ENTWURF EINES FRAMEWORKS FÜR CTI-LÖSUNGEN IM CALL CENTER

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Diplom-Informatiker Nikolai Bauer
geboren am 27. September 1969 in München

Gutachter:
Prof. Dr. habil. Alexander Schill, Technische Universität Dresden
Prof. Dr. habil. Heinrich Hußmann, Technische Universität Dresden
Prof. Dr. Johann Schlichter, Technische Universität München

Tag der Verteidigung:
6. Dezember 2002

Dresden, im Dezember 2002

Vorwort

Die vorliegende Arbeit entstand in der Zeit von 1998 bis 2002 im Rahmen einer externen Promotion an der Fakultät Informatik der Technischen Universität Dresden. Die Motivation, ein allgemeineres Anwendungsmodell und darauf aufbauend ein Framework für verteilte Anwendungen im Call Center zu entwerfen, entstand dabei aus den Erfahrungen und Erkenntnissen, die ich in konkreten Softwareprojekten in diesem Umfeld machen und gewinnen konnte.

Bedanken möchte ich mich an dieser Stelle besonders bei Herrn Professor Dr. Schill, der als Betreuer während dieser Zeit immer und prompt für Fragen und Diskussionen zur Verfügung stand sowie bei Herrn Prof. Dr. Hussmann und Herrn Prof. Dr. Schlichter, für hilfreiche Anregungen und kritische Anmerkungen.

Die Arbeit wäre darüber hinaus nicht möglich gewesen ohne die Unterstützung, die mir die Firma iSYS Software GmbH in München und hier insbesondere Herr Professor Dr. Mandl gegeben hat.

Zum Abschluss möchte ich meiner Frau Ingrid für die vielen Stunden danken, in denen sie mir bei der Erstellung und Korrektur des Manuskripts geholfen hat.

Zusammenfassung

Die rasante Entwicklung der Telekommunikationstechnologien bedeutet auch für die Softwareentwicklung viele Chancen aber auch Herausforderungen. Besonders in Call Centern spielt dabei die unter dem Begriff CTI (*Computer Telephony Integration)* zusammengefasste Integration von IT-Systemen und Telefonanlagen eine wichtige Rolle. Wenn auch diese Integration auf technischer Ebene in der Regel zufriedenstellend gelöst wird, zeigt ein Blick auf die Softwareentwicklung in diesem Bereich noch Nachholbedarf.

Die vorliegende Arbeit greift dieses Problem auf und versucht, den Ansatz CTI auf die Ebene der Entwicklung verteilter Anwendungen abzubilden. Ziel dabei ist es, Erkenntnisse darüber zu erzielen, inwieweit ein allgemeines Basismodell als Framework für die Entwicklung von CTI-Anwendungen definiert werden kann und welchen Mehrwert es mit sich bringt. Parallel dazu soll die Frage untersucht werden, inwieweit bewährte Methoden und Technologien verteilter Systeme auf diesem Spezialgebiet ihre Anwendung finden können.

Dazu wird ein allgemeines Anwendungsmodell für CTI-Lösungen und darauf aufbauend ein objektorientiertes, verteiltes Framework entworfen. Dabei wird unter anderem untersucht, inwieweit sich Abläufe mit Hilfe von Petrinetzen modellieren lassen. Wesentliche Inhalte des Basissystems bestehen in einer allgemeinen Schnittstelle für Telefonfunktionen, einem eigenständigen Zustandsautomaten sowie einer Verknüpfung verteilter Datenobjekte mit einem Anruf. Architektur und Design des Frameworks werden abschließend bewertet hinsichtlich Aspekten wie Variabilität, Abhängigkeit von Technologien und Mehrwert für die Entwicklung von CTI-Lösungen. Das Framework selbst wird als Prototyp implementiert und diversen Leistungsmessungen unterzogen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 EINLEITUNG

1.1 Hintergrund und Motivation

Die Verschmelzung von Informations- und Kommunikationstechnologien hat die Infrastrukturen und Einsatzgebiete von Call Centern in den letzten Jahren stark beeinflusst. Die Ansätze dazu reichen bis in die 70er Jahre zurück, wo in ersten Systemen Mainframes mit TK-Anlagen gekoppelt wurden. Einen ersten Aufschwung erfuhr der Bereich dann in den 80er Jahren vor allem in den USA, als besonders größere Unternehmen anfingen, den Kundenservice per Telefon immer stärker anzubieten. Seit den 90er Jahren haben sich Call Center und damit die CTI-Technologie weiter verbreitet und sind heute aus einem umfassenden Konzept zur Pflege von Kundenbeziehungen nicht mehr wegzudenken. In der unter dem Konzept von CTI immer stärker werdenden Verknüpfung von Telekommunikations- und IT-Technologie liegen viele Chancen für den Betrieb und die Effizienz von Call Centern. Es ergeben sich daraus jedoch auch eine Reihe von neuen Herausforderungen für verteilte Anwendungen, da das IT-System nun gefordert ist, die durch das in der Regel sehr leistungsfähige Telekommunikationssystem vorgegebenen Prozesse möglichst optimal zu unterstützen. Dazu kommt, dass die Integration solcher Systeme in eine bereits bestehende Infrastruktur zu berücksichtigen ist, denn obwohl in vielen Fällen für Call Center neue und eigene Systemumgebungen errichtet werden, so erfolgt der Zugriff auf unternehmensinterne Informationen in der Regel weiterhin über bestehende Systeme. Außerdem stellt ein Call Center in der Regel nur einen der vielen Kommunikationskanäle zu einem Unternehmen dar, der mit den übrigen in Einklang zu bringen ist. Aktuelles Beispiel dafür ist die immer größer werdende Kommunikation über eMail oder das WWW mit Unternehmen. Dadurch entsteht hier ebenfalls der Bedarf, unterschiedliche Systeme zu verbinden. Darüber hinaus entstehen auch im Call Center selbst zum Teil sehr heterogene Umgebungen, insbesondere wenn man den Trend hin zu verteilten oder sogar virtuellen Call Centern betrachtet.

So liegt eigentlich die Vermutung nahe, dass CTI-Systeme auch die Aspekte verteilter Anwendungen ausdrücklich berücksichtigen. Betrachtet man jedoch Lösungen, Konzepte und Forschung in diesem Bereich, so stellt man fest, dass trotz der eben dargestellten, hohen Anforderungen an Anwendungen im Call Center CTI bisher zum großen Teil nur technisch, nicht aber aus dem Blickwinkel der Softwareentwicklung und noch weniger aus Sicht der verteilten Anwendungen betrachtet wird und das, obwohl das Telefon noch immer das am meisten genutzte Kommunikationsmittel darstellt. So sind Anwendungen in diesem Bereich in der Regel immer Individualanwendungen und werden im Gegensatz etwa zu Workflowsystemen derzeit nicht wirklich als unternehmensweit verteilte Anwendungen betrachtet. Vielmehr liegt hier der Fokus auf der Interaktion von Rechner und Telefonanlage, ohne zu berücksichtigen, dass über einen Anruf mehrere Anwendungen gekoppelt werden. Ein Grund dafür ist sicherlich darin zu suchen, dass eine Mehrzahl der in Call Centern geführten Gesprächen nur zwischen Kunde und Mitarbeiter stattfindet und somit die Interaktion nur zwischen einem System und der Telefonanlage notwendig ist. Aus Sicht der Informatik ist dies aber zu wenig, da hier komplexere Prozesse wie Weiterleitung und Konferenz sowie die Integration in bestehende Infrastrukturen vernachlässigt werden.

Die für diese Arbeit aufgestellte These ist daher, dass CTI angewandt auf die Prozesse eines Call Centers zwangsläufig zu verteilten Anwendungen führt, da diese Prozesse von Natur aus verteilt, kollaborativ und zustandsbehaftet sind. Es wird in dieser Arbeit versucht, den Integrationsgedanken, also das I von CTI, auf der Ebene der unternehmensweit verteilten Anwendungen fortzuführen und dabei auch den verteilten Charakter des Telefonsystems zu berücksichtigen.

Aus dieser Grundthese resultiert nun die Fragestellung, inwieweit sich die bestehenden Methoden verteilter Anwendungen abbilden lassen um CTI auch auf dieser Ebene allgemein definieren zu können. Deshalb wird versucht, ein allgemeines Modell für eine verteilte CTI-Anwendung zu entwerfen und dabei auch einige der Methoden von der Spezifikation über den Entwurf bis hin zur Implementierung zur Anwendung zu bringen.

Daraus resultiert die Zielsetzung, anhand eines konkreten Frameworks zu untersuchen, wie tragfähig ein allgemeines Modell für verteilte CTI-Anwendungen ist und inwieweit man hier auf bewährte Methoden

der verteilten Systeme zurückgreifen bzw. welche Vorteile man daraus ziehen kann. Es soll dazu ein System entworfen werden, das folgende Aspekte anspricht:

- Unterstützung der wesentlichen Telefonie-Konzepte
- Unterstützung elementarer Geschäftsprozesse eines Call Centers
- Ansatz eines unternehmensweiten, verteilten Systems, das die Integration von bzw. die Anbindung an bestehende Systeme in den Vordergrund stellt.

Die Ergebnisse, die dabei erwartet werden, umfassen Erkenntnisse über Möglichkeiten und Grenzen eines derartigen Modells und des darauf aufbauenden Frameworks, Erkenntnisse bzgl. der Anwendung bewährter Methodiken verteilter Anwendungen auf dieses Spezialgebiet sowie den Beleg der aufgestellten These durch eine konkrete Implementierung und Test der Software.

Da das beschriebene Thema ganz allgemein von Bedeutung ist, wenn man IT-Anwendungen mit Telefonsystemen koppelt, ist zu erwarten, dass die erarbeiteten Ergebnisse auf einen großen Bereich von Anwendungen anwendbar sind und der Ansatz damit eine ausreichend allgemeine Gültigkeit besitzt.

1.2 Schwerpunkte

Ziel dieser Arbeit ist der Entwurf eines möglichst allgemeinen Basissystems für CTI-Anwendungen im Call Center. Dieses System muss dazu sowohl die Funktionalität moderner Telefonsysteme unterstützen als auch die Anforderungen berücksichtigen, die an eine zeitgemäße, verteilte Anwendung gestellt werden. Dazu werden in einem ersten Schritt die Abläufe eines Call Centers und die wesentlichen Prozesse einer Telefonanlage betrachtet und modelliert und somit die konkreten Anforderungen definiert.

Aus diesen Anforderungen ergeben sich dann im weiteren die wesentlichen Schwerpunkte dieser Arbeit. Dazu gehört der Entwurf eines möglichst allgemeinen System- und Anwendungsmodells für CTI-Anwendungen. Dazu wird untersucht, inwieweit sich bestehende CTI-Konzepte mit den Methoden und Technologien verteilter Systeme, insbesondere dem Einsatz einer Middleware wie CORBA, verknüpfen lassen. Diese Kombination soll eine Reihe von funktional hochwertigen Diensten besitzen, die direkt für die Entwicklung konkreter Anwendungen im Call Center genutzt werden können. Dazu gehört zum Beispiel der schnelle und einfache Zugriff auf Informationen, die mit dem Anruf bzw. der Person des Anrufers identifiziert werden, oder die einfache Steuerung auch komplexer Vorgänge wie etwa eine Weiterleitung oder eine Konferenz.

Als Basis für ein Framework werden dann Anforderungen an die notwendigen Technologien gestellt und derzeit verfügbare Vertreter dahingehend untersucht. Für den Bereich CTI werden dafür unterschiedliche Technologien verglichen und geprüft, inwieweit sie als Basis in ein mit erweiterter Funktionalität ausgestattetes System integriert werden können. Dazu gehören insbesondere neuere Systeme mit moderneren Schnittstellen, wie sie etwa in der JTAPI-Spezifikation im Umfeld von Java definiert werden. Analog dazu werden Middleware-Technologien und Programmiersprachen kurz auf die gestellten Anforderungen geprüft. Durch diese Untersuchung notwendiger Technologien wird die notwendige Unabhängigkeit des Ansatzes belegt.

Vor dem Hintergrund der eingangs definierten Ziele wird ein System entworfen, das als Basissystem für die Entwicklung konkreter Anwendungen im Call Center dienen soll. Die Konzeption stützt sich dabei auf eine möglichst formale Modellierung der grundlegenden Abläufe, in welchem Zusammenhang auch ermittelt wird, inwieweit hier Methoden wie Petrinetze geeignete Werkzeuge darstellen. Weiterhin stützt sich die Konzeption dann auf eine den Ansprüchen genügende, verteilte Architektur sowie die Auswahl geeigneter Basistechnologien. Darauf aufbauend werden die wesentlichen Funktionen, die notwendigen Schnittstellen sowie die elementaren Abläufe spezifiziert, so dass eine Basis für eine konkrete Implementierung geschaffen wird.

Neben einer Bewertung des Frameworks hinsichtlich Variabilität liegt der abschließende Schwerpunkt in einer prototypischen Implementierung, anhand der die erarbeiteten Ergebnisse nochmals überprüft wer-

den können und der gesamte konzeptionelle Ansatz vor dem Hintergrund eines konkreten Einsatzes in der Praxis eingeschätzt werden kann.

1.3 Einordnung der Arbeit

Durch die Weiterführung des CTI-Ansatzes berührt die Arbeit sowohl den Bereich der Telefontechnologie, insbesondere im Umfeld eines Call Centers, als auch den der verteilten Systeme.

Was Forschung und Entwicklung im Umfeld von CTI angeht, so konzentrieren sich die Arbeiten dort schwerpunktmäßig auf die technische Umsetzung des Integrationsansatzes durch Spezifikation von Architekturen, Schnittstellen und Anrufmodellen. Hier liefern vor allem die Hersteller von Telefonanlagen wie IBM oder Siemens und somit auch Industriegremien wie ECTF und Versit einen großen Beitrag. Eine der grundlegenden Arbeiten in diesem Bereich ist die CTI Encyclopedia von Versit (vgl. [Versit96a] [Versit96b], deren Inhalt nun in den Spezifikationen der ECTF weitergeführt wird.

Wesentliches Ziel dieser Spezifikationen ist, die Funktionalität von Telefonanlagen und vor allem die Anbindung von IT-Systemen einheitlich zu spezifizieren. Daneben haben vor allem Softwareunternehmen wie Novell, Microsoft oder Sun Microsystems Programmierschnittstellen wie TAPI oder JTAPI entwickelt, die einem Anwendungsprogramm Telefonfunktionalität zur Verfügung stellen und somit eine wichtige Basis für diese Arbeit darstellen. Darauf aufbauend werden in dieser Arbeit Aspekte wie Telefonsteuerung und Zustandsverwaltung vorwiegend aus Richtung der Anwendungsentwicklung betrachtet und vor diesem Hintergrund entsprechend erweitert.

Call Center selbst werden in der Literatur meist aus betriebswirtschaftlicher Sicht untersucht, wobei es in der Regel um die Modellierung und Optimierung der Geschäftsprozesse, die Organisation oder im Rahmen des Marketings um die Interaktion mit dem Anrufer als Kunden geht. Letzteres hat sich als CRM (*customer relationship management*) immer stärker als eigenständiger Bereich etabliert. Was Optimierung und Organisation angeht, so beschäftigen sich die Arbeiten hier mit Themen wie der maximalen Auslastung oder Effizienz, der Antwortzeit oder der Kosten eines Call Centers.

Ein verwandter Bereich ist das Thema Workflow und Workflowmanagement vor allem bezogen auf Call Center. Die Arbeit grenzt sich hier von bestehenden Publikationen dadurch ab, dass das vorgestellte System keine Workflow-Funktionalität beinhaltet und mit seiner engen Kopplung an das Telefonsystem als Ergänzung zu solchen Anwendungen verstanden wird und hier durch klare Schnittstellen die Möglichkeit der Integration bietet. Analog gilt dies auch für den Forschungsbereich Computer Supported Cooperative Work.

Der Bereich der verteilten Systeme ist ein sehr breites Forschungsgebiet der Informatik mit entsprechend vielen Schwerpunkten und Veröffentlichungen. Die Arbeit ist hier im Teilbereich des sog. *enterprise distributed computing* anzusiedeln, da hier CTI-Anwendungen in einem Call Center als Teil eines unternehmensweiten, auf einer Middleware basierenden, verteilten Systems angesehen werden. Wichtiger Schwerpunkt dabei sind Middleware-Ansätze wie CORBA, COM+ oder EJB, die durch Organisationen wie die OMG aber auch durch Unternehmen wie Sun oder Microsoft spezifiziert werden. Hier ist die Arbeit auch als konkretes Anwendungsbeispiel einer Middleware wie CORBA vor dem praktischen Hintergrund eines Call Centers zu sehen. Neben einzelnen Softwarelösungen wie etwa dem ehemalig von der Firma Nabnasset entwickelten System VESP, die CORBA für CTI-Lösungen einsetzen, sind derzeit keine Veröffentlichungen bekannt, die sich intensiver mit der Verknüpfung von CTI-Basissystemen und Middlewarelösungen beschäftigen.

Um der Modellierung von CTI-Anwendungen eine formalere Basis zu geben als allgemein üblich ist, werden die von C.A. Petri entwickelten Petrinetze in diesem Zusammenhang eingesetzt. Als bewährter Ansatz zur Beschreibung nebenläufiger Systeme werden sie in dieser Arbeit dazu verwendet, die komplexeren Abläufe innerhalb eines Call Centers wie Weiterleitung und Konferenz bzw. deren Abbildung auf das IT-System zu modellieren. Eine Einführung zu diesem Thema gibt [Reisig91].

Als Basissystem für CTI-Anwendungen berührt die Arbeit letztendlich noch den Bereich der objektorientierten Softwareentwicklung und hierbei insbesondere die Modellierung objektorientierter Frameworks. Auch dieser Bereich wird in der Literatur mehr als ausführlich behandelt, [Balzert96a], [Balzert96b] sowie [Gamma95] sind hier einige der verwendeten Publikationen. Grundsätzlich orientiert sich das Vorgehen bzgl. Spezifikation und Konzeption des Systems an den Vorgaben des *Rational Unified Process* (RUP) und stützt sich, was die objektorientierte Modellierung angeht, auf die *Unified Modeling Langauge,* wie sie in [OMG00], [Jacobson99] sowie [Rumb99] eingeführt wird.

1.4 Aufbau der Arbeit

Ziel dieser Arbeit ist die Konzeption und Entwicklung einer Softwarelösung, die als Basisframework für verteilte Systeme im Call Center dienen soll. Der Aufbau orientiert sich deshalb an einem Vorgehen, das sich in der Praxis in konkreten und vergleichbaren Softwareprojekten bewährt hat und sich dabei im Wesentlichen an den Vorgaben des *Rational Unified Process* (vgl. [Jacobson99]) orientiert. Somit gliedert sich diese Arbeit grob gesehen auch in die Einzelschritte, die während der Durchführung eines derartigen Softwareprojekts eine Rolle spielen.

In einem ersten Schritt wird die bestehende Ausgangssituation, nämlich die Prozesse und Abläufe eines Call Centers sowie die hier angewandten IT-Konzepte, beschrieben und analysiert. Dazu werden die wesentlichen Komponenten eines Call Centers untersucht sowie allgemeine und immer wieder auftretende Abläufe der Telefonie beschrieben. Darauf aufbauend wird die Integration von IT-Systemen in einem Call Center betrachtet, die zur Einführung von CTI und den damit verbundenen Basiskonzepten führt. Ergebnis dieser Analyse ist eine Reihe von Anforderungen, die an ein fortschrittliches, verteiltes System im Umfeld von Call Centern zu stellen sind, und deren Konzeption weiterer Inhalt dieser Arbeit ist.

Daraufhin werden alle für die weiteren Betrachtungen notwendigen Basiskonzepte wie Zustandsautomaten, Petrinetze sowie die Grundlagen objektorientierter Programmierung eingeführt. In Kapitel 5 werden Technologien bewertet und ausgewählt, die für die Implementierung des Frameworks notwendig sind.

In Kapitel 6 werden die gestellten Anforderungen weiter verfeinert und damit die Funktionalität des Frameworks, die verteilten Abläufe, die Zustandsautomaten sowie die konkreten Anwendungsfälle spezifiziert. Nächster Schritt ist nun der Entwurf eines Softwaresystems, das die spezifizierte Funktionalität zur Verfügung stellt. Wichtiger Teil dabei ist die Ausarbeitung einer geeigneten Architektur, der beteiligten Komponenten und ihrer Schnittstellen sowie ein konkretes Objektmodell. Verifiziert wird das Konzept, indem für ausgewählte Anwendungsfälle das Zusammenspiel der Komponenten bzw. Objekte überprüft wird.

Abschließend wird anhand des ausgearbeiteten Konzepts ein Prototyp implementiert und dadurch die praktische Umsetzung überprüft. Dieser Prototyp sowie das Konzept ist dann Gegenstand einer allgemeinen Bewertung, die neben einer allgemeinen Qualitätsanalyse auch die in einem Call Center gestellten Anforderungen an Leistungsfähigkeit und Skalierbarkeit in Betracht zieht. Kapitel 10 fasst schließlich die gewonnenen Ergebnisse nochmals zusammen und gibt einen kurzen Ausblick auf mögliche Erweiterungen und zukünftige Entwicklungen.

2 CTI IM CALL CENTER

Call Center sind derzeit das wohl wichtigste Einsatzgebiet von CTI Technologie. Aufgabe hier ist, eine relativ hohe Anzahl von Anrufen zu bedienen. Der Zweck, der dabei durch ein Unternehmen oder eine Organisation verfolgt wird, ist sehr vielfältig angesiedelt. Typische Anwendungsgebiete umfassen die Hilfe für den Gebrauch von Produkten (Help Desk), die Bestellung von Waren über das Telefon, allgemeine Serviceangebote oder das Abwickeln von Bankgeschäften (Telefonbanking). Die Mitarbeiter eines Call Centers, die häufig auch als Agenten bezeichnet werden, sind in der Regel für diese speziellen Aufgaben ausgebildet, um dem Anrufer einen entsprechenden Service bieten zu können.

Die professionelle Ausrichtung resultiert in einer sehr modernen Technologie im Bereich Telekommunikation und CTI, die man heute in modernen Call Centern findet.

2.1 Definition Call Center

Sehr allgemein gehalten ist ein Call Center der Bereich einer Organisation, in dem Mitarbeiter Telefongespräche führen, um die Informationen zu erhalten bzw. nach außen zu geben, die für die Interessen der Organisation bzw. des Unternehmens von Bedeutung sind.

Für die Beschreibung und Einteilung der verschiedenen Arten gibt es eine Reihe von Kriterien, die maßgeblich für die Architektur des Gesamtsystems sind. Im Vordergrund steht hier die Aufgabe, die aus Sicht des Unternehmens zu erfüllen ist. Einige Beispiele solcher Aufgaben sind:

Informationsdienste	Ein sehr weiter Bereich, der alle Arten von Informationen wie etwa Veranstaltungshinweise, Fahrpläne, Aktienkurse oder Börsennachrichten beinhaltet.
Verkauf	Hierzu zählen Dienstleistungen wie die Bestellannahme sowie Auftragsannahme, Angebotserstellung oder Reservierung.
Finanzdienstleistungen	Typische Aufgaben hier sind Überweisungen, Kreditkartenüberprüfung sowie der An- bzw. Verkauf von Wertpapieren.
Kundendienst	In diesen Bereich fallen typische Abläufe wie die Abwicklung von Garantiefällen, der technische Kundendienst oder die im Bereich der Softwareprodukte häufig anzutreffende Hotline bzw. das Help-Desk.

Tabelle 2-1 Einsatzgebiete von Call Centern

Diese Aufgaben umfassen in der Regel die Kommunikation mit Kunden, mit anderen Unternehmen sowie zwischen Abteilungen der Firma selbst (siehe [Minde99]).

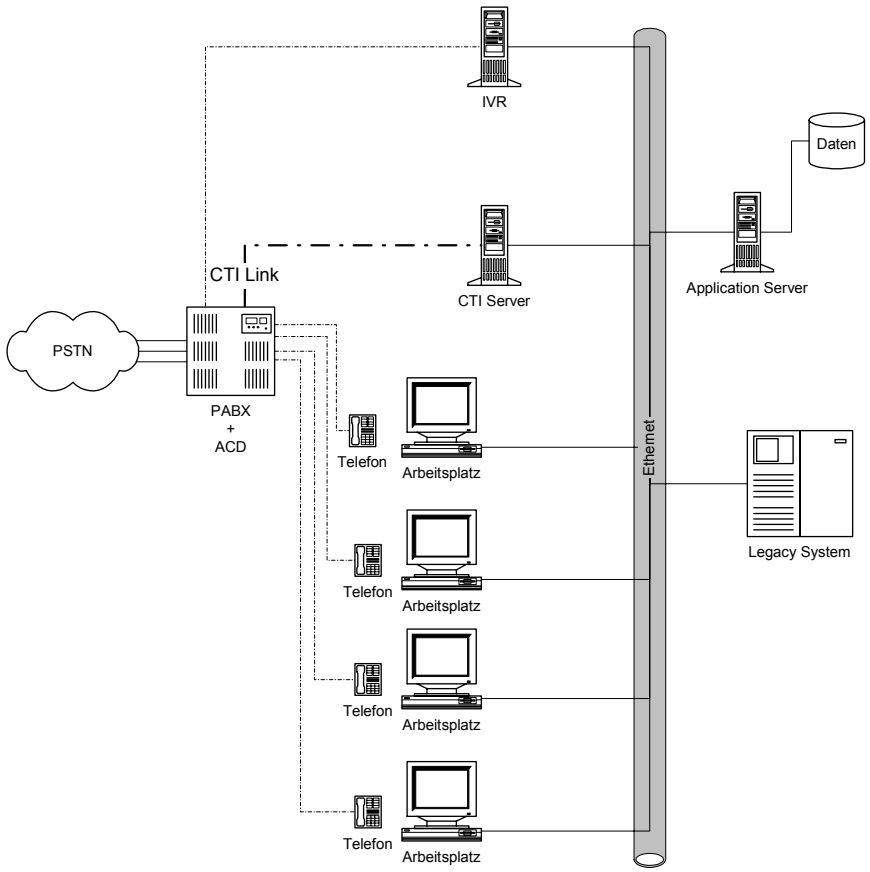
Darüber hinaus wird häufig auch von *Formal Call Center* und *Informal Call Center* gesprochen. Im ersten Fall handelt es sich um eine eigene Abteilung, die speziell für die Aufgabe, Anrufe zu tätigen bzw. entgegenzunehmen, eingerichtet wird. Dementsprechend ausgerichtet sind die Qualifikationen der Mitarbeiter sowie die technischen Einrichtungen des Call Centers, um eine möglichst hohe Zahl von Anrufen möglichst effizient bearbeiten zu können. Ein *Informal Call Center* ist dagegen eine Gruppe von Personen, die z.B. durch ein gemeinsames Projekt zeitweise in Telefonaktivitäten eingebunden werden. Ein Beispiel wäre technisches Personal, das für spezielle Fragen oder bei einem hohen Anrufvolumen Kundenfragen beantwortet.

Ein weiteres Kriterium teilt Call Center in die Bereiche *Inbound* sowie *Outbound* ein. Während im ersten Fall die Hauptaufgabe darin besteht Anrufe entgegenzunehmen, zielen Call Center der zweiten Kategorie darauf ab, selbstständig den Kontakt mit dem Kunden aufzubauen. In diesem Zusammenhang sind die unterschiedlichen Ansätze für das sog. *Outbound Dialing* interessant, auf die weiter unten noch genauer eingegangen wird. Sehr häufig findet man die Mischung beider Zielsetzungen in einem Call Center.

Durch die Integration neuer Technologien veränderte sich auch die Struktur eines Call Centers, so dass heute nicht nur Telefongespräche, sondern Medien wie Fax, eMail, Video oder das World Wide Web berücksichtigt werden. Durch die Möglichkeiten der Vernetzung treten Konzepte wie die eines virtuellen Call Centers bzw. eines verteilten Call Centers in den Vordergrund. Dabei verliert die örtliche Lage mehr und mehr an Bedeutung und große Firmen können verschiedene Abteilungen innerhalb eines Landes oder sogar international zusammenschließen. Die Vorteile bzgl. optimaler Auslastung und Effizienz sind hier enorm, die Ansprüche an die beteiligten DV-Systeme entsprechend hoch (siehe dazu auch [Farber98]).

2.2 Technologien und Systeme

Abbildung 2-1 zeigt die wesentlichen Komponenten, die derzeit in einem Call Center typischerweise zu finden sind. Die hier beteiligten Systeme werden in den folgenden Abschnitten näher erläutert. In der Grafik ist ebenfalls angedeutet, dass in einem Call Center einzelne Systeme nicht nur logisch sondern auch physikalisch miteinander kommunizieren. Während die Komponenten der DV-Welt über ein Standardnetzwerk wie etwa Ethernet verbunden sind, sind die einzelnen Telefone sowie auch der Sprachcomputer in der Regel über reine Telefonleitungen mit der Telefonanlage verbunden. Die Verbindung zwischen CTI-Server und PABX, die im Folgenden als *CTI-Link* bezeichnet wird, spielt dabei eine ganz besondere Rolle, da über die hier übertragenden Nachrichten sowie Ereignisse die Integration von Telefonie- sowie Computerfunktionalität ermöglicht wird.



PABX: Private Access Branch Exchange
PSTN: Public Switched Telephone Network
ACD: Automatic Call Distribution
IVR: Interactive Voice Response

Abbildung 2-1 Typische Komponenten eines Call Centers

Telefonanlage (Switch)

Zentrale Komponente der Telefonanlage sind die allgemein als PABX (*Private Access Branch Exchange)* bezeichneten Systeme. Dies sind in der Regel sehr leistungsfähige Vermittlungsanlagen, die den Anschluss aller internen Apparate an das öffentliche Telefonnetz realisieren.

Um die Funktionalität von CTI umsetzen zu können, ist eine entsprechende Schnittstelle (CTI-Link) zur Vermittlungsanlage Voraussetzung.

ACD

Die Funktionalität eines ACD-Systems (*Automatic Call Distribution)* beinhaltet im Wesentlichen das, was eine Telefonanlage eines Call Centers von einer üblicherweise in einem Büro eingesetzten unterscheidet. Die Anforderung, die zum Tragen kommt, ist die Verteilung einer eventuell sehr hohen Zahl von Anrufen auf momentan verfügbare Agenten. Ist dies nicht möglich, so bieten diese Systeme Warteschlangen, in der der Anrufer zur Überbrückung des Engpasses gehalten werden kann. Da eines der vorrangigen Ziele ist, diese Wartezeit zu verkürzen bzw. die Zahl der bearbeiteten Anrufe zu maximieren, bieten ACD-Systeme eine ganze Reihe von Konfigurationsmöglichkeiten an. So können beispielsweise Mitarbeiter (gemäß ihrer Qualifikation) zu einzelnen Gruppen zusammengefasst und die Verteilung der Anrufe darauf abgestimmt werden.

Die Funktionalität eines ACD-Systems ist entweder bereits im Switch integriert oder als eigenständiges System im Call Center vorhanden. Die Programmierung bzw. Konfiguration moderner Switch-Systeme inklusive ACD-Funktionalität besteht im Wesentlichen aus Batch-Programmen (*vector)*, die die Behandlung eines Anrufs regeln. Einzelne Kommandos werden dabei häufig als *vector command* bezeichnet. Ein solches Programm kann in etwa folgendermaßen aussehen:

1 Einreihen des Anrufs in Warteschlange für Gruppe X
2 Ansage nach 20 Sekunden Wartezeit
3 goto 1

Dieses Programm reiht den Anrufer in die Warteschlange für eine Gruppe X von Agenten ein. Er erhält nach jeweils 20 Sekunden, sofern kein Agent den Anruf entgegennimmt, eine Ansage.
Identifiziert werden solche Programme durch eine interne Telefonnummer, die sogenannte VDN (*Vector Directory Number*). Eine Weiterleitung an diese Nummer bewirkt, dass das Switch-System die vorgeschriebenen Schritte mit dem Anruf durchführt, ihn also beispielsweise an eine andere Skill-Gruppe weiterleitet.

VRU

Für ein Call Center sehr wichtige Systeme sind die sog. *Voice Response Units (VRU)* oder *IVR-Systeme (Interactive Voice Response)*. Dies sind Computersysteme, die eine Sprachausgabe und häufig auch eine Spracherkennung integriert haben. Dadurch ist es möglich, dass ein Anrufer eine Reihe von Aufgaben durch Kommunikation mit einem dieser Systeme erledigen kann. Konkretes Beispiel ist das Abfragen eines Kontostandes bei einer Bank. Die notwendigen Eingaben erfolgen dabei direkt über die Tastatur des Telefons, sofern dieses ein Mehrfrequenzwählverfahren unterstützt (Touch Tone Dialing). Ist dies nicht möglich, was besonders in Deutschland noch häufig der Fall ist, so lassen sich derartige Systeme auch über Spracheingabe steuern, die naturgemäß jedoch wesentlich aufwändiger und dementsprechend unzuverlässiger ist.

Workstation

An den Arbeitsplatzrechnern der Mitarbeiter findet konkret die Zusammenführung der Telefonfunktionalität mit den Aufgaben statt, auf die sich das Call Center konzentriert. Im Idealfall vollständiger Integration kann der Agent demnach mit einer Anwendung sowohl sein Telefon bedienen als auch alle notwendigen Aufgaben, die für den zu leistenden Service erforderlich sind, durchführen.

Application Server – Datenbank

Eine wichtige Komponente ist in der Regel das Datenbanksystem, das meistens zentral alle für den Betrieb eines Call Centers wichtigen Informationen verwaltet. Dies beinhaltet zum Beispiel Daten der einzelnen Agenten sowie der Anrufer. Darüber hinaus werden alle Informationen über den Betrieb des Call Centers hier abgelegt. Dies umfasst statistische Daten, die Historie aller Anrufe sowie allgemeine Konfigurationsdaten wie etwa Name und Spezifikation einzelner Gruppen von Agenten. In modernen Systemen stellt ein Application Server den Anwendungen der Agenten diese Daten zur Verfügung. Dies entspricht einer 3-Tier-Architektur, die in Client-Server-Systemen häufig eingesetzt wird.

Legacy Systems

Wie in vielen großen DV-Systemen spielen auch in Call Centern die bereits bestehenden Anlagen und Systeme eine entscheidende Rolle. Hier liegt in der Regel die Kernfunktionalität des Gesamtsystems, auf das sich die Firma stützt. Bei Banken sind dies typischerweise die Großrechnersysteme, die für die gesamte Kontoverwaltung zuständig sind. Im Falle des Telefonbankings ist daher der Zugriff auf diese Rechner notwendig.

CTI Server

Die Integration des Telefonsystems in das verteilte Computersystem wird in modernen Call Centern durch einen eigenen Server implementiert. Dieser ist einerseits an das lokale Netzwerk angeschlossen und verfügt andererseits über einen eigenen, für die Aufgaben von CTI spezifischen Zugang zu der jeweiligen Telefonanlage. Aufgabe dieses Servers ist es, die Kommunikation zwischen Computern sowie dem PABX-System zu ermöglichen. Darüber hinaus stellt er in der Regel Dienste für statistische Auswertungen, Systemkonfiguration bzw. Systemüberwachung zur Verfügung.

2.3 Abläufe und Prozesse

Neben der eingesetzten Technologie wird der Betrieb eines Call Centers natürlich wesentlich durch die Geschäftsprozesse bestimmt, die häufig auftreten und dementsprechend gut unterstützt werden müssen. Betrachtet man einige der bereits angesprochenen, häufig anzutreffenden Aufgaben etwas näher, so lassen sich bereits typische Anforderungen an die eingesetzten Systeme identifizieren.

Ein sehr allgemeiner und deshalb häufig anzutreffender Ablauf ist die Auskunft, die ein Kunde über einen Anruf im Call Center erhalten möchte. Hier gibt es sehr allgemeingültige Auskünfte wie Ansagen über Veranstaltungen oder Fahrpläne, die keine Information über den Anrufer erfordern. Individuellere Auskünfte, wie etwa das Abfragen eines Kontostands, des Lieferstatus einer Bestellung oder die Überprüfung einer Kreditkarte erfordern dagegen eine entsprechende Identifizierung des anrufenden Kunden. Lassen sich diese Auskünfte gut automatisieren, so kommen für derartige Dienste in der Regel Sprachcomputer zum Einsatz. Die notwendigen Informationen, die der Anrufer bereitstellen muss, werden dabei als Eingaben über die Telefontastatur oder als Spracheingaben verarbeitet. Handelt es sich um individuelle Anfragen, die nicht automatisch abgewickelt werden können, so unterscheidet man hier zwei Fälle. Im ersten kann der angerufene Mitarbeiter die benötigte Information selbst geben, wie etwa eine Auskunft über den Status einer Bestellung oder eine Preisauskunft. Bei komplexeren Anfragen vor allem im Bereich des technischen Kundendienstes ist häufig eine Weiterleitung an einen Spezialisten notwendig, der anhand der Anfrage und auch des Kundenprofils ausgewählt wird. Die Aufgabe des IT-Systems bei dieser Art von Prozessen ist es, Informationen aus Systemen des Unternehmens wie etwa einer Datenbank zu lesen, aufzubereiten und dem Client, in diesem Fall ein Sprachcomputer oder eine Anwendung an einem Arbeitsplatz des Call Centers, zur Verfügung zu stellen.

Ein anderer, sehr wichtiger Prozess ist die Annahme einer Bestellung oder eines Auftrags. Derartige Abläufe werden in der Regel nicht durch einen Sprachcomputer abgewickelt, da hier sehr viel Informationen etwa über das Produkt, den Kunden sowie Liefer- und Zahlungsmodalitäten ausgetauscht werden müssen. Ein entscheidender Unterschied zur allgemeinen Information liegt außerdem darin, dass hier ein verbindliches Geschäft abgeschlossen wird. Eine ausreichende Identifikation des Kunden ist deshalb unumgäng-

lich. In vielen Call Centern werden deshalb derartige Gespräche explizit mitgeschnitten und in einer zentralen Datenbank gespeichert, allerdings immer erst nach Zustimmung des anrufenden Kunden. Auch der Auftrag zur Überweisung oder der Kauf eines Wertpapiers gehört in diesen Bereich. Aktionen mit einem so eindeutigen Muster ließen sich prinzipiell sehr gut automatisieren, allerdings wird gerade im Bereich der Finanzdienstleistungen das persönliche Gespräch in den meisten Fällen bevorzugt.

Das Bearbeiten von Reklamationen sowie der technische Kundendienst können als Mischung zwischen den dargestellten Prozessen der reinen Information und der Auftragsannahme angesehen werden. Auch hier ist der Anrufer als Kunde und Inhaber eines Produkts bereits bekannt und muss entsprechend identifiziert werden. In vielen Fällen, kann sein Problem direkt durch entsprechende Antworten des ersten Mitarbeiters oder nach einer Weiterleitung durch einen Spezialisten gelöst werden. Im Falle eines echten Defekts muss hier die Abwicklung einer Reparatur oder eines Garantieanspruchs durchgeführt werden, vergleichbar mit dem Prozess der Auftragsannahme. Die Reklamation ist für Call Center deshalb eine Herausforderung, da hier der Anrufer in der Regel bereits negativ vorbelastet ist und deshalb ein möglichst guter Service angestrebt werden sollte. Hier bewährt sich die Nutzung einer Kontakthistorie für jeden Kunden, in der das System zentral alle wesentlichen Aktionen des Anrufers und entsprechende Hinweise der Bearbeiter speichert. So sieht ein Mitarbeiter zum Beispiel sofort, wenn ein Kunde wegen eines Problems schon öfter angerufen hat oder wie viel Aufwand ihm durch diese Reklamation schon entstanden ist. Solche Informationen können sehr hilfreich sein, allerdings muss die Speicherung solcher Informationen immer vor dem nachfolgend noch angesprochenen Hintergrund des Datenschutzes betrachtet werden.

Neben diesen Abläufen, die typisch für bestimmte Arten von Call Centern sind, gibt es auch allgemeine Prozesse, die in jeder derartigen Organisation auftreten können. So ist zum Beispiel eine Legitimation immer dann notwendig, wenn vertrauliche Informationen betroffen sind (etwa ein Kontostand) oder die Aktion eine Autorisierung des Anrufers voraussetzt (z.B. der Kauf eines Wertpapiers). Eine derartige Legitimation findet üblicherweise durch Eingabe einer geheimen PIN über die Telefontastatur statt. Überprüft wird die Eingabe normalerweise durch ein IVR-System, da hier die Geheimhaltung dieser Information am einfachsten gewährleistet werden kann. Muss ein Anrufer während eines Gesprächs mit einem Mitarbeiter nachträglich noch legitimiert werden, so werden hier in der Praxis unterschiedliche Vorgehensweisen eingesetzt. Eine Variante ist, den Anrufer zu einem Sprachcomputer zu vermitteln, der dann die Überprüfung übernimmt und das Gespräch daraufhin wieder an einen Mitarbeiter weiterleitet. Dieses Verfahren eignet sich, wenn die nachträgliche Legitimation eher die Ausnahme darstellt. Die dafür notwendige Vermittlung zu einem IVR-System und wieder zurück ist nämlich für den Anrufer eher umständlich. In manchen Fällen wird hier sogar noch die Forderung gestellt, dass nach der Legitimation wieder an den ursprünglichen Bearbeiter zurück vermittelt wird. Dieser Prozess wird in der Regel von CTI-Systemen nicht bzw. nur unzureichend unterstützt, da hier die Reservierung eines Arbeitsplatzes im Call Center notwendig wird. Kommt die Legitimation durch einen Mitarbeiter häufiger vor, so wird hier auch häufiger auf die vollständige Geheimhaltung dieser Information verzichtet und die PIN bzw. das Kennwort dem Mitarbeiter durchgegeben. Einige Banken setzten für ihr Telefonbanking auch die Variante ein, in der dem Mitarbeiter nur ein durch das System vorgegebener und jeweils zufällig ausgewählter Teil der gesamten PIN gesagt wird, so dass dieser nie die Gesamtinformation kennt. In allen diesen Fällen muss das IT-System die eingegebene Information mit einem Referenzwert aus der Datenbank vergleichen und über eine erfolgreiche Legitimation entscheiden. Je nach Sicherheitsanforderungen sind dabei Vorkehrungen zu treffen, diese Information gesichert zu übertragen und abzuspeichern.

Ein in jedem Call Center auftretender Prozess ist das Parken von Anrufern in Warteschlangen und das automatische Vermitteln an Benutzergruppen, wobei das Gespräch an den ersten freien Mitarbeiter dieser Gruppe vermittelt wird. Diese Funktionalität ist einer der Kernbestandteile des Telefonsystems sowie des CTI-Basissystems. Eine Anwendung und damit das betroffene IT-System müssen hier nur über geeignete Schnittstellen verfügen, um auch solche Abläufe berücksichtigen zu können. So ist beispielsweise die Dauer, die ein Anrufer bereits in der Warteschlange verbracht hat, eine für den Mitarbeiter sehr aufschlussreiche Information.

Ein anderer, nicht minder wichtiger Aspekt ist die laufende Speicherung möglichst vieler Informationen über den allgemeinen Betrieb eines Call Centers sowie individuell über anrufende Kunden und ihre Interaktion mit diesem. Die Daten stammen zum einen aus dem CTI-System sowie der Telefonanlage, die sehr

detaillierte, statistische Daten über alle Anrufe führt. Die Interaktion der Kunden lässt sich darüber hinaus durch eine entsprechend ausführliche Kontakthistorie sehr genau verfolgen. Solche Informationen sind in der Regel Basis sehr vielfältiger Auswertungen, die unter anderem die Qualität der angebotenen Dienstleistungen bewerten, aber auch als Ausgangspunkt diverser Marketingaktionen dienen. Allerdings baut sich hier ein Spannungsfeld zwischen Anonymität sowohl der Mitarbeiter als auch der Kunden auf der einen Seite und der gewünschten Auswertungen auf der anderen Seite auf. Das Management eines Call Centers muss deshalb sehr genau auf vorhandene rechtliche Grundlagen achten, die im Umfeld des Datenschutzes wie auch im Umfeld des Arbeitsrechts sowohl den anrufenden Kunden als auch den Mitarbeiter schützen. So bedarf beispielsweise ein Mitschneiden des Gesprächs sowohl der Zustimmung des Kunden als auch des Mitarbeiters. Auch die Speicherung und insbesondere die Weitergabe persönlicher Informationen eines Kunden ist ohne Zustimmung im Allgemeinen nicht erlaubt. Besonders interessant wird der Fall, wenn es sich um verteilte Call Center handelt, deren Standorte in mehreren Ländern vertreten sind, die eventuell unterschiedliche Rechtsgrundlagen haben. Was das betroffene IT-System angeht, so stellen solche Vorgaben lediglich die Anforderung, den Umfang der erfassten Daten entsprechend flexibel konfigurieren zu können.

Zusammenfassend wird demnach deutlich, dass all diese Prozesse eine wichtige Anforderung an das IT-System stellen, nämlich die wesentlichen Informationen über den Anrufer sowie die für ihn relevanten Daten aus den übrigen Systemen des Unternehmens schnell und zur richtigen Zeit zur Verfügung zu haben. Allen Abläufen gemeinsam ist natürlich, dass der gesamte Betrieb des Call Centers möglichst effizient gestaltet wird und Anrufer in möglichst kurzer Zeit bedient werden, was die geforderte Leistungsfähigkeit auch der betroffenen Anwendungen entsprechend beeinflusst.

2.4 Call Center und WWW

Es ist gerade in letzter Zeit zu beobachten, dass die immer weiter steigende Nutzung des World Wide Web auch einen wesentlichen Einfluss auf die Arbeit und die Anforderungen an Call Center hat. Dies liegt in erster Linie daran, dass beide Ansätze aus Sicht des Unternehmens prinzipiell den gleichen Zweck erfüllen, nämlich einen Kommunikationskanal zum Kunden oder Geschäftspartner zur Verfügung zu stellen. Während das Internet sich bisher hauptsächlich auf das Anbieten von Information konzentrierte, tritt nun im Rahmen von eCommerce und eBusiness immer mehr die Interaktion mit dem Benutzer in den Vordergrund, ein Service, auf den sich Call Center im Wesentlichen konzentrieren. Beide Systeme treten somit in vielen Bereichen als Konkurrenten auf, was man zum Beispiel sehr gut im Falle des Telefonbanking bzw. Internetbanking beobachten kann. Der große Vorteil der Internet-Lösung liegt hier in den wesentlich geringeren Kosten, die für eine Firma entstehen, um einen 24-Stunden-Service anbieten zu können. Ein Call Center kann dies nur durch den Einsatz von IVR-Systemen erreichen, die allerdings im Vergleich keine sehr komfortable Benutzerschnittstelle aufweisen können.

Es ist also damit zu rechnen, dass viele typische Einsatzbereiche von WWW-Anwendungen abgelöst werden. Allerdings zeigt die Erfahrung, dass die Kommunikation über das Telefon mit einer Person ganz eigene Vorteile hat, insbesondere wenn es um sehr spezielle Probleme etwa im Falle eines Help-Desk für Software geht. Auch in Bereichen wie Vertrieb und Marketing ist der individuelle und vor allem persönliche Kontakt derzeit noch nicht wegzudenken. Hier wird sich das Call Center demnach auch in nächster Zeit noch behaupten. Auch kann das Internet derzeit noch nicht mit der Verfügbarkeit mithalten, die ein Call Center bietet, das von jedem Telefon und insbesondere durch Mobiltelefone praktisch von jedem Ort aus genutzt werden kann. Als Alternative dazu könnten sich der Internet-Zugang per WAP oder zukünftige, mobile Kommunikationslösungen etablieren, deren Verbreitung allerdings derzeit nur schwer abzuschätzen ist.

Ein ganz neuer Anwendungsbereich wird sich allerdings durch die Kombination beider Medien ergeben, die sich in vielen Fällen gut ergänzen und so im Grunde voneinander profitieren können. So unterstützt das Informationsangebot eines Internet-Auftritts die Aufgaben eines Call Centers, wenn hier bereits ein Großteil der häufig verlangten Informationen verfügbar ist (z.B. FAQ) oder unkomplizierte und standardisierte Aktionen (Bestellung, Überweisung etc.) durch den Anwender selbstständig durchgeführt werden können. Das Call Center wird dadurch entscheidend entlastet und kann sich auf die komplexeren Prozesse der individuellen Beratung konzentrieren. Im Gegenzug dazu wird die Qualität einer Internet-Anwendung

enorm gesteigert wenn die Möglichkeit besteht, bei Problemen und individuellen Fragen automatisch mit einem Call Center verbunden zu werden.

Als Fazit kann man ziehen, dass das Call Center durch das Internet sicherlich herkömmliche Einsatzbereiche verlieren, aber vor allem als Backend-System für WWW-Anwendungen auch ganz neue hinzu gewinnen wird. Die Anforderungen werden sich allerdings ändern. So werden kleinere und dynamischere Center immer wichtiger werden und somit wird auch die Bedeutung der Flexibilität des CTI-Systems sowie der eingesetzten, verteilten Anwendungen weiter steigen. Eine Forderung, die bereits heute schon gestellt wird, ist die Anbindung der Web-Anwendung an die Telefonanlage, um zum Beispiel dem Benutzer über das Internet bei Fragen und Problemen einen schnellen und persönlichen Rückruf anbieten zu können.

2.5 Telefonie Konzepte

Das Telefonsystem ist Grundlage eines jeden Call Centers. Die Aufgabe dieses Systems ist im Wesentlichen die Verarbeitung von Anrufen. Die grundlegenden Konzepte und Verfahren, die dabei eingesetzt werden, werden in diesem Abschnitt eingeführt. Zu diesem Zweck wird eine graphische Notation benützt, die in [Bayer97] sowie [Versit96a] eingeführt wird.

2.5.1 Grundlagen

Wichtigstes Element eines Telefonsystems ist der Anruf selbst. Diese Abstraktion bezeichnet die Übertragung von Medien (Sprache, Ton, Bild) sowie Kontrollinformationen zwischen einem Endgerät und dem Telefonnetz. In der Notation wird ein Anruf als *call* bezeichnet und mit einem Kreis dargestellt, der den Buchstaben C sowie eine Nummer als Bezeichner beinhaltet.

