Anbindung und Kapselung existierender Anwendungssysteme

Diplomarbeit von Fabian Nilius

Arbeitsbereich Softwaretechnik
Fachbereich Informatik
Universität Hamburg

Februar 2002

Erklärung:

Hiermit versichere ich, diese Arbeit selbständig und unter ausschließlicher Zuhilfenahme der in der Arbeit aufgeführten Hilfsmittel erstellt zu haben.

Hamburg, den 27. Februar 2002

Fabian Nilius
Wiesendamm 17a
22305 Hamburg
040 / 299 65 00

Betreuung:

Prof. Dr. Heinz Züllighoven (Erstbetreuer)
Prof. Dr. Florian Matthes (Zweitbetreuer)

Prof. Dr. Heinz Züllighoven
Arbeitsbereich Softwaretechnik (SWT)
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
D-22527 Hamburg

Prof. Dr. Florian Matthes
Arbeitsbereich Softwaresysteme (STS)
Forschungsschwerpunkt 4 (IuK Technologie)
Technische Universität Hamburg-Harburg
Harburger Schloßstr. 20
D-21073 Hamburg

Zum Titelbild:

Das Foto zeigt das Gebäude einer Hamburger Krankenkasse (Ecke Habichtstraße / Bramfelder Straße). Vor dem Hauptgebäude - einem technischen Bau aus Glas und Stahl – steht als Schnittstelle zu den Konsumenten ein freundliches Backsteingebäude, das die Komplexität des Hauptgebäudes verbirgt.

»Alles auf der Welt ist eine zukünftige Antiquität.«
Bernard Buffet

Inhalt

1 Einleitung

Zwei Welten treffen aufeinander: Eine neue Software soll entwickelt werden, mit modernen softwaretechnischen Methoden, orientiert an den Anforderungen der Benutzer – auf der Basis des in die Jahre gekommenen Anwendungssystems.

Das existierende Anwendungssystem stellt häufig keine ideale Basis für die neue Anwendung dar, dafür gibt es mehrere Gründe:

- Die junge Disziplin Informatik entwickelt sich so rasch, daß die Technologien und Konzepte innerhalb weniger Jahre veralten.
- Anwendungssysteme werden um neue Funktionen erweitert und verlieren dadurch ihr einheitliches Benutzungskonzept.
- Mit der Zeit ändern sich die Anforderungen der Benutzer.

Auf das bisherige Anwendungssystem kann und soll aber nicht verzichtet werden; schließlich stecken in ihm viel Entwicklungszeit und Geld.

In dieser Diplomarbeit werden Konzepte vorgestellt, wie ein existierendes Anwendungssystem angebunden, seine Benutzungskonzepte herausgefunden und wie es mit einer *modernen Schnittstelle* gekapselt werden kann. Ziel der Kapselung ist eine Abkopplung der neuen Software von den Technologien und Konzepten des bisherigen Anwendungssystems: Die existierende Funktionalität wird hinter einer einheitlichen, modernen und an den aktuellen Anforderungen orientierten Fassade verborgen. Dadurch ist die neue Software frei von Relikten und alten Technologien.

Kern der Diplomarbeit ist ein *Vorgehensmodell*, das einen Rahmen für die Anbindung existierender Anwendungssysteme bietet. Das Modell reicht von der Analyse der Anforderungen und des existierenden Systems bis zur Implementierung der Kapselung. Das Vorgehensmodell basiert auf Überlegungen zum technischen *Benutzungsmodell* von existierendem System und neuer Schnittstelle. Zentraler Punkt des Vorgehensmodells ist die Ausarbeitung eines vereinheitlichten und fachlichen Benutzungsmodells auf Basis der aktuellen Anforderungen und der existierenden Funktionalität.

Das Vorgehen wurde in einem Projekt im Anwendungsbereich Telefonie erprobt, in dem mit Fachlichen Services ein *Application Programming Interface* (API) zu einem Callcenter-System geschaffen wurde. Diese Arbeit stellt die Ergebnisse und Erfahrungen aus diesem praktischen Projekt vor: Wie hat sich das Modell bewährt, wo lagen die Schwierigkeiten der Kapselung?

Zur Beschreibung und Konstruktion der Kapselung wird die Entwurfsmetapher der *Fachlichen Services* verwendet, die von Otto und Schuler im Kontext der Werkzeug-Automat-Material-Methode (WAM) am Arbeitsbereich Softwaretechnik der Universität Hamburg entwickelt wurde [Otto & Schuler 2000]. Fachliche Services sind Softwarekomponenten, die anderen Komponenten eine *Dienstleistung* anbieten. Hier basiert die Dienstleistung der Fachlichen Services auf der Funktionalität des existierenden

Anwendungssystems, die Fachlichen Services sind also die Schnittstelle, über die eine neue Anwendung das existierende System anspricht.

Diese Diplomarbeit führt eine neue *Notation für technische Benutzungsmodelle* ein, die einen Überblick über die Umgangsformen eines Fachlichen Services gibt; und diskutiert die Notation der Architektur mit Komponentendiagrammen.

Im letzten Schritt des Vorgehensmodell wird ausführlich die technische Umsetzung verteilter Fachlicher Services behandelt. Für den technischen Umgang wird der Begriff des *Protokolls* eines Fachlichen Services eingeführt. Eine *Checkliste* bietet eine Übersicht über die häufigsten Probleme und Entwurfsentscheidungen.

Diese Arbeit bietet eine wissenschaftliche Behandlung der Anbindung existierender Anwendungssysteme, darüber hinaus hoffe ich, daß sie auch eine Hilfestellung für Anbindungen in zukünftigen praktischen Projekten ist.

Danksagung

Mein Dank gilt allen, die mit Rat, Anregungen und Kritik zum Gelingen dieser Diplomarbeit beigetragen haben.

Professor Dr. Heinz Züllighoven danke ich für den Anstoß zu dieser Arbeit und für viele hilfreiche Anregungen. Professor Dr. Florian Matthes danke ich für die Zweitbetreuung der Arbeit.

Guido Gryzcan und besonders Henning Wolf vom Arbeitsbereich SWT danke ich für viel Feedback und Diskussionen.

Der Firma Tenovis Business Communication GmbH danke ich für die Unterstützung dieser Diplomarbeit, ebenso allen Kollegen und besonders Thorsten Görtz und Christian Fischer für die vielen Denkanstöße im praktischen Projekt.

Ein großer Dank gebührt auch meinem Vater Klaus Nilius für das Korrekturlesen.

Sprachgebrauch

Neu eingeführte Begriffe werden beim ersten Benutzen *kursiv* gesetzt.

Die deutsche Sprache teilt mit anderen Sprachen die Eigenschaft, daß bei Gruppen mit männlichen und weiblichen Teilnehmern als Gruppenbezeichnung die männliche Form gewählt wird. Dies führt allerdings dazu, daß die weibliche Form in der Sprache untergeht und somit die Präsenz von Frauen z.B. im Anwendungsbereich Telefonie nicht deutlich wird [vgl. Braun 97]. Für diese Arbeit wurde der folgende Weg gewählt: Allgemeine Abschnitte und Definitionen sind in der männlichen Form gehalten. Konkrete Beispiele und Szenarien verwenden hingegen beide Formen.

2 Grundlagen

Dieses Kapitel gibt eine Einführung in die softwaretechnischen und anwendungsfachlichen Grundlagen dieser Arbeit. Vorgestellt werden der WAM-Ansatz, das Konzept der Fachlichen Services und der Anwendungsbereich Telefonie und Callcenter.

2.1 WAM und Fachliche Services

Diese Arbeit entstand an der Universität Hamburg im Kontext der dort entwickelten WAM-Methodik. Dieser Abschnitt bietet zum Verständnis dieser Arbeit eine kurze Einführung in diesen Ansatz, für eine ausführliche Einführung sei auf [Züllighoven 98] verwiesen.

Werkzeug, Automat, Material (WAM)

Der WAM-Ansatz ist ein Methodenrahmen zur Unterstützung der Entwicklung objektorientierter Software: bei der Analyse und Dokumentation, der Konstruktion und generell in der Ausgestaltung des Softwareentwicklungsprozesses. WAM bietet einen umfangreichen Satz an Vorgehensweisen und Metaphern an, aus diesem Rahmen an Methoden kann sich ein Entwickler die für seine Zwecke geeigneten auswählen.

»Die zentrale Idee des Werkzeug & Material-Ansatzes ist, die Gegenstände und Konzepte des Anwendungsbereichs als Grundlage des softwaretechnischen Modells zu nehmen. Das Ergebnis sollte enge Korrespondenz zwischen dem anwendungsfachlichen Begriffsgebäude und der Softwarearchitektur sein.« [Züllighoven 98, S. 4]

Durch diese *Strukturähnlichkeit* finden sich Anwender schnell mit den Konzepten der Software zurecht. Die Systeme werden Änderungen gegenüber resistenter, solange sich nicht ebenfalls massive Änderungen im Anwendungsbereich ergeben. Wissen über den Anwendungsbereich ermöglicht ein Verständnis der Softwarearchitektur.

Die Gestaltung der Software orientiert sich dabei an den *Aufgaben* der Sachbearbeiter und an *Leitbildern*. Ein typisches WAM-Leitbild ist der „Funktionsarbeitsplatz für eigenverantwortliche Expertentätigkeit“. Einem solchen Benutzer zeichnet sein fachliches Expertenwissen aus, entsprechende Werkzeuge unterstützen ihn in seiner Arbeit, binden ihn aber nicht an starre Arbeitsabläufe.

Bei der Konstruktion der Software wird der Entwickler durch *Entwurfsmetaphern* unterstützt. Zu den klassischen Entwurfsmetaphern gehören *Werkzeuge, Automaten* und *Materialien,* die dem WAM-Ansatz seinen Namen gaben. Materialien verkörpern Gegenstände des Anwendungsbereichs. Mit Werkzeugen kann der Anwender Materialien bearbeiten, Automaten bearbeiten Materialien nach einer Initialisierung ohne weiteren Eingriff des Anwenders. *Fachwerte* modellieren Werte und Wertebereiche des Anwendungsbereichs (z.B. die Körpertemperatur).

Dabei werden im wesentlichen nur die Materialien und Fachwerte (das *fachliche Modell)* aus dem Anwendungsbereich in die Softwarearchitektur übernommen: Während sich die Gegenstände des Anwendungsbereichs oft gut in das Anwendungssystem übernehmen lassen, ist es schwierig, Funktionalität entsprechend abzubilden: Formulare lassen sich übernehmen, hingegen sind ein Stift oder eine Schreibmaschine und eine Textverarbeitung sehr verschieden. Die Werkzeuge stammen in der Regel nicht aus dem Anwendungsbereich, sondern sind neu entwickelte Hilfsmittel für die Sachbearbeiter, mit denen diese zukünftig ihre Arbeit erledigen sollen.

Weitere Entwurfsmetaphern sind z.B. die *Registratur* und die *Vorgangsmappen* zur Unterstützung kooperativer Arbeit.

WAM unterstützt die Umsetzung der Entwurfsmetaphern durch geeignete Entwurfsmuster. So werden Werkzeuge in eine *Funktionskomponente* und eine *Interaktionskomponente* getrennt, die über einen *Beobachtermechanismus* [vgl. Gamma 96, S. 257ff] verbunden sind.

Der Softwareentwicklungsprozeß WAM ähnelt dem *iterativen*, fachlich orientierten Unified Process [Jacobsen Booch Rumbaugh 99]. Im Mittelpunkt stehen *Dokumente*, wobei ein besonderer Schwerpunkt auf einem *Autor-Kritiker-Zyklus* mit den fachlichen Benutzern liegt. Hier benutzt WAM *Szenarien* (vergleichbar mit Business Use Cases), *Systemvisionen* (Use Cases)*, Glossare, Kooperationsbilder* (zeigen die kooperierenden Personen und die Kommunikationswege) und *Prototypen*.

Der nächste Abschnitt stellt die WAM-Entwurfsmetapher der Fachlichen Services vor.

Fachliche Services

Fachliche Services betreffen Konzepte und Fragestellungen, die sich im Kontext der Entwicklung von Unternehmenssoftware ergeben. Unternehmenssoftware wird dabei von Allen und Frost wie folgt charakterisiert:

»An enterprise system is one that meets the needs of a large business with complex business processes.« [Allen & Frost 98, S. 1]

In dieser Arbeit wird der Begriff *Anwendungssystem* synonym zu Unternehmenssoftware verwendet. Im Gegensatz zu einer Anwendung auf einem einzelnen Arbeitsplatz zeichnet sich ein Anwendungssystem dadurch aus, daß es

- sehr umfangreich und komplex ist und aus vielen zusammenarbeitenden Teilen besteht,
- aus fachlichen und technischen Gründen aufgeteilt auf mehreren Computern läuft, ein verteiltes Anwendungssystem ist und
- über einen längeren Zeitraum von mehreren Personen entwickelt wurde.

Während die Entwurfsmetapher Werkzeug gut zur Konstruktion von Software auf einem einzelnen Arbeitsplatz geeignet ist, wurde mit den *Fachlichen Services* in der Diplomarbeit von Otto und Schuler eine Entwurfsmetapher zur Konstruktion großer Anwendungssysteme eingeführt [vgl. Otto & Schuler 2000]. Dieser Abschnitt faßt die wesentlichen Definitionen und Ergebnisse dieser Arbeit zusammen.

Motivation für ihre Arbeit war ein Anwendungssystem, das über verschiedene Benutzungsschnittstellen wie *Desktop* oder *WWW-Schnittstelle (Webtop)* benutzt werden sollte. Aus den Schwierigkeiten, für diese unterschiedlichen Front-Ends eine gemeinsame Basis zu finden, entwickelte sich das Konzept der *Fachlichen Services*. Deren zentrale Idee ist, die allen Anwendungstypen gemeinsame *fachliche Funktionalität* zusammengefaßt für alle Benutzungsschnittstellen-Typen *als Dienstleistung* bereitzustellen.

Die Begriffe Dienstleistung und Service

Fachliche Services orientieren sich an dem Begriff der Dienstleistung. Hierzu schreibt der Brockhaus:

»*Dienstleistungen*, wirtschaftliche Verrichtungen, die nicht in der Erzeugung von Sachgütern, sondern in persönlichen Leistungen bestehen.« [Brockhaus 78, S.182]

In der Wirtschaft versteht man unter Kundenservice die Betreuung der Kunden bei allen Fragen rund um das Produkt. Hier ist Kundenservice eine Säule neben dem Vertrieb, dem Marketing und der Fertigung. Oft wird der Begriff Service auch benutzt, um eine besondere Kundenorientierung auszudrücken, z.B. in dem Begriff „Serviceorientiert“. Die Metapher *Service* in der Softwaretechnik übernimmt diese Vorstellung, einen kundenorientierten Dienst zu leisten.

Fachliche Services

Fachliche Services bieten eine Entwurfsmetapher für die Konstruktion großer Anwendungssysteme: Eine bestimmte Funktionalität wird als Dienstleistung betrachtet, das System wird in Analogie zu einem umgangssprachlichen Service konstruiert.

> **»Definition: Fachlicher Service**
> Ein Fachlicher Service fasst kohärente Funktionalität und Wissen eines Anwendungsbereichs unabhängig von einer Benutzungsschnittstelle zusammen, kapselt sie und stellt sie als Dienstleistungsbündel zur Verfügung.
>
> Die Dienstleistungen eines Fachlichen Services werden über Aufträge von Konsumenten in Anspruch genommen.« [Otto & Schuler 2000, S. 67]

Nach außen präsentiert sich der Fachliche Service über ein eindeutiges Interface, er verbirgt seine innere Realisierung vor dem Konsumenten. Über die Konsumenten werden vom Service so wenig Annahmen wie möglich gemacht. Eine Darstellung und Präsentation an der Benutzungsschnittstelle ist Aufgabe des Konsumenten. Ein Fachlicher Service kann für die Erfüllung seiner Funktionalität einen anderen Fachlichen Service als Konsument benutzen.

Von Otto und Schuler werden zwei Ausprägungen Fachlicher Services beschrieben:

- Fachliche Services, *die Material und Umgang bereitstellen*, gehen mit Gegenständen der Anwendung um und bewahren diese oft auch auf. Ein solcher Service kapselt in erster Linie das Wissen um den Umgang mit den Anwendungs-Gegenständen und bietet Funktionalität an.

- Fachliche Services, die *Vorgänge vergegenständlichen*, nehmen dem Benutzer Wissen über die komplexen Zusammenhänge und Implikationen von Vorgängen in der Anwendungswelt ab.

  »Ein solcher Fachlicher Service kapselt also das Wissen darüber, in welchen möglichen Reihenfolgen, mit welchen wechselnden Zuständigkeiten und mit welchen Folgen Vorgänge ablaufen. Dabei vergegenständlicht er einen Vorgang durch ein Material, das so genannte Prozeßmuster (siehe [Gryczan 96]), beispielsweise in Form eines Laufzettels.« [Otto & Schuler 2000, S. 72]

Ein Fachlicher Service muß *keine* fachliche Dienstleistung des Anwendungsbereichs verkörpern, er erbringt seine Dienstleistung aber in der Regel an Materialien des Anwendungsbereichs und wird zur Erfüllung spezieller Aufgaben des Anwendungsbereichs eingesetzt. Ein Material kann charakteristisch für einen Service sein und ausschließlich von diesem Service genutzt werden, es kann aber Materialien geben, die mit mehreren Services bearbeitet werden können.

Von Otto und Schuler wird eingehend die Frage diskutiert, in welcher Form *an der Schnittstelle* eines Fachlichen Services Materialien verfügbar sind:

»In Material und Umgang bereitstellenden Fachlichen Services ist die Funktionalität auf die Handhabung und Aufbewahrung der Materialien der Anwendung konzentriert. Daher ist der Umstand, ob und wie viel der Materialien auch außerhalb des Fachlichen Services zur Verfügung steht und dort bearbeitet werden kann, zentraler Punkt für die Kapselung dieser Art Fachlicher Services.« [Otto & Schuler 2000, S. 83]

Die folgenden Alternativen für den Umgang mit Materialien werden diskutiert:

- *Schnittstelle gibt nur Werte bzw. Fachwerte heraus*
  Eine solche Schnittstelle kapselt die verwendeten Materialien und gibt nur einzelne Attribute heraus. Otto und Schuler weisen darauf hin, daß eine solche strikte Kapselung kontraproduktiv sein kann, da dann die Konsumenten des Service die Werte oft wieder zu eigenen Materialien zusammensetzen.
- *Schnittstelle gibt nur Materialien nach WAM heraus*
  Diese Materialien erlauben einen fachlichen Umgang. Problematisch ist, daß dadurch den Konsumenten eine Änderung der Materialien möglich ist, die in der Regel ein tieferes Verständnis des Anwendungssystems voraussetzt.

  Damit eine Änderung wirksam wirkt, muß der Fachliche Service Methoden zum zurückstellen von Materialien anbieten.
- *Schnittstelle gibt nur Materialien nach WAM unter sondierendem Aspekt heraus*
  Diese Materialien können nur gelesen werden, sie können z.B. als schreibgeschützte Kopie verstanden werden. Eine Änderung von Materialien ist hier nur über die Dienstleistungen des Service möglich.

Diese Fragestellung wird auch im Rahmen dieser Arbeit weiter diskutiert, besonders im Kapitel 3.1 und im Kapitel 5.3 im Abschnitt über Kooperation und Verteilung. Ein Ergebnis ist, daß hier bei den Fachlichen Services die Einführung eines fachlichen Modells der Materialien einen wesentlichen Gewinn gegenüber der existierenden

wertbasierten Schnittstelle darstellt. Die Entscheidung für eine der oben skizzierten Alternativen sollte sich an dem fachlichen *Kooperationsmodell* orientieren.

Da Otto und Schuler in ihrer Arbeit Fachliche Services für verschiedene Benutzungsschnittstellen einsetzen wollen, müssen die Materialien *oberflächenneutral* in standardisierter Form angeboten werden.

Einordnung Fachlicher Services

Otto und Schuler stellen fest:

»Der Begriff „Service“ oder „Dienst“ wird in der Informatik insgesamt entweder sehr technisch oder sehr allgemeingültig verstanden. Im Sinne der Erbringung anwendungsfachlicher Leistungen taucht er hingegen fast gar nicht auf.«

[Otto & Schuler 2000, S.42]

Sie diskutieren ausführlich einige verwandte Konzepte, die an dieser Stelle nur kurz genannt werden sollen. Für eine tiefergehende Gegenüberstellung sei auf [Otto & Schuler 2000] verwiesen.

Softwaretechnische Grundlagen

Fachliche Services unterstützen die Aufteilung komplexer Anwendungssysteme und orientieren sich damit an klassischen Konzepten zur Modularisierung.

Grundlegend ist hier der Begriff des »Information Hiding« (Kapselung) und der Begriff der »Losen Kopplung«: Zwischen den Bereichen eines Softwaresystems findet eine Kommunikation über wohldefinierte öffentliche Schnittstellen statt. Das Wissen über andere Teile ist auf diese Schnittstellen beschränkt. Interne Entwurfsentscheidungen werden vor den anderen Teilen verborgen.

»Every Module [...] is characterized by it’s knowledge of a design decision which it hides from all others. Its interface or definition was chosen to reveal as little as possible about its inner workings« [Parnas 72, S. 1056]

Diese Kapselung ermöglicht lokale Änderungen, die keine Auswirkungen auf andere Bereiche des Softwaresystems haben. Dadurch daß in einem Bereich so wenig Wissen wie möglich über die anderen Bereiche benutzt wird, kann die Funktionalität eines Bereichs in wechselnden Einsatzkontexten benutzt werden, die ursprünglich nicht vorhersehbar waren.

Fachliche Services stellen eine Entwurfsmetapher zur Verfügung, die vergleichbar mit dem klassischen Modulkonzept ist. Die Entwurfsmetapher Service bietet – im Gegensatz zum Modul – unter anderem den Vorteil, daß sie fachlich leichter zu motivieren und anschaulicher ist.

Fachliche Services folgen auch dem Konzept der *Trennung von Interaktion und Funktion:* Softwaresysteme sind überschaubarer und Änderungen gegenüber stabiler, wenn die Funktionalität abgekoppelt von der reinen Präsentation entwickelt wird. Die Funktionalität kann so in verschiedenen Benutzungsschnittstellen verwendet werden, kleinere Änderungen der Funktionalität haben keine Auswirkungen auf die Benutzungsschnittstelle.

Ein Fachlicher Service bietet eine reine Funktionalität an. Eine eventuelle grafische Benutzungsschnittstelle wird vom Konsumenten umgesetzt, dabei wird die Handhabung des Konsumenten allerdings durchaus durch das Benutzungsmodell des Service beeinflußt.

Client-Server-Systeme

Um mit Verteilung und der Komplexität von Unternehmenssoftware umzugehen, werden schon seit den 70er Jahren Konzepte zur Aufteilung der Software entwickelt. Diese Darstellung folgt im wesentlichen [Singh 99] und der Darstellung bei [Otto & Schuler 2000].

- Ende der 70er Jahre herrschten Systeme vor, in denen die gesamte Geschäftslogik monolithisch auf einem zentralen Host-Rechner implementiert war *(One-Tier-Architekturen)*, die Benutzer verbanden sich über „dumme“ Terminals mit diesem Host-Rechner.
- Mit dem Aufkommen leistungsfähiger Arbeitsplatzrechner in den 80er und frühen 90er Jahren verlagerte sich die Logik auf die Benutzer-Rechner. Hier führte der *Client* lokale Aufgaben mit lokalen Ressourcen durch, der *Server* bot zentrale und gemeinsam genutzte Ressourcen an. In diesen frühen *Client-Server-Systemen* bot der Server zum Beispiel den Zugriff auf eine zentrale relationale Datenbank, der Client konnte SQL-Anfragen an den Server senden. Die wesentliche Geschäftslogik war auf dem Client-Rechner implementiert *(Fat Client)*.
- In den 90er Jahren wurden Datenbanken um Prozeduren *(stored prozedures)* erweitert, so daß an zentraler Stelle Konsistenzbedingungen geprüft werden und Funktionalität direkt in der Datenbank implementiert werden kann.
- Eine konzeptuelle Verallgemeinerung dieser Entwicklung für Client-Server-Systeme sind *Two-Tier-Architekturen*, bei denen zwischen der Benutzungsschnittstelle (auf dem Client), der Geschäftslogik (auf Client und Server) und der Datenbank (auf dem Server) unterschieden wird. Solche Systeme sind besser skalierbar als monolithische One-Tier-Architekturen und werden heutzutage im Small/Home-Office-Bereich mit ca. 100 bis 150 Mitarbeitern eingesetzt.
- Mit der zunehmenden Komplexität der Unternehmenssoftware und der wachsenden Bedeutung des Internet, wuchs der Bedarf an flexibleren und leistungsfähigeren Konzepten. *Three-Tier-Architekturen* verlagern die Geschäftslogik in eine eigene Schicht: Der Client übernimmt nur noch die Präsentation, die Geschäftslogik wird von einem *Applicationserver* angeboten, der wiederum auf zentralen Datenbanken und Ressourcen aufsetzt. Für die Lastverteilung interessant ist, daß die Geschäftslogik von mehreren gespiegelten Applicationservern angeboten werden kann, zwischen denen die Last aufgeteilt wird. Die technischen Aufgaben der mittleren Schicht (z.B. Caching, Last-Ausgleich) werden oft durch Infrastrukturen wie *Middlewares* unterstützt.

Otto und Schuler kritisieren an Client-Server-Konzepten, daß schichtenlokale Änderungen in der Praxis eher die Ausnahme sind, da Änderungen der Geschäftslogik auch Einfluß auf die Benutzungsschnittstelle haben. Weiter sei eine zentrale Geschäftslogik auch monolithisch und dadurch schwieriger zu handhaben.

»In dieser Arbeit soll deshalb vom Konzept mehrstufiger Softwarearchitekturen in erster Linie die Idee der Verteilung von Software und Ziehung von Systemgrenzen sowie

der Kerngedanke der Bündelung und Abtrennung fachlicher Funktionalität übernommen werden. Mehrstufige Softwarearchitekturen sollen nicht als Orientierung bei der Modellierung verwendet werden, hier orientieren wir uns am WAM-Ansatz und dessen rahmenwerksbasierten Architekturen (siehe [Züllighoven 98]) [...].«

[Otto & Schuler 2000, S. 51]

Ein wesentliches Konzept dieser rahmenwerksbasierten Architektur ist eine Orientierung der Aufteilung der Unternehmenssoftware an den fachlichen Geschäftsbereichen [vgl. Bäumer 98]. Fachliche Services werden – wie in Abschnitt 3.3 vorgestellt – in diesem Kontext als gemeinsame Ressourcen betrachtet.

Hervorzuheben ist noch einmal, daß Fachliche Services *kein* Schichtenmodell umfassen: Es macht im Kontext Fachlicher Services keinen Sinn z.B. über einen Präsentations-Service zu sprechen.

