaltet dabei nicht nur die Objekte, sondern auch deren Relationen wie Polymorphie, Vererbung und Referenzen.

Der hier vorgestellte Aufbau bewirkt, daß im Unterschied zu der VAA-Architektur kein unternehmensweites Datenmodell mehr benötigt wird. Auch die logischen Datensichten können entfallen. Statt logischer Teilsichten bei der Bearbeitung der Daten werden komplette Einzelobjekte verwaltet und fehlende referenzierte Objekte dynamisch nachgefordert.

Schnittstellen zum Anwendungssystem

Der Daten-Manager bietet dem Anwendungssystem drei Interfaces auf der Clientseite als Schnittstelle zur Datenhaltung an: Die Interfaces „DatenmanagerClient“, „Vorgangsspeicher“ und „Suche“. Diese Klassen bieten der Anwendung alle Methoden, die zur Datenbearbeitung erforderlich sind.

Beim Start eines neuen Vorgangs innerhalb der Anwendung kann durch den Aufruf von „erzeugeVorgangsspeicher“ im Interface DatenmanagerClient (Abbildung 23) die Transaktionsschnittstelle des Daten-Managers geöffnet werden. Ein neuer, leerer Vorgangsspeicher wird auf dem Clientsystem instanziiert. Nach Beendigung eines Vorgangs wird entweder durch den Aufruf von „commit“ die Übergabe der bearbeiteten Objekte an das Serversystem initiiert oder durch den Aufruf von „rollback“ der Vorgang ohne Transaktion beendet und in beiden Fällen der Vorgangsspeicher gelöscht.

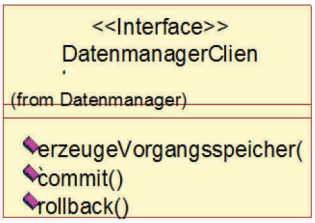

Abb.23: Interface DatenmanagerClient

Der Vorgangsspeicher aus Abbildung 24 bildet die Datenbearbeitungsschnittstelle. Die Anwendung bekommt durch ihn den benötigten Satz an Methoden zur Verfügung gestellt, die sie zur Bearbeitung der Datenobjekte verwenden kann. Dies sind die Methoden „erzeugeObjekt“ zum Neuerstellen eines Objekts, „holeObjekt“ zum Nachladen eines Objekts in den Vorgangsspeicher sowie „loescheObjekt“, um ein Objekt aus dem Gesamtsystem zu entfernen. Desweiteren lädt „sperreObjekt“ ein Objekt nachträglich im Schreiblesend-Zugriff, nachdem es zuvor im Lesend-Zugriff geöffnet wurde.

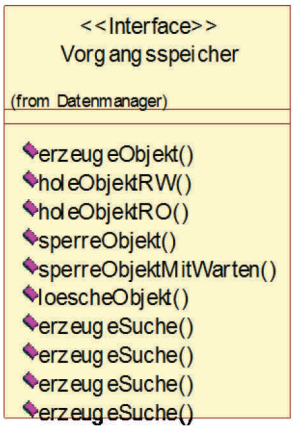

Abb.24: Interface Vorgangsspeicher

Mit der Methode „erzeugeSuche“ kann die Anwendung eine Suche im Daten-Manager-System beginnen. „erzeugeSuche“

erhält dabei unter anderem SQL-ähnliche Befehle als Sucheingabe, um nach bestimmten Datenobjektattributen zu suchen.

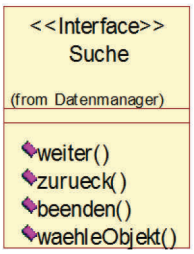

Abb.25:
Interface Suche

Das von der Methode erzeugte Suchobjekt in Abbildung 25 bietet nun die Möglichkeit, durch die Methoden „weiter“ und „zurueck“ in Schritten die Suchergebnisse zu ermitteln und zur Auswahl bereitzustellen. Die Auswahl von Objekten erfolgt dabei über die Methode „waehleObjekt“, woraufhin sie in den Vorgangsspeicher geladen werden. Die Methode „beenden“ löscht schließlich das Suchobjekt wieder und beendet dadurch die Suche.

Das Kapitel 3 gab einen Überblick über die Implementierung des Systems VAA++. Insbesondere wurde auf die Bestandteile der Steuerungsebene eingegangen, dies sind Dialog-, Vorgangs- und Workflow-Manager sowie das Anpassungspaket. Das Konzept der Workflow-Graphen und einige Knotentypen wurden vorgestellt. Die unter der Steuerungsebene liegende Anwendungsebene wurde anhand der allgemeinen Anwendungsbausteine Kommunikation und Partner beleuchtet, wobei insbesondere die verschiedenen Rollen eines Partners berücksichtigt wurden. Aus der Dienstebene wurde ausführlich der Daten-Manager und damit die Art und Weise des Datenzugriffs in der VAA++ beschrieben.

In den folgenden beiden Kapiteln wird nun die Benutzung des geschaffenen Systems anhand der durch die Projektgruppe ***eCCo*** realisierten Geschäftsprozesse aus der Versicherungswirtschaft dargestellt.

4 Der Geschäftsprozeß Lebensversicherungs-Antrag

Mit dem in Kapitel 3 vorgestellten System VAA++ wurde neben dem in Kapitel 5 vorgestellten Geschäftsprozeß „Kraftfahrzeug-Schaden“ auch der hier vorgestellte „Lebensversicherungs-Antrag“ implementiert. Dazu wurden Workflow-Graphen erstellt, die den Ablauf des Geschäftsprozesses definieren. Die einzelnen Teilschritte wurden in Vorgängen codiert, welche zur Benutzerinteraktion teilweise Dialoge benutzen. Zur Speicherung und Berechnung von Daten wurden schließlich die benötigten Anwendungsbausteine implementiert, die ebenfalls von den Vorgängen aufgerufen werden. Die Objekte der Anwendungsbausteine werden vom Daten-Manager in seiner Datenbank abgelegt.

In diesem Kapitel werden die Workflow-Graphen für den Geschäftsprozeß „Lebensversicherungs-Antrag“ mit den in Kapitel 3 eingeführten Knotentypen beschrieben (die Vorgangs-Knoten mit den jeweils aufgerufenen Vorgängen sind in Abbildung 26 grau unterlegt). Anschließend wird der Anwendungsbaustein „Leben“ erläutert und Beispiele für Dialoge des Geschäftsprozesses gezeigt.

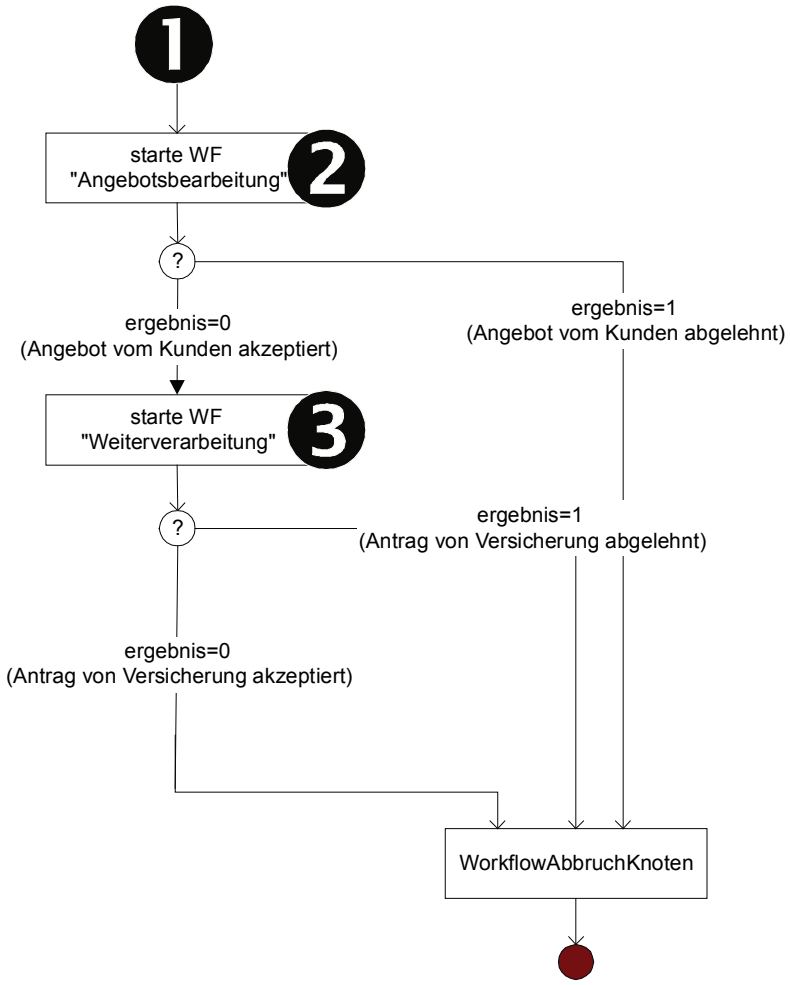

Abb. 26: Haupt-Workflow (1)

Der Geschäftsprozeß „Lebensversicherungs-Antrag bearbeiten“ beschreibt, wie bei einer Lebensversicherung der Weg von der Antragserfassung beim Kunden bis zur Policierung des Vertrages aussehen kann. Zentraler Bestandteil ist dabei die Risikoprüfung. Dieser Geschäftsprozeß wird verteilt bearbeitet, mehrere Beteiligte an mehreren Orten werden einbezogen.

Der Geschäftsprozeß besteht aus den in Abbildung 26 erkennbaren Phasen „Angebotsbearbeitung“ und „Weiterverarbeitung“. Diese zweite Phase gliedert sich wiederum in die in Abbildung 28 dargestellten Phasen „Vorprüfungen“, „Risikoprüfung“, „Freigabe“ und „Verbuchung“. Nach Rückkehr aus jeder dieser Phasen kann der Geschäftsprozeß abgebrochen werden.

4.1 Workflow „Angebotsbearbeitung“

Während der Angebotsbearbeitung gibt der Kundenbetreuer die Antragsdaten in das System ein, druckt ein oder mehrere Antragspapiere aus und läßt diese durch den Kunden unterschreiben. Danach sendet der Kundenbetreuer sowohl den elektronischen Datensatz als auch das unterschriebene Antragspapier an die Hauptverwaltung.

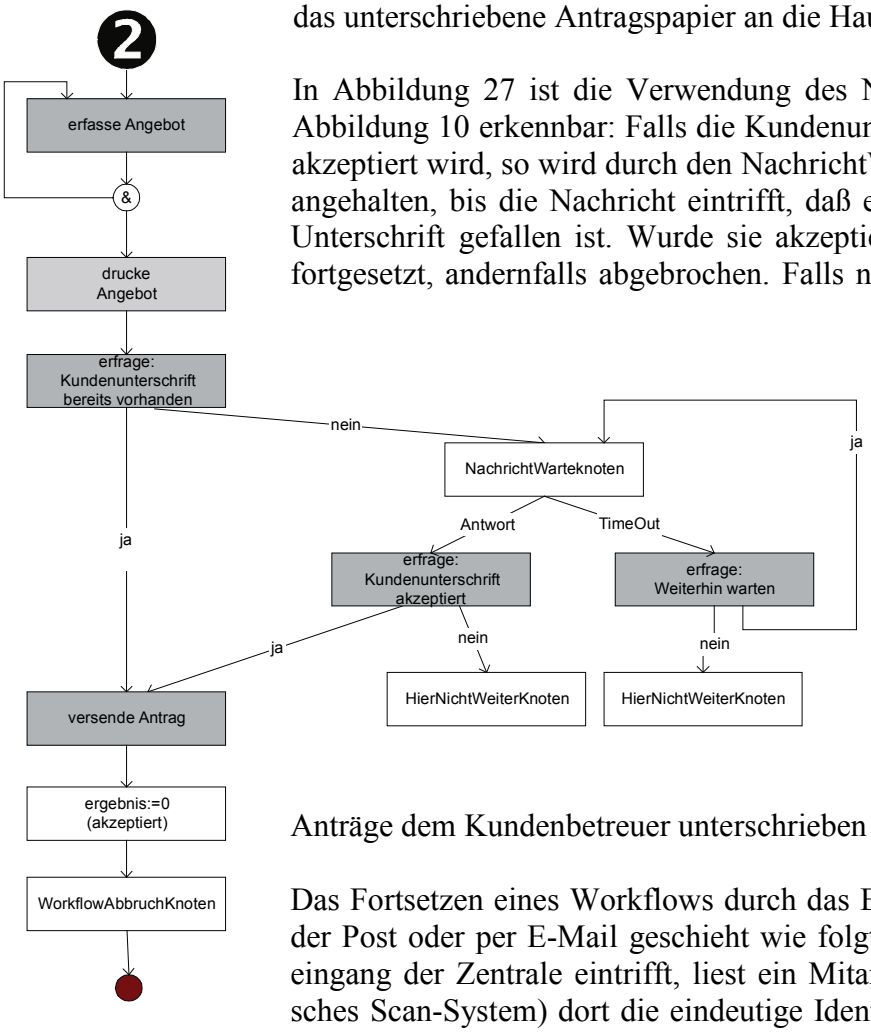

Abb. 27: Workflow (2) „Angebotsbearbeitung“

In Abbildung 27 ist die Verwendung des NachrichtWarteknotens aus Abbildung 10 erkennbar: Falls die Kundenunterschrift noch nicht sofort akzeptiert wird, so wird durch den NachrichtWarteknoten der Workflow angehalten, bis die Nachricht eintrifft, daß eine Entscheidung über die Unterschrift gefallen ist. Wurde sie akzeptiert, so wird der Workflow fortgesetzt, andernfalls abgebrochen. Falls nach einer festgelegten Zeit diese Nachricht noch nicht akzeptiert worden ist, wird eine Nachfrage angezeigt, ob noch weiter gewartet werden soll. In diesem Fall beginnt die Warteprozedur erneut. Diese Konstruktion ermöglicht, daß der Kunde die Antragsformulare einige Zeit behalten und anschließend einen der Anträge dem Kundenbetreuer unterschrieben zukommen lassen kann.

Das Fortsetzen eines Workflows durch das Eintreffen eines Briefes mit der Post oder per E-Mail geschieht wie folgt: Wenn ein Brief im Posteingang der Zentrale eintrifft, liest ein Mitarbeiter (bzw. ein automatisches Scan-System) dort die eindeutige Identifizierungsnummer ab und gibt sie in das System ein (falls eine solche aufgedruckte Brief-Nummer fehlt, kann mit diesem Brief kein wartender Workflow fortgesetzt wer-

den). Alle gespeicherten Briefe werden untersucht, ob ihre Briefnummer mit der des eingetroffenen Briefes übereinstimmt. Jeder Brief, bei dem dies der Fall ist, wird als Nachricht an den Workflow-Manager geschickt, der die auf diesen Brief wartenden Workflows fortsetzt.

Nach Beendigung dieser ersten Phase beim Kundenbetreuer wird der Geschäftsprozeß in der Hauptverwaltung von anderen Personen fortgesetzt. Beim implementierten Prototyp findet jedoch kein Systemwechsel vom Rechner des Kundenbetreuers zum Zentralrechner statt. Stattdessen wählt sich der Kundenbetreuer in den Zentralrechner ein und startet von dort aus den Client-Teil des Systems VAA++ als Applet in seinem WWW-Browser. Somit wird der Workflow von Beginn an durch den Workflow-Manager der Hauptverwaltung durchgeführt.

4.2 Workflow „Weiterverarbeitung“

Die zweite, in Abbildung 28 dargestellte Phase des Geschäftsprozesses besteht aus dem sukzessiven Aufruf der vier Workflows „Vorprüfungen“, „Risikoprüfung“, „Freigabe“ und „Verbuchung“. Nach Rückkehr von jedem dieser Aufrufe mit ablehnendem Ergebnis kann der Geschäftsprozeß abgebrochen werden.

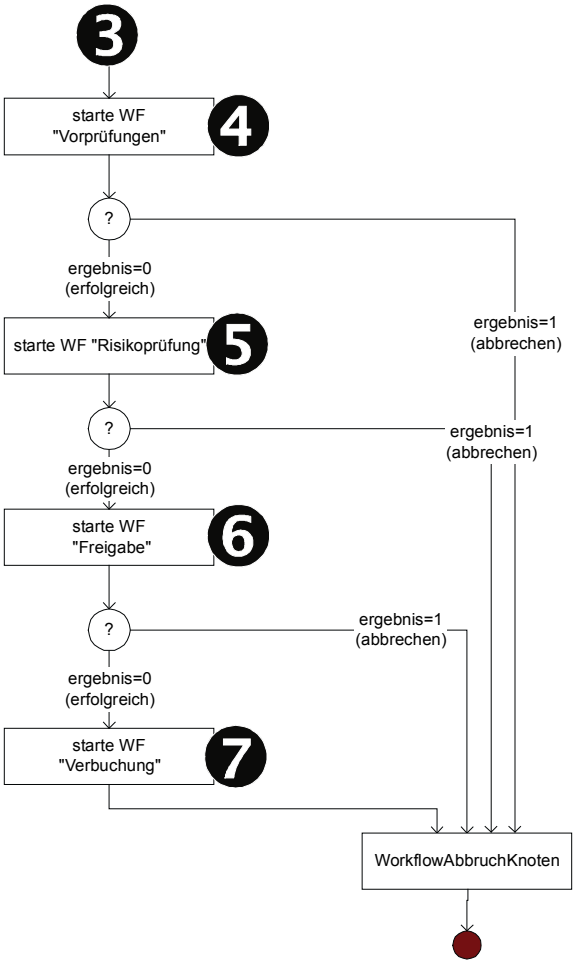

Abb. 28: Workflow (3) „Weiterverarbeitung“

4.2.1 Workflow „Vorprüfungen“

Die Vorprüfungen beginnen mit der Vergabe einer eindeutigen Versicherungsnummer für den Antrag, wie in Abbildung 29 zu sehen. Danach werden durch einen Kopierknoten drei Vorgänge angestoßen, deren Ergebnisse später in einem Synchronisations-Knoten zusammengeführt werden. In einem Vorgang wird das Ergebnis von einfachen maschinellen Prüfungen des Antrags auf Vollständigkeit und Plausibilität bestimmt, das Ergebnis des zweiten Vorgangs ist das Anfrageergebnis bei der Sonderwagnis-Datei der Versicherer. Dort werden Antragsteller gespeichert, deren Anträge bei anderen Versicherungen abgelehnt wurden.

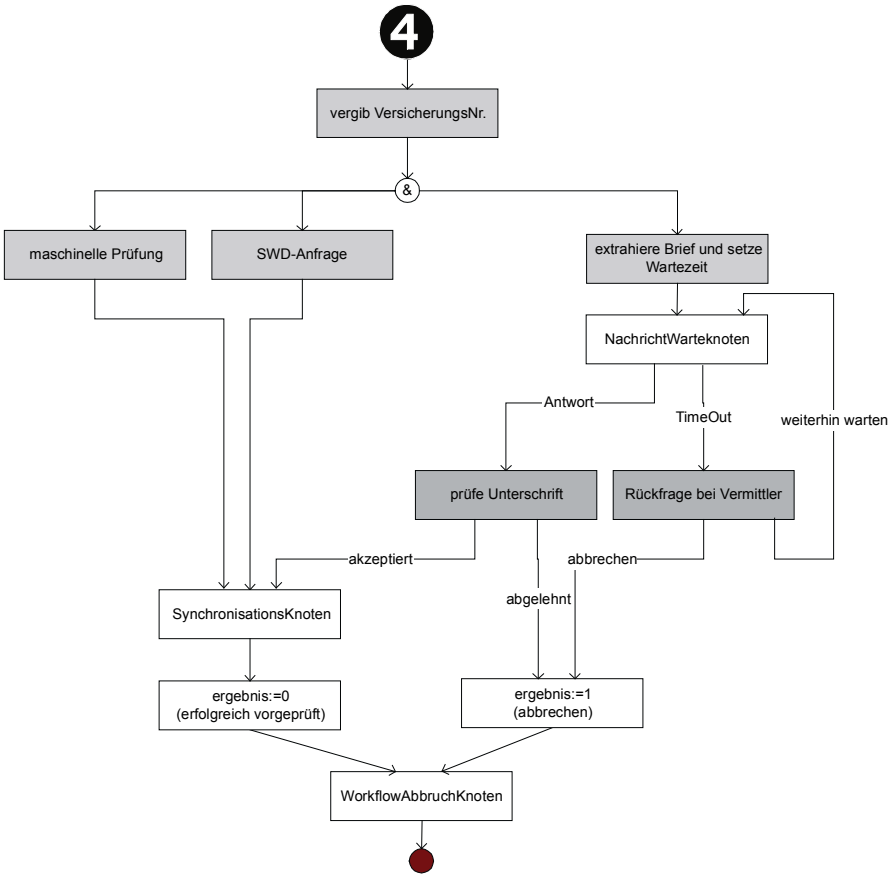

Abb. 29: Workflow (4) „Vorprüfungen“

Der dritte Vorgang stößt eine Reihe von Vorgängen an, mit denen auf das Eintreffen des Papierantrags in der Zentrale gewartet wird (NachrichtWarteknoten) und danach die Kundenunterschrift auf dem Antragspapier geprüft wird. Der Synchronisations-Knoten bekommt also als drittes Token die Entscheidung, ob die Unterschrift akzeptiert wird. Damit wird dieser Teil-Workflow beendet.