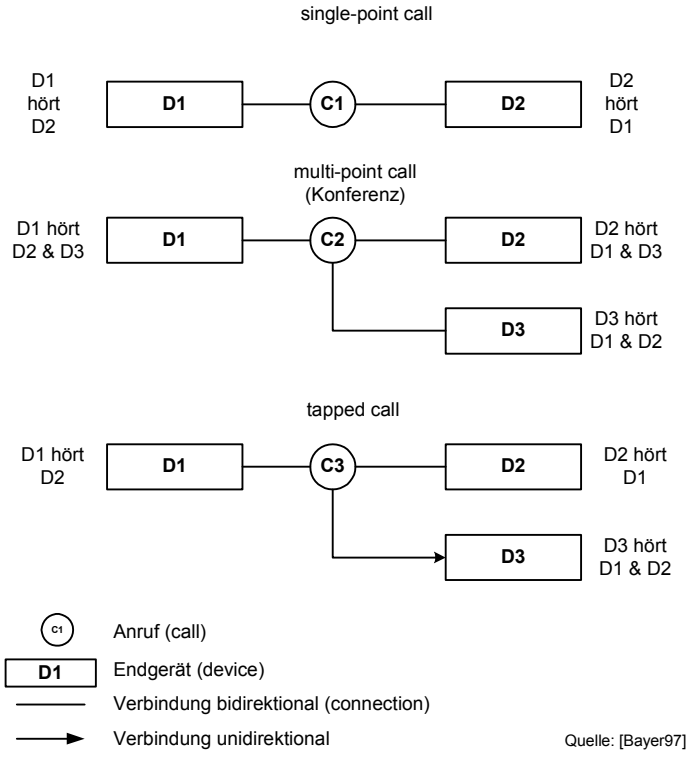


Abbildung 2-2 Arten von Anrufen

Geräte (*device)* sind die Elemente eines Telefonsystems, die die Ziele des Medienstroms darstellen. Sie werden durch ein Rechteck sowie den Buchstaben D und eine Nummer dargestellt. Die Beziehung zwischen

einem Anruf und einem Endgerät wird als Verbindung (*connection)* bezeichnet. In der Notation werden sie durch einfache Linien zwischen Gerät und Anruf dargestellt, wenn die Verbindung bidirektional ist. Im Sonderfall einer unidirektionalen Verbindung kann die Richtung durch Pfeile angedeutet werden.

Abbildung 2-2 gibt einen ersten Überblick darüber, wie die eben beschriebene Notation eingesetzt werden kann. Im ersten Fall ist ein üblicher Anruf dargestellt, in dem zwei Geräte (D2 und D2) miteinander verbunden sind. Der zweite Fall stellt dagegen eine Konferenz dar, in der drei unterschiedliche Geräte an einem Anruf teilnehmen. Das dritte Beispiel zeigt den Einsatz von gerichteten Verbindungen, da in diesem Fall der Teilnehmer D3 die Gespräche von D1 und D2 zwar abhören kann, jedoch selbst nicht am Telefongespräch teilnimmt.

Um in einem weiteren Schritt die wesentlichen Abläufe eines Telefonsystems modellieren zu können, werden verschiedene Zustände eingeführt, in denen sich eine Verbindung befinden kann. Ein Dienst eines Telefonsystems wie etwa das Annehmen eines Anrufs spiegelt sich analog immer in einem entsprechenden Zustandsübergang wider. Folgende, in Abbildung 2-3 dargestellte Zustände werden berücksichtigt.

Null	Dieser Zustand bezeichnet eine nicht existierende Verbindung.
Initiated	Eine Verbindung ist im Zustand *Initiated,* wenn ein angeschlossenes Gerät versucht, einen Anruf zu initiieren. Typisches Beispiel dafür ist etwa das Abnehmen des Hörers.
Connected	In diesem Zustand ist eine Verbindung zwischen den Geräten durch das Telefonsystem aufgebaut worden und die Datenübertragung kann stattfinden.
Alerting	Dieser Zustand zeigt an, dass das System versucht, einen Anruf mit einem entsprechenden Endgerät zu verbinden.
Hold	Im Zustand *Hold* wird ein bestimmtes Gerät weiterhin einem Anruf zugeordnet, die Übertragung von Medien allerdings vorübergehend unterbunden. Informationen, die den Anruf betreffen, wie etwa ein Hangup-Signal, werden allerdings weiterhin übertragen. Dieser Zustand unterscheidet sich also von der ebenfalls üblichen Stummschaltung (*Mute),* bei der das Endgerät lediglich das Mikrofon sowie den Lautsprecher abschaltet.
Queued	Dieser Zustand ist vergleichbar mit *Hold*, da auch hier der Fortschritt des Anrufs unterbrochen wird. Allerdings in diesem Falle, damit entsprechende Vermittlungsdienste ausgeführt werden können. Beispiel ist etwa das Halten des Anrufs in einer Warteschlange, bis ein geeigneter und freier Teilnehmer im Call Center ermittelt wurde.
Fail	Dieser Zustand tritt auf, wenn zwischen Anruf und Endgerät keine Verbindung aufgebaut werden kann. Typischer Grund dafür ist, dass der Teilnehmer belegt ist (*busy).*

Abbildung 2-3 zeigt die eben beschriebenen Zustände nochmals im Überblick und gibt die möglichen Übergänge an.

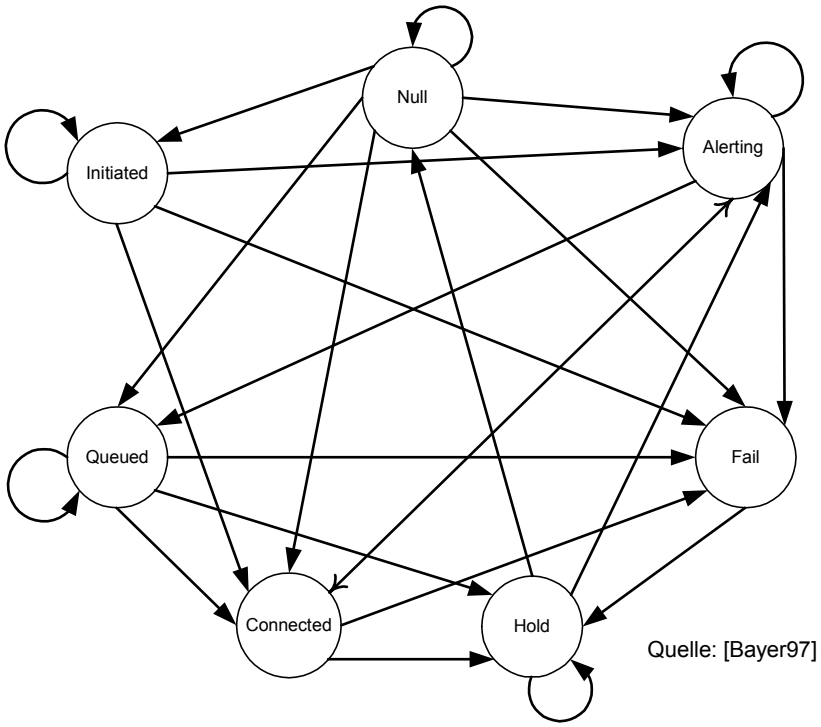


Abbildung 2-3 Verbindungszustände

Entsprechend den eingeführten Zuständen wird auch die graphische Notation geeignet erweitert. Dazu werden die Linien, die einzelne Verbindungen repräsentieren, mit Buchstaben für die einzelnen Zustände versehen. Dabei steht

‚a' für *alerting*
‚c' für *connected*
‚f' für *fail*
‚h' für *hold*
‚i' für *initiated*
‚n' für *null*
‚q' für *queued*

Zustandsübergänge werden in dieser Notation durch zwei oder mehrere Diagramme dargestellt, die horizontal oder waagerecht angeordnet durch Trennstriche unterteilt werden und für jeden Schritt die beteiligten Endgeräte, Verbindungen sowie ihre Zustände anzeigen.

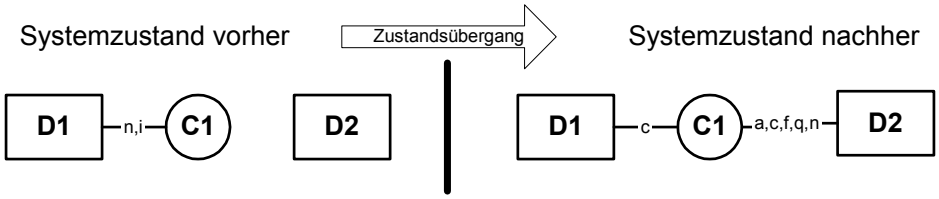


Abbildung 2-4 Zustandsübergang in einem Telefonsystem

2.5.2 Elementare Abläufe und Dienste

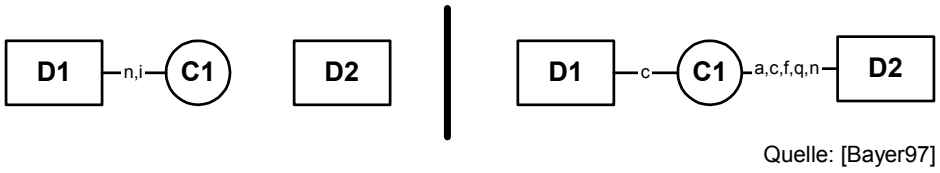
Auf Basis der in 2.5.1 eingeführten Notation lassen sich nun eine Reihe grundlegender Abläufe darstellen, die die wesentliche Funktionalität einer Telefonanlage, nämlich die Steuerung und Vermittlung von Anru-

fen, widerspiegeln. Die dazu notwendigen Dienste (*Call Control Services)* werden jeweils durch 2 Diagramme dargestellt, die die beteiligten Komponenten (Gerät, Anruf, Verbindung) sowie die jeweiligen Zustände der einzelnen Verbindungen vor bzw. nach Ausführung des Dienstes beschreiben. Weiterhin wird die Notation der Verbindungszustände erweitert:

‚!' für einen nicht weiter spezifizierten Zustand einer Verbindung
‚#' für einen nicht weiter spezifizierten Zustand, der allerdings nicht *null* ist
‚*' für einen nicht weiter spezifizierten Zustand, der durch den Dienst nicht betroffen ist

Auf Basis dieser Notation werden die wichtigen Dienste eines Telefonsystems und damit auch die wesentlichen Prozesse eines Call Centers wie in nachfolgendem Beispiel beschrieben.

Abbildung 2-5 zeigt den einfachsten Dienst *Make Call*. Die Verbindung des Geräts D1 ist entweder nicht vorhanden (*null)* oder im Zustand *initiated*, je nachdem, ob direkt mit aufgelegtem Hörer gewählt werden kann oder ein Abheben zuerst notwendig ist. Den ersten Fall, den man auch als o*n-hook dialing* bezeichnet, kennt man beispielsweise von Mobiltelefonen. Der Dienst ist beendet, sobald die Verbindung zwischen D1 und C1 den Zustand *connected* erreicht hat. Die Verbindung zu D2 kann zu diesem Zeitpunkt verschiedene Zustände besitzen. In den meisten Fällen wird dies *alerting* sein und das Gerät den Zustand entsprechend anzeigen (Klingeln). Es ist aber auch denkbar, dass die Verbindung direkt in den Zustand *connected* gelangt. In diesem Fall antwortet das Gerät D2 automatisch auf eingehende Anrufe (*auto answer).*

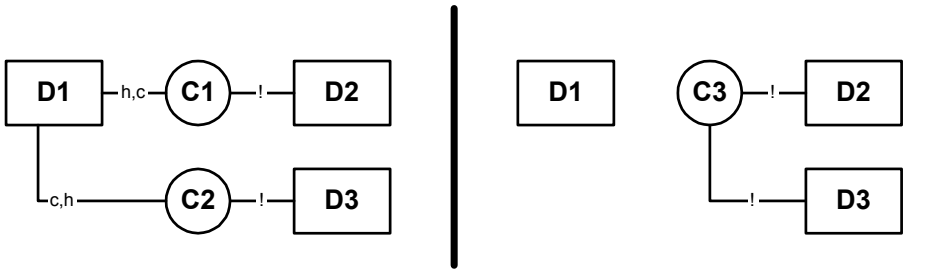


Quelle: [Bayer97]

Abbildung 2-5 Make Call

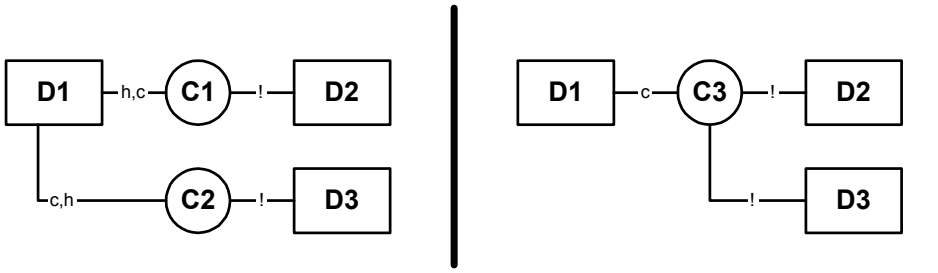
Eine anspruchsvollere Funktion des Telefonsystems ist die Weiterleitung eines Anrufs. In Abbildung 2-6 ist der dazu notwendige Dienst dargestellt. Dieser Dienst allein trennt D1 von den beiden Anrufen C1 und C2 und verbindet die Teilnehmer D2 sowie D3 mit einem neuen Anruf C3. Die Verbindung zwischen D1 und C1 bzw. C2 kann dabei den Zustand *hold* oder *connected* haben.

Ähnlich verläuft auch die Telefonkonferenz, die in Abbildung 2-7 dargestellt ist, allerdings mit dem Unterschied, dass hier im neuen Zustand alle drei Parteien miteinander verbunden sind.



Quelle: [Bayer97]

Abbildung 2-6 Transfer Call



Quelle: [Bayer97]

Abbildung 2-7 Conference Call

Analog diesen beiden Beispielen werden nun auch weitere und zum Teil komplexere Abläufe, etwa verschiedene Versionen der Weiterleitung oder die Definition einer Warteschlange, definiert. Die einzelnen Diagramme sowie eine kurze Erklärung dazu sind im Anhang aufgeführt. Diese elementaren Abläufe müssen von einem fortschrittlichen System eines Call Centers entsprechend unterstützt werden und sind daher bei der Konzeption eines verteilten Systems weiter zu verfeinern.

2.6 Computer Telephony Integration CTI

2.6.1 Definition

Anhand der Übersicht über die einzelnen Komponenten innerhalb eines Call Centers wird die Aufgabe von CTI sehr schnell deutlich. Sie liegt in der funktionalen Integration der auf der Seite der Telekommunikation liegenden Komponenten in die übrige Welt der DV-Systeme. Etwas konkreter lässt sich CTI auch in folgende Bereiche unterteilen:

- Austausch von Nachrichten und Kommandos zwischen Telefon- und Computer-Systemen
- Kontrolle, Koordination sowie Überwachung von Anrufen, Ereignissen des Telefonsystems, Anwendungen sowie Endgeräten
- Integration von bestehenden sowie neuen Computer- bzw. Telefonsystemen

Die Basis für die Entwicklung von CTI haben Firmen und Organisationen wie PTT (Post, Telegraph & Telephone) geschaffen, die individuelle Lösungen für den Kundendienst oder ähnliche Serviceleistungen entwickelt haben. Die ersten wirklichen Produkte entstanden jedoch erst Mitte der achtziger Jahre. Der erste Schritt dieser Entwicklung war die Nutzung eines Terminals, das sowohl Informationen der Telefonanlage erhielt als auch an das lokale DV-System (in der Regel ein Großrechner) angeschlossen war. Die Integration führte dabei noch der jeweilige Agent durch, indem er zwei getrennte Systeme bediente. Durch die Einführung von sprachverarbeitenden Systemen wurden dann einzelne Computer-Systeme direkt mit der Telefonanlage verbunden und konnten spezielle Aufgaben übernehmen. Eine echte Integration wurde jedoch erst durch die Verwendung einer CTI-Verbindung (CTI-Link) zwischen Telefonanlage und Computersystemen ermöglicht. Dadurch sind Client-Applikationen in der Lage, auf den jeweiligen Zustand der Telefonanlage bzw. des lokalen Apparats zu reagieren und im Zusammenspiel mit weiteren Systemen den Bearbeiter während des Anrufs entsprechend zu unterstützen. Die verschiedenen Stufen der Integration sind in Abbildung 2-8 nochmals dargestellt.

Darüber hinaus bieten alle gängigen CTI-Produkte heute Dienste an, die den Call Center Betrieb protokollieren und statistische Auswertungen vornehmen.

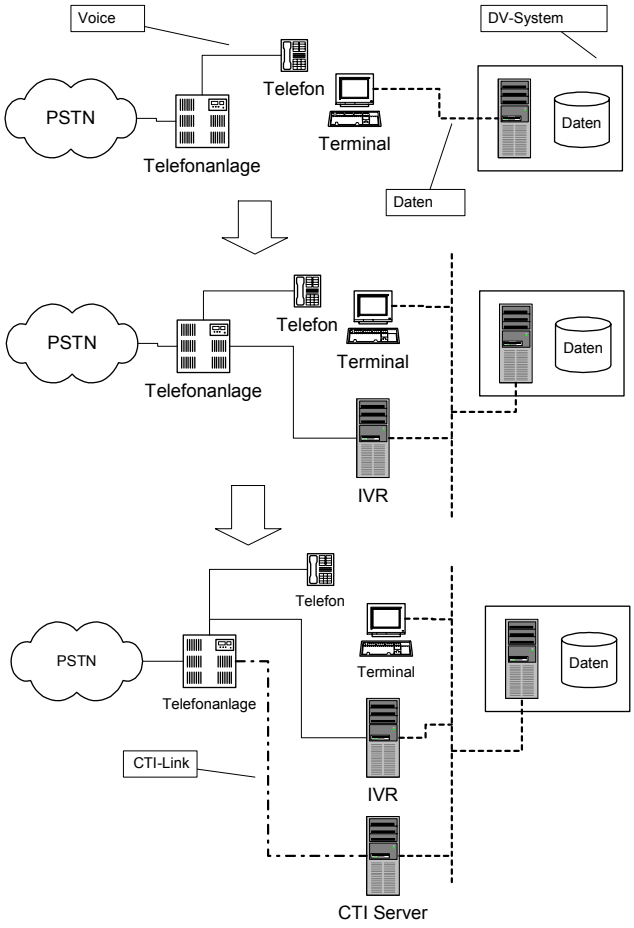


Abbildung 2-8 Stufen der Integration

Die weitere Entwicklung von CTI liegt in der Integration neuer Medien wie Video, WWW oder Fax als Alternative zum reinen Telefonanruf. Dies ist für Call Center aber auch für den weiten Bereich der Telekommunikation ein sehr interessanter Aspekt.

2.6.2 CTI-Konzepte und -Lösungen

Die konkreten Einsatzmöglichkeiten sowie die aus der Integration von Telefon- sowie IT-System resultierenden Funktionalitäten sind äußerst vielfältig. Eine Übersicht über die wichtigsten Beispiele gibt nachfolgende Liste.

First Party – Third Party

Ein wichtiges Kriterium in Bezug auf Anwendungen im Bereich CTI ist die Unterscheidung zwischen *first party* und *third party* (siehe [Strath96]). Dabei ist im ersten Fall die Applikation direkt Teilnehmer eines Telefongesprächs und hat dementsprechend die Kontrolle über den Anruf. Dies ist der Fall, wenn ein mit entsprechender Telefonhardware ausgestatteter PC direkt einen Anruf erhält. Die beteiligte Anwendung reagiert auf dieses Ereignis, nimmt beispielsweise den Anruf entgegen und führt weitere Aktionen durch. Im Falle von *third party* liegt dagegen die Kontrolle des Anrufs bei einem nicht unmittelbar beteiligten System. Hier ist die Telefonanlage nicht direkt mit einem PC sondern mit einem zentralen Server verbunden.

Screen Pop

Mit diesem Begriff wird eine der wesentlichen Aufgaben bezeichnet, die Anwendungen der Arbeitsplatzrechner eines Call Centers zu erfüllen haben. Gemeint ist damit die Anzeige möglichst vieler und gehaltvoller Information, die gleichzeitig bzw. noch vor einem Anruf auf dem Bildschirm erscheint. So kann sich der Bearbeiter vor dem Annehmen bereits auf den anstehenden Anruf vorbereiten. Voraussetzung hierfür ist, dass das CTI-System der lokalen Applikation genügend Informationen zur Verfügung stellt, die dann wiederum aus der lokalen Datenbank oder durch Kommunikation mit weiteren Systemen alle relevanten Daten anzeigt.

Sogenannte *Screen Consultations* sowie *Screen Conferences* verknüpfen dieses Prinzip mit den herkömmlichen Funktionen der Rückfrage bzw. der Konferenz, die Telefonanlagen üblicherweise bieten. Allerdings erhält hier der neue Teilnehmer eine Kopie des Bildschirms mit allen relevanten Daten des Anrufers.

Intelligent Call Routing

Durch in diesem Bereich von modernen CTI-Lösungen gebotene Funktionen wird es Anwendungen ermöglicht, Anrufe gemäß unterschiedlichster Kriterien und unter Einbeziehung verschiedenster Informationen in einem Call Center weiterzuleiten. Dies kann eine bestimmte Gruppe von Mitarbeitern, ein einzelner Agent oder ein bestimmter Sprachcomputer sein. Die CTI-Komponente stellt dafür Informationen wie etwa die gewählte Nummer (DNIS), oder die Nummer des Anrufers (CLI) bzw. individuell erfasste Daten zur Verfügung. Die Anwendung entscheidet daraufhin, welches Ziel für den Anruf geeignet ist und leitet ihn, wiederum mittels Diensten der CTI, weiter.

Ein Beispiel wäre hier die intelligente Weiterleitung im Call Center einer Versicherung. Der Kunde wird dabei zuerst von einem IVR-System begrüßt, dem er seine individuelle Versicherungsnummer mitteilt. Anhand dieser Information kann dann ein Transfer zu einem gerade freien Mitarbeiter eingeleitet werden, der für diesen Kunden bzw. seine Versicherung zuständig ist.

Intelligent Dialing

Ein sehr wichtiger Bereich, den man zu den sog. *Outbound*-Funktionen von CTI zählt, betrifft das Anrufen von Kunden, Interessenten oder ähnlichen Personen durch die Mitarbeiter eines Call Centers. Häufiges Ziel solcher Aktionen ist es, neue Kundenkontakte im Rahmen von Marketingaktionen zu knüpfen. Auch im Bereich der Reklamationsbearbeitung bzw. allgemeiner Serviceaufgaben ist häufig ein Rückruf notwendig, wenn neue Ereignisse eingetreten sind.

Wie allgemein in einem Call Center ist die Effizienz auch bei Outbound Aktionen sehr wichtig. Deshalb wird es angestrebt, dass die Mitarbeiter möglichst viele Kunden bedienen und dabei möglichst wenig Leerlauf haben. Kommen Gespräche nicht zustande, weil der Teilnehmer z.B. besetzt ist bzw. vergeht viel Zeit, bis der Mitarbeiter alle Daten für das Gespräch verfügbar hat, geht dies zu Lasten der Effizienz.

Die Unterstützung, die CTI hier leisten kann, lässt sich in mehreren Stufen klassifizieren:

- Auto Dialing
  In diesem Fall erhält der Mitarbeiter durch die CTI-Anwendung eine Liste von möglichen Teilnehmern. Wählt dieser hiervon einen aus, so wird die Verbindung hergestellt. Der Vorteil liegt hier im schnelleren und vor allem fehlerfreien Wählen durch die Applikation.
- Preview Dialing
  Bei diesem Verfahren trifft die Anwendung bereits die Auswahl eines einzigen Teilnehmers und präsentiert die verfügbaren Daten dem Agenten. Die Kontrolle des konkreten Anrufs liegt jedoch noch bei dem Bearbeiter.
- Progressive Dialing
  Die Kontrolle des Anrufs liegt hier ebenfalls in der Hand der CTI-Anwendung, die einen Teilnehmer auswählt, den Anruf initiiert und den Agenten direkt verbindet. Alle verfügbaren Daten werden dabei dem Agenten zur Verfügung gestellt.

- Predictive Dialing
  Um die Effizienz eines Call Centers noch weiter zu steigern, werden hier gleichzeitig mehrere Anrufe initiiert. Nimmt einer der angerufenen Teilnehmer das Gespräch entgegen, so wird er an den nächsten freien Agenten vermittelt. Ziel ist es demnach, keine Zeit durch nicht erfolgreiche Anrufe zu verschwenden. Die Anwendung muss jedoch anhand statistischer Daten sowie relativ komplexer Algorithmen die richtige Anzahl der Anrufe ermitteln, die parallel initiiert werden können, damit im Mittel immer mindestens ein Agent zur Verfügung steht.

2.6.3 Integration neuer Medien

Bereits mehrmals wurde die Integration neuer Medien angesprochen, die für Call Center neue Möglichkeiten und Aufgaben mit sich bringen. Deshalb ist es wichtig, hier einige Beispiele zu betrachten. Analog zu IVR-Komponenten bieten Fax-Server einem Anrufer die Möglichkeit, selbstständig Informationen abzurufen, die dann an sein persönliches Telefax-Gerät versendet werden. Dieses als *fax on demand* bekannte Verfahren ist inzwischen relativ weit verbreitet. Die Erweiterung des reinen Telefonanrufs durch eine Video-Übertragung ermöglicht einen engeren und persönlicheren Kontakt der Kommunikationspartner. Diese Technologie ist demnach dann interessant, wenn der eher anonyme bzw. unpersönliche Anruf z.B. vom Kunden nicht akzeptiert wird. Da jedoch die hierfür notwendige Technik noch relativ aufwändig und vor allem nicht sehr verbreitet ist, taucht die Integration dieses Mediums bisher nur in Sonderformen auf. Als Beispiel sei hier eine amerikanische Firma genannt, die in verschiedenen Supermärkten bzw. Drugstores spezielle Video-Terminals inklusive Scanner eingerichtet hat. An diesen Terminals kann der Kunde sein Rezept scannen lassen und sich dann über eine Videoverbindung bzgl. eines notwendigen Medikaments beraten lassen bzw. dieses bestellen.

Eine Ausnahmestellung nimmt das Internet auch als Medium im Hinblick auf CTI ein. Der Ansatz, Telefongespräche direkt über das Internet abzuwickeln, ohne die Dienste eines öffentlichen Telefonnetzes, ist derzeit vor allem aus Kostengründen für Auslandsgespräche interessant. Typische Call Center sind jedoch im Moment weniger an derartigen Lösungen interessiert, da vor allem die mangelnde Zuverlässigkeit der Internet-Verbindungen dem Anspruch an den Service gegenüber dem Kunden im Wege stehen.

Wesentlich interessanter ist dagegen die Einbeziehung des World Wide Web in das Service-Angebot eines Call Centers. Die Idee hier ist, dass der Anrufer zuerst Informationen über das WWW erhält und dabei die notwendigen Daten mit einem Server ausgetauscht werden. Wünscht er daraufhin ein individuelles Gespräch, so kann er dies über eine Funktion der Web-Seite initiieren. Anhand der bereits erfassten Daten sowie zentral gespeicherter Informationen stellt dann eine CTI-Applikation die Verbindung zwischen dem Kunden und einem gerade verfügbaren Agenten her. Dieser erhält dann beispielsweise eine Kopie der aktuellen Web-Seite an seinem Arbeitsplatz und kann so den Anrufer individuell beraten. In diesem Zusammenhang rückt auch die Bedeutung von JAVA in den Vordergrund, momentan die vorherrschende Programmiersprache für Applikationen des WWW. Denkbar ist auch, dass in Zukunft hier sowohl die Kommunikation des WWW als auch das Gespräch selbst über das Internet abgewickelt werden, wodurch der Rückruf durch Komponenten des Call Centers überflüssig wäre. Damit wäre eine noch bessere Synchronisierung zwischen Daten und Anruf gewährleistet.

3 ASPEKTE EINES FORTSCHRITTLICHEN CTI-SYSTEMS

Im letzten Kapitel wurden die wichtigsten Bereiche eines Call Centers, seine Prozesse und Technologien, die Grundfunktionalität eines Telefonsystems sowie die Möglichkeiten und Anforderungen an die CTI zusammengefasst. Nun stellt sich die Frage, was ein fortschrittliches, also in Funktion und Entwurf bisherige Anätze übertreffendes System in diesem Umfeld auszeichnet. Inhalt dieses Abschnittes ist es demnach, die wesentlichen Aspekte zu definieren, die ein modernes und innovatives IT-System vor diesem Hintergrund auszeichnen.

Als innovatives CTI-System findet es sich am Schnittpunkt zwischen Telefonanlage, bestehenden IT-Systemen, der CTI-Grundfunktionalität und den aus den Geschäftsprozessen resultierenden Anforderungen einer konkreten CTI-Anwendung. Die Technologie eines modernen Telefonsystems besitzt dabei von sich aus einen verteilten Charakter. Durch die Integration dieser Systeme in den IT-Bereich entsteht demnach automatisch ein verteiltes System. Eine innovative Lösung muss sich daher ebenfalls mit den Problemen und Möglichkeiten verteilter Anwendungen auseinandersetzen.

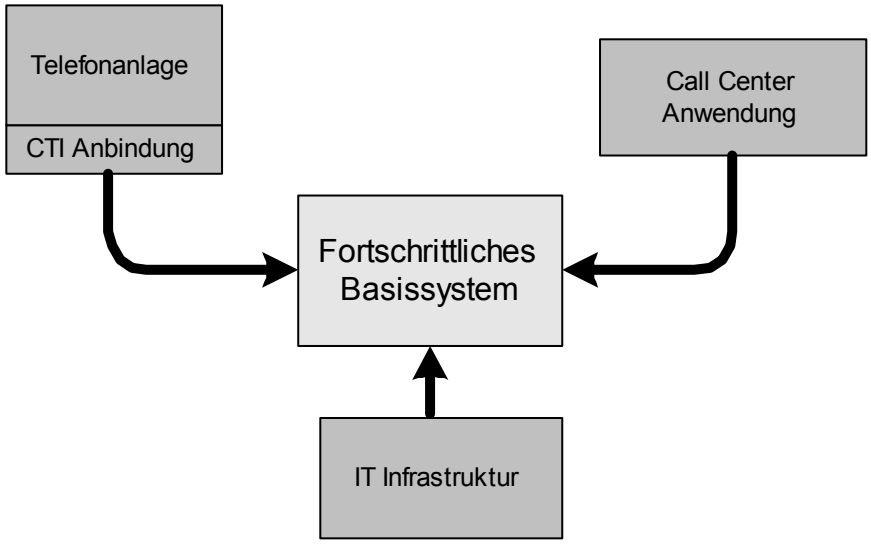


Abbildung 3-1 Positionierung des Systems

Der Ansatz ist deshalb, diese Anforderungen nicht in einzelnen Individuallösungen sondern möglichst in einem Basissystem abzudecken, das damit eine solide und möglichst allgemein spezifizierte Grundlage für die verschiedenen Belange moderner Lösungen eines Call Centers darstellt. Die einzelnen Aspekte, die ein derartiges System auszeichnen sollten, werden nun nachfolgend etwas detaillierter beschrieben.

3.1 Konzept und Architektur

Wichtige Grundlage eines anspruchsvollen Frameworks und damit ein Aspekt dieser Arbeit ist die Erstellung eines den Anforderungen entsprechenden IT-Konzepts sowie der Entwurf einer geeigneten Systemarchitektur.

3.1.1 Konzeption und Technologie

Erste Forderung an die Konzeption ist die Bereitstellung der Funktionalität, auf die nachfolgend noch genauer eingegangen wird. Es fehlt jedoch bei vielen derzeit bekannten CTI-Lösungen im Call Center eine allgemeine und teilweise auch theoretisch fundierte Konzeption. Die Lösungen konzentrieren sich zu einem großen Teil auf die Umsetzung ganz spezieller Funktionen mit Hilfe der in einer Systemumgebung

vorhandenen Technologie. Da es sich jedoch in den meisten Fällen um eine verteilte Anwendung handelt, liegt es im Grunde nahe, die hier bereits vorliegenden Erkenntnisse und Ansätze für CTI-Lösungen weiter zu verfolgen.

Daraus ergibt sich also, dass ein fortschrittliches IT-System für Call Center auf einer allgemeinen und soweit möglich auch theoretisch ausreichend fundierten Konzeption beruhen sollte. Daneben ist eine Unabhängigkeit von individuellen Technologien und Systemen insoweit anzustreben, als dadurch eine konkrete Realisierung nicht ausgeschlossen wird. Nicht zuletzt muss das System entsprechend erweiterbar und damit als Basis für möglichst viele CTI-Anwendungen einsetzbar sein. Es muss also ein Framework für individuelle Lösungen im Call Center darstellen.

3.1.2 Integrationsaspekte

Wichtiger Aspekt der Architektur ist die Integration in das bestehende IT-System eines Call Centers. Wie eingangs bereits erläutert, stützen sich diese in der Regel auf ein bzw. mehrere Systeme, welche wesentliche Teile der Geschäftsprozesse abbilden und somit für den Betrieb des Call Centers unerlässlich sind. Auf diese Systeme, die häufig als Legacy-Systeme bereits vorhanden sind, muss eine CTI-Anwendung zurückgreifen, um den Mitarbeiter während des Gesprächs ausreichend unterstützen zu können. Einige dieser Systeme seien an dieser Stelle kurz angesprochen.

Backend-Basissysteme

Systeme wie etwa die interne Kontoführung einer Bank, das Warenwirtschaftssystem eines Handelsunternehmens oder die Lagerverwaltung im Bereich des Versandhandels liefern wesentliche Informationen, die für den Service am Telefon von Bedeutung sind. Ein Zugriff auf die hier verfügbaren Informationen ist also für jede CTI-Anwendung ein wichtiges Thema. Handelt es sich bei diesen Systemen um transaktionsorientierte Applikationen, so spielt das Thema der verteilten Transaktionen hier ebenfalls eine Rolle.

Datenbank

Zentrale Komponente der meisten Call Center ist ein Datenbanksystem, das Informationen wie etwa die Historie aller Anrufe, Kundendaten, statistische Werte und dergleichen mehr speichert. Client-Anwendungen sind in der Regel auf diese Informationen angewiesen. Deshalb beinhaltet der Aspekt der Integration von CTI in dieser Hinsicht auch immer eine Anbindung an ein oder mehrere Datenbanksysteme. Dazu gehört auch hier der Aspekt der Transaktionssicherheit.

IVR-VRU

Die in Call Centern üblichen Sprachcomputer treten im Grunde wie Agenten auf, die Anrufe entgegennehmen, Informationen sammeln, Aktionen durchführen und den Anruf gegebenenfalls weiterleiten. In dieser Hinsicht unterscheidet sich die Kommunikation also nicht von der, die zwischen den Applikationen einzelner Agenten stattfindet. Da jedoch kein Gespräch für einen Informationsaustausch möglich ist, sind der Datenaustausch sowie ein geeignetes Protokoll hier umso wichtiger. Sehr häufig wird nämlich eine Weiterleitung von einem IVR-System zu einem Agenten vom Anrufer gewünscht, wenn er Probleme mit der Funktionalität des Sprachcomputers hat. In diesem Falle ist es sehr hilfreich, wenn der Agent bereits vorher möglichst viele Informationen über die Ursachen der Probleme erhält.

Sieht man das Call Center etwas allgemeiner als Komponente eines Unternehmens, so wird der Bereich des Workflowmanagements interessant. Da Anrufe von außen sehr häufig zu Aufträgen und damit zu einem Workflow in dem Unternehmen führen, ist zu überlegen, in welcher Art und Weise die CTI-Lösung hier bereits Schnittstellen zur Verfügung stellen sollte. Somit wäre gewährleistet, dass wichtige Informationen über den Anrufer, die durch CTI bekannt sind, in das Workflowsystem übernommen werden.

Diese Beispiele legen nahe, dass das Basissystem in der Lage sein muss, mit verschiedenen Systemen in einer heterogenen Welt zu kommunizieren bzw. eine Interaktion der CTI-Anwendung mit diesen Systemen zu unterstützen.

3.1.3 Integration weiterer Medien

Die Integration weiterer Medien wie Fax, Video sowie eine Anbindung an das Internet sind derzeit sehr aktuelle Themen für die Entwicklung neuer Call Center. Dementsprechend sollte ein hier angesiedeltes IT-System diese Themen nicht unberücksichtigt lassen. Im Bereich Internet ist vor allem die Integration in das World Wide Web sehr interessant (siehe 2.6.3). Was die Konzeption des skizzierten Lösungsansatzes angeht, so würden hier weitere Medien entsprechend neue Module innerhalb der verteilten Anwendung nach sich ziehen.

3.2 Funktionalität

Neben den Aspekten des Konzepts sowie der Architektur muss das System Funktionen beinhalten, die modernen CTI-Anwendungen zur Verfügung stehen sollten. Auch hier muss das System mehrere Vorgaben berücksichtigen, etwa die Möglichkeiten, die das Telefonsystem sowie die CTI-Basissoftware bieten oder aber die Ansprüche, die sich aus den Prozessen und damit der Funktionalität der konkreten CTI-Anwendung ableiten lassen.

Die im Folgenden definierten Funktionen sind dabei Ergebnisse, die einerseits aus den eingangs kurz erläuterten Modellen von Call Centern und den dort vorherrschenden Arbeitsabläufen sowie außerdem aus der Erfahrung bzgl. der konkreten Entwicklung von CTI-Anwendungen stammen.

3.2.1 Verbindung Anruf – Daten

Die Aufgabe, einem Anruf bestimmte Daten zuzuordnen, tauchte bereits mehrfach auf und ist eine der wesentlichen Merkmale von CTI. Ein allgemeiner Ansatz ist hier, einen bestimmten Satz von Informationen fest mit einem Anruf zu verbinden. Aufgabe des CTI-Servers bzw. der CTI-Middleware ist es dabei, diese Information sofort, d.h. mit möglichst geringer Verzögerung, an einen Rechner zu schicken, dessen Telefon den Anruf zugestellt bekommt. Die Daten, die hier mitgeliefert werden, haben typischerweise zwei Verwendungszwecke. So ist in der Regel ein eindeutiger Identifikator enthalten, über den die Applikation des Agenten aus einer Datenbank bzw. weiteren Systemen alle notwendigen Informationen des Kunden abrufen kann. Darüber hinaus sind hier auch die Informationen untergebracht, die, falls vorhanden, dem Agenten für das Telefongespräch hilfreiche Daten (Name, Charakteristik etc.) zur Verfügung stellen. Die Tatsache, dass diese Daten praktisch zeitgleich mit dem Anruf auf dem Bildschirm des Agenten erscheinen, ist hier ein wesentliches Kriterium.

Eine besondere Bedeutung haben diese Daten bei einer Weiterleitung. Dadurch wird es möglich, die Historie des Anrufes mit allen bereits durchgeführten Aktionen entweder direkt in diesen Daten oder aber in einer zentralen Datenbank zu speichern und jedem der beteiligten Agenten zur Verfügung zu stellen.

Bestehende CTI-Systeme bieten unterschiedliche Mechanismen für die Versorgung eines Anrufs mit Daten an. So verknüpft VESP von Nabnasset (inzwischen Quintus) eine Struktur (VDU *voice data unit*) mit jedem Anruf, die Basisinformationen wie etwa Telefonnummer (CLI) sowie die gewählte Nummer (DNIS) des Anrufers enthält, und deren Felder von Anwendungen individuell gefüllt sowie mit einem eigenen Identifikator versehen werden können. Entsprechende Dienste ermöglichen das Setzen und Lesen solcher Felder. Andere Systeme bieten hier lediglich einen Datenbereich an, den Anwendungen als unstrukturierten Datenstring verwenden können. In den meisten Fällen liegt aus heutiger Sicht die Verantwortung für die Verwaltung dieser Daten sowie die Definition geeigneter Protokolle in der Verantwortung der nutzenden Applikationen.

Aufbauend auf den Diensten des CTI-Systems ist die Einführung eines mächtigeren und komfortableren Konzepts hier sinnvoll. Grundgedanke ist dabei, den Ansatz von verteilten Objekten zu integrieren. In einem ersten Ansatz lassen sich einige Anforderungen aufzählen, die dabei zu berücksichtigen sind. Die flexible Definition individueller Objekte, aufbauend auf einem generischen Basisobjekt, ist die Voraussetzung für eine komfortable und mächtige Datenversorgung eines Telefonanrufs. Derartig definierte Objek-

te bzw. deren Instanzen sind lokal verfügbar, sobald ein Anruf an der Workstation eintrifft. Sie bieten den Applikationen Basisdienste an, die eine Verwaltung der Anrufdaten möglich machen.

3.2.2 Unterstützung komplexer Telefonabläufe

In Kapitel 2.5.2 wurden einige der elementaren Abläufe einer modernen Telefonanlage spezifiziert. Es geht also nun darum, diese Abläufe auf ein verteiltes System abzubilden. Dies betrifft einerseits die Schnittstelle des Systems zu einem CTI-Server, über die diese Abläufe initiiert werden können. Andererseits müssen Prozesse wie etwa eine Weiterleitung auch innerhalb des Systems entsprechend unterstützt werden, wenn es etwa um das Thema der Synchronisation geht.

Für die Realisierung solch wichtiger Abläufe, wie sie in modernen Call Centern auftreten, bieten momentan die typischen CTI Lösungen keine allgemeingültigen Lösungsansätze. Sie konzentrieren sich vielmehr darauf, die grundsätzlichen Funktionen einer Telefonanlage sowie die zu einem Anruf erhältliche Daten der Client Applikation zur Verfügung zu stellen. Hauptschwerpunkt liegt dabei eher auf Zuverlässigkeit, Performance und Skalierbarkeit als auf der Unterstützung solcher Abläufe.

Zu diesem Zwecke sind nachfolgend einige dieser Abläufe aufgelistet, die ein innovatives System unterstützen sollte.

Single Step Transfer

Eine Weiterleitung zu einer Gruppe von Mitarbeitern oder zu einem Sprachcomputer erfordert in der Regel keine Aktion des weiterleitenden Agenten mehr. Aus Gründen der Performance ist deshalb ein automatisches Auflegen häufig sinnvoll. Dies bedeutet, der Dienst erledigt das Initiieren und Abschließen des gewünschten Transfers in einem Schritt, wobei die möglichen Fehlerfälle wie die Nichterreichbarkeit einzelner Teilnehmergruppen behandelt werden müssen.

Direct Transfer

Die direkte Weiterleitung überlässt dem Agenten die Kontrolle des Anrufs. Das heißt, er kann vor der endgültigen Weiterleitung mit dem neuen Gesprächspartner reden und eventuell den Transfer auch abbrechen. Ein nicht durchgeführter Transfer erfüllt dann den Zweck des Rückfragens, der ebenfalls eine Grundfunktionalität jeder Telefonanlage darstellt.

Weiterleitung mit Rückgabe

Bei einer Weiterleitung mit Rückgabe kann ein Anruf wieder an den ursprünglichen Mitarbeiter zurück vermittelt werden. Solche Funktionen werden im allgemeinen nicht von CTI-Systemen unterstützt, da sie der Funktionsweise von größeren und effizienten Call Centern im Grunde nicht entsprechen. Hier wird eher der Ansatz verfolgt, dass der Mitarbeiter ein möglichst anonymer Ansprechpartner bleibt. Denn nur dadurch lassen sich sehr große Mengen von Anrufen über Warteschlangen gleichmäßig und flexibel verteilen. Jedoch zeigt die Praxis auch in größeren Unternehmen, dass auch hier häufig eine Identifizierung eines Mitarbeiters und damit eine persönlichere Beratung durch den Kunden gewünscht wird. Muss in diesem Falle das Gespräch weitergeleitet werden, zum Beispiel an einen Sprachcomputer oder aber an eine Fachabteilung, so ist die Möglichkeit, wieder zum ursprünglichen Teilnehmer zurückkehren zu können, ein sehr interessanter Dienst. Dazu ist eine Reihe von Aspekten zu berücksichtigen. So müssen ausreichende Daten dem Anruf zugeordnet werden, so dass eine eindeutige Identifizierung des ursprünglichen Agenten möglich ist. Dies ist vor allem im Hinblick auf verteilte Call Center interessant, da hier in mehreren Filialen die Nebenstellennummer allein kein eindeutiges Kriterium mehr ist.

Synchroner Transfer

Ein Sonderfall des eben angesprochenen Themas, den eine Telefonanlage in der Regel ebenfalls nicht bietet, ist ein Ablauf, der hier als synchroner Transfer bezeichnet wird, wobei dies kein allgemein bekannter Begriff im Bereich CTI ist. Gemeint ist damit die Anforderung, einen Anruf kurzfristig an einen anderen Teilnehmer weiter zu geben mit der Option, ihn in jedem Fall wieder zurückvermitteln zu können. Beispiel einer Bank ist die Vermittlung zu einem Sprachcomputer, der dem Kunden die Möglichkeit der Legi-

timation bietet, die aus Sicherheitsgründen häufig nicht im Gespräch mit dem Agenten möglich ist. Besteht der Wunsch des Kunden, wieder mit dem ursprünglichen Bearbeiter zu sprechen, so ist die Notwendigkeit eines synchronen Transfers gegeben. Dazu sind also die Voraussetzungen des eben genannten Punktes zu erfüllen. Weiterhin ist der ursprüngliche Agent bzw. seine Anwendung in einen Wartezustand zu versetzen sowie sicherzustellen, dass er in der Zwischenzeit keine anderen Anrufe erhält. In diesem Zusammenhang sind dann auch entsprechende Mechanismen für Timeout-Zustände zu berücksichtigen.

Konferenz

Die Telefonkonferenz mit mehreren Teilnehmern birgt grundsätzlich keine besonderen Anforderungen an das CTI-System, da diese Funktionalität von allen Telefonanlagen zur Verfügung gestellt wird. Erst die Verbindung des Anrufs mit individuellen Daten lässt hier interessante Aspekte auftreten. Von einem fortschrittlichen System ist zu erwarten, dass es die Synchronisation der Zugriffe aller Konferenzteilnehmer auf diese Daten gewährleistet. In diesem Zusammenhang sind mehrere Ausprägungen einer Konferenz denkbar, je nachdem ob alle Partner gleichberechtigt bzgl. Schreib- und Leserechten sind oder eine Art Master-Slave-Verhältnis besteht.

Weitere Funktionen

Neben komplexen Abläufen des Telefonbetriebs ist die Integration eines mit entsprechenden Zugriffsdiensten ausgestatteten Telefonverzeichnisses eine häufig notwendige Funktionalität. Die Verwaltung der Nummern aller Agenten ist bei größeren Call Centern eine absolut wichtige Komponente, die genutzt wird, wenn es um Rückfragen oder sehr gezielte Weiterleitungen geht. Dabei identifizieren moderne Telefonanlagen bzw. ACD-Systeme in der Regel Agenten über eigene Kennungen, mit denen man sich am System anmelden kann. Ein Gespräch kann somit zu einem bestimmten Agenten aufgebaut werden, unabhängig davon, an welchem Arbeitsplatz er sich gerade befindet. Ziel und somit Aufgabe einer CTI-Anwendung ist es demnach, einen geeigneten Mitarbeiter eines Call Centers schnell und komfortabel zu finden und anschließend eine Telefonverbindung herzustellen. Dabei sind Informationen des ACD-Systems bzw. der Telefonanlage sehr hilfreich, da so Agenten, die gerade ein Gespräch führen oder Bearbeitergruppen, die momentan ausgelastet sind, erkannt werden können. Dies kann dann dem Benutzer der Anwendung angezeigt bzw. die Suche entsprechend angepasst werden. Ergebnis ist also eine dynamische Telefonliste, die zentral in einer Datenbank zu verwalten ist. Dies ist gerade im Falle verteilter Call Center eine anspruchsvolle Aufgabe, nicht zuletzt durch die sich laufend ändernden Zustände der Agenten.

Eine weitere Funktionalität liegt in der Verwaltung mehrerer Telefonleitungen. Üblicherweise überlassen Systeme hier die Kontrolle mehrerer Leitungen dem Agenten. Da dies aber bei einer höheren Anzahl durchaus komplex werden kann, vor allem wenn man Funktionen wie Konferenz oder Weiterleitung integriert, ist eine Unterstützung durch CTI hier nicht uninteressant.

3.2.3 Allgemeiner Zustandsautomat

Während CTI-Systeme hauptsächlich die Ereignisse der Telefonanlage berücksichtigen und sich auf den Austausch von Nachrichten bzw. Kommandos zwischen den Client-Anwendungen und dem Switch konzentrieren, beinhaltet die Steuerung eines Telefons immer die Verwaltung eines Zustandsautomaten. Typische Zustände sind dabei *online, offline, onhold, ringing* etc., die das Verhalten der Anwendung entscheidend beeinflussen. In der Regel verwalten die CTI-Anwendungen derartige Automaten selbst, wobei sie individuell auf die einzelnen Zustände reagieren. Das Anzeigen der Kundendaten im Zuge des Screen-Pop erfolgt in der Regel im Zustand *ringing*.

Ein möglichst allgemeingültiger Zustandsautomat, der die wichtigsten Übergänge, die im Zusammenhang mit einer Telefonanlage auftreten können, beinhaltet, sollte daher eine Komponente einer innovativen CTI-Lösung sein.

3.3 Einsatz in der Praxis

Neben den bisher dargestellten Kriterien, die vor allem Konzeption und Funktionalität betrafen, darf der praktische Einsatz nicht völlig außer Acht gelassen werden. Das bedeutet, dass eine Lösung auf Basis dieser innovativen Ansätze auch gegenüber einem konkreten Einsatz in einer vorgegebenen Umgebung bewertet werden muss. Dazu gehört zum einen die Unterstützung vorhandener und in der Industrie verfügbarer Basistechnologien. Darüber hinaus gibt das Telefonieren an sich für CTI-Anwendungen gewisse Zeitschranken vor, die vor allem in einem Call Center, wenn man von einem hohen Anrufvolumen ausgeht, zu berücksichtigen sind. So ist die Zeit, die ein Agent hat, um sich auf einen neuen Anruf vorzubereiten, der ihm zugestellt wurde, nur sehr kurz. Dementsprechend schnell sollten erste Daten über diesen Anruf zur Verfügung stehen. Während des Gesprächs lassen sich dagegen Wartezeiten im Bereich von Sekunden relativ gut überbrücken. Wie in jedem DV-System gibt es jedoch auch hier Grenzen, die das Antwortzeitverhalten nicht überschreiten sollte. Für ein CTI-System bedeutet dies entsprechende Anforderungen vor allem an die Netzwerkperformance. Die Aufgabe, Daten gleichzeitig mit einem Anruf dem Client zuzustellen, wird in der Regel von der CTI-Middleware übernommen. Jedoch erfordern Dienste wie der Aufbau eines Begrüßungsbildschirms (Screen Pop) weitere und unter Umständen aufwändige Zugriffe auf eine Datenbank oder ein Fremdsystem. In diesem Falle ist eine ausreichende Performance ein ganz entscheidendes Thema jeder fortschrittlichen CTI-Lösung.