Zur Kapselung der physikalischen Datenspeicherung werden im Rahmen des WAM-Ansatzes Metaphern wie die *Registratur* (ehemals „Archiv“) vorgeschlagen [vgl. Züllighoven et al. 98, S. 434ff.]. Eine solche Registratur ermöglicht die Ablage von Materialien und speichert diese intern persistent; sie kann als Fachlicher Service umgesetzt werden.

Business Objects

Ein Vergleich Fachlicher Services mit Business Objects wird dadurch erschwert, daß der Begriff Business Object nicht fest definiert ist, sondern bei verschiedenen Autoren für unterschiedliche Konzepte steht. Bei einer Umfrage von Hung et al. äußerten viele der Teilnehmer Unsicherheiten über das Konzept Business Objects. Als eine Schlußfolgerung der Umfrage stellen Hung et al. fest: »It seems that effort could be invested most productively in the definition of standardized Business Objects.«

[Hung, Simons & Rose 98, S.12]

Offen ist also eine Standardisierung und Festigung des Begriffs Business Object. Allgemein akzeptiert ist die generelle Überlegung, daß Business Objects eine Orientierung der Software an dem Anwendungsbereich einfordern und unterstützen:

»Paradoxically, objects have not been widely used to represent the business itself. [...] OMG Business Objects are representations of the nature and behavior of real world things or concepts in terms that are meaningful to the business.« [OMG 95, S. 4]

Es lassen sich unter anderem die folgenden drei Interpretationen von Business Objects finden, von denen die letzte im Zusammenhang mit Fachlichen Services besonders interessant ist.

- *Business Objects als Gegenstände des Anwendungsbereichs*

  „Some authors have used the term BO almost synonymously with "domain" object, meaning an object concept that maps onto a primary noun in the domain of discourse.“ [Hung, Simons & Rose 98, S. 2]

  In dieser Interpretation sind Business Objects vergleichbar mit Materialien des WAM-Ansatzes. Diese Materialien sind bei einer Softwareentwicklung diejenigen Objekte, die oft direkt aus dem Anwendungsbereich übernommen werden können.

»A business can be “modeled“ in terms of objects that make up and reflect it. Objects can represent inventory and invoices, customers, and salespeople. Objects can also represent events in a business, such as purchases, sales, and other types of transactions.« [OMG 95, S. 4]

Im Gegensatz zu dieser Vorstellung der OMG liegt der Fokus bei WAM aber noch stärker auf den tatsächlichen Gegenständen des Geschäftsbereichs, also z.B. auf Akten, Formularen und Verträgen. Anstatt den Kunden direkt zu modellieren, würde bei WAM eine Akte den Kunden repräsentieren.

- *Business Objects als vom Benutzer manipulierbare Gegenstände*

  Im Gegensatz zu Materialien sollen Business Objects bei manchen Autoren eine direkte Repräsentation auf dem Desktop haben:

  »Since business objects are visible to the "desktop," any program or user can access and safely manipulate the objects of the business.« [OMG 95, S. 14]

  Dieser Ansatz führt zu aktiven Objekten, die sich z.B. selbst drucken können. Problematisch an dieser Metapher ist, daß hier Funktion und Darstellung vermischt werden: So sind keine verschiedenartigen Benutzungsarten wie Desktop und Webtop auf Basis einer gemeinsamen Funktionalität möglich. Weiter erlaubt die direkte Manipulation keine komplexen Handhabungen und Werkzeuge.

- *Business Objects als größere Anwendungskomponenten*

  Interessant für Fachliche Services ist die Sicht auf Business Objects als größere Komponenten, die rund um eine Aufgabe des Anwendungsbereichs aufgebaut sind:

  »The above remarks tend to support the idea that a Business Object is a coarser-grained higher level object abstraction, which corresponds directly to some natural business concept, in contrast with finer-grained lower level software system components.« [Hung, Simons & Rose 98, S.2]

  Der Fokus wandert hier von einzelnen Objekten des Anwendungsbereichs zu einem Konzept oder Prozeß des Anwendungsbereichs. Dies beschreibt auch Jeff Sutherland:

  »Technically, business objects encapsulate traditional lower-level objects that implement a business process (i.e., they are a collection of lower-level objects that behave as single, reusable units). User interfaces can be thought of as views of large-grained Business Objects. Databases maintain a record of the "state" of Business Objects as they change over time.« [Sutherland 97, S. 2]

  Diese Business Objects sind Fachlichen Services auch darin ähnlich, daß sie eine reine Funktionalität anbieten, auf der dann die Benutzungsschnittstellen aufsetzen.

Am ehesten vergleichbar mit fachlichen Services ist also die letzte Interpretation von Business Objects als größere, am Anwendungsbereich orientierte Bausteine des Softwaresystems. Als Unterschiede heben Otto und Schuler hervor:

»Bei den Business Objects etwa stehen Gegenstände der Geschäftswelt oder Geschäftsprozesse [...] im Zentrum des Entwurfs. Dem steht bei den Fachlichen Services das Bild der Dienstleistung [...] gegenüber, als die fachliche Funktionalität angeboten wird. Wegen der gemeinsamen Kernidee sind beide Arten von Ansätzen aber trotz der genannten Differenzen in Bezug auf Anforderungen, Eigenschaften und Lösungsansätze sinnvoll und Erfolg versprechend vergleich- und übertragbar.«

[Otto & Schuler 2000, S. 47]

Fachliche Services müssen selbst kein Konzept des Anwendungsbereichs verkörpern, sie arbeiten aber auf den Gegenständen des Anwendungsbereichs und ihre Funktionalität und ihr Benutzungsmodell sind durch den Anwendungsbereich geprägt.

Zusammengefaßt ist das Konzept der Business Objects zu wenig vereinheitlicht und konkret, um eine allgemeine Abgrenzung zu Fachlichen Services vornehmen zu können. Einzelne Umsetzungen können aber durchaus mit Fachlichen Services vergleichbare Architekturen liefern.

Fazit

Fachliche Services bieten eine Metapher zur Strukturierung großer Anwendungssysteme, die gegenüber vergleichbaren Konzepten der Softwaretechnik einige Vorteile bringt: Sie fokussieren eine fachliche Modellierung, unterstützen verteilte Systeme und die Trennung von Interaktion und Funktion.

2.2 Telefonie und Callcenter

Am Bereich Arbeitsbereich Softwaretechnik der Universität Hamburg bestand eine langjährige Zusammenarbeit mit der Firma Micrologica AG aus Bargteheide, die mit ihrem Produkt *Micrologica Communication Center* (MCC) zu den Marktführern im Bereich Callcenter gehörte. Die Entwicklungsabteilung und das MCC wurden 2001 nach der Insolvenz der Micrologica AG von der Firma Tenovis Business Communication GmbH aufgekauft.

Das in dieser Arbeit vorgestellte Vorgehensmodell wurde in einem Projekt bei Tenovis erprobt. In den folgenden Kapiteln werden konkrete Anforderungen und Erfahrungen aus diesem Projekt vorgestellt. Dieses Kapitel bietet dazu eine Einführung in den Anwendungsbereich Telefonie und Callcenter und stellt das MCC vor.

Kommunikationsunterstützung

Im Rahmen dieser Arbeit werden technische Kommunikationsmedien betrachtet, die eine Anbindung an Software ermöglichen.

> **Definition: Kommunikationsunterstützung**
> Unter Kommunikationsunterstützung wird hier die komfortable Unterstützung menschlicher Kommunikation durch Hardware und Software verstanden. Diese Unterstützung geht über die reinen Kommunikationsmedien wie Telefone, Fax oder E-Mail hinaus, sie erweitert diese Medien um Komfortmerkmale wie Anruferidentifizierung, automatische Anrufverteilung oder Integration mit Bürosoftware.

Obschon aktuelle Software zur Kommunikationsunterstützung eine Vielfalt an Kommunikationsmedien unterstützt, z.B. neben Telefonie auch das Verteilen von E-Mails, beschränkt sich diese Arbeit im folgenden auf die Telefonie, die immer noch im Zentrum der Kommunikationsunterstützung steht. Dies dient dazu, den Anwendungsbereich möglichst überschaubar zu halten und Formulierungen zu vereinfachen und reicht für diese Arbeit aus.

Da in allen Branchen Kommunikationsmedien genutzt werden, kann prinzipiell auch jede Firma von einer Kommunikationsunterstützung profitieren. Durch diese Bandbreite ergeben sich sehr unterschiedliche Anforderungen an die Unterstützung. Insbesondere ist auch innerhalb einer Firma an den unterschiedlichen Arbeitsplätzen ein unterschiedlicher Bedarf gegeben.

Im folgenden werden die zwei *Arbeitsplatztypen* Call-Center-Mitarbeiter und Vertriebsmitarbeiter vorgestellt, um so einen Eindruck von den Anforderungen an Kommunikationsunterstützung zu geben. Diese Typen unterscheiden sich in dem Anteil der Telefonie an der täglichen Arbeit: ob die Arbeit hauptsächlich aus solcher Kommunikation besteht oder ob das Telefon nur ein Hilfsmittel neben anderen ist.

Arbeitsplatztypen fassen gleichartige Arbeitsplätze zusammen; jeder Arbeitsplatztyp wird kurz mit seinen Tätigkeiten und Anforderungen charakterisiert. Dazu werden die *Aufgaben,* die an diesem Arbeitsplatztyp zu erledigen sind, in *Aufgabenszenarien* vorgestellt.

»Der Szenario-Subtyp Aufgabenszenario beschreibt, was eine einzelne Aufgabe ausmacht und wie eine einzelne Aufgabe insgesamt erledigt wird. Diese Beschreibung hat den Charakter eines »Drehbuchs« oder einer kleinen szenischen Schilderung. Alternativen an einzelnen Entscheidungspunkten der Handlung werden zwar erfaßt, aber möglichst sparsam.« [Züllighoven et al. 98, S. 612]

Diese Szenarien geben einen exemplarischen Einblick in die Arbeit an diesem Arbeitsplatz, aus denen sich dann die Anforderungen an die Kommunikationsunterstützung ergeben.

Callcenter-Mitarbeiter

Arbeitsplätze mit überwiegendem Telefonieanteil sind typischerweise in einem Callcenter angesiedelt. Ein Callcenter besteht aus einer Gruppe von Mitarbeitern, deren überwiegende Tätigkeit aus Telefonieren besteht. Die Gruppe kann an einem Ort oder auf mehrere Standorte verteilt arbeiten. Eine häufige Aufgabe eines Callcenters ist, die Firma von Routine-Anrufen zu entlasten.

Arbeitsplatztyp: Callcenter-Mitarbeiter
Die Arbeit ist geprägt von einer hohen Zahl an Telefonaten mit Kunden. Häufige Kundenfragen werden mit Hilfe von vorgegebenen Antworten und Skripten beantwortet. Es werden nur mäßige fachliche Kenntnisse benötigt, da die meisten Antworten dem vorgegebenen Schema folgen und Routine sind.

Hier finden sich die Aufgaben „Arbeit in einer Hotline“, „Arbeit in der Telefonzentrale“, „Mitarbeit in einer Telefonkampagne“.

Arbeit in einer Hotline

Eine Hotline ist eine zentrale Sammelrufnummer, unter der Firmenkunden anrufen können. Für die Beantwortung der Anrufe steht eine große Zahl an Mitarbeitern bereit. Diese sind für die Beantwortung häufig auftretender Fragen ausgebildet.

Da in der Hotline Anrufe von Kunden eingehen, spricht man vom *Inbound*-Bereich. Durch den Anruf bei der Sammelrufnummer drückt der Kunde aus, welchen Service er nutzen möchte, deshalb bezeichnet man diese Nummer auch als *Service Entry Point* (SEP).

Unter einem Service versteht man im Telefoniebereich eine Dienstleistung, die ein Kunde nutzen kann. Ein solcher Service kann z.B. eine Beratung sein. Diese wird von einer Gruppe von Mitarbeitern erbracht, die dazu qualifiziert ist.

Aufgabenszenario: Arbeit für eine Firmen-Hotline
Die Anrufe bei dieser Hotline werden von den 20 Mitarbeiterinnen und Mitarbeitern eines Callcenters beantwortet. Eine Hotline-Mitarbeiterin meldet sich zu Beginn ihrer Arbeit an ihrem Arbeitsplatz in ihrem *Agentenmonitor* an und signalisiert dem MCC, daß sie jetzt Anrufe entgegen nehmen kann.

Ein Kunde hat eine Frage zu einem Produkt und ruft die Hotline der Firma an. Der Kunde hört eine Wartefeldansage, bis ein Platz frei wird und er mit der

Hotline-Mitarbeiterin verbunden wird. Durch die Anrufer-Erkennung wurde der Kunde bereits identifiziert, die Mitarbeiterin sieht auf ihrem Bildschirm im *First-Screen* alle über den Kunden verfügbaren Daten. Da das Callcenter für mehrere Firmen arbeitet, bekommt sie auch angezeigt, mit welchem Firmennamen sie sich für diese Hotline melden soll.

Der Kunde stellt seine Frage. Die Mitarbeiterin hat ein Skript, das ihr Richtlinien für den Umgang mit den verschiedenen Fragen vorgibt. Direkt aus der Anzeige der Kundendaten kann sie auf die Informationen über die Produkte zugreifen und beantwortet die Frage. Nach dem Gespräch ergänzt sie die Kundendaten um eine kurze Notiz über den Inhalt des Gesprächs. Dann signalisiert sie dem System, daß sie wieder für neue Anrufe bereit ist.

In dem Fall einer ungewöhnlichen Frage, die nicht routinemäßig beantwortet werden kann, wird der Kunde mit einem Mitarbeiter mit detaillierteren Kenntnissen verbunden. In diesem Zusammenhang spricht man auch von der Hotline als *First Level* Support und dem höher qualifizierten Mitarbeiter als *Second Level Support*.

In dem Szenario wurden schon typische Mittel zur Kommunikationsunterstützung genannt:

- Die Callcenter-Mitarbeiterin kann im *Agentenmonitor* signalisieren, ob sie Anrufe bekommen möchte.
- Es gibt Wartefelder mit Ansage.
- Anrufe werden auf freie Plätze verteilt.
- Der Kunde wird automatisch identifiziert.
- Es werden automatisch im *First-Screen* die Kundendaten angezeigt.
- Aus der Anzeige der Kundendaten lassen sich auch andere Informationen abrufen.

Mitarbeiter in einer Telefonkampagne

Bei Telefonkampagnen wird eine Adressenliste mit sehr vielen Kunden angerufen. Dabei handelt es sich z.B. um Umfragen oder Informationen über neue Produkte. Die Adressen der Kunden können z.B. aus Werbeaktionen stammen. Geht der Kunde an das Telefon, wird mit ihm ein Informations- und Verkaufsgespräch geführt.

Da die Kunden von der Firma angerufen werden, spricht man vom *Outbound*-Bereich.

Aufgabenszenario: Anruf im Rahmen einer Telefonkampagne
Von der Firma wurde ein Gewinnspiel durchgeführt. Dabei hat ein Teil der Kunden angekreuzt, daß weitere Informationen über das Produkt erwünscht sind. Diese Kunden sollen jetzt angerufen werden.

Dazu wird von dem System die Kundenliste abgearbeitet, es werden Anrufe zu den Kunden generiert. Es werden mehr Anrufe generiert als Mitarbeiter frei sind, da viele Kunden nicht an das Telefon gehen werden. Das bezeichnet man als *Overdialling*.

Ein Kunde ist Zuhause und meldet sich. Er hört eine kurze Ansage und wird dann sofort mit einem freien Mitarbeiter verbunden. Der Mitarbeiter bekommt

die Daten des Kunden angezeigt und führt mit ihm ein Informationsgespräch über das Produkt. Es werden weitere Fragen des Kunden beantwortet. Ziel ist es, möglichst viele Kunden für das neue Produkt zu gewinnen.

In diesem Szenario gibt es folgende typische Mittel zur Kommunikationsunterstützung:

- Anruflisten werden von dem System abgearbeitet, es werden mehr Leute angerufen als Mitarbeiter zur Verfügung stehen, da nicht alle ans Telefon gehen werden (Overdialling).
- Auch hier kann der Mitarbeiter signalisieren, ob er frei ist, es gibt Wartefelder mit Ansage, Anrufe werden auf freie Plätze verteilt, es werden automatisch die Kundendaten angezeigt

Während beim Arbeitsplatztyp *Callcenter–Mitarbeiter* die Telefonie den überwiegenden Teil der täglichen Arbeit ausmacht, ist beim folgenden Arbeitsplatztyp das Telefonieren nur eine wichtige Tätigkeit neben anderen.

Vertrieb und Kundenbetreung

Ein Vertriebsmitarbeiter betreut einen Kunden, zum Beispiel während der Laufzeit eines Projektes. Er berät den Kunden, arbeitet für ihn Konzepte aus und sorgt für die Verwirklichung der Konzepte durch die eigene Firma. Er entwickelt die Produkte nicht selber, hat aber genügend Fachkenntnisse für eine Beratung.

Arbeitsplatztyp Vertriebsmitarbeiter Kundenbetreuung
Der Vertriebsmitarbeiter berät Kunden und fertigt für diese Konzepte und Aufstellungen der Kosten an. Für diese Arbeit hält er Kontakt zu vielen Personen und ist viel unterwegs.

Aufgabenszenario: Kundenbetreung
Ein Kunde benutzt bereits in einer Abteilung ein Produkt der Firma und will jetzt auch andere Abteilungen unterstützen. Der Vertriebsmitarbeiter trifft sich mit dem Kunden und bespricht dessen Wünsche und mögliche Umsetzungen. Nach dem Gespräch klärt er offene Fragen mit der Entwicklungsabteilung, erstellt ein ausführliches Konzept und ermittelt die Kosten.

Für viele dieser Handlungen benutzt er das Telefon, zum Beispiel um Termine zu vereinbaren und Rückfragen zu stellen. Wichtig für die Kundenbetreuung ist, daß der Vertriebsmitarbeiter für den Kunden gut erreichbar ist. Dazu wird seine Büronummer immer auf seinen aktuellen Aufenthaltsort oder sein Handy umgeleitet.

Bittet der Kunde um einen Rückruf am folgenden Tag, kann der Vertriebsmitarbeiter diesen Anruf im MCC vormerken und von System automatisch auslösen lassen.

Typische Mittel zur Kommunikationsunterstützung sind also:
- Der Anruf wird auf den aktuellen Aufenthaltsort weitergeleitet.
- Der Mitarbeiter muß auch ohne Computer das System steuern können und dem System seinen Aufenthaltsort signalisieren.

- Auch hier kann der Mitarbeiter mitteilen, ob Anrufe gerade erwünscht sind.
- Steht ein Arbeitsplatzrechner zur Verfügung, wird der Kunde automatisch identifiziert, es werden die Kundendaten angezeigt.
- Rückrufe können vorgemerkt werden und werden vom MCC automatisch erzeugt.

Kommunikationsunterstützung

Diese Abschnitt faßt noch einmal zusammen, welche Mittel zur Kommunikationsunterstützung gefunden wurden. Diese unterstützen die eben vorgestellten Arbeitsplätze bei ihren Aufgaben.

- *Agentenanmeldung und Signalisierung des Zustands*
  Der Mitarbeiter meldet sich im System an und kann dann Anrufe bekommen. Das MCC überwacht über die Telefonanlage sein Telefon und die Service Entry Points. Der Mitarbeiter kann mitteilen, ob er gerade Anrufe bekommen möchte. Mit einem Icon wird ihm dargestellt, ob er gerade angemeldet ist, telefoniert oder Pause macht. Diese *Agentenmonitor*-Funktionalität wird an allen Arbeitsplätzen ähnlich benutzt.
- *Automatische Kundenidentifizierung und Integration der Kundendaten*
  Die Rufnummer des Kunden wird automatisch identifiziert, z.B. durch die Telekom, eine persönliche Nachwahl oder eine Identifizierung im Wartefeld.

  Ist der Kunde identifiziert, können automatisch die entsprechenden Kundendaten gesucht und im *First-Screen* angezeigt werden. Dies sind z.B. der Name, die Adresse, die letzten Aufträge, der Rechnungsstand und ähnliches. Von diesem First-Screen aus starten arbeitsplatzspezifische Arbeitsabläufe und Werkzeuge.

Weitere Kommunikationsunterstützungen, die im folgenden nur eine untergeordnete Rolle spielen, sind

- *Ortsunabhängigkeit und virtuelle Callcenter*
  Mitarbeiter eines Callcenters können an beliebigen Orten sitzen.
- *Sammelrufnummern und automatische Anrufverteilung*
  Eine Dienstleistung wird hier von mehreren gleich qualifizierten Mitarbeitern erbracht. Es ist für den Kunden unerheblich, mit welchem dieser Mitarbeiter er verbunden wird. Ruft ein Kunde eine Sammelrufnummer an, bestimmt das MCC einen Mitarbeiter, der das Gespräch bekommt. Sind alle Mitarbeiter belegt, landet der Anrufer im Wartefeld.
- *Unterstützung von Telefonkampagnen*
  Adressenlisten werden angerufen. Wenn ein Kunde erreicht wird, übernimmt ein Mitarbeiter das Gespräch.

Fazit

Der Anwendungsbereich Telefonie berührt viele Bereiche des Arbeitslebens, entsprechend umfangreich sind auch die vorgestellten Möglichkeiten zur Kommunikationsunterstützung. Im Zentrum der folgenden Kapitel stehen die Agentensteuerung, die Kundenidentifikation und die Integration existierender Kundendaten.

3 Anbindung existierender Anwendungssysteme

Dieses Kapitel entwickelt auf der Basis softwaretechnischer Überlegungen ein Vorgehensmodell zur Anbindung existierender Anwendungssysteme. Eine solche Anbindung und Kapselung schafft eine technisch moderne Schnittstelle zu einer existierenden Funktionalität. Die neue Schnittstelle orientiert sich dabei an den aktuellen Anforderungen und Aufgaben. Es wird nicht unbedingt die gesamte Funktionalität des existierenden Systems angeboten, sondern eine fachlich motivierte Auswahl.

Ein darauffolgender zweiter Schritt kann die Migration des existierenden Anwendungssystems zu einen neu entwickelten System sein; eine solche Migration würde – für den Konsumenten unbemerkt – hinter der neu geschaffenen Fassade stattfinden. Diese Arbeit beschränkt sich auf den ersten Schritt, die Kapselung. Eine Übersicht über die verschiedenen Ansätze und Forschungsergebnisse zur darauf folgenden Migration geben Bisbal et al. in ihrer Arbeit „A Survey of Research into Legacy System Migration" [vgl. Bisbal 97].

Ein Grund dafür, die Migration aus dieser Arbeit auszuklammern, ist das im praktischen Teil untersuchte Anwendungssystem, das kein Altsystem, sondern ein modernes Client-Server-System ist. Dabei charakterisieren Bisbal et al. Altsysteme (legacy information systems) wie folgt:

- »[...] these systems run on obsolete hardware which is slow and expensive to maintain;
- maintenance of software is generally expensive; tracing faults is costly and time consuming due to the lack of documentation and a general lack of understanding of the internal workings of the system;
- integration with other systems is greatly hampered by the absence of clean interfaces;
- evolution of legacy systems to provide new functionality required to provide new functionality required by the organisation is virtually impossible.« [Bisbal 97, S.2]

Da das untersuchte Anwendungssystem nicht auf solch veralteter Technologie aufsetzt, ist eine Migration bzw. Ablösung durch ein neues System nicht geplant.

Trotzdem ist eine Kapselung des untersuchten Anwendungssystems sinnvoll und ähnelt der Kapselung eines Altsystems: Die existierenden Schnittstellen sind uneinheitlich, überfrachtet und schlecht dokumentiert, sie wurden von verschiedenen Mitarbeitern zu verschiedenen Zeiten entwickelt. Darüber hinaus setzen sie auf einer proprietären Technologie auf, die in aktuellen Projekten verborgen werden soll. Schließlich bestand wie bei einem Altsystem die Auflage, daß das existierende System nicht geändert werden durfte.

Kapselung mit Fachlichen Services

Im Rahmen dieser Arbeit wurden Fachlichen Services als Entwurfsmetapher benutzt, um die Funktionalität existierender Anwendungssysteme in komfortabler Form anzubieten. Fachliche Services eignen sich besonders gut zur Konstruktion solcher Schnittstellen, da sie eine fachliche Modellierung fokussieren, eine gute Unterstützung von Verteilung bieten und sich auf die Funktionalität beschränken.

Kapseln Fachliche Service ein existierendes Anwendungssystem, setzt ihre Dienstleistung auf der existierenden Funktionalität auf. Eine solche Kapselung wird auch von der OMG für Business Objects vorgeschlagen:

»A business object “wrapper” is written in the language of the existing ... [system] using a business object framework. The relationships, rules, and procedures for using the object are implemented as part of the business object using the existing libraries and methods of the legacy application.« [OMG 95, S. 8]

Für diese Kapselung wurde ein Vorgehensmodell entwickelt, das im folgenden Abschnitt eingeführt wird.

3.1 Technische Benutzungsmodelle

Softwaretechnische Grundlage für das in dieser Arbeit vorgestellte Vorgehensmodell sind Überlegungen zu den *Benutzungsmodellen* von existierendem Anwendungssystem und neuer Schnittstelle. Ein Benutzungsmodell beschreibt den Umgang mit einem Programm:

»Ein Benutzungsmodell ist ein fachlich orientiertes Modell darüber, wie Anwendungssoftware bei der Erledigung der anstehenden Aufgaben im jeweiligen Einsatzkontext benutzt werden kann. Das Benutzungsmodell umfaßt eine Vorstellung von der Handhabung und Präsentation der Software, aber auch von fachlichen Gegenständen, Konzepten und Abläufen, die von der Software unterstützt werden.

Es ist sinnvoll, ein Benutzungsmodell auf der Grundlage eines Leitbilds mit Entwurfsmetaphern zu realisieren.« [Züllighoven et al. 98, Kapitel 2.2]

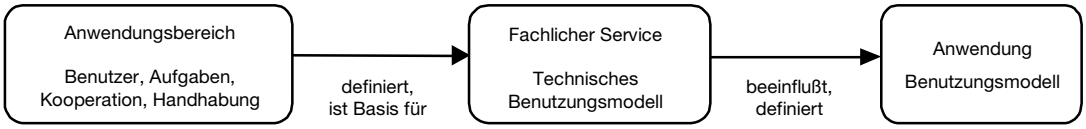
Da ein Fachlicher Service nicht direkt von Anwendern benutzt wird und auch kein *Graphical User Interface* (GUI) besitzt, wird im Folgenden für softwaretechnische Komponenten der Begriff „Technisches Benutzungsmodell“ benutzt.

Das technische Benutzungsmodell eines Fachlichen Services umfaßt eine Vorstellung
- von den Abläufen und Aufgaben, die von dem Service unterstützt werden,
- von den Konzepten und Gegenständen, mit denen der Service arbeitet,
- vom Umgang mit Verteilung und Kooperation,
- vom generellen Umgang und den Eigenschaften des Service.