4.2.2 Workflow „Risikoprüfung“

Im Anschluß wird der Subworkflow „Risikoprüfung“ aus Abbildung 30 durchgeführt mit dem Ziel einer Entscheidung, ob der Kunde eine Police erhält oder abgelehnt wird. Der Sachbear-

beiter kann dazu beliebig viele Briefe an Ärzte verschicken, die den Kunden behandeln, und von diesen Informationen anfordern. Er kann auch ein Anomalieangebot erstellen und an den Kunden versenden. Damit werden kleinere Änderungen der Vertragsbedingungen vereinbart, auf die sich der Kunde und die Versicherung einigen können.

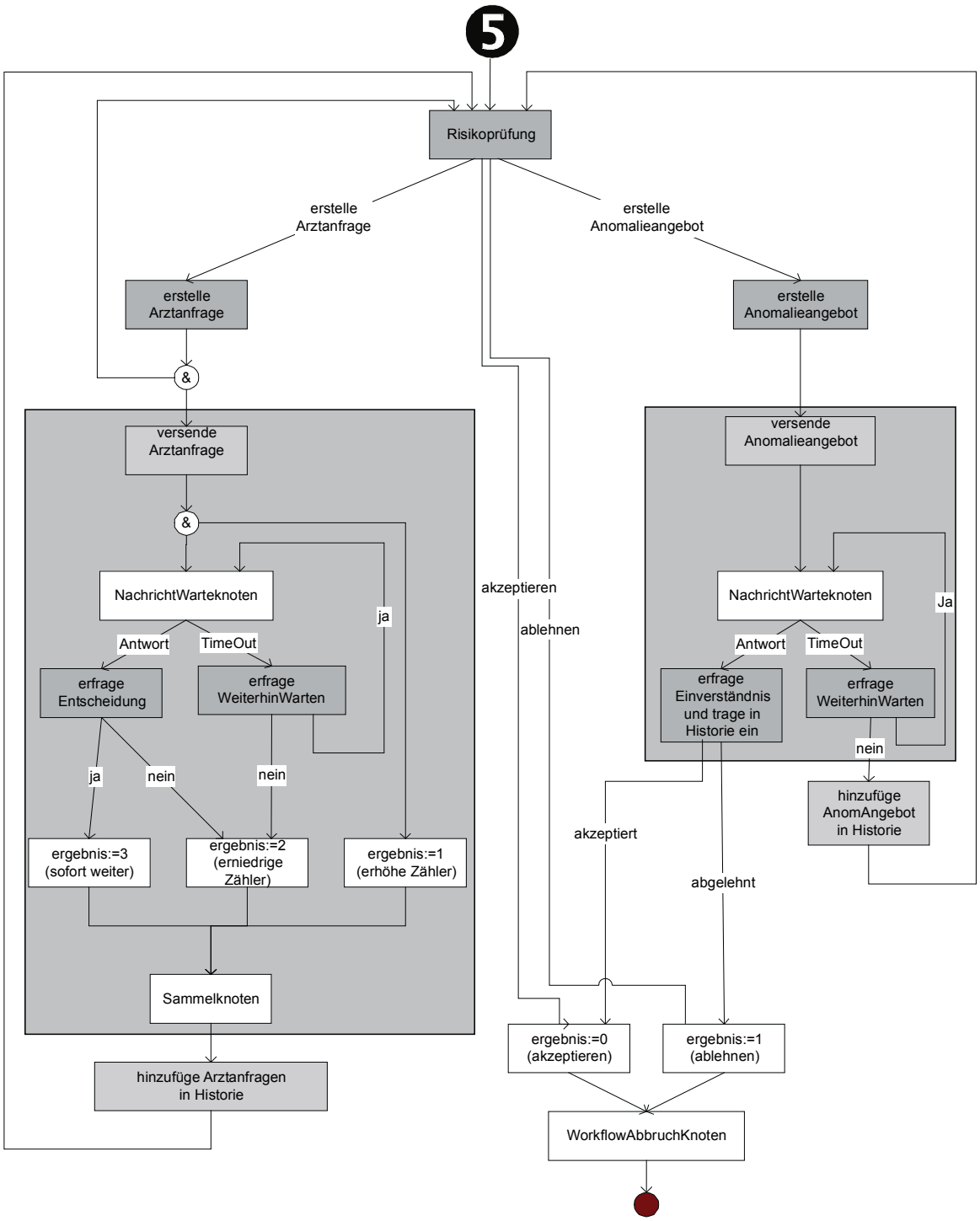

Abb. 30: Workflow (5) „Risikoprüfung“

In Abbildung 30 sind zwei Blöcke erkennbar, von denen der linke für Erstellung und Versand von beliebig vielen Arztanfragen und das Warten auf Rückantworten zuständig ist. Dieses

Warten kann jederzeit beendet werden, dadurch wird auf die ausstehenden Rückantworten nicht mehr gewartet. Der andere Block stellt Erstellung und Versand eines Anomalienagebotes und das Warten auf die Kundenantwort dar.

Interessant ist, daß im Arztanfragen-Block durch die Zusammenschaltung von NachrichtWarteknoten und Sammel-Knoten ein komplexes Muster modelliert wurde, welches auch allgemein Einsatz finden kann:

Vor Eintritt in den NachrichtWarteknoten wird für jede versendete Arztanfrage der Zähler des Sammel-Knotens um Eins erhöht, d.h. nun wird auf das Eintreffen eines weiteren Token gewartet. Der Sammel-Knoten wartet auf eine festzulegende Anzahl von Objekten (Arztanfragen-Antworten). Er wird durch drei Vorgänge gesteuert, die entsprechende Ergebnis-Werte besitzen.

Im NachrichtWarteknoten wird auf die Nachricht über eine Antwort zu einer versendeten Arztanfrage gewartet. Bei Eintreffen der Antwort wird der Sachbearbeiter befragt, ob er eine Entscheidung bezüglich der Einstufung des Kunden in der Risikoprüfung treffen möchte. Abhängig davon wird entweder bei einer möglichen Entscheidung das Warten auf ausstehende Objekte sofort beendet, und alle bisher gesammelten Objekte werden als Array im Token weitergeleitet. Oder es wird – falls noch keine Entscheidung getroffen werden konnte – lediglich der Zähler für die Anzahl, auf die gewartet werden soll, erniedrigt, weil nun eine Arztanfrage weniger ausstehend ist.

Abhängig von der anschließenden Entscheidung in der Risikoprüfung wird der Geschäftsprozeß entweder abgebrochen oder fortgesetzt.

4.2.3 Workflow „Freigabe“

Im auf die Risikoprüfung folgenden Freigabeverfahren aus Abbildung 31 wird beispielhaft bei einer hohen Vertragssumme eine Autorisierung durch einen Vorgesetzten gefordert. Auch hier ist die Verteilung des Geschäftsprozesses auf mehrere Bearbeiter und Orte erkennbar. Nach erfolgter Autorisierung wird der Antrag in einen Vertrag umgewandelt.

4.2.4 Workflow „Verbuchungen“

In der letzten Phase werden Verbuchungsarbeiten durchgeführt. Die Berechnung der Beitragssollstellung und der Druck der Police werden parallel ausgeführt, wie in Abbildung 32 zu sehen. Der Teil-Workflow wird beendet, wenn beide Vorgänge abgeschlossen sind.

Damit ist der Geschäftsprozeß beendet.

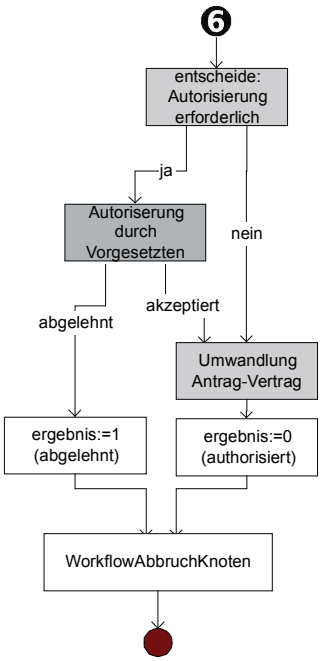

Abb. 31: Workflow (6) „Freigabe“

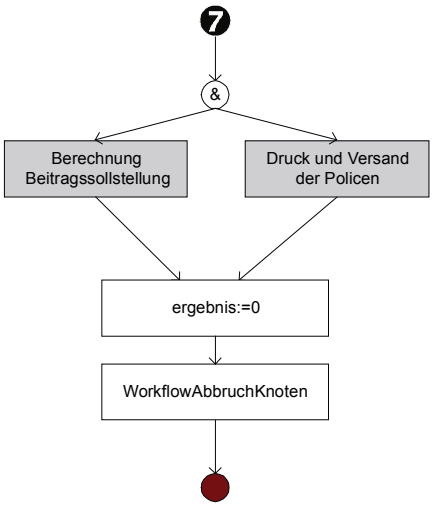

Abb. 32: Workflow (7) „Verbuchungen“

4.3 Anwendungsbaustein „Leben“

Alle Anwendungsbausteine, die spezielle Funktionalität für die Lebensversicherungs-Branche anbieten, werden im Paket „Versicherung“ der Anwendungsebene abgelegt, wovon ein Ausschnitt in Abbildung 33 zu sehen ist. Zu ihnen gehören u. a. die Anwendungsbausteine „Leben“, „Vertrag“, „Sonderwagnis-Datei“ und „Versicherungs-Partner“. Die hier ebenfalls dargestellten Bausteine „Kommunikation“, „Partner“ und „Finanzen“ stammen aus dem Paket „Allgemein“ und werden in der gezeigten Weise von den Versicherungs-Anwendungsbausteinen benutzt. Die Aufruf-Hierarchie, die insbesondere zirkuläre Aufrufe verbietet, ist deutlich zu erkennen.

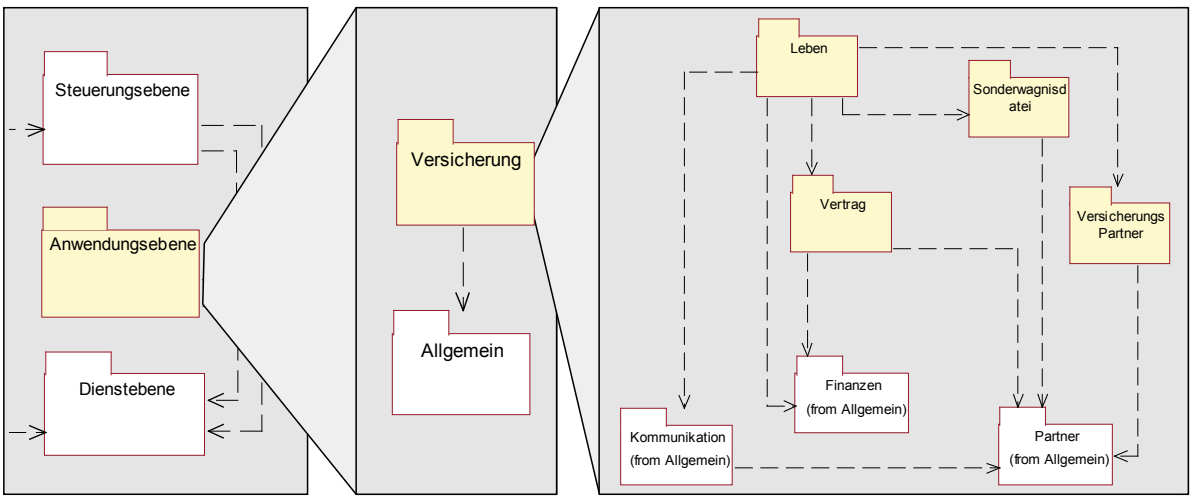

Abb. 33: Die Schichtung der Anwendungsebene und die Hierarchie des Versicherungs-Teils

Im Paket „Leben“ liegen die Anwendungsobjekte der Sparte „Lebensversicherungen“. Im oberen Teil des Klassendiagramms [Oes98] in Abbildung 34 erkennt man die Erweiterung der Klasse „VertragsDaten“, der Grundlage aller Versicherungsverträge, zur neuen Klasse „LVDaten“. Sie bildet eine Art Vertragskopf, der sich in den speziell benötigten Klassen „LVAntrag“, „LVVertrag“ und „LVAngebot“ wiederfindet. Im unteren Bildbereich sind die die benötigten Briefe AngebotsFormular, Anomalienagebot, Arztanfrage und LebenPolice zu sehen, die von der Klasse „Brief“ erben.

Weiterhin ist zu beobachten, daß – entgegen der intuitiven Vermutung – ein Lebensversicherter und auch ein Kunde nicht Erweiterungen eines natürlichen Partners des Partner-Anwendungsbausteins sind. Würden z.B. die Klassen Lebensversicherter und Arzt mit Vererbung modelliert, dann könnte der Fall, daß ein Mensch eine Lebensversicherung abschließt und gleichzeitig dort als Vertrauensarzt geführt wird, nicht abgedeckt werden. Durch die Benutzung der Referenz kann jedoch ein natürlicher Partner die Rollen eines Kunden, eines Lebensversicherten und eines Arztes gleichzeitig spielen.

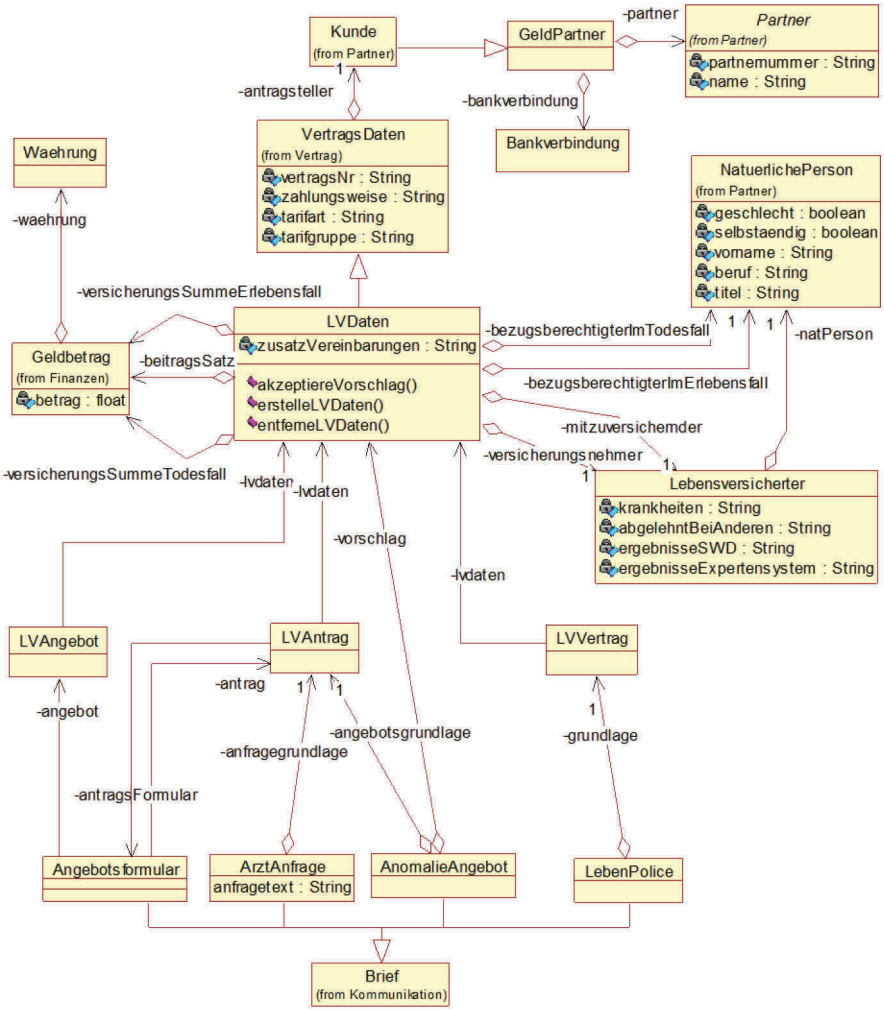

Abb. 34: Anwendungsbaustein „Leben“

4.4 Beispiele für Dialoge

Im folgenden werden beispielhaft einige der Dialoge erläutert, die im Verlauf der Durchführung des Geschäftsprozesses „Lebensversicherungs-Antrag“ angezeigt werden.

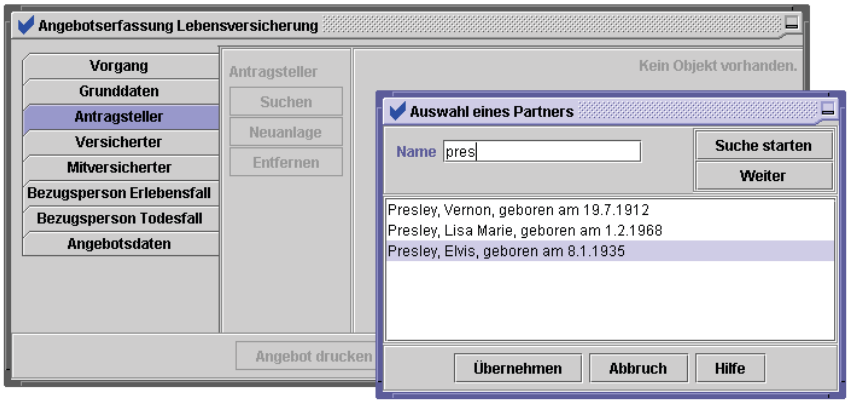

Abb. 35: Partner-Suchmaske in der Angebotserfassung

Der Einsatz einer Suchmaske für Partner bei der Antragserfassung ist in Abbildung 35 dargestellt. Hier wurde innerhalb des Reiters „Antragsteller“ auf „Suchen“ geklickt, um einen bestehenden Partner als Antragsteller in dieses Angebot einzutragen. Suchmasken sind generell gleich aufgebaut, egal ob wie hier Partner, Verträge oder sonstige Objekte gesucht werden. Der obere Bereich der Suchmaske ist auf die durchsuchte Klasse von Objekten anpaßbar, indem Eingabefelder für die suchbaren Attribute angezeigt werden. Durch „Suche starten“ wird der Daten-Manager mit diesen Kriterien nach Objekten der jeweiligen Klasse suchen. Im mittleren Bereich werden dann die gefundenen Treffer angezeigt, wovon ein Objekt ausgewählt werden kann und mit „Übernehmen“ als Ergebnis der Suche übernommen wird.

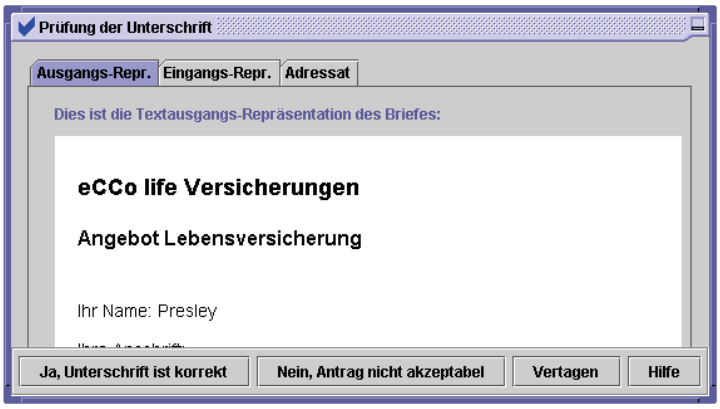

Abb. 36: Darstellung von Ausgangs- und Eingangskorrespondenz

Sobald die Papierversion eines Antrags in der Zentrale eintrifft, muß die Unterschrift von einem Sachbearbeiter geprüft werden. Dazu wird ein Vorgang gestartet, der den Dialog aus Abbildung 36 ausführt. Hier wird einerseits die als HTML gespeicherte Ausgangsrepräsentation dargestellt, die für den Kunden ausgedruckt wurde. Hinter dem Reiter “Eingangsrepräsentation” verbirgt sich ein Image der gescannten Version des unterschriebenen Antrags, in dem der Bearbeiter die Unterschrift prüfen kann.