3.4 Abgrenzung zu verwandten Ansätzen

Der dargestellte Ansatz ist in erster Linie verwandt mit Lösungen, die als CTI-Middleware die Basis für Softwarelösungen im Call Center darstellen. Der Softwaremarkt ist in diesem Bereich derzeit relativ uneinheitlich, was nicht zuletzt an gegenseitigen Übernahmen und Akquisitionen der namhaften Hersteller liegt. Als führendes Produkt ist hier *CallPath* zu nennen, das ursprünglich von IBM entwickelt inzwischen aber von der Firma Genesys übernommen wurde, die selbst mit ihrem T-Server Framework eine der führenden CTI-Plattformen anbieten. Als weitere Lösung sei an dieser Stelle noch VESP (*Voice Enhanced Services Platform)* erwähnt, eine auf CORBA basierende CTI-Middleware, die ursprünglich von der Firma Nabnasset entwickelt und dann von Quintus, einem Anbieter von Helpdesk-Systemen, übernommen wurde. Diese Produkte konzentrieren sich als Middleware-Lösung in erster Linie um die transparente Anbindung an möglichst viele Telekommunikationssysteme und bieten hier neben der reinen Anrufkontrolle vor allem Dienste an, die die Administration sowie das Management des Telefonsystems betreffen. Der hier dargestellte Ansatz grenzt sich dem gegenüber dahingehend ab, dass er die Implementierung eines verteilten Anwendungssystems in einem Call Center in den Vordergrund stellt und daher auf derartige Middleware-Produkte als Basissystem aufsetzt.

Neben dem transparenten Zugang zu diversen Telefonsystemen bieten diese Lösungen jedoch auch immer mehr Funktionen an, die dem Bereich des CRM zuzuordnen sind. Als Marktführer in diesem Bereich bieten hier Firmen wie Siebel [Siebel01] oder Oracle [Oracle01] Standardsoftware an. Derartige Lösungen sind in der Regel sehr umfassend und komplex, da sie versuchen, die gesamte Beziehung zwischen Unternehmen und Kunde inklusive aller damit verbundenen Geschäftsabläufe zu unterstützen. Der Kontakt über ein Call Center ist dabei nur einer von vielen Schnittstellen nach außen und steht somit auch nicht im Vordergrund solcher Ansätze. Das in dieser Arbeit angestrebte Framework sowie die darauf aufbauenden Lösungen stellen hier also eine bestimmte Komponente dar, die als wichtiger Teil eines gesamten CRM-Systems gesehen werden kann und den konkreten Anruf eines Kunden als Teil des langen Workflows der Beziehung Kunde/Unternehmen im Fokus hat. Einen guten Überblick über das derzeitige Angebot von Standardsoftware in diesem Bereich gibt [Born01].

Betrachtet man die komplexeren Funktionen wie Weiterleitung und Konferenz sowie die hier gestellten Anforderungen, so kommt man zum Teil in den Bereich der computergestützten Gruppenarbeit *(CSCW).* Einen Überblick sowie eine Einführung in dieses Thema gibt [Borghoff98]. Interessant sind in diesem Zusammenhang auch [Resnick93] sowie einige weiteren Arbeiten dieses Autors, der sich konkret mit dem Thema Gruppenarbeit über das Telefon beschäftigt hat. Hier steht allerdings nicht das Call Center sondern die direkte Kooperation von Benutzern z.B. über ein IVR-System im Vordergrund.

Was den Entwurf objektorientierter Frameworks als Basis für individuelle Anwendungssysteme angeht, werden in Arbeiten wie [Bern96], [Fayad97] und [Foote88] dazu interessante Grundlagen erarbeitet, die auch in diesem Zusammenhang an geeigneter Stelle aufgegriffen werden. Allerdings soll das hier dargestellte Framework eher einen konkretes Lösungsansatz im Umfeld eines Call Centers darstellen, ohne den Framework-Ansatz an sich weiter zu verfeinern.

Im Zusammenhang mit Arbeiten zum Thema der unternehmensweiten verteilten Anwendungen sei an dieser Stelle noch das Designmodell erwähnt, das zusammen mit der Enterprise Edition von Java entstanden ist und in [SUN00] zusammengefasst wird.

3.5 Zusammenfassung

Ausgehend von den Vorgaben, die sich aus den Geschäftsprozessen eines Call Centers sowie der Grundfunktionalität einer Telefonanlage für eine moderne CTI-Lösung ableiten lassen, wurden in diesem Abschnitt eine Reihe von Anforderungen definiert, die an ein fortschrittliches Basissystem gestellt werden sollten. Wie in Tabelle 3-1 dargestellt, umfassen diese zum einen das Konzept sowie die Architektur des Systems, die darauf aufbauende, konkrete Funktionalität sowie Aspekte, die den Einsatz in der Praxis betreffen.

Konzept und Architektur	Allgemeine Konzeption mit formaler und theoretischer Basis
	Unabhängigkeit von Systemen und Technologien
	Allgemeines Framework für vielfältige CTI-Anwendungen
Funktionalität	Verknüpfung von Anruf und Daten
	Unterstützung komplexer Telefonfunktionen
	Allgemeiner Zustandsautomat
Einsatz in der Praxis	Unterstützung verfügbarer Basistechnologien
	Ausreichende Performance

Tabelle 3-1 Anforderungen an ein innovatives System

Diese Anforderungen sind die Grundlage für den Entwurf eines Systems, das auf der Basis der CTI-Technologie anspruchsvolle Aufgaben in einem Call Center lösen kann.

4 BASISKONZEPTE

4.1 Zustandsautomaten

Die Funktion einer Telefonanlage und damit die eines CTI-Systems wird in hohem Maße vom aktuellen Zustand einzelner Komponenten beeinflusst.

Deshalb eignet sich der Ansatz endlicher Zustandsautomaten *(finite state machines)* für die Beschreibung des Verhaltens sowohl der Basissysteme als auch des konzipierten Frameworks. Als Ausgangsbasis für die weitere Verwendung werden hier zwei der in der Informatik sehr häufig verwendeten Automatenmodelle, der Mealy-Automat sowie der Moore-Automat, kurz beschrieben und eine der üblichen Notationen eingeführt.

4.1.1 Mealy-Automat

Mathematisch ist ein Mealy-Automat ein Sechstupel

$$M=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$$

mit

- Q einer endlichen Menge von Zuständen,
- Σ einem endlichen Eingabealphabet,
- Δ einem endlichen Ausgabealphabet,
- δ einer Übergangsfunktion, die $Q \times \Sigma$ auf Q abbildet,
- λ einer Abbildung von $Q \times \Sigma$ auf Δ sowie
- q_0 einem Anfangszustand.

Ausgehend von einem Startzustand liefert $\delta(q,a)$ eines Mealy-Automaten also für jeden Zustand q und jede Eingabe a einen Folgezustand. Die Abbildung λ liefert dazu für diesen Übergang die entsprechende Ausgabe.

4.1.2 Moore-Automat

Sehr ähnlich zum Automatenmodell von Mealy wird in dieser Variante die Ausgabe jedoch nicht mit einem Zustandsübergang, sondern direkt mit den einzelnen Zuständen assoziiert. λ ist hier folglich eine Abbildung von Q auf Δ, das heißt jedem Zustand wird eine Aktion bzw. eine Ausgabe zugeordnet. Wie der Mealy-Automat ist auch der Moore-Automat deterministisch, das heißt er bildet einen Zustand zusammen mit einer Eingabe genau auf einen Folgezustand ab.

Wichtig zu bemerken ist dabei, dass beide Modelle äquivalent sind und jederzeit ineinander überführt werden können. Demnach ist zu diesem Zeitpunkt auch noch keine endgültige Entscheidung über Einsatz des jeweiligen Modells notwendig. Der Schwerpunkt wird allerdings auf dem Modell von Mealy liegen.

4.1.3 Notation

Ein Zustandsautomat kann in verschiedener Art und Weise dargestellt werden, etwa als Zustandstabelle, als Zustandsmatrix oder als Diagramm. In der Informatik verbreitet ist die Darstellung mit den in Abbildung 4-1 angedeuteten Symbolen. Zustände werden dabei als Kreise dargestellt, Übergänge als beschriftete Pfeile. Die Beschriftung gibt dabei sowohl die Eingabe bzw. das Ereignis an, die den Übergang

in den nächsten Zustand auslöst, als auch die eventuell auftretende Ausgabe, die während dieses Übergangs auftritt. Hiermit kann auch eine durchzuführende Aktion beschrieben werden.

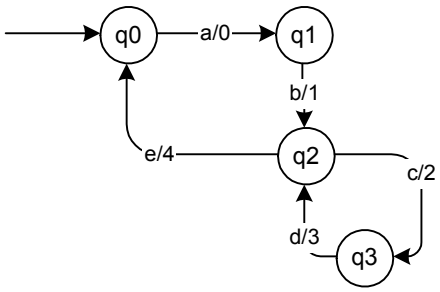


Abbildung 4-1 Graphische Darstellung eines endlichen Automaten

Als besondere Zustände werden der Startzustand sowie ein oder mehrere Endzustände extra gekennzeichnet. Üblich ist hier, Endzustände durch einen doppelten Kreis zu markieren und den Startzustand entweder direkt zu beschriften oder durch einen eigenen Pfeil zu kennzeichnen (siehe dazu auch [Balzert96a] und [Duden93]).

Der in Abbildung 4-1 skizzierte Mealy-Automat würde formal folgendermaßen dargestellt werden:

$Q = \{q0,q1,q2,q3\}$
$\Sigma = \{a,b,c,d,e\}$
$\Delta = \{0,1,2,3,4\}$
δ $\delta(q0,a)=q1$, $\delta(q1,b) = q2$, $\delta(q2,c)=q3$, $\delta(q2,e)=q0$, $\delta(q3,d)=q2$
λ $\lambda(q0,a)=0$, $\lambda(q1,b)=1$, $\lambda(q2,c)=2$, $\lambda(q2,e)=4$, $\lambda(q3,d)=3$
$q_0 = q0$

4.1.4 Weitere Modelle

Neben den eben dargestellten Modellen existieren noch weitere Varianten von Zustandsautomaten wie zum Beispiel Hybride Zustandsautomaten (auch Harel-Automaten genannt), die eine Kombination von Mealy- sowie Moore-Automaten darstellen. Für theoretische Betrachtungen und Beweise sehr nützlich sind auch nichtdeterministische Automaten, bei denen eine Eingabe verschiedene Zustandsübergänge auslösen kann. Solche werden unter anderem in [Hopcroft79] erläutert.

Diese Modelle sind jedoch für die Anforderungen dieser Arbeit im weiteren nicht relevant.

4.2 Petrinetze

Petrinetze sind ein Modell zur Darstellung und Analyse nebenläufiger und nichtdeterministischer Vorgänge. Vorgeschlagen wurde es 1962 von C.A. Petri in seiner Dissertation „Kommunikation mit Automaten". Das Modell eignet sich besonders zur Modellierung und Analyse dynamischer Systeme mit kooperierenden Prozessen. Einsatzgebiet dieser Netze sind deshalb insbesondere verteilte, ereignisorientierte Systeme, aber auch Betriebssysteme sowie allgemeine Organisationsabläufe. Hierzu gehören beispielsweise Herstellungsverfahren oder Büroabläufe, deren Organisation und Modellierung auch in dem Bereich *Workflow* betrachtet werden.

4.2.1 Definition

Ein Petrinetz ist ein gerichteter Graph, der zwei verschiedene Arten von Knoten besitzt, *Stellen* und *Transitionen*. Die Stellen entsprechen dabei einer Ablage für Informationen, die Transitionen beschreiben die

Verarbeitung dieser. Die Kanten dieses Graphen führen dabei immer von Knoten des einen Typs zu Knoten des anderen. Der Graph selbst spiegelt dabei die statische Ablaufstruktur des Systems wider. Um die Dynamik des Systems zu berücksichtigen, werden Stellen mit Objekten, sog. *Marken* belegt.

Formal ist ein Netz demnach ein 5-Tupel

$$P=(S,T,A,E,M) \text{ mit:}$$

- S einer nichtleeren Menge von Stellen,
- T einer nichtleeren Menge von Transitionen,
- $S \cap T = \emptyset$
- $A \subset S \times T$ einer Menge von Kanten, die von Stellen zu Transitionen führen,
- $E \subset T \times S$ einer Menge von Kanten, die von Transitionen zu Stellen führen sowie
- $M: S \rightarrow N0$ einer Markierungsfunktion, die jeder Stelle anfänglich eine Menge von Markierungen zuordnet.

Die Dynamik des Systems ergibt sich nun durch die für Petrinetze festgelegte *Schaltregel*, die den Bewegungsablauf von Marken innerhalb des Netzes festlegt. Zur Definition dieser Regel werden die Begriffe *Vorbereich* sowie *Nachbereich* einer Transition eingeführt.

So wird für jedes $t \in T$

$${}^\bullet t = \{s \in S \mid (s,t) \in A\} \text{ als } \textit{Vorbereich} \text{ von t und}$$

$$t^\bullet = \{s \in S \mid (t,s) \in E\} \text{ als } \textit{Nachbereich} \text{ von t}$$

definiert. ${}^\bullet t$ ist also die Menge der Stellen, von denen Kanten zur Transition t führen. Diese Menge nennt man auch *Eingabestellen* von t. $t^\bullet$ ist analog dazu die Menge von Stellen, zu denen eine Kante von t führt, die auch als *Ausgabestellen* von t bezeichnet werden. Die Schaltregel eines Petrinetzes ist nun folgendermaßen definiert:

Eine Transition kann schalten, wenn

$$M(s) > 0 \; \forall \; s \in {}^\bullet t,$$

wenn also alle Stellen im Vorbereich dieser Transition mindestens eine Markierung besitzen. Durch das Schalten einer Transition ergibt sich eine Bewegung der Markierungen, die sich durch eine neue Belegung, also eine neue Folgemarkierung M' darstellt. Dabei ist

$$M'(s) = M(s) + 1 \text{ , falls } s \in t^\bullet, \text{ aber } s \notin {}^\bullet t,$$

$$M'(s) = M(s) - 1, \text{ falls } s \in {}^\bullet t, \text{ aber } s \notin t^\bullet,$$

$$M'(s) = M(s), \text{ sonst.}$$

Anschaulich bedeutet dies, jeder Eingabestelle wird eine Markierung entzogen und jeder Ausgabestelle wird eine Markierung hinzugefügt.

Sind alle Stellen im Vorbereich belegt, jedoch mindestens eine Stelle im Nachbereich, so spricht man von einer Kontaktsituation, da hier die Transition nicht schalten kann, obwohl alle Vorbedingungen erfüllt sind.

Notation

Abbildung 4-2 zeigt die typischen Elemente eines Petrinetzes in der allgemein üblichen, graphischen Notation. Dabei werden Stellen als Kreise, Transitionen als Rechtecke und Markierungen als kleinere, schwarze Kreise dargestellt.

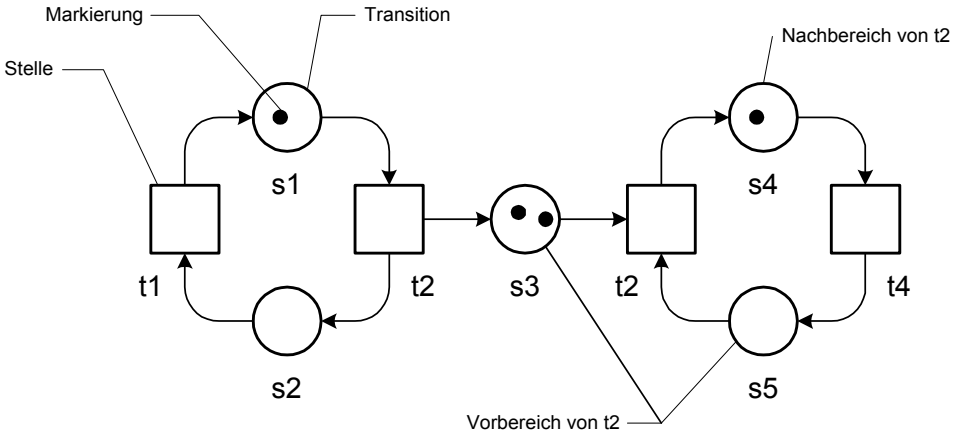


Abbildung 4-2 Notation eines Petrinetzes

In Abbildung 4-3 kann man nun sehen, wie sich die aktuelle Markierung eines Petrinetzes durch das Schalten von Transitionen dynamisch verändert. Im ersten Schritt schaltet Transition t2, wodurch die Stelle s1 ihre Markierung verliert und die Stellen s2 sowie s3 jeweils eine neue erhalten. Außerdem schaltet t4, wodurch die Markierung von s4 nach s5 wandert. Im zweiten Schritt schaltet dann t2, s3 sowie s5 geben eine Markierung ab und s4 erhält eine neue. In diesem Zustand sind sowohl t4 als auch t1 aktiviert und könnten als nächstes schalten. Durch Schalten von t1 ist man wieder im Ausgangszustand.

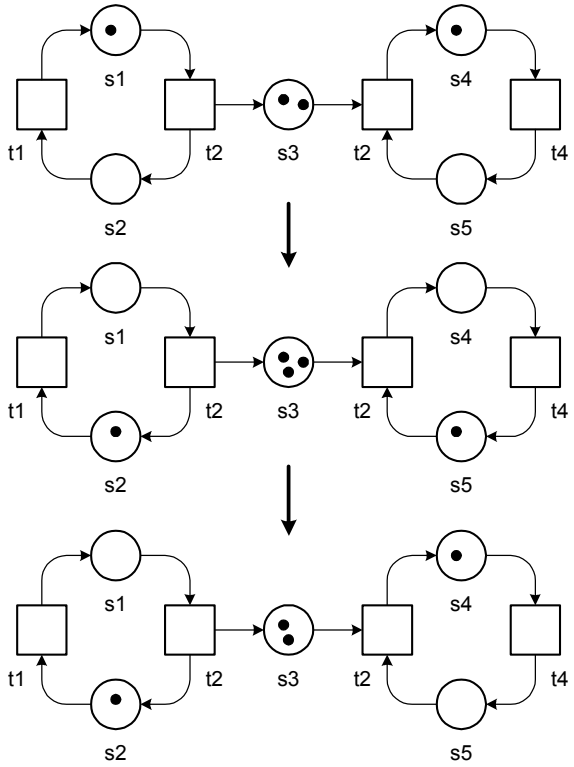


Abbildung 4-3 Schaltvorgänge eines Petrinetzes

Das dargestellte Beispiel zeigt die Modellierung des für verteilte Systeme oft typischen Erzeuger-Verbraucher-Problems. Die Stelle s3 dient dabei als zentraler Speicher, in den der Erzeuger (Stellen s1, s2 sowie Transitionen t1, t2) Objekte ablegt und aus dem der Verbraucher (s4, s5, t2, t4) Objekte entnimmt. Durch die Schaltregel wird zum Beispiel gewährleistet, dass der Speicher mindestens ein Objekt enthalten muss, ehe der Verbraucher eine Entnahme vornehmen kann.

4.2.2 Arten von Petrinetzen

Legt man den Datentyp der Markierungen fest, so erhält man spezielle Versionen der allgemeinen Petrinetze.

4.2.2.1 Bedingungs-/Ereignisnetz

In dieser Variante (*B/E-Netze)* haben die Markierungen den Typ *boolean*. Hier kann jede Stelle nur entweder eine oder keine Markierung enthalten. Damit verändert sich auch die Schaltregel, da nun eine Transition nur schalten kann, wenn der gesamte Vorbereich genau eine Markierung besitzt und weiterhin der gesamte Nachbereich der Transition leer ist. Man interpretiert in diesem Zusammenhang die Transitionen als Ereignisse und die Stellen als Bedingungen. Ein Ereignis tritt in diesem Modell nur dann ein, wenn jede der Vorbedingungen sowie keine der Nachbedingungen erfüllt ist.

4.2.2.2 Stellen-/Transitions-Netze

Hier sind die Markierungen vom Typ nat (natürliche Zahlen), das heißt eine Stelle kann im Gegensatz zu B/E-Netzen mehrere Markierungen enthalten. Darüber hinaus kann für jede Stelle eine Kapazität und für jede Kante eine Gewichtung angegeben werden. Schaltet eine Transition, so gibt das Gewicht der Kanten zur Eingabe- und Ausgabestelle an, wie viele Markierungen entnommen bzw. hinzugefügt werden. Dementsprechend kann eine Transition in dieser Variante nur schalten, wenn jede der Eingabestellen genügend Marken besitzt und die Kapazität aller Ausgabestellen nicht überschritten wird. Sind genügend Marken vorhanden aber reicht die Kapazität der Ausgabestellen nicht aus, erhält man auch hier eine Kontaktsituation.

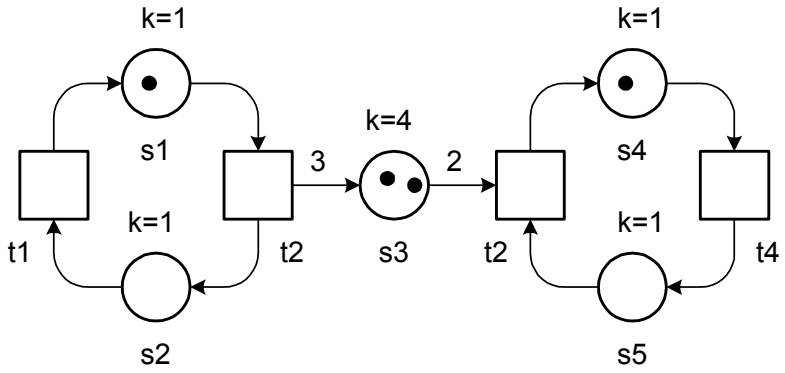


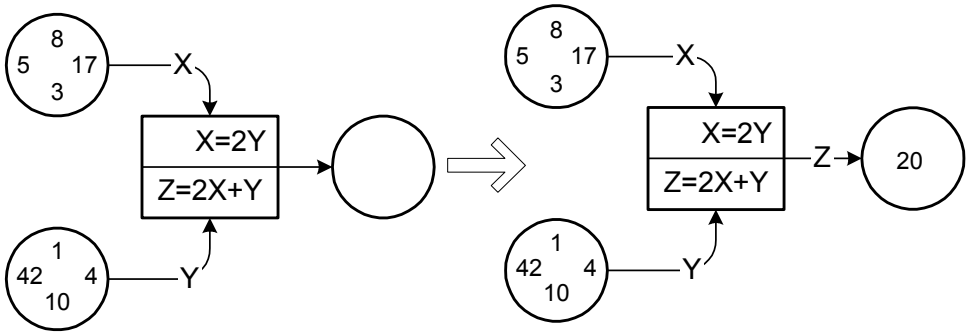
Abbildung 4-4 Stellen-/Transitions-Netz

Abbildung 4-4 zeigt die Modellierung eines Erzeuger-/Verbraucher-Systems als S/T-Netz. In diesem Fall hat der gemeinsame Speicher eine Speicherkapazität für vier Objekte. Der Erzeuger legt dabei auf einmal drei Objekte ab, während der Verbraucher jeweils zwei Objekte entnimmt. Befindet sich nur ein oder gar kein Objekt im Speicher, so muss der Verbraucher entsprechend warten, das heißt t2 kann nicht schalten.

4.2.2.3 Prädikat/Transitions-Netze

Während B/E- bzw. S/T-Netze keine Unterscheidung bzgl. der Markierungen vornehmen, können diese bei der Variante der Prädikat-/Transitions-Netze individualisiert werden. Dabei werden Marken durch einen Wert, etwa eine Zahl, repräsentiert. Darauf aufbauend hat eine Transition nun eine Schaltbedingung (*firing condition*) sowie eine Schaltwirkung (*firing result),* die als Formel aufbauend auf Variablen angegeben

werden. Die Zuweisung der Markierungen zu diesen Variablen erfolgt über die Beschriftung der Kanten des Graphen.



Quelle: [Balzert96a]

Abbildung 4-5 Prädikat-/Transitions-Netz

Abbildung 4-5 zeigt einen Schaltvorgang eines solchen Netzes. Durch die Schaltbedingung X=2Y kann die Transition mit den vorhandenen Markierungen 8 auf der Kante X und 4 auf der Kante Y schalten. Die Folgemarkierung wird dann durch die Schaltbedingung Z=2X+Y auf 20 gesetzt.

Durch die höhere Mächtigkeit dieser Netze lassen sich auch komplexere Vorgänge zum Teil übersichtlicher darstellen als mit den übrigen Varianten. Allerdings wird auch die Handhabung und insbesondere die Analyse schwieriger und aufwändiger. Bezogen auf die Anforderungen dieser Arbeit erscheint diese Variante deshalb nicht als geeignet, da Zustandsänderungen in Telefonanlagen in der Regel durch sehr einfache Bedingungen und Regeln erfolgen. Die höhere Modellierungsmächtigkeit der Prädikat-/Transitions-Netze wird deshalb kaum zum Tragen kommen.

4.2.3 Analyse von Petrinetzen

Neben der Modellierung dynamischer, verteilter Systeme gibt es eine Reihe von Gesichtspunkten, nach denen ein bestehendes Petrinetz analysiert werden kann. Wichtiger Aspekt ist etwa, ob ein Petrinetz *terminiert*, das heißt, ob nur endlich viele Transitionen schalten können oder ob ein unendlich andauerndes Schalten möglich ist. Ein weiteres Kriterium ist die *Lebendigkeit* von Transitionen und Markierungen. Eine Markierung wird als nicht lebendig, also tot bezeichnet, wenn keine der Transitionen unter dieser Markierung schalten kann. Analog dazu ist eine Transition tot, wenn ausgehend von einer Startmarkierung keine Folgemarkierung existiert, unter der sie schalten könnten. Im Gegensatz dazu ist eine Transition lebendig, wenn eine Markierung innerhalb des Netzes erreicht werden kann, in der diese Transition schaltet. Ein gesamtes Netz bezeichnet man dann analog als lebendig, wenn alle Transitionen lebendig sind.

Weitere für die Analyse interessante Situationen sind *Konflikte* sowie *Kontakte*. Zwei Transitionen stehen dabei im Konflikt zueinander, wenn sie beide aktiviert sind, aber nicht genügend Markierungen im Vorbereich vorhanden sind, um beide schalten zu können. Es entsteht dadurch ein Nichtdeterminismus, da durch Schalten einer Transition das Schalten der anderen verhindert wird. Ähnliches gilt auch für eine Kontaktsituation, hier entsteht allerdings die Konkurrenz durch den Nachbereich. Kann dieser nicht genügend Markierungen für das Schalten beider Transitionen aufnehmen, stehen diese in Kontakt zueinander und es kann nur eine oder sogar keine von beiden schalten.

Ein weiteres Ziel der Analysen ist das Erkennen auftretender Verklemmungen (*deadlock)*. Hier unterscheidet man zwischen partieller und totaler Verklemmung, je nachdem ob unter einer bestimmten Markierung nur ein Teil oder alle Transitionen eines Netzes nicht mehr schalten können. Solche Verklemmungen treten auf, wenn eine Menge von Stellen S nie wieder Markierungen erhält, wenn sie einmal alle Markierungen verloren hat. Dies ist insbesondere der Fall, wenn der Vorbereich von S eine Teilmenge des Nachbe-

reichs von S ist, wenn also •S⊆S•. Abbildung 4-6 zeigt ein Netz mit einer typischen Verklemmung. Die kritische Teilmenge S ist hier {s1,s2}. Beide Stellen haben in diesem Zustand keine Markierung und somit kann der Nachbereich S•, also die Transitionen t1, t2, t3, nicht schalten. Da aber ein Teil dieser Transitionen, nämlich t3 und t2 der Vorbereich von S sind, ist ein Schalten zumindest einer dieser Transitionen notwendig, damit die Stellen aus S wieder eine Markierung erhalten. Dadurch entsteht die klassische Verklemmung. Durch diese formale Definition von Bereichen, in denen Verklemmungen auftreten, ist ein systematisches Prüfen bestehender Petrinetze möglich.

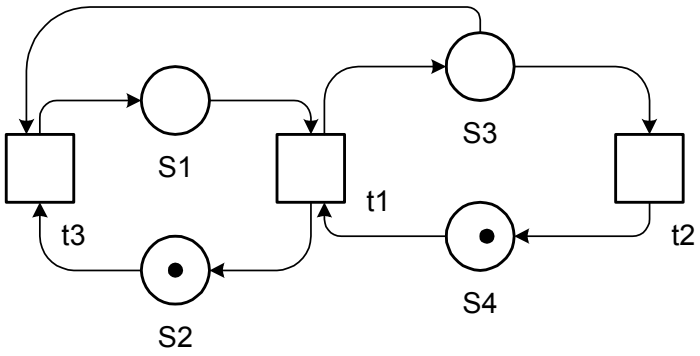


Abbildung 4-6 Deadlock in einem Petrinetz

Abschließend sei noch ein weiteres wichtiges Attribut eines Petrinetzes erwähnt, der Erreichbarkeitsgraph. Er gibt die Veränderung von Markierungen aufgrund verschiedener Schaltvorgänge an. Damit verbunden ist das Erreichbarkeitsproblem, also die Frage, ob es eine Schaltfolge gibt, die eine Markierung M_1 in eine Markierung M_2 überführt.

4.2.4 Bewertung

Petrinetze haben gewisse Ähnlichkeiten zu Zustandsautomaten, da auch hier durch Stellen und Transitionen Zustandsübergänge beschrieben werden. Ein Zustandsautomat befindet sich allerdings zu einem Zeitpunkt nur in einem Zustand, während ein Petrinetz durch die aktuelle Markenbelegung zu einem Zeitpunkt mehrere Zustände einnehmen kann. Darüber hinaus können hier in einem System unabhängig Zustandsübergänge erfolgen. Petrinetze stellen deshalb im Vergleich zu Zustandsautomaten ein eleganteres Modell dar, mit dem auch Konzepte wie Synchronisation oder Nichtdeterminismus gut modelliert werden können. Zusammengefasst weisen Petrinetze folgende Charakteristiken auf:

- Gute Modellierung der verteilten Aspekte
- Kopplung der einzelnen Zustände ist sehr übersichtlich darstellbar
- Benutzereingaben können eher schlecht modelliert werden
- Ereignisse von außen sind allgemein nicht so einfach zu modellieren
- Synchronisation ist naturgemäß eine der Stärken von Petri Netzen

Die für diese Arbeit interessanten Vorgänge basieren im Grunde alle auf den bereits beschriebenen Prozessen eines Telefonsystems, wie etwa das Annehmen eines Anrufs, die Weiterleitung oder eine Konferenz. Durch die Einführung von CTI wirken sich diese Prozesse auf das beteiligte, verteilte IT-System aus, wodurch insbesondere auch die Aspekte Nebenläufigkeit und Nichtdeterminismus eine Rolle spielen können. Daher sind Petrinetze hier in jedem Falle ein sehr interessantes Modell. Eine Betrachtung der Basis-

prozesse legt nahe, dass für die Vorgänge in einem Call Center in erster Linie B/E-Systeme interessant sind. S/T-Netze mit unterschiedlicher Anzahl von Markierungen sowie gewichteten Graphen sind dagegen eher ungeeignet. In der Literatur findet man Beschreibungen und Einführungen zum Thema der Petrinetze etwa in [Balzert96a], [Duden93] sowie [Reisig91] und [Reisig98].

4.3 Objektorientierter Programmentwurf

Um dem Anspruch eines modernen Softwaresystems zu genügen, sind die Konzepte objektorientierter Entwicklung zu berücksichtigen. Einige der wichtigen Begriffe und Konzepte werden daher einleitend eingeführt und betrachtet.

4.3.1 Konzepte

4.3.1.1 Klasse und Objekt

Basiselemente der objektorientierten Entwicklung sind Objekte und Klassen. Ein Objekt ist ein für das zu modellierende System interessanter Gegenstand, der zum Teil ganz konkret auch in der realen Welt existiert (ein Telefon zum Beispiel), aber auch abstrakter Natur sein kann (wie etwa ein Telefonanruf). Ein Objekt hat dabei eine Identität, eigene Daten (Attribute) sowie eine Menge von Methoden, die das Verhalten des Objekts bestimmen (vgl. [Balzert96a]).

Nach [Balzert96a] spezifiziert

„[..] eine Klasse die Gemeinschaft einer Menge von Objekten mit denselben Eigenschaften (Attributen), und demselben Verhalten".

Man erzeugt ein neues Objekt als sog. Instanz einer Klasse, das damit alle in der Klasse definierten Methoden und Attribute besitzt. Es hat während seiner Existenz einen individuellen Zustand, der sich in den Werten seiner Daten widerspiegelt und sich dynamisch verändert. Es gibt dabei allerdings auch Attribute, die nicht einzelnen Instanzen, sondern nur der Klasse zugeordnet sind (auch als Klassenvariablen bezeichnet) und dort statisch definiert werden. Einen Sonderfall stellen abstrakte Klassen dar, von denen keine Objektinstanzen erzeugt werden können. Sie sind im Zusammenhang mit dem Konzept der Vererbung interessant, das nachfolgend noch genauer erläutert wird.

Der Aufruf von Methoden eines Objekts wird häufig als Versenden einer Nachricht an dieses interpretiert. Das System besteht damit zur Laufzeit aus einer Menge von Objekten, die gegenseitig Aktionen durch das Versenden einzelner Nachrichten auslösen.

4.3.1.2 Geheimnisprinzip, Encapsulation

Wichtiges Paradigma der objektorientierten Entwicklung ist, dass die Daten eines Objekts nicht direkt, sondern nur durch Aufruf entsprechender Methoden des Objekts verändert werden. Das Geheimnisprinzip bedeutet demnach, dass Klassen die Attribute anderer Klassen nicht sehen (vgl. [Balzert96a]). Dadurch kann der interne Zustand eines Objekts nicht unmittelbar verändert werden und seine Details bleiben nach außen verborgen. Programmiersprachen bieten vor diesem Hintergrund in der Regel Mechanismen an, die Daten eines Objekts ganz konkret als private Daten kennzeichnen und vor Zugriff durch andere Methoden schützen.

4.3.1.3 Assoziation, Aggregation, Composition

Eines der Ziele der objektorientierten Softwareentwicklung ist, dass durch ein geeignetes Objektmodell ein flexibles, erweiterbares und dem Anwendungsfall sehr genau angepasstes IT-System implementiert werden kann, dessen Basiskomponenten einen hohen Grad an Wiederverwendung besitzen. Wichtige

Strategie ist dabei, komplexere Funktionalität nicht in einer Komponente zu lösen, sondern auf mehrere Objektklassen zu verteilen, die dann zur Laufzeit gemeinsam mit entsprechender Interaktion das gewünschte Verhalten zeigen (vgl. [Gamma95]). Deshalb sind die Abhängigkeiten und Beziehungen einzelner Objektklassen für das Design von wesentlicher Bedeutung. Man unterscheidet dabei unter anderem zwischen den folgenden drei Beziehungen:

Assoziation

Bezeichnet ganz allgemein jegliche semantische Relation zwischen zwei oder mehreren Klassen bzw. Typen (vgl. [Fire95]). Objekte assoziierter Klassen haben in der Regel gegenseitig Kenntnis voneinander und können untereinander Methoden aufrufen.

Aggregation

Eine etwas stärkere Form der allgemeinen Assoziation ist die Aggregation, die eine *is-part-of*-Beziehung beschreibt. Hier sind beide Teile nicht mehr gleichrangig, sondern es existieren über- bzw. untergeordnete Partner (vgl. [Balzert96a]). Klassisches Beispiel ist hier eine Klasse Auto, die aus den Teilen Rad, Motor und Karosserie besteht.

Composition

Eine noch stärkere Einschränkung der Aggregation wird häufig auch als Composition bezeichnet, dann nämlich, wenn nicht nur eine is-part-of-Beziehung zwischen den Objekten besteht, sondern das übergeordnete Objekt auch die alleinige Verantwortung für all seine Teile besitzt, und ein Objekt zu einer Zeit nur einer Komposition angehören kann (vgl. [Rumb99]). Dadurch wird auch die Lebensdauer der einzelnen Unterobjekte bestimmt. In der Literatur wird allerdings häufig nicht zwischen dieser und der allgemeinen Form der Aggregation unterschieden.

4.3.1.4 Vererbung

Nach [Balzert96a] bedeutet Vererbung,

> *„[..] dass eine Klasse 2 über die Eigenschaften und das Verhalten einer Klasse 2 verfügen kann".*

Eine Klasse kann Attribute und Methoden von einer übergeordneten Klasse erben, woraus sich eine Klassenhierarchie ergibt. Sie hat damit Zugriff auf alle Methoden und Attribute, die in der ihr übergeordneten Klasse definiert sind. Je nachdem ob eine Klasse nur eine oder aber mehrere Oberklassen besitzen kann, spricht man von Einfachvererbung oder Mehrfachvererbung (*multiple inheritance*). Abgeleitete Klassen verfeinern und ergänzen somit die Daten und das Verhalten der übergeordneten Klasse. Sie können darüber hinaus jedoch auch das geerbte Verhalten neu definieren, indem sie eine neue Implementierung einer bereits definierten Methode beinhalten. Somit ändert sich für den Aufrufer einer Methode die implementierte Funktionalität, wenn er die Methode der übergeordneten bzw. abgeleiteten Klasse nutzt. Eine spezielle Form, die abstrakten Klassen, implementieren eine oder mehrere Methoden überhaupt nicht mehr selbst, sondern überlassen dies abgeleiteten Klassen. Man kann demnach auch Instanzen von abstrakten Basisklassen erzeugen. Es wird hier lediglich die Signatur der Schnittstelle angegeben und die Implementierung anderen Objekten überlassen.

4.3.1.5 Polymorphismus

In der objektorientierten Programmierung bedeutet das Prinzip des Polymorphismus in der Regel, dass Objekte verschiedener Klassen mit gleicher Schnittstelle zur Laufzeit ausgetauscht werden können (vgl. [Gamma95]). Dieselbe Nachricht kann also an verschiedene Klassen gesendet werden, wobei Empfängerobjekte verschiedener Klassen ganz unterschiedlich darauf reagieren (vgl. [Balzert96a]). Dieses Konzept steht eng im Zusammenhang mit dem Prinzip der Vererbung sowie dem Konzept des „späten Bindens" (*late binding, dynamic binding*). Hier wird der konkrete Typ eines Objekts dynamisch erst zur Laufzeit durch das System und nicht statisch zur Zeit des Übersetzens festgelegt. Damit wird zur Laufzeit eine Verzwei-

gung auf unterschiedlichen Programmcode möglich, die in herkömmlichen Programmiersprachen statisch etwa durch *switch-case*-Anweisungen hergestellt werden müssen.

Durch die Kombination von Vererbung, abstrakten Basisklassen und Polymorphismus lassen sich sehr flexible Systeme erstellen. Es können dadurch nämlich unterschiedliche Implementierungen einer Funktionalität existieren. Der Client, der eine gewisse Funktionalität benötigt, kennt dadurch nur mehr die Schnittstelle, in der Dienste ausreichend spezifiziert werden, und trifft nur sehr wenig Annahmen über den Erbringer dieser Dienstleistungen. Dadurch, dass durch Vererbung und Polymorphismus einzelne Schnittstellen ganz unterschiedlich implementiert werden können und die einzelnen Objekte, die eine Schnittstelle implementieren, auch zur Laufzeit noch ausgetauscht werden können, können Objekte sehr gut voneinander entkoppelt werden. Dies kommt der Flexibilität eines Gesamtsystems zugute.

Für eine allgemeinere Übersicht zu den Grundlagen des objektorientierten Entwurfs auf [Balzert96a] sowie [Rumb99] und [Gamma95] verwiesen.

4.3.2 UML Notation

Zur Modellierung und visuellen Darstellung objektorientierter Softwaresysteme hat sich in letzter Zeit die Verwendung von UML (*Unified Modeling Language*) durchgesetzt und wird deshalb auch für diese Arbeit als Notation herangezogen. Was die gesamte Spezifikation von UML angeht, so sei an dieser Stelle auf die geeignete Literatur wie etwa [Rumb99] oder [OMG00] verwiesen. Die wichtigsten und für die späteren Designschritte elementaren Bestandteile dieser Sprache sollen in diesem Abschnitt jedoch kurz angesprochen werden.

4.3.2.1 Class Diagrams

Abbildung 4-7 zeigt die UML-Notation eines Klassenmodells. Für jede Klasse werden dabei der Name, die Attribute und Methoden angegeben. Um die Übersichtlichkeit in größeren Diagrammen zu erhalten, können die zwei unteren Bereiche auch unterdrückt werden. Für jedes Attribut wird der entsprechende Typ angegeben, für jede Methode Name und Typ der Parameter sowie eventuelle Rückgabewerte. Eine abgeleitete Klasse wird durch den dargestellten Pfeil mit der ihm übergeordneten Klasse verbunden. Ebenso wird eine Aggregation durch eine entsprechende Verbindung dargestellt. Eine Assoziation zwischen zwei Klassen wird durch eine einfache Linie angezeigt. Für Beziehungen wie Assoziation oder Aggregation kann in UML eine Multiplizität (*multiplicity)* angegeben werden, die angibt, wie viele Instanzen einer Klasse in Beziehung zu einer anderen stehen. Im dargestellten Beispiel erben also die Klassen *Employee* und *Customer* von *Person.* Jedem Kunden ist genau eine Adresse zugeordnet, die wiederum aus genau einem Basisteil sowie einer oder mehreren Erweiterungen besteht.

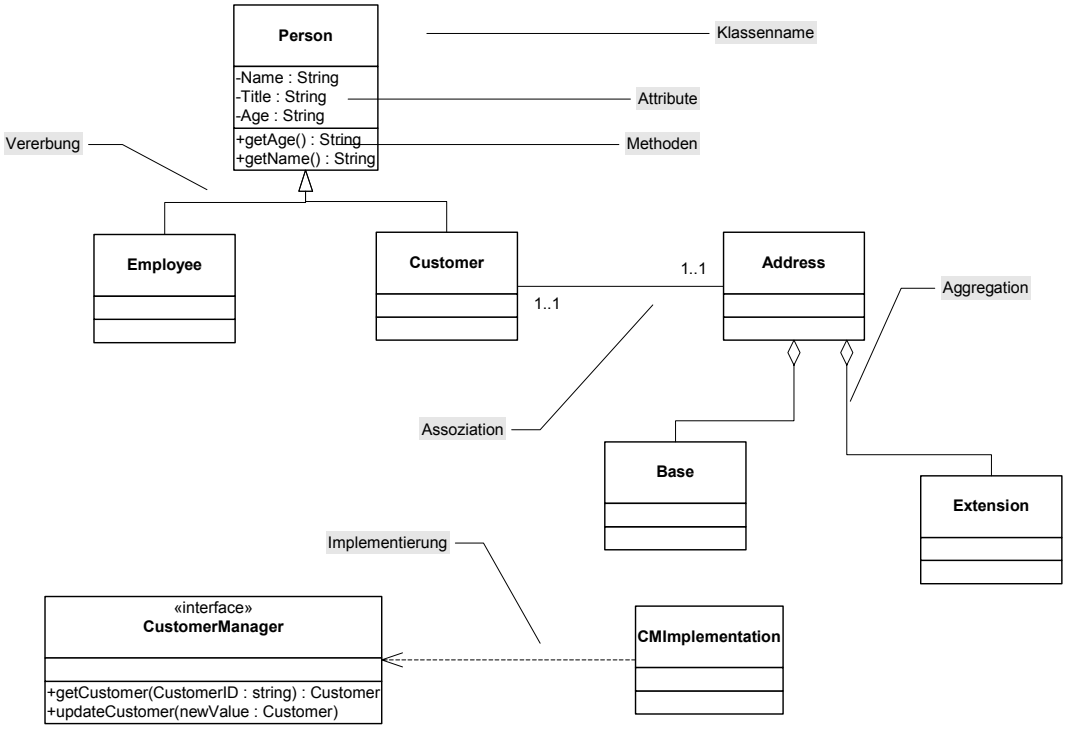


Abbildung 4-7 Klassendiagramme in UML

Es werden dabei in der Regel Klassen dargestellt, konkrete Instanzen nur in Sonderfällen als zeitgebundenes Abbild des Systems.

4.3.2.2 Use Cases

Um den Blick von außen auf die Funktionalität eines Gesamtsystems darzustellen, existiert das Use Case Konzept. Es modelliert die Funktionalität eines Systems, wie sie von unterschiedlichen Benutzern, sogenannten Akteuren (*actors*) wahrgenommen wird. Ein Use Case wird durch eine einfache Ellipse dargestellt, die den entsprechenden Namen beinhaltet. Die Akteure, die mit einem Use Case kommunizieren, sind mit ihm durch eine einfache Linie verbunden. Akteure können dabei reale Benutzer wie etwa ein Kunde oder ein Angestellter aber auch Komponenten wie ein IVR-System oder ein Backoffice-System sein. Um komplexere Fälle zu modellieren, können Use Cases auch unterteilt werden. So beinhaltet im Beispiel die Eröffnung eines Kontos (*open account*) das Sammeln von Kundendaten, eine Kreditüberprüfung sowie den endgültigen Eintrag im kontoführenden System.

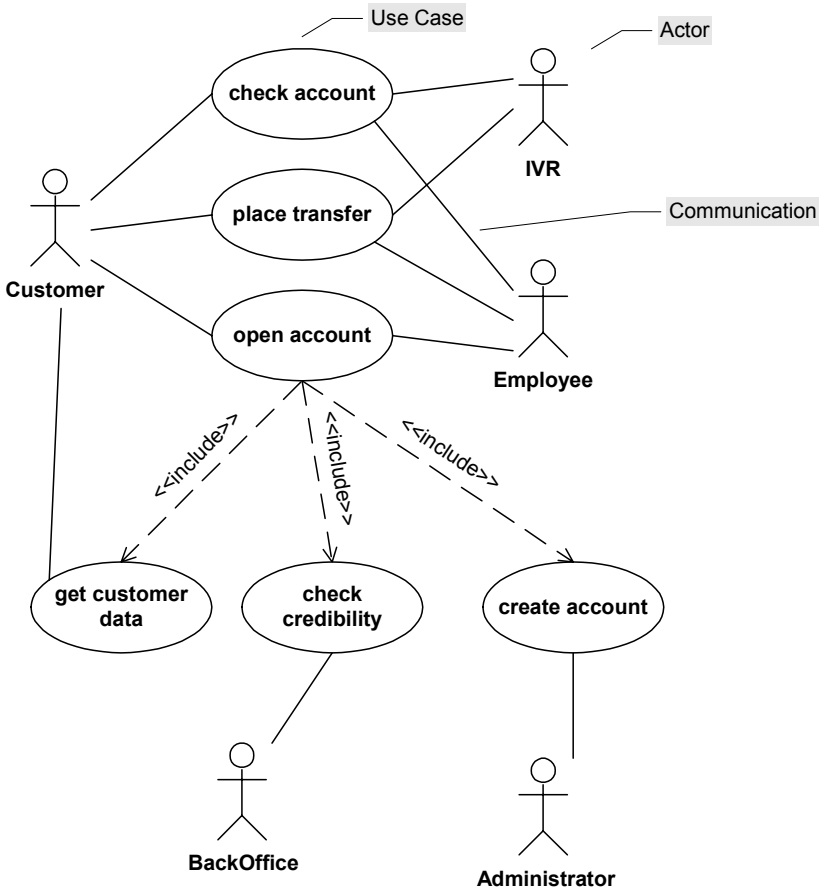


Abbildung 4-8 Use Case Diagramm

4.3.2.3 Sequence Diagram

Während Klassendiagramme und Use Case Diagramme die statische Natur eines Systems beschreiben, dienen Sequenzdiagramme der Modellierung dynamischer Abläufe. Hier kann der zeitliche Ablauf der Kommunikation einzelner Komponenten eines Systems dargestellt werden. Abbildung 4-9 skizziert, wie in einem Call Center einer Bank ein Überweisungsauftrag eines Kunden durch mehrere Objekte des Systems durchgeführt werden kann. Der Auftrag wird dabei an ein Managerobjekt weitergeleitet, das dafür eine neue Instanz der Klasse *Transfer* erzeugt. Durch die unterbrochene Linie wird angezeigt, dass dieses Objekt vorher nicht existiert. Nachdem die relevanten Daten übergeben wurden, erhält das Objekt den Auftrag, die Überweisung durchzuführen. Es wendet sich dazu wiederum an eine bereits existierende Instanz einer BackOffice-Klasse, die für Überweisungen zuständig ist. Nach positiver Quittung beendet sich das Transfer-Objekt und der Manager kann eine entsprechende Meldung für den Benutzer erzeugen. Das Kreuz zeigt an, dass das Objekt der Klasse *Transfer* explizit zerstört wird.

Darüber hinaus bietet UML noch eine Reihe von weiteren Notationen für die Modellierung der statischen sowie dynamischen Aspekte eines IT-Systems. Dazu gehören unter anderem eine eigene Notation für Zustandsautomaten (*state machine*), *collaboration diagrams, components, packages, subsystems.*

Da jedoch nicht alle dieser Modelle für die konkret vorliegenden Anforderungen notwendig bzw. geeignet sind, sei für eine weitergehende Einführung wie eingangs bereits erwähnt auf die geeignete Fachliteratur verwiesen.