Definition: Technisches Benutzungsmodell
Ein technisches Benutzungsmodell beschreibt den Umgang mit einer softwaretechnischen Komponenten, z.B. einem Fachlichem Service. Es umfaßt eine Vorstellung von der Dienstleistung, den fachlichen Gegenständen, Konzepten und Abläufen, die von der Komponente unterstützt werden. Insbesondere umfaßt das technische Benutzungsmodell eine explizite Vorstellung der Kooperation zwischen verschiedenen Konsumenten der Dienstleitung.

Auch wenn der Fachliche Service nur von weiteren Softwarekomponenten direkt genutzt wird, hat sein technisches Benutzungsmodell durchaus Auswirkungen auf die Benutzung der Anwendung durch den Endanwender: Eine Handhabung, die von dem vom Service vorgegebenen Benutzungsmodell abweicht, läßt sich häufig nur unter größeren Schwierigkeiten implementieren. Bietet der Fachliche Service zum Beispiel keine Möglichkeit für Konsumenten, sich als Beobachter für Änderungen anzumelden, werden die Konsumenten nur mit Schwierigkeiten ein GUI umsetzen können, in dem Änderungen anderer Benutzer sofort angezeigt werden.

Damit also die Benutzung der Anwendung aufgabenangemessen möglich ist, muß auch das technische Benutzungsmodell des Fachlichen Service durch den Anwendungsbereich motiviert sein.



Ein Teil des technischen Benutzungsmodell ist das Kooperationsmodell, also die Vorstellung, wie die verschiedenen Benutzer kooperieren. Das Kooperationsmodell ist bei der Anbindung großer Anwendungssysteme besonders wichtig, weil hier in der Regel viele Interaktionen zwischen den Anwendern stattfinden.

Kooperationsmodell

Arbeiten mehrere Personen zusammen, um eine Aufgabe zu erfüllen, handelt es sich um kooperative Arbeit:

»Bei kooperativer Arbeit arbeiten verschiedene Personen geplant und koordiniert zusammen, um ein gemeinsames Ergebnis zu erreichen.« [Züllighoven et al. 98, S. 428]

Die Personen müssen sich dabei nicht an einem Ort befinden, sie können räumlich getrennt arbeiten. Die Kommunikation kann z.B. über Briefe, Telefon oder Computersysteme erfolgen. Teilweise wird kooperative Arbeit durch diese Kommunikationswege erst möglich. Im weiteren Sinne handelt es sich auch um kooperative Arbeit, wenn die Personen nicht explizit miteinander kommunizieren, sondern implizit auf geteilten Materialien arbeiten, z.B. mit den selben Vorgangsakten.

Die Teile eines Anwendungssystems, die von mehreren Anwendern benutzt werden und der Kooperation dieser Anwender dienen, können als eine *gemeinsame Ressource* verstanden werden.

> **»Definition: Gemeinsame Ressource**
> Eine gemeinsame Ressource kann von mehreren Arbeitsumgebungen aus zugegriffen werden. Fachliche gemeinsame Ressourcen setzen einen Teil des Benutzungsmodells der sie verwendenden Applikation um. Technische gemeinsame Ressourcen bleiben ohne Einfluß auf das Benutzungsmodell einer Anwendung.«
> [Otto & Schuler 2000, S. 64]

Fachliche gemeinsame Ressourcen können die Umsetzung einer WAM-Entwurfsmetapher wie der *Registratur* sein, in der Anwender Dokumente ablegen und ausleihen können, oder ein zentraler Nachrichten- und Postdienst. Technische gemeinsame Ressourcen sind zum Beispiel Datenbanken.

»Fachliche Services sind gemeinsame Ressourcen in WAM-Umgebungen. Sie stehen dem Benutzer nicht direkt zur Verfügung – die Leistung der Services wird dem Benutzer über geeignete Werkzeuge oder Automaten zur Verfügung gestellt, welche die Services ihrerseits benutzen.

Fachliche Services, so wie wir sie später einführen, sind immer fachliche gemeinsame Ressourcen, sei es dadurch, dass sie Gegenstände der Anwendung und deren Umgang als Dienstleistung bereitstellen oder dadurch, dass sie Vorgänge der Anwendungswelt vergegenständlichen.« [Otto & Schuler 2000, S. 64]

Fachliche Services sind gemeinsame Ressourcen
Fachliche Services sind eine gemeinsame fachliche Ressource in WAM-Umgebungen. Fachliche Services unterstützen die Kooperation der Benutzer und setzen einen Teil des Benutzungsmodells der Benutzer um.

Offen bleibt, ob Fachliche Services in der Anwendung direkt dem Benutzer als notwendige Ressourcen signalisiert werden. Vorteil einer solchen Handhabung ist die einleuchtende Signalisierung von Störungen. Offen bleibt auch die Untersuchung von Services, die konzeptionell für eine rein lokale Nutzung gedacht sind. Solche Services könnten z.B. eine komplexe Dienstleistung anbieten oder auf einem Laptop einen gemeinsam genutzten Service ersetzen.

Das Kooperationsmodell eines Fachlichen Services

Soll ein zentraler Fachlicher Service verteilte kooperative Arbeit unterstützen, umfaßt das technische Benutzungsmodell auch eine Vorstellung, wie verteilt gearbeitet wird. Dieses *Kooperationsmodell* leitet sich aus den fachlichen Umgangsformen im Anwendungsbereich her: Hier finden sich Besonderheiten, die sich aus der Verteilung ergeben. Diese Besonderheiten werden in der Regel auch den Service beeinflussen, die Umgangsformen lassen sich teilweise direkt auf den Service übertragen.

»Definition: Kooperationsmodell
Ein Kooperationsmodell ist ein Modell, das entweder explizit oder implizit die Zusammenarbeit (Kooperation) beschreibt und regelt.

Wir betrachten hier Kooperationsmodelle, die Bestandteil des Arbeitsplatzsystems sind.« [Züllighoven et al. 98, S. 427]

Ein explizites Kooperationsmodell benutzt im Computersystem Metaphern wie die Registratur oder Postkörbe und unterstützt damit komplexe Kooperationsvorgänge. Ein implizites Kooperationsmodell wird im Computersystem nur in Ausnahmefällen erkennbar, z.B. wenn mehrere Benutzer gleichzeitig auf ein Dokument zugreifen möchten.

Kooperationsmodelle führen einen Ortsbegriff ein: An welchen Orten befinden sich die Gegenstände? Diese Orte sind meist fachlich interpretierbar und werden oft aus dem Anwendungsbereich übernommen. Beispiele für Orte sind: „Beim Sachbearbeiter“, „Im zentralen Archiv“.

Entsprechende Fragen bei der Gestaltung des Fachlichen Services sind:
- Welche Gegenstände befinden sich an welchem Ort?
- Welche Materialien und Daten werden lokal, welche zentral gehalten?
- Welchen Zugriff auf die Materialien und Daten haben die einzelnen Benutzer? Bekommen sie Originale oder Kopien?
- Wie werden Materialien zwischen den Benutzern weitergereicht?

Ein Beispiel für eine Form der expliziten Kooperation sind die von Guido Gryczan beschriebenen Kommunikationsmittel [vgl. Gryczan 96]. Diese leiten sich ab von dem Konzept der Vorgangsmappen mit Laufzettel. Dokumente, die zu einem Vorgang gehören, werden in einer Mappe zusammengefaßt und zwischen Sachbearbeitern wei-

tergereicht. Welcher Sachbearbeiter die Dokumente als nächstes bekommt, wird auf einem Laufzettel eingetragen, der von den Sachbearbeitern jederzeit geändert werden kann.

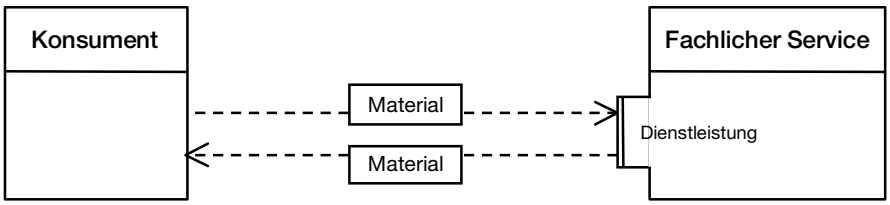
Grafische Notation technischer Benutzungsmodelle

Um einen Überblick über die Dienstleistungen und Umgangsformen eines Fachlichen Services zu bekommen, ist eine grafische Notation des technischen Benutzungsmodells wünschenswert.

Hierzu sind Dokumenttypen von WAM [Züllighoven 98, S. 601ff] und aus der UML [Fowler 99] nur eingeschränkt geeignet: Klassendiagramme zeigen für einen Überblick zu viele Details, insbesondere technische Details der Implementierung. Interaktionsdiagramme zeigen keinen Überblick, sondern eine konkrete komplexe Interaktion. Komponentendiagramme zeigen die Architektur, also die technische Aufteilung, aber nicht die Benutzung der Komponente. Szenarien, Use-Cases und Kooperationsbilder hingegen richten sich an die Anwender und zeigen nur die Benutzungsschnittstellen, aber nicht den inneren Aufbau des Anwendungssystems.

Deshalb wird hier eine neue Notation für technische Benutzungsmodelle eingeführt, in der Benutzungskonzepte direkt darstellbar sind. Diese Notation bietet einen allgemein gehaltenen Überblick über den Umgang mit einer technischen Komponente und abstrahiert dabei von der Implementierung. Der Überblick zeigt alle Dienstleistungen der Komponente, beschränkt sich also nicht etwa auf eine Aufgabe.

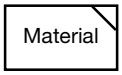
Die folgende Darstellung zeigt die Anforderung einer Dienstleistung durch einen Konsumenten und die Antwort des Fachlichen Service.



Jede angebotene Dienstleistung wird beim Fachlichen Service mit einer Erhöhung dargestellt. An die Pfeile, die wie bei Kooperationsbildern die Richtung der Interaktion zeigen, werden ggf. die ausgetauschten Materialien gezeichnet. Funktionsparameter, die nicht als Materialien interpretierbar sind, werden nicht dargestellt. Wird die Dienstleistung vom Fachlichen Service stillschweigend, d.h. ohne eine explizite Antwort ausgeführt, kann der Rückpfeil auch weggelassen werden.

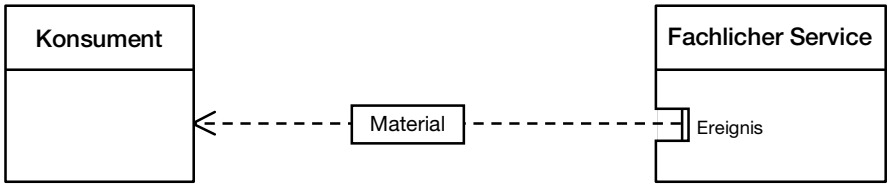
In dieser Arbeit liegt der Schwerpunkt auf der Darstellung der Dienstleistungen der Fachlichen Services, deshalb handelt es sich bei den Konsumenten immer um abstrakte Rollen. Grundsätzlich sind aber auch Bilder denkbar, in denen gezeigt wird, welche Dienstleistungen ein konkreter Konsument – z.B. ein anderer fachlicher Service – benutzt.

Als Kopie übergebenes Material wird mit einer durchgestrichenen Ecke markiert:



Die in dieser Arbeit untersuchten Fachliche Services bieten die Möglichkeit, sich als Beobachter für Ereignisse zu registrieren. Es handelt sich dabei um Ereignisse aus dem fachlichen Anwendungsbereich. Im Gegensatz zu normalen Antworten treten Ereignisse zu einem unvorhersehbaren Zeitpunkt ein.

Kann ein Konsument sich für ein Ereignis anmelden, wird dieses beim Fachlichen Services mit einer Vertiefung dargestellt:



Die nötige zusätzliche Interaktion zum An- und Abmelden wird dabei als Implementationsdetail betrachtet, von dem im technischen Benutzungsmodell abstrahiert wird.

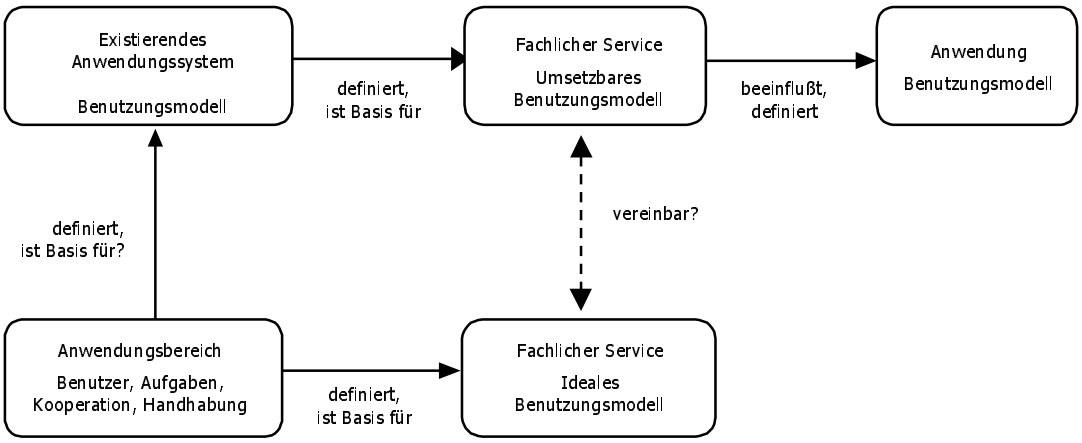
Diese Notation wird in dieser Arbeit für die vorgestellten Benutzungsmodelle verwendet. Am praktischen Projekt werden noch einmal die Unterschiede zu Interaktionsdiagrammen (S. 43) und Klassendiagrammen (S. 75) gezeigt.

Fazit

Technische Benutzungsmodelle beschreiben den Umgang mit einer technischen Komponente. Für sie wurde in dieser Arbeit erstmals eine spezielle Notation eingeführt. Überlegungen zu diesen Benutzungsmodellen stellen im folgenden die softwaretechnische Motivation dar für das Vorgehensmodell zur Kapselung existierender Anwendungssysteme.

3.2 Vorgehensmodell zur Anbindung existierender Anwendungssysteme

Soll ein Fachlicher Service seine Funktionalität auf der Basis eines existierenden Anwendungssystems anbieten, wird sein technisches Benutzungsmodell durch das existierende Benutzungsmodell geprägt:



Ausgehend von den Anforderungen und Aufgaben der Benutzer ist ein ideales Benutzungsmodell denkbar, das diese Anforderungen erfüllt und dabei einen einheitlichen und fachlich verständlichen Umgang bietet.

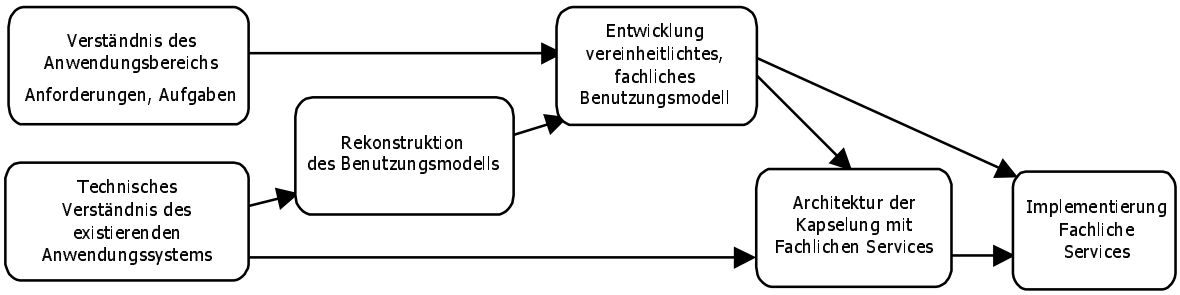
Dem gegenüber ist es unklar, ob das Benutzungsmodell des existierenden Anwendungssystems noch die aktuellen Anforderungen erfüllt: Oft ändern sich Aufgaben und Schwerpunkte und Technologien veralten. Darüber hinaus bieten existierende Anwendungssysteme durch ihre andauernde Weiterentwicklung oft eine uneinheitliche Schnittstelle.

Kritische Frage bei der Kapselung ist deshalb, welches technische Benutzungsmodell sich auf der Basis des existierenden Anwendungssystems umsetzen läßt und ob dieses mit den aktuellen Anforderungen sowie den Konzepten des idealen Benutzungsmodells vereinbar ist.

Ein Vorgehensmodell zur Kapselung muß also den Zwiespalt zwischen aktuellen Anforderungen und den Randbedingungen aus dem existierenden Anwendungssystem berücksichtigen.

Das Vorgehensmodell

Mit diesen Überlegungen wurde ein Vorgehensmodell entwickelt, das in der folgenden Zeichnung dargestellt wird:



Dabei sind die einzelnen Schritte nicht als getrennte, streng aufeinander folgende Abschnitte der Entwicklung zu verstehen, sondern als logische Phasen, die ggf. auch parallel oder in einem iterativen Prozeß ausgeführt werden können. Die Richtung der Pfeile gibt an, daß in der rechten Phase in der Regel ein Ergebnis oder Verständnis aus der linken Phase benutzt wird.

Am Anfang des Vorgehensmodells steht das Verständnis des Anwendungsbereichs und der Anforderungen an die Fachlichen Services. Diese aufgabenorientierte Analyse wird sinnvollerweise zusammen mit Nutzern der zukünftigen Schnittstelle durchgeführt. Ziel soll es sein, die zentralen und häufigsten Anforderungen in der Sprache der Nutzer festzuhalten. Eine solche Orientierung an den aktuellen Anforderungen der Konsumenten wird auch von Allen und Frost gefordert:

»The temptation is often to wrap legacy assets on the basis that it “seemed like a good idea at the time.” Business-oriented component modeling helps ensure that the correct business requirements are addressed as a foundation for wrapping.«

[Allen Frost 98, S. 211]

Sie heben dabei hervor, daß die aktuellen Anforderungen keineswegs mit dem Benutzungsmodell des existierenden Anwendungssystems übereinstimmen müssen.

Parallel zu der Analyse der Anforderungen ist ein technisches Verständnis des existierenden Anwendungssystems notwendig. Dieses beginnt mit einem Einblick in die Grundkonzepte, Benutzung und Administration des Systems. Für ein tieferes Verständnis sind dann eine Analyse der Schnittstellen und – je nach Qualität der Dokumentation – auch der Quelltexte notwendig.

Aus dem technischen Verständnis erwächst eine Vorstellung von dem existierenden Benutzungsmodell. Dieses kann uneinheitlich sein, es kann Spezialfälle und Ausnahmen von der Regel geben.

An diesem Punkt werden das existierende Benutzungsmodell und die Anforderungen gegenübergestellt, das Ziel ist eine überschaubare und einheitlichen Schnittstelle, die die Anforderungen der Benutzer erfüllt. Dazu wird das existierende Benutzungsmodell vereinheitlicht und fachlich strukturiert und es wird aufgabenorientiert eine Auswahl

aus der vom existierenden System angebotenen Funktionalität getroffen. Häufige Aufgaben werden im vereinheitlichten Benutzungsmodell komfortabel unterstützt.

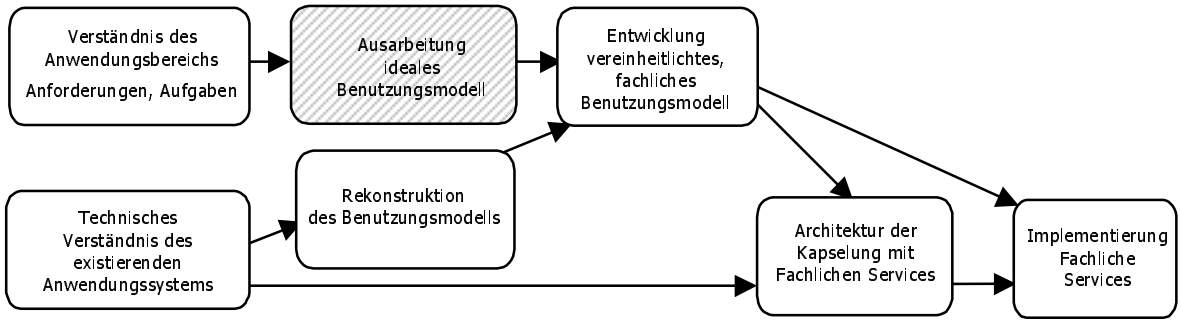
Kritisch ist die Frage, ob das Benutzungsmodell und die Funktionalität des existierenden Systems tragfähig genug sind: Ein existierendes Anwendungssystem läßt sich erfolgreich mit fachlichen Services kapseln, wenn sich ein solches einheitliches Benutzungsmodell finden läßt, das die Aufgaben und Kooperation der Benutzer unterstützt und widerspiegelt und sich gleichzeitig mit der vorhandenen Funktionalität umsetzen läßt. Je weiter das existierende Benutzungsmodell und das durch die Anforderungen bestimmte ideale Benutzungsmodell auseinandergehen, um so größer wird der Aufwand und die Komplexität, um ein angemessenes Benutzungsmodell noch umzusetzen.

Schließlich wird – aufgrund des technischen Verständnisses des existierenden Anwendungssystems und der Anforderungen der Konsumenten – eine Architektur der Kapselung entwickelt. Diese Architektur umfaßt oft Verteilungsaspekte und wird damit relativ komplex.

Mit diesen Ergebnissen können die Fachlichen Services implementiert werden. Dabei sind einige technische Fragen zu beachten, die aus der Komplexität der Anwendungssysteme und der Verteilung resultieren.

Neuentwicklung und Migration

Das vorgestellte Vorgehensmodell verzichtet darauf, ein ideales Benutzungsmodell explizit auszuarbeiten. Dies wäre parallel zur Rekonstruktion des existierenden Benutzungsmodells möglich:



Ein solches ideales Benutzungsmodell basiert allein auf den Anforderungen der Konsumenten, seine Ausarbeitung ähnelt der Neuentwicklung des Anwendungssystems von Grund auf. Im Rahmen einer Migration entspricht das ideale Benutzungsmodell der technischen Systemvision des zukünftigen, neu zu entwickelnden Anwendungssystems.

Die Entwicklung des idealen Benutzungsmodells ist nicht einfach: Sie ist – wie jede Neuentwicklung – alleine schon infolge der Komplexität des Anwendungssystems sehr aufwendig. Kennen die Entwickler bereits das existierende Anwendungssystem, sind ihre Vorstellungen von diesem geprägt. Kennen sie es nicht, können sie nicht aus den bisherigen Erfahrungen lernen. Es ist am Anfang unklar, ob das ideale Benutzungsmodell schließlich überhaupt technisch umsetzbar ist. Eine Umsetzung auf Basis des exi-

stierenden Anwendungssystems bereitet zusätzliche Schwierigkeiten, wenn sie überhaupt möglich ist.

Andererseits bietet die Ausarbeitung des idealen Benutzungssystems aber auch große Vorteile: Im neuen, vereinheitlichten Benutzungsmodell verstecken sich keine Einschränkungen und Altlasten aus dem existierenden Anwendungssystem. Damit kann eine Kapselung auf Basis des idealen Benutzungssystems auch noch benutzt werden, wenn das existierende Anwendungssystem irgendwann im Rahmen einer Migration durch ein neues System ersetzt wird.

Im praktischen Projekt dieser Arbeit wurde wegen des Aufwands auf die detaillierte Ausarbeitung eines idealen Benutzungsmodells verzichtet, deshalb können dazu keine Erfahrungen vorgestellt werden. In zukünftigen Projekten – z.B. im Rahmen einer Migration – wäre es interessant, die Vor- und Nachteile kennenzulernen.

Die in dieser Arbeit nach dem ursprünglichen Vorgehensmodell entwickelte Kapselung ist damit eine Weiterentwicklung und Verbesserung des existierenden Benutzungsmodells und keine komplette Neuentwicklung. Wegen der Orientierung an den Anforderungen bleibt die kritische Frage, ob es möglich ist, auf Basis des existierenden Anwendungssystem ein fachlich orientiertes und einheitliches Benutzungsmodell zu schaffen, das die aktuellen Aufgaben der Konsumenten unterstützt.

Fazit

Auf der Basis von grundsätzlichen Überlegungen zum Benutzungsmodell wurde in diesem Kapitel ein Vorgehensmodell vorgeschlagen, das im Rahmen dieser Arbeit in einem praktischen Projekt ausprobiert wurde. Ein interessanter Punkt für künftige Projekte bleibt, ob die Ausarbeitung des idealen Benutzungsmodell weitere Vorteile bietet.

In den folgenden Kapiteln wird für jeden der Schritte das Vorgehen im Detail besprochen und es werden die praktischen Erfahrungen mit dem Vorgehensmodell vorgestellt.

4 Vom alten zum neuen Benutzungsmodell

Dieses Kapitel beschreibt die Erfahrungen mit den ersten vier Schritten des Vorgehensmodells: Der Evaluation der Anforderungen, der Analyse des existierenden Anwendungssystems, der Rekonstruktion des Benutzungsmodells und der Einführung eines neuen Benutzungsmodells. Jedem dieser Schritte ist ein eigener Abschnitt gewidmet.

4.1 Evaluation der Anforderungen

Im Kapitel 2.2 wurde das MCC vorgestellt, das eine komfortable Unterstützung der Telefonie bietet. Kaum ein Arbeitsplatz wird aber ausschließlich mit dem MCC arbeiten: Zusätzlich wird es ein fachliches Anwendungsystem geben, das die Kundendaten verwaltet und die Tätigkeit der Sachbearbeiter unterstützt. Das MCC bietet also eine Basisdienstleistung an, die mit dem fachlichen Anwendungssystem integriert werden muß.[1]

Zu dem Zeitpunkt, da eine Firma plant, Software zur Kommunikationsunterstützung einzusetzen, besitzt diese Firma in der Regel bereits Anwendungssoftware, mit der sie auch ihren Kundenbestand verwaltet und pflegt. Durch die Vielfalt der von solchen Firmen eingesetzten Anwendungen kann die Integration sehr spezielle Lösungen erfordern.

Im Rahmen dieser Arbeit wurde in einem praktischen Projekt eine Schnittstelle entwikkelt, die eine Integration zwischen MCC und Anwendungssoftware ermöglicht. In diesem Projekt wurde das vorgestellte Vorgehensmodell ausprobiert.

Evaluation der Anforderungen

Zu der Evaluation der Anforderungen und Aufgaben eines Anwendungsbereichs sind bereits sehr viele wissenschaftlichen Ansätze und Überlegungen veröffentlicht wurden. Gerade bei WAM hat die Beteiligung der Anwender und die Anwendungsorientierung einen hohen Stellenwert [vgl. Züllighoven 98, S. 126ff].

Diese Überlegungen sind auch auf die Evaluation der Anforderungen bei der Anbindung existierender Anwendungssysteme übertragbar. Deshalb soll hier nur kurz die Besonderheiten hervorgehoben werden, um dann die im Rest des Kapitels die konkreten Anforderungen aus dem CAI-Projekt vorzustellen.

Wesentlich für die Evaluation ist ein Abstand vom existierenden System, damit die vermeintlichen Anforderungen nicht durch die Möglichkeiten des Anwendungssystems

[1] Die in dieser Arbeit untersuchten Fachlichen Services kapseln das Kommunikationsunterstützungssystem von Tenovis (MCC) und *nicht* das fachliche Anwendungssystem.

geprägt sind. Ideal ist eine personelle oder zeitliche Trennung: Die Entwickler, die mit den Anwendern sprechen, kennen das existierende Anwendungssystem noch nicht.