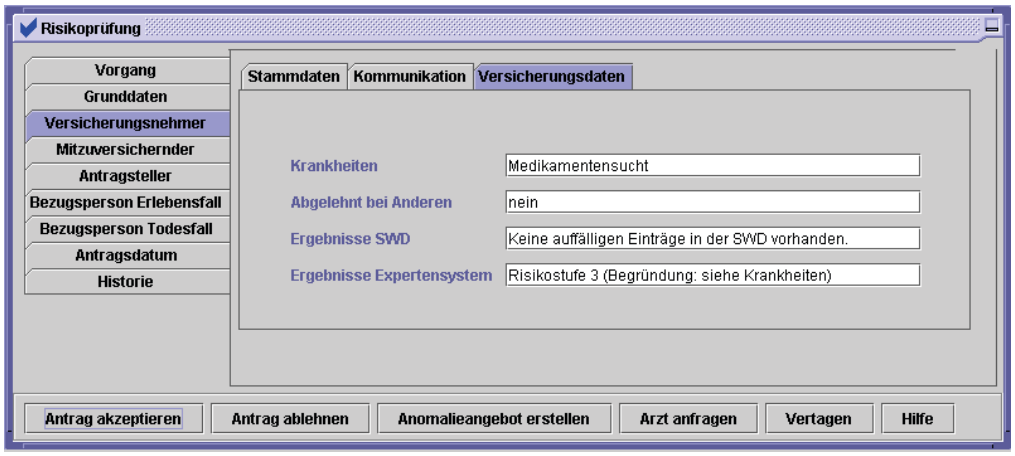

Abb. 37: Risikoprüfung mit Erweiterung der Partner-Maske zum Lebensversicherten

Abbildung 37 zeigt im rechten Teil, wie die Eingabemaske für einen natürlichen Partner mit den Reitern „Stammdaten“ und „Kommunikation“ um den Reiter „Versicherungsdaten“ zur Eingabemaske für einen „Lebensversicherten“ erweitert wurde. Die Anpassung der Masken für verwandte Objekte ist also sehr einfach.

Genauso einfach ist das Zusammensetzen größerer Masken aus mehreren kleinen. So wird die Lebensversicherten-Maske hier als Teil des Dialogs „Risikoprüfung“ eingesetzt, die außerdem einige weitere Reiter zur Darstellung eines Antrags und zusätzlicher Daten enthält, anhand derer der Sachbearbeiter die Entscheidung in der Risikoprüfung treffen kann. Die getroffene Entscheidung wird dem System durch einen der Knöpfe „Antrag akzeptieren“, „Antrag ablehnen“, „Anomalieangebot erstellen“ und „Arzt anfragen“ mitgeteilt.

Zusätzlich gibt es hier mit „Vertagen“ die Möglichkeit, den Vorgang auf einen späteren Zeitpunkt zu vertagen, zu dem er dem Sachbearbeiter automatisch wieder vorgelegt wird. Im System VAA++ kann grundsätzlich jeder Vorgang vertagt werden. Jedoch wurde der entsprechende Knopf nur in Dialogen derjenigen Vorgänge vorgesehen, bei denen eine Vertagung sinnvoll ist.

In Kapitel 4 wurde erläutert, wie der Geschäftsprozeß Lebensversicherungs-Antrag in das System VAA++ integriert wurde. Die dazu erstellten Workflows, Vorgänge und Dialoge wurden im Anpassungspaket der Steuerungsebene abgelegt, während in der Anwendungsebene die benötigten Anwendungsbausteine implementiert wurden, unter anderem der beschriebene Baustein „Leben“. Die Anwendungsobjekte werden vom Daten-Manager persistent gemacht und sind somit von dieser Aufgabe befreit.

Der zweite von der Projektgruppe ***eCCo*** implementierte Geschäftsprozeß „Kraftfahrzeug-Schaden“ wurde analog zu diesem mit dem System VAA++ verwirklicht und wird im nun folgenden Kapitel erläutert.

5 Der Geschäftsprozeß Kraftfahrzeug-Schaden

Eine weitere Aufgabe der Projektgruppe ***eCCo*** war es, die Schadensbearbeitung einer Kfz-Versicherung elektronisch zu unterstützen. Bei der Entwicklung des entsprechenden Anwendungssystems stand eine Anlehnung an die VAA im Vordergrund, wobei zur Ermöglichung einer praxisnahen Realisierung analog zur Unterstützung des Geschäftsprozesses „Lebensversicherungs-Antrag“ (s. Kap. 4) die selbstentwickelte Architektur VAA++ (s. Kap. 2, 3) eingesetzt wurde. Dabei wurden die gleichen Anwendungsbausteine aus der Anwendungsebene, die in Kapitel 3 erläutert worden ist, verwendet. Insbesondere handelte es sich um die Anwendungsbausteine Partner und Kommunikation. Desweiteren sind aus der VAA++ Architektur der Workflow-Manager, der Vorgangs-Manager, der Daten-Manager und der Dialog-Manager benutzt worden (s. Kap. 2, 3), um die Realisierung des Geschäftsprozesses „Kraftfahrzeug-Schaden“ in der gleichen Umgebung und Infrastruktur wie der des Geschäftsprozesses „Lebensversicherungs-Antrag“ zu ermöglichen.

5.1 Workflow Kfz-Schadenbearbeitung

Im folgenden wird der Workflow für den Ablauf einer Schaden- und Leistungsbearbeitung vorgestellt. Dieser ist vollständig implementiert und verwendet den in Kapitel 3 beschriebenen Workflow-Manager. Der Workflow besteht aus verschiedenen Vorgängen, im einzelnen sind dieses Schadenfallaufnahme, Aufnahme des Unfallberichts, Deckung- und Haftungsprüfung, Kulanzentscheidung, Zuordnung des Schadenereignisses, Schadenberechnung, Abrechnung mit anderen Versicherungsunternehmen und einer abschließenden Mitteilung. Der Ablauf ist in Abbildung 38 zu sehen und die einzelnen Vorgänge werden anschließend näher erläutert.

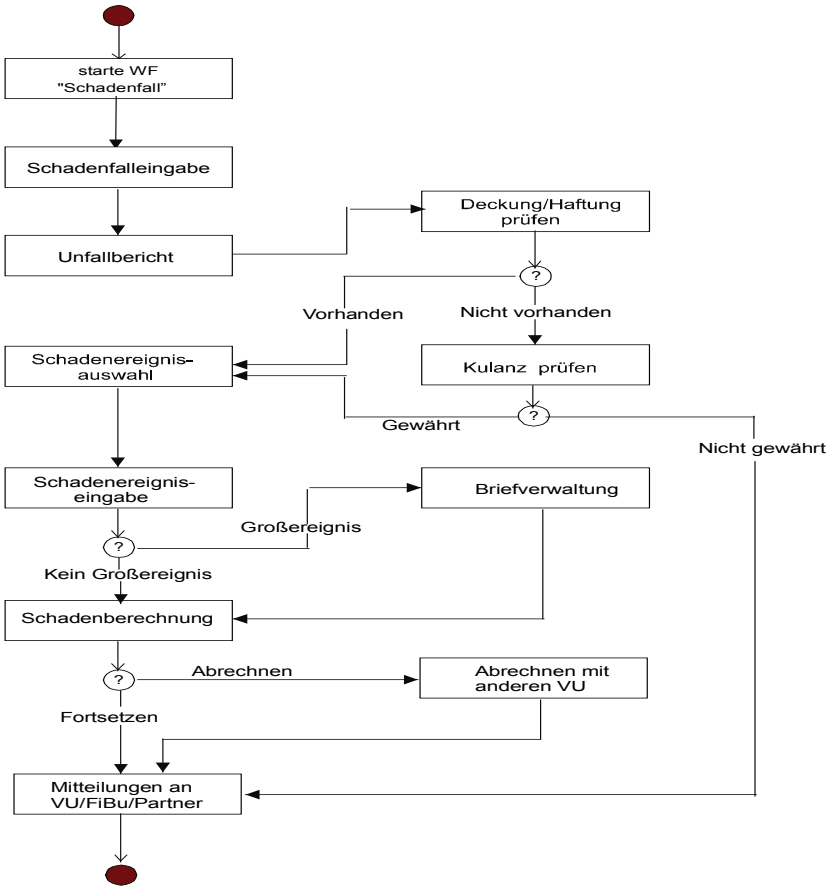

Abb. 38: Workflow zur Schadenfallaufnahme und Schadenbearbeitung

Der Workflow beginnt zunächst mit dem Aufruf der Schadenfalleingabe-Maske (Abb. 39). In dieser werden die zum Schadenfall erforderlichen Daten eingegeben, die auf Plausibilität und Vollständigkeit überprüft werden. Desweiteren wird hier eine eindeutige Schadenfallnummer automatisch vergeben. Die Eingaben werden nach Muß- und Kann-Feldern unterschieden. Weiterhin ist es möglich, einen vorhandenen Schadenfall anhand der Schadenfallnummer aus der Datenbank zu laden und zu bearbeiten.

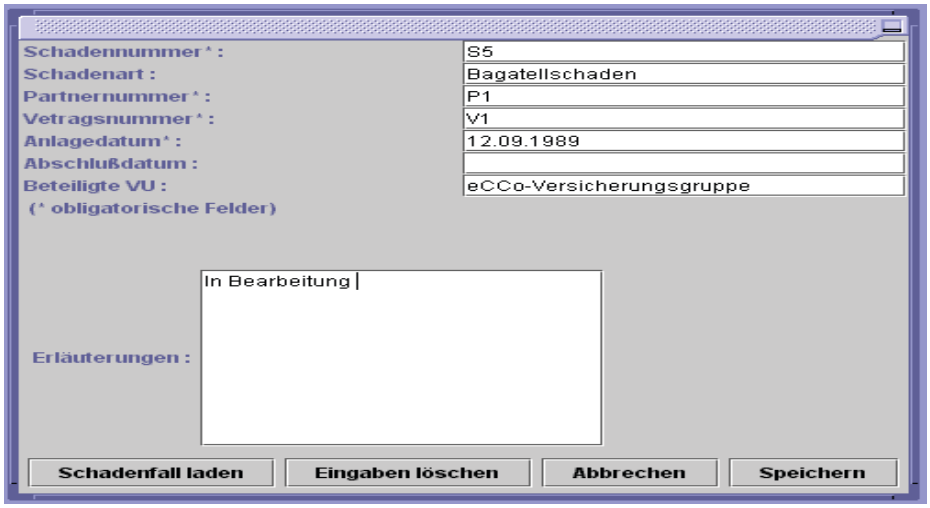

Abb. 39: Screenshot zur Maske „Schadenfalleingabe“

Nach der Schadenfalleingabe wird der Unfallbericht aufgenommen (Abb. 40), dem automatisch eine Nummer zugewiesen wird. Anhand der Polizeiberichtsdaten wird entschieden, ob fahrlässig gehandelt worden ist oder nicht, d.h. ob im vorliegenden Fall eine Haftung gegeben ist. Zusätzlich kann vom Sachbearbeiter ein Kommentar abgegeben werden.

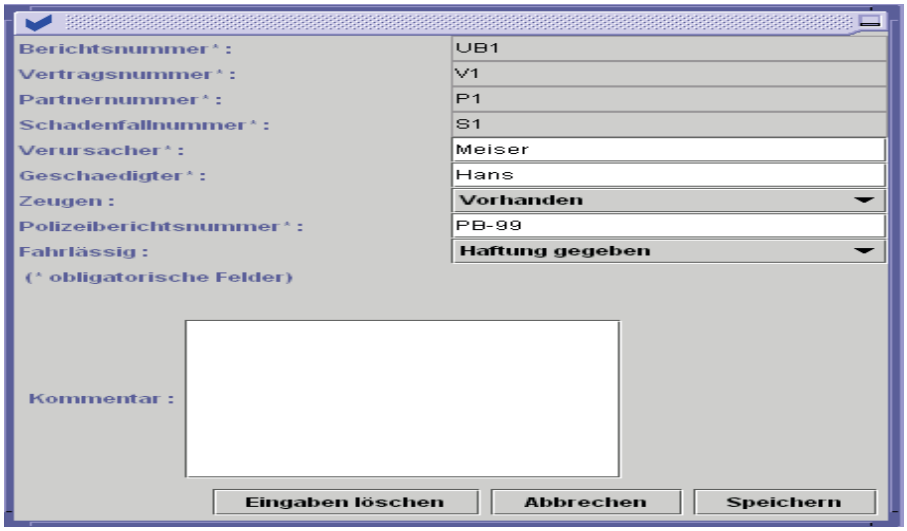

Abb. 40: Screenshot zur Maske „Unfallbericht“

Im folgenden Schritt wird anhand des zuvor eingegebenen Unfallberichtes festgestellt, ob Haftung gegeben ist. Anschließend werden die zum Schadenfall zugehörigen Vertragsdaten geladen und auf Gültigkeit des Vertrages untersucht. Mit Hilfe dieser Informationen in der Maske kann somit entschieden werden, ob eine ausreichende Deckung vorhanden ist (Abb.41). Im positiven Fall, d.h. Haftung und Deckung gegeben, läuft der Workflow normal weiter. Im negativen Fall muß noch überprüft werden, ob Kulanz gewährt werden kann.

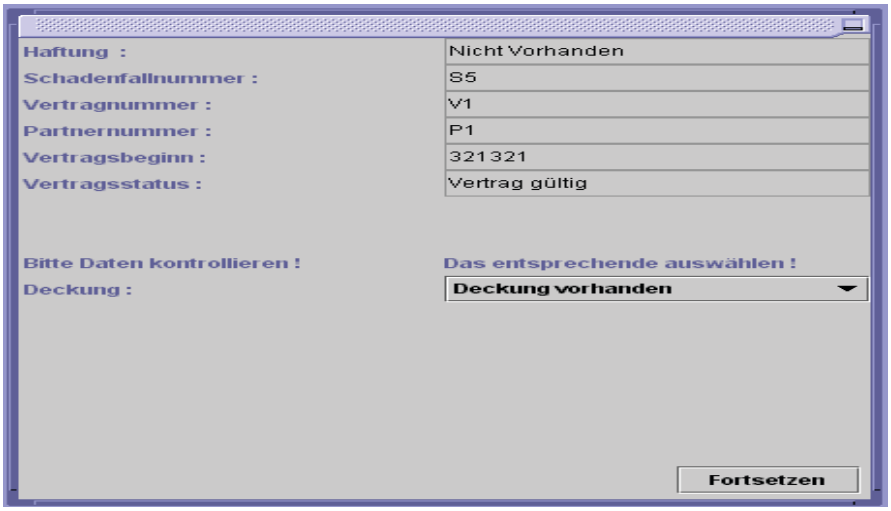

Abb. 41: Screenshot zur Maske „Deckung/Haftung“

Als nächstes muß der Schadenfall einem Schadenereignis zugeordnet werden (Abb. 42). Dazu kann mit Hilfe einer Auswahlmaske ein bereits eingegebenes Schadenereignis ausgewählt oder ein neues Ereignis eingegeben werden. Es ist möglich, sich verschiedene Schadenereignisse zu einem bestimmten Ereignisort oder Ereignisdatum über das „Suche starten“-Button ausgeben zu lassen.

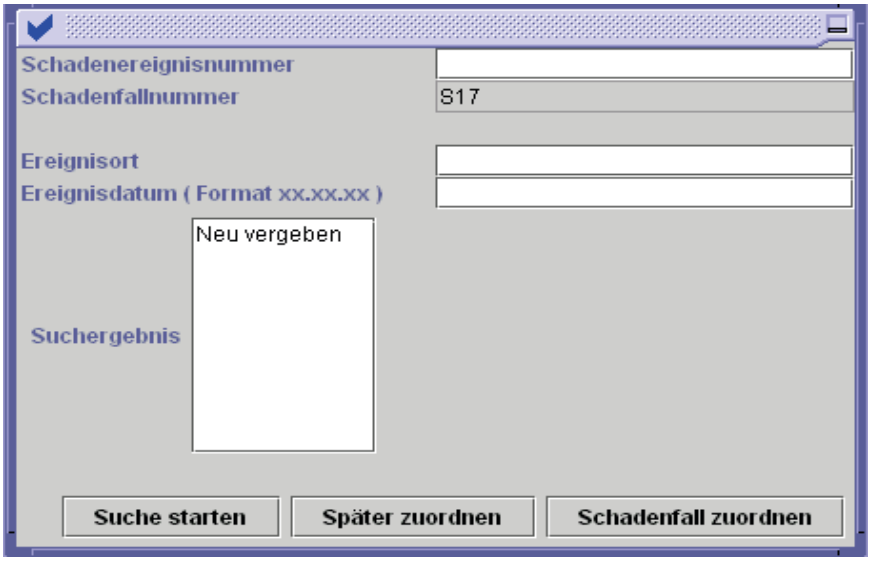

Abb. 42: Screenshot zur Maske „Schadenereignisauswahl“

Nachdem ein Ereignis ausgewählt oder neu angelegt worden ist, kann der Benutzer in einer Maske (Abb. 43) die Schadenereignisdaten eingeben bzw. ändern. In dieser Maske werden außerdem die bereits zugeordneten Schadenfälle angezeigt. Weiterhin ist es möglich, anzugeben, ob ein Großereignis vorliegt. In diesem Fall kann anschließend vom Benutzer eine entsprechende Mitteilung an die Finanzbuchhaltung erstellt werden.

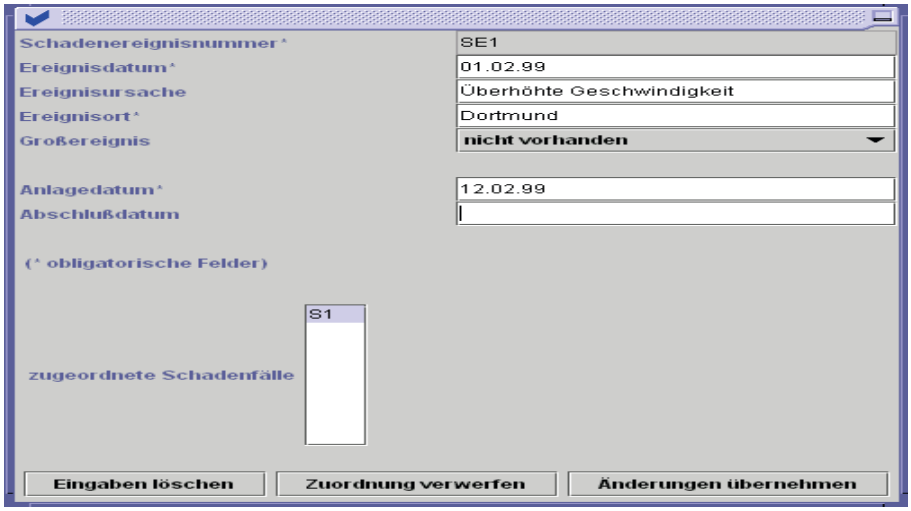

Abb. 43: Screenshot zur Maske „Schadenereigniseingabe“

In der nun folgenden Schadenberechnung (Abb. 44) werden anhand der Schadenhöhe die Entschädigungsleistung und die sonstigen Kosten anhand zuvor angelegter Tabellen ermittelt. Dem Sachbearbeiter steht auch hier ein Eingabefeld für Erläuterungen zur Verfügung.

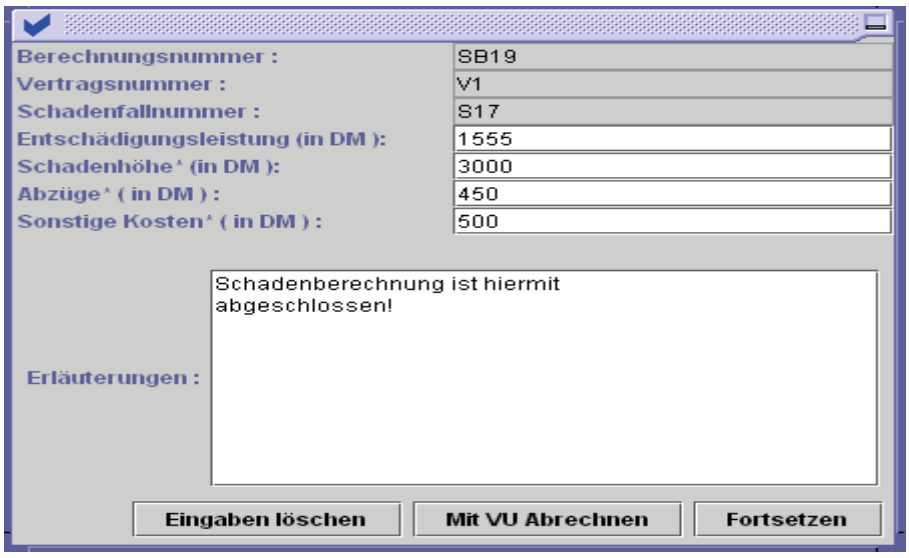

Abb. 44: Screenshot zur Maske „Schadenberechnung“

Danach kann der Sachbearbeiter auswählen, ob mit anderen Versicherungsunternehmen abgerechnet werden muß, worauf eine entsprechende Maske aufgerufen wird.

Abschließend wird die Briefverwaltung (Abb. 45) aufgerufen, mit deren Hilfe eine Mitteilung an den Versicherungsnehmer erstellt werden kann.