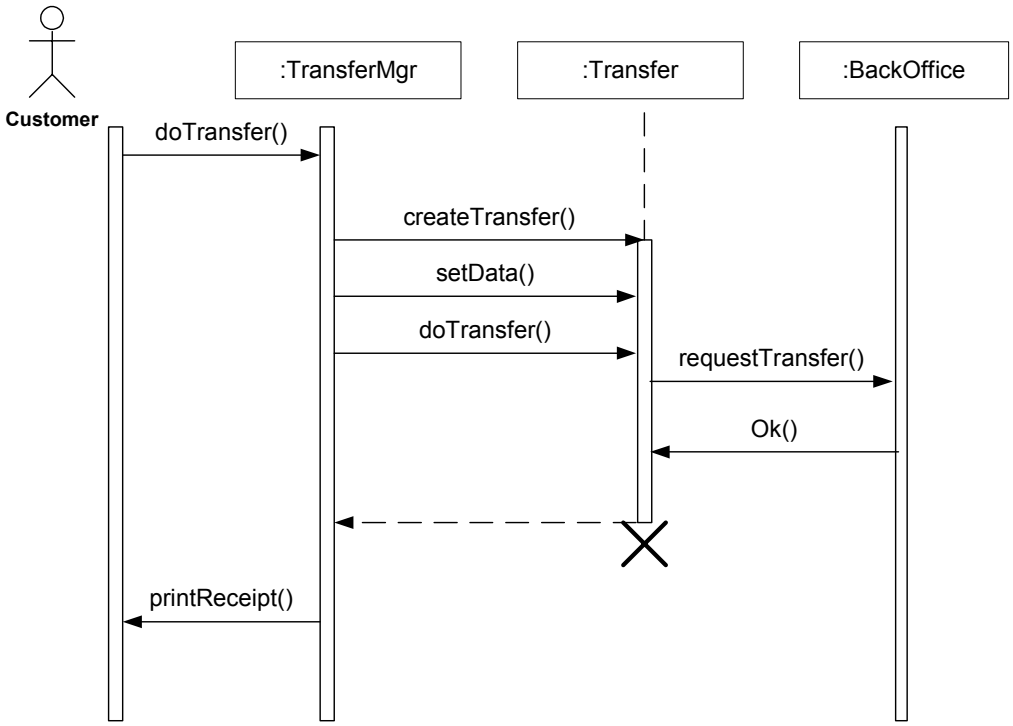


Abbildung 4-9 UML Sequence Diagramm

Mit den beschriebenen Basiskonzepten für Zustandsautomaten, Petrinetze sowie den objektorientierten Entwurf stehen nun die Modelle zur Verfügung, die für die Spezifikation wie auch den Entwurf eines verteilten Systems herangezogen werden können.

4.4 XML

XML (Extensible Markup Language) ist als Untermenge von SGML (Standard Generalized Markup Language) eine Metasprache zur Definition von Dokumentbeschreibungs- oder auch Auszeichnungssprachen. Verabschiedet wurde die Version 1.0 durch das W3C, und darin durch die Autoren Tim Bray, Jean Paoli, C. M. Sperberg sowie Eve Maler. Wichtiges Ziel dabei war, eine einfache und im Internet möglichst überall verwendbare Technologie zu etablieren, die einige der Schwächen von HTML beseitigt. Diese liegen vor allem in der mangelnden Strukturierungsmöglichkeit von Daten und in der fehlenden Trennung von Information und Darstellung. Im Gegensatz zu HTML ist es in XML erlaubt, eigene Tags und Attribute zu definieren und somit möglich, Struktur in eine Textdatei zu bringen. Sie bestehen damit aus einzelnen Elementen, wobei jedes Element immer durch ein open- bzw. close-Tag begrenzt wird. Wie Abbildung 4-10 zeigt, können dabei auch geschachtelte Strukturen abgebildet werden, was insbesondere der Abbildung von relationalen oder objektorientierten Modellen sehr zugute kommt. Eine XML-Datei wird immer einem Dokumenttyp zugeordnet, der in einer separaten Definition DTD (Document Type Definition) spezifiziert wird. Darin wird festgelegt, welche Tags erlaubt sind und wie die Hierarchisierung der Struktur erfolgt.

Was die Struktur eines XML-Dokuments angeht, so unterscheidet man hier zwischen *well-formed* und *valid*. Erstes Kriterium erfüllt ein Dokument, wenn es die allgemeinen Regeln der Spezifikation erfüllt. Dazu gehört zum Beispiel die XML-Deklaration in der ersten Zeile sowie ein Textkörper mit mindestens einem Tag-Element, der die allgemeinen Strukturregeln erfüllt. So muss ein geöffnetes Element auch immer wieder geschlossen werden und es sind keine Überlappungen von Elementen zulässig. Es wird jedoch nicht festgelegt, welche Elemente enthalten sein müssen und welche genaue Struktur eingehalten werden muss. Dies legt die DTD fest und eine XML-Datei, die alle dort beschriebenen Eigenschaften besitzt, heißt dann *valid*.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Autoren.xsl"?>
<Personen>
 <Autor>
     <Vorname>Frank</Vorname>
     <Nachname>Mueller</Nachname>
 </Autor>
 <Autor>
     <Vorname>Stefan</Vorname>
     <Nachname>Maier</Nachname>
 </Autor>
</Personen>
```

Abbildung 4-10 XML-Beispiel

Neben der reinen Spezifikation der Sprache gehören eine Reihe von Technologien zum Umfeld von XML. Hier ist einmal XSL (Extensible Stylesheet Language) zu nennen. Damit kann festgelegt werden, wie die graphische Ausgabe einer XML-Datei zu erfolgen hat. So ist dadurch zum Beispiel die Umwandlung in HTML und damit die Anzeige in einem Web-Browser möglich. Damit wird auch die strikte Trennung von Information und Präsentation erreicht. Daneben existiert noch XSLT (Extensible Stylesheet Language Transformations), womit die Überführung von einem XML-Format in ein anderes festgelegt werden kann.

Für die Verwendung von XML in der Softwareentwicklung existiert hier das DOM (Document Object Model), das die Darstellung einer XML-Datei als baumartige Objektstruktur ermöglicht und umgekehrt. Im Falle von Java existieren inzwischen auch Spezialbibliotheken wie etwa JDOM, die die Überführung solcher objektorientierter Strukturen nach XML wesentlich erleichtern.

Die derzeitigen Einsatzgebiete von XML reichen von der Dokumentverwaltung über die Verwendung in Konfigurationsdateien (z.B. Deployment-Deskriptor von EJB) bis hin zur allgemeinen Kommunikation in lose gekoppelten Systemen. XML ist darüber hinaus auch als echte Middleware-Variante für verteilte Systeme im Gespräch, hier hat sich der echte Mehrwert jedoch noch nicht gezeigt. In naher Zukunft wird wohl eher XML als Ersatz von HTML zu erwarten sein und hier seinen Vorteil durch die klare Trennung von Information und Darstellung ausspielen. Dies kommt dann auch dem ursprünglichen Ziel der Autoren sehr nahe, wie man nachfolgendem Zitat entnehmen kann:

"XML will be the ASCII of the Web – basic, essential, unexciting" (Tim Bray)

5 NOTWENDIGE TECHNOLOGIEN

Nachdem die eher allgemeinen Basiskonzepte betrachtet wurden, ist vor der konkreten Spezifikation eine Auswahl und Bewertung der einzusetzenden Technologien angebracht. Die Abhängigkeit bzw. Unabhängigkeit von konkreten Technologien ist ein wichtiges Qualitätsmerkmal für Softwaresysteme und Softwarearchitekturen, da hier die Einsatzmöglichkeiten und die Wiederverwendbarkeit entscheidend beeinflusst werden. Da gerade im Bereich CTI und Call Center die verfügbaren Technologien und Produkte einen äußerst starken Einfluss haben, ist hier eine Abstraktion notwendig. Es stellt sich also die Frage, auf Basis welcher verfügbaren Technologien der gewählte Ansatz zu realisieren ist und welche Abhängigkeiten sich daraus ergeben. Die Funktionalität des Frameworks stützt sich neben den Werkzeugen der objektorientierten Softwareentwicklung hauptsächlich auf zwei Basistechnologien, eine CTI Schnittstelle sowie eine Middleware. Erstere wird benötigt, um grundlegende Telefonfunktionen innerhalb einer Softwarelösung nutzen zu können. Eine Middleware bietet sich an, da das Framework als verteiltes System ausgelegt werden muss, in dem mehrere Komponenten über ein Netzwerk analog den telefonierenden Teilnehmern miteinander interagieren müssen.

Es ist daher sinnvoll, für die notwendigen Basissysteme CTI und Middleware klare Vorgaben bzgl. der benötigten Funktionalität zu haben und verfügbarere Technologien gegen diesen Anforderungskatalog zu prüfen. Dazu werden für beide Bereiche die notwendigen Funktionen in drei Kategorien eingeteilt, je nachdem, ob sie unbedingt notwendig (A), hilfreich (B) oder nicht unbedingt notwendig (C) für das Framework sind. Danach werden die derzeit in Frage kommenden Technologien kurz eingeführt und anhand dieser Anforderungen bewertet, so dass abschließend eine Auswahl an Basistechnologien für die weitere Konzeption möglich ist.

5.1 CTI Schnittstelle

Für die Anbindung an das Telefonsystem verwendet das Framework eine CTI-Schnittstelle, die es erlaubt, die Telefonanlage aus einem Computerprogramm heraus ansprechen zu können. Die Implementierung einer solchen Schnittstelle, häufig auch als CTI-Middleware bezeichnet, ist dann für die Umsetzung einer grundlegenden Anrufkontrolle verantwortlich. Sie kommuniziert dazu in der Regel direkt mit den Hardwarekomponenten der Telefonanlage über meist proprietäre Protokolle.

5.1.1 Anforderungen

Erster Bereich der benötigten Funktionen sind Dienste für die allgemeine Kontrolle des Anrufs. Dazu sind folgende Funktionen notwendig:

- MAKE CALL
  Leitet einen neuen Anruf ein.
- ANSWER CALL
  Nimmt einen eingehenden Anruf entgegen.
- HOLD
  Setzt einen Anruf auf „Halten“.
- RECONNECT
  Stellt eine Verbindung zu einem gehaltenen Anruf wieder her.
- TRANSFER CALL
  Leitet einen Anruf weiter.
- CONFERENCE
  Stellt eine Konferenz her.

Alle diese Funktionen sind der Kategorie (A) zuzuordnen, da sie unbedingt notwendig für die Funktionalität sind. Weiterhin muss das verwendete CTI-System dem Framework folgende Ereignisse übermitteln können:

- ALERT
  Ein neuer Anruf liegt an.
- CALLING
  Das System baut gerade eine neue Verbindung auf.
- ANSWERED
  Eine Gegenstelle hat den Anruf entgegengenommen.
- ACCEPT
  Eine Verbindung wurde hergestellt.
- HANGUP
  Eine Verbindung wurde getrennt.
- HOLD
  Eine Verbindung wird gehalten.
- RECONNECT
  Eine gehaltene Verbindung wurde wiederhergestellt.

Auch hier können alle Kriterien mit (A) bewertet werden.

Abschließendes Kriterium ist dann die Verknüpfung bestimmter Daten mit einem Anruf, die während der gesamten Lebensdauer dieses Anrufs aufrecht erhalten bleibt. Eine Struktur dieser Daten ist in diesem Fall nicht erforderlich, da dies ja eine wesentliche Funktionalität des Frameworks ist. Auch dieses Kriterium ist für CTI-Anwendungen notwendig und deshalb der Kategorie (A) zuzuordnen.

Als potentielle Basissysteme werden nun folgende Technologien bzw. Produkte einer genaueren Betrachtung unterzogen:

- JTAPI, eine Java basierte Programmierschnittstelle für Telefonanwendungen
- TSAPI, der derzeit noch verbreitetste CTI-Standard
- TAPI, die in der Microsoft-Welt dominierende Schnittstelle zu Telekommunikationsgeräten

5.1.2 JTAPI

JTAPI (*Java Telephony API)* ist eine von den Firmen Sun, Lucent, Nortel, Novell, Intel und IBM entwickelte Programmierschnittstelle zur Entwicklung von CT-Anwendungen unter Java. Um den Einsatz dieser Schnittstelle bewerten zu können bzw. als Grundlage für die Entwicklung eines Prototypen, ist das allgemeine Modell dieser Schnittstelle etwas näher zu untersuchen.

Ziel von JTAPI ist es, ein einheitliches, erweiterbares und objektorientiertes Modell zu schaffen, das für einen großen Bereich von Aufgaben im Umfeld der Computertelefonie verwendet werden kann. Dieser Bereich reicht von individuellen Desktop-Anwendungen bis hin zu Systemen verteilter Call Center. Zu diesem Zweck definiert JTAPI eine Menge von Objekten, die der Kontrolle eines Anrufes inklusive aller beteiligten Systeme dienen. Diese Objekte sind in einer Reihe von Java *packages* enthalten, die in Kombination die jeweils gewünschte Funktionalität bieten. JTAPI unterstützt die Ansätze first-party sowie third-party. Im ersten Fall wird das System in einer Desktop-Konfiguration eingesetzt, in der die Workstation neben der JTAPI-Anwendung auch alle Telefonressourcen z.B. über entsprechende Steckkarten verwaltet. Im zweiten Fall greift das lokale Runtime-System von JTAPI auf einen zentralen CTI-Server zu, der wiederum Zugriff auf die jeweilige Telefonhardware wie etwa eine PBX hat. Für die Entwicklung des Frameworks, das ja in erster Linie für den Einsatz in einem Call Center bzw. in ähnlichen Umgebungen konzipiert wird, ist der zweite Fall, also der Einsatz von JTAPI, in einer Netzwerkkonfiguration primär interes-

sant. Die hohe Plattformunabhängigkeit, die Java bietet, setzt sich auch bei der Verwendung von JTAPI durch, was bedeutet, dass JTAPI prinzipiell auf allen Plattformen verwendet werden kann, auf denen eine Laufzeitumgebung für Java, sowie eine Anbindung an ein vorhandenes Telefonsystem existieren.

5.1.2.1 Modell

Das Modell, auf welchem die Spezifikation von JTAPI aufbaut, besteht im Wesentlichen aus den 6 Objekten, die in Abbildung 5-1 dargestellt sind. Der Telefonanruf an sich wird dabei durch ein eigenes Objekt *call* modelliert. Erzeugt werden kann so ein Objekt über einen *provider*, eine Softwarekomponente, die den Zugang zu einem vorhandenen Telefonsystem realisiert. Ein derartiges System wäre eine zentrale Telefonanlage oder eine lokale Steckkarte eines PCs mit Telefon- oder Faxfunktionalität. Ein Anruf wiederum besitzt eine Menge von Verbindungen (*connection),* die allerdings auch leer sein kann. Ein typischer Anruf zwischen zwei Parteien besitzt zwei Verbindungen, eine Telefonkonferenz dagegen drei oder sogar mehr. Diese Verbindungen haben unterschiedliche Zustände (IDLE, INPROGRESS, DISCONNECTED etc.) und beschreiben die Beziehung zwischen einem Anruf und einer Adresse (*address)*, die in der Regel aus einer Telefonnummer besteht. Dies ist jedoch nicht zwingend notwendig und so kann eine Adresse in einer IP-basierten Umgebung auch aus einer IP-Nummer bestehen. Die Objekte *call, connection* sowie *address* beschreiben einen Telefonanruf aus logischer Sicht, nämlich als Menge von zwei oder mehreren Adressen, deren Zustand innerhalb dieses Anrufs durch die einzelnen Verbindungen beschrieben wird. Die physikalische Sicht eines Anrufs wird durch die Objekte *terminal* sowie *terminal connection* modelliert. Das Terminalobjekt repräsentiert dabei das jeweilige Endgerät, also ein Telefon oder etwa eine entsprechend ausgestattete Workstation. Die Beziehung zwischen einer logischen Verbindung und diesem Gerät wird über das Objekt *terminal connection* beschrieben.

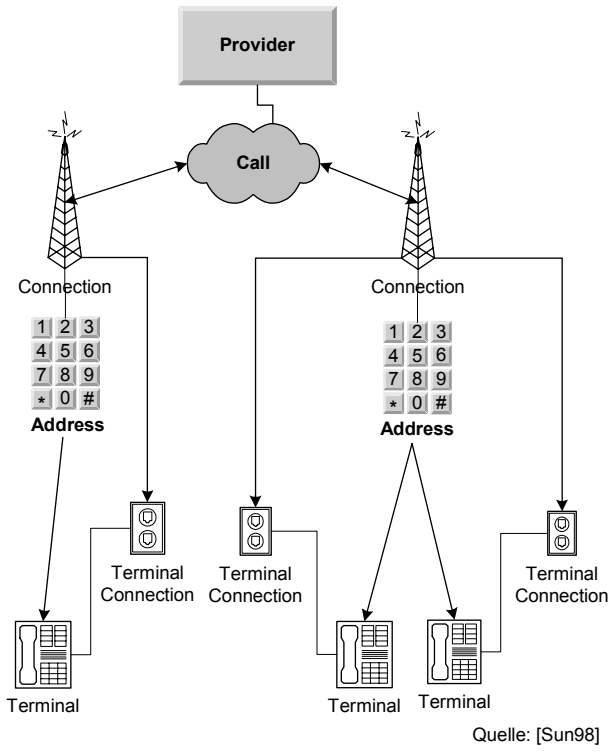


Abbildung 5-1 Call Modell von JTAPI

5.1.2.2 Dienste

Die Funktionalität von JTAPI wird in mehreren Java *packages* implementiert, wobei jedes einen bestimmten Teilaspekt der Computertelefonie aufgreift. Verschiedene Implementierungen der JTAPI-Server wäh-

len anhand der verfügbaren Hardware und Plattform sowie anhand des gewünschten Einsatzgebietes diejenigen dieser Pakete aus, die unterstützt werden. Anwendungen wiederum können von einem Server die Menge der *packages* erfahren, die dieser momentan implementiert hat.

Nachfolgend sind einige der wichtigen Dienste sowie die Pakete, in denen sie definiert sind, aufgeführt.

Dienst	Paket	Beschreibung
Call.connect	Core	Initiiert einen neuen Anruf.
TerminalConnection.answer	Core	Nimmt einen Anruf auf dieser Verbindung entgegen.
CallControlCall.transfer	CallControl	Leitet einen Anruf an eine angegebene Adresse weiter.
CallControl.TerminalConnection.hold	CallControl	Anruf halten.
CallControlCall.conference	CallControl	Fügt diesen und einen weiteren Anruf zu einer Konferenz zusammen.

Tabelle 5-1 Dienste von JTAPI

5.1.2.3 Zustandsautomaten

Alle Elemente des JTAPI-Modells haben eine Menge von definierten Zuständen sowie eine Reihe von Zustandsübergängen. Zwei der für die Betrachtung wesentlichen Zustandsautomaten sind in Abbildung 5-2 dargestellt. Daraus ist zu entnehmen, dass vor allem das Objekt *Connection* in der erweiterten Form des Pakets CallControl alle grundlegenden Zustände umfasst.

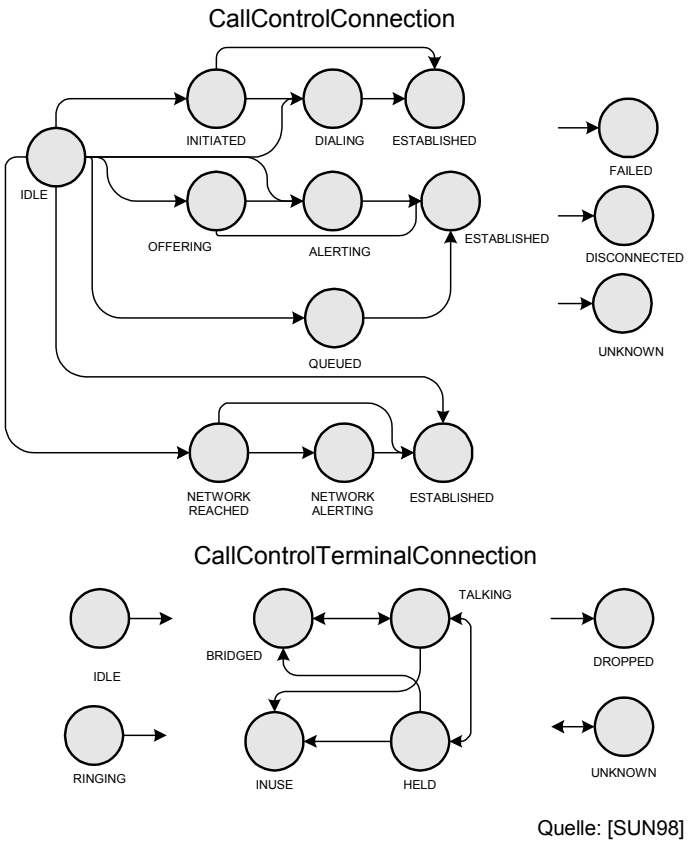


Abbildung 5-2 JTAPI Zustandsübergänge

5.1.2.4 Ereignisse

Um asynchron auftretende Ereignisse, etwa das Eintreffen eines neuen Anrufs einer Clientanwendung mitzuteilen, nutzt JTAPI den *observer/observable*-Mechanismus von Java. Für die Objekte *provider, call, terminal* und *address* werden entsprechende Observer-Objekte definiert, deren Schnittstelle die Clientanwendung entsprechend zu implementieren hat. So ist für das Objekt *CallObserver* eine Methode

```
public abstract void callChangedEvent(CallEv eventList[])
```

definiert, die aufgerufen wird, sobald sich der Zustand eines Anrufs verändert. JTAPI meldet der Client-Applikation jede Veränderung in einem der Objekte des Modells über Ereignisse, die dem jeweiligen Observer-Objekt übermittelt werden. Da mehrere solcher Ereignisse für ein Objekt gleichzeitig auftreten können, werden sie in *arrays* an die Observer-Objekte geschickt.

Das Basisobjekt aller Ereignisse wird über die Schnittstelle *Ev* beschrieben. Davon abgeleitet existiert für jedes Objekt, für das ein Observer-Objekt definiert ist (also *Provider, Call, Address* und *Terminal*), ein Ereignisobjekt, nämlich *ProvEv, CallEv, AddrEv,* und *TermEv*. Die Ereignis-Objekte für *Connection* bzw. *TerminalConnection*, also *ConnEv* und *TermConnEv,* werden von *CallEv* abgeleitet, da diese Objekte im Call-Modell Ereignisse über den Observer des Call-Objekts propagieren. Jedes Ereignis beinhaltet einen Identifikator sowie Information über die Ursache, die das Ereignis auslöste (*cause code*). Beispiele dafür sind CAUSE_NORMAL, CAUSE_CALL_CANCELLED oder CAUSE_NEW_CALL.

Sogenannte Meta-Codes dienen dazu, die verschiedenen Ereignisse nochmals zu gruppieren, und somit eine Einteilung auf einer höheren Abstraktionsebene zu erhalten. Dies soll der Client-Applikation helfen, die unter Umständen sehr zahlreichen Events in den verschiedenen Objekten zu verwalten. Ein Beispiel

eines derartigen Codes ist META_CALL_PROGRESS. Alle Ereignisse die auftreten, wenn ein Anruf angenommen wird und eine Verbindung somit zustande kommt, tragen diesen Code.

In der neuesten Version 1.3 der Spezifikation wird dieser Mechanismus in den Basispaketen bereits teilweise durch das Listener-Konzept abgelöst, das sich auch in Java seit der Version 1.1 für die Ereignisverarbeitung durchgesetzt hat. In der derzeit geplanten Version 2.0 soll dann der Observermechanismus vollständig durch dieses Konzept abgelöst werden. Auch in diesem Ansatz muss die Anwendung die Methoden der Listener-Interfaces implementieren und dies dann entsprechend registrieren. Wesentlicher Unterschied des neuen Ansatzes ist, dass nun für jedes Ereignis eine bestimmte Methode des Listeners aufgerufen wird und so Ereignisse immer getrennt voneinander übermittelt werden, auch wenn mehrere Veränderungen der Telefonanlage auf einmal gemeldet werden. Im bisherigen Modell wurden hier die Ereignisse in dem angegebenen Array (eventList[]) zusammengefasst.

5.1.2.5 Anruf/Daten

Um Daten mit einem Anruf zu verknüpfen, enthält die Klasse *CallCenter.Call* die Methoden `setApplicationData()` sowie `getApplicationData()`. Damit kann ein beliebiges Java-Objekt an einen Anruf gebunden werden und steht nach einer Weiterleitung der neuen Applikation wieder zur Verfügung. Die Tatsache, dass ein Anruf weitere Daten enthält, wird über das Ereignis *CallCentCallAppDataEv* mitgeteilt. Damit steht dem Entwickler ein äußerst flexibler und mächtiger Mechanismus zur Verfügung, wobei natürlich eine geeignete Implementierung durch einen Provider vorausgesetzt werden muss.

5.1.2.6 Bewertung

Die für das zu entwerfende System relevante Funktionalität wird, wie in Tabelle 5-2 dargestellt, von JTAPI vollständig abgedeckt. Benötigt werden dazu die Module *telephony, callcontrol* sowie *callcenter*.

	Dienst	Unterstützt durch
CallControl	MAKE CALL	Call.connect
	ANSWER CALL	TerminalConnection.answer
	HOLD	CallControl.TerminalConnection.hold
	RECONNECT	CallControl.TerminalConnection.unhold
	TRANSFER CALL	CallControlCall.transfer
	CONFERENCE	CallControlCall.conference
Ereignisse	ALERT	CallCtlConnAlertingEv
	CALLING	CallCtlConnDialingEv
	ANSWERED	CallCtlConnEstablishedEv
	ACCEPT	CallCtlConnEstablishedEv
	HANGUP	TerminalConnection.Dropped
	HOLD	CallCtlTermConnHeldEv
	RECONNECT	CallCtlTermConnTalkingEv
Anruf / Daten		CallCenterCall.getApplicationData

Tabelle 5-2 Funktionalität von JTAPI

Zusammen mit dem vergleichsweise modernen und flexiblen Anrufmodell und der Nutzung vieler Sprachmittel von Java bietet JTAPI derzeit einen der modernsten Ansätze im Bereich der CTI-Technologie. Dazu zählt auch die sehr enge Integration in die gesamte Java-Plattform und die dort weiterhin angebotenen Technologien.

Zu beachten ist bei dieser Übersicht jedoch, dass es sich hier um die Funktionalität des spezifizierten JTAPI-Standards handelt. Konkrete Implementierungen weichen hier in der Regel ab, da sie nicht alle Dienste implementieren. Gerade die Funktionalität, applikationsspezifische Daten mit einem Anruf dauerhaft zu verknüpfen, wird nicht von allen derzeit verfügbaren Implementierungen unterstützt, da dies in der Regel mit dem zusätzlichen Aufwand der zuverlässigen Datenübertragung außerhalb der reinen Tele-

fonkommunikation verbunden ist. Da die JTAPI-Spezifikation noch relativ neu ist, kann noch nicht damit gerechnet werden, dass sehr viele Implementierungen am Markt sind, die vollständig und vor allem bereits entsprechend ausgereift sind.

5.1.3 TAPI

Die Schnittstelle TAPI (*Telephony Application Programming Interface)* ist eine proprietäre Entwicklung der Firma Microsoft, hat sich allerdings neben TSAPI als einer der bedeutendsten Industriestandards vor allem im PC-Bereich stark verbreitet. Sie liegt derzeit in der Version 2.2 für C-Anwendungen sowie in der Version 3.0, allerdings noch im Beta-Stadium, für COM-Anwendungen vor. Ein ganz wesentlicher Schwerpunkt von TAPI 3.0 ist *voice over IP*, so dass dieses Produkt in Zukunft unter Umständen gegenüber den hier dargestellten Alternativen einen Vorteil haben könnte. Außerdem ist TAPI 3.0 als Version für Windows 2000 auch als Variante für den Einsatz von COM+ zu wählen. Im Vergleich zu den hier betrachteten CTI-Basissystemen ist jedoch die Version 2.2 in erster Linie von Bedeutung, da eine Festlegung auf COM zu diesem Zeitpunkt nicht sinnvoll ist.

5.1.3.1 Modell

Im Vergleich zu JTAPI oder TSAPI basiert TAPI auf einem relativ einfachen Telefonie-Modell, das sich auf zwei abstrakte Komponenten stützt, *line device* und *phone device*. Entsprechend diesen Komponenten werden die verfügbaren Dienste auch in zwei Gruppen eingeteilt. Erste Komponente ist die abstrakte Repräsentation von Geräten, über die Informationen mit einem realen Telefonnetz ausgetauscht werden können. Beispiele dafür sind ein Modem, ein Fax-Modem oder eine ISDN-Steckkarte. Ein *phone device* repräsentiert dagegen das herkömmliche Telefon und enthält als solches Elemente wie Lautstärkeregler, LCD-Anzeige, Klingel, Tastatur etc. Je nach Unterstützung durch den jeweiligen Hardwaretreiber erlaubt TAPI die Überwachung und Kontrolle dieser Elemente.

Aufbauend auf diesen Basiskomponenten verwenden Anwendungen TAPI-Dienste im Sinne von first-party. Das heißt die CTI-Anwendung ist immer auch Teilnehmer des aktuellen Anrufs, den sie überwachen und kontrollieren kann. Ein third-party Ansatz, bei dem ein zentraler CTI-Server auch Anrufe zwischen zwei weiteren Teilnehmern initiieren und kontrollieren kann, ist hier nicht vorgesehen.

5.1.3.2 Zustände

Analog den übrigen Systemen führt auch TAPI eine Reihe von möglichen Zuständen, die ein Anruf zu einer bestimmten Zeit besitzen kann. Die aus Sicht der gestellten Anforderungen relevanten Zustände sind in Tabelle 5-3 aufgeführt. Der aktuelle Status eines Anrufs kann durch die Funktion *lineGetCallStatus* jederzeit abgefragt werden.

Call State	Beschreibung
idle	Grundzustand, bei dem keine Aktivität für den Anruf vorhanden ist.
Offering (incoming)	Anzeige, dass ein neuer Anruf anliegt.
Accepted (incoming)	Ein Anruf in diesem Zustand wurde von einer der beteiligten Anwendungen übernommen.
dialing (outgoing)	Die angegebene Nummer wird gewählt.
Proceeding (outgoing)	Der Anruf wird aufgebaut. Dieser Zustand tritt nach *Dialing* und vor *Busy* oder *Connected* ein.
Busy (outgoing)	Der Anruf erhielt ein Besetzt-Signal.
Connected (incoming and outgoing)	Die Verbindung ist hergestellt und Informationen werden übertragen.
On hold (incoming and outgoing)	Der Anruf wird gehalten.
Conferenced (incoming and outgoing)	Der Anruf wurde einer Konferenzschaltung hinzugefügt.
On hold pending conference (incoming and outgoing)	Der Anruf wird gehalten, um einen neuen Teilnehmer zur Konferenz zu schalten.
On hold pending transfer (incoming and outgoing)	Der Anruf wird für eine Weiterleitung gehalten.
Disconnected (incoming and outgoing)	Der Anruf wurde durch die Gegenstelle beendet.
Unknown (incoming and outgoing)	Der aktuelle Zustand ist nicht bekannt.

Tabelle 5-3 Relevante Anrufzustände bei TAPI

5.1.3.3 Dienste

Für die allgemein übliche Telefonie-Funktionalität bietet TAPI Dienste in zwei Gruppen an, nämlich *basic services* und *supplementary services*. Die erste Gruppe stellt die Minimalfunktionalität dar, die jeder TAPI-Provider leisten muss. Sie beinhaltet die wesentlichen administrativen Funktionen, wie das Initialisieren von TAPI oder das Öffnen bzw. Schließen einer Kommunikationsleitung sowie Dienste zur Ereignisverwaltung bzw. Statusabfrage. Ebenso enthalten sind die grundlegenden Funktionen zur Anrufkontrolle wie „Anrufen", „Antworten" oder „Auflegen". Die zweite Gruppe erweitert diese Dienste um Funktionen, wie sie in modernen Telefonanlagen zu finden sind. Hier finden sich Dienste, um einen Anruf zu halten, weiterzuleiten oder eine Konferenz einzuleiten. Diese Dienste sind als optional anzusehen, das heißt die jeweilige Implementierung eines TAPI-Providers entscheidet, welche Funktionalität auf der zugrunde liegenden Hardware realisiert werden kann. Hier wird also wieder der first-party Ansatz deutlich mit seiner engen Verknüpfung der Anwendung an die lokale Telefonhardware.

Neben diesen Diensten spezifiziert TAPI noch eine minimale Untergruppe, die als *assisted telephony* bezeichnet wird und dazu dient es Anwendungen, die aus einem anderen Bereich als der Telefonie stammen (etwa eine Textverarbeitung oder eine Tabellenkalkulation), zu ermöglichen, einen Anruf zu initiieren.

Nachfolgend werden auch für TAPI einige wesentliche Dienste aufgeführt, die im Rahmen des neuen Frameworks herangezogen werden müssen.

LineOpen	Ehe Telefonie-Funktionen ausgelöst werden können, muss eine Leitung dafür initiiert werden, was mit diesem Dienst möglich ist.
LineAnswer	Diese Funktion nimmt einen Anruf entgegen. Als Parameter können neben dem konkreten Anruf auch Benutzerinformationen über einen Zeiger auf ein Textfeld an die Gegenstelle übermittelt werden. Voraussetzung dafür ist allerdings die Unterstützung durch die verwendete Infrastruktur.
LineMakeCall	Einleiten eines neuen Anrufs, wobei neben einer gültigen Leitung eine Telefonnummer angegeben wird.
LineDrop	Beenden eines aktuellen Anrufs. Dabei besteht jedoch noch die Möglichkeit, individuelle Informationen anzugeben, die daraufhin noch übermittelt werden.
LineHold/LineUnhold	Halten und Wiederverbinden des Anrufs.
LineSetupTransfer *LineCompleteTransfer*	Die Weiterleitung wird durch zwei Dienste ermöglicht. Im ersten Schritt wird der aktuelle Anruf gehalten und ein neues Anrufobjekt erzeugt, über das eine weitere Verbindung (der sog. *consultation call)* hergestellt werden kann. Daraufhin kann mittels der zweiten Funktion der ursprüngliche Anruf an die neue Gegenstelle übergeben werden.
LineSetupConference *linePrepareAddToConference* *lineAddToConference* *lineRemoveFromConference*	Analog der Weiterleitung wird auch die Konferenz in zwei Schritten aufgebaut. Zusätzlich existieren hier jedoch noch weitere Funktionen, um zum Beispiel neue Teilnehmer der Konferenz hinzufügen bzw. wieder entfernen zu können.

5.1.3.4 Ereignisse

Um Anwendungen über Zustandsänderungen zu informieren, werden drei Mechanismen unterstützt, aus denen die Anwendung wählen kann. Es werden dabei sowohl durch die Anwendung initiierte (*solicitated)* als auch durch von außen angestoßene Ereignisse unterstützt. Die Anwendung kann beim Initialisieren einer Leitung eine dieser drei Arten wählen:

- *callback function*
  In dieser ersten Variante öffnet TAPI ein neues, verstecktes Fenster, an das Ereignisse geschickt werden. Sobald die Anwendung den Systemaufruf *GetMessage* aufruft, springt das System an eine vordefinierte Routine, in der nun wiederum TAPI eine vorher spezifizierte CallBack-Methode der Anwendung aufruft. Dieser Mechanismus ist der einzige, der kompatibel mit älteren Versionen von TAPI ist.

- *win32 events*
  In dieser Variante erzeugt TAPI ein Event-Objekt, das die Anwendung mit herkömmlichen Win32-Methoden wie *WaitForSingleObject* überwachen kann. Der Inhalt eines neuen Ereignisses kann mittels der TAPI-Funktion *lineGetMessage* abgerufen werden.

- *completion ports*
  Die letzte Variante nutzt die in Windows NT neu eingeführten *completion ports*. Dahinter verbirgt sich ein Synchronisationsobjekt, über das mehrere asynchrone Ein- bzw. Ausgabekanäle (Dateien, sockets, pipes) koordiniert werden können. Die Anwendung muss dabei einen speziellen Port anlegen und diesen bei der Initialisierung einer Leitung angeben. Sie hat daraufhin die Möglichkeit, mittels *GetQueuedCompletionStatus* den Status und über einen Rückgabeparameter dieser Methode auch den Inhalt eines neuen Ereignisses abzufragen.

Die derart übermittelten Ereignisse werden in mehrere Bereiche gegliedert, von denen die interessantesten in den sog. *line device messages* zusammengefasst sind. Hier findet sich unter anderem das Ereignis LINE_CALLSTATE, das auftritt, sobald sich der Status eins Anrufs verändert hat. Die Anwendung kann

daraufhin den aktuellen Zustand abfragen, der wie bereits erläutert während der Dauer eines Anrufs unter anderem die folgenden Werte annehmen kann:

LINECALLSTATE_OFFERING
LINECALLSTATE_BUSY
LINECALLSTATE_CONNECTED
LINECALLSTATE_ONHOLD
LINECALLSTATE_CONFERENCED
LINECALLSTATE_DIALING

5.1.3.5 Anruf/Daten

Der Verknüpfung von individuellen Anwendungsdaten mit einem Anruf trägt TAPI durch die Unterstützung sog. *user-user information* Rechnung. Durch ein eigens dafür spezifiziertes Feld in der Datenstruktur eines Anrufs ist es prinzipiell möglich, individuelle Informationen zwischen Teilnehmern eines Anrufs zu versenden. Dabei ist die Übertragung unter anderem während des Aufbaus, des Annehmens, der Weiterleitung eines Anrufs oder vor dem Auflegen vorgesehen. Das Eintreffen einer neuen Nachricht wird in diesen Fällen der Anwendung durch ein Ereignis *Line_Callinfo* übermittelt.

Voraussetzung dabei ist allerdings auch bei TAPI, dass der jeweilige Service-Provider die Übermittlung solcher Daten auf Basis der jeweiligen Netzinfrastruktur umsetzen kann.

5.1.3.6 Bewertung

Wie die nachfolgende Übersicht nochmals darstellt, bietet auch TAPI alle wesentlichen Dienste und Funktionen an, die für die Realisierung eines Basissystems in einem Call Center benötigt werden. Im Prinzip ist also auch der Einsatz eines Systems denkbar, das auf einem einfacheren Modell basiert und zum Beispiel nur den first-party Ansatz unterstützt.

	Dienst	Unterstützt durch
CallControl	MAKE CALL	*lineMakeCall*
	ANSWER CALL	*lineAnswer*
	HOLD	*lineHold*
	RECONNECT	*lineUnhold*
	TRANSFER CALL	*lineSetupTransfer, lineCompleteTransfer*
	CONFERENCE	*lineSetupConference, linePrepareAddToConference, lineAddToConference, lineRemoveFromConference*
Ereignisse	*ALERT*	*LINECALLSTATE_OFFERING*
	CALLING	*LINECALLSTATE_DIALING*
	ANSWERED	*LINECALLSTATE_ACCEPTED*
	ACCEPT	*LINECALLSTATE_CONNECTED*
	HANGUP	*LINECALLSTATE_DISCONNECTED*
	HOLD	*LINECALLSTATE_ONHOLD*
	RECONNECT	*LINECALLSTATE_CONNECTED*
Anruf / Daten		User-user information

Tabelle 5-4 Funktionalität von TAPI

Allerdings muss das System nicht nur Dienste für das Framework bereitstellen, sondern auch als CTI-System den Betrieb des Call Centers gewährleisten. Und in diesem Zusammenhang ist der Ansatz von TAPI gerade für größere Installationen eher kritisch zu betrachten. Auch die Tatsache, dass hier eine enge Verknüpfung mit der Windows-Plattform vorhanden ist, kann gerade im Serverbereich ein Problem darstellen, da hier auch in Call Centern meist relativ heterogene Umgebungen herrschen.

5.1.4 TSAPI

NetWare© Telephony Services sowie die dazu gehörende Spezifikation von TSAPI (*telephony services application programming interface)* wurde von Novell und AT&T entwickelt und gilt als einer der verbreitetsten CTI-Standards. Er basiert im Wesentlichen auf dem CSTA-Standard der European Computer Manufacturers Association (ECMA).

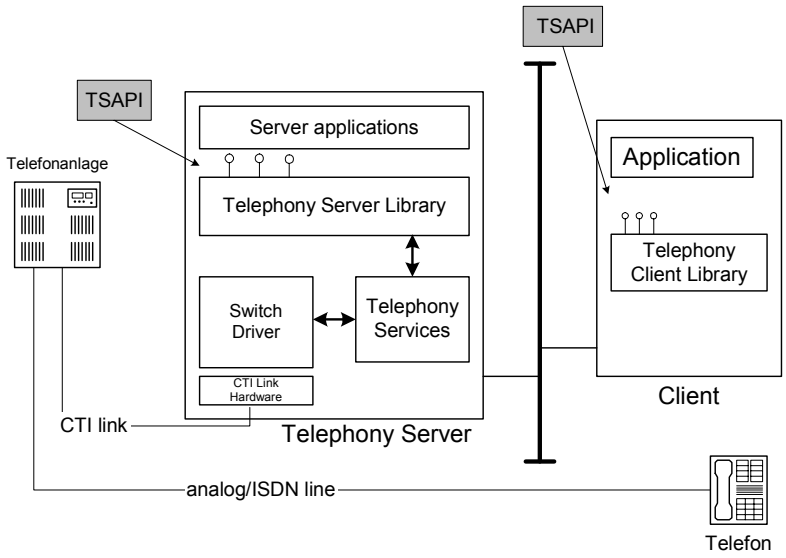


Abbildung 5-3 TSAPI-Architektur

Die Funktionalität dieser Dienste entspricht der klassischen Definition von CTI, wobei der third-party-Ansatz gewählt wurde. Ziel ist demnach die Integration der auf der Seite eines ausgezeichneten Servers liegenden Kontrolle über die Telefonanlage in die verschiedenen Anwendungen eines Netzwerkes. Dadurch wird ohne weitere Hardware eine rein logische Verbindung des Telefons eines Arbeitsplatzes mit einer entsprechenden Anwendung geschaffen. Lediglich der ausgezeichnete Server hat eine dedizierte Verbindung zur Telefonanlage (*CTI link*). Die konkrete Architektur ist in Abbildung 5-3 dargestellt.

Kernstück der Architektur ist der Serverrechner, der die Anbindung an den Switch realisiert. Er benötigt dazu proprietäre Hardware sowie entsprechende Treiber für die Kommunikation mit der Telefonanlage. Als *telephony server* wird eine Instanz der *telephony services* bezeichnet, die die TSAPI-Dienste mittels der jeweiligen Hardware sowie Treiber umsetzt. Über Client- bzw. Server-Bibliotheken können unterschiedliche Anwendungen nun unter Nutzung von TSAPI die unterschiedlichen CTI-Dienste in Anspruch nehmen. Diese Schnittstelle beinhaltet in der C-Syntax die Definition aller notwendigen Funktionen, Datentypen und Ereignisse, die für CTI-Anwendungen von Bedeutung sind.

5.1.4.1 Modell

Das zugrunde liegende Modell von TSAPI ist dem in Kapitel 2.5 dargestellten sehr ähnlich. Hier zeigt sich deutlich die enge Verwandtschaft mit dem CSTA-Standard. TSAPI definiert daher ebenfalls eine Reihe von Objekten wie etwa *device, call* oder *connection*, die das Modell dieser Spezifikation beschreiben.

Device

Ein Device ist analog den anderen Modellen ein physikalisches Endgerät oder aber ein logisches Gerät, wie etwa eine Gruppe von Endgeräten oder ein ACD-System. Es wird durch die Attribute *Type* und *Class* beschrieben. Durch den Typ werden die verschiedenen Geräte wie etwa das klassische Telefon (*station)*,

eine Leitung mit mehreren Geräten (*line)* oder aber ein ACD-System (*acd)* unterschieden. Diese werden weiterhin in verschiedene Klassen eingeteilt, je nachdem welche Kommunikationsmedien (*voice, image, data, other)* sie unterstützen. Eine TSAPI-Anwendung greift auf diese Objekte über einen *device identifier* zu, der entweder statisch oder aber dynamisch zugewiesen wird. Darüber hinaus besitzt jedes Gerät einen Zustand, der die Zustände aller Verbindungen beschreibt, die zu einem Zeitpunkt diesem Gerät zugewiesen sind.

Connection

Eine Verbindung (*Connection)* beschreibt die Beziehung zwischen einem Gerät (*Device)* und einem Anruf (*Call)*. Anwendungen, die analog den anderen Objekten über einen eindeutigen Identifikator Zugriff auf eine Verbindung erhalten, können diese überwachen sowie steuern. Der Zustand einer Verbindung beschreibt den Zustand eines Anrufs bezogen auf ein Gerät. Abbildung 5-4 zeigt einen beispielhaften Zustandsautomaten einer Verbindung. Da TSAPI allerdings unabhängig von einzelnen Telefonanlagen bleiben soll, kann sich dieser Automat bei individuellen Implementierungen auch verändern.

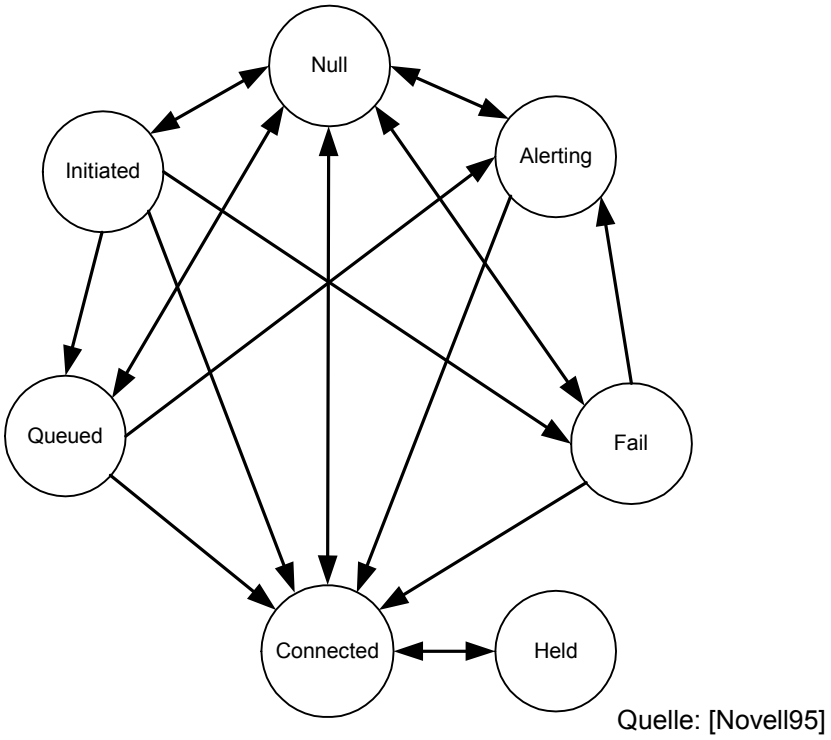


Abbildung 5-4 Zustandsautomat einer TSAPI-Connection

Call

Der Anruf selbst wird durch das Objekt *Call* beschrieben. Es hat wiederum einen eindeutigen Identifikator (*Call Identifier*) sowie einen Zustand (*Call State)*. Der Zustand eines Anrufs ist definiert als die Menge aller Zustände der Verbindungen zu allen an diesem Anruf beteiligten Geräten. Für den Fall von zwei Verbindungen führt TSAPI einen neuen Zustand ein, wie in Tabelle 5-5 dargestellt.

5.1.4.2 Zustände

Die Kombination der Zustände einzelner Verbindungen ergibt einen Zustand für den jeweiligen Anruf, der in TSAPI ebenfalls definiert wird und in folgender Tabelle zusammengefasst ist. So wird beispielsweise zwischen den Zuständen *Delivered* sowie *Established* unterschieden, je nachdem ob der zweite Teilnehmer bereits abgenommen hat oder sich seine lokale Verbindung noch im Zustand *Alerting* befindet.

Verbindung 1	Verbindung 2	Anruf
Alerting	Connected	Received
Alerting	Hold	Received-On Hold
Connected	Alerting	Delivered
Connected	Connected	Established
Connected	Failed	Failed
Connected	Hold	Established-On Hold
Connected	Null	Originated
Connected	Queued	Queued
Hold	Alerting	Delivered-Held
Hold	Connected	Established-Held
Hold	Failed	Failed-Held
Hold	Queued	Queued-Held
Initiated	Null	Pending
Null	Null	Null

Tabelle 5-5 Zustände eines TSAPI-Anrufs

5.1.4.3 Dienste

Basierend auf dem eben dargestellten Modell erhält eine Anwendung über eine Reihe von Diensten Zugriff auf die jeweilige CTI-Funktionalität. Dabei unterscheidet TSAPI zwischen ACS (*API control services)* und *CSTA services*. Über ACS kommunizieren Anwendungen mit dem jeweiligen TSAPI-Server, während sie über CSTA-Dienste die übliche und im Folgenden noch näher beschriebene Kontrolle über einen Anruf erhalten. So muss eine Anwendung einen *ACS stream* öffnen, der eine logische Verbindung zur Telefonanlage darstellt und über welchen dann CSTA-Dienste wie etwa Anrufen oder Auflegen ausgeführt werden.

Diese Dienste sind allerdings im Hinblick auf eine Bewertung von TSAPI als CTI-Basissystem weniger interessant, weshalb im Folgenden die Dienste näher betrachtet werden, die der Anrufkontrolle dienen und in der TSAPI-Spezifikation als *switching functions* bezeichnet werden. Sie dienen der konkreten Kontrolle eines Anrufs. TSAPI-Anwendungen nutzen diese, um die üblichen Aktionen wie das Einleiten eines neuen Anrufs oder das Annehmen eines ankommenden Anrufs zu veranlassen. Um einen Eindruck der Funktionalität dieses Teils der Schnittstelle zu erhalten, werden einige der Funktionen exemplarisch aufgeführt.

CstaAlternateCall( )	Diese Funktion entspricht dem Makeln zwischen zwei Anrufen. Eine gerade aktive Verbindung wird auf Halten gesetzt und ein ankommender bzw. gerade gehaltener Anruf wird aktiviert.
CstaAnswerCall( )	Annehmen eines Anrufs. Der Zustand der Verbindung geht dabei von *alerting* in *connected* über.
CstaClearCall( )	Trennt alle beteiligten Geräte von einem Anruf und löscht das Objekt.
CstaConferenceCall( )	Konferenzschaltung, die einen gehaltenen sowie einen aktiven Anruf zu einem neuen, aktiven Anruf zusammenschaltet.
CstaConsultationCall( )	Dieser Dienst setzt den aktiven Anruf auf *hold* und initiiert daraufhin einen neuen Anruf.
CstaHoldCall( )	Setzt eine Verbindung auf *hold.*
CstaMakeCall( )	Initiiert einen Anruf zwischen zwei Geräten.
CstaTransferCall( )	Übergibt einen gehaltenen Anruf an einen gerade aktiven Anruf.

Tabelle 5-6 TSAPI-Dienste

Diese Dienste entsprechen den elementaren Abläufen aus Abschnitt 2.5.2.

5.1.4.4 Anruf/Daten

Wie in der JTAPI-Spezifikation (vgl. 5.1.2) ist auch in TSAPI die Übertragung nicht standardisierter Information zwischen Anwendung und Telefonanlage über sog. *privat data* vorgesehen. Der Großteil der Ereignisse und Dienste sieht dafür nicht strukturierte Datenbereiche vor, die für individuelle Erweiterungen insbesondere der einzelnen Switch-Hersteller genutzt werden können.

5.1.4.5 Ereignisse

Wie in allen CTI-Systemen werden Zustandsänderungen der einzelnen Objekte durch Ereignisse an die Anwendung gemeldet. TSAPI übermittelt Ereignisse über den ACS *stream.* Sie werden dort in chronologischer Reihenfolge eingestellt und in einer internen Warteschlange gepuffert. Die Anwendung hat die Möglichkeit, auf die Ereignisse durch zwei Dienste zu reagieren. Im ersten Fall (*blocking mode)* wird ein Ereignis durch den Dienst *acsGetEventBlock()* abgeholt, der erst bei Eintreffen eines solchen wieder zurückkehrt. Diese Methode kann für reine Monitoring-Funktionen sinnvoll sein. Im anderen Fall (*non-blocking mode*) kann die Applikation mittels *acsGetEventPoll( )* zur jeder Zeit ein aktuell eingetroffenes Ereignis abrufen. Dieser Dienst kehrt sofort wieder zurück, entweder mit dem aktuellen Ereignis oder dem Ergebnis, dass die Warteschlange derzeit leer ist. Die Anwendung muss also selbst eine entsprechende Verwaltung der Ereignisse implementieren, um jederzeit ausreichend aktuelle Zustandsinformationen zu besitzen.

Jeder Dienst der TSAPI-Spezifikation wird, sofern er erfolgreich ausgeführt werden konnte, durch ein sog. *confirmation event* bestätigt. Beispiele dafür sind *CSTAAnswerCallConfEvent* oder *CSTAClearCallConfEvent.* Während diese Ereignisse nur nach erfolgtem Aufruf eines Dienstes auftreten, muss ein CTI-System jedoch auch jederzeit auf asynchron auftretende Ereignisse wie etwa einen neuen Aufruf reagieren können. Auch die Interaktion eines Benutzers mit dem lokalen Telefon kann zu Ereignissen führen, die Zustände von Verbindungen und Geräten verändern und somit vom CTI-System verarbeitet werden müssen. Für derartige, asynchrone Ereignisse bietet TSAPI eigene Funktionen mit denen Monitore eingerichtet werden können, die den Zustand von Geräten und Anrufen überwachen und bei Zustandsänderung ein entsprechendes Ereignis initiieren. Sie können dabei auch Filter enthalten, die für die Anwendung uninteressante Ereignisse unterdrücken. Wichtige Meldungen solcher Monitore sind sog. *call event reports*, die die Anwendung über den Zustand eines Anrufs informieren. Beispiele solcher Meldungen sind:

CSTADeliveredEvent (Alerting)	Zeigt an, dass ein neuer Anruf an einem Gerät ankommt.
CSTAConferencedEvent	Meldung, dass zwei Anrufe zu einer Konferenz geschaltet wurden.
CSTAEstablishedEvent	Zeigt an, dass ein Anruf einem Gerät zugeordnet wurde.
CSTAFailedEvent	Fehlermeldung, wenn ein Anruf nicht hergestellt werden konnte.
CSTAHeldEvent	Meldung, dass ein Anruf gehalten wird.
CSTAOriginatedEvent	Zeigt an, dass die Telefonanlage aufgrund eines Dienstaufrufs versucht, einen Anruf durchzuführen.

Tabelle 5-7 TSAPI-Ereignisse

Es ist zusammenfassend festzuhalten, dass die Mechanismen der Ereignisverwaltung etwa im Vergleich zu JTAPI weniger Funktionalität aufweisen und dadurch höhere Anforderungen an die Implementierung der Anwendung bzw. im konkreten Fall der Framework-Lösung gestellt werden.

5.1.4.6 Bewertung gemäß den Anforderungen

Nachfolgende Tabelle stellt die Dienste und Ereignisse von TSAPI den allgemeinen Anforderungen, die an ein CTI-Basissystem zu stellen sind, nochmals gegenüber.

	Dienst	Unterstützt durch
CallControl	MAKE CALL	cstaMakeCall( )
	ANSWER CALL	cstaAnswerCall( )
	HOLD	cstaHoldCall( )
	RECONNECT	cstaReconnectCall()
	TRANSFER CALL	cstaTransferCall( )
	CONFERENCE	cstaConferenceCall( )
Ereignisse	ALERT	CSTADeliveredEvent
	CALLING	CSTAOriginatedEvent
	ANSWERED	CSTAEstablishedEvent
	ACCEPT	CSTAEstablishedEvent
	HANGUP	CSTAConnectionClearedEvent
	HOLD	CSTAHeldEvent
	RECONNECT	CSTARetrieveEvent
Anruf / Daten		Private data

Tabelle 5-8 Funktionalität von TSAPI

Bis auf die Verknüpfung von individuellen Daten zu einem Anruf liefert TSAPI ausreichend Funktionalität, um als Basissystem für eine fortschrittliche CTI-Lösung genutzt werden zu können. Lediglich die Verknüpfung individueller Informationen mit einem Anruf wird in TSAPI nicht spezifiziert. Abhängig von den unterschiedlichen Telefonanlagen und den dafür implementierten TSAPI-Treibern kann dies über den Private-Data Mechanismus in der Regel realisiert werden.

Als CTI-Basis für Call Center ist TSAPI die wohl am häufigsten eingesetzte und von den meisten Produkten unterstützte Schnittstelle. In dieser Hinsicht ist sie gegenüber neueren Ansätzen wie JTAPI eindeutig im Vorteil. Es ist zwar zu erwarten, dass sich auch hier neuere Ansätze im Laufe der Zeit ebenfalls durchsetzen werden, die dominierende Stellung von TSAPI in konkreten Call Centern wird jedoch noch einige Zeit bestehen bleiben.

Abschließend ist natürlich die Verwendung einer reinen C-Schnittstelle für die Realisierung einer modernen und insbesondere objektorientierten Lösung ein wichtiger Aspekt, der wohlüberlegt sein sollte. Dadurch entsteht in jedem Fall ein Bruch zwischen objektorientiertem und prozeduralem Ansatz, der sowohl Einfluss auf das Design des gesamten Systems, als auch auf die Wahl einer geeigneten Middleware hat.

5.1.5 Fazit

Die drei betrachteten Alternativen bieten alle eine ausreichende Grundfunktionalität, um als CTI-Basissystem für das Framework eingesetzt zu werden. Lediglich TAPI ist hier durch das sehr einfache Modell etwas kritisch zu bewerten. Weiterhin zeigt sich, dass in allen Fällen die Zustandsautomaten zwar eine Basis, aber noch nicht alle Zustände, die in der Spezifikation des Frameworks definiert wurden, anbieten und die Verknüpfung eines Anrufs mit Daten zum Teil von individuellen Implementierungen abhängt.

Durch den durchgehend objektorientierten Ansatz von JTAPI sowie die durch Java erlangte Unabhängigkeit von Hardware- und Systemen hat dieser Ansatz jedoch als Basissystem für das angestrebte Framework eindeutige Vorteile und wird deshalb auch als bevorzugte Alternative betrachtet.

5.2 Middleware

Wie die meisten modernen, verteilten Systeme nutzt auch das CTI-Framework eine Middleware für die Kommunikation bzw. Interaktion der einzelnen Komponenten. Analog der Analyse einiger CTI-Basissysteme sind auch für eine Middleware die Anforderungen konkreter zu spezifizieren, um daraufhin einige Alternativen bewerten zu können.

Vorab ist es jedoch notwendig, eine möglichst eindeutige Definition für den Begriff Middleware zu finden, da dieser häufig sehr allgemein und vielfältig benutzt wird. Eine sehr intuitive Definition geben [Orfali96a], die Middleware als

> *„[..] the slash (/) component of client/server"*

bezeichnen. [ISM98] bezeichnet Middleware konkreter als Softwareschicht, die zwischen den Anwendungen und der Netzwerkschicht heterogener Plattformen und Protokollen liegt. Eine sehr ähnliche Definition gibt auch [Bern96], der Middleware anhand der angebotenen Dienste (*middleware services)* definiert. Von derartigen Diensten wird gefordert, dass sie mehrere Plattformen unterstützen, dass sie verteilt sind, standardisierte Schnittstellen und Protokolle aufweisen und generisch im Hinblick auf Anwendungen und Einsatzgebiete sind.

[ISM98] gibt weiterhin eine recht nützliche Einteilung der verschiedenen Middleware-Ansätze in 5 Bereiche:

- Auf RPC (*remote procedure call)* basierende Middleware
- Message Oriented Middleware (MOM)
- Portable Transaction Processing (TP) Monitors
- Object Request Brokers
- Database Middleware

In der Praxis wird der Begriff Middleware leider äußerst frei definiert und auch verwendet, so dass die angegebenen Definitionen nützlich sind. Im Hinblick auf das zu entwerfende Framework wird Middleware deshalb definiert als eine aus mehreren Bausteinen bestehende Software, welche die Interaktion der einzelnen Komponenten über ein Netzwerk und auch über unterschiedliche Komponenten hinweg wesentlich unterstützt. Dabei sind eine Reihe von Diensten erforderlich, die nachfolgend genauer spezifiziert werden.

5.2.1 Anforderungen

Es existiert eine Reihe von unterschiedlichen Anforderungen, die an Middleware-Technologien gestellt werden, was nicht zuletzt an der sehr allgemeinen Definition dieses Begriffes und dem damit sehr weiten Einsatzgebiet von Middleware liegt. Angesichts der spezifizierten Funktionalität des Frameworks lassen sich jedoch für diesen Fall die benötigten Dienste bzw. die geforderte Funktionalität etwas eingrenzen, die die Interaktion der einzelnen Komponenten unterstützen müssen. Es handelt sich dabei einmal um die Einzelkomponenten des Frameworks selbst, um eventuell betroffene Komponenten des CTI-Systems sowie um die Anwendung, die als Client auf dem Arbeitsplatzrechner des Call Centers läuft.

Interaktion bedeutet zuerst einmal die Kommunikation zwischen diesen Programmteilen, auch über das Netzwerk hinweg. Anforderung an die Middleware ist dabei, dass Details der Kommunikation weitgehend verborgen werden und somit die entfernte Komponente analog einer lokalen direkt angesprochen werden kann. Weitere Kriterien sind die Plattformen, auf denen die Komponenten ablaufen sowie die Programmiersprachen, mit Hilfe derer sie implementiert werden. Im Falle von modernen Call Centern kann man davon ausgehen, dass hier Windows NT bzw. Windows 2000 sowie Unix als Plattformen und Java sowie C++ als Programmiersprachen zu finden sind.

Weiterer Aspekt ist die Mächtigkeit, mit der sich die Schnittstellen verteilter Komponenten definieren lassen. Da die Konzepte der Objektorientierung auch in modernen verteilten Systemen Einzug gehalten haben und diese dementsprechend verteilte Objekte nutzen, sind die Möglichkeiten der Schnittstellenbeschreibung auch im Hinblick auf die Umsetzung objektorientierter Ansätze hin zu betrachten. Identifiziert werden Objekte und ihre Schnittstellen über einen Naming Service.

Neben der herkömmlichen Kommunikation ist die Interaktion über Ereignisse gerade im Bereich der Telefonsysteme eine wichtige Anforderung. Da hier an verschiedenen Stellen auch asynchron Ereignisse, wie etwa ein neuer Anruf oder aber eine plötzlich abgebrochene Verbindung, auftreten können, sollte die Middleware Mechanismen zur Verfügung stellen, diese Ereignisse an alle oder ausgewählte Partner zu propagieren.

Da Call Center zu gewissen Zeiten aus- bzw. überlastet sind, geht es auch um die Frage, wie skalierbar die Systeme gestaltet werden können. Erstes Kriterium ist hier einmal die Performance, die die Middleware grundsätzlich zu leisten im Stande ist. Weiterhin gehören hierzu Funktionen, die es ermöglichen, Anfragen von Clients automatisch auf mehrere verfügbare Server gleicher Funktionalität zu verteilen und damit die jeweilige Last entsprechend auszugleichen (*load balancing)*. Solche Mechanismen und Technologien wie die Unterstützung von Threads oder ein intelligentes Verwalten von Netzwerkverbindungen steigern die Skalierbarkeit des Gesamtsystems.

Ein letzter Aspekt ist die Sicherheit und Zuverlässigkeit des Systems. Dazu gehören zum einen die Mechanismen, die sensible Daten durch Verschlüsselung und Authentifizierung berechtigter Benutzer vor unberechtigtem Zugriff schützen. Eine verschlüsselte Übertragung dieser Informationen wird dabei in den meisten Fällen eine untergeordnete Rolle spielen, da es sich in der Regel um lokale Intranet-Umgebungen handelt. Interessant wird dies allerdings bei echten verteilten Call Centern, die über öffentliche Netze, etwa das Internet, verbunden werden. Eine andere Technologie, die die Zuverlässigkeit des gesamten Systems erhöht, ist die Integration von Transaktionsverarbeitung. Betrachtet man die bisherige Spezifikation des Frameworks, so werden verteilte Transaktionen für die Funktionalität des Systems selbst im Grunde nicht benötigt. Wichtig wird dies jedoch, wenn es um die engere Integration bestehender Systeme wie etwa einer Host-Anwendung einer Bank geht. In diesem Falle muss zumindest die Client-Anwendung mit transaktionsverarbeitenden Systemen kommunizieren und könnte dabei auf Dienste einer Middleware zurückgreifen.

Zusammenfassend existieren also vor dem Hintergrund des zu entwerfenden Systems eine Reihe von Kriterien, die in Tabelle 5-9 nochmals zusammengefasst und den einzelnen Kategorien zugeordnet sind. So ist die Kommunikation und Definition von Schnittstellen sowie ein Naming Service ein absolut notwendiges Kriterium, während auf Unterstützung mehrerer Plattformen und Sprachen, Ereignisse und Skalier-

barkeit unter Umständen auch verzichtet werden kann. Funktionen zur Sicherheit und Transaktionsunterstützung sind hilfreich aber nicht notwendig.

Anhand dieser Aspekte werden nun einige wichtige und aktuelle Middleware-Technologien betrachtet.

	Kategorie
Transparente Kommunikation	A
Unterstützung mehrerer Plattformen und Programmiersprachen	B
Schnittstellendefinition	A
Naming Service	A
Ereignisse	B
Skalierbarkeit	B
Sicherheit	C
Transaktionsunterstützung	C

Tabelle 5-9 Anforderungen an eine Middleware

5.2.2 Middleware-Ansätze

Die Menge der Ansätze von Middleware-Lösungen ist zu umfassend, um alle Alternativen für einen eventuellen Einsatz zu untersuchen. Wie einleitend schon angesprochen, lassen sich hier verschiedene Bereiche unterscheiden wie RPC, MOM, TP-Monitore, Database-Middleware oder Object Request Broker. Anhand der bereits verfügbaren Spezifikation des Frameworks lassen sich diese vielfältigen Ansätze jedoch relativ gut eingrenzen. Diese Spezifikation zielt ganz wesentlich auf ein verteiltes, objektorientiertes System ab. Somit rücken auch die Middleware-Ansätze der letzten Kategorie in den Vordergrund, die eine Interaktion von einzelnen Objekten unterstützen.

Betrachtet man die derzeitig verfügbaren Technologien im Bereich der objektorientierten, verteilten Systeme, so haben sich am Markt drei Lösungen etabliert, die somit auch Basis der nachfolgenden Betrachtungen sind. Dies ist zum einen CORBA, die *Common Object Request Broker Architecture* der Object Management Group, Java RMI (*remote method invocation)* der Firma SUN bzw. die auf Java aufsetzende Spezifikation EJB *(enterprise java beans)* sowie der Standard für verteilte Komponenten im Umfeld von Microsoft Windows, COM+.

Diese drei Vertreter werden also in den nachfolgenden Abschnitten etwas genauer untersucht. Dabei soll ein erster Überblick über die unterschiedlichen Technologien gegeben werden, ohne auf die einzelnen Details einzugehen. Für eine genaue Beschreibung der einzelnen Vertreter sei in diesem Zusammenhang auf die offiziellen Spezifikationen sowie die zu diesem Thema reichlich vorhandene Literatur verwiesen.

5.2.3 CORBA

CORBA (*Common Object Request Broker Architecture) (*siehe [OMG98]) als Standard der OMG für die Entwicklung verteilter Anwendungen und Systeme ist zur Zeit die interessanteste und vorherrschende Middleware.

Ein CORBA-Objekt implementiert eine bestimmte Funktionalität und stellt Clients eine Reihe von Diensten zur Verfügung, die diese über den Aufruf von Methoden nutzen können. Diese Dienste werden unabhängig von einer konkreten Programmiersprache in IDL (*interface definition language)* spezifiziert. Sowohl der Ort eines Objektes, die Plattform des Servers und die Sprache, in der es implementiert wird, ist für den Client dabei transparent. So müssen Objekte nicht unbedingt über das Netzwerk verteilt sein, sondern können sich auch auf dem lokalen Rechner befinden. In diesem Falle verwendet man CORBA als Middleware für die Kommunikation zwischen einzelnen Anwendungen. Konkrete CORBA-Implementierungen sind dabei für die effiziente Realisierung der Kommunikation verantwortlich (vgl. [Orfali98]).

Wie diese Implementierungen diese Aufgabe umsetzen, zum Beispiel als zentraler Server oder als Bibliothek, bleibt ihnen überlassen. So wickelt beispielsweise der ORB von Inprise die Kommunikation über *shared memory*, über Interprozesskommunikation oder über TCP/IP ab, je nachdem, wo sich Client und Server befinden. Dabei müssen Details wie etwa die effiziente Verwaltung von Netzwerkverbindungen ebenfalls durch die Implementierung behandelt werden. Innerhalb eines ORBs werden Objekte anhand einer eindeutigen Objektreferenz identifiziert, die solange gültig ist, solange das Objekt instanziert ist. Auch hier bleibt die konkrete Implementierung einer Objektreferenz dem Hersteller eines ORBs überlassen. Allerdings schreibt CORBA vor, wie der Zugriff auf diese Referenzen in den einzelnen Programmiersprachen umgesetzt werden muss. Anwendungen bleiben also prinzipiell lauffähig, auch wenn die ORB-Implementierung gewechselt wird.

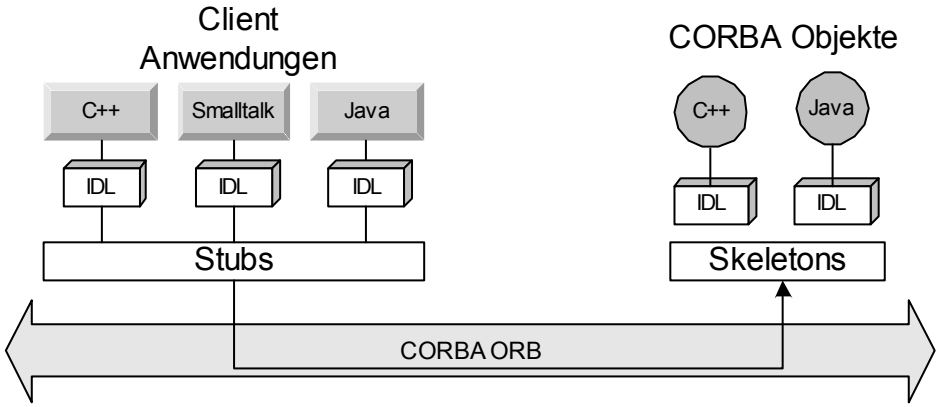


Abbildung 5-5 Client/Server-Beziehung mit CORBA

Schnittstellen werden in CORBA in IDL beschrieben und sind damit unabhängig von einzelnen Programmiersprachen. Die Grammatik von IDL entspricht im Wesentlichen der von C++, erweitert um einige Schlüsselworte, die den Aspekten verteilter Systeme dienen. Sie ist allerdings rein deklarativ, enthält also keine Konstrukte, um Algorithmen oder Variablen zu definieren. Abbildung 5-6 zeigt als Beispiel eine IDL-Definition der Schnittstelle `Customer`. Im ersten Abschnitt werden Strukturen definiert, die dann als Parameter oder Rückgabewert einzelner Methoden verwendet werden können. Neben Strukturen können auch Arrays oder Sequenzen variabler Länge als zusammengesetzte Typen definiert werden. Diese basieren wiederum auf den vorgegebenen Basistypen, die allgemein übliche Datentypen für Ganz- und Fließkommazahlen, Zeichen und Zeichenketten sowie Boolesche Werte umfassen. Darüber hinaus definiert IDL noch den Typ *any*, der als neutraler Wert jeden in IDL gültigen Typ repräsentiert.