Eine solche strikte Trennung war im CAI-Projekt nicht möglich, da alle Entwickler schon vorher mit dem MCC vertraut waren. Deshalb ist es wahrscheinlich, daß die ermittelten Anforderungen bei aller Sorgfalt doch durch das MCC geprägt sind. Auch die Anwender können – wenn sie das existierende System kennen – durch dessen Umgangsformen geprägt sein. Hilfreich ist die Untersuchung anderer vergleichbarer Anwendungssysteme, um so den Horizont zu erweitern.

Wichtig für die Kapselung ist die Ausarbeitung einer klaren Zielgruppe: Für alle Funktionen des existierenden Anwendungssystems lassen sich Benutzer vorstellen. Deshalb muß klar sein, wer die angepeilten Nutzer der Kapselung sind.

Neben der Befragung der Nutzer konnten im CAI-Projekt bereits existierende MCC-Integrationen ausgewertet werden. Diese Integrationen wurden von Tenovis-Mitarbeitern entwickelt, und setzten auf den internen MCC-Schnittstellen auf.

Im Folgenden werden die zentralen Anforderungen vorgestellt, die sich aus der Evaluation ergaben.

Zielgruppe

Wesentlich war die Festlegung einer klaren Zielgruppe: Das CAI sollte typische Integrationen unterstützen, bei denen Werkzeuge für die Agenten im Mittelpunkt stehen.

Dabei sollten die Integrationen auch von externen Entwicklern umgesetzt werden, mit Vorteilen für beide Seiten. Die externen Entwickler kennen das fachliche Anwendungssystem und die gewünschte Handhabung, die Entwicklung bei Tenovis wird entlastet.

> Zielgruppe der Schnittstelle sind Integrationen zwischen fachlichem Anwendungssystem und MCC, bei denen die Agentenwerkzeuge im Zentrum stehen. Die Integrationen sollen auch von externen Entwicklern umgesetzt werden.

Ohne eine solche Zielgruppe läuft man Gefahr, schließlich doch die gesamten Funktionalität des existierenden Anwendungssystems in der Kapselung Services anzubieten, da für alle Funktionen Benutzer denkbar sind.

Kundenidentifikation

Im Zentrum der Anforderungen stehen die automatische Identifikation des Kunden und dann die automatische Anzeige der Kundendaten im sogenannten *First-Screen*. Der First-Screen öffnet sich bei einem Anruf und zeigt die Kunden- und Vorgangsdaten eines Anrufers an. Dazu wird die Kundentelefonnummer von der Telefongesellschaft übermittelt oder die Kundennummer in einem Wartefeld abgefragt.

Szenario: Integration des First-Screens
Ruft ein Kunde an, öffnet sich beim Sachbearbeiter im First-Screen eine Liste der letzten Rechnungen und Vorgänge. Bei der Auswahl einer der Rechnungen wird das fachliche Anwendungssystem gestartet, in dem dann Details der Rechnung eingesehen werden können.

Hier ist eine Integration zwischen den Anruferdaten des MCC sowie den Kundendaten und Werkzeugen des Mitarbeiters nötig: Die Kundendaten, die zur Identifizierung genutzt und angezeigt werden, stammen aus dem Anwendungssystem. Vom First-Screen ausgehend sollen die Kundendaten direkt mit dem passenden Werkzeug angezeigt und bearbeitet werden. Vom First-Screen werden also andere Werkzeuge gestartet (inklusive der Auswahl einer bestimmten Bildschirmseite) und Workflows angestoßen.

> Wesentlich ist eine Weitergabe der Anruferdaten (Telefonnummer, Kundennummer) vom MCC an das fachliche Anwendungssystem.

Agentensteuerung

Von Tenovis wird mit dem Agentenmonitor ein Werkzeug geliefert, mit dem der Sachbearbeiter seinen Agenten steuern und seinen Systemzustand sehen kann. Der Sachbearbeiter kann hier zum Beispiel eine Pause beginnen oder die Jobbearbeitung beenden (vgl. Kapitel 2.2).

Bei vielen Firmen besteht aber der Wunsch, daß die Handhabung und das Aussehen dieses Agentenmonitors an das eigene fachliche Anwendungssystem angepaßt werden. Oft soll die Funktionalität des Agentenmonitors nicht mehr als eigenständiges Werkzeug existieren, sondern ein Teil des Werkzeugs zur Kundenverwaltung werden, so daß der Sachbearbeiter nicht mit vielen einzelnen Fenstern arbeiten muß.

> Für eine solche Integration des Agentenmonitors in die eigene Anwendung muß der Agent angemeldet und gesteuert werden. Das MCC muß Informationen über den Agentenzustand liefern.

Diese Anforderungen für First-Screen und Agentenmonitor stellen den Kern der fachlichen Anforderungen an eine Integration dar und genügen für die Darstellung in dieser Arbeit. In der Praxis werden sie natürlich um weitere Anforderungen ergänzt, z.B. um Zusatzinformation bei Anrufen im Rahmen von Telefonkampagnen.

Technologische Anforderungen

Die Technologie der fachlichen Anwendungssysteme ist sehr unterschiedlich: Die Bandbreite reicht von älteren Host-Anwendungen über maskenbasierte CRM-Systeme bis zu Client-Server Systemen. Das folgende Szenario liefert ein Beispiel für eine Host-Anwendung:

Szenario: Steuerung des fachlichen Anwendungssystems
Die Bürosoftware der Firma ist eine alte Host-Anwendung. Die Kundendaten werden auf einem Host-Rechner verwaltet. Die Mitarbeiter greifen über Terminal-Emulationen auf diesen Host zu.

Wenn ein Anruf eingeht, werden Tastendrücke an das auf dem Mitarbeiterrechner laufende Terminal geschickt, so daß das Terminal die Kundenmaske anzeigt.

Es ist eine enge Verzahnung zwischen Kommunikationssoftware und Anwendungssystem gewünscht, allerdings ist diese Integration für jede Firma anders – je nach benutzter Bürosoftware, Terminal-Emulation und ähnlichem – und muß jeweils neu entwickelt werden.

Grundsätzlich sind sehr unterschiedliche Ansätze denkbar, um für die Kundensysteme eine Möglichkeit zur Integration mit dem MCC zu geben:

- *Konfiguration einer Standardimplementation*
  Anpassungen in geringem Umfang können dadurch ermöglicht werden, daß eine Standardimplementation die Konfiguration der Präsentation erlaubt, es also z.B. konfigurierbar ist, welche Icons benutzt werden oder wie die GUI-Elemente angeordnet sind. Problematischer ist hingegen der Zugriff auf die Kundendaten: Hier ist in der Regel eine erhebliche Anpassung der Funktionalität nötig, die nur durch eine Programmierung – und nicht durch reine Konfiguration – möglich ist. Entsprechendes gilt, wenn aus den Agentenwerkzeugen heraus andere Anwendungen angesprochen werden sollen. Zusammenfassend erfüllt eine konfigurierbare Standardimplementation nur in den wenigsten Fällen die Anforderungen.

- *Anpassung einer Standardimplementation*
  Flexibler, aber auch deutlich aufwendiger ist der Weg, die Standardimplementation entsprechend den Anforderungen anzupassen, d.h. auf der Quellcode-Ebene Änderungen vornehmen. Diese Lösung erfordert von dem Entwickler einen aufwendige Einarbeitung. Es ist eine genaue Vorausplanung nötig, an welchen Stellen das Standardprogramm erweiterbar sein soll.

  Außerdem widerspricht eine Standardimplementation des Agentenmonitors oder First-Screens als eigenständiges Werkzeug den Lösungen, bei denen die Agentenwerkzeuge vollständig in das fachliche Anwendungssystem integriert werden sollen.

- *Agentenmonitor und First-Screen als Komponenten*
  Die wesentlichen Funktionselemente können als Komponenten – z.B. Active-X Controls – angeboten werden, die dann in eine Anwendung eingebunden werden können. Dabei würde es sich teilweise um GUI-Komponenten handeln, die z.B. den Agentenzustand darstellen, und teilweise um GUI-lose Komponenten, die Anruferdaten für die First-Screen-Integration bereitstellen und für die grundlegende Middleware-Verbindung zum MCC sorgen.

Vorteil ist die einfache und relativ flexible Handhabung, besonders in der Programmiersprache Visual Basic. Nachteilig ist hier, daß das Aussehen nur eingeschränkt anzupassen ist, daß GUI-lose Programme auf Basis der MCC-Funktionalität damit nicht möglich sind und daß nicht alle fachlichen Anwendungssysteme diese Komponenten-Technologie unterstützen.

- *Ein Application Programming Interface (API)*
  Ein API bietet die Telefonie-Dienstleistungen als reine Funktionalität an, als Bibliothek bzw. Service. Die Dienstleistungen werden im Sinne eines Fachlichen Service möglichst komfortabel und fachlich verständlich angeboten.

  Vorteil ist hier die große Flexibilität, unter anderem sind auf Konsumentenseite sehr unterschiedliche Technologien denkbar. Dafür erfordert diese Lösung auch mehr Implementationsaufwand beim Konsumenten als die Komponenten-Lösung.

Besonders in Betracht der technisch ganz unterschiedlichen MCC-Integrationen bei verschiedenen Kunden bietet die letzte Lösung die flexibelsten Möglichkeiten. Deshalb wurde bei Tenovis ein API aus Fachlichen Services unter dem Namen *Customer Application Interface* (CAI) entwickelt. Bei der Entwicklung dieses CAI wurde das in dieser Arbeit vorgestellte Vorgehensmodell erprobt.

Fazit

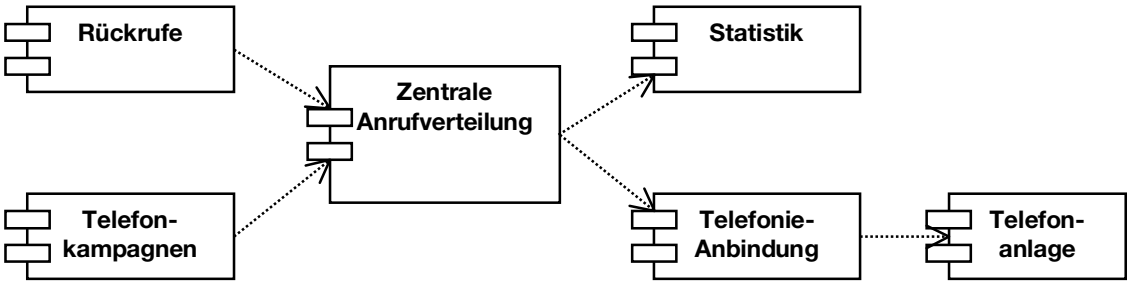
Die Analyse der Anforderungen an die Kapselung eines existierenden Anwendungssystems ist vergleichbar mit der Analyse der Anforderungen in einem normalen Entwicklungsprozeß. Wichtig ist hier vor allem ein Abstand vom existierenden System.

Zielgruppe der im CAI-Projekt entwickelten Schnittstelle sind Integrationen des MCC mit dem Kundensystem, bei denen die Agentenwerkzeuge im Vordergrund stehen. Die Integrationen sollen auch von externen Entwicklern durchgeführt werden.

Im Mittelpunkt der fachlichen Anforderungen stehen die Kundenidentifikation und die Agentensteuerung. Technologisch bietet ein API aus Fachlichen Services die flexibelsten Möglichkeiten.

4.2 Das Anwendungssystem

Die im CAI-Projekt untersuchten Fachlichen Services kapseln das MCC. Das MCC ist als verteiltes Client-Server-System implementiert: Die Funktionalität ist auf mehrere unabhängige Serverprozesse aufgeteilt. Ein solcher Serverprozeß übernimmt zum Beispiel die Anbindung an die verschiedenen Telefonanlagen, ein anderer verwaltet die Zustände der Agenten und teilt die Jobs den Agenten zu. Diese Serverprozesse kommunizieren auch untereinander: Über die Telefonie-Anbindung überwacht die zentrale Anrufverteilung die Telefone und die Service Entry Points.



Kommunizierende Serverprozesse im MCC[2]

Die Serverprozesse kommunizieren über die selbstentwickelte Echtzeit-Middleware *Process System Interface* [Kracke 96]. Eine wesentliche Eigenschaft von PSI ist die schnelle und präzise Erkennung von Verbindungsfehlern und Netzwerkstörungen. Dies ist wichtig, da das MCC unter Echtzeitbedingungen arbeitet: Hinter den Objekten und Ereignissen der Software stehen Anrufe und Kunden, die nur eingeschränkte Toleranz gegenüber Wartezeiten und Verbindungsabrissen mitbringen.

An die Serverprozesse wenden sich über PSI die verschiedenen Anwendungen, zum Beispiel der Agentenmonitor, um so ihre Dienstleistung auf Basis der Funktionalität der Serverprozesse anzubieten. Diese Middleware unterstützt das Versenden flacher Datenstrukturen; die Schnittstellen der Prozesse können nur in sehr eingeschränkter Form beschrieben werden, es gibt keine eigene IDL.

Analyse der Schnittstellen des MCC

Wie bei gewachsenen Anwendungssystemen wohl häufig, waren die Schnittstellen und das Benutzungsmodell nur spärlich dokumentiert. Dies machte eine tiefergehende Analyse der Schnittstellen notwendig, um hieraus das existierende Benutzungsmodell zu rekonstruieren.

Im wesentlichen bestehen die Funktionsaufrufe bei einem MCC-Server-Prozeß aus einer Funktionsnummer, mit der die Funktion angegeben wird, und einer flachen Da-

[2] Für die Notation werden Komponentendiagramme der *Unified Modelling Language* [Fowler 99] benutzt. Diese notieren eine Komponente als Kasten mit zwei waagerechten Balken. Das Interface der Komponente kann explizit als Kreis dargestellt werden. Verteilungsdiagramme ergänzen diese Notation um perspektivisch gezeichnete Kästen, die einzelne Rechner darstellen.

tenstruktur, in der die Parameter übergeben werden. Als Antwort werden eine oder mehrere Datenstrukturen gesendet.

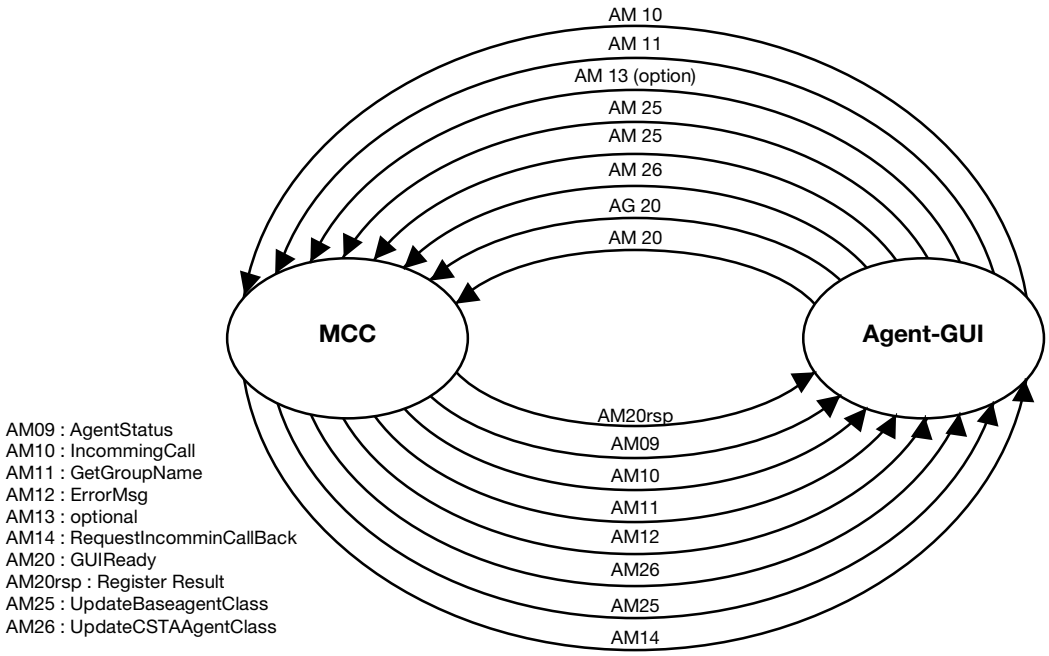
Bei Tenovis ließen sich zwei Arten von Dokumentation finden: Die eine gibt einen Überblick über die Reihenfolge der Interaktion und die ausgetauschten Datenstrukturen, die andere beschreibt detailliert die einzelnen Parameter und Attribute.

Beschreibung der Interaktion

Der MCC-Server-Prozeß bietet eine umfangreiche Schnittstelle, über die der Agentenmonitor die Agenten anmeldet, Befehle sendet und über Ereignisse informiert wird: ein wesentlicher Teil der im CAI gewünschten Funktionalität.

Die Interaktion an dieser Schnittstelle wurde von Hars [Hars 95] und Bleek [Bleek 97a] beschrieben. Neben einem Eindruck von dieser Dokumentation und von der Schnittstelle interessiert im folgenden auch der Vergleich zwischen der in der Praxis gefundenen Dokumentation und der grafischen Notation technischer Benutzungsmodelle.

Bei Hars findet sich ein Überblicksbild, das die gesamte Schnittstelle beschreibt:

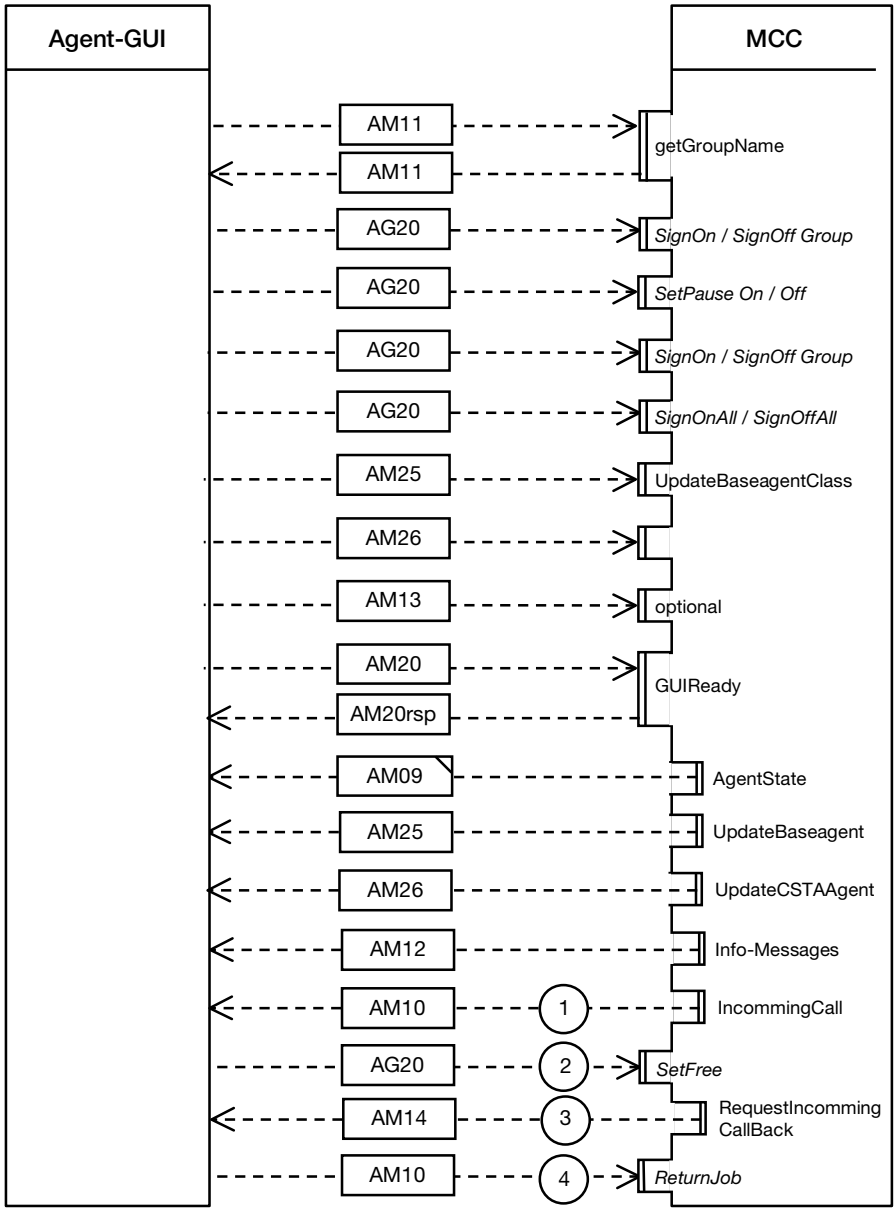


Zeichnung aus [Hars 95]

Im Zentrum der Interaktion zwischen MCC und Agent-GUI stehen die ausgetauschten Datenstrukturen, deren Bedeutung aber unklar bleibt, da sie nicht fachlich benannt sind. Deshalb wurde die Zeichnung um eine Liste ergänzt, die für einen Teil der Datenstrukturen einen aussagekräftigeren Namen angibt. Bei diesen Namen vermischen sich Materialien (AgentStatus) und Funktionsnamen (GetGroupName).

Unklar bleibt weiter, mit welchen Zweck und im welchem Kontext die Datenstrukturen geschickt werden, ob sie als Antwort oder als Ereignis gesendet werden.

Im technischen Benutzungsmodell – mit denselben Datenstrukturen – wird dieser Zusammenhang klar:



Zur Bezeichnung der Dienstleistungen wurde die Liste aus dem Bild von Hars benutzt. Waren einzelne Interaktionen dort nicht aufgeführt, wurde der bei Tenovis übliche Namen in kursiv ergänzt. Das Benutzungsmodell zeigt, daß die Datenstruktur AG20 bei mehreren Funktionen gesandt wird – vielleicht ein Grund dafür, daß sie bei Hars nicht in der Liste auftaucht.

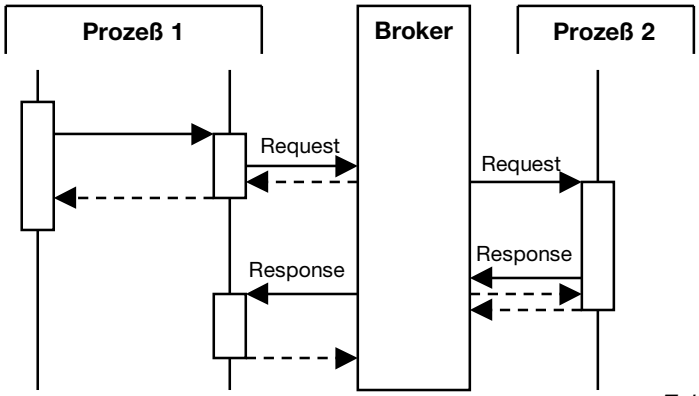
Weiter wird hier deutlich, daß die AM11 einmal als Antwort gesandt wird und ein Teil der Datenstrukturen (AM09 ... AM14) über Ereignisse informiert. Die unteren vier Pfeile beschreiben eine etwas komplexere Interaktion: Beim IncommingCall wird eine AM10

an das Agent-GUI übergeben und als Reaktion auf das Ereignis RequestIncoming-CallBack zurückgegeben.

Das Grundproblem, die nicht ausreichende Benennung und Beschreibung der Dienstleistungen, kann durch das technische Benutzungsmodell natürlich nicht so einfach gelöst werden. Es ergänzt die Notation von Hars aber um zusätzliche Informationen.

Interaktionsdiagramme

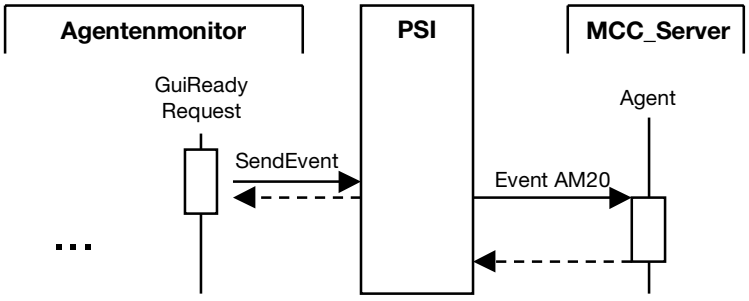
In [Bleek 97a] werden der innere Aufbau des Agentenmonitors und ebenfalls die Agentenmonitor-Schnittstelle des MCC dokumentiert. Es gibt keine grafische Darstellung der gesamten Schnittstelle, dafür werden einzelne Interaktionen mit *Prozeß-Interaktions-Diagrammen* dargestellt. In [Bleek 97] wurden dazu Interaktionsdiagramme [Fowler 99] so erweitert, daß sie eine asynchrone Interaktion mehrerer Prozesse darstellen können:



Zeichnung aus [Bleek 97a]

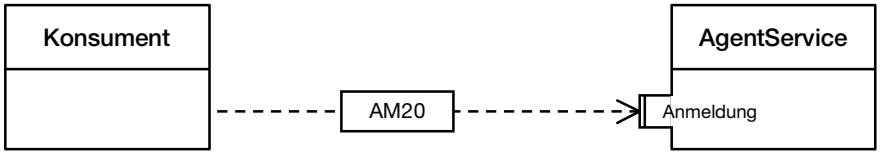
Wie üblich verläuft die Zeit von oben nach unten. Prozesse werden mit Prozeßrahmen zusammengefaßt, zusätzlich zu den Prozessen existiert ein Broker, der die Anfragen zwischen den Prozessen weiterleitet. Während die Kommunikation innerhalb der Prozesse synchron ist, kommt der Kontrollfluß von *Prozeß 1* bei Anfragen *(Requests)* an den anderen Prozeß sofort zurück. Trifft die Antwort dann später im *Prozeß 1* ein *(Response)*, entsteht ein neuer Kontrollfluß.

Mit Prozess-Interaktions-Diagrammen wurde die Interaktion der Agentenanmeldung bei Bleek wie folgt dargestellt (ohne inneren Aufbau des Agentenmonitors):



Zeichnung aus [Bleek 97a].

Hier sendet der Agentenmonitor den Dialog AM20, dadurch meldet sich der Agent am MCC an. Bei Bleek wurde das komplexe Innere des Agentenmonitors dargestellt, die übrige Interaktion ist sehr einfach. Als technisches Benutzungsmodell läßt sie sich wie folgt darstellen:



Die Darstellung als Interaktionsdiagramm ist deutlich aufwendiger und bietet sich nur für sehr komplexe Interaktionen an. Sie zeigt z.B. auch die Unterschiede zwischen synchroner und asynchroner Benutzung, von denen im technischen Benutzungsmodell abstrahiert wird. Für die Darstellung mehreren Dienstleistungen – wie bei Hars – eignen sich Interaktionsdiagramme nicht, da zwischen den Aufrufen der einzelnen Dienstleistungen keine zeitliche Reihenfolge besteht.