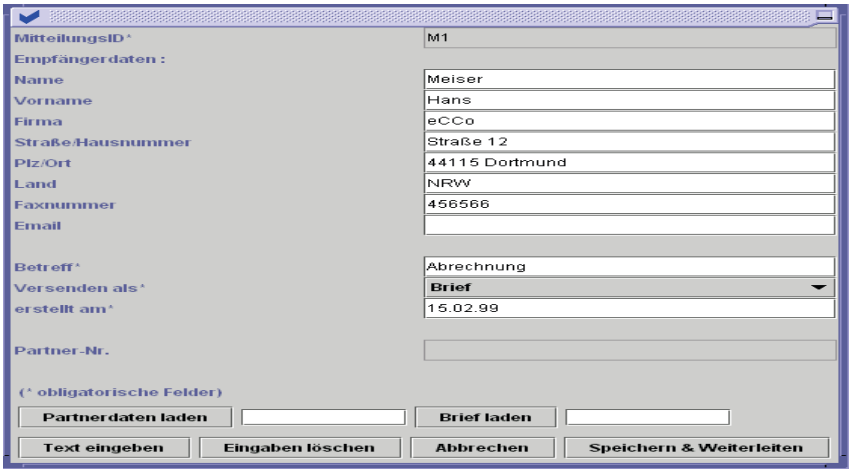

Abb. 45: Screenshot zur Maske „Briefverwaltung“

5.2 Anwendungsbaustein Schaden/Leistung

Bei der Erstellung des Anwendungssystems „Schaden/Leistung“ wurde als Grundlage der späteren Arbeitsschritte ein an der Praxis orientiertes Modell eines Geschäftsprozesses benötigt. Dieser wurde innerhalb der Projektgruppe intuitiv entwickelt, d.h. es wurde kein tatsächlich bestehender Geschäftsprozeß eines Versicherungsunternehmens übernommen.

Anhand ausgiebiger Informationen von Fachleuten aus der Versicherungswirtschaft wurden die einzelnen Vorfälle einer Kfz-/Schadenbearbeitung zusammengetragen. Basierend auf den gesammelten Daten wurde nach intensiver Entwicklungsarbeit die Gesamtstruktur des Geschäftsprozesses ausgearbeitet. Eine Übersicht des ausgearbeiteten Geschäftsprozesses ist in der Abbildung 46 in UML-Notation (im Gegensatz zur Abbildung 26) veranschaulicht.

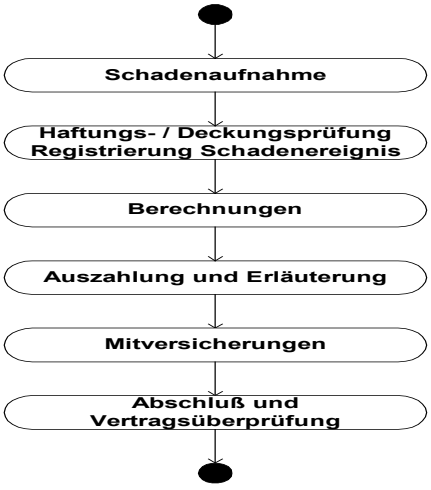

Abb. 46: Übersicht des Aktivitätsdiagramms zum Geschäftsprozeß Kfz-Schaden/Leistung (in UML)

Die Abbildung 47 in UML-Notation enthält einen Ausschnitt aus dem verfeinerten Geschäftsprozeß „Schaden/Leistung“ . Dabei handelt es sich um den Geschäftsvorfall „Berechnungen“.

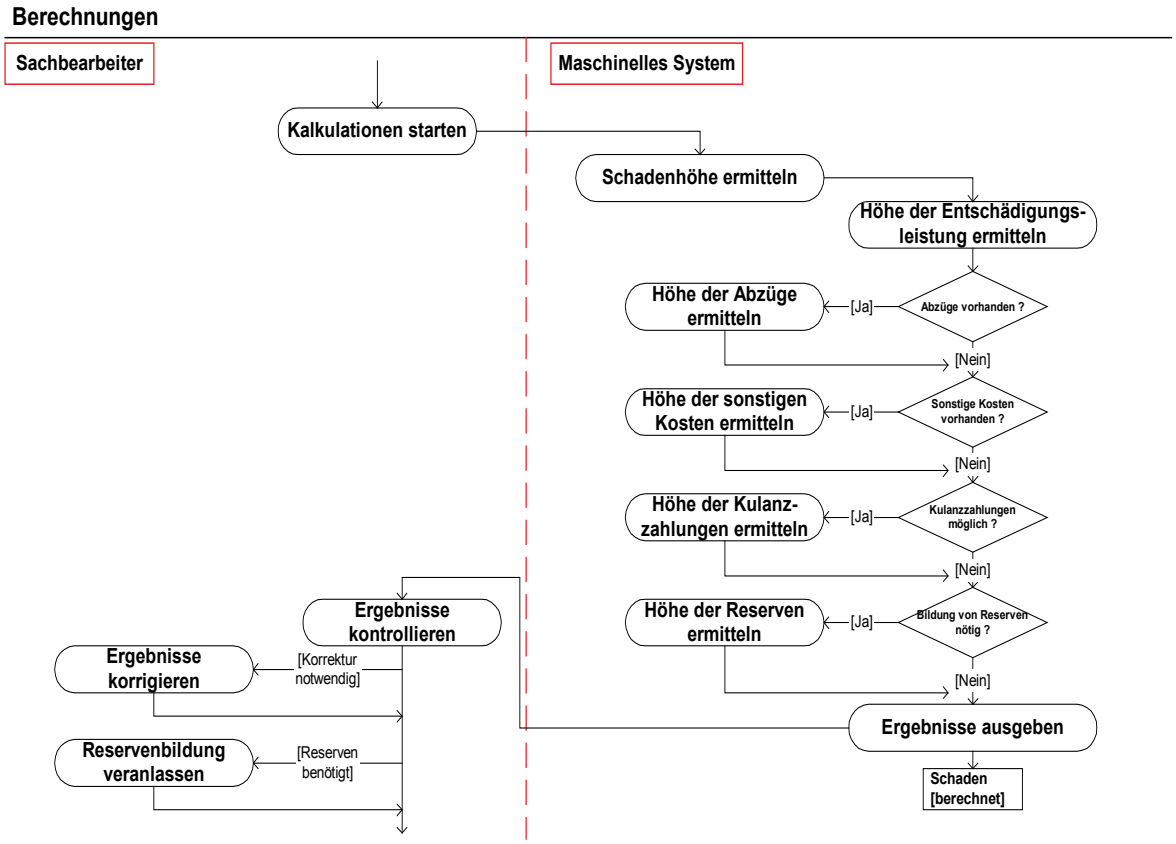

Abb. 47 : Ausschnitt aus dem Aktivitätsdiagramm „Berechnungen“ (in UML)

Die Assoziation der Klassen des Schaden-/Leistungssystems ist in dem mit Rational Rose 98 erstellten Diagramm (Abb. 48) veranschaulicht. Diese Darstellung gibt eine Übersicht über die Klassen des Schaden-/Leistungssystems und verdeutlicht das Zusammenspiel der Klassen untereinander.

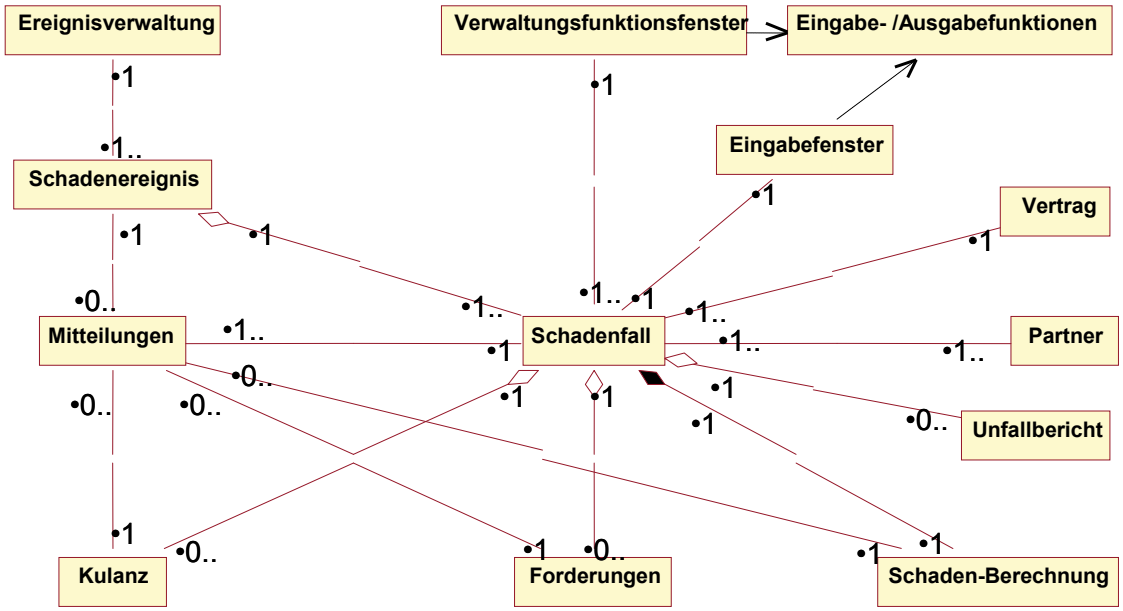

Abb. 48: Assoziation der Klassen

Im Zustandsdiagramm der Klasse „Schadenereignis“ (Abb. 49) sind die interne Struktur und die Funktionalitäten zu erkennen. Es ermöglicht einen reibungslosen Übergang zur Implementierung.

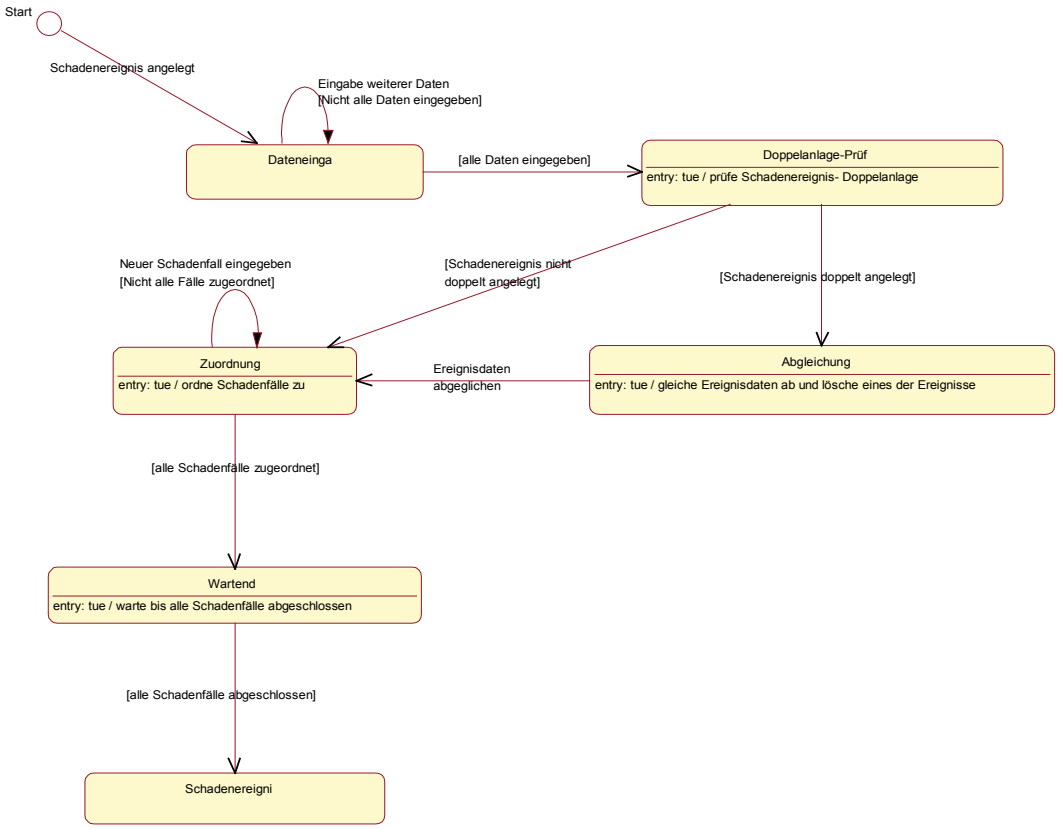

Abb. 49: Zustandsdiagramm der Klasse „Schadenereignis“

Interaktionsdiagramme dienen zur Beschreibung und Darstellung der Methodenaufrufe gegenseitig benutzender Klassen. In der Implementierungsphase können Sie als Leitfaden zur Umsetzung des objektorientierten Designs verwendet werden.

In Abbildung 50 ist das Interaktionsdiagramm „Deckung prüfen“ dargestellt:

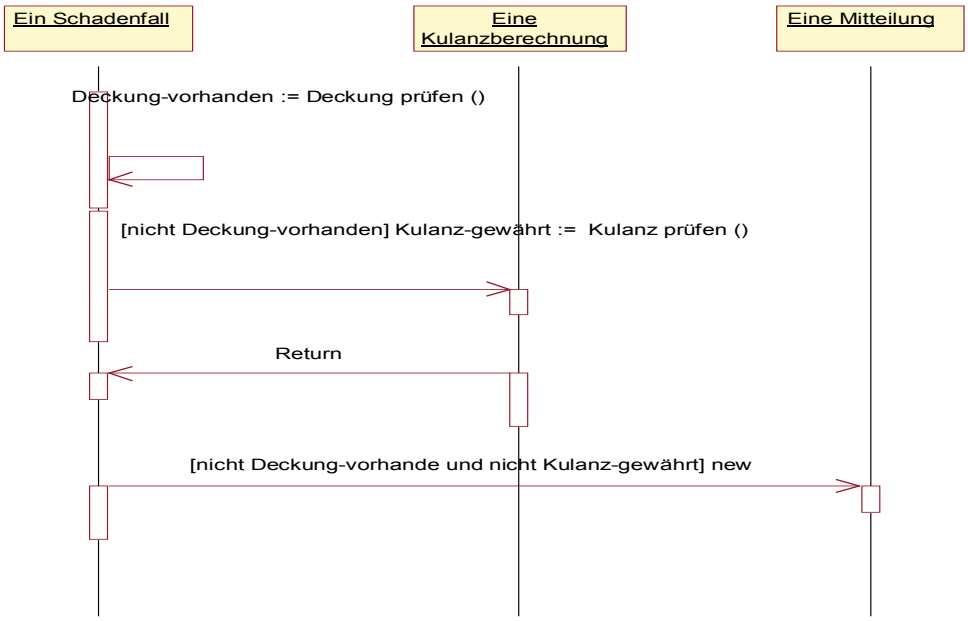

Abb. 50: Interaktionsdiagramm „Deckung prüfen“

Der folgende Kapitel 6 „Der Editor für verteilte Geschäftsprozesse“ ist ein weiteres Teilprojekt von ***eCCo***. Dieser Editor stellt ein graphisches Werkzeug dar, mit dem verteilte Geschäftsprozesse modelliert und simuliert werden können.

Die beiden vorangegangenen Geschäftsprozesse „Lebensversicherungs-Antrag“ und „Kraftfahrzeug-Schaden“ aus Kapitel 4 und 5 sind verteilte Geschäftsprozesse, die dem Editor als Grundlage zu einer beispielhaften Modellierung dienen können. Die Modellierung des verteilten Geschäftsprozesses „Lebensversicherungs-Antrag“ mit Hilfe dieses Editors wird exemplarisch in Kapitel 7 „Modellierung des Geschäftsprozesses Lebensversicherungs-Antrag“ vorgestellt.

6 Der Editor für verteilte Geschäftsprozesse

6.1 Einleitung

Als weiteres Teilprojekt wurde von der Projektgruppe ***eCCo*** ein graphisches Werkzeug entwickelt, mit dem verteilte Geschäftsprozesse modelliert und simuliert werden können. Im Gegensatz zu marktüblichen Geschäftsprozeß-Modellierungswerkzeugen wurde hierbei der Schwerpunkt auf die Modellierung derjenigen Aspekte eines Geschäftsprozesses gelegt, die für die Verteilung relevant sind [GWM96].

Damit werden nur diejenigen Teile des Prozesses modelliert, die einen Einfluß auf das Verhalten der Kommunikationskanäle haben - wobei jede Verbindung zwischen zwei geographisch entfernten Orten, über die Daten ausgetauscht werden können, als Kommunikationskanal bezeichnet wird. Mit Hilfe dieses Werkzeugs kann untersucht werden, welche Teile eines verteilten Geschäftsprozesses verändert werden sollten, um die von den Kommunikationskanälen verursachten Verzögerungen und Kosten zu optimieren.

Die Projektgruppe ***eCCo*** hat dazu eine Symbolsprache entwickelt, mit der der Benutzer effizient verteilte Geschäftsprozesse modellieren kann. Diese Symbolsprache beinhaltet unter anderem spezielle Elemente, die Verzögerungen im Geschäftsprozeß und Änderungen der transportierten Datenmenge darstellen. Auf diese Weise kann in der Simulation das Zeit- und Kostenverhalten der Kommunikationskanäle untersucht und ausgewertet werden. Ein ähnlicher Ansatz zur Modellierung verteilter Geschäftsprozesse wird bei [Sto98] verfolgt.

Die Modellierung eines verteilten Geschäftsprozesses mit der Definition der entsprechenden Orte und Kommunikationskanäle wird im folgenden Kapitel 6.2 detailliert beschrieben. Auf die Modellierung der relevanten Attribute der Kommunikationskanäle, wie zum Beispiel Übertragungsfrequenz und Kosten, wird in Kapitel 6.3 näher eingegangen. Nachdem ein Geschäftsprozeß sowie die zugehörigen Kommunikationskanäle geeignet modelliert wurden, kann daran anschließend die Simulation des verteilten Geschäftsprozesses gestartet werden. Die Simulation wird in Kapitel 6.4 genauer beschrieben. In Kapitel 6.5 werden einige ausgewählte Aspekte des Entwurfs vorgestellt, um abschließend mit Kapitel 6.6 einen Ausblick auf weitere Entwicklungsmöglichkeiten hinsichtlich des vorgestellten Editors zu geben.

6.2 Modellierung einer Geschäftsprozeßlandschaft

Dieses Kapitel beschreibt im Detail, wie Orte, Kommunikationskanäle und Geschäftsprozesse definiert werden können. Zunächst wird die Verteilungssicht beschrieben, die Orte und Kanäle enthält, danach die Geschäftsprozeßsicht. Daran anschließend wird näher auf die einzelnen Symbole eingegangen, die in der Geschäftsprozeßsicht verwendet werden können.

6.2.1 Verteilungssicht

Beim Starten des Editors öffnet sich die Verteilungssicht (Abb. 51). Hier können Orte und Kommunikationskanäle, die diese Orte verbinden, angelegt werden. Die Werkzeugleiste (engl. toolbar) enthält drei Werkzeuge: das Auswahlwerkzeug (dargestellt als Pfeil), das Ortswerkzeug, und das Kanalwerkzeug. Wenn auf das Ortswerkzeug gedrückt wird, verändert der Mauszeiger seine Form. Nun können neue Orte auf der leeren weißen Fläche, der sogenannten Leinwand, angelegt werden, indem mit der Maus auf die gewünschte Stelle geklickt wird.

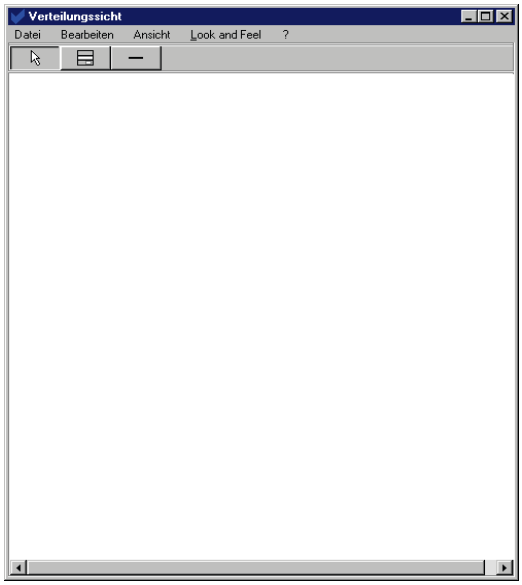

Abb. 51: Verteilungssicht

Es erscheint ein rechteckiges Objekt, das einen Ort darstellt (Abb. 52). Im unteren Bereich kann nun der Name des Ortes eintragen werden (z.B. "Hauptverwaltung") und die Anzahl der Instanzen dieses Ortes. Hier sollte eine größere Zahl als eins eintragen werden, wenn in diesem Ortsobjekt mehrere nicht zu unterscheidende Orte zusammengefaßt werden sollen, also z.B. 50 Geschäftstellen, auf denen jeweils derselbe Geschäftsprozeß abläuft und die jeweils dieselbe Kommunikationsinfrastruktur haben. In diesem Fall würde der Benutzer als Namen "Geschäftsstelle" und als Anzahl "50" eintragen.

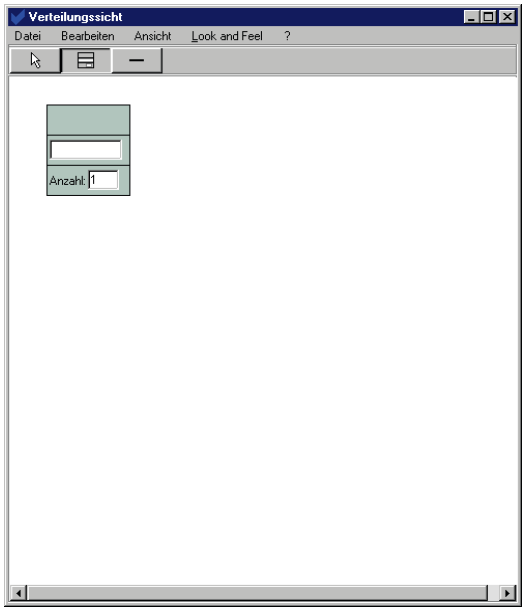

Abb.52: Verteilungssicht mit einem Ort

Sobald ein weiterer Ort angelegt worden ist, kann der erste Kommunikationskanal definiert werden. Dazu drückt man auf das Kanalwerkzeug, um anschließend mit der Maus eine Linie von einem Ort zu einem anderen ziehen. Diese Linie stellt einen Kommunikationskanal dar. Wenn ein Knick in einen Kommunikationskanal eingefügt werden soll, kann mit Hilfe des Mauszeigers mit aktiviertem Auswahlwerkzeug auf einen beliebigen Punkt des Kanals gedrückt werden, um ihn zur gewünschten Stelle zu ziehen, an der der Mausknopf losgelassen wird (Abb. 53).