```
Interface CustomerManager
{
struct Customer
{
string name;
string firstName;
string title;
};
struct AddressData
{
string street;
string zip;
string city;
string country;
};

typedef sequence <AddressData> AddressDataList;
typedef sequence <CustomerData> CustomerDataList;

// ======== services //================================================
// search and read Customer and Address
CustomerData getCustomer(in string customerId)
                                        raises (ExternalError, InternalError);

CustomerDataList searchCustomer(in string name, in string firstName, in string
dateOfBirth,
in unsigned long maxNumberOfResults, out unsigned long numberOfHits)
                                        raises (InternalError);

AddressDataList getCustomerAddresses(in string customerId)
                                        raises (ExternalError, InternalError);
};
```

Abbildung 5-6 Schnittstellenbeschreibung mit IDL

Parameter von Methoden müssen als In-, Out- oder InOut-Parameter gekennzeichnet werden, wodurch angegeben wird, ob darüber Information nur vom Client zum Server, vom Server zum Client oder in beide Richtungen weitergegeben werden soll. Methoden können im Fehlerfall auch *Exceptions* hervorrufen, die dann dem Client entsprechend übermittelt werden müssen.

Einer der wesentlichen Ansätze der CORBA-Spezifikation ist die Unterstützung mehrerer Programmiersprachen für die Implementierung von Client-/Serverprogrammen, die über einen bzw. auch mehrere Object Request Broker miteinander kommunizieren können. Durch die Definition von Schnittstellen in einer neutralen IDL-Syntax und die klar definierte Abbildung auf Sprachen wie C, C++, Smalltalk, Cobol, Ada sowie Java eignet sich dieser Ansatz besonders für den Einsatz in heterogenen Systemen oder zur Anbindung bestehender Komponenten. Die Plattformen, die unterstützt werden, sind dabei allerdings abhängig von den konkreten Implementierungen, die zur Verfügung stehen. Man kann derzeit jedoch davon ausgehen, dass die vorhandenen Implementierungen verschiedener Firmen (Inprise, IONA, BEA, IBM) so gut wie alle gängigen Systemplattformen abdecken.

Um die Interoperabilität verschiedener CORBA-Produkte zu gewährleisten, definiert die Spezifikation ebenfalls die Protokolle und Nachrichtenformate zur Kommunikation von ORB zu ORB. Dazu gehört das *General Inter-ORB Protocol* (GIOP), das sehr allgemein die Nachrichten definiert, die zur Interaktion über ein beliebiges, verbindungsorientiertes Transportprotokoll genutzt werden. Eine spezielle Form davon ist das *Internet Inter-ORB Protocol (IIOP),* das auf TCP/IP aufsetzt. Zu diesem Protokoll gehört auch die Definition einer allgemeingültigen Version der Objektreferenz, der *Interoperable Object Reference (IOR).*

5.2.3.1 Wichtige Eigenschaften

Skalierbarkeit

Die CORBA-Spezifikation definiert selbst eine Reihe von Mechanismen, die zur Skalierbarkeit größerer Systeme herangezogen werden können. Zentrale Komponente dabei ist der *Object Adapter*, der bestimmt, wie und wann Serverobjekte aktiviert und Methodenaufrufe durchgeführt werden. Bereits in Version 2.0 werden dazu in der Definition des *Basic Object Adapter (BOA)* vier wesentliche Strategien vorgegeben, die

eine CORBA-Implementierung unterstützen muss, nämlich *shared server*, *unshared server*, *server-per-method* und *persistent server*. Im ersten Fall wird ein Serverprozess bei der ersten Anfrage eines Clients an eines seiner Objekte gestartet, dem dann auch alle nachfolgenden Aufrufe zugestellt werden. Dagegen erhält in der zweiten Variante jedes Serverobjekt seinen eigenen Prozess. Die Strategie *server-per-method* weist den Adapter an, für jede Anfrage einen neuen Prozess zu starten. Ein persistenter Server wird dagegen nicht automatisch gestartet, sondern muss außerhalb des Adapters initialisiert werden. Mit diesen Strategien kann man den grundlegenden Anforderungen an Skalierbarkeit begegnen, indem zum Beispiel die Anzahl von Serverprozessen entsprechend erhöht wird, oder die Anforderung an Ressourcen über gemeinsam genutzte Server niedrig gehalten wird.

Seit der Version 2.2 wird anstatt des BOA der *Portable Object Adapter (POA)* eingeführt, der neben einer besseren Portabilität auch erweiterte Konfigurationsmöglichkeiten bietet. Die erweiterte Architektur führt dazu unter anderem die Komponente *Servant* ein, über die Methodenaufrufe abgewickelt werden. Ein Servant stellt also eine zur Laufzeit instanzierte Implementierung eines Objektes dar. Ein vordefinierter POA (*root POA*), den eine CORBA-Implementierung zur Verfügung stellt, unterstützt nun einige Strategien, wie die verfügbaren Servants verwaltet und aufgerufen werden, die anhand mehrerer Parameter geeignet konfiguriert werden. Die Konfigurationsmöglichkeiten sind hier sehr vielfältig und übersteigen noch die Ansätze, die der BOA integriert hat.

Unabhängig von diesen in der Spezifikation klar definierten Mechanismen bieten die Hersteller konkreter CORBA-Implementierungen meist ebenfalls noch Komponenten und Erweiterungen an, die der Skalierbarkeit nützen. Als Beispiel sei hier der Smart Agent des VisiBroker von Inprise erwähnt. Dieses Programm vereint einen einfachen Namensdienst mit einer grundlegenden load-balancing Strategie. Sind im Netz mehrere Objektimplementierungen zu einer Zeit verfügbar, so werden Anfragen automatisch gleichmäßig an alle Server verteilt.

Event Service

Der CORBA *Event Service* (siehe [OMG97]) ist einer von mehreren Diensten, die die OMG spezifiziert hat. Er ist für eine Implementierung einer Call Center Lösung interessant, da die Kommunikation mit einer CTI-Middleware ebenfalls eine Reihe von Ereignissen beinhaltet, die den Client über verschiedene Zustandsänderungen informiert. Für die Nutzung von JTAPI wird dies in Kapitel 5.1.2.4 näher erläutert.

Während Aufrufe von Objekten in CORBA synchron abgewickelt werden, dient der Event Service einer entkoppelten Kommunikation zwischen einzelnen Objekten. Dabei nehmen beteiligte Objekte entweder die Rolle eines Erzeugers (*supplier)* ein, der neue Ereignisse produziert, oder eines Verbrauchers (*consumer*), der die Daten solcher Ereignisse verarbeitet. Für den Austausch von Ereignissen zwischen Verbraucher und Erzeuger gibt es zwei unterschiedliche Ansätze. Das *push-model* sieht vor, dass der Erzeuger aktiv Daten eines Ereignisses an einen Verbraucher sendet, während im alternativen Modell der Verbraucher selbst diese Daten abfragen muss (*pull-model)*. Zentrale Komponente ist dabei der *event channel*, ein CORBA-Objekt, das die Ereignisse sowie die damit verbundenen Daten zwischen den beteiligten Parteien vermittelt. Als zentraler Vermittler ist dieses Objekt sowohl Verbraucher als auch Erzeuger von Ereignissen und ermöglicht die asynchrone Kommunikation von mehreren Verbrauchern mit mehreren Erzeugern.
Neben der generischen Kommunikation, in der die vordefinierte push- bzw. pull-Methode verwendet wird, kann auch die Kommunikation über eine definierte Schnittstelle erfolgen. Diese wird in IDL spezifiziert und muss bei allen beteiligten Partnern bekannt sein.

Was die Zuverlässigkeit sowie Leistungsfähigkeit dieses Services angeht (*qualitiy of service)*, so gibt die Spezifikation der OMG hier keine genauen Vorgaben. Es bleibt also den einzelnen Implementierungen überlassen, in welchen Bereichen sie hier Optimierungen vornehmen (von *at_most_once* Semantik bis hin zu *exactly-once*). Gleiches gilt auch für Kriterien wie Datendurchsatz oder Skalierbarkeit.

Der CORBA Event Service kann demnach für eine Implementierung genutzt werden, um ein Ereignis wie das Eintreffen eines neuen Anrufs dem Client zustellen zu können.

Transaction Service

Der *Objcet Transaction Service* (*OTS*) (siehe [OMG97]) der OMG spezifiziert die Komponenten und Schnittstellen für die Unterstützung verteilter Transaktionen und verknüpft diese mit den objektorientierten Konzepten der CORBA-Middleware. Somit wird das für stabile und hochverfügbare Anwendungen unverzichtbare Paradigma der Transaktionen auch in CORBA-Anwendungen einsetzbar. OTS ermöglicht es verschiedenen verteilten Objekten, auch über mehrere ORBs hinweg, an Transaktionen teilzunehmen, wobei sowohl flache (*flat*) als auch verschachtelte (*nested)* Transaktionen unterstützt werden.

Transaktionen werden durch einen Client mit *begin* angestoßen und können auf zwei Arten beendet werden. Durch *commit* werden alle durch die Transaktion hervorgerufenen Änderungen dauerhaft gespeichert, durch *rollback* werden sie vollständig rückgängig gemacht. OTS definiert dazu die Schnittstellen, die es CORBA-Objekten erlaubt, an verteilten Transaktionen teilzunehmen. Löst ein Client eine Transaktion durch *begin* aus, so erstellt der Transaction Service implizit einen Transaktionskontext, den er mit dem Client-Programm (seinem Thread) verknüpft. Bei jedem Aufruf von Methoden innerhalb dieser Transaktion wird dieser Kontext automatisch an alle beteiligten Server übertragen. Beendet der Client die Transaktion durch *commit*, so führt der Transaction Service in der Rolle eines Koordinators ein *two-phase commit* mit allen beteiligten Objekten durch. Dazu haben die betroffenen Objekte ihre jeweiligen Ressourcen beim Transaction Service angemeldet und stellen Dienste bereit, über die das two-phase commit Protokoll abgewickelt werden kann.

OTS ist derzeit als vielversprechendster Ansatz im Bereich der verteilten Transaktion zu sehen. Dementsprechend gut ist derzeit auch die Akzeptanz bei verschiedenen Herstellern von CORBA-Middleware, die seit längerer Zeit schon Implementierungen dieser Spezifikation anbieten (z.B. BEA, Inprise oder IONA). Allerdings sind verteilte Transaktionen immer mit einigen Herausforderungen vor allem im Bereich der Administration und Verwaltung verbunden, die nicht unberücksichtigt bleiben sollten. Da die Unterstützung von Transaktionen jedoch keine Hauptanforderung des Frameworks ist, kann man die hier spezifizierte Funktionalität als völlig ausreichend bewerten. Für Grundlagen von Transaktionssystemen sei hier auf [Gray93] und im Rahmen von CORBA auf [Orfali96b] verwiesen.

Security Service

Die Sicherheitskonzepte von CORBA sind im *Security Service* festgelegt, der ein Framework beschreibt, über das die wesentlichen Sicherheitsziele implementiert werden können. Diese Ziele beinhalten Themen wie die Identifikation und Authentifizierung von Benutzern, die Autorisierung und damit verbundene Zugriffskontrollen sowie eine sichere und zuverlässige Kommunikation auch über unsichere Kanäle. Weiterhin umfasst der Service ein Konzept für ein Sicherheits-Audit, das die Aufzeichnung aller sicherheitsrelevanten Ereignisse vorsieht und somit die Verantwortung für bestimmte Aktionen einzelnen Benutzern zuordnen kann. Weiteres Ziel ist das Prinzip *non-repudiation,* bei dem über Erstellung von geschützten Beweisen (*proof of origin, proof of receipt)* einzelner Ereignisse sichergestellt werden kann, dass ein Partner eine Aktion nachträglich nicht abstreiten kann bzw. sich auf eine nie durchgeführte Aktion berufen kann. Als letzten Bereich beinhaltet die Spezifikation ebenso die Aspekte der Administration des Sicherheitssystems. Basis des Security Service ist ein Referenzmodell, das diese Ansätze verfeinert und die dazu notwendigen Schnittstellen definiert. Dabei handelt es sich jedoch eher um ein Meta-Modell, das möglichst unabhängig von konkreten Plattformen und Technologien bleibt. Es werden demnach Implementierungsdetails nicht behandelt und ganz bewusst offen gelassen. So macht das Sicherheitsmodell zum Beispiel keine Angaben über die zu verwendenden Technologien wie etwa den Einsatz von Verschlüsselungstechniken. Aus diesem Grund ist auch der Umfang dieses Modells und damit der Spezifikation relativ hoch. Somit sei im Rahmen dieser kurzen Übersicht bzgl. weiterer Details auf die entsprechende Dokumentation verwiesen.

Wichtig zu erwähnen ist in diesem Zusammenhang noch, dass derzeit noch keine vollständige Implementierung dieser Spezifikation bekannt ist. Viele Implementierungen bieten allerdings Teilaspekte in ihren Produkten an, etwa die verschlüsselte Kommunikation über SSL.

5.2.4 Java RMI

Remote Method Invocation ist seit der Version 1.1 fester Bestandteil des JDK. Damit wird direkt das Modell verteilter Objekte in Java umgesetzt und ein ORB fast transparent für den Benutzer integriert. Der Entwickler kann also analog zu CORBA Schnittstellen von Objekten definieren und auf diese wie auf lokale Objekte zugreifen, auch wenn sie über das Netzwerk verteilt auf anderen Rechnern existieren. RMI definiert dazu die Semantik „at most once". Dadurch ist gewährleistet, dass Aufrufe, die nicht erfolgreich durchgeführt werden konnten, was durch eine entsprechende *RemoteException* angezeigt wird, maximal einmal ausgeführt wurden.

Basis für den Aufruf von Methoden eines entfernten Objekts ist eine Referenz auf dieses Objekt, die einen Endpunkt mit den Informationen (IP-Adresse, Port, Object ID) darstellt. Über diese Referenz erhält ein Client Zugriff auf die Schnittstelle dieses Objekts, wobei der Aufruf vergleichbar mit CORBA über ein lokales Stub-Objekt abgewickelt wird. Parameter werden dabei nicht wie bei lokalen Aufrufen üblich als Referenz übergeben sondern als Kopie (*pass-by-value)*, wobei hier keine Migration der Objekte stattfindet. Vielmehr stützt sich Java hier auf das Konzept der Objektserialisierung, bei dem der gesamte Status eines Objekts in einen Datenstrom geschrieben wird, der dann als Parameter über das Netz gesendet wird. Ausnahme davon sind hier natürlich die verteilten Objekte, auf die wie im Falle von CORBA über die gerade angesprochenen Referenzen zugegriffen wird.

RMI unterstützt als Bestandteil des JDK natürlich nur Java als Programmiersprache. Eine Anbindung von Komponenten in anderen Programmiersprachen ist nur indirekt über das JNI, das *java native interface* möglich. Als Plattform werden damit alle Systeme unterstützt, auf denen Java-Anwendungen innerhalb des Runtime-Environments ablaufen können.

Eine Schnittstelle eines verteilten RMI-Objekts wird als Java-Interface definiert, das von der Schnittstelle *java.rmi.Remote* abgeleitet werden muss. Damit stehen dem Entwickler alle Datentypen von Java für die Definition zur Verfügung. Bereits angesprochen wurde die Besonderheit von Methoden verteilter RMI-Objekte im Hinblick auf die Parameterübergabe, die in diesem Falle per-value erfolgt. Jede Methode eines verteilten Objektes kann eine *RemoteException* auslösen, die zur Laufzeit dann dem aufrufenden Client übermittelt wird und dort wiederum eine Exception hervorruft. Abbildung 5-7 zeigt ein Beispiel für eine derartige Schnittstelle.

```
Import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CustomerManager extends Remote
{
     Customer getCustomer(String customerId) throws RemoteException
     CustomerDList searchCustomer( String name,String firstName,String dateOfBirth,
                                   int maxNumberOfResults)   throws RemoteException
     AddressDataList getCustomerAddresses(in string customerId)
                                                             throws RemoteException
}
```

Abbildung 5-7 RMI-Schnittstelle

Die zwei Beispiele in Abbildung 5-8 sowie in Abbildung 5-9 zeigen, wie das Prinzip der Objektserialisierung in RMI genutzt werden kann. Es wird zuerst ein Objekt `Customer` definiert, das im Wesentlichen als Datencontainer für die Attribute eines Kunden dient, im Prinzip aber neben den skizzierten Get- und Set-Methoden auch weitere Funktionalität enthalten kann. Weiterhin wird ein Listenobjekt definiert, das mehrere solcher Kundenobjekte verwaltet und den Zugriff auf einzelne Elemente erlaubt. Beide Objekte werden als serialisierbar gekennzeichnet, wodurch Java versucht, diese zur Laufzeit in einen Datenstrom umzuwandeln. Da das Kundenobjekt lediglich Standardtypen verwendet und das Listenobjekt von einem Objekt Vector abstammt, das eine Serialisierung von Haus aus unterstützt, ist für die dargestellten Objekte hier keine weitere Implementierung notwendig. Ansonsten müsste eine Methode zur Verfügung gestellt werden, die selbst die Umwandlung der Objektdaten in einen Datenstrom realisiert.

```
Public class Customer implements java.io.Serializable
{
  private String _street;
  private String _zip;
  private String _city;
  private String _country;
  private String _firstName;
  private String _name;

  public String getStreet() {
      return _street;
  }

  public void setStreet(String value) {
      _street = value;
  .....
  .....
}
```

Abbildung 5-8 Serialisierbares Datenobjekt

```
public class CustomerList extends Vector implements java.io.Serializable
{
     public void addItem(Customer c) {

     }

     public Customer elementAt(index I) {
         return (Customer) getAt(i);
     }
}
```

Abbildung 5-9 Serialisierbare Liste von Datenobjekten

Diese beiden Objekte können nun für die RMI-Schnittstelle sowohl als Parameter als auch als Rückgabewert eingesetzt werden. Das Laufzeitsystem überträgt dann jedes Mal alle Informationen, die den aktuellen Status der einzelnen Objekte beschreiben, so dass hier entsprechend identische Kopien beim Client bzw. Server verfügbar sind. Voraussetzung dafür ist allerdings, dass die Implementierung der Klassen in den entsprechenden class-Dateien auf beiden Seiten existiert bzw. dynamisch geladen werden kann.

5.2.4.1 Weitere Eigenschaften

Die Unterstützung, die RMI zur Entwicklung ausreichend skalierbarer Systeme anbietet, ist relativ gering. Einziger Mechanismus dazu ist die Verwendung von Threads, die in Java grundsätzlich sehr gut unterstützt werden. So können Serveranwendungen selbst eine hohe Anzahl von Anfragen an ein Serverobjekt in verschiedenen Threads abhandeln. Die RMI-Spezifikation sieht einen derartigen Mechanismus zwar prinzipiell auch vor, schreibt ihn aber nicht zwingend vor. Auch Ansätze wie eine Lastverteilung müssen durch die Anwendung selbst oder aber durch eine geeignete RMI-Implementierung umgesetzt werden. Die Referenzimplementierung der Firma SUN bietet solche Funktionen nicht an. Auch die Namensauflösung über die *rmiregistry* bietet im Grunde keine Funktionalität, die für eine hohe Skalierbarkeit ausgenutzt werden könnte. Somit muss festgehalten werden, dass in diesem Zusammenhang RMI keine ausreichende Funktionalität mit sich bringt und die Namensauflösung fast ausschließlich der Anwendungsentwicklung überlässt. Auch bietet Java RMI keine Konzepte an, die die Übermittlung asynchroner Ereignisse unterstützen.

Basis für die Implementierung von Sicherheitsmechanismen in Java-Anwendungen ist die *Java Security Architecture*, die für die Version 1.2 des JDK gründlich überarbeitet wurde. Hier werden die Schnittstellen definiert, über die Sicherheitsregeln und Zugriffsrechte eingestellt werden können, die zur Laufzeit festlegen, auf welche Ressourcen ein Programm Zugriff haben darf. Darunter fällt zum Beispiel der Zugriff auf das Netzwerk über Sockets oder der Zugriff auf Dateien. Die Sicherheitskonzepte umfassen dabei auch das für Java typische, dynamische Laden von Programmcode, das unter anderem bei der Verwendung von Applets zu Sicherheitsrisiken führen kann. Die Authentizität eines Programmcodes wird dabei durch digitale Zertifikate nach dem X.509-Standard sichergestellt.

Neben diesen Diensten existiert noch die *Java Cryptography Architecture,* die ein Framework darstellt, um Verschlüsselungsverfahren nutzen bzw. implementieren zu können. Damit bietet die Entwicklungsumgebung alle notwendigen Basismechanismen, um die für ein System notwendigen Sicherheitskonzepte umzusetzen. Diese Mechanismen können somit auch für ein verteiltes System auf Basis von RMI implementiert werden. Allerdings stellt RMI selbst keine direkte Unterstützung derartiger Sicherheitsmechanismen zur Verfügung. Es sind jedoch aktuell auch hier Arbeiten im Gange, die eine Erweiterung der RMI-Spezifikation hinsichtlich genau dieser Ansätze zum Ziel haben.

Abschließend ist noch festzuhalten, dass RMI selbst keine Transaktionsunterstützung anbietet.

5.2.5 Enterprise Java Beans

EJB ist der aktuelle Komponentenansatz der Firma SUN im Rahmen des Enterprise-Pakets von Java der derzeit in der Version 1.1 (siehe [SUN99]) verbreitet ist. Es beinhaltet demnach die wesentlichen Dienste, die für die Entwicklung einer verteilten Enterprise-Anwendung benötigt werden. Laut Spezifikation hat die Architektur zum Ziel, einen Standard für verteilte, objektorientierte und serverbasierte Komponenten zu schaffen, die in Java erstellt werden. Ein weiteres Ziel ist es, nicht nur die Anwendungsentwicklung (*development)* zu vereinfachen, sondern auch die Auslieferung (*deployment)* und den Betrieb (*runtime)* solcher Komponenten zu vereinheitlichen.

Damit Serverkomponenten gemäß dem Ansatz WORA (*write once run anywhere*) in unterschiedlichen Umgebungen ohne Eingriff in die Implementierung verwendet werden können, laufen sie nicht direkt auf einem System, sondern werden innerhalb eines sog. Containers ausgeführt. Diese Container stellen eine standardisierte Laufzeitumgebung mit einer Reihe von Diensten zur Verfügung, die eine Komponente nutzen kann. Zusammengefasst werden die Container in einem EJB-Server, der die gesamte Laufzeitumgebung zur Verfügung stellen muss. Das Zusammenspiel zwischen Container und Bean regeln dabei die sog. *Contracts* der Spezifikation. Elementare Aufgabe dieser Container ist die Verwaltung des Lebenszyklus (*life cycle*) der Enterprise Java Beans. Er stellt dazu eine Factory-Schnittstelle, das sog. Home-Interface (*EJBHome*) zur Verfügung, über das ein Client Zugriff auf eine gewünschte Serverkomponente erhält. Weiterhin generiert ein Container aus der Schnittstelle der Komponente die konkrete Schnittstelle, auf die der Client über das Netzwerk zugreifen kann. Die EJB-Spezifikation definiert für diese Serverkomponenten zwei grundlegende Varianten, nämlich *session beans* und *entity beans*. Dabei repräsentieren letztere persistente Informationen, in der Regel einer Datenbank, im weiteren Sinne jedoch auch einer anderen Ressource. Diese Informationen bleiben also auch über den Lebenszyklus einer einzelnen Bean hinaus bestehen. Die konkrete Speicherung der Informationen übernehmen solche Komponenten entweder selbst (*bean managed persistency)* oder sie wird automatisch durch den Container übernommen (*container managed persitency)*. Session beans sind dagegen dadurch charakterisiert, dass sie einer konkreten Clientanwendung zugeordnet werden und die für diese notwendige Geschäftslogik implementieren. Sie haben in der Regel einen Status, der allerdings nicht persistent ist, und ihr Lebenszyklus richtet sich nach dem des Clients.

Da die Spezifikation nicht nur die Aspekte der Entwicklung sondern auch der Administration bzw. des Betriebs von Anwendungen anspricht, legt sie sechs Rollen fest, die in Abbildung 5-10 neben den wesentlichen Elementen der Architektur nochmals dargestellt sind. Die Schnittstelle zwischen den Rollen Application Assembler bzw. Deployer und den Entwicklern von Enterprise Java Beans stellt der Deployment Descriptor dar. In dieser Datei werden im XML-Format alle Informationen zusammengestellt, die beschreiben, wie diese Komponenten zur Laufzeit zu behandeln sind. Diese Information enthält unter anderem die Struktur der Komponente, ihre Methoden, ihren Typ (session oder entity bean) sowie diverse Attribute, die das Verhalten in Bezug auf die Speicherung persistenter Daten oder aber Transaktionen spezifizieren.

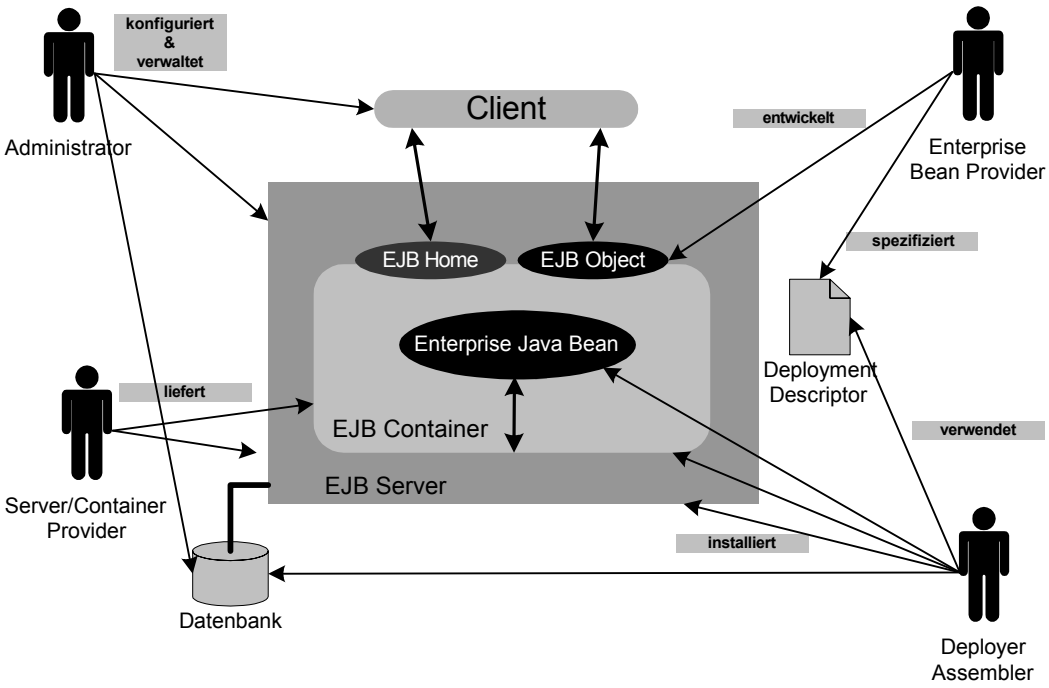


Abbildung 5-10 EJB-Komponenten und Rollen

Die Schnittstellen eines Enterprise Java Beans werden als Java-Interface spezifiziert. Der Entwickler hat also hier die gleichen Möglichkeiten wie im Falle von RMI, die bereits behandelt wurden. Die Schnittstellen müssen dabei von vordefinierten Interfaces (EJBHome, Remote etc.) abgeleitet werden. Auch was die Kommunikation und Interaktion verteilter EJB-Komponenten angeht, setzt die Spezifikation hier auf RMI auf. Sie legt jedoch weiter fest, dass für diese Schnittstellen nur Typen verwendet werden, die konform zu RMI/IIOP sind. Somit ist gewährleistet, dass Hersteller von Containern das RMI/IIOP-Protokoll implementieren und somit in heterogenen Umgebungen eine Interoperabilität gewährleisten können. Die Spezifikation 1.1 schreibt allerdings dies nicht als zwingendes Protokoll vor.

Als Bestandteil des Java-Pakets unterstützt EJB zur Entwicklung von EJB-Komponenten nur Java als Programmiersprache. Allerdings ermöglicht der Einsatz von CORBA als Basis-Middleware prinzipiell einen Zugriff auch auf andere CORBA-Objekte, die dann wiederum in anderen Programmiersprachen implementiert werden können (etwa C oder C++). Es handelt sich bei diesen Objekten dann jedoch nicht mehr um Enterprise Java Beans, die innerhalb eines Application-Servers verwaltet werden. Als Plattform kommen auch hier wie bei RMI alle Systeme in Frage, die ein Java-Laufzeitsystem anbieten. Allerdings benötigen EJB-Komponenten darüber hinaus einen Application-Server, der die notwendigen Container und die zugehörigen Dienste und Werkzeuge beinhaltet. Die Verfügbarkeit eines entsprechenden Produkts für die gewünschte Plattform ist deshalb wichtige Voraussetzung. Einen Vergleich von CORBA und EJB findet man in [Orfali98].

5.2.5.1 Weitere Eigenschaften

Die EJB-Spezifikation enthält keinen Dienst, über den asynchron auftretende Ereignisse von einem Server zu einem oder mehreren Clients übermittelt werden können. EJB spezifiziert selbst auch keine konkreten Mechanismen zur Unterstützung der Skalierbarkeit sondern überlässt dies vielmehr der Implementierung spezieller Application Server. Diese haben die Verantwortung, durch geeignetes Prozess- und Thread-Management sowie durch eine ausreichende Lastverteilung die Leistungsfähigkeit des gesamten Systems auch bei steigender Anzahl von Clients zu gewährleisten. Gewisse Vorgaben zum Beispiel bei der Verwaltung einer hohen Anzahl von EJB-Instanzen werden durch die Spezifikation allerdings schon gegeben, da

diese nämlich für den Lebenszyklus einiger EJB-Typen die Verwaltung in einem Pool vorsieht. Ideal für eine effiziente Verwaltung von vielen Instanzen sind dabei die zustandslosen Session-Beans, deren Instanzen in zwei Zuständen existieren: *„does not exist"* sowie *„method-ready pool"*. Sobald ein Container eine Instanz benötigt, erzeugt er sie, setzt den richtigen Kontext und fügt sie seinem Pool hinzu. Aufrufe des Clients werden dann an eine beliebige Instanz des Pools weitergeleitet. Die Anzahl der vorgehaltenen Beans innerhalb seines Pools kann der Container dabei selbst steuern. Ebenfalls in einem Pool verwaltet werden die Entity-Beans, wobei sie dort noch keine Identität besitzen und mit einem konkreten Datensatz identifiziert werden. Ein Client erhält also auch hier bei Bedarf eine beliebige Instanz, über die er dann einen bestimmten Datensatz ermittelt kann. Erst dann ist diese Instanz im Zustand *Ready*. Für zustandsbehaftete Session-Beans ist keine Verwaltung in einem Pool vorgesehen und die Verwaltung von Bean-Instanzen in einem Pool wird durch die Spezifikation auch nicht zwingend vorgeschrieben.

Das Sicherheitskonzept der EJB-Spezifikation sieht vor, den Anwendungsentwickler von dieser Problematik weitgehend zu befreien. Die Verantwortung dafür wird auf den Hersteller des EJB-Containers (Container Provider) sowie auf den Deployer bzw. Systemadministrator verlagert. Diese Rollen sorgen dafür, dass einerseits die notwendige Infrastruktur zur Verfügung steht, in der die Sicherheitsmechanismen implementiert sind, sowie andererseits konkrete Sicherheitsregeln für ein Anwendungssystem definiert werden. Sicherheitsmechanismen werden also nicht durch den Entwickler sondern erst später zur Laufzeit definiert, so dass EJBeans auch auf unterschiedlichen Servern mit unterschiedlichen Sicherheitsmechanismen lauffähig bleiben. Der Application Assembler definiert dazu im Deployment Descriptor sog. *security roles* sowie *method permissions* für jede dieser Rollen. Die Rollen sind als logische Rollen zu verstehen, da die konkrete Zielumgebung ja noch nicht bekannt ist. Der Deployer setzt dann diese Vorgaben gemäß dem operativen Betrieb um. Dazu weist er Teilnehmern des verteilten Systems (etwa einzelne Benutzer oder Benutzergruppen) einzelne Rollen zu. Ein Client kann zur Laufzeit eine Methode nur dann aufrufen, wenn der Teilnehmer, für den er diese Aktion ausführt, mindestens eine Rolle hat, die diese Methode gemäß den *method permissions* auch aufrufen darf.

Für die Entwicklung transaktionsorientierter Systeme unterstützt EJB verteilte Transaktionen, so dass Zugriffe auf mehrere, im Netzwerk verteilte Ressourcen in einer Transaktion geklammert werden können. EJB sieht dafür die Integration eines Transaktionsservices vor und definiert dazu als grundlegende Schnittstelle JTA (*Java Transaction API)*, über die beteiligte Instanzen die betroffenen Transaktionen steuern können. Darüber hinaus stützt sich EJB auf die Spezifikation JTS (*Java Transaction Service*), die Java-Version der Spezifikation OTS 1.1 (*Object Transaction Service)* der OMG. JTS enthält damit alle notwendigen Definitionen, um die erforderliche Infrastruktur für einen Transaktionsdienst innerhalb eines EJB Application Servers zu implementieren, der dann JTA unterstützt. Allerdings legt EJB nicht fest, dass ein Container JTS einsetzen muss. Das Transaktionsmodell, das EJB dabei zugrunde legt, unterstützt im Gegensatz zu CORBA/OTS keine geschachtelten, sondern nur flache Transaktionen. Die eingangs beschriebene Rollenverteilung bestimmt auch die Steuerung sowie die Verwaltung und Umsetzung verteilter Transaktionen im Falle von EJB. Auch hier wird der Entwickler einer EJB-Komponente vollständig von der technischen Umsetzung des Transaktionsdienstes befreit, die vom Container übernommen wird. Er ist lediglich für die Steuerung der Transaktionen verantwortlich, die je nach Anforderung in mehreren Varianten erfolgen kann. So können die Transaktionsgrenzen entweder vom Client der Anwendung selbst (*client-managed demarcation)*, durch die Bean-Komponente (*bean-managed demarcation)* oder aber durch den *Container (container-managed demarcation)* festgelegt werden. Entscheidend beeinflusst wird das Verhalten von Beans im Hinblick auf Transaktionen weiterhin durch eine Reihe von Attributen, die im Deployment-Deskriptor pro Methode einer Komponente definiert werden.

Die leicht überarbeite Version 2.0 der Spezifikation, die Verbesserungen vor allem im Hinblick auf die Persistenzmechanismen sowie die Einführung nachrichtenorientierter Komponenten (*message-driven beans)* enthält, findet man in [SUN01].

5.2.5.2 Fazit

Die EJB-Spezifikation beinhaltet als Weiterentwicklung der Java-Technologie in Richtung *Enterprise Computing* alle wesentlichen Aspekte größerer, verteilter Systeme, wobei die Entwicklung hier in Richtung

Komponententechnologie geht. Insofern ist ein direkter Vergleich mit CORBA auch nicht ganz einfach, da der Ansatz von EJB wesentlich allgemeiner ist. So sieht die Spezifikation ja auch eine Implementierung auf Basis von CORBA als Alternative zu RMI vor. Ein wesentlicher Unterschied besteht jedoch in der Zahl der unterstützten Programmiersprachen, die sich in diesem Falle auf Java beschränkt. Gerade im Hinblick auf die Anbindung bestehender Systeme fehlt es dadurch an Möglichkeiten. Auch die Anzahl der Dienste, die als Erweiterungen der grundsätzlichen Middleware-Funktionalität angeboten werden, ist im Vergleich zu CORBA geringer. Deshalb ist gerade im Hinblick auf komplexere und umfangreichere Systeme in jedem Falle eine Implementierung von EJB auf Basis eines der verfügbaren und in der Regel sehr ausgereiften CORBA-Produkten vorzuziehen. Damit erhält man derzeit eine der aussichtsreichsten Plattformen für komponentenorientierte, verteilte Systeme. Dadurch ist prinzipiell auch die Kompatibilität zu reinen CORBA-Systemen und der Einsatz anderer Programmiersprachen als Java, sei es aus Gründen der Performance oder zur Anbindung von Legacy-Systemen, weiterhin möglich.

5.2.6 COM+

COM+ ist das neueste Komponentenmodell von Microsoft, das seit der Einführung von Windows 2000 fester Bestandteil des Betriebssystems ist. Hinter diesem Begriff verbirgt sich keine vollständig neue Technologie sondern die Fortsetzung und Zusammenführung der bestehenden Komponenten-Technologien wie COM, DCOM, des Transaktionsservers MTS sowie des Message Queue Servers MSQS.

COM+-Komponenten setzen auf den Technologien von COM bzw. DCOM und damit auf COM-Objekten auf, die allerdings weniger als Objekte im Sinne der OO-Technologien sondern vielmehr als im Binärformat vorliegende Softwarekomponenten zu verstehen sind. Die Interaktion zwischen diesen Komponenten kann dabei innerhalb eines Prozesses, innerhalb eines Betriebssystems oder aber auch verteilt innerhalb eines Netzwerkes stattfinden. In letzterem Fall stützt sich DCOM auf ein eigenes RPC-Protokoll (*remote produre call)*, das im Wesentlichen auf der Spezifikation von DCE basiert, aber an einigen Stellen erweitert wurde.

Da COM das Binärformat dieser Komponenten bzw. ihrer Schnittstellen definiert, können diese in unterschiedlichen Programmiersprachen entwickelt werden. Die Entwicklungsumgebung von Microsoft bietet derzeit COM-Unterstützung für ihre Sprachvarianten Visual-C++ (C und C++), Visual-J++ (Java) und Visual Basic an. Inwieweit die aktuellen Ankündigungen von Microsoft, anstatt Java nun eine eigene Sprache C-# zu unterstützen, die Kompatibilität von COM bzw. COM+ und Java einschränken wird, kann zum aktuellen Zeitpunkt nur schwer eingeschätzt werden. Aus dem Binärformat ergibt sich allerdings auch, dass COM-Objekte sehr abhängig von einer Plattform sind. COM+ unterstützt als Produkt von Microsoft und Bestandteil des Betriebssystems Windows 2000 somit in erster Linie die Windows-Plattform. Die bereits ältere DCOM-Lösung wurde zwischenzeitlich einmal auf unterschiedliche UNIX-Plattformen wie etwa OS/390 portiert, die Strategie von COM+ dahingehend ist derzeit allerdings nicht bekannt. Insgesamt ist jedoch festzuhalten, dass mit COM+ nicht die Sprachunabhängigkeit und Plattformunabhängigkeit von CORBA erreicht werden kann.

Wesentliches Kriterium auch eines COM-Objekts ist seine Schnittstelle, in der alle Methoden inklusive Übergabe- bzw. Rückgabeparameter definiert werden. Konkrete Instanzen solcher Schnittstellen sind dann letztendlich Zeiger auf Funktionen, die in einem Array, der sog. *vtable*, verwaltet werden. Es ist demnach durchaus möglich, solche Instanzen auch manuell etwa in C oder C++ zu erstellen, obwohl in der Praxis auch hier eine Beschreibungssprache eingesetzt wird. Im Unterschied zu CORBA oder Java RMI unterstützen COM-Objekte keine Mehrfachvererbung von Schnittstellen. Dafür können diese jedoch mehrere Schnittstellen besitzen. Da die Kommunikation sowie der Transport von Daten in COM auf dem RPC-Mechanismus basiert, werden die Schnittstellen der Komponenten in einer dazu konformen IDL-Datei spezifiziert. Microsoft liefert dazu eine eigene Spezifikation *MIDL* sowie einen eigenen Compiler, den *MIDL compiler*, der aus einer derartigen Datei die notwendigen Header-Dateien und Stubs generiert. MIDL kann damit auch dazu genutzt werden, die Schnittstellen von COM-Objekten zu definieren. Eine kurze Betrachtung der Syntax dieser Beschreibungssprache zeigt, dass hier durchaus Ähnlichkeiten zu CORBA-IDL vorhanden sind, allerdings die Version von Microsoft noch enger an C bzw. C++ angelehnt ist.