Beschreibung einzelnen Attribute

Wie wir gesehen haben, wurde an den Schnittstellen des MCC kein fachliches Objektmodell angeboten, sondern Parametersammlungen wurden einfach nur numeriert. Dies hat seine Wurzeln darin, daß die Middleware PSI nur das Versenden flacher Datenstrukturen unterstützt. Es war im CAI-Projekt schwierig, die Bedeutung der einzelnen Datenstrukturen zu verstehen: Die einzelnen Attribute und ihre Wertebereiche waren unzureichend und nur manchmal im Headerfile dokumentiert. Deshalb mußte bei der Analyse der Schnittstellen oft Rücksprache mit den ursprünglichen Entwicklern gehalten werden.

Der folgende Ausschnitt eines Headerfiles gibt einen Eindruck von den Schnittstellen: Er zeigt den Funktionsaufruf, mit dem sich ein Sachbearbeiter morgens beim MCC anmeldet.

```
//    FILENAME    : am20.hpp
//    DESCRIPTION : dialog from someone to the agent
//                  for set agent busy
//    Copyright (C) by Tenovis Business Communication GmbH
///// All rights reserved

#include "psihdr.hpp"
#include "acd_def.hpp"
////////////////////////////////////////////////////////////////////

#define A_RECEIVE_GUI_READY              0x4D410000
#define A_ALREADY_IN_USE                 0x47410024
#define A_NO_LICENCE_FOR_GUI_OBTAINED    0x47410040
```

```
typedef struct
{
      HEADER
      char  Location[LOCATIONLEN];
      Word  AgentOpState;
      Word  AgentSubState;
      char  UserName[AGENTNAMELEN];
      Word  PBXType;
      Dword ClientID;

}
Am20_3;
```

am20.hpp

Der Client schickt dem MCC die Datenstruktur Am20_3. Dabei gibt er im genormten *Header* den Funktionscode `A_RECEIVE_GUI_READY` an – der Agentenmonitor ist gestartet. Die *Location* ist die Telefonnummer des Agenten, die *States* geben an, ob der Agent nach dem Starten mit einer Pause beginnen soll oder sofort bereit ist für Anrufe. Im *UserName* steht der Name des Agenten, der *PBXType* gibt an, ob sein Telefon mit der Telefonanlage verbunden ist oder ob es sich um ein Handy handelt. Der Parameter *ClientID* wird nicht mehr benutzt.

Im Fehlerfall schickt das MCC als Antwort eine AM20rsp mit dem Funktionscode `A_ALREADY_IN_USE` (der Agent ist bereits angemeldet) oder `A_NO_LICENCE_FOR_GUI-_OBTAINED` (es gibt keine Lizenzen für Agentenarbeitsplätze mehr). Im positiven Fall schickt das MCC den aktuellen Zustand des Agenten und informiert über Zustandswechsel und eingehende Anrufe. Diese Interaktion ist nicht aus dem Headerfile ersichtlich, sondern nur aus der zusätzlichen Dokumentation.

Fazit

Wie aus den Beispielen deutlich wurde, war die Analyse des existierenden Anwendungssystems im wesentlichen eine Fleißarbeit, da alle Parameter einzeln auf ihre Bedeutung und möglichen Werte überprüft werden mußten. Es stellte dabei einen erheblichen Aufwand dar, die konkrete Bedeutung einzelner Parameter herauszufinden, die oft wenig mit den Variablennamen oder Kommentaren gemein hatte. Weiter wurde das MCC an diversen Stellen für Speziallösungen bei einzelnen Kunden erweitert, so daß einzelne Parameter nur manchmal benutzt wurden oder sogar neue Bedeutungen bekamen. Von großer Bedeutung im CAI-Projekt war dabei, daß die Entwickler des MCC zu der Schnittstelle befragt werden konnten.

4.3 Rekonstruktion des Benutzungsmodells

Aus der Analyse der Schnittstellen des MCC wurde das Benutzungsmodell rekonstruiert. Da hier vom MCC kein explizites Objektmodell angeboten wurde, lag ein Schwerpunkt auf der Rekonstruktion der Gegenstände des Anwendungsbereichs. Ein Kooperationsmodell beantwortet darüber hinaus die Frage, wie diese Gegenstände ausgetauscht werden und an welchem Ort sie sich befinden.

Agenten, Gruppen, Telefone

Wie beschrieben, werden im MCC zwischen Serverprozessen und Client nur kryptische Datenstrukturen ausgetauscht. Intern haben die Entwickler durchaus mit fachlichen Objekten gearbeitet, diese aber an der Schnittstelle nicht explizit verwendet.

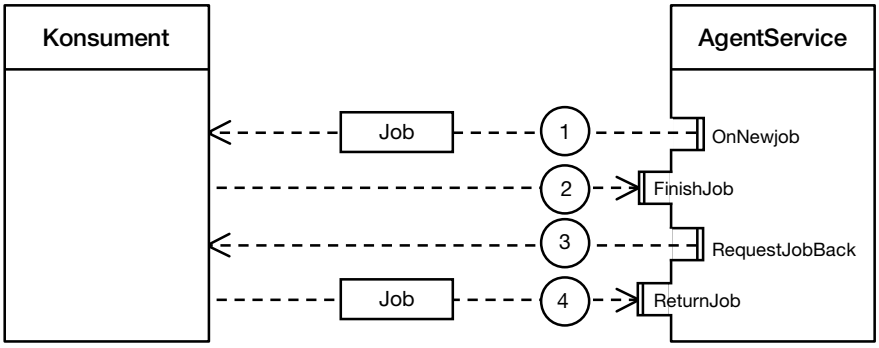
Im Kern des MCC gibt es Stellvertreter für die realen Agenten, Gruppen und Telefone. Diese Stellvertreter setzen die Logik der Jobverarbeitung und Anrufsteuerung um, sind also aktive Objekte. In ihrem Verhalten spiegeln sich Ereignisse der Telefonanlage und der Sachbearbeiter wieder. Durch die Schnittstelle können diese Objekte gesteuert und sondiert werden. So ließen sich dann auch einige übertragene Datenstrukturen als fachliche Objekte interpretieren: Der Dialog AM09 an der Agentenmonitor-Schnittstelle entspricht dem Agentenzustand.

Der Konsument wird mit dem Ereignis über Änderungen des Zustands informiert. Da sich der Zustand des Agenten zwischen zwei Aufrufen sondierender Funktionen durch die Aktionen anderer Konsumenten ändern kann, sind nicht einzelne Attribute abzufragen, statt dessen liefert das Ereignis eine konsistente Kopie des vollständigen Zustands. Diese Kopie ist ein Schnappschuß des MCC-Objekts zur Zeit der Abfrage. (eine ausführlichere Diskussion dieses Punktes findet sich auf Seite 71).

Andere Datenstrukturen enthalten nur Funktionsparameter – wie eine Gruppen-Id – und sind nicht als Materialien zu interpretieren.

Material Job als Kommunikationsmittel

Der Dialog AM10 entspricht einem Job, dem Stellvertreter eines Anrufs im MCC. Er enthält Informationen über den Anrufer und über den Stand der Bearbeitung. Er ist ein rein passives Material und wird zwischen den Gruppen und Agenten als Kommunikationsmittel weitergereicht. Erhält ein Sachbearbeiter einen Job, wird ihm das Job-Objekt – als Original – übergeben. Wenn er den Job fertig bearbeitet hat, muß er ihn explizit an das MCC zurückgeben.



Diese Interaktion ist im MCC wie folgt umgesetzt: Als erstes wird vom MCC ein neuer Job (AM10) an den Konsumenten gegeben. Dieser teilt nach der Bearbeitung mit, daß der Job zurückgegeben werden kann. Darauf fordert das MCC den Job explizit zurück, worauf der Agent dann tatsächlich den Job zurückgibt.

Ein Job befindet sich immer bei der Person, die auch mit dem Anrufer telefoniert. Vom Sachbearbeiter können im Job zusätzliche Information eingetragen werden, zum Beispiel, ob das Gespräch ein Erfolg war. Wird der Anruf an einen anderen Sachbearbeiter weitergeleitet, wird auch der Job als Kommunikationsmittel zwischen den Bearbeitern weitergereicht.

Fazit

Obwohl kein explizites Benutzungsmodell der Schnittstellen existierte, zeigte es sich, daß sich für viele der benutzten Umgangsformen und Datenstrukturen eine fachliche Interpretation finden ließ. Hier waren also die unzureichende Dokumentation und schlecht gewählte Bezeichner das Haupthindernis beim Verständnis des Benutzungsmodells.

Dieses Kapitel stellte nur die zentralen Konzepte des Benutzungsmodells des MCC vor. Wie wir im nächsten Teil sehen werden, gab es gerade in den Details einige Unstimmigkeiten und Widersprüche, die es erschwerten, daraus ein einheitliches fachliches Benutzungsmodell zu entwickeln.

4.4 Ein einheitliches Benutzungsmodell

Wesentlicher Unterschied zwischen den Schnittstellen des MCC und den Fachlichen Services ist das fachliche Benutzungsmodell der Services. Dieses wurde auf Basis der Anforderungen aus dem existierenden Benutzungsmodell entwickelt. Wichtige Schritte waren die Auswahl von Funktionen, das Vereinheitlichen von Gegenständen, Fragen der Granularität und die Strukturierung in Fachliche Services; diesen Schritten ist im Folgenden jeweils ein Absatz gewidmet.

Auswahl der Funktionen

Beim Vergleich der Anforderungen mit dem existierenden Benutzungsmodell ergab sich, daß das MCC alle notwendigen Funktionen anbietet – es bietet sogar deutlich *mehr* Funktionalität an als für typische Agentenmonitore und First-Screens notwendig. Für die Fachlichen Services wurde also eine Auswahl der Funktionalität getroffen, die im CAI sinnvoll zu benutzen ist.

Diese Auswahl anhand der Anforderungen erforderte eine nochmalige sehr sorgfältige Prüfung, ob ein Verzicht auf einzelne Funktionen und Attribute keine verborgenen Folgen hat. Die Prüfung stellte sicher, daß auch die eingeschränkte Schnittstelle noch eine ausreichende Funktionalität anbot.

Bei der Betrachtung des Benutzungsmodells des MCC wurde relativ schnell deutlich, welche Funktionen für die Zielgruppe kaum eine Rolle spielen. Neben dem Mut, sich überhaupt auf eine Reduktion einzulassen, ist auch ein iterativer Softwareentwicklungsansatz unerläßlich: Wird die Schnittstelle als erste Version in einem Zyklus betrachtet, ist es nicht problematisch, später eine benötigte Funktion hinzuzufügen.

Ein weiterer Aspekt bei der Auswahl der Funktionalität war die Elimination von Redundanzen: So findet sich in den MCC-Schnittstellen an drei Stellen die Möglichkeit, den Zustand eines Agenten abzufragen:

- Der Agent kann seinen Zustand mitteilen.
- Das MCC kann eine Liste der Zustände aller Agenten liefern.
- Eine Gruppe kann den Zustand eines ihrer Agenten mitteilen.

Im CAI wurden diese Abfragen auf eine Version beschränkt. Dadurch wurde die Schnittstelle überschaubarer und der nötige Implementationsaufwand verringerte sich. Schlüssel für die Reduzierung waren Referenzen in Form von AgentenIDs: Die Gruppe verweist so nur auf die IDs der Agenten, über die beim AgentService eine Kopie des Agentenzustands angefordert werden kann. Ebenso liefert die Abfrage der existierenden Agenten eine Liste mit IDs.

Durch das Aufstellen der Anforderungen ergab sich eine Schnittstelle, die nur ca. ein Viertel der Methoden umfaßte, die die MCC-Schnittstellen für Agentenmonitor und First-Screen anboten. Dies spricht noch einmal für das Ermitteln der Anforderungen unabhängig von der angebotenen Funktionalität: Eine blinde Umsetzung der MCC-Funktionalität hätte komplexere Fachliche Services geliefert, die auch eine aufwendigere Implementierung erfordert hätten. Weiter war die Reduzierung eine wichtige Grundlage, um ein einheitliches fachliches Modell der Gegenstände einzuführen.

Einführung eines fachlichen Modells der Gegenstände

Wie beschrieben wurden im MCC nur flache Parametersammlungen benutzt, aus denen aber fachliche Objekte rekonstruiert werden konnten. Hierfür wurde ein Modell der von den Servern bearbeiteten Gegenstände eingeführt. Die Vereinheitlichung dieser Objekte zu einem Objektmodell für das CAI erwies sich als problematisch: Viele Objekte wurden im MCC an verschiedenen Stellen unterschiedlich dargestellt, mit unterschiedlichen Zusammenstellungen der Attribute.

So erhält man bei der Abfrage des Agentenzustands

- beim Agenten Informationen über den Agenten und seine Gruppen, aber keine Informationen über den Job;
- bei der Liste aller Agentenzustände eine sehr eingeschränkte Sicht (ohne Gruppen), die dafür zusätzlich den Typ des Agenten enthält;
- über die Gruppe Informationen über den Agenten, seinen Job, den Typ des Agenten und Informationen zur Verbindung zu dieser Gruppe.

Diese unterschiedlichen Versionen sind nicht fachlich erklärbar, sie sind aus der historischen Entwicklung des Anwendungssystems erwachsen. Die Schnittstellen wurden von verschiedenen Leuten zu verschiedenen Zeitpunkten entwickelt, dabei wurden nur die Informationen übergeben, die das Werkzeug gerade benötigt.

Eine Vereinheitlichung für ein übergreifendes Objektmodell ist schwierig: Informationen, die nicht Teil des übergebenen Objekts sind, können nur mit Mühe ergänzt werden. Andererseits will man bei den vollständigeren Objekten nicht auf diese Informationen verzichten.

Am Anfang der Vereinheitlichung stand die Reduktion, durch die das Benutzungsmodell des MCC überschaubarer wurde:

- *Elimination von Attributen*
  Genau wie bei den Funktionen ließen sich bei den Attributen einige finden, die für die betrachtete Zielgruppe keine Rolle spielen. Diese Attribute wurde im neuen Benutzungsmodell nicht berücksichtigt.

- *Reduktion auf eine Version*
  Wird ein Objekt wie im Beispiel in mehreren redundanten Funktionen benutzt, dann ist bei der Reduktion auf eine Funktion nur noch eine Version nötig. Hier kann man z.B. die aussagekräftigste Version auswählen.

Sind nach der Reduktion an der Schnittstelle immer noch unterschiedliche Versionen desselben fachlichen Objektes zu finden, bleiben zwei weitere Optionen:

- *Ergänzen der fehlenden Daten*
  Manchmal kann man das Objekt intern aus mehreren Funktionsaufrufen akkumulieren. Statische Werte – wie der Typ eines Agenten – können beim Start des Services abgefragt und dann bei Bedarf ergänzt werden.

- *Durchscheinen der Versionen im Service*
  Schließlich kann ein fachliches Objekt auch im Service in verschiedenen Formen und Rollen angeboten werden – unter unterschiedlichen Namen oder mit manchmal nicht gesetzten Attributen. Dies führt aber immer zu einem Ausfasern des Objektmodells und einer unklaren Schnittstelle, da die verschiedenen Rollen des Objektes nur technisch aus dem existierenden Anwendungssystem motiviert sind. Für den Konsumenten ist es nicht nachzuvollziehen, warum er manchmal einzelne Attribute nicht erhält. Dieses Problem zeigt sich schon darin, daß diese unterschiedlichen Versionen sehr schwierig zu benennen sind.

Im CAI wurden einige Werte – z.B. Agententypen und Gruppennamen – beim Starten des Service abgefragt, um dann bei Bedarf die fehlenden Attribute zu ergänzen. Seltener wurde ein Objekt auch aus mehreren Funktionsaufrufen akkumuliert. Hervorzuheben ist, daß die Einführung des Objektmodells in den Fachlichen Services einen wesentlichen Verständnisvorteil gegenüber den Variablensammlungen des MCC bietet.

Aufgabenorientierung und Granularität

Da das neue Benutzungsmodell sich an den Aufgaben und Anforderungen der Zielgruppe orientiert, sollen häufige Aufgaben auch komfortabel zu erledigen sein. Bei der Betrachtung der MCC-Schnittstelle stellt man fest, daß manche Funktionen umständlicher umzusetzen sind als andere: Hier stimmt die Granularität der angebotenen Dienstleistung nicht.

Beispielsweise enthält der Agentenzustand die Information, in welchen Gruppen der Agent an- und abgemeldet ist; hier werden aber nur die GruppenIDs übergeben. Für die Anzeige werden aber immer auch die Gruppennamen benötigt, die damit zusätzlich einzeln abgefragt werden müssen. Eine komfortable Funktion würde also immer gleich die Gruppennamen mitliefern, anstatt dies in zwei Aufrufe zu teilen.

Ein anderes Beispiel ist die komplexe Interaktion, mit der ein Job an das MCC zurückgegeben wird (vgl. Kapitel 4.3). Hier kann im CAI der Job direkt beim Aufruf von *JobFinished* mitgegeben werden. Das folgende Ereignis *RequestJobBack* und der Aufruf von *ReturnJob* werden intern gekapselt.

Eine Funktion, die in dieser Form auf die Anforderungen angepaßt wird, erledigt dieselbe Aufgabe mit weniger Aufrufen, sie ist grobkörniger. Dies ist besonders bei einem asynchronen Benutzungsmodell nicht zu unterschätzen: Sollen hier mehrere Aufrufe nacheinander erfolgen, muß jeweils der Callback des vorhergehenden Aufrufs abgewartet werden. Der entsprechende Kontrollfluß verteilt sich damit im Code. In einem synchronen Modell können die Aufrufe hingegen einfach hintereinander stehen.

Weitere Beispiele für komfortable, aufgabenorientierte Anpassungen im CAI sind:

- Die Abfrage der E-Mail-Informationen: Wenn ein E-Mail-Job eingeht, können im MCC weitere Informationen abgefragt werden, die so nicht im Job enthalten sind. Dies sind z.B. der Sender und der Betreff der E-Mail. Im CAI wird ein eingehender E-Mail-Job automatisch um diese Informationen ergänzt, die Abfrage der Informationen ist für den Benutzer unsichtbar.
- Der Zugriff auf die Konfigurationsdaten: Manche Konfigurationsdetails lassen sich nur aus der Konfigurationsdatenbank des MCC auslesen. Da der Aufbau der Datenbank komplex ist, wurde das CAI um Funktionen ergänzt, die häufig benötigte Einstellungen liefern.

Solche Anpassungen der Granularität können aber auch Probleme aufwerfen: Sie können zu weitgehende Annahmen über die Benutzung voraussetzen und damit die möglichen Verwendungen einschränken. Teilweise stehen sie im Widerspruch zu einer schlanken, orthogonalen Schnittstelle, wenn sie Redundanzen und zusätzliche Informationen einführen.

Besonders kritisch ist der Umgang mit Sonderfällen und Fehlersituationen: Wenn eine Teilfunktion fehlschlägt, kann die gesamte Komfortfunktion fehlschlagen, vergleichbar mit einer Transaktion, es kann aber auch ein Teilergebnis weitergereicht werden. Die Fehlersituation muß dem Konsumenten verdeutlicht werden, obwohl dieser die Hintergründe der komplexen Funktion nicht kennt.

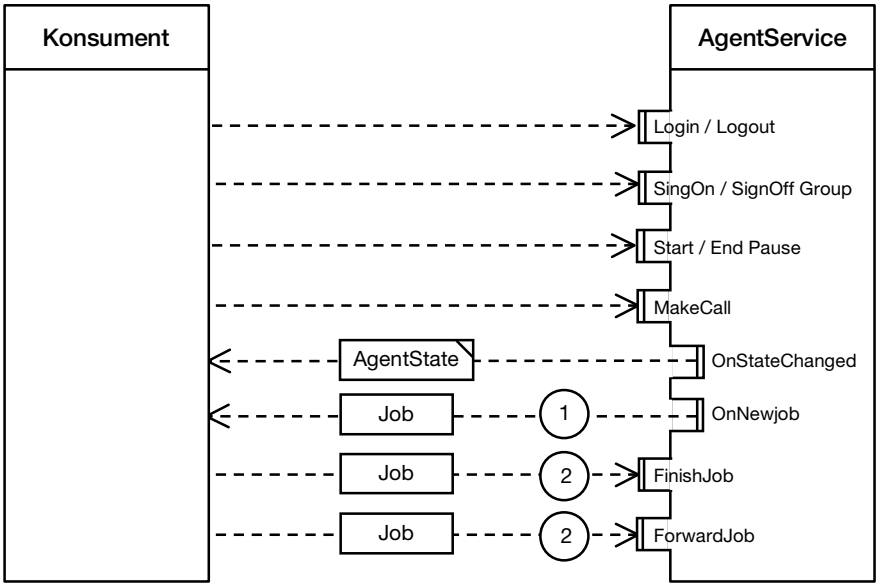
Eine solche Situation ergab sich bei der Ergänzung des Jobs um E-Mail-Informationen: Das CAI sah vor, daß ein beim Agenten ankommender E-Mail-Job um Informationen über die E-Mail (z.B. den Betreff) ergänzt wird. Ließ sich die Information nicht ergänzen, wurde der Job nicht angenommen und dem MCC zurückgegeben. Nun wurden E-Mail-Jobs bei einem Kunden aber auch für andere elektronische Medien verwendet, in diesem Fall mußte die Abfrage des Betreffs fehlschlagen. Der Job hätte trotzdem durchgereicht werden müssen.

Insgesamt können Änderungen der Granularität einen sehr großen Komfortgewinn bringen, bergen aber auch das Risiko unbeabsichtigter Seiteneffekte.

Strukturierung in Fachliche Services

Um die Funktionalität überschaubar und strukturiert anzubieten, wurde sie in mehrere Fachliche Services aufgeteilt. Diese Services bieten ihre Funktionalität rund um die zentralen Objekte des MCC – Agenten, Gruppen und Telefone – an.

Der AgentService bietet z.B. alle Funktionen zur Steuerung und Überwachung von Agenten, bei ihm kann man sich für Ereignisse – wie eingehende Jobs – registrieren. Die folgende Darstellung zeigt sein technisches Benutzungsmodell:



Der Konsument kann sich beim MCC anmelden, sich bei einer Gruppe für Jobs eintragen, die Pause beginnen und einen Anruf machen. Über Zustandsänderungen wird er durch Ereignisse informiert; bei Anrufen erhält er einen Job, der unter anderem die Nummer des Anrufers enthält. Diesen Job kann er zurückgeben und beenden oder alternativ an einen anderen Agenten weiterleiten.

Im Vergleich zum technischen Benutzungsmodell der Agentenmonitor-Schnittstelle des MCC (Kapitel 4.2) ist der AgentService verständlicher und überschaubar. Trotzdem bietet er alle Funktionalität, die von der Zielgruppe zur Agentensteuerung benötigt wird. Die Darstellung als technisches Benutzungsmodell gibt einen guten Überblick über die Möglichkeiten und Umgangsformen dieses Services.

Fazit

Insgesamt wurde ein verständliches, fachliches motiviertes und komfortables Benutzungsmodell gefunden. Dieses war aber nur durch trickreiche Programmierung möglich, die ein gutes Detailwissen des MCC erforderte. Wesentlich für eine schlanke und komfortable Schnittstelle war die Auswahl der sinnvollen Funktionen und Objektattribute anhand der Anforderungen.

Bewährt hat sich die grafische Notation für technische Benutzungsmodelle, die einen guten Überblick über die an der Schnittstelle angebotenen Dienstleistungen liefert.

5 Technische Umsetzung

Die in diesem Kapitel vorgestellten beiden letzten Schritte des Vorgehensmodells behandeln die technische Umsetzung der Kapselung: Die Ausarbeitung einer Software-Architektur und die technische Umsetzung der Fachlichen Services. Danach wird schließlich exemplarisch für das CAI die Umsetzung des AgentService vorgestellt.

Zuerst wird in diesem Kapitel aber eine besondere Problematik Fachlicher Services diskutiert: Die unterschiedlichen Technologien der Konsumenten können Einschränkungen nötig machen, durch die eine im Benutzungsmodell gewünschte Ereignisbehandlung verhindert wird.

5.1 Fachliche Services und Ereignisse

Zu den Anforderungen an das CAI gehört die Information der Sachbearbeiter über Ereignisse, z.B. über einen eingehenden Job. Solche Fachlichen Services, die eine Schnittstelle zum Registrieren für Ereignisse anbieten, sollen im Folgenden als „aktive“ Fachliche Services bezeichnet werden, im Gegensatz zu „passiven“ Fachlichen Services, die nur auf Anfragen von Konsumenten reagieren.

> **Definition: Aktiver Fachlicher Service**
> Ein Fachlicher Service ist aktiv, wenn er selbsttätig über einen Benachrichtigungsmechanismus Interessenten über Ereignisse informiert. Der Zeitpunkt dieser Information ist nicht vorhersehbar.

Aktive Services findet man unter anderem im technischen Umfeld: Hier wurde von Bleek die Entwurfsmetapher der *Sonde* eingeführt, die technische Geräte überwacht und über Änderungen informiert [vgl. Bleek 97]. Ein anderer aktiver Service wäre ein *Materialverwalter*, der bei gemeinsam genutzten Materialien über Änderungen durch einen anderen Benutzer informiert. Eine solche Funktionalität wird von Otto und Schuler auch für zentrale Fachliche Services diskutiert:

»Um fachliche Mehrbenutzerfähigkeit sinnvoll realisieren zu können, ist ein Benachrichtigungsmechanismus für bestimmte Ereignisse am Fachlichen Service sinnvoll. So kann sich ein Konsument am Fachlichen Service registrieren lassen, um beispielsweise bei Änderungen dargestellter Daten benachrichtigt zu werden oder um die Namen konkurrierender Benutzer anzeigen zu können.« [Otto & Schuler 2000, S. 89]

Einschränkungen bei Webtops

Problematisch bei der Umsetzung aktiver Fachlicher Services ist, daß ganz unterschiedliche Technologien auf Konsumentenseite auf dem Service aufsetzen können: Ein wichtiges Merkmal Fachlicher Services ist, daß sie eine reine Funktionalität anbieten. Auf dieser Funktionalität setzen z.B. Desktop-Programme oder auch Internet-Anwendungen (Webtops) auf. Die Bedeutung solcher Webtops nimmt immer mehr zu, so daß fast jedes große Anwendungssystem neben Desktop-Oberflächen auch einen

Web-Zugang anbieten soll. Diese Technologien auf HTML-Basis eignen sich aber nicht für ein Ereignis-Modell, wie es aktive Fachliche Services anbieten:

»Hierbei muß beachtet werden, daß es für Benutzungsschnittstellen-Typen mit rein reaktivem Charakter (z.B. Webtop [...]) sinnlos ist, sich für derlei Nachrichten anzumelden, da sie diese dem Benutzer nicht unmittelbar präsentieren können. Unterstützt man mehrere Benutzungsschnittstellen-Typen, so muß diese Einschränkung beim Entwurf der kompletten Anwendungsarchitektur berücksichtigt werden, da sonst eventuell vorgesehene Koordinationstechniken scheitern könnten.«

[Otto & Schuler 2000, S. 89]

Dies ist ein zentraler Punkt bei Otto und Schuler, die damit noch einmal den Unterschied zwischen klassischen WAM-Funktionskomponenten und Fachlichen Services herausarbeiten: Während Funktionskomponenten ihr GUI typischerweise über einen Beobachtermechanismus informieren, müssen Fachliche Services auch auf Web-Konsumenten Rücksicht nehmen, die eine solche Technologie nicht unterstützen.