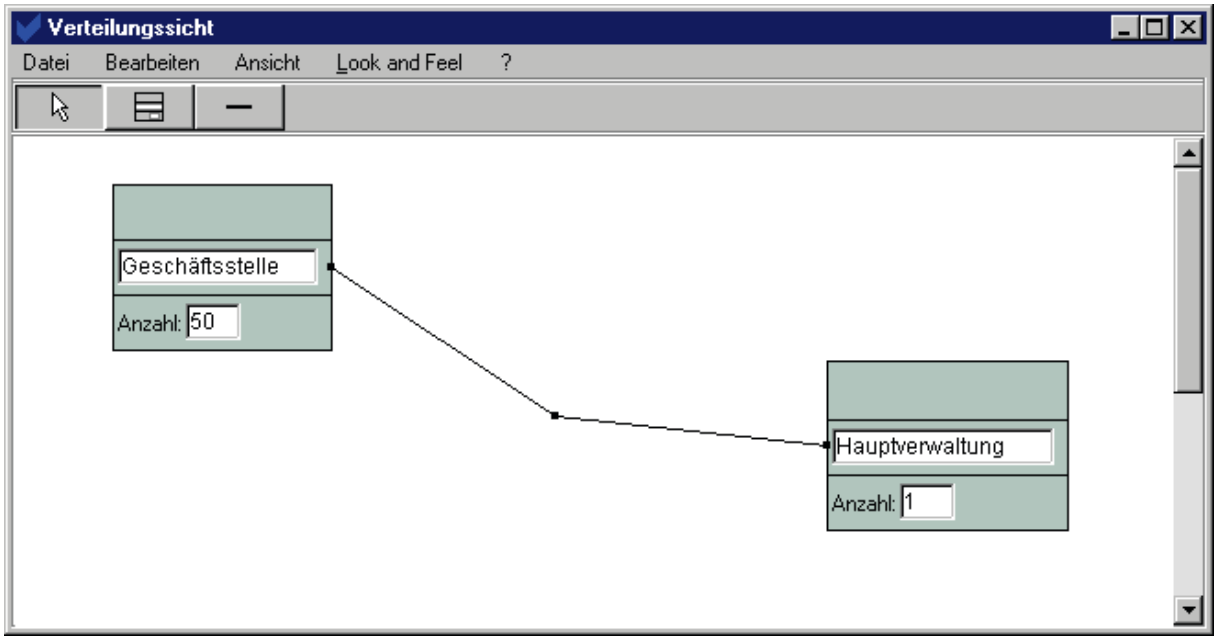

Abb. 53: Verteilungssicht mit zwei Orten und einem Kommunikationskanal

Der Benutzer kann beliebig viele Orte und Kanäle definieren; es kann durchaus auch mehrere Kommunikationskanäle geben, die vom selben Ort ausgehen. Kommunikationskanäle sind grundsätzlich bidirektional - es macht also keinen Unterschied, von welchem der beiden zu verbindenden Orte gestartet wird. Wenn man einen der Orte an eine andere Position bewegen möchte, kann man das Auswahlwerkzeug aktivieren und ihn einfach an den gewünschten Ort ziehen.

Sollte der Benutzer einen Fehler gemacht haben, kann er die “Rückgängig”-Funktion benutzen. Sie befindet sich im Balkenmenü unter “Bearbeiten” (Abb. 54).

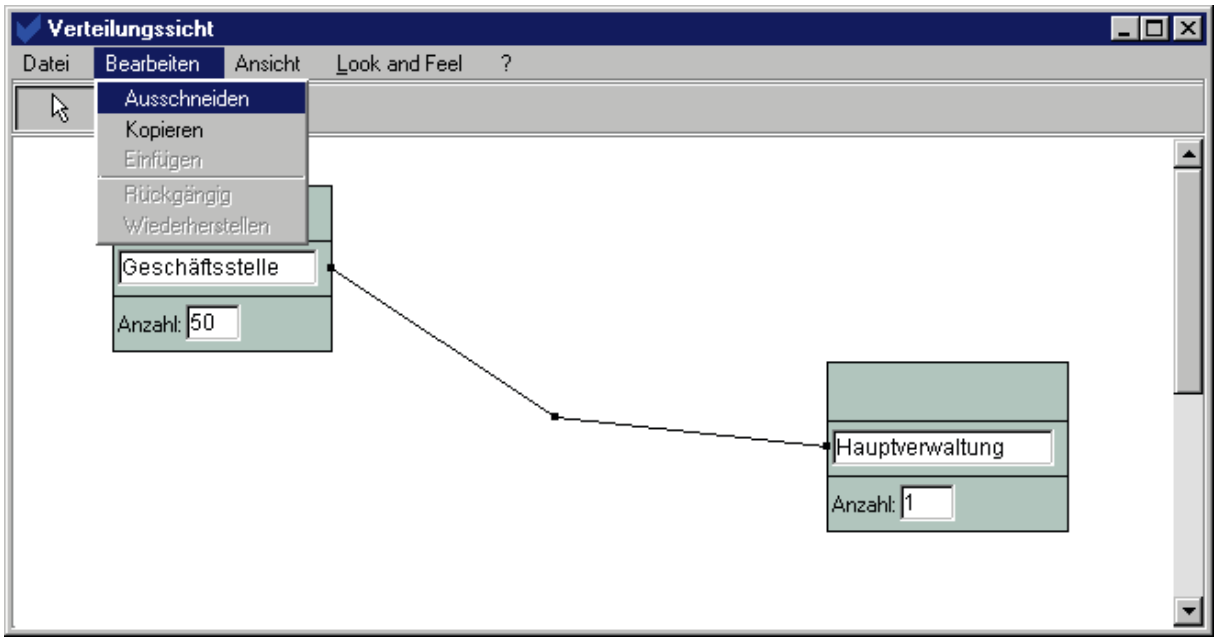

Abb. 54: Verteilungssicht mit dem “Bearbeiten”-Balkenmenü

Falls einer der Orte oder einer der Kanäle wieder gelöscht werden soll, kann man das mit dem Löschen-Befehl erledigen, der sowohl im Balkenmenü unter "Bearbeiten" als auch im Kontextmenü zu finden ist. Um den Löschen-Befehl aus dem Balkenmenü zu benutzen, muß man zuerst das zu löschende Objekt selektieren und dann auf "Löschen" drücken. Ein Objekt kann selektiert werden, indem einmal mit dem Auswahlwerkzeug darauf geklickt wird. Das Kontextmenü öffnet sich, wenn auf dem jeweiligen Objekt die entsprechende Maustaste gedrückt wird (unter Microsoft Windows: die rechte Maustaste) (Abb. 55). Auch hier ist der Löschen-Befehl zu finden.

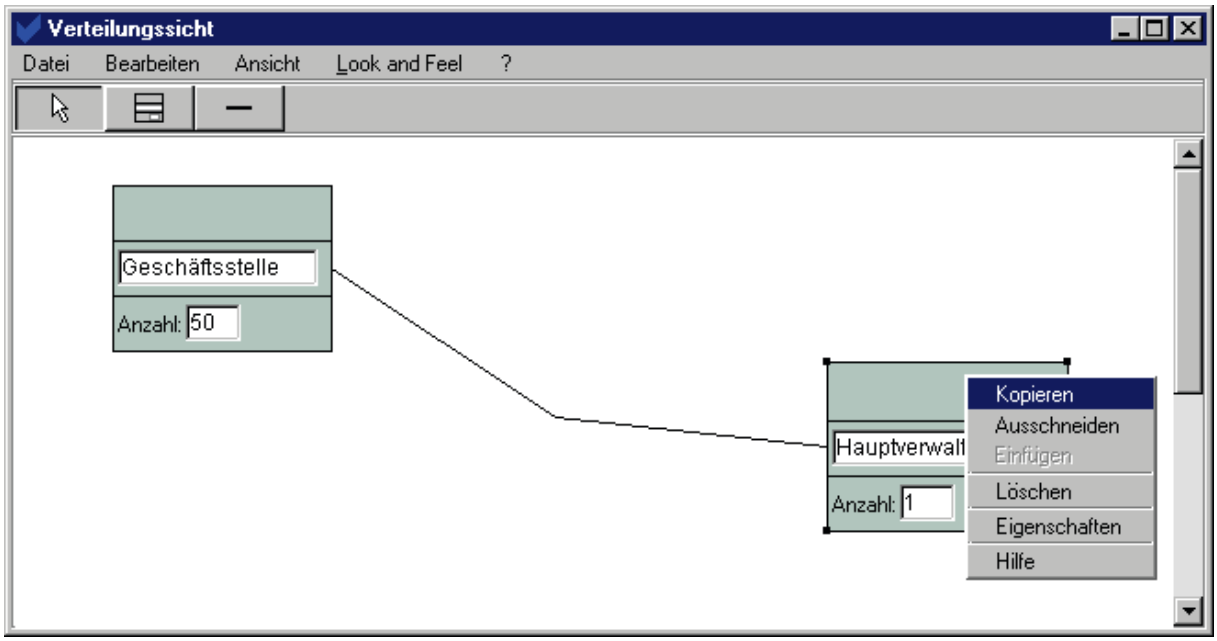

Abb. 55: Verteilungssicht mit zwei Orten und geöffnetem Kontextmenü

Wenn die Modellierung der Orte und Kanäle abgeschlossen ist, sollte die entstandene Verteilungslandschaft abgespeichert werden. Den zugehörigen Befehl findet man im Balkenmenü unter Datei/Speichern.

6.2.2 Geschäftsprozeßsicht

Nach abgeschlossener Modellierung der Verteilungslandschaft in der Verteilungssicht kann man in der Geschäftsprozeßsicht die Geschäftsprozesse beschreiben, die in den Orten und Kommunikationskanälen ablaufen. Im Balkenmenü befindet sich unter "Ansicht" der Befehl "Geschäftsprozeßsicht". Wenn man darauf drückt, öffnet sich ein neues Fenster. Hier sieht man dieselben Orte und Kanäle, die in der Verteilungssicht definiert wurden, allerdings in leicht abgewandelter Darstellung. Auch die Werkzeugleiste hat sich hier geändert: Neben dem bekannten Auswahlwerkzeug kann man jetzt das Pfeilwerkzeug und zehn Werkzeuge zur Erzeugung von Geschäftsprozeßelementen sehen.

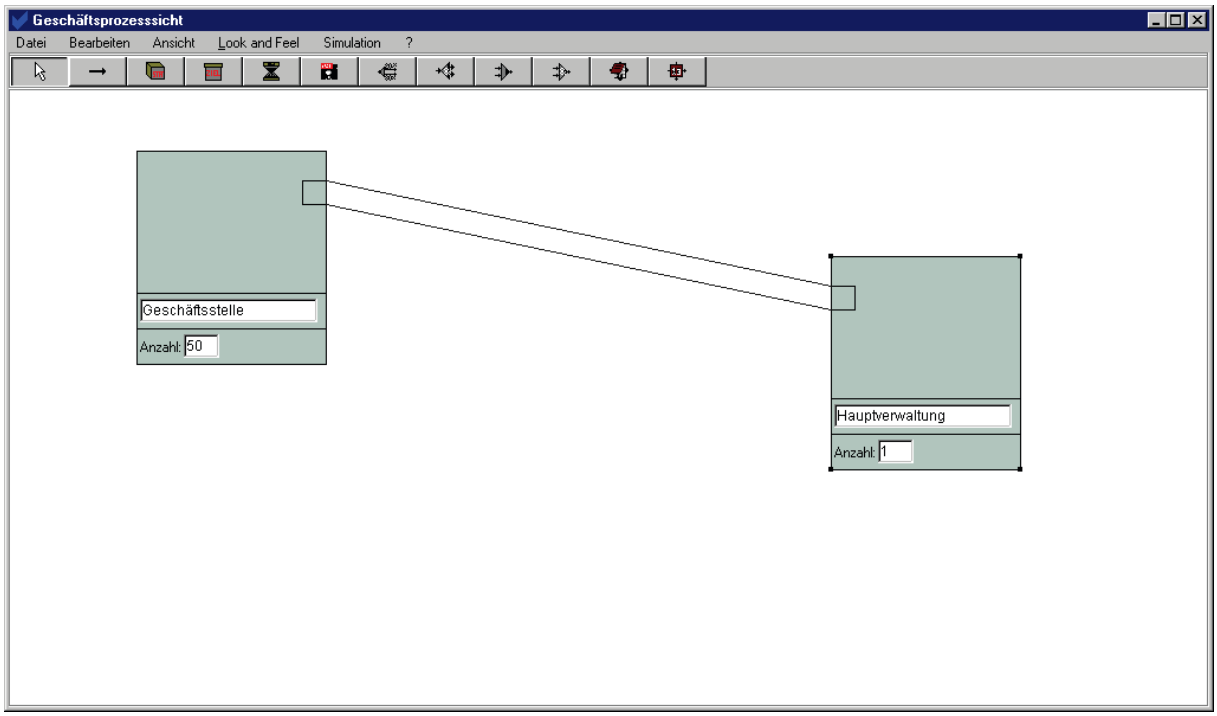

Abb. 56: Geschäftsprozeßsicht mit zwei Orten und einem Kommunikationskanal

Bei einer Verteilungslandschaft bestehend aus zwei Orten und einem Kommunikationskanal (Abb. 56) kann man nun einen kleinen Geschäftsprozeß definieren. Die dazu notwendige Vorgehensweise sieht wie folgt aus:

- Man drückt auf das Startwerkzeug (vgl. Kap. 6.2.3 für Abbildungen und Beschreibungen der einzelnen Geschäftsprozeßelemente) und erzeugt im linken Ort ein neues Startelement. Geschäftsprozeßelemente können wie gewohnt mit der Maus verschoben und gelöscht werden.
- Auf die gleiche Weise erzeugt man im rechten Ort ein neues Endeelement.

- Man schaltet jetzt um auf das Pfeilwerkzeug, um die beiden Elemente zu verbinden.
- Man zieht einen Pfeil vom Startelement zu demjenigen Ende des Kommunikationskanals, das sich in demselben Ort befindet. Es wird automatisch ein weiterer Pfeil gezogen, der sich in dem Kanal befindet und von links nach rechts geht. Dieser Pfeil stellt einen Service dar, das heißt eine Kante eines Geschäftsprozesses, die sich in einem Kanal befindet.
- Man zieht einen Pfeil von dem schwarzen Quadrat am rechten Ende des letzten Pfeils zum Endeelement.

Die Durchführung der beschriebenen Schritte liefert als Ergebnis den in Abbildung 57 dargestellten Geschäftsprozeß.

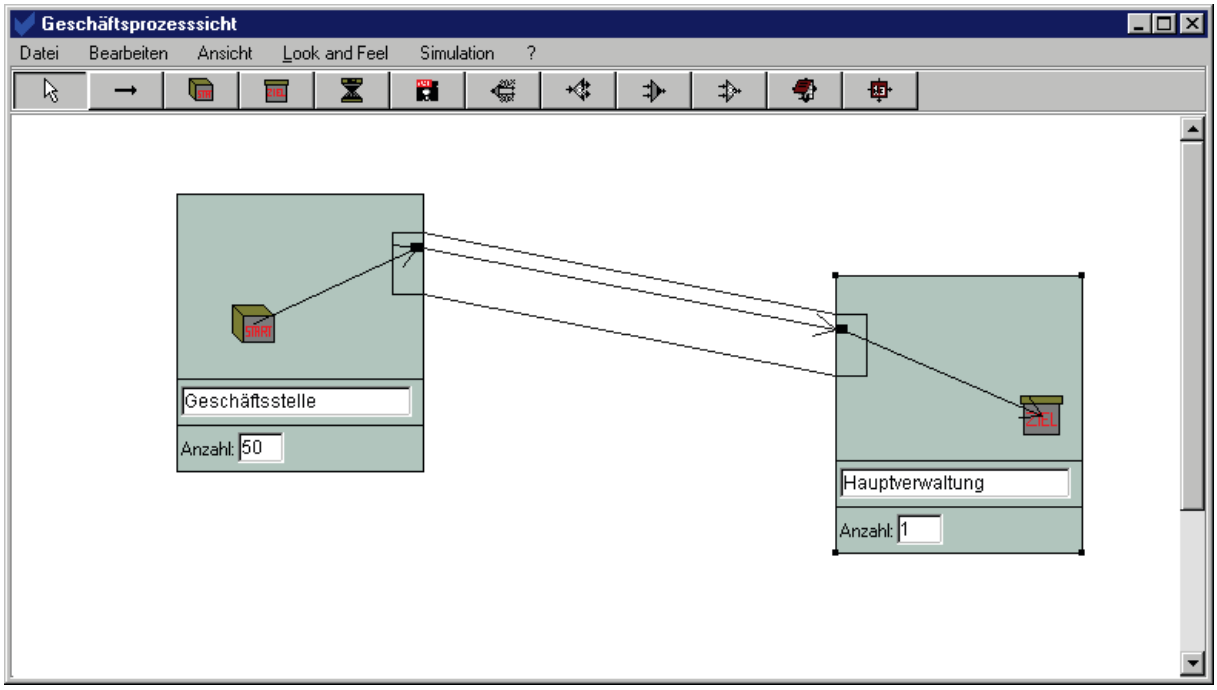

Abb. 57: Geschäftsprozeßsicht mit zwei Orten und einem definierten Geschäftsprozeß

Allgemein ist zu berücksichtigen ist, daß jedes Element eine festgelegte Anzahl Eingänge und Ausgänge hat, die jeweils nur entsprechend ihrer Richtung angeschlossen werden können. Durch Doppelklick auf ein Element öffnet sich, sofern vorhanden, ein Dialogfenster, in dem die Parameter dieses Elements angegeben werden können.

Wenn in einem Ort nicht genug Platz ist, kann der Benutzer ihn vergrößern, indem er mit dem Auswahlwerkzeug den Ort selektiert und die an den Ecken des Rechtecks, das den Ort darstellt, erscheinenden kleinen schwarzen Quadrate (vgl. rechter Ort in Abb. 57) an die gewünschte Stelle zieht.

Im folgenden Unterkapitel werden die einzelnen zur Verfügung stehenden Geschäftsprozeßelemente vollständig aufgeführt und erläutert.

6.2.3 Geschäftsprozeßelemente

1. Start

Startet einen Geschäftsprozeß, indem es neue Token erzeugt. Jedes Token stellt eine bestimmte Datenmenge dar (z.B. 50 Kilobyte), die zu einem bestimmten Zeitpunkt im Geschäftsprozeß verarbeitet wird.

- Anschlüsse: 1 Ausgang
- Parameter: Frequenz, z.B.: 12h (d.h. alle 12h - in simulierter Zeit - wird ein neues Token erzeugt)

2. Ende

Beendet einen Geschäftsprozeß. Jedes hier ankommende Token wird in der Gesamtauswertung der Simulation berücksichtigt.

- Anschlüsse: 1 Eingang
- Parameter: keine

3. Verzögerung

Fügt eine zeitliche Verzögerung in den Geschäftsprozeß ein; d.h. jedes ankommende Token muß eine gewisse Zeit warten, bevor es weitergeschickt wird. Nützlich u. a. zur Simulation einer Datenverarbeitung außerhalb des Rechners, z.B. von Sachbearbeitern.

- Anschlüsse: 1 Eingang, 1 Ausgang
- Parameter: Typ; entweder konstant, gaußverteilt oder uniform. Bei konstant zusätzlich: die konstante Wartezeit, bei gaußverteilt zusätzlich: Scheitelpunkt und Standardabweichung der Gaußkurve, bei uniform zusätzlich: minimale und maximale Wartezeit

4. Änderung der Datenmenge

Ändert die Datenmenge, die ein Token repräsentiert.

- Anschlüsse: 1 Eingang, 1 Ausgang
- Parameter: die neue Datenmenge (z.B. 100 kB)

5. Alternative Verzweigung

Verzweigt den Geschäftsprozeß, indem jedes Token in genau eine der beiden möglichen Richtungen weitergeleitet wird.

- Anschlüsse: 1 Eingang, 2 Ausgänge
- Parameter: Wahrscheinlichkeit, mit der der obere Ausgang gewählt wird (z.B. 60%). Die Wahrscheinlichkeit des unteren Ausgangs ergibt sich als die Differenz zu 100%.