```
[
    uuid(12345678-1234-1234-1234-123456789ABC),
]
interface CustomerManager : Iunknown

{
    #define MAX_RESULTS 100

    typedef struct {
         char * name;
         char * firstname;
    } CustomerData;

    typedef CustomerData CustomerDataList[MAX_RESULTS];

    CustomerDataList getCustomer([in] char *name);

}
```

Abbildung 5-11 IDL-Schnittstelle von COM/DCOM

Wie das Beispiel in Abbildung 5-11 zeigt, besteht die Beschreibung der Schnittstelle aus zwei Bereichen, dem *header*, der die Attribute und den Namen der Schnittstelle angibt sowie dem *body*, der die Deklaration aller Prozeduren beinhaltet, die die Schnittstelle bereitstellt sowie dazu notwendige Typdefinitionen und Konstanten. Wie im Beispiel dargestellt, kann die Schnittstelle von einer bereits bestehenden abgeleitet werden, deren Eigenschaften sie dann entsprechend erbt. Die Schnittstellen von COM-Objekten müssen dabei in jedem Fall von *Iunknown* abgeleitet werden. Die Parameter von Prozeduren werden wie im Falle von CORBA als [in], [out] oder [in,out] deklariert, je nachdem, ob Informationen vom Client zum Server, umgekehrt oder in beide Richtungen fließen.

Die gesamte Spezifikation von MIDL ist erwartungsgemäß umfangreicher, als es dieser kurze Überblick wiedergeben kann, man kann jedoch festhalten, dass hiermit ausreichend funktionale Schnittstellen von Objekten definiert werden können, wenn auch die Spezifikation gegenüber CORBA und Java RMI etwas umständlicher erscheint.

5.2.6.1 Weitere Eigenschaften

Für die Übertragung asynchroner Ereignisse zwischen einzelnen Komponenten führt COM+ ein Ereignissystem ein, das die Mechanismen *publish* sowie *subscribe* unterstützt. Dabei handelt es sich um sog. *loosely-coupled events (LCE)*, die nicht voraussetzen, dass Sender und Empfänger von Ereignissen direkt miteinander verbunden sind und zur gleichen Zeit ablaufen. Das System speichert Ereignisse von unterschiedlichen Erzeugern (*publisher*) dazu in einem COM+-Katalog. Die Verbindung zwischen Erzeuger und Empfänger eines Ereignisses erfolgt auch bei COM+ über ein besonderes Objekt der Klasse *EventClass,* das Methoden enthält, über die ein Erzeuger ein Ereignis auslösen kann. Ein Empfänger dagegen muss die angegebene Schnittstelle eines solchen Objekts implementieren, damit ihm Ereignisse zugestellt werden können. Wird über ein konkretes Objekt ein Ereignis ausgelöst, indem ein Erzeuger eine bestimmte Methode des Event-Objekts aufruft, so sucht das System alle möglichen Empfänger, die für dieses Objekt bzw. genau diese Methode als Empfänger eingetragen sind (*subscription)*, und ruft dort die entsprechende Methode auf. Empfänger erzeugen derartige Abonnements (*subscriptions),* um die Zustellung von bestimmten Ereignissen zu verwalten. Sie enthalten unter anderem Informationen darüber, wer der Empfänger ist und wo er sich befindet, über die einzelnen Methoden, die als Ereignisse übermittelt werden sollen sowie über den Erzeuger der Ereignisse.

Für die Realisierung ausreichend skalierbarer Systeme bietet COM+ unter anderem die Dienste *Object Pooling* sowie *Load Balancing* an. Der erste Dienst verwaltet mehrere Komponenten automatisch innerhalb eines Pools, in dem sie auf Anfragen von Clients warten. Ein Pool besteht dabei immer aus einer Art von Komponenten, wobei eine Minimal- bzw. Maximalanzahl von Instanzen angegeben werden kann. Beim ersten Start des Pools wird die minimale Anzahl von Komponenten erzeugt und alle Anfragen an Methoden dieser Instanzen laufen nun über den Pool-Manager. Dieser leitet Anfragen an gerade nicht benutzte Komponenten weiter und versucht somit, bestehende Instanzen soweit wie möglich wieder zu verwenden. Ist keine freie Komponente verfügbar, so erzeugt der Manager eine neue Instanz, bis der maximale Wert

erreicht ist. Dadurch werden eventuell längere Anlaufphasen von Komponenten vermieden und benötigte Ressourcen wie zum Beispiel Datenbankverbindungen besser kontrolliert.

Um den zweiten Dienst, die automatische Lastverteilung (*Component Load Balancing CLB),* nutzen zu können muss ein Rechner im lokalen Netzwerk als Server (*CLBS*) geeignet konfiguriert und dort alle Rechner eingetragen werden, an die Methodenaufrufe weitergeleitet werden können. Dieser Server überwacht daraufhin periodisch (etwa 20 mal pro Sekunde) die Last der eingetragenen Maschinen. Dazu werden über sog. Interceptors bei jedem Methodenaufruf entsprechende Zeitmessungen gespeichert, auf die der Server zugreifen kann. Voraussetzung dafür ist allerdings, dass die betroffenen COM+-Objekte dafür konfiguriert sind, denn eine Überwachung aller Objekte wäre zu aufwändig. Aus Sicht der Clients werden dann Objekte nur mehr über diesen CLB-Server erzeugt und verwendet. Der Server ist dann für das Routing der eingehenden Aufrufe an den jeweils günstigsten Rechner verantwortlich.

Als letzter Mechanismus zur Steigerung der Leistungsfähigkeit eines Systems ist abschließend noch die Verwaltung von Threads und damit die Unterstützung von Nebenläufigkeit zu erwähnen. COM+ hat dazu ein Modell, das sich auf die Verwendung sog. *apartments* stützt. Dies sind Sammlungen von Objekten, wobei Aufrufe innerhalb eines Apartments direkt erfolgen können, zwischen unterschiedlichen Apartments nur indirekt über Proxies und Stubs. Ein Prozess wird demnach in mehrere Apartments aufgeteilt, die wiederum unterschiedliches Verhalten bzgl. des Einsatzes von Threads besitzen. Objekte in einem *single-threaded apartment (STA)* werden genau innerhalb eines Threads und ihre Methoden demnach seriell ausgeführt. Objekte in einem *multithreaded apartment* (MTA) können dagegen in mehreren Threads und ihre Methoden demnach auch nebenläufig ausgeführt werden. Darüber hinaus existiert noch eine Variante, die sog. *neutral apartments*, die beide Varianten unterstützen und deren Verhalten vom aufrufenden Thread bestimmt wird. Für die Synchronisation, die bei Verwendung der Variante MTA notwendig wird, können in COM+ sog. *activities* genutzt werden, die einen oder mehrere Kontexte zusammenfassen und einen exklusiven Zugriff darauf gewährleisten.

Das Sicherheitsmodell von COM+ basiert auf einem Rollenmodell, über das Zugriffsrechte vergeben bzw. abgeprüft werden können. Eine Rolle ist dabei ein symbolischer Name, der einen oder mehrere Benutzer definiert. Die Zugriffsrechte solcher Rollen werden dann in der Regel administrativ festgelegt indem bestimmt wird, auf welche Ressourcen einer Applikation ein Mitglied einer solchen Rolle zugreifen darf. Dies können die gesamte Anwendung, einzelne Komponenten, ausgewählte Schnittstellen oder aber nur einzelne Methoden sein. Reicht es nicht aus, diese Zugriffsrechte administrativ zu steuern, so kann eine Anwendung dies auch selbst über eine API zur Laufzeit übernehmen.

Umgesetzt wird dieser Mechanismus über die Sicherheitspakete von COM und Windows 2000, die nun eine Reihe von Protokollen unterstützen. Dazu gehört einmal das in DCOM integrierte Protokoll NTLMSSP (*NT LAN Manager Security Support Provider).* Weiterhin wird das Kerberos Protokoll in der Version 5 sowie über das Paket *Schannel* das Protokoll SSL unterstützt. SSL führt Verschlüsselung und Authentifizierung über asymmetrische Kryptoverfahren durch und ist derzeit das Standardprotokoll im Internet-Umfeld.

Die Unterstützung von verteilten Transaktionen basiert auf MTS, dem Transaktionsmonitor von Microsoft, dessen Funktionalität nun in COM+ integriert wird. COM+ nutzt dazu die Dienste des *Microsoft Distributed Transaction Coordinator* (MS DTC), der Aufgaben wie die Generierung einer Transaktions-ID sowie die Kommunikation und Koordination der Ressource-Manager übernimmt. Die Architektur von MS DTC sieht für jeden Rechner einen lokalen Transaktionskoordinator vor, wobei einer dieser Instanzen die übergeordnete Koordination und damit die Abwicklung des two-phase commit Protokolls übernimmt. Der Zugriff auf Ressourcen erfolgt auch hier über den Ressource-Manager. MS DTC unterstützt primär das Transaktionsprotokoll von OLE, über das Datenbanken wie etwa SQL-Server angesprochen werden können, sowie das XA-Protokoll der X/Open, über das Ressourcen wie Oracle, DB2, Sybase oder Ingres verwendet werden können. COM+ unterstützt dabei keine verschachtelten Transaktionen.

Das Verhalten von Komponenten in Bezug auf verteilte Transaktionen wird im Rahmen der Konfiguration und Administration anhand von Attributen gesteuert, die für jede Komponente definiert werden können, ist also vom Ansatz her vergleichbar mit den Konzepten von EJB.

5.2.6.2 Fazit

Auch COM+ vereint vergleichbar mit EJB die wesentlichen Ansätze komponentenorientierter, verteilter Systeme in einer gemeinsamen Spezifikation und integriert dabei im Wesentlichen bestehende Technologien wie DCOM oder MTS. Die dadurch für die Entwicklung zur Verfügung stehenden Basisdienste sind gegenüber denen einer Kombination von CORBA und EJB allerdings etwas geringer. Wichtigstes Merkmal ist jedoch die enge Verbindung zur Plattform Windows 2000. Obwohl eine Anbindung von Fremdsystemen durch weitere Produkte möglich ist, kann man davon ausgehen, dass eine verteilte Anwendung auf Basis von COM+ auf Windows-Systemen laufen muss. Um die neuesten Funktionen zu nutzen, sollten die wesentlichsten Komponenten sogar unter Windows 2000 implementiert werden. Diese enge Integration der Middleware mit dem Betriebssystem zieht sich bis zur Verwendung der Programmiersprachen durch, da die Entwicklungsumgebungen von Microsoft mit ihren Produkten Visual-C++ oder Visual-Basic die Mechanismen von COM+ am besten unterstützen.

5.2.7 Microsoft .NET und SUN ONE

Was das Thema Internetanwendungen angeht präsentieren derzeit die Firmen Microsoft und SUN mit ihren Initiativen .NET und ONE (*Open Network Environment)* Frameworks, die in Zukunft die Basis auch für unternehmensweit verteilte Applikationen darstellen sollen. Grund genug, beide Ansätze an dieser Stelle kurz zu beleuchten.

Eine der Kernideen beider Ansätze ist das Konzept der Web Services. Damit sind Dienste gemeint, die prinzipiell von jedem Gerät aus (PC, Workstation, PDA, Handy etc.) über das Internet genutzt werden können. Basistechnologie ist dabei neben HTTP vor allem XML, die als Basis für den Datenaustausch genutzt wird. Darauf aufbauend definieren neue Protokolle wie SOAP (*simple object access protocol)* die Kommunikation zwischen Anwendungen über das Internet, ähnlich dem RPC-Mechanismus. Besonders Microsoft versucht mit .NET eine grundlegend neue Basis für die Entwicklung verteilter Anwendungen zu schaffen und damit nicht zuletzt Boden gegenüber der inzwischen sehr verbreiteten Entwicklungsumgebung von Java gut zu machen.

Kern der .NET-Architektur ist ein vom Prinzip her sprachunabhängiges Laufzeitsystem (*Common Language Runtime*), das Byte-Code in einem internen Format (*internal language)* ausführen kann. Programme, die in unterschiedlichen Sprachen geschrieben werden, werden in dieses gemeinsame Format übersetzt und können so nahtlos miteinander interagieren. Dies wird insbesondere auch dadurch unterstützt, dass mit CTS (*Common Type System)* ein von einzelnen Programmiersprachen unabhängiges Konzept für Datentypen entworfen wurde. Neben bereits bestehenden Sprachen wie C++ oder Visual Basic, die durch geeignete Compiler ebenfalls kompatibel mit dem CLR-System werden sollen, hat Microsoft mit C# eine neue Sprache entwickelt, die eng mit dem Kern der .NET-Architektur verknüpft ist (vgl. 5.3.4).

Auf diese Basis setzen nun eine Reihe von Klassenbibliotheken auf, die dem Entwickler alle notwendigen Basisdienste etwa für Netzwerkkommunikation oder Ein-/Ausgabe zur Verfügung stellen. In diesem Bereich ist auch ADO+ (bzw. ADO.NET) angesiedelt, eine Erweiterung der bestehenden Technologie ADO (*ActiveX Data Objects*). Die Erweiterungen beziehen sich dabei vor allem auf die Bereiche XML und SOAP, die nun verstärkt für den Datenaustausch genutzt werden. Für die Realisierung graphischer Benutzerschnittstellen stehen als oberste Schicht des .NET-Frameworks folgende Bibliotheken zur Verfügung:

- ASP.NET (bzw. ASP+)
  Eine Erweiterung der bestehenden Technologie ASP (Active Server Pages), mit der dynamische WWW-Anwendungen entwickelt werden können. Neu dabei ist, dass in Zukunft jede Sprache unterstützt wird, die in das interne Format des Runtime-Systems übersetzt werden kann, aktuell also auch C#.

- Web Forms
  HTML-Elemente wie Textboxen oder Schaltflächen zur individuellen Gestaltung von WWW-Seiten.

- Web Services
  Ein Modell zur Implementierung von Web-Diensten.
- Win Forms
  Klassenbibliothek zur Implementierung herkömmlicher, graphischer Benutzeroberflächen. Diese Bibliothek löst damit die *Microsoft Foundation Classes* (MCF) sowie die ActiveX-Komponenten ab.

SUN Microsystems setzt mit ONE im Wesentlichen auf bestehende Technologien des J2EE-Frameworks. So ist hier Java weiterhin die unterstützte Programmiersprache, deren Laufzeitsystem allerdings plattformübergreifend verfügbar ist. Um die Idee der neuartigen, auf XML und Web-Services basierenden Anwendungen realisieren zu können, setzt ONE auf die bewährten Funktionen der Java-API, auf dynamische WWW-Anwendungen mit JSP sowie die bereits vorhandenen XML-Bibliotheken. Ein wesentlicher Unterschied zu .NET ist dabei, dass SUN mit Java auf nur eine Programmiersprache setzt. Die von Microsoft angestrebte Unterstützung verschiedenster Sprachen ist im Falle von J2EE nur über die Verwendung von CORBA im Rahmen der EJB-Spezifikation möglich.

Während Microsoft also versucht, neue Ansätze wie XML und SOAP möglichst eng in ihr .NET-Framework zu integrieren und dies auch als Alleinstellungsmerkmal gegenüber anderen Ansätzen wie ONE darzustellen, setzt SUN auf ihre bewährten und bereits auf vielen Plattformen verbreiteten Technologien und überlässt die Umsetzung mehr dem Entwickler bzw. individuellen Erweiterungen.

Bezogen auf die in dieser Arbeit gestellten Anforderungen sind Aspekte wie Web Services eher als Ausblick bzgl. zukünftiger Erweiterungen zu betrachten. Somit sind sowohl .NET als auch ONE in diesem Zusammenhang interessante Alternativen, die aber für die konkrete Implementierung noch keinen echten Mehrwert bieten. .NET ist außerdem derzeit wegen der Einschränkung auf die Windows-Plattform bzw. des noch frühen Entwicklungsstands des Frameworks noch keine echte Alternative. Somit werden beide Varianten auch nicht für den Vergleich zu den anderen Technologien herangezogen .

5.2.8 Zusammenfassung

Inhalt dieses Abschnittes ist, eine Übersicht der derzeit wichtigsten Middleware-Ansätze zu geben mit dem Ziel, einen kurzen Vergleich hinsichtlich des Einsatzes für ein Framework in einem Call Center durchführen zu können. Die dafür wesentlichsten Kriterien sind in Tabelle 5-10 nochmals zusammengefasst. Dabei ist zu berücksichtigen, dass CORBA und RMI eher grundlegende Middleware-Technologien darstellen, während die Ansätze von EJB und COM+ darüber hinaus viele Aspekte der Komponententechnologie behandeln.

Betrachtet man die Funktionalität, die die verschiedenen Ansätze zur Entwicklung verteilter Systeme bieten, so steht hier CORBA derzeit immer noch an erster Stelle. Die zahlreich vorhandenen Services decken das gesamte Spektrum der Anforderungen auch komplexer und großer Systeme ab. Dazu kommt, dass CORBA die derzeit wohl ausgereifteste Spezifikation einer Middleware darstellt und auch die verfügbaren Implementierungen seit geraumer Zeit am Markt und deshalb entsprechend stabil sind.

Kategorie		CORBA	RMI	EJB	COM+
A	Transparente Kommunikation	Ja	Ja	Ja	Ja
A	Schnittstellendefinition	IDL	Java-Interface	Java-Interface	MIDL
A	Naming Service	Ja	Ja, aber sehr einfach	Ja	Ja
B	Unterstützte Plattformen	Alle gängigen Systeme	Alle gängigen Systeme	Alle gängigen Systeme	Windows Windows 2000
B	Unterstützte Sprachen	C, C++, Java, Cobol	Java	Java	C++, C, Visual Basic
B	Ereignisse	Ja	Nein	Nein	Ja
B	Skalierbarkeit	Hoch	Niedrig	Hoch	Hoch
C	Sicherheit und Zuverlässigkeit	Hoch	Niedrig	Hoch	Hoch
C	Transaktionsunterstützung	Ja	Nein	Ja	Ja

Tabelle 5-10 Vergleich der Middleware-Produkte

RMI ist im Vergleich dazu eher für kleinere Systeme geeignet, die keine besonderen Anforderungen an Skalierbarkeit oder verteilte Transaktionen haben. Es bietet dafür allerdings durch seine sehr gute Integration in die Java-Plattform eine einfache und unkomplizierte Möglichkeit, Objekte über das Netzwerk zu verteilen. Im Falle von COM+ bzw. DCOM ist die reine Verteilung von Objekten nicht ganz so einfach und in der Regel nur mit Toolunterstützung effizient möglich, da hier ein Binärformat der Schnittstellen vorgegeben wird.

Ein weiteres Unterscheidungskriterium ist die Unterstützung verschiedener Plattformen sowie Programmiersprachen. Hier legt man sich mit RMI und EJB auf Java fest, bleibt aber durch das Konzept *write once run anywhere* relativ flexibel, was die endgültige Plattform angeht. Setzt man dagegen COM+ ein, so ist man derzeit auf die Windows-Plattform, für die gesamte Funktionalität sogar auf das Windows 2000 festgelegt. Dafür ist man in der Wahl der Programmiersprachen etwas freier, wobei auch hier hauptsächlich die jeweiligen Varianten von Microsoft insbesondere wegen der notwendigen Unterstützung durch die Entwicklungsumgebungen zu bevorzugen sind. Auch hier erweist sich CORBA als die flexibelste Lösung, da man hier sowohl was die Plattformen als auch was die Programmiersprachen angeht weitgehend unabhängig bleibt.

Es stellt sich also die Frage, welche der einzelnen Kriterien für den Einsatz in einem Call Center von Bedeutung sind. Dazu gehört sicherlich die Möglichkeit der Skalierbarkeit, da die Last eines Call Centers sehr starken Schwankungen unterworfen ist und Spitzen zu ganz besonderen Zeiten auftreten. Gleiches gilt auch für die Stabilität des Systems, da auch kürzere Ausfälle während der Geschäftszeiten die Funktionalität wesentlich beeinträchtigen. Betrachtet man mehr die technischen Details von Telefonsystemen und CTI-Lösungen so fällt auf, dass sehr viele Funktionen durch asynchron auftretende Ereignisse ausgelöst werden. Die Unterstützung von verteilt auftretenden Ereignissen in der Middleware ist deshalb hilfreich und nützlich.

Was die Unterstützung von Transaktionen angeht, so hängen die Anforderungen hier sehr stark von der jeweiligen Anwendung ab. In vielen Fällen etwa in Call Centern einer Bank haben die Applikationen am Arbeitsplatz einen Anschluss an einen Transaktionsmonitor. Das CTI-System ist davon allerdings in der Regel nicht betroffen. So wird auch die Teilnahme an verteilten Transaktionen für das zu entwerfende

Framework eine eher untergeordnete Rolle spielen. Ähnliches gilt auch für die Anbindung bestehender Legacy-Systeme. Diese spielt in den meisten Call Centern eine Rolle, da sich die Informationen des Kerngeschäfts üblicherweise in solchen Systemen befinden und die CTI-Anwendungen erst später entwickelt werden. Sie ist aber in erster Linie für die konkreten Applikationen und weniger für das Framework von Bedeutung. Da man jedoch in einem Unternehmen idealerweise nur eine Middleware-Technologie verwendet, ist also die Unterstützung von Transaktionen sowie die Verfügbarkeit auch auf Legacy-Plattformen durchaus ein wichtiges Argument. Dies betrifft auch den Einsatz unterschiedlicher Programmiersprachen, da hier durch bestehende Systeme und Anwendungen eventuell individuelle Vorgaben zu erfüllen sind.

Somit sprechen derzeit die meisten Gründe für den Einsatz von CORBA als Basissystem, da damit der größte Funktionsumfang und sicherlich eine der stabilsten Technologien gewählt wird. Interessant ist dabei noch die EJB-Spezifikation, die eine Kombination mit CORBA ausdrücklich zulässt. Man kann ganz allgemein erkennen, dass die Unterstützung von Java durch CORBA sehr gut ist. Dies legt die Überlegung nahe, eine Lösung auf Basis von CORBA zu einem späteren Zeitpunkt zu einer EJB-Lösung zu erweitern. Dies sollte demnach schon bei der Konzeption des Systems an den wichtigen Stellen berücksichtigt werden.

5.3 Programmiersprache

Analog zum Einsatz einer Middleware ist die Wahl der Programmiersprache und den damit verbundenen Entwicklungsumgebungen eine grundlegende Entscheidung bei der Entwicklung eines Softwaresystems. Allerdings ist hier eine Bewertung der Alternativen weitaus weniger umfangreich, da durch die Vorgabe einer objektorientierten Entwicklung sowie die derzeit klare Dominanz einiger Programmiersprachen die Auswahl doch wesentlich eingeschränkt wird. In diesem Abschnitt werden einige der wichtigsten Kriterien zusammengestellt, anhand derer der Einsatz einer konkreten Programmiersprache bewertet werden kann. Anschließend werden die derzeit wichtigsten Alternativen für die objektorientierte Entwicklung, nämlich C++ und Java, kurz beleuchtet.

5.3.1 Anforderungen

Die Anforderungen, die derzeit an eine moderne Programmiersprache gestellt werden, sind inzwischen sehr vielfältig. Für eine Einschätzung im Hinblick auf die Verwendung als Sprache für das Framework reichen jedoch einige der wichtigsten Kriterien aus:

- Unterstützung der objektorientierten Entwicklung
  Vererbung, Polymorphismus etc.
- Plattformunterstützung
- Programmiermodell
  (references, call by value, dynamic binding, garbage collection, Fehlertoleranz, Exceptions)
- Stabilität und Performance
  Die Anforderungen in einem Call Center an diese Kriterien sind bekanntlich sehr hoch. Deshalb muss eine Programmiersprache und ihre Entwicklungsumgebung auch in dieser Hinsicht Ausreichendes bieten.
- Standardisierung
  Gerade für größere Systeme, die über einen längeren Zeitraum Änderungen zu erwarten haben, ist es wichtig, wie standardisiert die Programmiersprache ist, die für die Entwicklung verwendet wird. Nur so kann eine ausreichend lange Pflege und Wartung gewährleistet werden.

Bezogen auf die einleitend definierten Kategorien kann man hier alle Anforderungen als unbedingt notwendig (also A) bezeichnen, bis auf den Punkt der Standardisierung, der der Kategorie B zuzuordnen ist.

5.3.2 Java

Die allgemeine Akzeptanz, die Java in der letzten Zeit in den verschiedensten Anwendungsbereichen erfahren hat, zeigt sich auch im CTI-Umfeld. Die neuartigen Sprachkonzepte sowie die gute Unterstützung für die Entwicklung verteilter Systeme sind Grund genug, den Einsatz von Java zu berücksichtigen.

Die allgemeinen Vorteile dieser Programmiersprache liegen einerseits im Ansatz, eine einfache, stark objektorientierte Sprache zur Verfügung zu stellen, sowie andererseits im Bemühen, eine hohe Plattformunabhängigkeit zu erreichen. In der Literatur existieren bereits eine ganze Reihe von Untersuchungen, die Vor- und Nachteile von Java diskutieren sowie die verschiedenen Aspekte wie Unabhängigkeit, Robustheit, Sicherheit, Garbage Collection, die Möglichkeiten von Multithreading oder die Unterstützung bei der Entwicklung verteilter Anwendungen diskutieren und bewerten (siehe zum Beispiel [Orfali98] oder [Sing98]). Es ist demnach in diesem Zusammenhang ausreichend, die für den konkreten Anwendungsfall wesentlichen Kriterien kurz zusammenzufassen.

Das Programmiermodell von Java zeigt eine sehr durchgehende Umsetzung der objektorientierten Ansätze, die teilweise an das Modell von Sprachen wie Objective C angelehnt ist. Wichtiger Aspekt dabei war für die Designer, die Komplexität von C++ ganz bewusst zu vermeiden. Beispiele dafür sind die automatische Bereinigung des Speichers von nicht mehr referenzierten Objekten (*garbage collection*) oder der Wegfall der in C++ intensiv verwendeten Zeiger. Außerdem unterstützt Java das dynamische Binden (*dynamic binding),* wobei der Typ eines Objektes erst zur Laufzeit festgestellt wird, wodurch Polymorphismus sehr einfach umgesetzt werden kann.

Ein häufig angesprochener Nachteil von Java ist die gegenüber C oder C++ schlechtere Performance. Dies resultiert unter anderem aus der Tatsache, dass Java eine interpretierte Sprache ist sowie aus verschiedenen Konzepten, wie etwa der asynchronen Garbage-Collection oder der Typüberprüfung zur Laufzeit. Allerdings bietet Java eine sehr gute und eng in das System integrierte Unterstützung von Threads an, wodurch relativ schnell gut skalierbare Anwendungen erstellt werden können und so ein Teil der schlechteren Performance oft wieder aufgeholt werden kann.

Für Client-Anwendungen in Call Centern ist Performance in der Regel von nicht allzu großer Bedeutung, da die typischen Abläufe nicht besonders zeitkritisch sind. Eine Ausnahme bildet lediglich der *screen pop*, da hier ja möglichst viel Informationen des Anrufers in der kurzen Zeit präsentiert werden müssen, bevor ein Bearbeiter das Gespräch entgegennimmt. Anders sieht es hier bei den Serverkomponenten aus, da deren mittlere Antwortzeit nicht wesentlich unter der liegen darf, die durch das Telefonsystem vorgeben wird. Die Anforderungen an die gesamte Leistung des Systems sind dabei jedoch auch im Zusammenhang mit einer möglichen Skalierbarkeit der Middleware zu sehen.

Betrachtet man die Plattformunabhängigkeit, die Java bietet, so wurde hier ein im Gegensatz zu herkömmlichen Sprachen wie C++ ein ganz neues Konzept (*„write once run anywhere“)* gewählt. Programme werden dabei nicht in Maschinensprache sondern in einen neutralen Bytecode übersetzt und zur Laufzeit durch die sog. *virtual machine* auf der jeweiligen Zielplattform interpretiert. Das Programm nutzt demnach alle Dienste dieses virtuellen Systems und ist damit ohne weitere Änderungen auf jedem System lauffähig, das Java unterstützt. Dabei ist jedoch zu bedenken, dass dazu keine speziellen Anforderungen an Ressourcen wie etwa die Hardware eines Systems gestellt werden dürfen. Geräte wie Joysticks oder Mäuse mit mehreren Tasten werden von Java nicht unterstützt. Bezogen auf CTI wird dieser Aspekt besonders wichtig, wenn es um *first-party* Lösungen geht, die Telefonfunktionalität also direkt über eigene Hardware des PCs bzw. der Workstation gesteuert wird. Die Telefonie-Anwendungen, die üblicherweise in Call Centern zum Einsatz kommen und demnach für die hier dargestellten Untersuchungen von Bedeutung sind, sind in der Regel in höheren Schichten angesiedelt und nutzen CTI-Dienste über spezielle Schnittstellen, die auf mehreren Plattformen verfügbar sein sollten. Im Falle von Java ist dies zum Beispiel JTAPI. Derartige Schnittstellen existieren für die verschiedensten Bereiche und sind eng an die Java-Plattform gekoppelt. So gibt es Bibliotheken zur Entwicklung von GUI-Komponenten, zur Netzwerkprogrammierung und WWW-Anbindung, zur Erstellung von 2D- bzw. 3D-Graphiken oder die bereits angesprochene RMI-Middleware. Diese Schnittstellen erhöhen die Portierbarkeit von Java enorm und sind mit ein Grund für die schnelle Verbreitung und Akzeptanz dieser Sprache.

Ein weiterer, sehr interessanter Aspekt von Java ist die Möglichkeit, Java-Applets zu entwickeln, die von WWW-Browsern über das Internet oder ein lokales Intranet geladen und ausgeführt werden. Dieser Ansatz hat zur Idee der *network computer* geführt, die selbst über keinen permanenten Plattenspeicher mehr verfügen und die gesamte Anwendungssoftware bei Bedarf über ihren WWW-Browser laden. Im Gegensatz zu vielen anderen Anwendungen, die hohe Anforderungen an Ressourcen wie Plattenplatz, Grafikleistung oder Multimediafähigkeiten haben, sind CTI-Anwendungen eines Call Centers hier durchaus geeignet. Der Großteil der Telefonfunktionalität wird ja durch die lokale Hardware, die zentrale Telefonanlage sowie eine CTI-Middleware bereit gestellt, somit sind im Grunde relativ kleine Clientanwendungen ausreichend. Die Praxis zeigt aber, dass zum Beispiel schon die Gestaltung etwas aufwändigerer Oberflächen in der Regel die Nutzung einer Reihe von recht umfangreichen Bibliotheken voraussetzt, die die Größe des Applets erheblich beeinflussen. Die daraus resultierende Netzbelastung sowie entsprechend lange Ladezeiten schränken unter Umständen die Einsatzmöglichkeiten solcher Systeme wieder ein.

Der Einsatz eines CTI-Clients als Java-Applet ist zwar für die spezifizierte Funktionalität des Frameworks nicht relevant, könnte aber in weiteren Ausbaustufen gerade im Hinblick auf eine engere Integration der CTI-Technologie mit dem World Wide Web ein sehr interessanter Aspekt werden.

Was die Standardisierung von Java angeht, so kann man hier derzeit lediglich von einem Industriestandard sprechen, da die Firma SUN die Sprache bis jetzt noch nicht für eine offizielle Normierung durch ANSI freigegeben hat. Sie hat aber inzwischen die meisten der großen IT-Firmen in den Prozess der Spezifikation dieser Sprache eingebunden, so dass zumindest dadurch eine gewisse Stabilität erreicht werden kann. Eine gut Bewertung von Java auch im Hinblick dieser Aspekte gibt [Tyma98].

5.3.3 C++

C++ ist derzeit die wohl verbreitetste objektorientierte Programmiersprache. Sie wurde in den achtziger Jahren von Bjarne Stroustroup in den Bell Labs der Firma AT&T als Weiterentwicklung von C entworfen. Ursprünglich war C++ ein pre-Compiler, der spezielle Anweisungen in C-Code übersetzte, der dann mit einem üblichen C-Compiler genutzt werden konnte. Es entstanden jedoch sehr bald eigenständige C++-Compiler, die den Quellcode direkt übersetzten. Aus dieser Vorgehensweise stammt die enge Verwandtschaft zu C, da C++ als Übermenge von C auch alle dort vorhandenen Sprachkonstrukte beherrscht. Diese Kompatibilität zu C war ein wesentlicher Grund für den Erfolg dieser Sprache, da einerseits der Umstieg für Entwickler sehr einfach war und andererseits auch bestehende C-Sourcen weiterverwendet werden konnten. Als ISO-Standard ist die Spezifikation der Programmiersprache C++ auch von unabhängiger Stelle eindeutig definiert worden.

Angesichts der weiten Verbreitung von C++ und der Zeit, die diese Sprache bereits im Einsatz ist, existiert mehr als ausreichend Literatur, die alle Aspekte von allgemeinen Einführungen bis hin zur detaillierten Spezifikation des Sprachumfanges umfassen. Als Beispiele seien hier [Ellis90] oder [Lippman91] genannt.

C++ unterstützt alle Konzepte objektorientierter Programmierung und ist vom Sprachumfang wesentlich umfassender als etwa Java. So wird etwa Mehrfachvererbung, das Überladen von Operatoren oder die Verwendung von Templates unterstützt. Da C++ darüber hinaus auch noch die gesamte Funktionalität von C beinhaltet, wie etwa die Verwendung von Zeigern, bieten sich dem Entwickler sehr umfangreiche Möglichkeiten. Allerdings auch die Schwierigkeit, immer komplexeren Programmcode zu beherrschen. Diese Komplexität ist deshalb auch einer der Kritikpunkte an dieser Sprache, da diese dem wirklich konsequent objektorientierten Design oft im Wege steht. So unterstützt C++ keine garbage collection, der Entwickler muss also den verwendeten Speicher selbst verwalten. Auch das Prinzip des späten Bindens (*late binding*), das sehr effizient für den Einsatz von Polymorphismus verwendet werden kann, wird nicht unterstützt. Diese Eigenschaften waren mit ein Grund für die Entscheidung, die Sprache Java ganz bewusst von der teilweise redundanten Funktionalität von C++ zu befreien und wesentlich einfacher für den Entwickler zu gestalten.

Ein wesentlicher Vorteil von C++ ist die Geschwindigkeit und Stabilität, die mit entsprechend gut entwickelten Programmen erreicht werden kann. Dies liegt mit daran, dass C++ ähnlich wie C im Vergleich zu

Java relativ hardwarenah ist und demnach durch den Compiler sehr effizient in einen plattformspezifischen Binärcode zu übersetzen ist. Durch die lange Zeit, die diese Sprache nun bereits am Markt ist, haben die Compiler nun auch eine Stabilität erreicht, die den Entwicklungsumgebungen für Java derzeit noch überlegen sind.

Was den Einsatz von C++ auf unterschiedlichen Plattformen angeht, so ist dafür kein besonderes Konzept wie etwa bei Java vorgesehen. Voraussetzung für eine Portierung ist deshalb zum einen die Verfügbarkeit eines C++-Compilers auf beiden Plattformen sowie zum anderen, dass das Programm selbst ausreichend portabel, also nicht systemspezifisch geschrieben ist. Da derzeit für alle gängigen Systeme C++-Compiler zur Verfügung stehen, stellt die erste Voraussetzung in der Regel kein Problem dar. Hauptverantwortung für die Portabilität liegt jedoch weiterhin bei den Entwicklern, die darauf achten müssen, keine proprietären Bibliotheken zu verwenden. Dies ist insbesondere dann ein Problem, wenn man zum Beispiel relativ nahe an der Hardware entwickeln muss oder aber wenn man eine graphische Benutzeroberfläche benötigt, die in C++ nicht standardisiert angeboten wird. Hier ist wiederum Java im Vorteil, da das JDK als Referenzumgebung neben der reinen Programmiersprache auch die wichtigsten Bereiche wie GUI oder Netzwerkprogrammierung anbietet und somit weitgehend standardisiert.

Abschließend muss noch erwähnt werden, dass C++ im Gegensatz zu Java ein echter ANSI-Standard ist. Für den Entwurf größerer Systeme bzw. für systemkritische Komponenten ist dies ein wichtiges Argument für den Einsatz dieser Sprache.

5.3.4 C#