Mögliche Auswege

Da die Information über Ereignisse ein Teil der Konsumentenanforderungen ist, sind zwei Wege aus dem Dilemma möglich:

Eine Möglichkeit ist eine zweite Schnittstelle am fachlichen Service, an der die Ereignisse auch abgefragt werden können (Polling). Nun können sich Desktop-Konsumenten bei der Ereignis-Schnittstelle anmelden, während Webtop-Konsumenten die Ereignisse regelmäßig abfragen. Problematisch bleibt dabei, daß z.B. in einem Webtop-First-Screen die Ereignisse sehr häufig abgefragt werden müßten, um eine zeitgetreue Anzeige bei Anrufen zu gewährleisten. Dies würde zu einem Flackern und zu einer hohen Netzlast führen.

Eine andere Möglichkeit ist die Nutzung von HTML-Erweiterungen auf Konsumentenseite. In einem Projekt bei Tenovis, das einen HTML-Agentenmonitor entwickelte, wurde für die Ereignisse Java benutzt: Ein kleines Java-Applet aktualisierte bei Ereignissen die Anzeige und holte ggf. das First-Screen-Fenster in den Vordergrund.

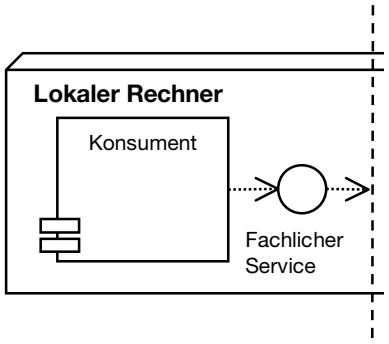
Ob die Nutzung solcher Erweiterungen möglich ist, hängt von den Anforderungen im konkreten Projekt der Konsumenten und vom Anwenderkreis der Software ab – und ist damit nicht unbedingt bei der Konstruktion der Fachlichen Services festzustellen.

Fazit

Zusätzlich zu den Ereignissen bietet das CAI auch eine Möglichkeit zum Polling. Der Anwendungsbereich Telefonie mit Echtzeitanforderungen und technischen Ereignissen eignet sich aber nicht besonders für rein reaktive Webtechnologien. Hier bot nur die Nutzung von HTML-Erweiterungen einen Ausweg.

5.2 Entwicklung der Architektur

Der Konsument wird einen Fachlichen Service immer über sein wohldefiniertes Interface ansprechen, und viele bisher angesprochene Punkte – wie das Benutzungsmodell – beziehen sich ausschließlich auf diese Schnittstelle.



Sicht des Konsumenten

Dieses Kapitel diskutiert, wie der Fachliche Service intern aufgebaut ist, vor allem wenn seine Dienstleistung auf der Basis existierender Anwendungssysteme erbracht wird. In den folgenden Diagrammen ist die Schnittstelle zum Konsumenten als Kreis explizit eingezeichnet: An dieser Schnittstelle erbringt der Fachliche Service seine Dienstleistung.

Verteilung und Proxies

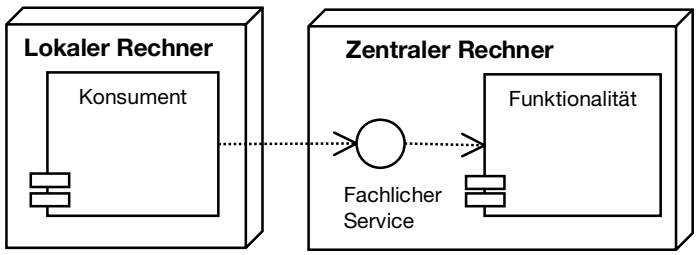
Konzeptionell ist ein Fachlicher Service eine gemeinsam genutzte Ressource, die ihre Dienstleistung über ein wohldefiniertes Interface anbietet; technisch wird dies dadurch umgesetzt, daß die Rechner der kooperierenden Benutzer und die zentralen Server durch ein Netzwerk verbunden werden.

Die Funktionalität eines solchen Fachlichen Service ist also immer mit einem Netzwerkzugriff verbunden:

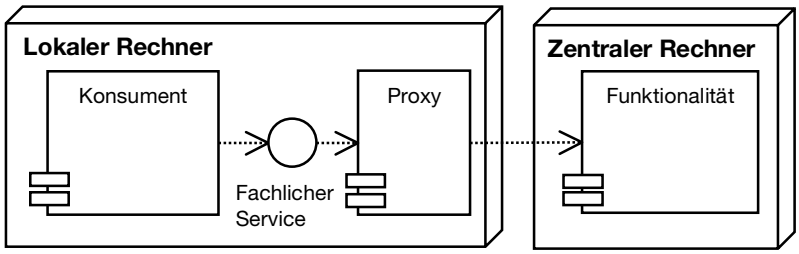
> Ein Fachlicher Service als gemeinsame Ressource benötigt zur Erfüllung seiner Funktionalität Netzwerkzugriffe, z.B. wenn er
>
> - von entfernten Rechnern über eine Netzwerkverbindung genutzt werden kann oder
> - intern zur Erfüllung seiner Aufgabe über eine Netzwerkverbindung auf eine andere Ressource zugreift.

Mehrere Architekturen ermöglichen die physikalische Verteilung Fachlicher Services:

Der Fachliche Service kann über eine Middleware im Netzwerk angeboten werden. Beispiel wäre ein Service, der als CORBA-Service implementiert ist [vgl. Redlich 96]. Auf diesen Service kann ein Konsument über Standard-Technologien verteilt zugreifen. Der verteilte Zugriff wird dabei von der Middleware komfortabel unterstützt.



Alternativ kann der Konsument lokal durch einen Proxy [vgl. Gamma et al. 96, S. 227ff] beim Zugriff auf den zentralen Service unterstützt werden. Dieser Proxy übernimmt im wesentlichen die Aufgabe, die Benutzung der Middleware zu kapseln.



Diese zweite Architektur wird auch von Otto und Schuler vorgeschlagen:

»Ein Proxy zu einem Fachlichen Service wird auf Clientseite instanziiert, kapselt die Verbindungsaufnahme zum verteilten Fachlichen Service und leitet Aufrufe von dessen Diensten weiter. Der Proxy wird dabei mit der gleichen Schnittstelle wie der Fachliche Service selbst angesprochen, gibt jedoch zusätzlich verteilungsbedingte Fehler an die Anwendung weiter.« [Otto & Schuler 2000, S. 90]

Vorteil einer solchen Architektur ist, daß je nach Middleware durch den Proxy eine deutliche Entlastung des Konsumenten stattfinden kann. Weiter werden bei einem Wechsel der Middleware weniger Änderungen am Konsumenten notwendig. Dabei verbirgt der Proxy, welche Middleware benutzt wird, nicht aber, daß ein Netzwerkzugriff erfolgt: Der Netzwerkzugriff wird durch entsprechende Fehlermeldungen deutlich. Die Probleme mit dem vollständigen Verbergen von Verteilung werden in Kapitel 5.3 im Abschnitt „Verteilungstransparenz“ diskutiert.

Otto und Schuler nennen als Vorteile dieser Architektur den Komfort durch den Proxy und die Unabhängigkeit von der Middleware.

»Unwichtige oder nur temporäre Netzfehler kann der Proxy dabei eventuell selbst ausgleichen, beispielsweise, indem er die Verbindung neu aufsetzt. Bei einem Wechsel des Verteilungsmechanismus müssen nur die Stellvertreter-Objekte der Fachlichen Services entsprechend angepaßt werden – die Fachlichen Services selbst sowie die Client-Anwendungen bleiben unverändert.« [Otto & Schuler 2000, S. 90]

Im CAI-Projekt wurden Proxies dementsprechend benutzt, um den Konsumenten einen komfortablen Zugriff auf die CORBA-Services zu bieten. Dabei verbargen die Proxies die Middleware soweit wie möglich. Dies war auch schon deshalb notwendig, da Entwickler mit ganz unterschiedlicher Vorbildung mit dem CAI arbeiten sollten. Dabei

zeigte sich als der gravierendste Nachteil, daß der Proxy beim Zugriff über verschiedene Programmiersprachen jeweils neu implementiert und dokumentiert werden muß.

Anbindung existierender Anwendungssysteme

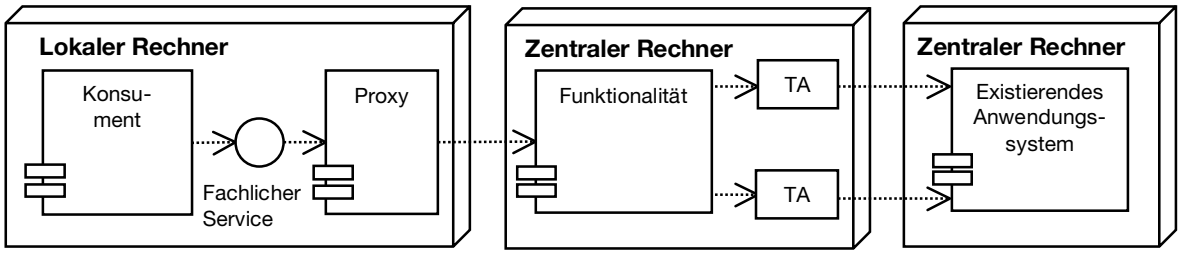
Eine Architektur zur Anbindung existierender Anwendungssysteme wurde am Arbeitsbereich SWT zuerst im *eFrame*-Projekt untersucht. In diesem Projekt wurde die Entwurfsmetapher der Thin Adapters entwickelt, die sich besonders gut eignet, wenn der Fachliche Service nur an wenigen Stellen das existierende Anwendungssystem benutzt. Für Services wie das CAI, die die existierende Funktionalität auf sehr breiter Front benutzen, eignen sich Thin Adapters nur bedingt. Für solche Services wird eine alternative Architektur vorgestellt.

Thin Adapters

Im eFrame-Projekt wurden Fachliche Services entwickelt, die Funktionalität aus einem SAP-System anbieten. Die Services setzten hier nur an wenigen Stellen auf dem existierenden Anwendungssystem auf, der überwiegende Teil der Dienstleistung war unabhängig vom existierenden System.

Für diese einzelnen Funktionen, die vom existierenden System benötigt werden, wurde jeweils ein *Thin Adapter* entwickelt, der den Zugriff auf die Funktion des existierenden Systems kapselt. Thin Adapters verhindern, daß sich die Funktionalität des Fachlichen Services mit technischen Details des Anwendungssystems mischt und erledigen das Marshalling.

> **Definition: Thin Adapter**
> Bei der Integration existierender Anwendungssysteme bietet ein *Thin Adapter* eine einzelne Funktionalität des Anwendungssystems an. Dabei bildet er die Funktionalität unverändert ab, verbirgt aber die Details des Zugriffs auf das Anwendungssystems. Die benutzten Datenstrukturen wandelt der Thin Adapter in fachliche Materialien um.



Durch den Thin Adapter konnte die Funktionalität des Fachlichen Services rein objektorientiert implementiert werden, der Zugriff auf das Anwendungssystem war im Adapter verborgen. Die Middlewares zwischen Proxy und Funktionalität respektive Thin Adapter und Anwendungssystem können sich dabei deutlich unterscheiden, z.B. wenn zum Anwendungssystem das proprietäre Protokoll eines Altsystems eingesetzt wird, zwischen Adapter und Proxy aber eine moderne Middleware.

„Thin“ sind die Adapter in dem Sinne, daß sie zwar Datenstrukturen in Materialien wandeln, ansonsten aber direkt eine Funktionalität des Anwendungssystems anbieten.

Im Fachlichen Service wird dann eine an den aktuellen Anforderungen orientierte Dienstleistung angeboten, die von der existierenden Funktionalität abstrahiert.

Thin Adapters eignen sich besonders, wenn nur wenige einzelne Funktionen eingebunden werden. Der folgende Abschnitt diskutiert die Anbindung umfangreicher Funktionalität.

Anbindung umfangreicher Funktionalität

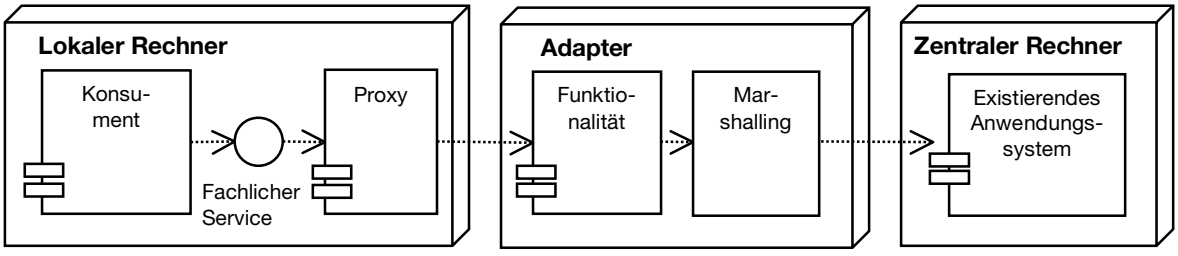
Im CAI-Projekt wird ein umfangreicher Teil der MCC-Funktionalität als Fachlicher Service angeboten. Das CAI hat dabei in der Architektur im wesentlichen die Aufgabe eines Adapters [vgl. Gamma et al. 96, S. 151ff], es implementiert keine eigene Funktionalität.

Zwei Gründe sprechen im CAI dagegen, die Metapher der Thin Adaptern direkt zu verwenden: Zum einen werden sehr viele Funktionen des existierenden Anwendungssystems genutzt. Eine Architektur, die jeweils einen Thin-Adapter – z.B. als eigene Klasse – verwenden würde, wäre sehr unübersichtlich. Zum anderen ist die Wandlung in Materialien im CAI sehr aufwendig, die Adapter wären also nicht „dünn“.

Im CAI wurde eine Schicht zum Marshalling verwendet, die sich an den Konzepten der Thin Adapters anlehnt. Wie bei diesen kapselt sie Details der Anbindung an das Anwendungssystem und die benutzte Middleware. Auch hier orientiert sich die Struktur am existierenden System: Jeder Teil der Marshalling-Schicht stellt die Funktionalität einer existierenden Schnittstelle zur Verfügung.

Wie in Kapitel 4.4 beschrieben war die Umwandlung der im MCC benutzten Datenstrukturen in fachliche Materialien im CAI-Projekt nicht einfach. Oft mußten fehlende Attribute ergänzt werden.

Deshalb wandelt die Marshalling-Schicht – im Gegensatz zu der Idee der Thin Adapters – die Datenstrukturen noch nicht in die endgültigen Materialien. Existieren von einem Material aus technischen Gründen mehrere Versionen, sind diese an der Marshallig-Schicht immer noch zu sehen. Eine fachliche Benennung der Funktionen, Attribute und Klasssen wird aber schon eingeführt. Datenstrukturen, die nur als Parametersammlungen dienen, werden als einzelne Parameter und nicht als Material interpretiert. Nicht benutzte Attribute und Funktionen werden von der Marshalling-Schicht versteckt.



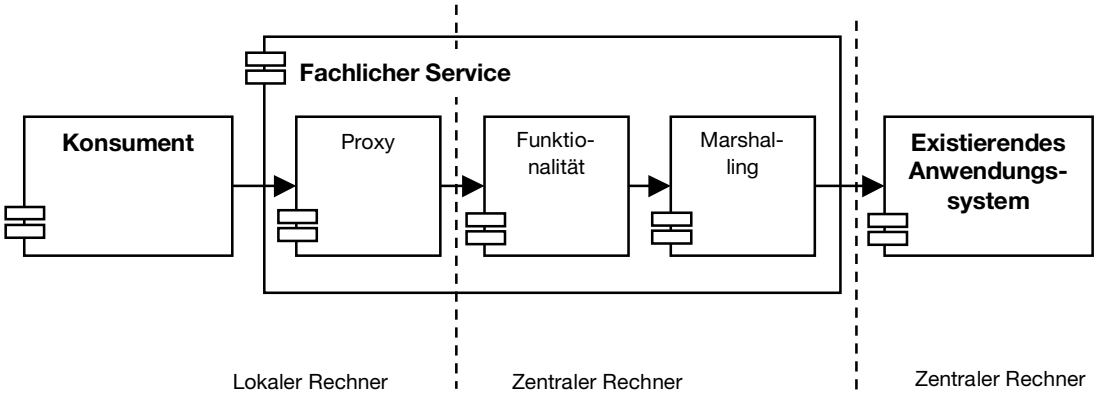
Auf der Marshalling-Schicht setzt dann die Funktionalitäts-Schicht auf, in der Anpassungen der Granularität vorgenommen und die technischen Materialien in die Materialien des Objektmodells umgewandelt werden. Dazu werden beim Start des CAI einige

statische Eigenschaften beim MCC abgefragt, die dann bei Bedarf verfügbar sind. Teilweise werden Materialien auch aus mehreren Funktionsaufrufen akkumuliert.

Darstellung der Architektur

Bei den Komponentendiagramme der UML nennt Fowler innerhalb von Komponenten als Feinstrukturen nur Klassen [Fowler 99]. In den gängigen Tools lassen sich die Komponenten aber beliebig schachteln: Komponenten können hier weitere Komponenten enthalten.

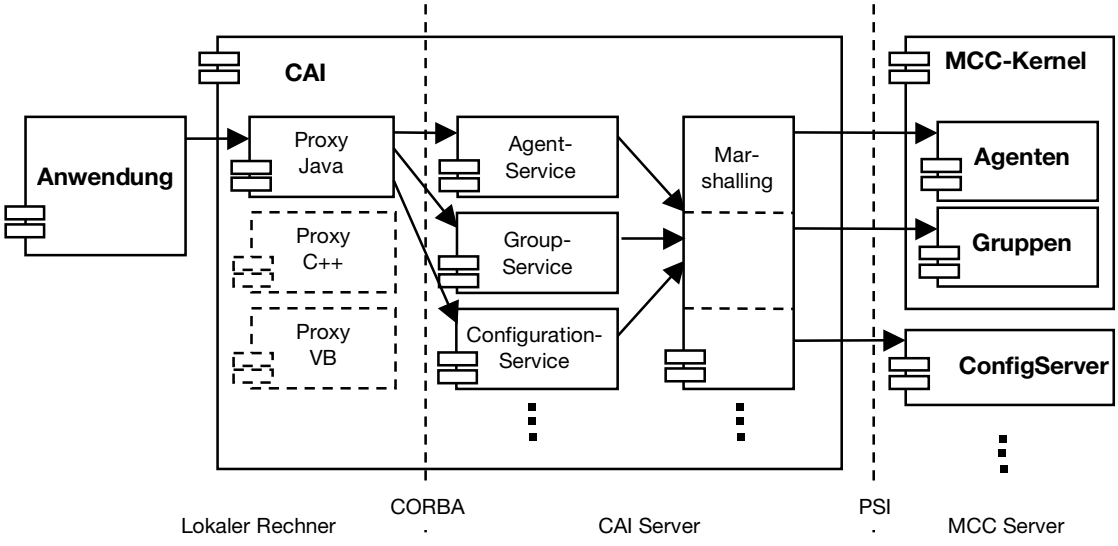
Dies ist auch für hier dargestellten Komponenten sinnvoll: Der Fachliche Service in der letzten Abbildung ist eine Komponente, die aus den Komponenten Proxy, Funktionalität und Marshalling zusammengesetzt ist. Eine Darstellung, bei der diese drei Bestandteile zusammengehören, wird durch die massiven Rechnergrenzen der Verteilungsdiagramme erschwert. Stellt man die Rechnergrenzen nur als dünne Striche dar, könnte das letzte Diagramm so aussehen:



Da diese Darstellung besser die zusammengehörigen Komponenten zeigt, wird sie im folgenden Abschnitt zur Darstellung des CAI verwendet.

Architektur des CAI

Das CAI wurde entsprechend der vorgestellten Architektur zur Anbindung existierender Anwendungssysteme umgesetzt:



Die Anwendung wird auf dem lokalen Rechner durch Proxies unterstützt, die in den Programmiersprachen Java, C++ und Visual Basic verfügbar sind. Die Proxies bieten einen Zugriff auf alle Fachlichen Services des CAI. Sie kapseln die Middleware und greifen intern über CORBA auf die Fachlichen Services des CAI zu.

Die Fachlichen Services bieten die Dienstleistungen als CORBA-Service an. Sie benutzen eine Marshalling-Schicht, die den Zugriff auf das MCC und die Middleware PSI kapselt. Die Marshalling-Schicht ist nach den Schnittstellen des MCC strukturiert. Die Fachlichen Services können beliebige Funktionen aus der gesamten Schicht benutzen, also auch aus unterschiedlichen Schnittstellen des MCC.

In den Fachlichen Services wird die Granularität angepaßt und die Materialien werden vereinheitlicht. Um fehlende Attribute bei Materialien ergänzen zu können, merkt sich das CAI einmalig statische Informationen von MCC-Objekten. Diese Informationen stehen übergreifend für alle Services zur Verfügung.

Fazit

Fachliche Services zur Anbindung existierender Anwendungssysteme spielen in der Architektur die Rolle eines Adapters. Mit der vorgestellten Architektur verteilt sich ein Fachlicher Service auf einen lokalen Proxy, eine zentrale implementierte Funktionalität und eine Marshalling-Schicht, die das existierende Anwendungssystem anbindet.

Bei der Notation mit Komponenten- und Verteilungsdiagramme empfiehlt sich zur Schachtelung von Komponenten eine dünne Markierung der Rechnergrenzen.

Insgesamt erfordern die Fachlichen Services des CAI durch die Verteilung und die Anbindung des MCC eine relativ komplexe Architektur, die den Einsatz einer Middleware voraussetzt.

5.3 Technische Umsetzung

Durch die komplexe Architektur Fachlicher Services stellen sich auch in der Implementierung einige technische Fragen, die aus der Verteilung und der Mehrbenutzerfähigkeit herrühren.

Diese Fragen und mögliche Lösungen werden in diesem Kapitel vorgestellt. Das technische Benutzungsmodell ist immer der Ausgangspunkt für die Lösungen, allerdings lassen sich die technischen Fragen nur teilweise aus diesem Umgang beantworten.

Das Protokoll eines Fachlichen Services

Für die technische Gestaltung eines Fachlichen Service wird im folgenden der Begriff des *Protokolls* des Fachlichen Service benutzt:

> **Definition: Protokoll eines Fachlichen Service**
> Das Protokoll eines Fachlichen Service beschreibt seinen Umgang auf technischer Ebene. Grundlage für die Entwurfsentscheidungen des Protokolls sind die fachlichen Anforderungen, die im technischen Benutzungsmodell beschrieben werden. Das Protokoll wird auch durch die benutzte Infrastruktur beeinflußt.

Einen Überblick über die technischen Fragen, die in einem solchen Protokoll abgedeckt werden, bietet die folgende Checkliste:

- ☐ Durch welche Middleware/Infrastruktur wird die Verteilung umgesetzt? (S. 62)
- ☐ Wie werden Materialien über das Netz versandt? (S. 63)
- ☐ Werden die Services synchron oder asynchron benutzt? (S. 65)

  Im synchronen Fall:

  - ☐ Werden Fehler über Exceptions gemeldet?
  - ☐ Wird Multithreading benutzt, damit der Konsument nicht blockiert ist?

  Im asynchronen Fall:

  - ☐ Wie werden die Callbacks mit anderen Eventquellen synchronisiert?
- ☐ Wie werden Fehler an den Konsumenten gemeldet? Wie werden Verteilungsfehler behandelt? Gibt es Time-Out-Mechanismen? (S. 64)
- ☐ Was sind die Konzepte zur Speicherverwaltung? (S. 65)
- ☐ Wie wird Mehrbenutzerfähigkeit sichergestellt? (S. 68) Welche Operationen sind atomar? (S. 69) Gibt es Transaktionen? (S. 69)
- ☐ Ist der Service zustandslos? (S. 70) Gibt es Sessions? (S. 69)
- ☐ Wie werden fachliche Ereignisse dem Konsumenten mitgeteilt? (S. 71)

Notation des Protokolls

Da diese Fragen den Umgang mit einem Fachlichen Services auf einer sehr niedrigen und technischen Ebene behandeln, wird im folgenden zur Notation des Protokolls di-

rekt CORBA-IDL [vgl. Redlich 96] benutzt. Üblich sind bei Kommunikationsprotokollen auch Darstellungen mit endlichen Automaten:

»Zur *Protokollbeschreibung* werden Automaten verwendet, die die beteiligten Instanzen und den Basisdienst beschreiben. Häufig werden Phasen des Ablaufs (Verbindungsaufbau, -abbau, Datenaustausch) getrennt modelliert. Initiator und Beantworter werden mit verschiedenen, auf diese Funktion reduzierten Automaten beschrieben.«
[Rechenberg & Pomberger 99, S.682]

Da die hier behandelten Services überwiegend zustandslos sind, bietet sich eine solche Darstellung am ehesten noch für Sessions an.

Infrastruktur und Middleware

Einen wesentlichen Einfluß auf die technische Umsetzung hat eine unterliegende Middleware, die als Infrastruktur genutzt werden kann. Dieser Einfluß ist offensichtlich, wenn keine Proxies verwendet werden, sondern sich die Konsumenten direkt über die Middleware an den Service wenden. Aber auch bei der Verwendung von Proxies hat die Auswahl der Middleware Auswirkungen auf das Protokoll: Nur mit zusätzlichem Aufwand können Alternativen umgesetzt werden, die die unterliegende Middleware nicht unterstützt oder die den Konzepten der Middleware sogar widersprechen.

Otto und Schuler diskutieren in ihrer Arbeit den Einsatz von Enterprise Java Beans, CORBA und von SAP Business Objects als Middleware [vgl. Otto & Schuler 2000, S. 54ff und S. 97ff]. In dieser Arbeit wurde zur Konstruktion Fachlicher Services CORBA und die bei Tenovis entwickelte Middleware *Process System Interface* benutzt, die besonders auf Echtzeitanforderungen ausgelegt ist [Kracke 96]. An den Stellen, an denen diese Middlewares die Umgangsformen der Services beeinflussen, wird dies im folgenden gesondert angemerkt.

Kooperation und Verteilung

Das Kooperationsmodell formuliert die fachlichen Vorstellungen, wie im Service mit Verteilung umgegangen werden soll. Die Sicht auf Fachliche Services als gemeinsam genutzte Ressource wird technisch durch Netzwerkzugriffe und Verteilung umgesetzt, bei der Konstruktion Fachlicher Services entstehen zusätzliche technische Fragen. In den folgenden Abschnitten werden zu je einer Designfrage verteilter Services verschiedene Ansätze vorgestellt.

Verteilungstransparenz

Mit dem Stichwort *Verteilungstransparenz* wird das Ziel beschrieben, die technische Verteilung eines Services oder einer Datenbank soweit wie möglich vor dem Konsumenten zu verbergen.