6. Parallele Verzweigung

Verzweigt den Geschäftsprozeß, indem jedes Token in zwei Token aufgespalten wird. Die beiden Kinder-Token sind Brüder und „erkennen" sich bei Bedarf wieder (siehe z.B. Und-Zusammenfluß).

- Anschlüsse: 1 Eingang, 2 Ausgänge
- Parameter: keine

7. Oder-Zusammenfluß

Führt zwei Geschäftsprozeßzweige zusammen. Jedes ankommende Token wird unverändert weitergeleitet.

- Anschlüsse: 2 Eingänge, 1 Ausgang
- Parameter: keine

8. Und-Zusammenfluß

Führt zwei Geschäftsprozeßzweige zusammen. Jedes ankommende Token muß auf einen Bruder (siehe parallele Verzweigung) warten. Sobald sich zwei Brüder getroffen haben, werden sie wieder zu einem Token vereinigt und anschließend weitergeleitet.

- Anschlüsse: 2 Eingänge, 1 Ausgang
- Parameter: keine

9. Zeitbeobachter

Speichert ein Token, wartet, ob innerhalb einer vorgegebenen Zeit ein Bruder ankommt und reagiert abhängig davon. Es gibt eine Steuerleitung und eine Datenleitung (mit je einem Eingang und je einem Ausgang). Jedes in der Steuerleitung ankommende Token startet einen Wartevorgang. Wenn innerhalb der vorgegebenen Zeit ein Bruder in der Datenleitung ankommt, werden die beiden Brüder vereinigt und über die Datenleitung ausgegeben. Läuft die Zeit ab, ohne daß ein Bruder angekommen ist, wird das Steuer-Token zu seinem Vater gemacht und über die Steuerleitung ausgegeben. Dieses Geschäftsprozeßelement ermöglicht es also, eine bestimmte Zeit auf Ereignisse zu warten, die möglicherweise nicht eintreten. Tritt das Ereignis nicht ein, wird daraufhin eine Alternativhandlung gestartet.

- Anschlüsse: 2 Eingänge, 2 Ausgänge
- Parameter: maximale Wartezeit (z.B. 7d)

10. Anzahlbeobachter

Speichert ein Token, wartet ab, bis eine bestimmte Anzahl von Brüdern vorbeigekommen ist und reagiert dann. Wie beim Zeitbeobachter gibt es eine Steuerleitung und eine Datenleitung. Jedes in der Steuerleitung ankommende Token startet einen Wartevorgang. Wenn die gegebene Anzahl von Brüdern die Datenleitung durchlaufen hat, wird das gespeicherte Token über die Steuerleitung ausgegeben.

- Anschlüsse: 2 Eingänge, 2 Ausgänge
- Parameter: Anzahl der zu erwartenden Brüder (z.B. 2)

Außer der Verteilungslandschaft und den Geschäftsprozessen müssen noch die Attribute der Kommunikationskanäle beschrieben werden, um eine Simulation zu ermöglichen. Die Modellierung der Kommunikationskanäle wird im folgenden Kapitel dargestellt.

6.3 Modellierung der Kommunikationskanäle

6.3.1 Attribute der Kommunikationskanäle

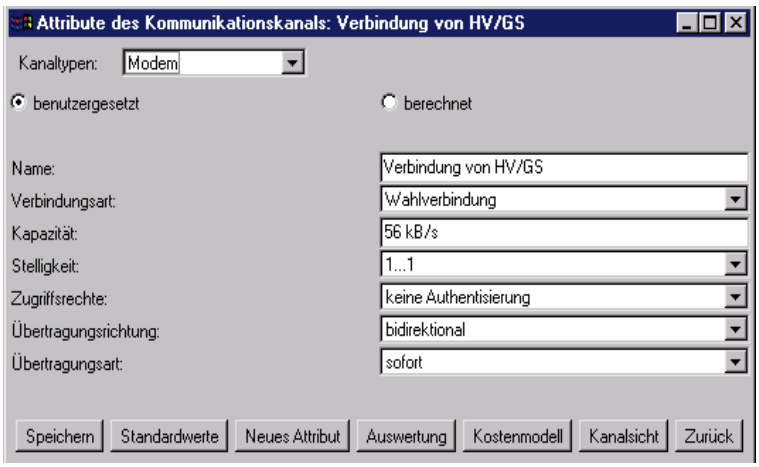

Abb. 58: Kanalattributesicht - benutzergesetzte Attribute

Kommunikationskanäle verfügen über eine Reihe von Attributen, die entweder vom Benutzer gesetzt werden können (Abb. 58) oder aus den vom Benutzer gesetzten Attributen berechnet werden. Der Benutzer hat auch die Möglichkeit, weitere Attribute hinzuzufügen. Die im Screenshot angezeigten Attribute werden vom Programm vorgegeben. Weiterhin werden die folgenden vier berechneten Attribute vorgegeben (Abb. 59):

- Durchschnittskosten
- Verweildauer
- Übertragungsdauer
- Datenvolumen

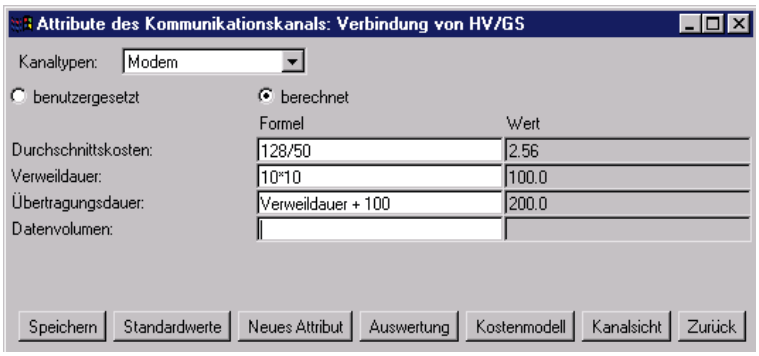

Abb. 59: Kanalattributesicht - berechnete Attribute

Auch hier kann man neue „berechnete“ Attribute hinzufügen, die, falls eine Formel angegeben wurde, automatisch berechnet werden.

Als Kommunikationskanäle werden standardmäßig folgende sechs Typen vorgegeben :

- Modem,
- ISDN,
- Standleitung,
- E-Mail,
- Brief und
- der selbstdefinierte Kanal.

Dies sind diejenigen Typen von Kommunikationskanälen, die zur Zeit am meisten verbreitet sind. Sie sollen dem Benutzer zur Orientierung dienen und haben sonst nur Einfluß auf die Standardwerte. Das bedeutet, daß der Benutzer die Attribute des Kommunikationskanals (außer dem Attribut „Name“ des Kanals) auch mit Standardwerten belegen kann.

Der selbstdefinierte Kanal ist für den Fall, daß ein Kanal definiert werden soll, der durch die anderen Kategorien nicht abgedeckt wird. Zielsetzung hierbei ist, dem Benutzer soviel Freiheit wie möglich zu geben.

6.3.2 Kostenmodell

Abb.60: Kostenmodell

Durch das Kostenmodell (Abb. 60) ergibt sich eine Auswertungsmöglichkeit für die Simulation (Kapitel 6.4). Beispielhaft wird im Programm das Kostenmodell der Telekom dargestellt. Nachdem der Benutzer zu den jeweiligen Tarifzeiten und Tarifzonen die „Preise pro Minute“ angegeben hat, kann er die Gesamtkosten einer simulierten Verbindung aus der Geschäftsprozeßsicht berechnen lassen. Zuvor muß er im Attributfenster der Kommunikationskanäle die Entfernung zwischen den Orten angeben. Weiterhin besteht auch die Möglichkeit, die Kostenmodelle anderer Anbieter zu definieren.

6.3.3 Kanalsicht

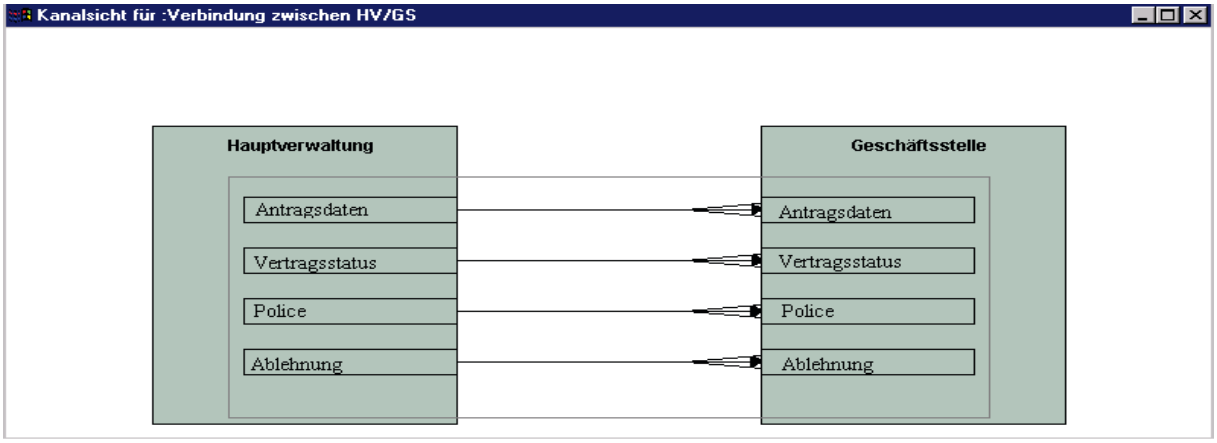

Abb. 61: Kanalsicht

In der Kanalsicht sieht man die einzelnen Services, die einen Kommunikationskanal benutzen. Ein Service ist in diesem Sinne eine Schnittstelle zwischen zwei Geschäftsprozessen, die sich an verschiedenen Orten befinden.

Die zwei großen Kästen stellen die beiden Orte dar, die durch den Kanal verbunden werden. Services werden durch Pfeile zwischen diesen Orten dargestellt.

Folgende Attribute der Services können allgemein graphisch dargestellt werden :

- Übertragungsrichtung (durch die Richtung des Pfeils)
- Verbindungsart (gestrichelte Linie: Wahlverbindung, durchgezogene Linie: Festverbindung)
- Stelligkeit

Abbildung 61 zeigt den Kanal „Hauptverwaltung/Geschäftsstelle“ aus dem Geschäftsprozeß Lebensversicherungs-Antrag (siehe Kapitel 6.2.4). Die Services, die zwischen Hauptverwaltung und Geschäftsstelle bestehen, sind: Antragsdaten, Vertragsstatus, Police, Ablehnung. Die Namen der Services können jederzeit durch den Benutzer (Doppelklick auf das jeweilige Servicerechteck) verändert werden.

In diesem Beispiel sieht man, daß es sich um eine Übertragungsrichtung von der Hauptverwaltung zur Geschäftsstelle handelt. Zwischen diesen Orten besteht eine Festverbindung und eine „1-n“-Stelligkeit.

6.4 Simulation

Bevor die Simulation gestartet wird, sollte überprüft werden, ob die Geschäftsprozesse vollständig modelliert worden sind. Insbesondere darf es keine losen Enden geben: das heißt, jeder Eingang eines Geschäftsprozeßelements muß mit einem Ausgang eines anderen Geschäftsprozeßelements verbunden sein und umgekehrt. Das bedeutet auch, daß ein Geschäftsprozeß immer mit einem Startelement beginnen und mit einem Endeelement enden muß. Wenn es doch ein loses Ende gibt, wird das zu einer Fehlermeldung bei der Simulation führen, da die Simulation dann nicht zu Ende geführt werden kann.

Im Balkenmenü befindet sich unter "Simulation" der Befehl "Simulation starten." Ein Dialogfenster öffnet sich, in dem man die Uhrzeit und den Wochentag, an dem der Geschäftsprozeß starten soll, eingeben kann (z.B. Montag 8.00 Uhr), sowie den zu simulierenden Zeitraum (z.B. 7 Tage, 12 Stunden). Außerdem muß die Zeitumsetzung angegeben werden, d.h. wieviel realen Stunden eine Sekunde in der Simulation entsprechen soll. (z.B. 2h (real) = 1s (simuliert)) (Abb. 62). Selbstverständlich können bei all diesen Angaben auch die Vorgabewerte akzeptiert werden.

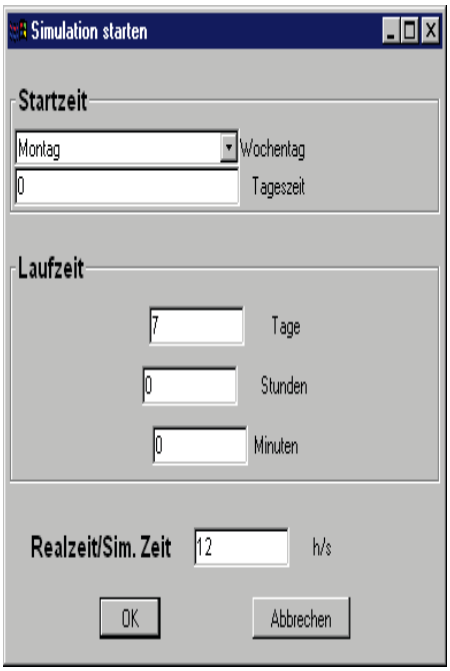

Abb. 62: Simulation starten

Wenn der Benutzer jetzt mit "OK" bestätigt, startet die Simulation. Nachdem die Simulation beendet ist, kann die Auswertung geöffnet werden. Sie befindet sich im Balkenmenü unter Simulation/Auswertung (Abb. 63). Hier werden folgende Informationen angezeigt: die angegebene Startzeit, die angegebene Laufzeit, der erreichte Datendurchsatz (sowohl als Summe als auch als Durchschnittswert), und die aufgetretenen Gesamtkosten.

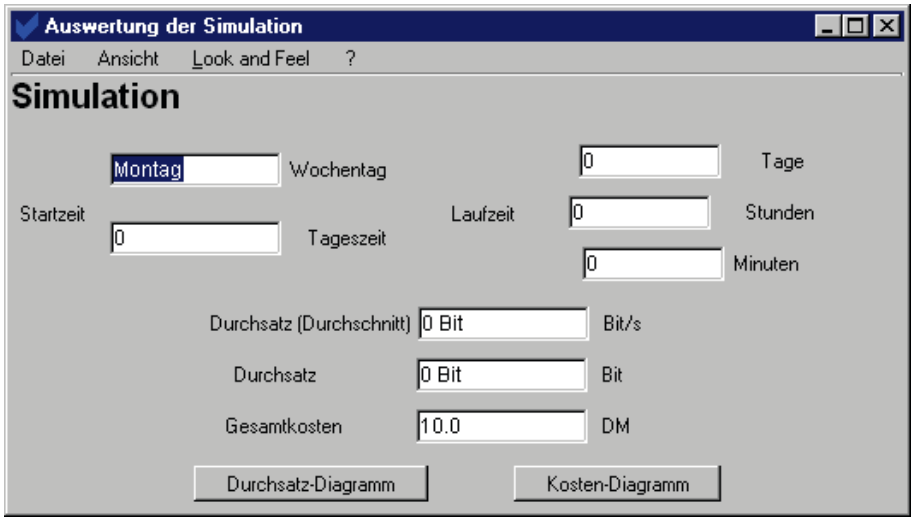

Abb. 63: Auswertung der Simulation

Bei der Berechnung des Durchsatzes werden alle definierten Endeelemente berücksichtigt; es werden alle an einem Endeelement angekommenen Token addiert. Das Feld „Gesamtkosten“ zeigt die Summe der von allen vorhandenen Kommunikationskanälen während der Simulation erzeugten Kosten an.

Der Schalter „Durchsatz-Diagramm“ öffnet ein Fenster, das die Auswertung der Simulation nach Durchsatz als Liniendiagramm zeigt. Abbildung 64 zeigt das Ergebnis einer beispielhaften Simulation, welche fünf Sekunden lief, und bei der die Geschäftsprozeßelemente „Start“, „Änderung der Datenmenge“ und „Ende“ benutzt wurden. Die Datenmenge wurde auf 5000 Bits gesetzt.

Die Y-Achse zeigt die Menge der Bits, die versendet wurden. Auf der X-Achse kann man sehen, wie lange die Simulation lief. Das Programm faßt übersichtshalber die Token in Sekunden zusammen. In Abbildung 64 würde dies bedeuten, daß in der ersten Sekunde zum Beispiel 4200 Bits den Geschäftsprozeß durchlaufen hatten.

Neben dieser Gesamtauswertung gibt es auch eine Auswertung für jeden einzelnen Kommunikationskanal. Wenn das Kanalfenster geöffnet wird, kann mit dem Schalter "Auswertung" die auf diesen Kanal beschränkte Auswertung nach Durchsatz und Kosten geöffnet werden.

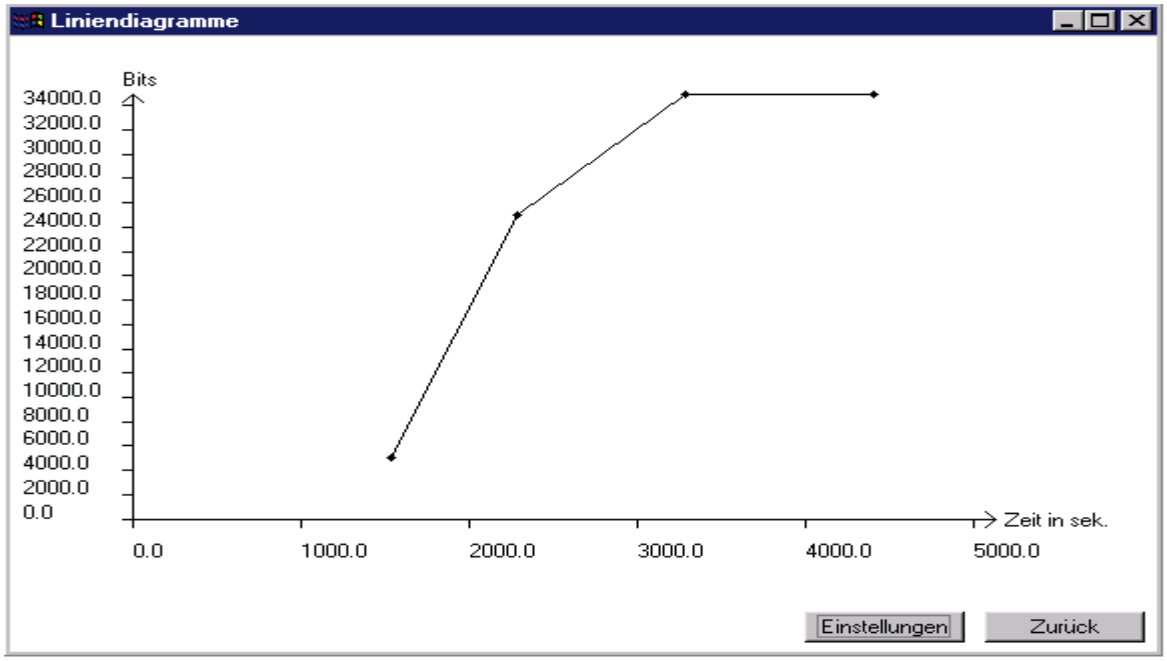

Abb. 64: Liniendiagramm

Damit ist die Beschreibung der Funktionalität des Programms abgeschlossen. Das folgende Kapitel wird sich mit einigen Details des Entwurfs beschäftigen.

6.5 Entwurf

Nachfolgend werden ausgewählte Aspekte des Entwurfs vorgestellt. Da das gesamte Programm mehr als 150 Klassen umfaßt, und daher eine umfassende Beschreibung des gesamten Entwurfs den Rahmen dieses Dokumentes sprengen würde, beschränkt sich dieses Kapitel auf diejenigen Aspekte, die möglicherweise auch für andere, ähnlich gelagerte Projekte von Interesse sein könnten:

- die Klassenbibliothek für graphische Objekte,
- die Schichtenarchitektur (“Model-View-Controller”),
- die bei der Modellierung von Anschlüssen aufgetretenen Probleme und
- den internen Aufbau der Kommunikationskanäle.

6.5.1 Graphische Klassen

Der Hauptteil des Programms beschäftigt sich mit der graphischen Darstellung der modellierten Orte, Kanäle und Geschäftsprozesse. Die hierzu benötigten Klassen wurden vollständig neu entwickelt. Basisklasse für alle graphischen Objekte ist die Klasse VisObject. VisObject und einige seiner Subklassen sind in Abbildung 65 dargestellt.