Als dritte Alternative sei an dieser Stelle noch C# erwähnt, die von Microsoft im Zuge der Einführung der .NET-Architektur neu vorgestellte Sprache. Ähnlich wie Java wird auch C# in einen von der Hardware unabhängigen Zwischencode übersetzt, der dann in einer eigenen Laufzeitumgebung ausgeführt wird. Prinzipiell folgt auch C# der Strategie von Java und eliminiert einige der mächtigen aber auch komplexen Merkmalen von C++, um die allgemeine Handhabung durch den Entwickler zu erleichtern. Automatische Speicherbereinigung, hierarchische Namensräume sowie die Vermeidung von Zeigern und globalen Funktionen werden beispielsweise auch in C# umgesetzt. Erklärtes Ziel von Microsoft ist es allerdings, mit C# mehr Wert auf die Effizienz des Codes und damit die erzielte Performance zu legen. So erlaubt die Sprache die Deklaration von Strukturen, die im Gegensatz zu Klassen auf dem Stack und nicht auf dem Heap verwaltet werden. Diese Strukturen werden auch für die Umsetzung eines einheitlichen Typkonzepts herangezogen, wodurch auch Basistypen wie int oder long genau wie Klassen Methoden, wie etwa die Umwandlung in einen String, anbieten.

Darüber hinaus versucht C#, sich durch eine Reihe von neuen Funktionalitäten von Java abzugrenzen. So werden Ereignisse (events) als integraler Bestandteil der Sprache unterstützt. Auch das Designmuster, Attribute von Klassen mit expliziten Get- und Set-Methoden zu versehen, wird hier direkt durch die Verwendung sog. *Properties* unterstützt. Dabei wird in einer Deklaration sowohl der Inhalt der Variable als auch die Zugriffsmethoden definiert. Als weiteres Merkmal sei abschließend noch erwähnt, dass vor allem im Zusammenhang mit dem .NET-Framework die Interaktion von C# mit anderen Sprachen wie zum Beispiel Visual Basic durch den gemeinsamen Zwischencode erheblich erleichtert wird.

Analog dem .NET-Framework ist C# für die Realisierung eines Frameworks in einem Call Center in Zukunft eventuell eine interessante Alternative. Da alle objektorientierten Mechanismen hier ebenso unterstützt werden, würde dem Einsatz dieser Sprache für die Implementierung des Frameworks nichts im Wege stehen. Derzeit ist jedoch diese Sprache durch die fehlende Marktreife und die noch sehr enge Verknüpfung mit der Windows-Plattform nur in Sonderfällen für die Implementierung zu empfehlen.

Ein gute Übersicht über C# vor allem im Vergleich zu Java gibt [Albahari00].

5.3.5 Fazit

Die in den kurzen Übersichten von Java, C++ und C# herausgestellten Kriterien sind in Tabelle 5-11 nochmals zusammengefasst. Daraus geht hervor, dass die beiden Sprachen doch relativ unterschiedliche Ansätze verfolgen. C++ versucht, möglichst alle Möglichkeiten moderner Programmentwicklung abzudecken und dabei weiterhin kompatibel zu C zu bleiben. Daraus resultierte eine sehr hohe Komplexität, die der Qualität größere Systeme und vor allem der Umsetzung stark objektorientierter Konzepte oft im Wege steht. Wichtigster Vorteil von C++ ist nach wie vor die hohe Stabilität und Leistungsfähigkeit der daraus resultierenden Programme.

	Java	C++	C#
OO Konzepte (A)	Werden sehr konsequent unterstützt	Werden gut unterstützt. Umsetzung ist allerdings komplex	Werden sehr konsequent unterstützt
Plattformunterstützung (A)	Sehr gut durch WORA Konzept	Gut durch hohe Anzahl verfügbarer Compiler	Bisher beschränkt auf Windows
Programmiermodell (A)	Komfortabel und mächtig	Sehr mächtig und komplex	Komfortabel und mächtig
Stabilität und Performance (A)	Niedrig bis Mittel	Hoch	Noch unklar
Standardisierung (B)	Noch kein echter Standard	Ja (ANSI)	Standardisierung angestrebt

Tabelle 5-11 Vergleich von Java, C++ und C#

Java setzt dagegen auf ein Programmiermodell, das dem Entwickler möglichst komfortable Mechanismen zur Verfügung stellt und ihn von Aufgaben wie die Verwaltung von Speicher und Zeigern entlastet. Dadurch und durch Konzepte wie das späte Binden lassen sich objektorientierte Ansätze in Java sehr effizient und konsequent umsetzen. Darüber hinaus beinhaltet Java ein ganz neues Konzept der Plattformunabhängigkeit durch die Verwendung eines neutralen Byte-Codes, der zur Laufzeit interpretiert wird. All diese Konzepte haben neben dem Komfort für den Entwickler allerdings Einbußen in Bezug auf die Leistungsfähigkeit der Software zur Folge. So sind Java-Programme hier und was den Bedarf an Ressourcen wie etwa Hauptspeicher angeht im Nachteil. Java erfüllt jedoch wie C++ die Anforderungen der Kategorie und ist daher als geeignet einzustufen.

Für die prototypische Entwicklung eines Frameworks bietet sich demnach Java an, da es hier in erster Linie um die Umsetzung neuer Konzepte geht und damit die einfachere und vor allem schnellere Implementierung durch Java einen wesentlichen Vorteil darstellt. Dazu zählen auch die Möglichkeiten im Bereich der GUI-Entwicklung. Auch die gute Unterstützung durch RMI/EJB und CORBA sprechen für einen Einsatz dieser Sprache. Die dadurch zu erwartende schlechtere Performance wird dabei bewusst in Kauf genommen, da zum einen die Leistungsfähigkeit der Hardware weiterhin rapide zunimmt und zum anderen auch immer mehr Optimierungen der Java-Technologien in diesem Bereich stattgefunden haben. Ob die endgültige Performance allerdings letztendlich ausreicht, müssen die Ergebnisse dieser Arbeit zeigen.

5.4 Bewertung der Technologieunabhängigkeit

Wie die vorangegangenen Untersuchungen gezeigt haben, erfüllen JTAPI, TAPI und TSAPI alle notwendigen Kriterien mit Ausnahme der Verknüpfung des Anrufs mit Daten. Für den Bereich Middleware erfüllt CORBA alle wichtigen Eigenschaften, RMI und EJB ebenfalls bis auf einen Ereignisdienst, sowie COM+ bis auf die Unterstützung mehrerer Betriebssysteme. Was den Einsatz objektorientierter Programmiersprachen angeht, so kann man festhalten, dass die Alternativen C++, Java und C# ebenfalls alle grundsätzlich für eine Implementierung in Frage kommen. Somit wird durch die Analyse belegt, dass ausreichend Alternativen bzgl. notwendiger Basistechnologien zur Verfügung stehen und somit die Voraussetzungen für einen technologieunabhängigen Ansatz gegeben sind. Ein nicht zu vernachlässigendes Ergebnis der Technologieanalyse ist jedoch auch, dass aus der qualitativen Bewertung heraus eindeutige Empfehlungen für CORBA, Java und JTAPI gegeben werden können.

Was die Technologieunabhängigkeit angeht, so muss jedoch noch zwischen

1. der Neuimplementierung auf Basis dieser Technologien und
2. dem nachträglichen Wechsel einer dieser Technologien unterschieden werden.

Für den ersten Fall ist eine echte Unabhängigkeit gegeben, da sich alle derzeit gängigen Technologien grundsätzlich für eine Implementierung eignen, wenn auch im Falle der Middleware außer bei CORBA einige Kompromisse eingegangen werden müssen. Dazu kommt, dass die Mehrzahl der Technologien weitgehend standardisierte Spezifikationen darstellen und somit ein weiterer Freiheitsgrad bzgl. konkret einzusetzender Produkte besteht. So stehen im Falle von CORBA oder EJB eine ganze Reihe geeigneter Implementierungen zur Verfügung.

	JTAPI	TAPI	TSAPI	CORBA	RMI	EJB	COM+	Java	C++	C#
JTAPI	-	😐 →C	😐 →C							
TAPI	😐 →Java	-	😐							
TSAPI	😐 →Java	😐	-							
CORBA				-	☺ -F	☺	☹ -F			
RMI				☺	-	☺	☹ -F →C++			
EJB				☺ -F	☺ -F	-	☹ -F →C++			
COM+				😐 -F	☹ →Java	☹ -F →Java	-			
Java								-	☹ → C++	☹ → C#
C++								☹ →Java	-	☹ → C#
C#								☹ →Java	☹ → C++	-

-F	Verlust von Funktionen	☺	geringer Aufwand
→C	notwendige Umstieg	😐	mittlerer Aufwand
		☹	hoher Aufwand

Tabelle 5-12 Abhängigkeiten verschiedener Technologien

Etwas anders ist die Situation, wenn man das Framework hinsichtlich der nachträglichen Änderung eingesetzter Technologien betrachtet. Welchen konkreten Einfluss dieser Wechsel auf die Implementierung hat, wird in Kapitel 9.2 nochmals diskutiert. Nicht zu vernachlässigen ist jedoch, dass der Wechsel von Technologien Auswirkungen auf das Softwaresystem hat, die nicht unmittelbar mit der Architektur zusammenhängen. So bedeutet der Wechsel von COM+ zu EJB auch den Wechsel von C++ hin zu Java, oder der Wechsel von JTAPI nach TAPI den Wechsel von Java zu C. In beiden Fällen bedeutet dies eine Neuimplementierung bereits bestehenden Programmcodes. Umgekehrt kann der Wechsel zum Beispiel des Betriebssystem oder der Programmiersprache einen Wechsel der CTI-Anbindung bedeuten. Dazu kommt, dass Systeme wie eine Middleware unternehmensweit von Bedeutung sind und nicht nur aus Sicht des hier dargestellten Frameworks betrachtet werden können.

Tabelle 5-12 skizziert hierzu nochmals die möglichen Varianten.

Insgesamt kann an dieser Stelle also festgehalten werden, dass das Framework unabhängig gegenüber der Implementierung auf Basis verschiedener Technologien ist, aber für den Fall einer nachträglichen Änderung berücksichtigt werden sollte, dass es hier eine Reihe von weiteren Abhängigkeiten gibt, die einen konkreten Wechsel erschweren. Die aus der Analyse gezogene Empfehlung konkreter Technologien sollte demnach auch in diesem Zusammenhang bei den weiteren Betrachtungen berücksichtigt werden.

5.5 Zusammenfassung

Ziel dieses Kapitels war es, die für eine Implementierung des Frameworks verfügbaren Basistechnologien anhand eines definierten Anforderungskatalogs zu untersuchen und so die Technologieunabhängigkeit des Ansatzes zu stärken. Dabei wurden für den Bereich CTI die Alternativen JTAPI, TAPI sowie TSAPI betrachtet, die alle die wesentlichen Dienste bereitstellen, die für das spezifizierte Framework in einem ersten Schritt benötigt werden. Dabei stellt TAPI einen relativ einfachen Zugang zu einem Telefonsystem zur Verfügung, der vom Ansatz her eher für die direkte Verknüpfung eines Telefons mit einem Arbeitsplatz als für ein zentrales CTI-System in einem Call Center gedacht ist. TSAPI dagegen stellt hier derzeit den Standard dar, der für die Anbindung von IT-Systemen an Telefonanlagen in der Regel verwendet wird.

JTAPI wählt dabei einen neuen, teilweise sehr modernen Ansatz und integriert die CTI-Funktionalität möglichst eng an die Technologie der Java-Plattform sowie die Konzepte objektorientierter Entwicklung. Dadurch ist vom Prinzip her eine relativ komfortable Nutzung der Dienste möglich. Außerdem lässt sich ein objektorientierter Ansatz konsequenter umsetzen als es bei den anderen Schnittstellen der Fall ist. Allerdings müssen diese Ansätze durch geeignete Implementierungen auch effizient umgesetzt werden und in der Praxis eine ausreichende Verbreitung finden, damit diese Technologie auch in größeren Systemen eingesetzt werden kann.

Als Alternativen zur Middleware wurden daraufhin CORBA, Java RMI, EJB und DCOM betrachtet. Als Fazit eines Vergleichs im Hinblick auf die spezifizierten Anforderungen hin kristallisierte sich dabei CORBA als primäre Variante heraus, mit der Option einer späteren Erweiterung in Richtung EJB und Komponententechnologie. Dabei waren Stabilität und Skalierbarkeit sowie die Unabhängigkeit von Plattform und Programmiersprache ausschlaggebende Argumente.

Abschließend wurden dann noch die derzeit wichtigsten objektorientierten Programmiersprachen C++ und Java miteinander verglichen. Der Schluss dabei ist, dass Java hier einen modernen Ansatz darstellt, der zwar derzeit noch nicht die Performance und Zuverlässigkeit von C++-Programmen erreicht, aber eine effiziente und komfortable Umsetzung objektorientierten Designs erlaubt und im Rahmen der Java-Plattform auch viele Bereiche der Softwareentwicklung wie etwa graphische Oberflächen bereits sehr gut abdeckt.

Man muss sich insgesamt für die Realisierung des spezifizierten Frameworks also entscheiden, ob man eher auf bewährte und in der Praxis bereits oft eingesetzte oder aber neuere und dafür vom Ansatz her modernere Technologien setzt. Da die Konzeption eines neuen Systems ganz klar im Vordergrund dieser Arbeit stehen, liegt es nahe, modernere Ansätze von Technologien höher einzustufen als zum Beispiel der langjährige Einsatz in der Praxis. Eine Ausnahme stellt dabei CORBA dar, das immer noch die größte Funktionalität für verteilte Systeme bietet und sich gleichzeitig inzwischen auch in vielen konkreten Projekten bewährt hat. Als Programmiersprache wird deshalb Java bevorzugt und dazu JTAPI als Schnittstelle zum CTI-System. Alle Komponenten bauen dabei auf CORBA als Middleware auf.

Nach einer ersten Analyse hinsichtlich der geforderten Unabhängigkeit von einzelnen Technologien konnte festgestellt werden, dass diese Forderung erfüllt wird, dabei aber insbesondere auch die Abhängigkeiten der Technologien untereinander im weiteren Verlauf in Betracht gezogen werden müssen.

6 SPEZIFIKATION

Nachdem die grundlegenden Anforderungen an ein fortschrittliches IT-System im Call Center grob spezifiziert wurden besteht der nächste Schritt darin, mit Hilfe der eingeführten Methoden die Funktionalität des Systems im Sinne einer Definitionsphase weiter zu verfeinern und insbesondere ausreichend formal zu spezifizieren.

Einleitend dazu sollten die allgemeinen Ziele nochmals zusammengefasst werden. Das neuartige System soll ein Basisframework für verteilte CTI-Anwendungen sein, das den bereits geschilderten Anforderungen eines modernen Call Centers genügen muss. Diese Anforderungen umfassen:

- Die Unterstützung verteilter Abläufe
- Eine erweiterte, in der Anwendungsebene angesiedelte Schnittstelle zur Anrufkontrolle
- Ein erweiterter Zustandsautomat
- Eine flexible und komfortablere Verknüpfung von Anruf mit Daten

Darüber hinaus ist auf die Integration des Framworks in eine bestehende IT-Landschaft eines Call Centers zu achten, das System muss also im Hinblick auf Portabilität sowie Schnittstellen den Anforderungen eines offenen Systems nach konzipiert werden.

6.1 Definition eines Frameworks

Ehe die Funktionalität des Systems genauer festgelegt wird, sollte der Begriff *Framework* betrachtet und im Zusammenhang mit dieser Arbeit eindeutig definiert werden. Der Begriff Framework wird im Umfeld der objektorientierten Entwicklung im Zusammenhang mit dem Konzept der Wiederverwendbarkeit als wichtiger Ansatz diskutiert.

[John97a] liefert für den Begriff zwei sehr allgemeine Definitionen, die Framework als eine Technik im Rahmen der Wiederverwendbarkeit sehen:

> *"[..] a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact",*
>
> *"[..] a framework is the skeleton of an application that can be customized by an application developer".*

Eine sehr ähnliche Definition gibt [Foote88]:

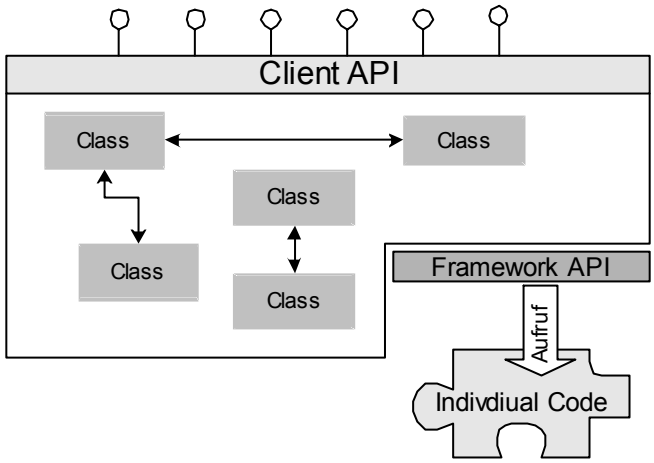
> *„An object-oriented framework is a set of classes that provide the foundation for solutions to problems in a particular domain. Individual solutions are created by extending existing classes and combining these extensions with other existing classes".*

Einige wichtige Eigenschaften eines Frameworks stellt auch [Gamma95] zusammen:

> *„The framework dictates the architecture of your application.[…] The framework captures the design decisions that are common to its application domain. […]"*

Es handelt sich dabei also um ein objektorientiertes Basissystem, das durch entsprechende Änderungen und Erweiterungen eines Entwicklers zu einer individuellen Software-Lösung wird. Wichtig dabei ist jedoch, dass hier nicht nur neutrale Funktionalität einzelner Klassen zur Verfügung gestellt wird. Vielmehr besitzt ein Framework bereits eine sehr mächtige Basisfunktionalität und legt neben der Architektur vor allem auch die Interaktion zwischen seinen Klassen fest, und das in der Regel speziell für einen gewissen Anwendungsbereich. Frameworks beinhalten für den Anwender deshalb als wichtiges Element das grundlegende Design, das für konkrete Lösungen als Basis verwendet wird. Somit wird ein derartiges System von einem Entwickler auch nicht wie eine Komponente einer Bibliothek verwendet. Während hier das Hauptprogramm in der Regel den gesamten Programmablauf definiert und für benötigte Funktionalität

Dienste der Komponenten aufruft, wird der individuelle Programmcode in ein bestehendes Framework lediglich integriert. Das heißt, der wesentliche Programmablauf ist bereits Bestandteil des Framworks und wird wiederverwendet. Somit ruft in diesem Fall das Framework aus seinem definierten Ablauf heraus die individuellen und neuen Programmteile auf und nicht umgekehrt. Dieses Prinzip wird auch als *inversion of control* bezeichnet und ist eine weitere Charakteristik dieses Ansatzes.



Quelle: [Taligent93] [Orfali96b]

Abbildung 6-1 Framework Ansatz

In Abbildung 6-1 ist dieser Ansatz nochmals dargestellt. Nach außen hat ein Framework eine klar definierte Schnittstelle, die von Client-Anwendungen verwendet werden können. Frameworks werden somit genutzt, indem seine Klassen instanziert und deren Methoden aufgerufen werden. Sie werden angepasst, indem vorhandene Klassen abgeleitet und Methoden neu definiert werden, die dann im Rahmen des vorgegebenen Programmablaufs aufgerufen werden.

Das Framework-Konzept ist nur eines von mehreren Ansätzen mit dem Ziel, die Wiederverwendbarkeit objektorientierter Systeme zu steigern. Objektorientierte Klassenbibliotheken (*class libraries*), Komponenten (*components)* sowie Entwurfsmuster (*design patterns)* sind weitere Technologien, die in diesem Umfeld entstanden sind und versuchen, den Aufwand für die Entwicklung komplexer Systeme zu reduzieren. Gegenüber Klassenbibliotheken grenzen sich Frameworks dadurch ab, dass sie den Programmablauf und damit die Interaktion der einzelnen Klassen vorgeben. Somit findet auch das Prinzip *inversion of control* bei Klassenbibliotheken in der Regel keine Anwendung. Der Unterschied ist dabei allerdings eher fließend, je nach Design der Bibliothek oder eines Frameworks.

Enger verwandt sind Frameworks dagegen mit den Konzepten von Komponenten und Entwurfsmustern. Dabei konzentriert sich der Ansatz der Komponenten auf die Wiederverwendung von Programmcode, der Ansatz von Entwurfsmustern auf die Wiederverwendung von Programmdesign. Frameworks liegen hier eher in der Mitte, da sie einen sehr großen Teil des Designs einer Anwendung bereits beinhalten, im Gegensatz zu reinen Entwurfsmustern allerdings als Programmtext vorliegen. Entwurfsmuster grenzen sich hier dadurch ab, dass sie abstrakter und allgemeiner sind als Frameworks und eher als Teile dieser anzusehen sind, nicht umgekehrt (siehe [Gamma95]).

Da Frameworks bereits viele, für den Anwendungsbereich wichtige Annahmen und Vorgaben beinhalten, können sie als Umgebung für den Einsatz bestehender oder auch die Entwicklung neuer Komponenten dienen. Ein Framework wird weiterhin, wie bereits erwähnt, eine Reihe von allgemeinen Entwurfsmustern als Grundelemente enthalten. Die drei Technologien sind also sehr eng miteinander verknüpft und dementsprechend auch nicht völlig getrennt voneinander zu betrachten (siehe auch [John97a] sowie [John97b]).

Abbildung 6-2 stellt noch einen etwas anderen Zusammenhang zwischen Frameworks, Libraries und Komponenten dar. Klassenbibliotheken werden dabei als sog. *white box* bezeichnet, als Softwarekomponenten, deren Verhalten im Gegensatz zur *black box* wesentlich verändert werden kann. Komponenten sind hier das andere Extrem, da hier bis auf eine vorgesehene Anpassung von Parametern keine echte Änderung der Funktionalität möglich ist. Entsprechend einfach ist deshalb auch der Einsatz von Komponenten bzw. hoch der Grad der Wiederverwendbarkeit. Frameworks liegen hier bzgl. beider Aspekte in der Mitte.

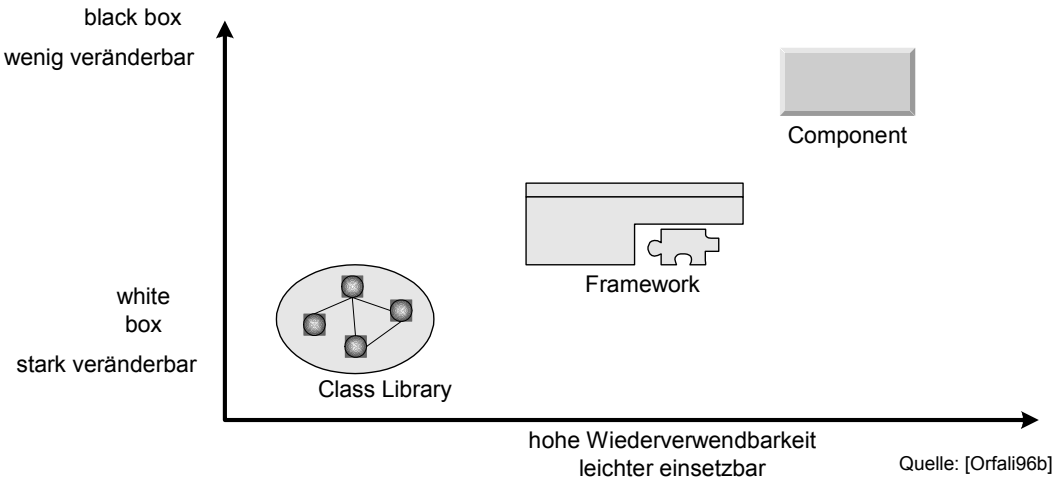


Abbildung 6-2 Einordnung von Klassenbibliotheken, Frameworks und Komponenten

Konkrete Beispiele von Frameworks, die in der Praxis eingesetzt werden, finden sich häufig im Umfeld graphischer Benutzeroberflächen. In diesem Bereich ist es sehr wichtig, dem Entwickler neben der reinen Basisfunktionalität vor allem auch konzeptionelle Ansätze vorzugeben, die für das Verhalten der Oberfläche maßgeblich sind. So sind das Smalltalk-80 User Interface mit seinem Ansatz MVC (*model view controller)*, das Andrew Toolkit der Universität Carnegie Mellon, InterViews der Universität Stanford, OpenStep von NeXT und MFC von Microsoft Frameworks in diesem Bereich. Aber auch Systeme wie OpenDoc, OLE, DSOM und Java Beans sind Frameworks, die sehr verbreitet sind und in der Praxis häufig eingesetzt werden.

Wendet man nun die angesprochenen Charakteristika eines Frameworks auf die konkrete Aufgabenstellung an, so ergeben sich bereits sehr konkrete Anforderungen an die Spezifikation. Es muss demnach eine Menge von Klassen konzipiert werden, die als Basis für verteilte CTI-Applikationen im Call Center dienen. Wichtige Designvorgaben werden dabei bereits getroffen, um wesentliche Abläufe (z.B. Weiterleitung) und Funktionen (zum Beispiel durch einen individuellen Zustandsautomaten) besser unterstützen zu können. Der Entwickler einer konkreten Anwendung kann dieses Framework dann benutzen und an geeigneter Stelle verändern bzw. durch eigene Komponenten erweitern, um es den individuellen Gegebenheiten exakt anzupassen.

Offen ist dabei natürlich noch die Frage, ob die Entwicklung eines Frameworks auch der passende Ansatz für ein fortschrittliches CTI-System ist. Alternative dazu wäre die Konzeption eines möglichst vollständigen Softwareprodukts, das lediglich durch Konfiguration noch an die entsprechenden Gegebenheiten angepasst werden kann. Die bisherigen Untersuchen haben jedoch gezeigt, dass die Anforderungen an die Funktionalität einer modernen CTI-Anwendung in einem Call Center in der Regel individuelle Erweiterungen voraussetzen, insbesondere wenn dabei auch die Integration bestehender Systeme berücksichtigt werden soll. Und da ja gerade diese, nicht standardmäßig vorhandene Funktionalität Inhalt dieser Untersuchung ist, stellt ein Framework, das Anpassung und Erweiterung an individuelle Anforderungen explizit berücksichtigt, einen sinnvollen Ansatz dar.

Als Fazit kann man ziehen, dass die Definition eines objektorientierten Frameworks und das damit verbundene Konzept der Wiederverwendbarkeit von Programmcode und Design die geeignete Basis darstel-

len, um ein konkretes IT-System zu implementieren, das den sehr unterschiedlichen Anforderungen im Call Center gerecht wird. Die Entwicklung reiner Komponenten, die von einem Anwender nur mit vorgegebener Funktionalität genutzt werden können, erscheint dagegen vom Ansatz her als nicht ausreichend flexibel zu sein. Nächster Schritt ist demnach nun, die Funktionalität und das Design eines entsprechenden Frameworks weiter zu verfeinern.

6.2 Verteilte Abläufe

Ersten Bereich der Funktionalität stellen die verteilten Abläufe dar, die durch das System unterstützt werden. Um diese Abläufe eindeutig zu spezifizieren und dabei die hier notwendigen Anforderungen zu konkretisieren bietet es sich an, die wesentlichen Abläufe mit Hilfe von Petrinetzen zu modellieren. Es bieten sich dabei die B/E-Netze an, um die einzelnen Ereignisse, die in einem Telefonsystem bzw. einem CTI-System auftreten, in einem verteilten System modellieren zu können. Damit steht eine formale und eindeutige Definition zur Verfügung, die einerseits bereits erste Analysen der einzelnen Vorgänge erlaubt und später als Basis für den konkreteren Entwurf des Systems dient. Die Verwendung des vergleichsweise einfachen Modells der B/E-Netze hat im Gegensatz zu erweiterten Varianten unter anderem den Vorteil der einfacheren Analysemöglichkeiten, wie sich später noch genauer zeigen wird. Inwieweit das Modell allerdings ausreichend mächtig ist, muss die konkrete Spezifikation noch zeigen.

6.2.1 Anruf

Der Anruf ist wohl der grundlegendste Vorgang in einem Telefonsystem und damit auch in einem Call Center. Dieser Vorgang ist in Abbildung 6-3 dargestellt. Die zwei Systeme der Teilnehmer sind dabei über die Bedingung *Ringing* miteinander gekoppelt. Sobald der erste Teilnehmer den Anruf einleitet, tritt diese Bedingung ein und die Verbindung kann vom zweiten Teilnehmer hergestellt werden. Beide Systeme können allerdings unabhängig voneinander von *Online* in den Zustand *Offline* gehen. In diesem Fall ist der zweite Teilnehmer zwar weiterhin *Online*, es besteht allerdings keine direkte Verbindung mehr. Eine Kopplung dieser Aktionen könnte man durch weitere Ereignisse und Bedingungen ebenfalls modellieren, wodurch das Modell allerdings erheblich komplexer werden würde. Dies ist allerdings im Grunde hier nicht notwendig, da das Verhalten der Telefonanlagen in so einem Fall ohnehin nicht klar definiert ist bis auf die Tatsache, dass keine Verbindung mehr zwischen den Teilnehmern besteht.

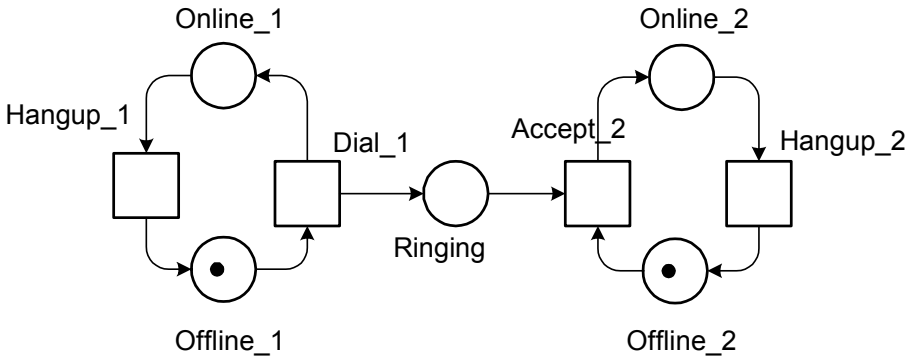


Abbildung 6-3 Petrinetz eines Anrufs / erste Version

Um nun die Eigenschaften des so definierten Ablaufs genauer zu untersuchen, muss das Petrinetz analysiert werden. Dies wurde mit Hilfe eines entsprechenden Softwaretools durchgeführt, die konkreten Ergebnisse sind im Anhang detailliert aufgeführt. Es berechnet im Wesentlichen den Erreichbarkeitsgraphen, also alle möglichen Netzmarkierungen sowie alle möglichen Erreichbarkeiten. Die wichtigsten daraus gewonnenen Ergebnisse sind:

- Das Netz ist beschränkt.
- Die größte Stellenmarkierung ist 1.
- Das Netz ist sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Das Netz ist konfliktfrei.
- Es treten 2 Kontaktsituationen auf.
- Das Netz ist lebendig.

Die Analyse zeigt also im Grunde kein problematisches Verhalten des Ablaufs bis auf die zwei erwähnten Kontaktsituationen. Sie treten bei folgenden Markierungen des Netzes auf:

- (Ringing, Offline1, Offline2)
- (Ringing, Offline1, Online2)

Bei diesen Belegungen kann die Transition *Dial* nicht schalten, da *Ringing* belegt ist. Erst wenn der zweite Teilnehmer den Anruf entgegennimmt, wäre diese Blockade aufgehoben. Dies stellt zwei problematische Situationen dar, da hier das erste System in den Offline-Zustand gehen kann, ohne dass der Zustand *ringing* verlassen wird. Dies spiegelt auch nicht die Realität wider, da das Telefon eines Teilnehmers nicht mehr klingelt, sobald der Anrufer auflegt. Die Modellierung ist daher noch nicht ausreichend, die Teilsysteme der beiden Partner müssen noch enger miteinander verknüpft werden. Man sieht allerdings bereits hier, dass die Spezifikation solcher Vorgänge durch Petrinetze derartige Schwächen sehr gut aufzeigt.

Abbildung 6-4 zeigt eine entsprechend erweiterte Version, die die eben dargestellten Schwächen berücksichtigt. Es wird im Wesentlichen ein neuer Zustand *Calling* sowie eine neue Transition *Cancel* eingeführt. Der Anrufer ist somit nachdem er gewählt hat in einem Zwischenzustand, der eng mit dem zweiten Teilnehmer verbunden ist. Nimmt dieser nämlich das Gespräch an, schaltet also die Transition *Accept,* so werden beide in den jeweiligen Online-Zustand überführt. Der Wählvorgang kann allerdings auch durch *Cancel* vorzeitig abgebrochen werden, wodurch der Anrufer wieder in den Zustand *Offline* geht und gleichzeitig die Bedingung *Ringing* gelöscht wird. Genau betrachtet ist der Zustand *Calling* nicht unbedingt notwendig, sondern könnte auch mit *Ringing* zusammengelegt werden. Allerdings trägt diese Aufteilung zur besseren Übersichtlichkeit bei.

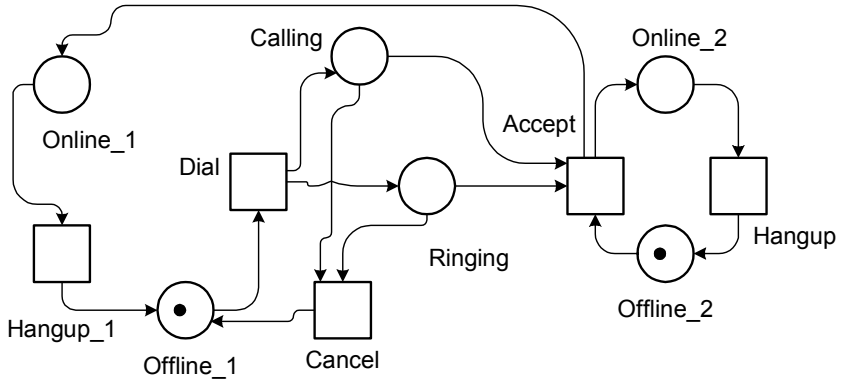


Abbildung 6-4 Petrinetz eines Anrufs/ erweiterte Version

Man erkennt leicht, dass das Netz mit den dargestellten Ergänzungen das Verhalten eines Telefonsystems wesentlich besser modelliert. Analog dem ersten Versuch lässt sich nun auch dieses Netz formal analysieren. Die Ergebnisse daraus sind:

- Das Netz ist beschränkt.
- Die größte Stellenmarkierung ist 1.
- Das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei einer Netz-Markierung treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Das Netz ist lebendig.

Die Kontaktsituationen wurden also durch diese erweiterte Modellierung aufgelöst. Es tritt nun allerdings bei einer Markierung ein Konflikt auf, also eine Situation, bei der zwei aktivierte Transitionen in Konkurrenz zueinander stehen und nur alternativ möglich sind. Im konkreten Beispiel tritt der Konflikt bei folgender Markierung auf:

- Ringing_1, Calling_1 und Offline_2

Die hier betroffenen Transitionen sind *Cancel* sowie *Accept*. Dieser Konflikt stellt allerdings kein Problem dar, sondern spiegelt im Grunde nur die Realität wider. Bei einem Anruf stehen diese zwei Ereignisse immer in Konkurrenz zueinander und die Situation, dass ein Anrufer wieder auflegt, ehe man das Gespräch annehmen kann, ist deshalb ganz alltäglich.

6.2.2 Weiterleitung

Der Ablauf einer Weiterleitung ist in Abbildung 6-5 dargestellt. Er basiert auf den Abläufen eines herkömmlichen Anrufs, wobei nun ein dritter Teilnehmer als Vermittler auftritt. Gekoppelt werden die Systeme der ersten beiden Partner wieder über die Bedingung *Ringing* (der Eindeutigkeit wegen als *Ringing_2* bezeichnet). Sobald der Vermittler das Gespräch angenommen hat kommt er in den Zustand Online, wodurch nun die Bedingung für eine Weiterleitung gegeben ist. Sie wird durch das Ereignis *InitTransfer* eingeleitet, wodurch ein dritter Teilnehmer angerufen wird. Es treten nun die Zustände *Ringing_3* sowie *Calling_2* ein. Der ursprüngliche Teilnehmer wird dabei gehalten (*OnHold_1)*. Nimmt der dritte Teilnehmer das Gespräch an (*Accept)*, so besteht eine Verbindung zwischen beiden Teilnehmern, sie haben den Zustand *TransferPrepared* sowie *Online*. Der Vermittler kann nun die beiden anderen Partner verbinden und selbst wieder in den Offline-Zustand übergehen. Um den Zustand *Calling_2* auch verlassen zu können ohne dass der Anruf angenommen wird, ist eine weitere Transition notwendig, die die Weiterleitung vorzeitig beendet (*CancelTransfer*). Dabei geht der Vermittler wieder in den Zustand *Online_2* über, ist also wieder mit dem ersten Teilnehmer verbunden. In beiden Fällen ändert sich der Zustand des ersten Teilnehmers wieder auf *Online_1*.

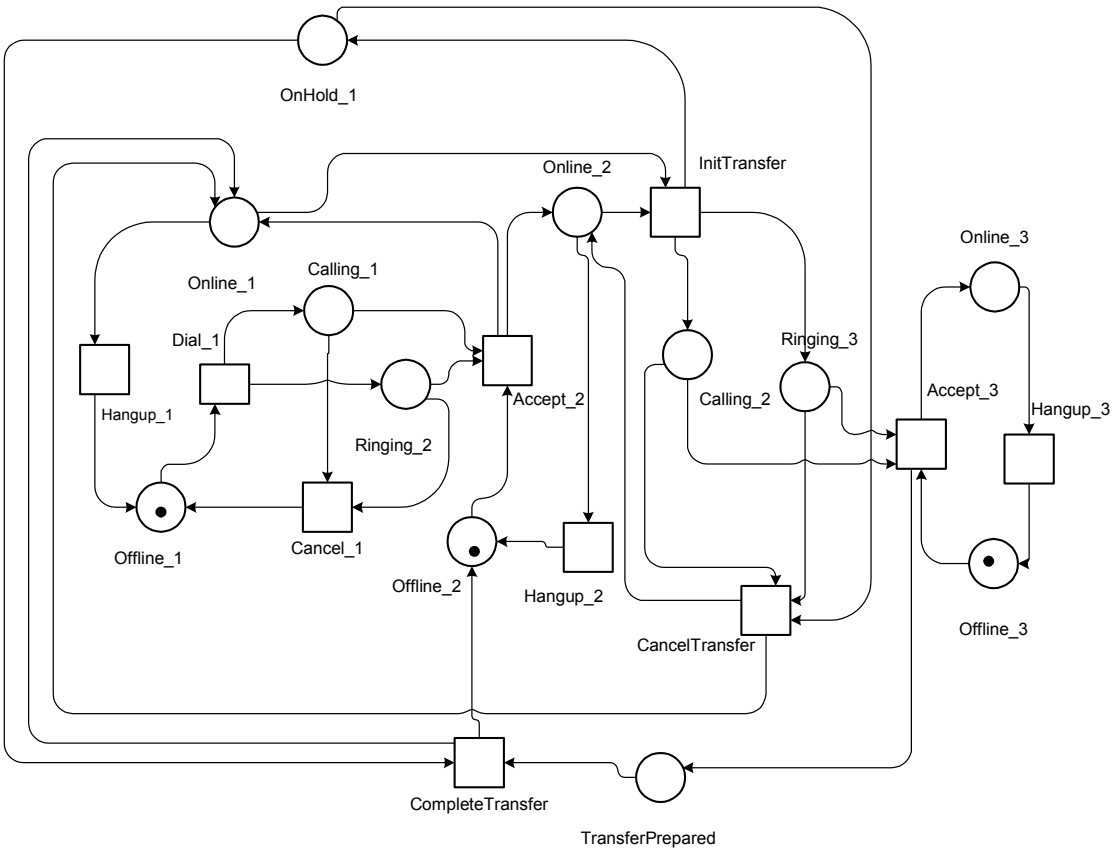


Abbildung 6-5 Petrinetz einer Weiterleitung

Das Beispiel stellt den für ein Call Center typischen Fall dar, dass der angerufene Mitarbeiter den Transfer initiiert. Eine weitere Variante ist hier, dass der Mitarbeiter selbst einen Anruf zu einem Teilnehmer herstellt und dann weiterleitet. Auch dieser Fall würde aber analog dem dargestellten Beispiel durch ein Petrinetz modelliert werden, nur dass dann ein Teilnehmer sowohl über die Transitionen *Dial* als auch *Init-/CompleteTransfer* den Anruf steuern würde.

Die Analyse des Netzes kommt zu folgenden Ergebnissen:

- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei 5 Netz-Markierungen treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Das Netz ist lebendig.

Das Netz zeigt also kein problematische Verhalten, allerdings ist die Anzahl der Konflikte stark angestiegen, da nun neue Zustände hinzugekommen sind, in denen Transitionen nur in Konkurrenz zueinander schalten können.

6.2.3 Weiterleitung mit Synchronisation der Arbeit

Eine Erweiterung der normalen Weiterleitung, die vor allem für CTI-Anwendung von Bedeutung ist, ist in Abbildung 2-1 dargestellt. Sie enthält neben den bereits dargestellten Abläufen eine damit verbundene Synchronisation der Teilnehmer während der Weiterleitung beim Zugriff auf gemeinsame Daten. Da mit

einer Weiterleitung eines Anrufers an einen weiteren Mitarbeiter des Call Centers in der Regel auch die Bearbeitung seiner Angelegenheit und damit der Zugriff auf zentrale Ressourcen weitergegeben wird, ist eins Synchronisierung sehr hilfreich. Durch den dargestellten Ablauf wird hier ein konkurrierender Zugriff vermieden und die Übergabe klar geregelt. Dazu ist anzumerken, dass der erste Teilnehmer als Anrufer von außen keinen Zugriff auf Ressourcen hat und somit für die Synchronisation nicht berücksichtigt werden muss. Weiterhin modelliert das Netz die Synchronisierung zwischen Vermittler und drittem Teilnehmer nur während der Weiterleitung. Zu diesem Zweck wird eine Synchronisationsbedingung *Sync* eingeführt, die den gegenseitigen Ausschluss der Teilnehmer umsetzt. Nach dem Annehmen des Gesprächs sichert sich der Vermittler mit *BeginWork_2* den Zugriff auf die Ressourcen. Diese Reservierung bleibt solange bestehen, bis der Transfer erfolgreich abgeschlossen oder aber abgebrochen wurde. Der dritte Teilnehmer muss, nachdem er wiederum sein Gespräch angenommen hat, auf die Freigabe warten, ehe er durch *BeginWork_3* Zugriff auf die Ressourcen bekommt.

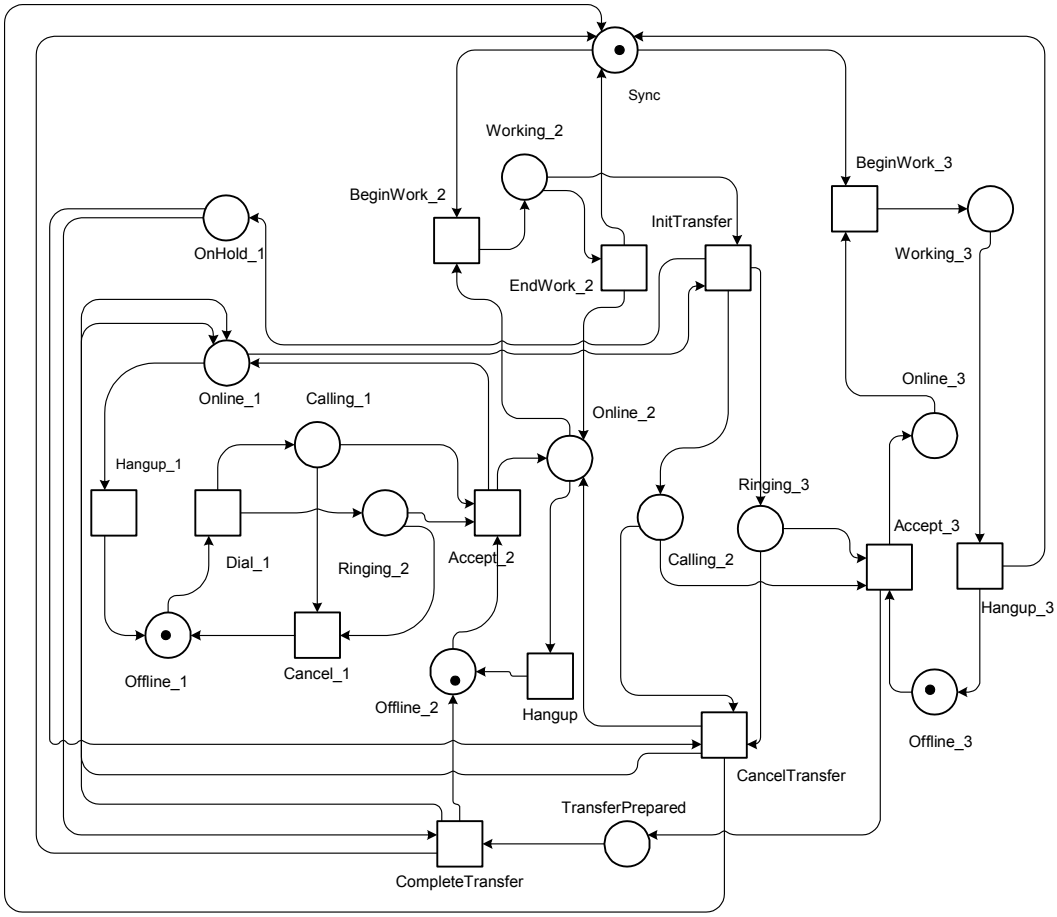


Abbildung 6-6 Petrinetz einer Weiterleitung mit Synchronisation

Durch die enge Verwandtschaft dieses Ablaufs mit der einfachen Version einer Weiterleitung ergibt sich auch in diesem Fall kein problematisches Verhalten, wie die Analyse zeigt. Es entsteht hier jedoch eine weitere Konfliktsituation durch die Einführung der neuen Synchronisationsstelle.

6.2.4 Konferenz

Die vollständige Modellierung einer Konferenz ist sehr schwierig, da hier die Anzahl der Teilnehmer und damit das verteilte System vom Prinzip her dynamisch sind. Eine Telefonanlage erlaubt in der Regel die Hinzunahme beliebiger Teilnehmer zu einer Konferenz. Für ein Petrinetz bedeutet dies allerdings, dass hier dynamisch weitere Systemteile mit eigenen Zuständen abgebildet werden müssen, was ohne weiteres nicht möglich erscheint. Es wird deshalb nur der einfache Fall betrachtet, bei dem genau ein weiterer Teil-