Dittrich schreibt im „Informatik Handbuch“ hierzu: »Datenbanken sollen nicht immer zentral in einem Rechner verwaltet werden, sondern in einem Rechnernetz und damit auch geographisch verteilt sein können. Trotzdem soll die Anwendung meist soweit wie möglich von den Details der Verteilung abgeschirmt bleiben (Verteilungstransparenz)« [Rechenberg & Pomberger 99, S. 901]

Auf die Spitze getrieben bedeutet Verteilungstransparenz, daß entfernte und lokale Aufrufe nicht mehr zu unterscheiden sind. Damit wäre ein verteilter Service genau wie ein lokaler Service zu handhaben. Dieses Ziel ist aus mehreren Gründen zu hinterfragen. So kritisieren Guerraoui und Fayad:

»This is a very misleading approach because, as we will discuss, distribution transparency is impossible to achieve in practice. Precisely because of that impossibility, it is dangerous to provide the illusion of transparency.« »If a failure occurs somewhere between the caller and the callee..., then it is impossible to continue to provide any illusion of transparency...« [Guerraoui & Fayad 99, S. 103]

Schafft man bei einem entfernten Aufruf die Illusion, dieser sei direkt und ohne Netzwerkzugriff möglich, dann stößt dieses Benutzungsmodell an seine Grenzen, wenn Netzwerkstörungen auftreten. Typisches Beispiel sind Anwendungen, die dann eine Sanduhr zeigen und auf keine Eingaben mehr reagieren. Da es unmöglich ist, solche Netzwerkstörungen zu verbergen, ist ein Benutzungsmodell angebracht, das die Netzwerkverbindung explizit behandelt.

Auch aus einem weiteren Grund müssen gemeinsam genutzte Fachliche Services essentiell anders behandelt werden als lokale Dienste: In der kooperativen Arbeit finden sich Besonderheiten, die durch die Verteilung bedingt sind und im Umgang mit dem Service berücksichtigt werden müssen. Wie beschrieben, sollte ein solcher Service mit einem expliziten *Kooperationsmodell* konstruiert werden.

Versand von Materialien

Das Kooperationsmodell orientiert sich an den fachlichen Aufgaben und Abläufen, die im Anwendungsbereich existieren. Eine sich häufig hieraus ergebende Anforderung ist, daß Materialien als Original oder Kopie versendet werden sollen. Ein Beispiel ist der Versand von Materialien unter sondierendem Aspekt, wie er im Kapitel über Fachliche Services diskutiert wurde (Seite 12f.). Gemäß dem Kooperationsmodell befindet sich das Material nach dem Versenden durch den Service lokal beim Konsumenten.

Um das Material über die Netzwerkverbindung zu versenden, gibt es mehrere Ansätze:

- *Der Ansatz „Flache Materialien“*
  Technisch relativ einfach umzusetzen ist der Versand von Materialien, die als flache Datenstruktur angelegt sind. Ein solches Material besteht aus der Ansammlung einzelner Attribute, vergleichbar mit einem s*truct* in C++ oder einem *record* in Pascal. Solche flachen Materialien ermöglichen den Zugriff und die Änderung einzelner Werte, bieten aber im Gegensatz zu einem fachlich modellierten Objekt keine fachlichen Umgangsformen. Ein entsprechend modelliertes Konto würde etwa eine Operation *KontostandSetzen* anbieten, nicht aber fachliche Methoden *BetragEinzahlen* und *BetragAbheben*.

  Problematisch an diesem Ansatz ist, daß ein fachliches Modell des Anwendungsbereichs mit diesen Materialien nicht forciert wird: Die zwischen Service und Konsumenten übertragenen Materialien werden schnell als beliebige Variablenzusammenstellungen betrachtet und nicht mehr als Objekte des Anwendungsbereichs.

- *Der Ansatz „CORBA“*
  CORBA erlaubt das Versenden flacher Materialien, ein Versenden objektorientierter Klassen wird hingegen nicht unterstützt. Wird eine Instanz einer CORBA-Klasse an den Konsumenten geschickt, wird sie nicht wirklich versandt, sondern bleibt am selben Ort. Sie ist jetzt aber vom Konsumenten entfernt zugreifbar. Obwohl der Konsument annimmt, das Material liege – nach dem Kooperationsmodell – lokal vor, ist jeder Zugriff mit einem Netzwerkzugriff verbunden, der vor dem Benutzer verborgen wird.

  Wie bereits im Abschnitt über Verteilungstransparenz (Seite 61) kritisiert, trägt dieses Modell nicht mehr, sobald Störungen und Verzögerungen in der Netzwerkverbindung auftreten. Mit dieser Alternative können Materialien nicht so versandt werden, daß sie wirklich lokal vorliegen.

- *Der Ansatz „Kapselung im Service“*
  Dem Standardansatz von CORBA ähnlich ist die Idee, die Materialien vollständig im Service zu kapseln. Jeder Zugriff auf die Materialien wird an den zentralen Service gerichtet. Nachteil ist, daß auch hier – wie bei CORBA – beim Konsumenten nur die Illusion eines lokal vorliegenden Materials geschaffen werden kann, die endet, sobald Netzwerkstörungen auftreten.

- *Der Ansatz „Memento“*
  Im Entwurfsmuster Memento [vgl. Gamma et al. 96, S. 317] wird das beschriebene Versenden flacher Materialien als Hilfsmittel benutzt, um echte Objekte zu versenden: Der Zustand des Materials wird als abstrakter Zustand extrahiert (serialisiert) und über die Netzwerkverbindung versendet. Beim Empfänger wird das Material wieder zusammengesetzt und bietet wieder alle Umgangsformen.

  Das Entwurfsmuster Memento bietet sich besonders für die Programmiersprache Java an, die eine Serialisierung unterstützt. Bei diesem Entwurfsmuster ist zu beachten, daß die Materialien in sich abgeschlossen sein müssen: Sie dürfen keine programmiersprachlichen Referenzen *(Pointer)* auf weitere Objekte enthalten, da diese Referenzen nach der Übertragung nicht mehr aufgelöst werden können. Eine Lösung sind hier fachliche Referenzen, zum Beispiel eine Kundennummer, über die – z.B. über einen eigenen Fachlichen Service – auch später noch solche Querbezüge aufgelöst werden können.

  Dieser Ansatz unterstützt am besten eine fachliche Modellierung, ist aber auch am aufwendigsten umzusetzen.

Da die Middleware PSI und CORBA den Versand flacher Materialien direkt unterstützten, wurde im CAI-Projekt der Ansatz „Flache Materialien“ verwendet. Die versandten Materialien wurden dabei – soweit möglich – fachlich modelliert und interpretiert. Der Ansatz „Memento“ wurde nicht verfolgt.

Fehlererkennung

Eine weitere allgemeine Eigenschaft von verteilten Services ist die Erkennung von Fehlern während der Übertragung von Daten zum zentralen Service. Diese Fähigkeit kann weitgehend unabhängig vom konkreten Service implementiert werden.

Neben der Erkennung von Verbindungsstörungen können auch Time-Outs sinnvoll sein, bei denen der Konsument dem Proxy angibt, wie lange er auf die Antwort warten will. Im Gegensatz zu echten Verbindungsstörungen kann der Server hier durchaus erreichbar, aber überlastet sein. Ein Time-Out kann dann von Konsumenten wie eine Verbindungsstörung behandelt werden.

Zusätzlich kann der Proxy auch den Wiederaufbau einer Verbindung im Fehlerfall unterstützen (vgl. den Abschnitt Sessions auf Seite 69). Dabei kann es je nach Konsument unterschiedliche Wünsche geben, wie ein Verbindungsabriß behandelt wird. Wie lange und wie oft soll versucht werden, die Verbindung aufzubauen? Fragen wie diese sind nur aus den fachlichen Anforderungen des Konsumenten zu beantworten. Oft dauert es eine gewisse Zeit, bis z.B. der zentrale Service neu gestartet wird. Deshalb sollte der Verbindungsabriß dem Anwender signalisiert werden und kann vom Proxy nicht vollständig vor dem Konsumenten verborgen werden.

Konzepte zur Speicherverwaltung

Auch in Sprachen mit Garbage-Collection, wie Java, werden durch die Verteilung Fragen der Speicherverwaltung wieder aktuell. Dies liegt daran, daß eine verteilte Garbage-Collection technisch kaum umsetzbar ist. Bei CORBA wird ein Service z.B. nicht informiert, wenn der letzte entfernte Client seine Referenz auf den Service gelöscht hat [vgl. Redlich 96]. Dies macht Konzepte zur Speicherfreigabe notwendig. Der Client kann auch explizit mitteilen, daß er den Service nicht mehr benutzt oder regelmäßig mit einem „Heartbeat“ überprüft werden.

Als Faustregel empfiehlt es sich, im Server nicht für jeden Client neue Instanzen zu erzeugen, sondern mit wenigen statischen Objekten zu arbeiten. Im CAI wurde im AgentService für jeden benutzten Agenten nur eine statische Instanz angelegt, so daß unabhängig von der Laufzeit und der Zahl der Clienten die Menge der Objekte überschaubar blieb.

Synchrone und asynchrone Nutzung

Während lokale Programme fast immer synchron arbeiten, ist diese Entscheidung bei verteilten fachlichen Services nicht so einfach. Dieser Abschnitt diskutiert die Vor- und Nachteile synchroner und asynchroner Schnittstellen.

Synchrone Funktionsaufrufe

Werden Funktionsaufrufe synchron abgearbeitet, ist der Aufrufende blockiert, bis er eine Antwort erhält, er kann nach dem Aufruf direkt mit der Antwort weiterarbeiten.

Wird von einem Konsumenten eine Funktion eines verteilten Fachlichen Service aufgerufen, dann verbirgt sich hinter diesem Aufruf ein Netzwerkzugriff. Dieser Netzwerkzugriff kann spürbar länger dauern als ein lokaler Aufruf, zum Beispiel bei Fehlern, ausgelasteten Services oder fernen Verbindungen. Benutzt der Konsument nun das übliche synchrone Programmiermodell, dann ist seine Anwendung während dieser Zeit blockiert, der Anwender kann weder weiterarbeiten noch den Netzwerkzugriff abbrechen.

Fehler werden in solchen synchronen Programmen oft über *Exceptions* gemeldet. Trifft eine Fehlermeldung ein, wird das Programm nicht direkt hinter dem Aufruf fortgesetzt, sondern dort, wo die Exception gefangen wird.

```
function () {
  try
  {
    result = service->doSomethingSync();
  }
  catch (Exception serviceException)
  {
    // handle error
  }

  // ... work with result
}
```

Synchrones Modell

Abhängig von der benutzten Middleware kann es längere Zeit dauern, bis der Fehler erkannt wird. Will man vermeiden, daß der Konsument währenddessen für den Benutzer blockiert ist, muß man mit Multithreading arbeiten: Die Anfrage an den Service wird von einem eigenem Thread gestartet, der unabhängig vom restlichen Programm arbeitet.

Asynchrone Funktionsaufrufe

Die Alternative ist ein asynchrones Modell: Der Aufruf einer Methode des Service liefert nicht sofort ein Ergebnis, vielmehr gibt beim Aufruf der Konsument sich selbst als Rückrufadresse an. Sobald das Ergebnis eingetroffen ist, wird ein Callback des Konsumenten gerufen; ein eigener Callback wird gerufen, um über Verbindungsstörungen zu unterrichten. So läuft das aufrufende Programm während der Bearbeitung der Anfrage weiter, es können andere Arbeiten durchgeführt werden.

Die Arbeit mit asynchronen Aufrufen ist allerdings ungewohnt: Statt eines linearen Kontrollflusses wird der Kontrollfluß unterbrochen und in der Callbackmethode fortgesetzt.

```
function ()
{
  service->doSomethingAsync();
}

// callback:
handleResponse (Result result)
{
  // ... work with result
}
```

```
// callback:
handleError (Error error)
{
  // ... handle error
}
```

Asynchrones Modell

Die Benutzung ähnelt der Nutzung moderner Fensterbibliotheken: Auch hier wird asynchron über Benutzereingaben informiert, der Kontrollfluß wird von einer Eventloop gesteuert, die außerhalb der Kontrolle des Benutzers liegt.

Ein Vorteil des asynchronen Modells ist, daß hier nicht aktiv gewartet wird: Der Konsument ist nie blockiert, während er auf den Service wartet.

Implementierung Asynchroner Schnittstellen

Bei der Implementierung ist zu beachten, daß durch das asynchrone Programmiermodell im Konsumenten eine neue Ereignisquelle existiert: Neben z.B. GUI-Ereignissen des Benutzers können jetzt auch Service-Ereignisse wie Antworten und Fehlermeldungen eintreffen. Diese beiden Eventquellen sollten synchronisiert werden, so daß nur ein Ereignis zur Zeit abgearbeitet wird.

Hilfreich ist hier die Implementierung des asynchronen Programmiermodells mit einem in der Programmiersprache oder Middleware vorhandenen Ereignis-Modell. Dies ist z.B. in Visual Basic möglich. Trifft die Antwort ein, wird sie mit einem solchen Ereignis dem Konsumenten mitgeteilt. Die Ereignisse in Visual Basic sind auch automatisch mit den GUI-Ereignissen synchronisiert.

Wandlung zwischen synchronen und asynchronen Schnittstellen

Eine synchrone Schnittstelle kann mit etwas Aufwand in eine asynchrone umgewandelt werden und umgekehrt. Um eine synchrone Schnittstelle asynchron zu nutzen, muß bei jedem Aufruf ein eigener Thread gestartet werden, der dann blockiert ist, während das Hauptprogramm weiterläuft. Auch hier trifft wie bei einer asynchronen Lösung die Antwort in einem eigenen Thread ein.

Will man eine asynchrone Schnittstelle synchron nutzen, muß das Hauptprogramm in einer Schleife blockieren, bis der Callback aus dem Eventthread mitteilt, daß die Antwort eingetroffen ist.

Bewertung

Das asynchrone Modell bietet auf den ersten Blick technisch einige Vorteile: Der Aufrufer muß nicht aktiv warten, seine Anwendung ist bei Störungen nicht blockiert. Allerdings wird dies mit einer deutlich komplexeren Schnittstelle und Programmierung erkauft: Beim MCC wurde in der Vergangenheit bereits die Erfahrung gemacht, daß die Konzepte asynchroner Programmierung für Integratoren einen großen Einarbeitungsaufwand bedeuten. Eine synchrone Schnittstelle ist einfacher zu benutzen. Deshalb wurden im CAI synchrone Schnittstellen benutzt, diese empfehlen sich für alle Services, die von externen Entwicklern mit wenig Einarbeitungszeit benutzt werden sollen.

Mehrbenutzerfähigkeit

Während eine Arbeitsplatzsoftware oft nur von einem Benutzer verwendet wird, muß ein verteilter Service in der Regel auch mehrbenutzerfähig sein, damit seine verschiedenen Nutzer an verschiedenen Arbeitsplätzen ihn auch gleichzeitig verwenden können.

> **Definition: Mehrbenutzerfähig**
> Ein zentraler Service, der kooperative Arbeit unterstützt, wird von mehreren Konsumenten benutzt, oft gleichzeitig. Wenn ein Service für eine solche gleichzeitige Nutzung konstruiert wurde und diese unterstützt, ist er *mehrbenutzerfähig*.

Wie Otto und Schuler schreiben, ist die Mehrbenutzerfähigkeit ein Unterschied zwischen klassischen WAM-Werkzeugen und Fachlichen Services:

»Anwendungen, die nach dem Werkzeug & Material-Ansatz entwickelt wurden, laufen üblicherweise auf dem Desktop-Rechner eines Anwenders ab, werden also von nur einer Person zur Zeit benutzt. Wollte man nun mehrere Benutzungsschnittstellen einfach dadurch unterstützen, daß man die Funktionskomponenten der Werkzeuge aus dem WAM-Ansatz auf dem Server laufen läßt, so würde dies auch deshalb scheitern, weil die Funktionskomponenten nicht auf die gleichzeitige Benutzung durch mehrere Nutzer ausgelegt sind.« [Otto & Schuler 2000, S. 77]

Mehrbenutzerfähigkeit reicht weiter als das bisher betrachtete Kooperationsmodell: Beim parallelen Zugriff mehrerer Benutzer entstehen technische Fragen, die sich nur teilweise aus dem fachlichen Kooperationsmodell beantworten lassen. Das Kooperationsmodell gibt aber einen Rahmen vor, in dem die technischen Entscheidungen getroffen werden. In diesem Sinne wird das Kooperationsmodell von Otto und Schuler auch als *Fachliche Mehrbenutzerfähigkeit* bezeichnet.

Sinnvoll ist eine Orientierung am Kooperationsmodell und den fachlichen Umgangsformen beim Umgang mit Dokumenten. Ein Dokument kann ausgeliehen sein und gerade bearbeitet werden und damit für andere Benutzer gesperrt sein, es kann zurückgegeben, weitergereicht und kopiert werden.

Die folgenden darüber hinausgehenden Strategien zum Zugriff mehrerer Benutzer sind technisch geprägt. Sie behandeln Fragen, die sich im fachlichen Anwendungsbereich nicht stellen.

»Eine Software, die mehrbenutzerfähig auf einem Server laufen soll, muss konsistent auf ihre Daten zugreifen können, auch wenn sie nebenläufig von mehreren Benutzern verwendet wird. Dies schließt insbesondere Transaktionssicherheit ein: Eine Methode, die den Zustand eines Materials sondiert, muss innerhalb einer Transaktion mit einer Methode, die das Material auf Grund des Ergebnisses dieser Sondierung verändert, ausführbar sein. Diese Anforderung ist technisch alles andere als trivial lösbar.« [Otto & Schuler 2000, S. 77f]

Die nächsten Abschnitte behandeln die technischen Konzepte rund um Mehrbenutzerfähigkeit: Atomare Operationen, Transaktionen, Sessions und zustandslose Services.

Atomare Operationen

Benutzen mehrere Konsumenten Operationen eines mehrbenutzerfähigen Services, dürfen sich diese nicht stören: Das Ergebnis der Operation soll so erscheinen, als wäre der Konsument zu diesem Zeitpunkt der einzige Benutzer des Services. Die einzelnen Operationen eines solchen Services werden als *atomar* bezeichnet.

Im MCC und CAI wird ein einfacher Weg benutzt, um in den mehrbenutzerfähigen Services atomare Operationen zu garantieren: Die Aufrufe der Konsumenten werden über eine Warteschlage serialisiert und dann nacheinander ausgeführt.

Transaktionen

Manchmal ist es fachlich notwendig, mehrere Operationen zusammen durchzuführen, erst nach all diesen Operationen wird ein stabiler, konsistenter Zustand erreicht. Mit *Transaktionen* können mehrere Operationen zusammengefaßt werden. Dabei werden die folgenden ACID-Eigenschaften garantiert:

- *Atomar:* Transaktionen werden als Einheit durchgeführt, es werden alle oder keine Operationen ausgeführt.
- *Konsistenz (Consistence):* Die Daten sind nach der Transaktion in einem konsistenten Zustand
- *Isolation:* Gleichzeitig ablaufende Transaktionen mehrerer Benutzer haben keinen Einfluß aufeinander
- *Dauerhaftigkeit:* Das Ergebnis der Transaktion bleibt dauerhaft erhalten

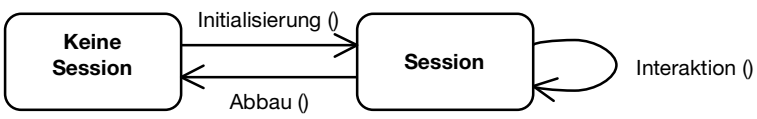
Transaktionen werden zum Beispiel bei datenintensiven Aufgaben benötigt, wenn an mehreren Stellen Änderungen vorgenommen werden, die nur zusammen Sinn ergeben. Ein Beispiel sind in vielen Systemen Änderungen an der Konfiguration, bei der an mehreren Stellen Einträge vorgenommen werden müssen, damit wieder eine konsistente Konfiguration vorliegt.

Für die mit dem CAI zu erledigenden Aufgaben spielten Transaktionen keine Rolle.

Sessions

Eine Session ist eine andauernde logische Verbindung zwischen Konsument und Service, beispielsweise wenn der Konsument sich am Anfang einer Session authentifiziert. Während einer Session ist der Konsument dem Service als Kommunikationspartner bekannt, der Service merkt sich zu jedem Konsumenten Informationen. Im Gegensatz zu Transaktionen werden die einzelnen Operationen einer Session nicht im Block durchgeführt.

Ein Merkmal von Sessions ist, daß am Anfang und am Ende zusätzliche Informationen zwischen Service und Konsument ausgetauscht werden.



Beim CAI wurden Sessions für aktive Services benutzt: Solange die Session läuft, bekommt der Konsument vom Fachlichen Service Informationen über Zustandsänderungen.

Zustandslose Services

Services können sich Informationen über die mit ihnen kommunizierenden Konsumenten merken: Welche Operationen wurden als letztes von diesem Konsumenten durchgeführt? Welche Einstellungen hat dieser Konsument gewählt? Diese Informationen über den Konsumenten bleiben auch zwischen den einzelnen Operationsaufrufen des Konsumenten erhalten.

Ein Service, der darauf verzichtet und sich nach dem Abschluß der vom Konsumenten aufgerufenen Operation keine Informationen über den Konsumenten mehr merkt, wird als *zustandslos* bezeichnet. Ein Service, der Transaktionen oder Sessions benutzt, ist *nicht* zustandslos, da hier die Transaktion oder Session mehrere Anfragen des Konsumenten als Klammer zusammenfaßt.

> **Definition: Zustandslos**
> Ein Service, der sich zwischen den einzelnen vom Konsumenten aufgerufenen Operationen keine Informationen über den Konsumenten merkt, wird als zustandslos bezeichnet.

Das Wort zustandslos ist insofern irreführend, als auch ein zustandsloser Service zwischen den Sessions seinen eigenen Zustand behält, Informationen zu den Konsumenten merkt er sich aber nur während der Bearbeitung eines Operationsaufrufs.

Im „Informatik Handbuch“ gibt Borrmann ein Beispiel für einen zustandslosen Service: »Viele Realisierungen verteilter Dateisysteme halten im Server Zustandsinformationen über die von den Clients geöffneten Dateien. Gehen diese Informationen durch einen Systemausfall verloren, so hat das Rückwirkungen auf den Client. Um solche Rückwirkungen zu vermeiden, wurde *NFS* nach dem Schema des *zustandslosen (stateless) Servers* realisiert. Hier ist jede Operation in sich abgeschlossen, das Öffnen und Schließen von Dateien entfällt. Nachteil ist ein semantisch anderes Verhalten, verglichen mit einem lokalen Dateisystem. So ist dem Server nicht bekannt, welche Dateien auf den Clients bearbeitet werden. Eine automatische Synchronisation konkurrierender Zugriffe oder die Implementierung eines strengen Kohärenzprotokolls sind unmöglich.« [Rechenberg & Pomberger 99, S. 667]

Aus diesem Beispiel wird noch einmal deutlich, daß auch ein zustandsloser Service wie das NFS seinen eigenen Zustand behält, in diesem Fall den Zustand des Dateisystems. Für die Konsumenten bedeutet der zustandslose NFS-Service eine zusätzliche Verantwortung: Sie müssen sich selbst Handles auf die von ihnen geöffneten Dateien merken. Dafür ist das Kommunikationsprotokoll des Services einfacher, da Dateien nicht mehr geöffnet oder geschlossen werden müssen.

Ein weiteres Beispiel liefert der Vergleich von *Hypertext Transfer Protocol* (HTTP) und *File Transfer Protocol* (FTP): Während die einzelnen Seitenaufrufe bei HTTP zustandslos sind und unabhängig voneinander erfolgen, steht bei FTP am Anfang eine Authentifizierung, die einzelnen Anfragen sind hier in einer Session verbunden. Auch über ein

zustandsloses Protokoll wie HTTP können aber Sessions realisiert werden, indem der Konsument bei jedem Aufruf eine Session-Id mitgibt: Dann ist die unterliegende Technologie zwar zustandslos, der darauf aufsetzende Service aber nicht.

Ein Vorteil zustandsloser Services ist die Möglichkeit, die Last auf mehrere gleichartige Services aufzuteilen: Die Anfrage eines Konsumenten kann immer von dem Service bearbeitet werden, der am wenigsten ausgelastet ist. Die Services müssen natürlich untereinander ihren eigenen Zustand koordinieren.

Im Rahmen des CAI wurden Services entwickelt, die bei der Steuerung und Sondierung zustandslos arbeiten. Die Registrierung für Zustandsänderungen bei aktiven Services hingegen benutzt Sessions und ist nicht zustandslos.

Umsetzung aktiver Services

Aktive Services informieren interessierte Konsumenten über Ereignisse und Zustandsänderungen, solche Ereignisse werden auch als *Events* bezeichnet. Die Ereignisse sind dabei in aller Regel fachlich interpretierbar und werden dem Benutzer oft direkt signalisiert. Tritt ein Ereignis ein, werden die Konsumenten gerufen, die sich dafür registriert haben (Pushing).

Agentenmonitor: Der Telefonzustand wird direkt signalisiert

Wie im Kapitel 5.1 diskutiert wurde, sind aktive Services von Webtop-Konsumenten nur schlecht zu nutzen. Für diese Konsumenten muß neben der hier beschriebenen Lösung oft noch eine Polling-Schnittstelle angeboten werden, die alternativ zu den aktiven Services nutzbar ist.

Aktive Services sind in der Regel mehrbenutzerfähig: Mehrere Konsumenten können sich für die gleichen Ereignisse anmelden und werden informiert. Da Service und Konsument lose gekoppelt sein sollen, darf der Service beim Mitteilen eines Ereignisses so wenig Annahmen wie möglich über den Konsumenten machen. Offen bleibt, welche Informationen vom Konsumenten zur Behandlung des Ereignisses benötigt werden. Ein Ansatz ist hier, den Konsumenten nur abstrakt über das Ereignis zu informieren, so daß er die nötigen Informationen am Service über die allgemeinen sondierenden Funktionen abfragen kann.

Dieser Ansatz stößt aber bei verteilten mehrbenutzerfähigen Services an seine Grenzen, da hier nicht garantiert werden kann, daß sich der Zustand bis zur Abfrage nicht geändert hat. In einer solchen Umgebung ist es üblich, direkt mit der Ereignismitteilung begleitende Informationen und Materialien zu senden, so daß möglichst alle Konsumenten die benötigten Informationen erhalten. Die Materialien werden als Schnappschuß, als Kopie des aktuellen Zustands verstanden.

Um aktive Services umzusetzen, bietet sich das Konzept der Sessions an: Als Start einer Session meldet sich ein Konsument beim Service an und erhält während der Session Informationen über Ereignisse. Entsprechend gibt der Konsument beim Start der Session diverse Callbacks an, die bei den unterschiedlichen Ereignissen gerufen werden. Beim Start der Session wird weiter angegeben, für welche Ereignisse sich der Konsument interessiert. Bei einigen Services kann auch die gewünschte Qualität der Beobachtung angegeben werden: Sollen alle Ereignisse sofort mitgeteilt werden oder reicht ein Update in regelmäßigen Abständen? Eine Session wird vom Konsumenten explizit beendet. Sollte der Service eine Session beenden, wird diese Störung dem Konsumenten als Fehler-Ereignis mitgeteilt.