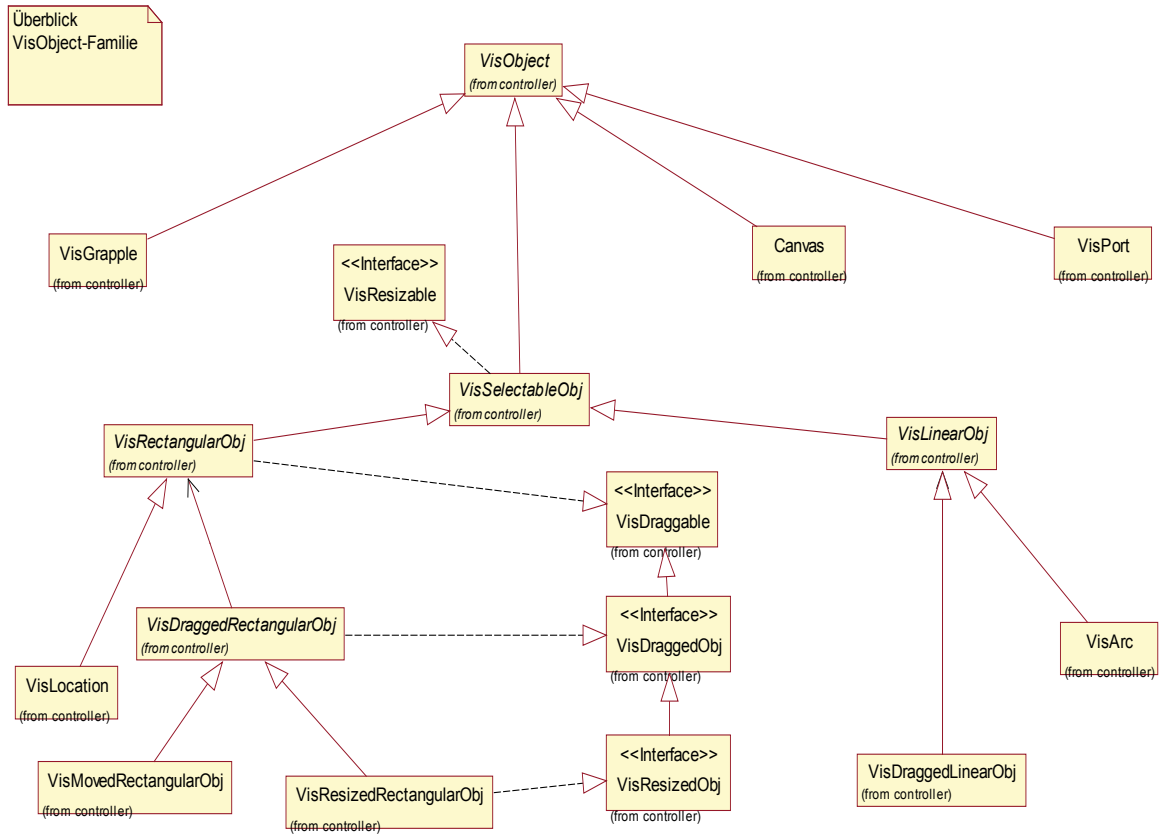

Abb. 65: VisObject und einige seiner Subklassen

Ein VisObject verfügt über folgende Attribute:
- eine Fläche auf dem Bildschirm, auf der es sich darstellen kann
- ein Vater-VisObject
- beliebig viele Kinder-VisObjects, die sich innerhalb der Fläche des VisObjects befinden müssen.

Einige interessante Subklassen von VisObject:
- Canvas. Eine leere Fläche, auf der andere Objekte vom Benutzer angelegt und verschoben werden können. RootCanvas ist eine Subklasse von Canvas.
- VisSelectableObj. Ein Objekt, das vom Benutzer selektiert, gelöscht, verschoben, kopiert usw. werden kann.
- VisRectangularObj. Eine Subklasse von VisSelectableObj, die ein rechteckiges selektierbares Objekt darstellt.
- VisLinearObj. Eine Subklasse von VisSelectableObj, die ein linienförmiges selektierbares Objekt darstellt.
- VisLocation. Eine Subklasse von VisRectangularObj, die einen Ort darstellt.
- VisArrow. Eine Subklasse von VisLinearObj, die einen Pfeil zwischen zwei Geschäftsprozeßelementen darstellt.
- VisMovedRectangularObj. Eine Subklasse von VisRectangualrObj, die ein rechteckiges Objekt darstellt, das gerade vom Benutzer verschoben wird und daher nur als Umriß zu sehen ist.

Die Wurzel in der Vater-Kind-Hierarchie von VisObjects ist ein RootCanvas, der die leere weiße Fläche, die man bei Programmstart sieht, darstellt. Alle Befehle werden mit Hilfe einer Zuständigkeitskette [GHJ95] weitergeleitet; d.h. wenn der Benutzer klickt, wird der damit verbundene Befehl zunächst an den RootCanvas gegeben. Dieser gibt den Befehl an dasjenige seiner Kinder weiter, das sich an der Stelle des Klicks befindet, das gibt es an sein entsprechendes Kind weiter usw. - bis ein VisObject entscheidet, diesen Befehl selbst zu behandeln.

6.5.2 Model-View-Controller-Architektur

Die Model-View-Controller-Architektur [GHJV95] dient dazu, die Sicht (view und controller) von der Datenhaltung (model) zu entkoppeln. Das ermöglicht es zum Beispiel, verschiedene Sichten auf dasselbe Datenobjekt zu haben, die automatisch konsistent gehalten werden. In diesem Programm wird die MVC-Architektur benutzt, um die Geschäftprozeßlandschaft, bestehend aus Orten, Kanälen und Geschäftsprozessen (das Modell), von den verschiedenen Sichten darauf (erstens in der Verteilungssicht, zweitens in der Geschäftsprozeßsicht) zu trennen.

6.5.3 Anschlüsse

Beim Entwurf der für die Modellierung von Geschäftsprozessen nötigen Funktionen trat folgende Fragestellung auf: Wie stellt man die Verknüpfung zwischen einem Geschäftsprozeßelement und einem davon ausgehenden Pfeil dar? Die naheliegende Lösung wäre eine einfache gerichtete Assoziation: das Element kennt den Pfeil, so daß es während der Simulation ausgehende Token an den Pfeil weitergeben kann. Diese Lösung hat aber einen schwer-

wiegenden Nachteil - was passiert, wenn der Benutzer den Pfeil löscht? Um den Pfeil aus der Simulation entfernen zu können, müßten jeder Pfeil und jedes Geschäftsprozeßelement wissen, wer es referenziert. Das würde die Implementierung sehr kompliziert machen. Die eingesetzte Lösung sieht so aus: Es gibt spezielle, für den Benutzer unsichtbare Objekte, die die Verbindung zwischen zwei benutzersichtbaren Objekten verkapseln. Diese Objekte heißen Anschlüsse (Ports). Jeder Anschluß gehört zu einem Pfeil oder einem Geschäftsprozeßelement, ist entweder eingehend oder ausgehend und kann mit einem Anschluß der entgegengesetzten Polarität verknüpft werden. Wenn nun ein Pfeil vom Benutzer gelöscht wird, kann sich der eingehende Anschluß des Pfeils von dem ausgehenden Anschluß des Geschäftprozeßelements abkoppeln, und der andere Anschluß weiß, daß er nicht mehr verbunden ist.

Ein weiterer Vorteil dieser Lösung ist, daß Geschäftsprozeßelemente mehr als nur einen Ein- oder Ausgang haben können. Durch die Verkapselung des Anschlußverhaltens in eine eigene Klasse wird hier Redundanz vermieden. Anstatt jeden Anschluß selbst verwalten zu müssen, braucht jedes Geschäftsprozeßelement nur noch eine entsprechende Anzahl von Anschlüssen als Kinder zu haben.

6.5.4 Kanäle

Es gibt viele verschiedene Typen von Kommunikationskanälen, die modelliert werden können. Um diese einfach und effizient simulieren zu können, werden Kanäle nach dem Baukastenprinzip aus verschiedenen Komponenten zusammengesetzt.

Abbildung 66 zeigt ein Schema, in das die verschiedenen Kanaltypen eingeordnet sind:

	empfängt sofort	**Empfängt später**
sendet sofort / überträgt synchron	Standleitung	--
sendet sofort / überträgt asynchron	E-Mail mit SMTP	E-Mail mit Sender: SMTP, Empfänger: POP
sendet später / überträgt synchron	gepuffertes Modem	--
sendet später / überträgt asynchron	E-Mail mit Sender: POP, Empfänger: SMTP	Briefpost, E-Mail mit POP

Abb. 66: Kanaltypen-Schema

Dieses Schema ist folgendermaßen implementiert: Ein Kanal (Channel) besteht aus zwei ChannelPorts und einem Transmitter, der sie verbindet. Jeder ChannelPort wiederum besteht aus einem Sender und einem Receiver. Die in der Tabelle dargestellten unterschiedlichen Verhaltensweisen lassen sich durch verschiedene Subklassen realisieren:

- „Sender“ hat die Subklassen OnRequestSender (sendet ankommende Token sofort an den Transmitter) und BufferedSender (speichert die Token zwischen, bis eine bestimmte Zeit abgelaufen oder der Puffer voll ist)
- „Transmitter“ hat die Subklassen SynchTransmitter (überträgt synchron, das heißt alle Token verlassen ihn in derselben Reihenfolge, wie sie angekommen sind, und die Übertragungszeit ist deterministisch) und AsynchTransmitter (überträgt asynchron, das heißt

die Reihenfolge der ihn verlassenden Token ist unbestimmt, und die Übertragungszeit ist nicht deterministisch)
- „Receiver“ hat die Subklassen OnRequestReceiver (nimmt Token jederzeit vom Transmitter entgegen) und TimedReceiver (holt sich nur zu bestimmten Zeitpunkten Token vom Transmitter ab)

6.6 Zusammenfassung und Ausblick

Der vorgestellte Editor stellt die prototypische Realisierung eines Werkzeuges für verteilte Geschäftsprozesse dar. Daher gibt es noch einige technische Verbesserungsmöglichkeiten für den Editor:

- Verfeinerung der Simulation
- Verfeinerung der Auswertung
- Verbesserung der graphischen Benutzeroberfläche

Nach der Vorstellung des Editors wird dieser nachfolgend in Kapitel 7 zur Modellierung des in Kapitel 4 beschriebenen Geschäftsprozesses „Lebensversicherungs-Antrag“ eingesetzt.

7 Beispiel: Modellierung des Geschäftsprozesses Lebensversicherungs-Antrag mit dem graphischen Editor

Mit dem in Kapitel 6 vorgestellten Editor wurden die Verteilungsaspekte des in Kapitel 4 vorgestellten Geschäftsprozesses „Lebensversicherungs-Antrag“ modelliert. Die konkret durchgeführten Vorgänge wurden dabei jeweils durch das in Kapitel 6 beschriebene Verzögerungselement ersetzt, da das Hauptinteresse bei dieser Modellierung nicht auf der vollständigen Beschreibung eines Geschäftsprozesses liegt, sondern lediglich auf denjenigen Bestandteilen, die das Verhalten der Kommunikationskanäle verändern.

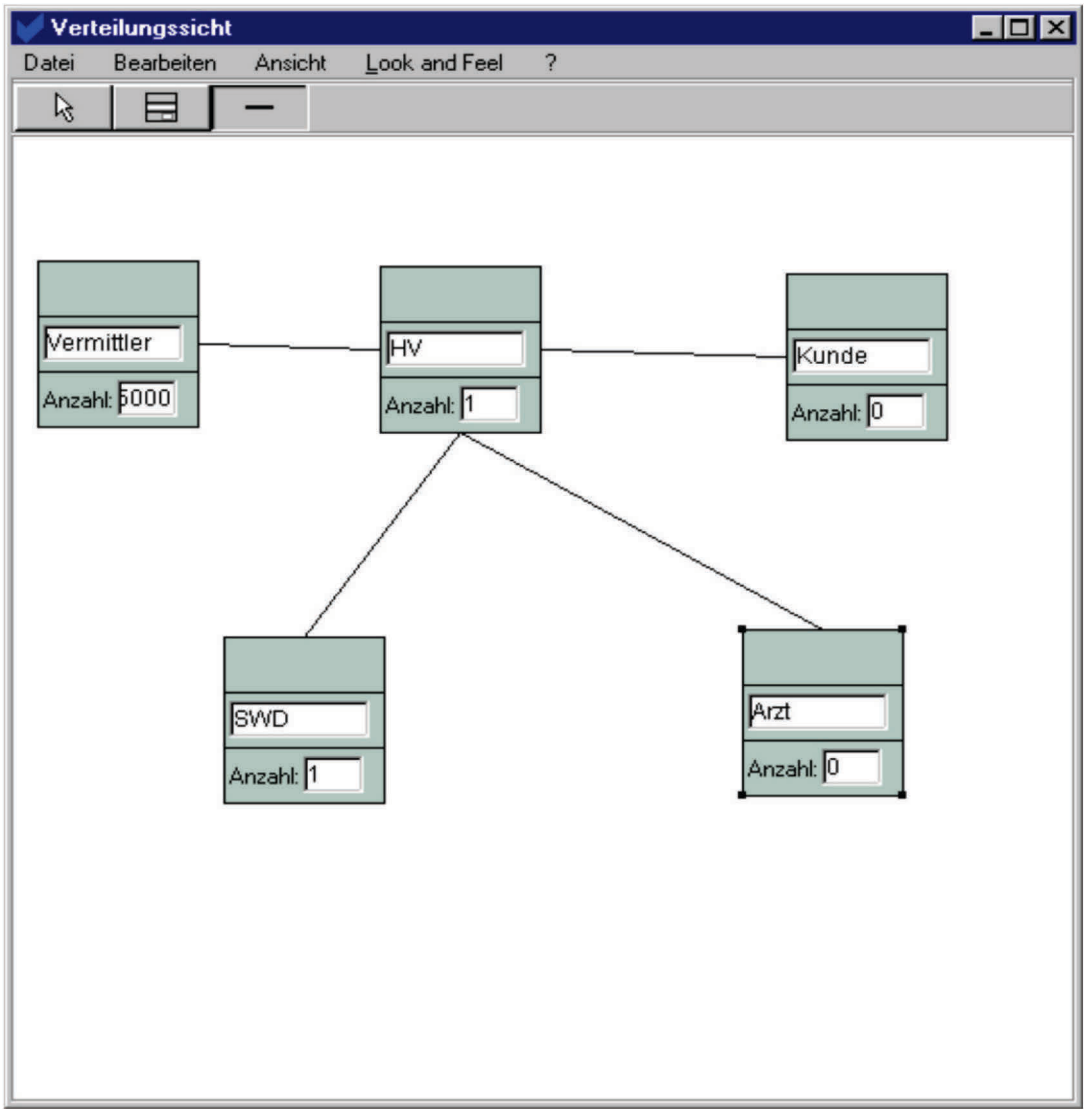

Abb. 67: Die Verteilungssicht des Geschäftsprozesses „Lebensversicherungs-Antrag“

In Abbildung 67 ist die Verteilungssicht des betrachteten Geschäftsprozesses abgebildet. Zentraler Ort der Verteilungslandschaft ist die Hauptverwaltung der Versicherung.

Diese ist über Kommunikationskanäle einerseits mit ihren Vermittlern und der Sonderwagnisdatei der Versicherer (SWD) verbunden und andererseits mit den Kunden und mit Ärzten der Kunden.

Die Hauptverwaltung besitzt wie die Sonderwagnisdatei genau eine Instanz. Als Anzahl der Vermittler wurde hier 5000 eingetragen. Ärzte und Kunden stellen bezogen auf die Anzahl der Instanzen einen Sonderfall dar. Hier wurde jeweils „0“ eingetragen, da die Anzahl zunächst unbekannt ist und die Kommunikation mit diesen Orten erst bei einem bestimmten Ablauf des Geschäftsprozesses stattfindet.

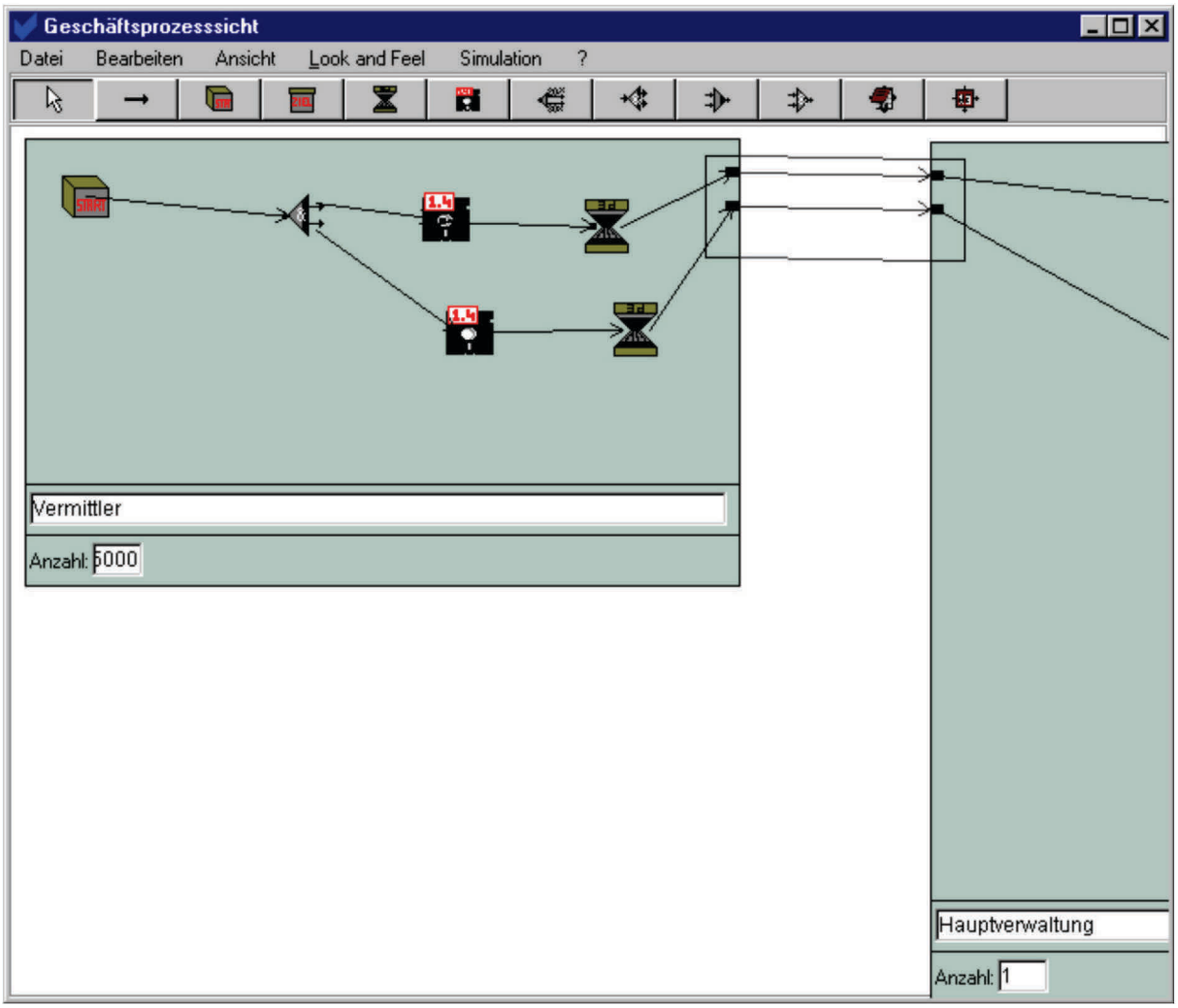

Abb. 68: Ausschnitt aus der Geschäftsprozeßsicht – Antragserfassung und Versand an die Hauptverwaltung

Abbildung 68 zeigt als Ausschnitt den Beginn des Prozesses bei den Vermittlern. Eine Eigenschaft des Startelements ist die Frequenz, mit der ein Geschäftsprozeß gestartet wird, d.h. wie oft ein Vermittler einen Neuantrag auf Lebensversicherung bearbeitet. Die im Startelement produzierten Token durchlaufen anschließend die parallele Verzweigung und werden dort in zwei Token aufgeteilt.

Der obere Zweig stellt den Bearbeitungsweg der Papierversion des Antrags dar und beinhaltet zunächst einen Knoten zur Änderung der Datenmenge. Mit ihm wird eine entsprechende Datenmenge für den nach einem Verzögerungselement folgenden Postversand eingestellt.
Das Verzögerungselement steht für die Bearbeitung des Antrags und hält jedes Token für eine gewisse Zeitdauer fest, die nach einstellbaren Parametern zufallsabhängig ermittelt wird. Der untere Zweig ist analog zum oberen aufgebaut und beinhaltet die Bearbeitung der digital gespeicherten Version des Antrags, die durch einen elektronischen Kommunikationskanal an die Hauptverwaltung übermittelt wird.

In der Hauptverwaltung wird dann die Bearbeitung des Geschäftsprozesses fortgesetzt. Einen Ausschnitt daraus zeigt Abbildung 69.

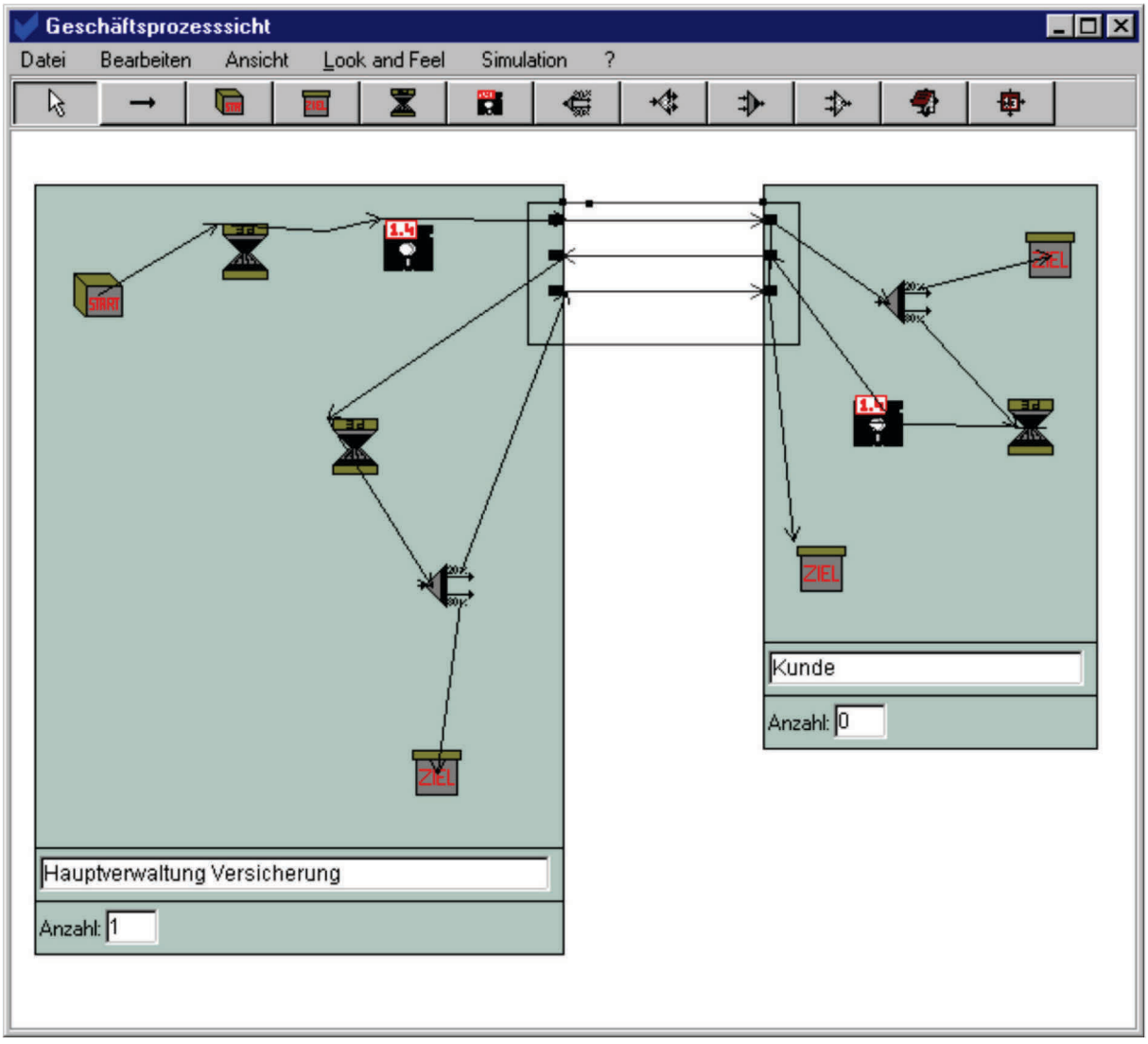

Abb. 69: Ausschnitt aus der Geschäftsprozeßsicht – Versand von Anomalieangebot und Police an den Kunden

Als Orte sind Hauptverwaltung und Kunde dargestellt. Der Prozeßausschnitt beginnt nach Ausführung der Risikoprüfung und der Entscheidung zur Abgabe eines Anomalieangebots an den Kunden. Das erste Verzögerungselement symbolisiert die Erstellung dieses Anomaliean-

gebots durch einen Sachbearbeiter. Anschließend wird daraus eine bestimmte Datenmenge erzeugt, die z.B. eine E-Mail darstellt.

Diese wird dann an den Kunden versendet, der mit einer gewissen Wahrscheinlichkeit das Anomalieangebot ignoriert, was durch die alternative Verzweigung und das direkt folgende Endeelement modelliert ist. Reagiert der Kunde nach einer gewissen Zeit (dargestellt durch das Verzögerungselement) auf das Anomalieangebot, so erzeugt auch er eine E-Mail (Änderung der Datenmenge) und schickt diese an die Hauptverwaltung zurück.

Dort wird nach einer weiteren Bearbeitungszeit mit einer hohen Wahrscheinlichkeit an der alternativen Verzweigung der Versand einer Versicherungspolice ausgelöst, wodurch der Geschäftsprozeß beim Kunden endet. Eine geringere Anzahl von Anträgen wird aber an dieser Stelle abgelehnt, so daß keine Police versendet wird und der Geschäftsprozeß in der Hauptverwaltung endet.

8 Fazit und Ausblick

Die Projektguppe ***eCCo*** konnte im Verlauf ihrer einjährigen Arbeit vielfältige Einblicke in die Versicherungsbranche gewinnen. Im Mittelpunkt standen dabei die Bemühungen des Gesamtverbandes der deutschen Versicherungswirtschaft (GDV) um die Entwicklung einer standardisierten Architektur für Versicherungsanwendungen (VAA). Die Projektgruppe ***eCCo*** konnte diese Architektur erfolgreich weiterentwickeln, indem moderne Verteilungskonzepte integriert und objektorientierte Technologien eingesetzt wurden. Von den damit verbundenen Vorteilen gegenüber der ursprünglichen VAA sind besonders die erleichterte Wartbarkeit und die Wiederverwendbarkeit von Teilen der Software zu erwähnen. Diese erweiterte Architektur VAA++ wurde prototypisch implementiert. Exemplarisch wurden innerhalb dieser Anwendung zwei Geschäftsprozesse mit Verteilungsaspekten aus der Versicherungsbranche technisch unterstützt, um die Realisierbarkeit der gewählten Lösung zu demonstrieren.

Einen weiteren Schwerpunkt der Projektgruppe ***eCCo*** bildete die Modellierung verteilter Geschäftsprozesse und deren Verteilungslandschaften sowie die Simulation von Leistungsfähigkeit und Kosten verschiedener modellierter Szenarien. Zu diesem Zweck wurde ein graphischer Editor entwickelt. Mit diesen beiden Hauptergebnissen wurden relevante Aspekte verteilter Geschäftsprozesse behandelt.

Durch den prototypischen Charakter der vorgestellten Ergebnisse bieten sich neben technischen Verfeinerungen und Verbesserungen in der Zukunft genügend Möglichkeiten für Anpassungen und Erweiterungen, die sich im Rahmen der aktuellen Forschung ergeben.

Beispiele für Möglichkeiten der weiteren Entwicklung:

- Die Steuerungs- und die Dienstebene von VAA++ können unabhängig vom Versicherungskontext verwendet werden. Durch die Bereitstellung von Anwendungsbausteinen, Dialogen, Vorgängen und Workflows können beliebige Geschäftsbereiche unterstützt werden.

- Es können auch neue Graphknoten hinzugefügt werden, um die Steuerungsmöglichkeiten von Workflows zu erweitern. Die vorhandene Knotenmenge erlaubt es, speziell die Geschäftsprozesse „Lebensversicherungs-Antrag“ und „Kraftfahrzeug-Schaden“ zu bearbeiten. Mit neuen Geschäftprozessen könnten andere Typen von Graphknoten erforderlich werden.

- In der benutzten Programmiersprache Java sind die erstellten Dialoge generell plattformunabhängig. Bei anderen Programmiersprachen, die keine plattformunabhängige Oberflächenbibliothek bieten, ist es vorteilhaft, bei der Definition von Dialogen eine Metasprache zu benutzen, um deren Struktur zu beschreiben. Dadurch werden sie unabhängig von ihrer Darstellungsart. So kann beispielsweise ein Dialog auf einem Textterminal oder auf einem graphikfähigen Bildschirm angezeigt werden, wobei dieselbe Definition benutzt wird.

- Auch Workflows können durch eine Metasprache beschrieben werden. Da es sich bei ihnen um Graphen handelt, kann man graphische Editoren benutzen, um den sie aufzubauen und in einem Beschreibungsformat abspeichern. Der Workflow-Manager könnte die erzeugte Datei lesen, um daraus die interne Darstellung eines Workflow-Graphen zu erzeugen. Vorteil dieser Lösung ist die noch leichtere Änderbarkeit von Geschäftsprozessen ohne Programmieraufwand. Zur eben beschriebenen Modellierung von Workflows für das System VAA++ könnte z.B. der in der Projektgruppe ***eCCo*** entstandene graphische Editor benutzt werden.

- Umgekehrt könnten dem Editor Informationen über diejenigen Workflows zur Verfügung gestellt werden, die in VAA++ durchgeführt werden. Damit wäre der Editor in der Lage, tatsächlich existierende Workflows in der Simulation auf Schwachstellen und Engpässe zu überprüfen. So können die zugrundeliegenden Analyse-Modelle an Praxisbeispielen getestet und verbessert werden.

Andere Weiterentwicklungen sind beim Daten-Manager möglich:

- Die Angabe der Suchkriterien bezieht sich bisher lediglich auf eine einzige Tabelle. Hier könnte die Möglichkeit geschaffen werden, auch verknüpfte Abfragen auszuführen, um effektiv beispielsweise nach einem Kunden mit einem bestimmten Namen in einem bestimmten Ort suchen zu können.

- Die Speicherung von Daten des Workflow-Managers und des Parameter-Systems geschieht in der bisherigen Version durch Serialisierung der Objekte, da diese Daten nicht über die vom Daten-Manager realisierte Transaktionssemantik angesprochen werden können. Es ist eine weitere Schnittstelle zum Daten-Manager erforderlich, um den direkten Zugang zu Daten zu ermöglichen. Wenn der Daten-Manager auch diese Daten speichert, existiert eine zentrale Schnittstelle für alle Daten innerhalb der VAA++.

- Der Daten-Manager sollte zur Verwaltung der eindeutigen Integer-Referenzen benutzt werden, die für jedes in einem Token versandte Objekt vom System vergeben werden. Bisher geschieht dies außerhalb des Daten-Managers durch die Klasse PersistenteReferenz. Der Daten-Manager kann entweder die von ihm selbst verwalteten Surrogate veröffentlichen oder einen zweiten Nummernkreis einführen. Mit der Veröffentlichung der Surrogate würde jedoch ein Implementierungs-Detail des Daten-Managers exportiert, was dem Konzept der Kapselung widerspräche.

- Für die Anbindung von Rechnern, die mit der Zentrale nicht über eine Standleitung verbunden sind, ist eine Möglichkeit zu implementieren, den Datenabgleich zwischen Zentralrechner und den nur zeitweise angeschlossenen Rechnern zu ermöglichen. Dieser Offline-Datenabgleich ist bereits entworfen und kann implementiert werden.

Die Durchführung aller genannter Verbesserungen und Erweiterungen würde den Rahmen dieser Projektgruppenarbeit sprengen. Die von der Projektgruppe ***eCCo*** erzielten Ergebnisse bieten jedoch ein breites Fundament für zukünftige Arbeiten und Weiterentwicklungen.

9 Glossar

Aktivitätsdiagramm
Ein Aktivitätsdiagramm beschreibt die Reihenfolge und Abhängigkeiten von logisch zusammengehörenden Aktivitäten. Eine Aktivität ist dabei ein einzelner Schritt in einem Verarbeitungsablauf. Die Aktivitäten eines Aktivitätsdiagramms sind eindeutigen Objekten zugeordnet. (wichtiger Unterschied zu Datenflußdiagrammen !).

Anwendungsbaustein
Er besteht aus Anwendungsobjekten und enthält logisch zusammenhängende Daten und Methoden aus einem Teil des Anwendungsgebietes. Die Methoden manipulieren die Daten oder berechnen daraus Werte. Die Objekte verschiedener Bausteine können dazu im Rahmen der vorgegebenen Hierarchie zusammenarbeiten. Die Funktionalität eines Anwendungsbausteines ist innerhalb des Anwendungsbereiches typisch und wird selten verändert.

Dialogmaske
Sie dient dazu, dem Benutzer Daten anzuzeigen oder ihm die Eingabe von Daten zu ermöglichen. Sie werden von Vorgängen zur Anzeige gebracht. Dialoge können insbesondere die Eingaben und Knopfdrücke des Benutzers nicht selbständig verarbeiten, sondern melden diese dem Vorgang, der sie aufgerufen hat.

HTML
Hyper**t**ext **M**arkup **L**anguage zur Beschreibung von Dokument-Inhalten im Internet

Interaktionsdiagramm
Sequenzdiagramme werden zusammenfassend als Interaktionsdiagramme bezeichnet. Diese Art von Diagrammen werden benutzt, um das Verhalten innerhalb eines Anwendungsfalls detailliert zu beschreiben. Sie liefern keine formale Beschreibung! Sie beschreiben, wie Gruppen von Objekten miteinander interagieren. Typischerweise beschreibt ein Interaktionsdiagramm einen Anwendungsfall (auf der Ebene der involvierten Objekte). Es werden Aussagen über die Anzahl der involvierten Objekte und über den Nachrichtenaustausch zwischen den Objekten gemacht.

Kanal
siehe Kommunikationskanal

Klasse
Begriff aus der objektorientierten Software-Entwicklung: Eine Klasse ist die Definition der Attribute, Operationen und der Semantik für eine Menge von Objekten. Alle Objekte einer Klasse entsprechen dieser Definition. Sie enthält die Beschreibung der Struktur und des Verhaltens von Objekten, die sie erzeugt oder mit ihr erzeugt werden können. Die Definition einer Klasse setzt sich aus Attributen und Operationen (auch Methoden genannt) zusammen.

Klassendiagramm
Eine Diagrammform aus der Unified Modeling Language (UML) bei der die Klassen und ihre Beziehungen untereinander gezeigt werden.

Kommunikationskanal
Ein Kommunikationskanal ist eine Verbindung zwischen zwei geographisch getrennten Orten, über die Daten ausgetauscht werden. Zum Beispiel eine Standleitung, eine Verbindung per Modem über das Telefonnetz, oder ein regelmäßiger Datenausstausch per E-Mail oder Post.

Leinwand
Eine leere weiße Fläche in einem Fenster, auf der andere graphische Objekte angelegt und verschoben werden können.

Ort
Ein geographischer Ort, an dem Teile eines Geschäftsprozesses stattfinden können. Orte können durch Kommunikationskanäle verbunden sein, so daß Geschäftsprozesse auch über mehrere Orte verteilt ablaufen können.

Service
Derjenige Teil eines Geschäftsprozesses, der einen Kommunikationskanal benutzt. Wird der Geschäftsprozeß als Graph dargestellt, ist eine Kante des Graphen dann ein Service, wenn sie an einem anderen Ort beginnt als endet.

Token
Eine Informationseinheit, die in einem Geschäftsprozeß (oder Workflow) transportiert wird.
a) im Geschäftsprozeß-Editor:
Ein Token wird während der Simulation als Einheit betrachtet. Jedes Token befindet sich zu einem bestimmten Zeitpunkt der Simulation an einem bestimmten Ort; jedoch kann ein Token in mehrere Token aufgespalten werden. Die Bruder-Token können später wieder vereinigt werden; aus ihnen wird dann wieder ihr Vater gemacht.
b)in der VAA++:
Ein Token enthält ein numerisches Ergebnis, eine Referenz auf eine Menge von Anwendungsobjekten und Informationen darüber, wann es an den nächsten Knoten weitergeleitet werden soll. Dies kann zu einem festgesetzten Zeitpunkt oder bei Eintreffen einer angegebenen Nachricht erfolgen.

UML-Notation
Die UML ist eine Modellierungssprache zur objektorientierten Software-Entwicklung. Sie wird zur Modellierung von Objekten, Komponenten und Geschäftsprozessen verwendet. Sie beschreibt eine einheitliche Notation und Semantik und dient insbesondere der Definition eines Metamodells.

Vererbung
Vererbung bedeutet in der objektorientierten Software-Entwicklung die Wiederverwendung von Eigenschaften einer Klasse. Dabei werden alle Eigenschaften einer Klasse, der sogenannten Oberklasse, von anderen Klassen, den sogenannten Unterklassen weiterverwendet und gegebenenfalls mit weiteren Eigenschaften erweitert und spezialisiert.

Verteilung, verteilt, Verteilungslandschaft
Ein Geschäftsprozeß ist verteilt, wenn er sich über mehrere Orte erstreckt. Das bedeutet, daß er einen oder mehrere Kommunikationskanäle benutzen muß. Verteilungslandschaft heißt die Gesamtheit aller Orte und Kommunikationskanäle, die zur Beschreibung eines Geschäftsprozesses (oder einer Menge von Geschäftsprozessen) benötigt wird.

Vorgang
Dabei handelt es sich um eine einzelne Teilaufgabe des Geschäftsprozesses, die von einem Bearbeiter in einer zusammenhängenden Arbeitssitzung bearbeitet werden kann. Alle Änderungen an Daten, die innerhalb eines Vorgangs anfallen, werden gleichzeitig gesichert oder verworfen („Alles-oder-nichts-Prinzip“, auch „Transaktionsprinzip“). In einem Vorgang wird aus einem Eingabeobjekt (Token) ein neues Ausgabe-Token erzeugt.

Werkzeugleiste
engl. toolbar. Eine Leiste am oberen Rand eines Fensters, unterhalb des Menübalkens, auf der sich durch graphische Symbole dargestellte Werkzeuge befinden. Jedes dieser Werkzeuge ist ein Schaltknopf und löst beim Drücken eine bestimmte Aktion aus, z.B. kann es die Funktionsweise (und ggf. das Aussehen) des Mauszeigers ändern.

Workflow
Er beschreibt die Reihenfolge der Ausführung von Vorgängen innerhalb eines Geschäftsprozesses und definiert den Fluß der Arbeitsergebnisse mehrerer Bearbeiter an mehreren Orten zu verschiedenen Zeiten. Ein Workflow ist ein Graph mit Knoten, welche die Ausführung eines Vorgangs starten zur Steuerung des Workflows dienen. Entlang der Graphkanten werden die Arbeitsergebnisse durch ein Token von einem Knoten zum nächsten transportiert.

Zustandsdiagramm
Zustandsübergangsdiagramme beschreiben die erlaubten Zustandsübergänge eines Objektes und die Ereignisse, die die Übergänge auslösen. Zustände werden durch abgerundete Rechtecke dargestellt, Übergänge durch Pfeile. Syntax für Pfeilanschriften: Ereignis [Bedingung] Aktion wobei alle drei Teile optional sind. Wenn in einem Zustand interne Operationen ausgeführt werden sollen, so finden sich diese nach dem Schlüsselwort "tue/" im unteren Teil eines Zustandes.

10 Abbildungsverzeichnis

11 Literatur

[Ach97] W. Achtert: Eine Beispielarchitektur für objektorientierte Systeme; in: Objekt Fokus Ausgabe 11/12; 1997.

[Bal98] Helmut Balzert: Lehrbuch der Software-Technik (in zwei Bänden), Spektrum Akademischer-Verlag, Heidelberg, Berlin 1996, 1998.

[Bef96] Vortrag von Volker Befurt, Karlsruhe, 1996.
(www.inovis.deinovis2/aktuell/vortrag/v-vb1.htm)

[Boo93] Grady Booch: Object Oriented Analysis and Design with Applications, Benjamin/Cummings 1993.

[Col95] J. Coldewey, W. Keller: Objektorienterte Datenzugriffe auf dem VAA-Datenmanager; sd&m für GDV-Arbeitskreis VAA Datenmanager; November 1995.

[EECT] Esprit's Electronic Commerce Team: Electronic Commerce an Introduction
(www.cordis.lu/esprit/src/ecomint.htm)

[Far98] J. Farley: Java – Distributed Computing, 1. Auflage; O´Reilly & Associates, 1998.

[Fow97] Martin Fowler, Kendall Scott: UML – konzentriert, Addison-Wesley, 1998.

[GDV96] GDV: System zur Dokumentation und Präsentation der Versicherungs-Anwendungs-Architektur (Version 1.0), 1996.

[GHJ95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissidis: Entwurfsmuster, Bonn, 1996.

[GWM96] Geschäftsprozeßmodellierung und Workflow-Management, Bonn, Thomson 1996.

[Krü97] Guido Krüger: Java 1.1 lernen, Addison-Wesley, Bonn, 1997.

[Mey88] Bertrand Meyer: Objektorientierte Software-Entwicklung, Hanser, München, Wien, 1990, Prentice-Hall, London, 1988.

[Oes98] Bernd Oestereich: Objektorientierte Software-Entwicklung Analyse und Design mit der Unified Modeling Language, R. Oldenbourg Verlag, München, 1998.

[Raa98] Jörg Raasch: Eine Komponentenarchitektur für Versicherungsanwendungen, in: Versicherungswirtschaft 8/1998, S.514-520.

[Rin94] Rinza, Peter: Projektmanagement, VDI Verlag, Düsseldorf, 1994.

[Som92] Ian Sommerville: Software Engineering, fourth edition, Addison-Wesley 1992.

[Sto98] Christoph Stoppe: Service Engineering – eine Methode zur Definition, Visualisierung und Simulation von Electronic-Commerce-Anwendungen, Diplomarbeit am Lehrstuhl Software-Technologie, Fachbereich Informatik, Universität Dortmund 1998.