Die Ereignisse aktiver Services können mit derselben Technik umgesetzt werden wie die Antworten bei asynchronen Services und z.B. über Events oder Beobachter- und Schablonenmuster realisiert werden. Auch hier entsteht eine neue Ereignisquelle, die mit den bisherigen Ereignisquellen synchronisiert werden muß. Selbst bei einem ansonsten synchronen Umgang wird man Ereignisse bevorzugt asynchron senden, um so das Polling zu vermeiden.

Ereignisse, wie sie aktive Services und Fenstersysteme liefern und asynchrone Antworten sollten aber nicht gleichgesetzt werden. Im Programm werden diese Aufrufe zwar technisch gleich behandelt, sie sind aber fachlich unterschiedlich: Ein aktiver Service teilt ein fachliches Ereignis des Anwendungsbereichs mit. Asynchrone Antworten stellen hingegen kein Ereignis des Anwendungsbereichs dar, sondern sind nur die Antwort auf eine Anfrage, die über die Netzwerkverbindung auch verzögert erfolgen kann. Dieser Unterschied kann durch eine Namenskonvention in der Benennung der Callback-Klassen und -Methoden deutlich gemacht werden.

Aktive Services wurden im CAI bei allen Services benutzt, die über Zustandsänderungen von MCC-Komponenten informieren.

Unterstützung der Umsetzung in Rahmenwerken

Sehr hilfreich kann die Umsetzung der besprochenen Eigenschaften von Fachlichen Services in einem Rahmenwerk sein, das dann aktive Services oder die Fehlererkennung direkt unterstützt. Dies hat mehrere Vorteile: Zum einen wird die Entwicklung der Fachlichen Services vereinfacht, da allgemeine Funktionalität nicht mehrmals implementiert werden muß; zum anderen wird eine einheitliche Gestaltung der Services unterstützt. Im CAI-Projekt wurden entsprechende Klassen entwickelt, die eine Basis für die Fachlichen Services bildeten.

Für viele allgemeine Konzepte – z.B. die Fehlererkennung – gibt es bei der Implementierung im Rahmenwerk mehrere mögliche technische Umsetzungen. Die Entscheidung für eine Alternative folgt den generellen Anforderungen des Anwendungsbereichs, ist aber stark von technischen Gesichtspunkten – wie der unterliegenden Middleware – geprägt.

Interessant ist ein Ansatz von Felber, Guerraoui und Fayad, der die Entscheidung für eine Umsetzungsalternative vermeidet. In ihrem Artikel »Putting OO Distributed Programming to Work« untersuchen sie exemplarisch die Fehlererkennung verteilter Services und stellen verschiedene Alternativen vor. Sie kommen zu dem Schluß, daß in

einer objektorientierten Modellierung von Services diese Fehlererkennung für jeden Service konfigurierbar sein sollte:

»When following an OO modeling style, failure detectors should be considered first-class citizens. That is, failure detection should be viewed as an abstraction, the complexity of which is encapsulated behind well-defined interfaces. The various roles of a failure detection service should all be represented by first-class objects. As a consequence, one can reuse existing failure detection protocols as they are, or define new ones through composition or refinement.« [Felber, Guerraoui & Fayad 99, S. 98]

Dieser Idee entsprechend würde man die Unterstützung Fachlicher Services im Rahmenwerk modular entwerfen, so daß verschiedene Alternativen austauschbar und erweiterbar sind. Dadurch wären die verschiedenen Fachlichen Services weniger durch die Basis eingeschränkt, allerdings auch weniger einheitlich.

Fazit

Die Gestaltung des Protokolls eines Fachlichen Services erfordert fundierte Kenntnisse diverser Techniken verteilter Systeme. Der Einsatz einer modernen Middleware ist die Grundlage für eine effiziente Implementierung, hilft aber nur bedingt bei den Designentscheidungen.

Eine Unterstützung bei der technischen Umsetzung Fachlicher Services ist sehr wünschenswert. Die flexible Umsetzung in einem Rahmenwerk stellt ein interessantes Forschungsfeld für zukünftige Arbeiten dar.

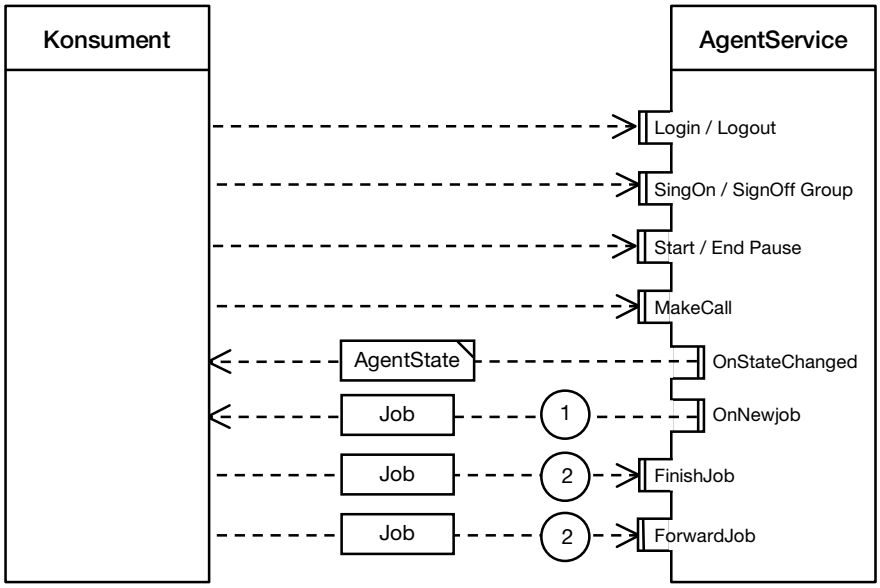
5.4 Fachliche Services des CAI

Dieser Teil stellt den AgentService des CAI vor, als Beispiel für einen Fachlichen Service, der nach dem in dieser Arbeit dargestelltem Vorgehensmodell entwickelt wurde. An diesem Service werden die bisher besprochenen Konzepte noch einmal anschaulich verdeutlicht.

Der AgentService

Das MCC verwaltet Agenten, die Stellvertreter der Sachbearbeiter sind. Diese Agenten modellieren einen abstrahierten Zustand des Sachbearbeiters („frei“, „beschäftigt“, „in Pause“) und seines Telefons („aufgelegt“, „abgehoben“, „Gespräch“).

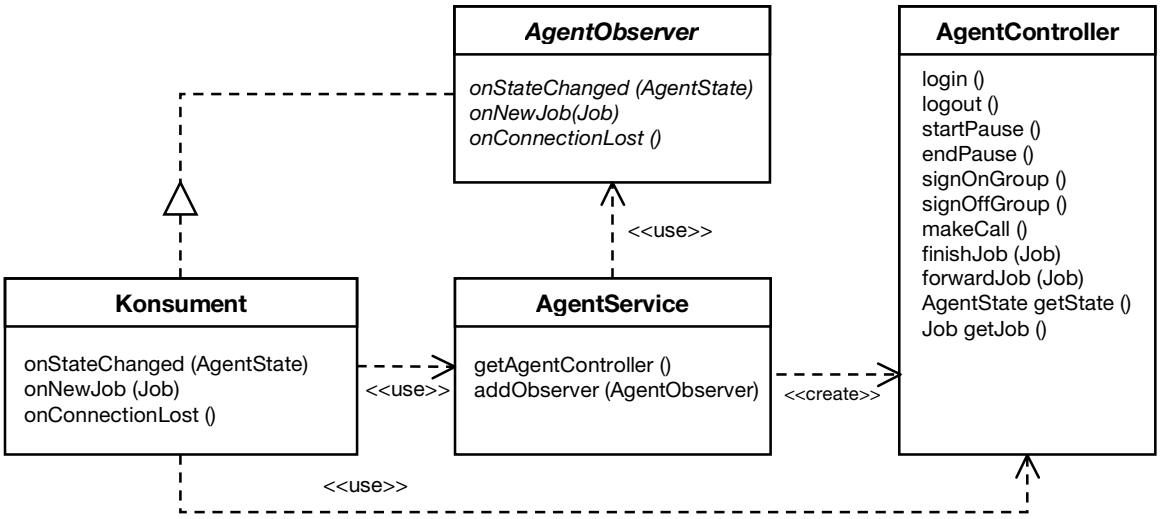
Der AgentService bietet alle Dienstleistungen, um mit diesen Agenten zu arbeiten und setzt dabei das technische Benutzungsmodell um, das in Kapitel 4.4 vorgestellt wurde:



Über den AgentService kann

- mit der Klasse `AgentObserver` ein Agent beobachtet werden. Der Benutzer wird so über Zustandswechsel und eingehende Jobs informiert.
- mit der Klasse `AgentController` ein Agent gesteuert werden, z.B. eine Pause begonnen oder die Jobbearbeitung beendet werden.

Das folgende Klassendiagramm bietet einen Überblick:



Im Vergleich zum technischen Benutzungsmodell fällt auf, daß das Klassendiagramm komplexer ist: Hier sind Implementationsdetails wie die Aufteilung auf Klassen und die Umsetzung aktiver Services deutlich zu sehen. Dafür sind die Umgangsformen des Service – die Ereignisse, der Austausch von Materialien, die Reihenfolge der Interaktion bei der Job-Beendigung – in dieser Darstellung nicht mehr direkt zu erkennen.

AgentService

Die Klasse AgentService bietet mit den Methoden `addObserver` und `getAgentController` den Einstieg in den fachlichen Service:

```
interface AgentService
{
  ObserverHandle addObserver (in ID agent, in AgentObserver observer)
    throws ServerNotReachable;

  AgentController getAgentController (in ID agent)
    throws ServerNotReachable;
};
```

`AgentService`

Mit der `agentID` wird angegeben, welcher Agent gesteuert bzw. beobachtet werden soll. Sind der CAI-Service bzw. das MCC nicht erreichbar, wird die Exception `ServerNotReachable` geworfen.

AgentObserver

Mit der Methode addObserver wird eine neue Session begonnen, in der ein Konsument Informationen über Zustandsänderungen und eingehende Jobs des Agenten bekommt. Dazu muß er das Interface `AgentObserver` implementieren. Um die Sessi-

on zu beenden, ruft der Konsument am `ObserverHandle` die Methode `removeObserver` auf.

```
struct AgentState
{
  AgentStateCode state;                    // The state of the agent
  TelephoneStateCode telephoneState;       // Telephone state
  SeqOfGroupConnections groupConnections;  // All groups of the agent
  string name;                             // The name of the agent
  string telephone;                        // Agents telephone number
};

struct Job
{
  string            callingNumber;
  string            calledNumber;
  boolean           inbound;
  JobResult         jobResult_;
  ApplicationData   applicationData_;

};

interface AgentObserver
{
  void onStateChanged   ( in AgentState state );
  void onNewJob         ( in Job newJob );
  void onConnectionLost ();
}
```

`AgentObserver`

Bei jeder Zustandsänderung erhält der Konsument eine aktuelle Kopie des Agentenzustands. Diese enthält

- den Zustand des Agenten,
- Zustand und Telefonnummer seines Telefons,
- die Gruppen, bei denen der Agent an- und abgemeldet ist,
- den Namen des Agenten.

Bekommt der Agent einen neuen Job zugeteilt, dann erhält der Beobachter diesen Job. Der Job enthält alle Informationen zu einem Anruf:

- Die Telefonnummer des Anrufers (falls von der Telefonanlage ermittelt).
- Die angerufene Telefonnummer der Hotline.
- Eine Information, ob der Kunde angerufen hat oder ob der Anruf im Rahmen einer Telefonkampagne ausgehend war.
- Ein Attribut jobResult, mit dem das Ergebnis des Anrufes bewertet werden kann (positiv/negativ).
- Ein Feld ApplicationData, mit dem im Job zwischen Sachbearbeitern Informationen weitergeleitet werden können.

Ist die Verbindung zwischen Proxy und CAI bzw. MCC gestört, wird der Callback `onConnectionLost` gerufen.

AgentController

Der `AgentController` bietet die steuernden und sondierenden Funktionen:

- `startPause, endPause`
  Pause beginnen und beenden
- `signOnGroup, signOffGroup`
  Agenten in einer Gruppe anmelden (Agent will Jobs für diese Gruppe bearbeiten)
- `finishJob`
  Job zurückgeben (der Job ist fertig bearbeitet)
- `forwardJob`
  Job an einen anderen Sachbearbeiter weiterleiten
- `makeCall`
  Eine Telefonnummer anrufen (erzeugt einen neuen Job)
- `getState`
  Den Zustand des Agenten abfragen
- `getJob`
  Den aktuellen Job des Agenten abfragen

```
interface AgentController
{
  void login (
    in string  telephone,
    in boolean callcenterMonitorsTelephone,
    in boolean startInPause)
    throws NoLicencesAvailable, ServerNotReachable;

  void logout       ()                      throws ServerNotReachable;
  void finishJob    (in Job aJob)           throws ServerNotReachable;
  void startPause   ()                      throws ServerNotReachable;
  void endPause     ()                      throws ServerNotReachable;
  void signOnGroup  ( in ID groupId )       throws ServerNotReachable;
  void signOffGroup ( in ID groupId )       throws ServerNotReachable;
  void forwardJob   ( in Job aJob, in ID agentId ) throws ...
  void makeCall     ( in string targetTelephone )  throws ...

  AgentState getState ()                    throws ServerNotReachable;
  Job        getJob   ()                    throws ServerNotReachable;
};
```

AgentController

Die Methode `login` benutzt den im Kapitel 4.2 vorgestellten MCC-Dialog `Am20_3`. Hier ist noch einmal deutlich zu sehen, wie die Parameter fachlich benannt und viele unwichtigen Details verborgen wurden.

Mit der Methode `finishJob` wird der Job an das MCC zurückgegeben, der dem Konsumenten in der Methode `onNewJob` übergeben wurde.

Konsumenten werden normalerweise über den `AgentObserver` über Zustandsänderungen informiert. Die sondierenden Funktionen `getState` und `getJob` geben darüber hinaus Konsumenten wie Webtops, die nicht mit aktiven Services umgehen können, eine Möglichkeit den Zustand des Agenten auch zu pollen (vgl. Kapitel 5.1).

Fazit

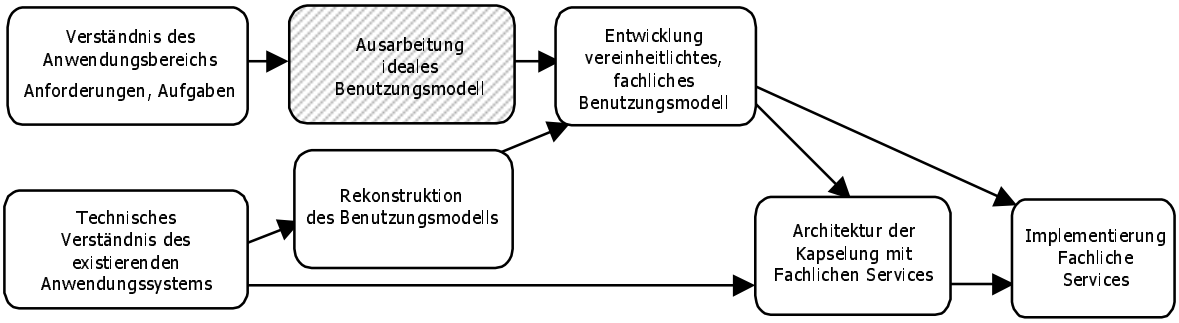
Im AgentService lassen sich einige bisher besprochenen Konzepte wiederfinden:

- Es wird in allen Services ein gemeinsames Objektmodell benutzt, von dem hier der Job und der AgentState vorgestellt wurden.
- Der Job wird als Kooperationsmittel vom Konsumenten wieder zurückgegeben, der Konsument kann Änderungen am Job vornehmen.
- Materialien werden als flache Strukturen modelliert, da dieses von CORBA direkt unterstützt wird.
- Die Funktionen sind synchron, die Ereignisse des aktiven Service werden als asynchrones Event gesendet.
- Verbindungsstörungen werden über Exceptions mitgeteilt, beim aktiven Service über ein asynchrones Ereignis.

Es stehen im AgentService die wesentlichen Funktionen zur Verfügung, um einen Agentenmonitor oder First-Screen zu implementieren. Die Schnittstelle ist dabei sehr übersichtlich und an den fachlichen Begriffen und Aufgaben orientiert.

6 Fazit

In dieser Diplomarbeit wurde ein Vorgehensmodell für die Anbindung und Kapselung existierender Anwendungssysteme vorgestellt. Ziel der Kapselung ist die Entkopplung der neuen Anwendung von der Technologie und dem Benutzungsmodell des existierenden Anwendungssystems. Die Kapselung ist eine Voraussetzung für eine spätere Migration des Anwendungssystems zu einem neu entwickelten System. Eine solche Migration wurde in dieser Arbeit aber nicht untersucht.



Das Vorgehensmodell

Am Anfang des Vorgehensmodells steht die Analyse der aktuellen Anforderungen der Benutzer an die neue Schnittstelle. Parallel dazu wird aus dem technischen Verständnis des existierenden Systems das existierende Benutzungsmodell rekonstruiert.

Auf Basis der Anforderungen kann ein ideales Benutzungsmodells detailliert ausgearbeitet werden. Das bringt eine weitere Entkopplung der neuen Schnittstelle vom existierenden Anwendungssystem, bedeutet aber auch viel Aufwand bei der Entwicklung und Umsetzung.

Dann werden die Anforderungen und das existierende Benutzungsmodell gegenübergestellt: Läßt sich ein auf Basis des existierenden Anwendungssystems ein einheitliches fachliches Benutzungsmodell finden, das die aktuellen Anforderungen erfüllt?

Findet sich ein solches Modell, wird die Architektur der Kapselung entwickelt. Die passende Architektur ergibt sich aus den Anforderungen der Konsumenten und aus dem Aufbau des existierenden Systems. Mit diesen Ergebnissen werden die Fachlichen Services implementiert.

Erfahrungen mit dem Vorgehensmodell

Das Vorgehensmodell zur Kapselung existierender Anwendungssysteme wurde in einem praktischen Projekt erprobt. Dabei zeigte sich, daß die einzelnen Phasen des Vorgehensmodells ganz unterschiedliche Herausforderungen stellen.

Die Einarbeitung in das existierende System erforderte viel Detail- und Fleißarbeit, oft mit Zuhilfenahme der Quelltexte. Im CAI-Projekt waren die Schnittstellen des MCC nur minimal dokumentiert, so daß für die einzelnen Funktionen und Parameter die Bedeutungen und möglichen Werte umständlich bestimmt werden mußten.

Für die Ausarbeitung der Anforderungen hingegen war ein Abstand zum existierenden System notwendig. Wichtig war hier die Aufgabenorientierung und eine klare Zielgruppe, um so die wirklichen Bedürfnisse und die vom existierenden Anwendungssystem benötigte Funktionalität zu finden.

Da im CAI-Projekt ein ideales Benutzungsmodell nicht explizit ausgearbeitet wurde, konnten dessen Vor- und Nachteile in dieser Diplomarbeit nicht untersucht werden. Ein zukünftiges Projekt kann hier weitere Erfahrungen sammeln – z.B. im Rahmen einer Migration, bei der das ideale Benutzungsmodell die technische Systemvision des zukünftigen Anwendungssystems darstellt.

Die kritische Stelle im Vorgehen ist die Entwicklung eines vereinheitlichten Benutzungsmodells, da sich nicht zu jedem existierenden Anwendungssystem ein befriedigendes einheitliches Benutzungsmodell finden läßt. Im CAI-Projekte wurde ein fachliches, in sich schlüssiges und komfortables Benutzungsmodell gefunden. Dieser Schritt verlangte Abstraktionsfähigkeit und den Mut zum Weglassen.

Durch die Aufgabenorientierung und die klare Zielgruppe umfaßt die CAI-Schnittstelle nur einen Bruchteil der vom MCC angebotenen Funktionalität, der Schlüssel zum Erfolg war diese schlanke Schnittstelle. Besondere Schwierigkeiten bestanden darin, ein einheitliches Objektmodell aufzubauen, dazu war häufig trickreiche Programmierung notwendig.

Fachliche Services haben sich als Entwurfsmetapher für die Kapselung existierender Anwendungssysteme bewährt. Sie unterstützen die Aufteilung großer Systeme, fokussieren dabei die fachliche Modellierung und bieten eine gute Unterstützung der Verteilung.

Für die technischen Benutzungsmodelle Fachlicher Services wurde eine neue Notation eingeführt, die einen guten Überblick über die Schnittstelle einer Softwarekomponente gibt. Hier sind die Umgangsformen des Service direkt zu erkennen: Die Dienstleistungen, Ereignisse und die ausgetauschten Materialien. Vorteil gegenüber ähnlichen Notationen – z.B. Klassendiagrammen – ist auch die Abstraktionsebene, die zwischen allgemeinen Übersichtsbildern und speziellen technischen Details liegt.

Die Architektur verlangte im CAI-Projekt softwaretechnisches Wissen über die Konstruktion großer Anwendungssysteme, über Verteilung und über den Einsatz von Middlewares. Besonders zu berücksichtigen sind die Schwierigkeiten von Webtops mit den Ereignissen aktiver Fachlicher Services.

Schließlich stellte auch die technische Umsetzung der Services wegen der Verteilung und Mehrbenutzerfähigkeit besondere Anforderungen an die Entwickler. Hierzu wurde eine Checkliste mit den kritischen Punkten vorgestellt. Raum für zukünftige Arbeiten bietet die Frage, wie die Implementierung Fachlicher Services am besten durch Rahmenwerke unterstützt werden kann.

Durch die ganz unterschiedlichen Anforderungen der einzelnen Schritte stellt die Kapselung existierender Anwendungssysteme eine Herausforderung dar.

Das in dieser Diplomarbeit entwickelte Vorgehensmodell und die damit gesammelten Erfahrungen bieten eine gute Grundlage für zukünftige Arbeiten in diesem Bereich. Spannende Forschungsfelder sind die Unterstützung der Fachlichen Services durch Rahmenwerke und die Untersuchung von Migration.

Auch in praktischen Projekten ist die Strukturierung und Anleitung durch das Vorgehensmodell hilfreich – im CAI-Projekt hat sie sich sehr bewährt.

7 Literatur

[Allen & Frost 98] Allen, P. & Frost, S. (1998) *Component-Based Development for Enterprise Systems. Applying The SELECT Perspective*. Cambridge University Press, Cambridge

[Bäumer 98] Bäumer, D. (1998) *Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme.* Dissertationsschrift, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik

[Bisbal 97] Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R. & O'Sullivan, D. (1997) *A Survey of Research into Legacy System Migration.* Trinity College, Broadcom Éireann Research, Dublin, Ireland

[Bleek 97] Bleek, W.-G. (1997) *Techniken zur Konstruktion verteilter und technisch eingebetteter Anwendungssysteme*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik

[Bleek 97a] Bleek, W.-G. (1997) *Agentenmonitor Kommunikationsschnittstelle*. Tenovis Business Communication GmbH, Bargteheide

[Braun 97] Braun, F. (1997) *Mehr Frauen in die Sprache. Leitfaden zur geschlechtergerechten Formulierung.* Ministerium für Frauen, Jugend, Wohnungs- und Städtebau des Landes Schleswig-Holstein

[Brockhaus 78] (1978) *Der große Brockhaus, achtzehnte Auflage.* Dritter Band, F. A. Brockhaus, Wiesbaden

[Felber, Guerraoui & Fayad 99] Felber, P., Guerraoui, R. & Fayad, M. E. (1999) *Putting OO Distributed Programming to Work*. In: *Communications of the ACM* (Vol. 42, No 11., November 1999)

[Fowler 99] Fowler, M. & Scott, K. (1999) *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 2nd Edition.* Addison-Wesley

[Gamma et al. 96] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1996) *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software.* Bonn; Reading, Massachusetts [u.a.]: Addison-Wesley

[Gryczan 96] Gryczan, G. (1996) *Prozeßmuster zur Unterstützung kooperativer Tätigkeit*. Wiesbaden: DUV, Deutscher Universitäts-Verlag

[Guerraoui & Fayad 99] Guerraoui, R., Fayad, M. E. (1999) *OO Distributed Programming Is* Not *Distributed OO Programming*. In: *Communications of the ACM* (Vol. 42, No 4., April 1999)

[Hars 95] Hars, R. (1995) *Agt-Server & Agt-GUI Communication.* Tenovis Business Communication GmbH, Bargteheide

[Hung, Simons & Rose 98] Hung, K., Simons, T., Rose, T. (1998) *'The Truth Is Out There?': A Survey of Business Objects.*
Quelle: http://www.dcs.shef.ac.uk/~kitty/OOIS98_Submission.htm, 13. 2. 2002

[Jacobsen, Booch & Rumbaugh 99] Jacobson, I., Booch, G. & Rumbaugh, J. (1999) *The Unified Software Development Process.* Bonn; Reading, Massachusetts [u.a.]: Addison-Wesley

[Kracke 96] Kracke, C (1996) *Proceßsystem Interface - Beschreibung.* Tenovis Business Communication GmbH, Bargteheide

[OMG 95] OMG (1995) *OMG Business Application Architecture*, *White Paper.* Framingham, MA. Quelle: http://jeffsutherland.org/oopsla/bowp2.html, 13. 2. 2002

[Otto & Schuler 2000]: Otto, M. & Schuler, N. (2000): *Fachliche Services: Geschäftslogik als Dienstleistung für verschiedene Benutzungsschnittstellen-Typen.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik

[Parnas 72] Parnas, D. L. (1972) *On the Criteria to be Used in Decomposing Systems into Modules.* In: *Communications of the ACM* (Vol 15, No 12, December 1972)

[Rechenberg & Pomberger 99] Rechenberg, P., Pomberger, G. (Hrsg.) (1999) *Informatik-Handbuch*. 2., aktualisierte und erweiterte Auflage. München, Wien: Carl Hanser Verlag

[Redlich 96] Redlich, J.-P. (1996) *Corba 2.0. Praktische Einfuehrung fuer C++ und Java.* Bonn; Reading, Massachusetts [u.a.]: Addison-Wesley

[Singh 99] Singh, H. (1999) Progressing to distributed multiprocessing. Prentice Hall, Upper Saddle River

[Sutherland 97] Jeff Sutherland (1997) *The Object Technology Architecture: Business Objects for Corporate Information Systems.*
Quelle: http://jeffsutherland.com/papers/boa_pap.html, 13. 2. 2002

[Züllighoven 98] Züllighoven, H. (1998) Das objektorientierte Konstruktionshandbuch nach dem Werkzeug- & Material-Ansatz. Heidelberg: dpunkt-Verlag

Das Zitat auf Seite 5 ist von dem französischen Maler Bernhard Buffet (1928 bis 1999).