nehmer zu einer Dreier-Konferenz hinzugenommen wird. Dieser Fall deckt im Wesentlichen den Ablauf einer Konferenz ab, da das Hinzunehmen weiterer Parteien hier nur eine Wiederholung des Basisablaufs darstellt.

Abbildung 6-7 zeigt das hierfür entworfene Petrinetz. Analog zur Weiterleitung geht auch hier der Vermittler über *InitConference* in den Zustand *Calling_2* über. Nimmt der dritte Teilnehmer das Gespräch an, so ist die Konferenz vorbereitet und alle drei Teilnehmer können zusammengeschaltet werden. Im Gegensatz zur Weiterleitung bleibt allerdings der Vermittler in diesem Falle online, da er ja an dem weiteren Gespräch teilnimmt. Als Erweiterung wäre hier noch die Einführung einer weiteren Stelle *OnConference* sinnvoll, damit das System von Teilnehmer 2 entsprechend unterschiedlich reagieren kann, je nachdem ob es nur mit einem Teilnehmer verbunden ist (*Online_2)* oder aber an einer Konferenz teilnimmt. Das Netz wird dadurch aber noch komplexer und da ein Telefonsystem in der Regel diese Unterscheidung ebenfalls nicht vornimmt, erscheint diese Erweiterung nicht unbedingt notwendig.

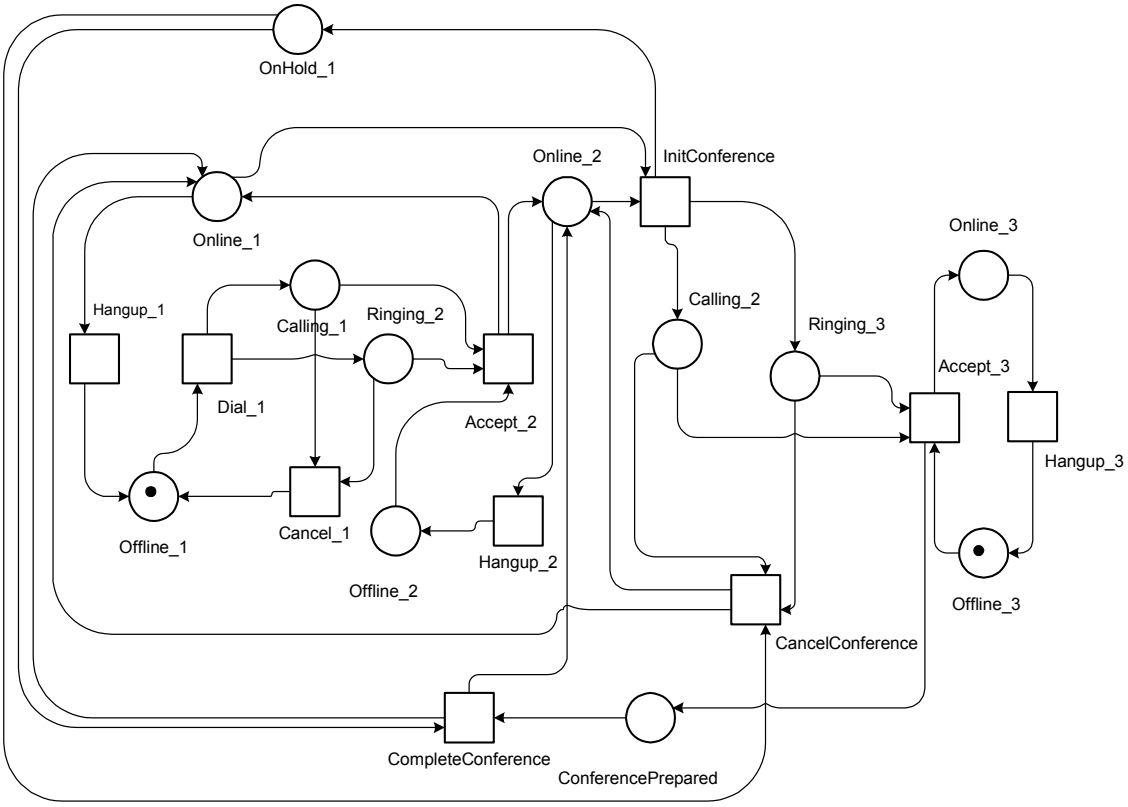


Abbildung 6-7 Petrinetz Konferenz

Die Analyse des Netzes einer Konferenz bringt ebenfalls kein kritisches Verhalten mit sich, da hier eine enge Verwandtschaft zu dem Fall einer Weiterleitung besteht

6.2.5 Konferenz mit Synchronisation

Die Synchronisation des Zugriffs auf gemeinsame Ressourcen ist im Falle einer Konferenz im Gegensatz zur Weiterleitung nicht nur auf einen Teilablauf beschränkt, sondern muss während der gesamten Konferenz gewährleistet sein, da ja der Vermittler ebenfalls wieder am Gespräch teilnimmt. Im Gegensatz zur Weiterleitung können hier also zwei Partner abwechselnd exklusiven Zugriff auf Ressourcen erhalten. Wie in Abbildung 6-8 dargestellt ist demnach der exklusive Zugriff der beiden Teilnehmer, der durch die Transitionen *BeginWork_x* sowie *EndWork_x* eingerahmt wird, von dem Einleiten und Herstellen einer Konferenz abgekoppelt. Sie können beide aus dem Online-Zustand heraus im gegenseitigen Ausschluss Ressourcen reservieren und müssen diese Reservierung wieder freigeben, ehe sie weitere Telefonaktionen

durchführen können. Wie schon in den anderen Abläufen wird der erste Teilnehmer als externer Anrufer ohne Zugriff auf zentrale Daten angesehen und somit für die Synchronisation nicht berücksichtigt.

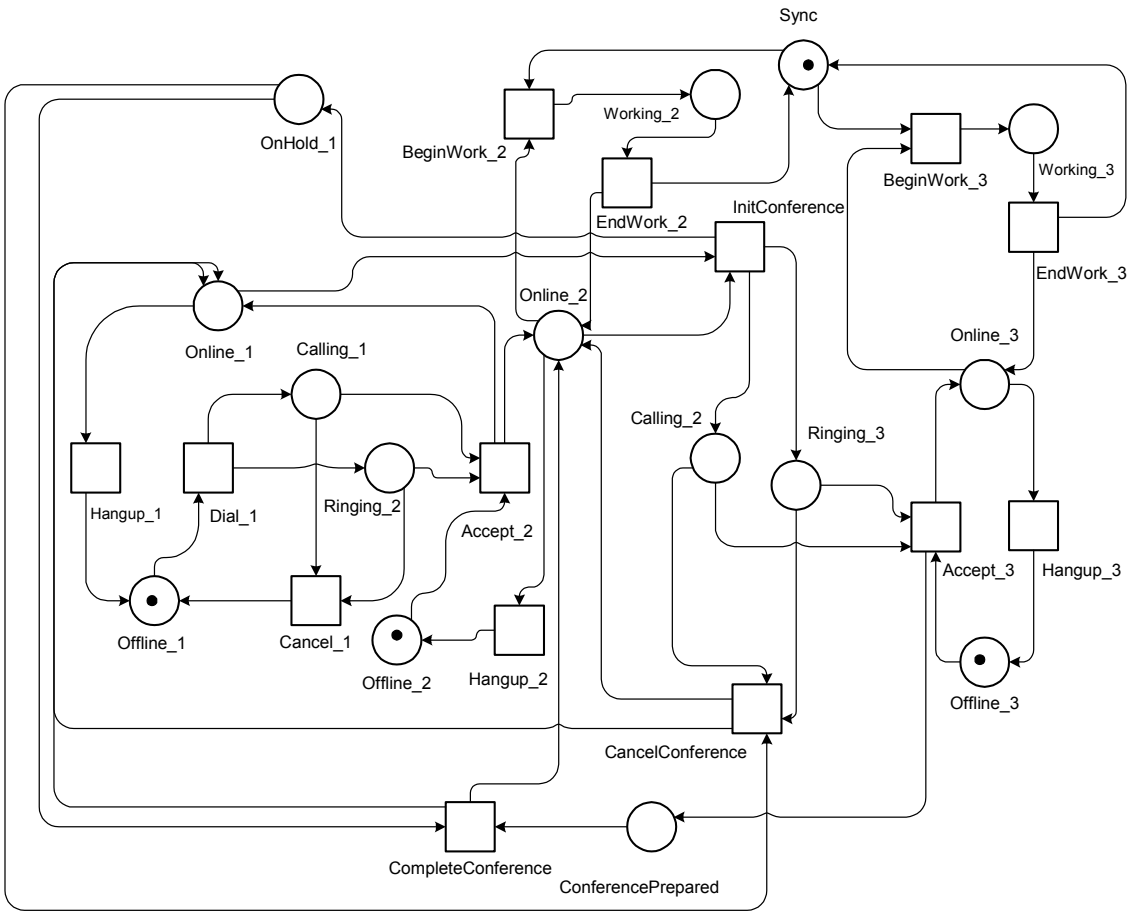


Abbildung 6-8 Petrinetz Konferenz mit Synchronisation

Die Analyse des Netzes zeigt neben der nun schon relativ hohen Komplexität lediglich eine weiterhin hohe Anzahl von möglichen Konflikten.

6.3 Zustandsautomat

Nachdem nun die wesentlichen Abläufe des verteilten Systems anhand von Petrinetzen beschrieben wurden, kann daraus der für die lokale CTI-Anwendung relevante Zustandsautomat abgeleitet werden. Durch die Modellierung für das gesamte Framework wurde ja auch die Kopplung der jeweils lokalen Zustandsautomaten berücksichtigt und dadurch ergaben sich unter anderem auch neue Zustände, die bei einer reinen lokalen Betrachtung eines CTI-Clients so nicht auftauchen. Dennoch benötigt eine derartige Anwendung darüber hinaus einen lokalen Zustandsautomat, um ihre Funktionalität wie etwa die graphische Benutzeroberfläche darauf abstimmen zu können. In diesem Abschnitt wird deshalb auf der eben erarbeiteten Basis durch Abbildung der Petrinetze auf nur eine Systemkomponente ein lokaler Zustandsautomat entworfen.

An dieser Stelle ist nochmals die Frage aufzunehmen, welche der angesprochen Modelle eines Automaten konkret für die Modellierung geeignet sind. Wie in Abschnitt 4.1 bereits beschrieben besteht der Unterschied beider Modelle darin, dass das Modell von Mealy Ausgaben bzw. Aktionen an Zustandsübergänge, das Modell von Moore an Zustände selbst knüpft. Betrachtet man die modellierten Vorgänge wie etwa das Eintreffen eines Anrufes oder eine Weiterleitung, so erkennt man, dass Aktionen bzw. Ausgaben in dieser Stufe noch nicht berücksichtigt wurden. In dieser konkreteren Definition der Zustände können jedoch

auch diese Ausgaben des Automaten berücksichtigt und als Ereignisse interpretiert werden. Analog der Verhalten der meisten CTI-Systeme zeigen solche Ereignisse einen Zustandsübergang an. Es empfiehlt sich deshalb auch bei der Modellierung dieses konkreten Automaten, das Modell von Mealy zu übernehmen.

Abbildung 6-9 zeigt den Zustandsautomaten, den das System unterstützen muss.

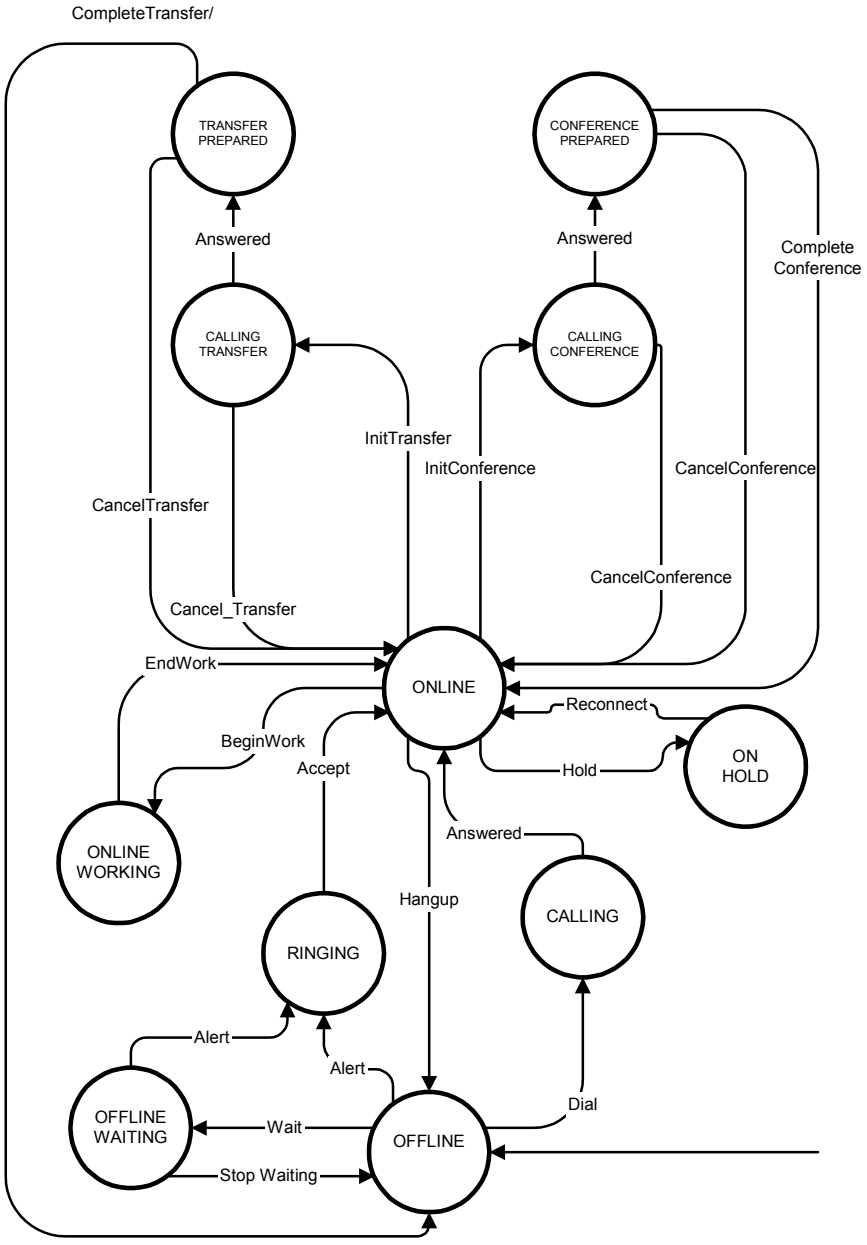


Abbildung 6-9 Zustandsautomat des Systems

Die verwendeten Zustände haben dabei im einzelnen folgende Bedeutung:

OFFLINE
Dies ist der Grundzustand des Systems, wenn kein aktueller Anruf vorliegt.

RINGING
Sobald ein Anruf ankommt und das lokale Telefon klingelt, geht das System in den Zustand RINGING über. Der Übergang in diesen Zustand wird durch ein Alert-Ereignis des CTI-Systems hervorgerufen und der Anwendung wiederum durch ein äquivalentes Ereignis gemeldet.

CALLING
Wird der Anruf durch die Anwendung selbst initiiert, geht das System in diesen Zustand über. Er zeigt an, dass die Telefonanlage versucht, die Verbindung auzubauen. Aus Sicht der CTI ist also hier eine Verbindung des Anrufs im Zustands *altering* und eine im Zustand *connected* (vgl. 2.5.1).

ONLINE
Der Zustand zeigt an, dass eine Verbindung hergestellt wurde. Beide Verbindungen des Anrufs sind im Zustand *connected*.

ONHOLD
Der Anruf wird gehalten, sofern die Anwendung dies anfordert. Dieser Zustand kann durch ein entsprechendes Ereignis *ReConnect* wieder verlassen werden, worauf das System wieder in den Zustand ONLINE übergeht.

CALLING_TRANSFER, TRANSFER_PREPARED
Wie aus der Modellierung der Vorgänge hervorging, muss der Vorgang der Weiterleitung über zwei Zwischenschritte erfolgen, die durch diese beiden Zustände reflektiert werden. Der Transfer kann nach dem ersten Schritt abgebrochen werden, wenn keine Verbindung zu einem neuen Partner zustande kommt. Das System geht dann wieder in den Online-Zustand über. Kommt die Verbindung zustande, so kann die Weiterleitung weiterhin abgebrochen oder aber durchgeführt werden, wodurch das System dann in den Offline-Zustand übergeht.

CALLING_CONFERENCE, CONFERENCE_PREPARED
Analog zur Weiterleitung geben diese Zustände die Zwischenschritte bei dem Aufbau einer Konferenz an. Wie schon in der Modellierung im vorangegangenen Abschnitt besteht der Unterschied hier darin, dass das System in jedem Falle wieder in den Online-Zustand übergeht, wenn die Konferenz zustande kommt, aber auch wenn der Vorgang abgebrochen wird.

ONLINE_WORKING
In diesem Zustand hat die Anwendung Ressourcen eines Anrufs exklusiv gesperrt. Dadurch können sich mehrere Parteien etwa bei einer Weiterleitung oder einer Konferenz bzgl. des Zugriffs auf gemeinsame Informationen synchronisieren.

OFFLINE_WAITING
Dieser Zustand zeigt an, dass die Anwendung zwar gerade keinen Anruf hat, aber auf ein bestimmtes Ereignis wartet und bis dahin für gewisse Aktionen blockiert ist. Dies ist zum Beispiel der Fall, wenn darauf gewartet wird, dass ein an einen Sprachcomputer zur Legitimation weitergeleiteter Anruf wieder zurückkommt. Der Zustand selbst ist also kein Ergebnis der vorliegenden Modellierung sondern wurde direkt aus den fachlichen Anforderungen abgeleitet.

Insgesamt lässt sich also an dieser Stelle festhalten, dass eine direkte Ableitung eines Zustandsautomaten aus den modellierten Petrinetzen möglich ist.

6.3.1 Ereignisse

Der Zustandsautomat definiert auch die Eingaben, die für die jeweiligen Zustandsänderungen maßgeblich sind. Diese Eingaben leiten sich im Framework entweder durch Aktionen des Benutzers her (z.B. BEGIN_WORK) oder aber aus Ereignissen des Telefonsystems. Da diese Ereignisse für eine nachfolgende Bewertung zu verwendender CTI-Technologien entscheidend ist, werden sie an dieser Stelle nochmals zusammengefasst.

Ereignis	Beschreibung
ALERT	Neuer Anruf liegt an
CALLING	Es wird versucht, eine Verbindung aufzubauen
ANSWERED	Ein Anruf wurde von Gegenstelle angenommen
ACCEPT	Ein Anruf wird lokal angenommen
HANGUP	Eine Verbindung wurde getrennt
HOLD	Ein Anruf wird gehalten
RECONNECT	Ein gehaltener Anruf ist wieder verbunden

Tabelle 6-1 Notwendige Ereignisse des Telefonsystems

6.4 Anwendungsfälle

Aus den bisher spezifizierten Abläufen und den eingangs beschriebenen Abläufen in einem Call Center sowie einem CTI-System lassen sich nun die konkreten Anwendungsfälle (Use Cases) spezifizieren und damit die Funktionalität des Frameworks festlegen. Diese lassen sich in drei Kategorien einteilen. Die erste beinhaltet dabei die wesentlichen Dienste, die die übliche Steuerung der Telefonanlage erlaubt während die zweite erweiterte, für die Prozesse in einem Call Center zugeschnittene Funktionen bereitstellt. Darüber hinaus gibt es noch einige administrative Anwendungsfälle, die in einer dritten Kategorie definiert werden. Nachfolgend werden die einzelnen Use Cases jeder Kategorie genauer spezifiziert.

6.4.1 Basisdienste

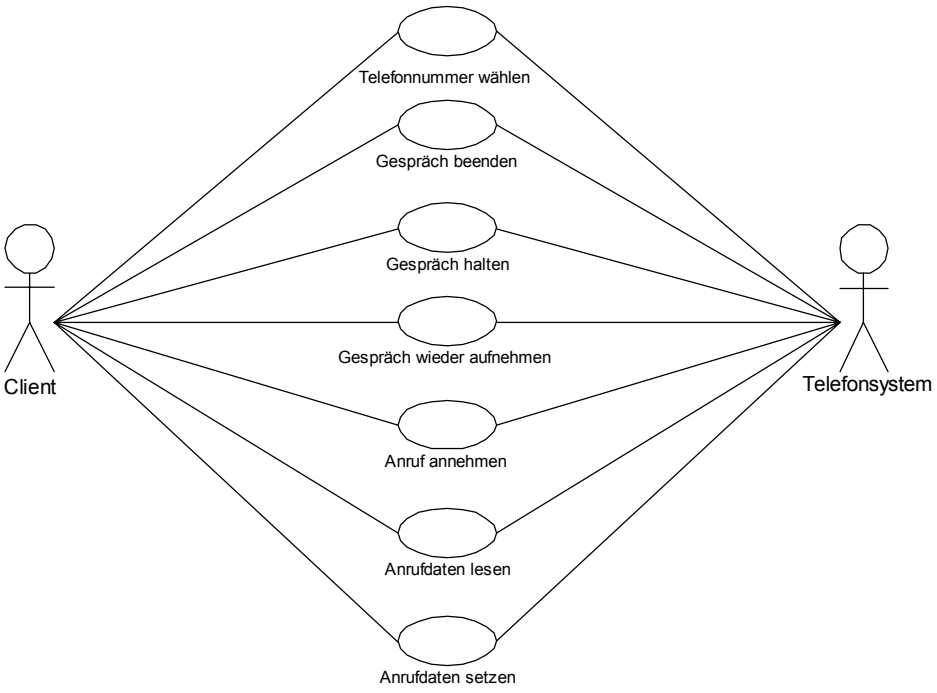


Abbildung 6-10 Use Cases für Basisdienste

Diese grundlegenden Anwendungsfälle umfassen die einfache Kontrolle eines Telefonanrufs, wie sie auch ein übliches Telefon bietet. Neben dem Annehmen eines Anrufs sowie dem Auflegen gehören hierzu noch Funktionen wie das Wählen einer neuen Nummer oder aber das Halten eines Anrufes.

`Telefonnummer wählen`	
Funktionalität	Stellt eine Verbindung zur angegebenen Telefonnummer her
Voraussetzung	System befindet sich im Zustand OFFLINE
Erfolgsfall	• Anruf läutet bei anderem Teilnehmer • System ist im Zustand DIALING
Fehlerfälle	• Nummer besetzt

`Anruf annehmen`	
Funktionalität	Nimmt ein ankommendes Gespräch entgegen
Voraussetzung	System befindet sich im Zustand RINGING
Erfolgsfall	Telefonverbindung ist hergestellt
Fehlerfälle	• System in ungültigem Zustand • Fehler CTI-System

`Gespräch halten`	
Funktionalität	Hält das aktuelle Gespräch
Erfolgsfall	Aktuelles Gespräch wird gehalten, Leitung ist stumm
Voraussetzung	System befindet sich im Zustand ONLINE
Fehlerfälle	• Ungültiger Zustand • Fehler CTI-System

`Gespräch wieder aufnehmen`	
Funktionalität	Stellt Verbindung zu gehaltenem Gespräch wieder her
Voraussetzung	System befindet sich im Zustand HOLD
Erfolgsfall	Verbindung ist wieder hergestellt
Fehlerfälle	• Ungültiger Zustand • Kein Gespräch mehr vorhanden

`Gespräch beenden`	
Funktionalität	Beendet das aktuelle Gespräch
Voraussetzung	System befindet sich im Zustand ONLINE bzw. HOLD
Erfolgsfall	Verbindung ist beendet
Fehlerfälle	• Ungültiger Zustand

`Anrufdaten lesen`	
Funktionalität	Liest Informationen, die zu dem Anruf gespeichert sind
Voraussetzung	System befindet sich im Zustand ONLINE bzw. HOLD
Erfolgsfall	Informationen können gelesen werden
Fehlerfälle	• Ungültiger Zustand • Kein Zugriff auf Informationen

`Anrufdaten setzen`	
Funktionalität	Speichert Informationen zu einem Anruf
Voraussetzung	System befindet sich im Zustand ONLINE bzw. HOLD
Erfolgsfall	Informationen werden abgelegt
Fehlerfälle	• Ungültiger Zustand • Kein Zugriff auf Informationen

6.4.2 Erweiterte Dienste

Die erweiterte Funktionalität des Systems spricht besonders die Anforderungen in einem Call Center an. Sie umfasst dabei Funktionen, um einen bestehenden Anruf weiterleiten oder aber eine Konferenz einleiten zu können. Darüber hinaus bietet das Framework noch sehr spezielle Dienste wie etwa die direkte Weiterleitung, bei der der Anruf sofort in eine Warteschlange eingereiht wird, bis der gewünschte Teilnehmer verfügbar ist. Notwendiger Parameter ist hier wie auch bei der Einleitung einer Konferenz immer eine Telefonnummer, die entweder einen einzelnen Teilnehmer oder aber wie in vielen Call Centern üblich auch eine Gruppe von Mitarbeitern identifizieren kann. Diese Unterscheidung ist dabei jedoch für die Anwendung und im Grunde auch für das Framework transparent, da sie von der Telefonanlage bzw. dem Switch-System umgesetzt und auch dort konfiguriert wird.

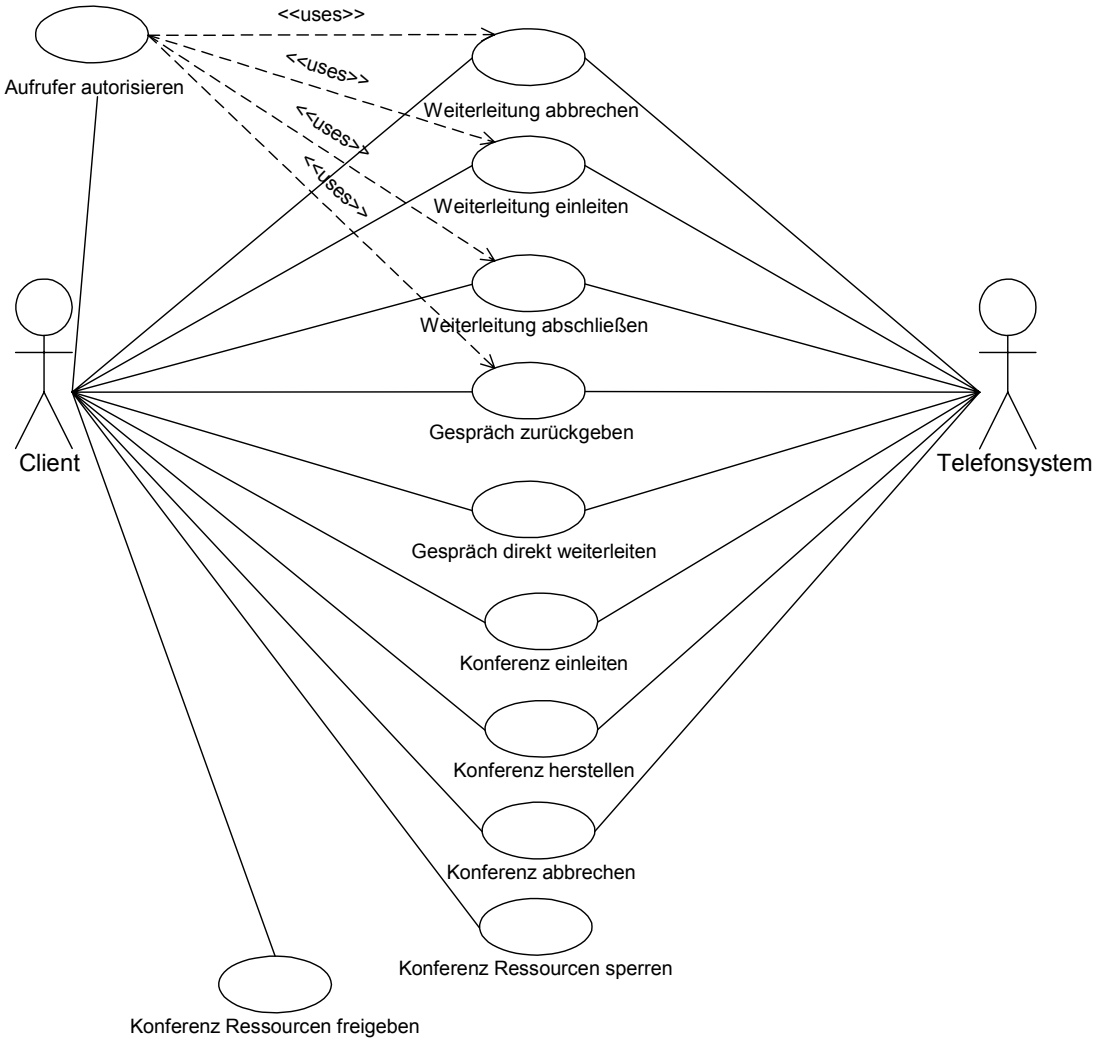


Abbildung 6-11 Use Cases für erweiterte Dienste

Auch hier werden die konkreten Use Cases nachfolgend aufgeführt.

Weiterleitung

`Weiterleitung einleiten`	
Funktionalität	Stellt eine neue Verbindung für eine Weiterleitung her
Voraussetzung	System befindet sich im Zustand ONLINE
Erfolgsfall	• Erstes Gespräch wird gehalten • Zweiter Anruf läutet bei weiterem Teilnehmer
Fehlerfälle	• Ungültiger Zustand • Teilnehmer nicht erreichbar

`Weiterleitung abschließen`	
Funktionalität	Reicht ein Gespräch an einen neuen Teilnehmer weiter
Voraussetzung	System befindet sich im Zustand CALLING_TRANSFER
Erfolgsfall	• Die beiden anderen Teilnehmer sind verbunden • Das lokale System hat keine Verbindung mehr
Fehlerfälle	• Ungültiger Zustand

`Weiterleitung abbrechen`	
Funktionalität	Bricht eine Weiterleitung ab
Voraussetzung	System ist im Zustand CALLING_TRANSFER bzw. TRANSFER_PREPARED Ursprüngliche Verbindung ist noch vorhanden
Erfolgsfall	• Verbindung zu zweitem Teilnehmer wird getrennt • System ist wieder mit erstem Teilnehmer verbunden
Fehlerfälle	• Ungültiger Zustand • Ursprüngliche Verbindung nicht mehr vorhanden

`Gespräch direkt weiterleiten`	
Funktionalität	Direkte Weiterleitung
Voraussetzung	System ist im Zustand ONLINE
Erfolgsfall	• Das lokale System hat keine Verbindung mehr • Die beiden anderen Teilnehmer sind verbunden bzw. das Telefonsystem versucht, den Anruf durchzustellen
Fehlerfälle	• Ungültiger Zustand

`Gespräch zurückgeben`	
Funktionalität	Leitet ein Gespräch an den Teilnehmer weiter, von dem es übergeben wurde
Voraussetzung	System ist im Zustand ONLINE, Gespräch wurde weitergeleitet und Quelle ist bekannt
Erfolgsfall	Gespräch ist wieder bei ursprünglichen Teilnehmern
Fehlerfälle	• Fehler im CTI-System

`Anrufer autorisieren`	
Funktionalität	Leitet ein Gespräch an einen Sprachcomputer weiter und gibt es nach Legitimation mit geändertem Status wieder zurück
Voraussetzung	System ist im Zustand ONLINE und Sprachcomputer ist verfügbar
Erfolgsfall	Gespräch wird zurück vermittelt und Benutzer ist nun legimitiert
Fehlerfälle	• Keine erfolgreiche Legitimation • Keine Rückgabe des Gesprächs

Konferenz

`Konferenz einleiten`	
Funktionalität	Wählt eine Telefonnummer, um den Teilnehmer zu einer Konferenz hinzuzufügen
Voraussetzung	System befindet sich im Zustand ONLINE
Erfolgsfall	• Die bereits bestehenden Verbindungen werden gehalten • Neuer Anruf läutet bei angegebenem Teilnehmer
Fehlerfälle	• Teilnehmer nicht erreichbar • Ungültiger Zustand

`Konferenz abbrechen`	
Funktionalität	Bricht den Vorgang ab, einen neuen Teilnehmer einer Konferenz hinzuzufügen
Voraussetzung	System befindet sich im Zustand CALLING_CONFERENCE bzw. CONFERENCE_PREPARED
Erfolgsfall	• Zuletzt aufgebaute Verbindung wird getrennt • System ist wieder mit alten Teilnehmern in Konferenz verbunden
Fehlerfälle	• Ungültiger Zustand

`Konferenz herstellen`	
Funktionalität	Fügt den aktuellen Teilnehmer der Konferenz hinzu
Voraussetzung	System befindet sich im Zustand CONFERENCE_PREPARED
Erfolgsfall	Alle Teilnehmer befinden sich in einer Konferenz
Fehlerfälle	• Fehler im CTI-System • Ungültiger Zustand

`Konferenz Ressourcen sperren`	
Funktionalität	Setzt eine Sperre für den exklusiven Zugriff auf Ressourcen
Voraussetzung	System hat Zugriff auf Ressourcen
Erfolgsfall	Ressourcen sind exklusiv gesperrt
Fehlerfälle	• Ressourcen sind bereits gesperrt • Systemfehler bei Sperre

`Konferenz Ressourcen freigeben`	
Funktionalität	Gibt belegte Ressourcen wieder frei
Voraussetzung	Ressourcen wurden bereits belegt
Erfolgsfall	Ressourcen sind freigegeben
Fehlerfälle	• Systemfehler bei der Freigabe

Die Dienste „Gespräch zurückgeben" (*ReturnTransfer)* sowie „Anrufer autorisieren" (*AuthorizeCall)* sind Beispiele von Funktionen, die über die allgemeine CTI-Funktionalität hinaus gehen und ganz bestimmte Geschäftsprozesse eines Call Centers ansprechen. Sie spiegeln die in 3.2.2 angesprochenen Anforderungen wider.

Im ersten Fall muss das System während einer Weiterleitung ausreichend Informationen speichern, um den Anruf wieder zurück vermitteln zu können.

Der Dienst *AuthorizeCall* ist ein Beispiel für eine synchrone Weiterleitung. Durch diese Methode wird der Anruf an eine vordefinierte Nummer eines Sprachcomputers weitergeleitet. Dieser kann dann anhand der mitgelieferten Daten (Kundenname, Kundennummer etc.) den Anruf über Abfrage einer Geheimnummer identifizieren und dann an den ursprünglichen Anrufer zurück geben (*ReturnTransfer)*. Das System sorgt

dabei dafür, dass der Mitarbeiter in der Zwischenzeit für weitere Anrufe blockiert ist. Sobald der Anruf wieder zurück kommt, kann anhand der Informationen der aktuelle Legitimationsstatus abgefragt werden. Kann der Anruf nicht mehr zum Mitarbeiter zurück vermittelt werden, zum Beispiel weil der Anrufer selbst auflegt, so muss dieser Fall vom System erkannt und über ein entsprechendes Ereignis der Anwendung des Mitarbeiters mitgeteilt werden.

Administration

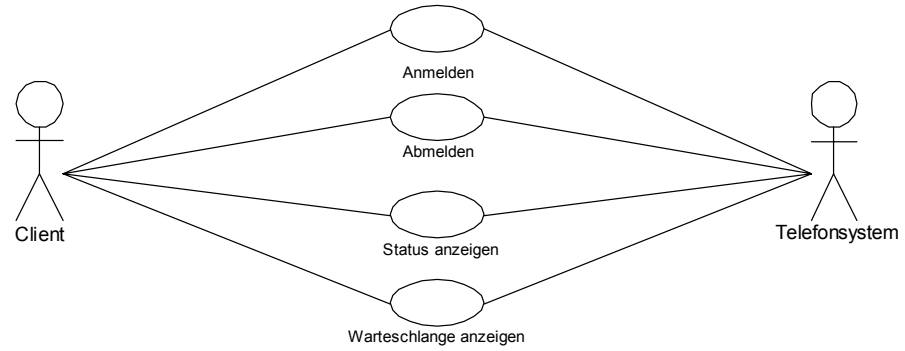


Abbildung 6-12 Use Cases für Administration

Anmelden	
Funktionalität	Anmelden des Benutzers am System
Voraussetzung	Benutzer ist noch nicht angemeldet
Erfolgsfall	Benutzer ist im System angemeldet
Fehlerfälle	• Benutzer nicht vorhanden • Keine gültige Identifikation

Abmelden	
Funktionalität	Abmelden des Benutzers vom System
Voraussetzung	Benutzer bereits angemeldet
Erfolgsfall	Benutzer ist abgemeldet
Fehlerfälle	• Benutzer kann wg. Systemfehler nicht abgemeldet werden

Status anzeigen	
Funktionalität	Statusinformationen über den Zustand des Benutzers
Voraussetzung	Benutzer ist angemeldet
Erfolgsfall	Informationen über aktuellen Zustand werden angezeigt
Fehlerfälle	• Informationen sind nicht verfügbar • Keine Verbindung zur Telefonanlage

Warteschlange anzeigen	
Funktionalität	Anzeige der Anrufe, die in der Warteschlange stehen
Voraussetzung	Benutzer ist angemeldet
Erfolgsfall	Anzahl der wartenden Anrufe wird angezeigt
Fehlerfälle	• Informationen sind nicht verfügbar • Keine Verbindung zur Telefonanlage

6.5 Verknüpfung eines Anrufs mit Daten

In Kapitel 3.2.1 wurde die Verbindung eines Anrufs mit individuellen Informationen als wesentliche Anforderung definiert. Ziel muss dabei sein, die in den meisten CTI-Systemen für diese Funktion verfügbaren Mechanismen zu nutzen und einen höherwertigen Dienst zur Verfügung zu stellen.

Die Implementierung muss demnach als erstes gewährleisten, dass die in derzeitigen Lösungen übliche Integration von Daten und Anruf ohne größere Restriktionen umgesetzt wird. Hier muss sichergestellt werden, dass diese Daten zusammen mit dem Anruf der CTI-Anwendung des Arbeitsplatzes genau dann zur Verfügung stehen, wenn dieser Anruf an das jeweils zugeordnete Telefon durchgestellt wird. Was jedoch die Definition bzw. den Zugriff auf diese Informationen angeht, so ist hier ein objektorientierter Ansatz umzusetzen, der die Daten nicht als reine Struktur sondern im Sinne von verteilten Objekten behandelt. Erster Schritt dazu ist die Definition eines allgemeinen Anruf-Objekts. Wie in Abbildung 6-13 dargestellt, empfiehlt sich dabei eine Aufteilung in eine Basisversion sowie ein davon abgeleitetes, erweitertes Objekt.

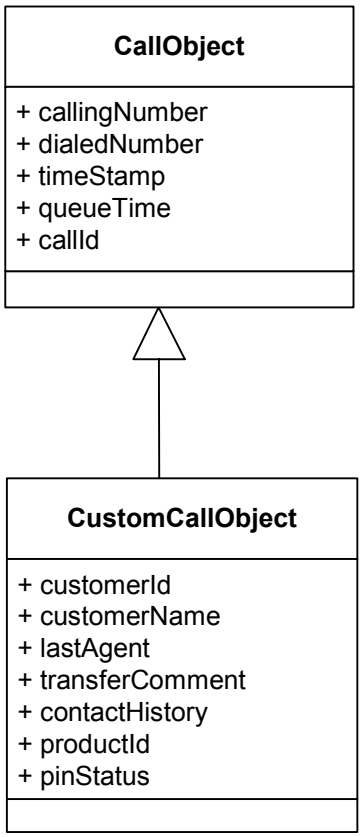


Abbildung 6-13 Anrufobjekt

Für allgemeine Daten der Telefonanlage bzw. des CTI-Systems nimmt das Basisobjekt Informationen eines Anrufs wie etwa die Nummer des Anrufers auf. Um den Anforderungen unterschiedlicher CTI-Anwendungen gerecht werden zu können, kann davon abgeleitet dann ein individuell zugeschnittenes Objekt definiert werden, das speziell abgestimmte Daten wie etwa eine Kunden- oder Artikelnummer beinhaltet.

Aufgabe des neuen Frameworks ist es nun, dieses Objekt mit einem Anruf derartig zu verbinden, dass es etwa bei einer Weiterleitung immer den jeweils betroffenen Anwendungen zur Verfügung steht. Außerdem müssen geeignete Zugriffsmechanismen bereitgestellt werden, so dass die gespeicherten Informatio-

nen verändert bzw. ausgelesen werden können. In Sonderfällen wie etwa einer Konferenz muss dieser Zugriff dann auch entsprechend synchronisiert werden.

6.6 Einsatz von Petrinetzen

Das Konzept der Petri-Netze hat sich in vielen Anwendungsfällen zur Modellierung verteilter Systeme und Algorithmen bewährt und damit ist die Idee naheliegend, es auch auf den Bereich telefonbasierter Abläufe anzuwenden. Die Eignung von Petrinetzen ist dabei nicht ohne weiteres zu erwarten, da durch Telefonanlagen ganz eigene Charakteristiken von verteilten Prozessen vorgegeben werden. So findet man auch in der Literatur zwar verschiedene Zustandsautomaten im Rahmen von CTI-Systemen oder die in Abschnitt 2.5 eingeführte Notation, die aber nicht die Mächtigkeit und den formalen Charakter von Petrinetzen besitzen. Somit besteht an dieser Stelle durchaus Bedarf an geeigneten Modellierungskonzepten. So war der Ansatz dabei, mit einer einfachen Version der B/E-Netze eine Modellierung durchzuführen und wie die nachfolgenden Punkte erläutern zeigt sich, dass auch das vergleichsweise einfache Konzept dieser Art von Petrinetzen für die Modellierung von Abläufen eines Telefonsystems durchaus geeignet ist. Dazu kommt, dass durch die graphische Notation Petrinetze ein sehr übersichtliches Bild auch komplexer Abläufe geben und schon dadurch Bewertung und Analyse erleichtern.

Mit Petrinetzen schlecht zu modellieren sind spontan auftretende Ereignisse wie etwa das Auflegen eines Kommunikationspartners. Technisch orientierte Spezifikationen würden an dieser Stelle den Übergang zumindest einiger beteiligter Komponenten von ONLINE nach OFFLINE fordern, so wie es innerhalb der Telefonanlage durch ein elektronisches Signal realisiert wird. Dies ist mit Hilfe der B/E-Netze so nicht zu modellieren, da spontan auftretende und gleichzeitig erzwungene Zustandsübergänge zumindest bei dieser Version nicht vorgesehen sind. Die Spezifikation hat deshalb in diesem Zusammenhang ganz bewusst die Sicht der CTI-Anwendung eingenommen, bei der die einzelnen Komponenten selbst und individuell derartige Zustandsübergänge durchführen. Deshalb erscheint eine Erweiterung des Petrinetz-Modells oder die Verwendung mächtigerer Varianten nicht notwendig.

Entscheidender Vorteil ist jedoch, dass durch die mathematisch sehr exakte Definition eine methodische Analyse und Simulation mit Hilfe von Software möglich ist, die zuverlässig Probleme wie Lebendigkeit, Kontaktsituationen oder Konflikte aufzeigt. Dies ist insbesondere bei B/E-Netzen möglich, die ihrer Komplexität wegen eine automatische und softwaregestützte Analyse vereinfachen. Einige der daraus gewonnenen Ergebnisse und ihre Bedeutung werden nachfolgend kurz erläutert.

Lebendigkeit

Die Analyse hinsichtlich Lebendigkeit ist für den Fall telefonbasierter Abläufe eher weniger interessant, da immer von einem lebendigen System ausgegangen werden kann. Wird ein Anruf beendet, so gehen nämlich alle teilnehmenden Systeme in jedem Fall wieder in den Zustand OFFLINE. Deshalb sind in einem realen System normalerweise auch keine Verklemmungen möglich. Die Analyse dient aber in jedem Fall der Überprüfung der Korrektheit des modellierten Systems.

Analog zu vorher gilt jedoch, dass wenn die gesamte Steuerung über die Anwendungssoftware erfolgt, deren Oberfläche zum Beispiel zustandsorientiert ist und man nicht von erzwungenen Zustandsübergängen ausgehen kann, fehlende Lebendigkeit und Verklemmungen durchaus mögliche und dann kritische Situationen darstellen.

Kontaktsituationen

Sie zeigen zuverlässig Probleme des konkreten Modells und in den durchgeführten Beispielen die Notwendigkeit neuer Zustände oder Zustandsübergänge auf. Wie das erste Beispiel gezeigt hat, ist die unzureichende Kopplung zweier Systeme ein Grund für auftretende Kontakte, wenn dabei Markierungen (z.B. an der Stelle RINGING) nicht sorgfältig für alle Fälle betrachtet werden. Diese Eigenschaft ist ein wesentlicher Vorteil, den der Modellierungsansatz mit sich bringt. Im Vergleich etwa zur eher technisch orientierten CTI-Spezifikationen mit der in Abschnitt 2.5 dargestellten Notation zeigt sich, dass hier wesentlich genauer auf die Anforderungen der Anwendungsentwicklung eingegangen werden kann.

Konflikte

Konflikte zeigen durch das Eintreten konkurrierender Ereignisse die Stellen des Systems auf, an denen Nichtdeterminismus auftritt. Diese Eigenschaft ist bei Telefonsystemen und darauf aufbauenden CTI-Anwendungen üblich, da hier in der Regel das nichtdeterministische Verhalten der telefonierenden Teilnehmer modelliert wird. Die Analyse von B/E-Netzen kann hier eine gute Aussage über die mögliche Anzahl und die jeweiligen Voraussetzungen solcher Konflikte geben und auch die Situationen aufzeigen, in denen kein Nichtdeterminismus möglich ist.

Bereits diese kurze Übersicht zeigt, dass die Analyse von B/E-Netzen einige sehr aufschlussreiche Details über das modellierte System hervorbringen kann, die im Zusammenhang mit CTI-basierten Anwendungen durchaus von Bedeutung sind. Nun stellt sich allerdings noch die Frage, welche Abbildung der so definierten Netze auf das IT-System möglich sind. Dazu ist zu sagen, dass hier die Zustände und Zustandsübergänge im Vordergrund stehen und durch die formale Spezifikation der Zustandsautomat des Systems an Qualität gewinnt. Insgesamt lässt sich also der Schluss ziehen, dass Petrinetze für das Einsatzgebiet von CTI-Anwendungen eine äußerst nützliche Modellierungsmethode darstellen und sich gegenüber vorhandenen Ansätzen vor allem durch die Berücksichtigung nebenläufiger Abläufe abheben. Da sich die grundlegenden Abläufe mit einfachen B/E-Netzen darstellen lassen, ist hier die Analyse und Simulation ein weiterer positiver Faktor. Dadurch wird es möglich, einen qualitativ höherwertigen Zustandsautomaten bereitzustellen, der Anwendungen auch bei komplexeren Abläufen ausreichend Unterstützung bietet und somit die individuelle Definition innerhalb einer CTI-Anwendung überflüssig macht.

Abschließend sei an dieser Stelle noch der Vergleich mit konkreten CTI-Projekten herangezogen. Hier kann man feststellen, dass formale Methoden eher selten eingesetzt werden und die Definition der Zustände und Modellierung verteilter Abläufe dabei individuell und getrieben durch die Vorgaben der Basissoftware erfolgt. Ein Vergleich mit dem hier vorgestellten Ansatz zeigt jedoch, dass sich die Ergebnisse zum Teil sehr ähneln. So weisen beispielsweise die CTI-Komponenten einer konkreten Bankanwendung ebenso Zwischenzustände wie TRANSFER_PREPARED auf, die so durch das CTI-System nicht modelliert werden (vgl. [iSYS98]). Dies ist Beleg dafür, dass der Einsatz von Petrinetzen zur Modellierung durchaus auch praxisrelevante Ergebnisse liefert.

Als Ausblick wäre eine interessante Anwendung der hier dargestellten Modellierung mit Petrinetzen die Durchführung von Performanceanalysen, um die Leistungsfähigkeit eines Call Center zu bewerten. Dazu ist eine Erweiterung in Richtung zeitbehaftete Petrinetze Voraussetzung, die aber relativ problemlos durchzuführen wäre. Abbildung 6-14 zeigt ein Beispiel eines solchen Netzes, bei dem entscheidende Zustände wie CALLING oder ONLINE mit Zeitschranken versehen sind. Auf dieser Basis sind nun verschiedene Optimierungsmodelle denkbar, etwa um die Effizienz und mittlere Auslastung zu steigern.

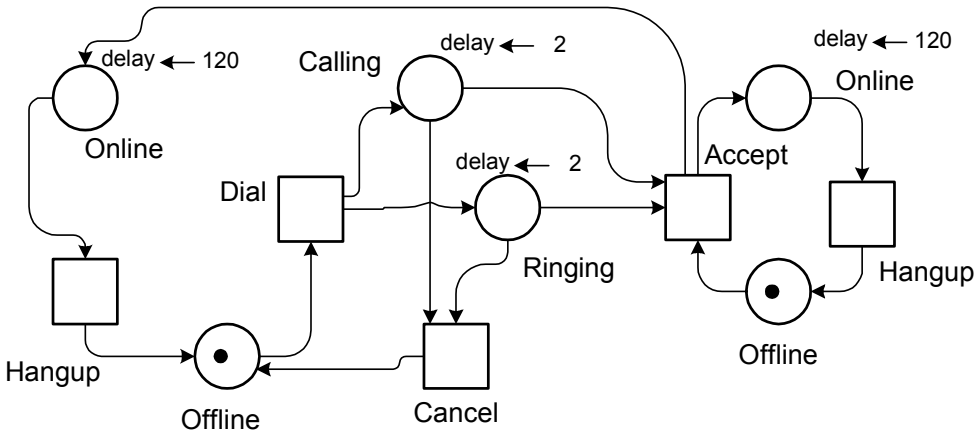


Abbildung 6-14 Beispiel eines zeitbehafteten Petrinetzes

6.7 Zwischenergebnis

Die vorliegende Spezifikation ist als erster Schritt hin zu einem Framework auch ein erstes Ergebnis, das an dieser Stelle hinsichtlich der grundlegenden Ziele kurz diskutiert werden soll. Eines davon besteht darin, CTI-Anwendungen aus Sicht der verteilten Anwendungsentwicklung zu betrachten und mit Hilfe bewährter Methoden ein möglichst allgemeines Modell zu entwerfen. Der Einsatz von Petrinetzen hat gezeigt, dass dieses Konzept auch für die Modellierung von CTI-Prozessen durchaus geeignet ist und vor allem durch den formalen Charakter Vorteile hinsichtlich Analyse und Bewertung hat. Es konnte durch diese Modellierung ein Zustandsautomat definiert werden, der gegenüber den Vorgaben von CTI-Systemen umfangreicher aber auch allgemeingültiger ist. Beide Ergebnisse belegen also die Vermutung, dass mit Hilfe von bewährten Methoden ein allgemeineres Modell zu erreichen ist. Auch die Spezifikation der notwendigen Funktionalität mit Hilfe von Use Cases sowie die Definition eines allgemeinen Anrufobjekts sind in dieser Phase noch weitgehend unabhängig von bestehenden CTI-Anwendungen und CTI–Technologien. Somit ist dieses Zwischenergebnis im Einklang mit der grundlegenden Zielsetzung. Wie gut dieser noch allgemeingültige Ansatz auf ein konkretes Framework abgebildet werden kann, muss die nachfolgende Konzeption zeigen.

7 KONZEPTION

Nachdem nun mit CORBA, JTAPI und Java die wesentlichen Basissysteme definiert und somit die Technologie festgelegt ist, auf deren Basis das Framework implementiert werden kann, ist der folgende Schritt und damit Inhalt dieses Abschnitts die konkrete Konzeption des IT-Systems. Dazu ist die bereits erarbeitete Spezifikation hinsichtlich einer konkreten Realisierung weiter zu verfeinern und insbesondere ein Vorschlag für eine mögliche Systemarchitektur zu entwerfen. Weiteres Ziel dieses Schrittes ist es, die entscheidenden Softwarekomponenten für eine anschließende Implementierung ausreichend zu beschreiben.

In diesem Abschnitt werden ausgehend von einer allgemeinen Basisarchitektur die Hauptkomponenten des Frameworks konzipiert und dabei auch unterschiedliche Lösungsansätze aufgezeigt. Diese Komponenten werden dann durch ein Objektmodell weiter verfeinert und in eine konkrete Systemarchitektur integriert. Abschließend wird anhand ausgewählter Anwendungsfälle das Zusammenspiel der einzelnen Komponenten dargelegt und somit die Tragfähigkeit des Konzepts überprüft.

7.1 Basisarchitektur

Ausgangspunkt für eine Konzeption ist eine allgemeine Architektur, die die in den vorangegangenen Kapiteln beschriebene Funktionalität sowie die Basistechnologien berücksichtigt. In diesem Zusammenhang wurde CORBA und JTAPI ausgewählt, um einerseits die Telefonfunktionalität nutzen zu können und andererseits eine Middleware für die Realisierung eines verteilten Systems zur Verfügung zu haben. Darauf aufsetzend sind nun verschiedene Architekturansätze denkbar, die in den folgenden zwei Abschnitten genauer beschrieben werden.

7.1.1 Java/CORBA-Telephony-Framework

Der erste Architekturansatz besteht in der Einbettung einer auf JTAPI basierenden CTI-Anwendung in ein CORBA-Framework, wie in Abbildung 7-1 skizziert. Das bedeutet, dass die CTI-Grundfunktionalität bereits über eine vorhandene JTAPI-Implementierung zur Verfügung gestellt wird. Die Erweiterungen des Frameworks beziehen sich demnach hauptsächlich auf die Unterstützung eines flexiblen Call Objects sowie der Bereitstellung eines mächtigeren Zustandsautomaten. Es ist dazu eine Anbindung des Frameworks an JTAPI zu implementieren, so dass darüber genutzte Funktionen (z.B. *CreateCall()*) innerhalb des Frameworks die entsprechenden Aktionen auslösen. Welche Möglichkeiten existieren, hier selbst definierte Module zu integrieren, ist dabei noch zu untersuchen.

Die Vorteile dieses Ansatzes liegen vor allem in der direkten Nutzung von JTAPI. Da diese Schnittstelle grundlegende CTI-Vorgänge in ihrem *Call Model* bereits ziemlich vollständig beschreibt und auch Zustände wie *ALERTING* oder *IDLE* berücksichtigt, kann auf die dort spezifizierte Funktionalität zurückgegriffen werden. So erhält eine auf JTAPI aufbauende Anwendung das Ereignis eines neu eintreffenden Anrufs direkt über das Observer-/Observable-Modell von Java. Eine Abbildung dieses Mechanismus auf CORBA bedarf hier des Einsatzes eines Eventing-Dienstes. Ein weiterer Vorteil wäre die bereits spezifizierte Verknüpfung eines Anrufs mit einem Daten-Objekt.

Der Nachteil dieser Lösung liegt dagegen in der engen Verknüpfung der Gesamtarchitektur mit einer Schnittstellenspezifikation. So ist durch die Nutzung von JTAPI ebenfalls Java als Programmiersprache festgelegt. Außerdem ist zum jetzigen Zeitpunkt noch nicht abzusehen, inwieweit sich diese Spezifikation gegenüber Ansätzen wie TAPI oder TSAPI wirklich durchsetzen wird.

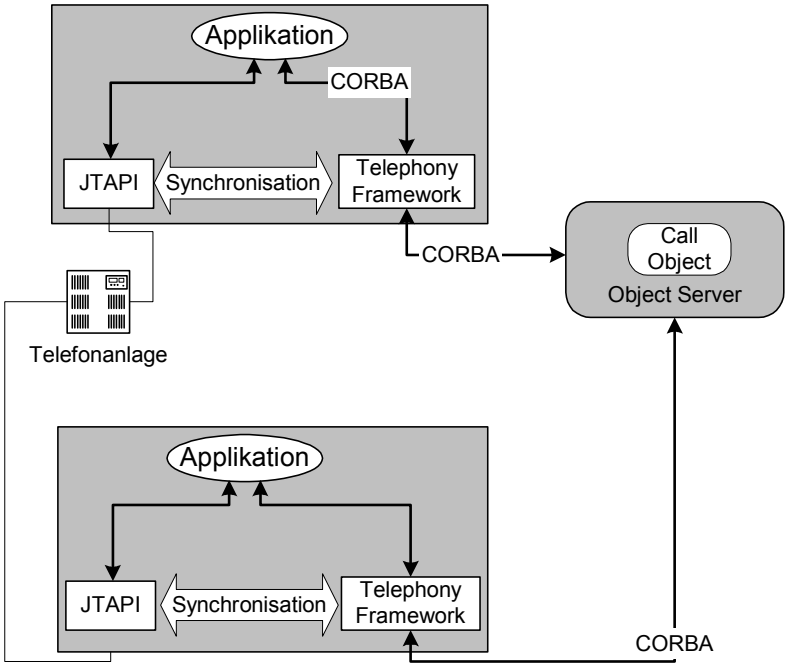


Abbildung 7-1 Java/CORBA-Telephony-Framework

Was die Implementierung erweiterter Telefoniefunktionen wie etwa einer synchronisierten Konferenz angeht, so existieren im Grunde zwei Alternativen. Zum einen könnte das Framework eine Reihe von Java-Klassen anbieten, die durch die Clientanwendung parallel zu JTAPI genutzt werden. Alternativ dazu ist es denkbar, diese Funktionen als entsprechende Erweiterungen der JTAPI-Spezifikation auszuarbeiten und zu integrieren.

7.1.2 CORBA-Framework

Ein alternativer Ansatz ist, dem Client eine reine CORBA-Schnittstelle zur Verfügung zu stellen. Unabhängig von der jeweiligen Telefonie-API setzt dieser dann lediglich auf einer Reihe von in IDL spezifizierten Diensten auf (vgl. Abbildung 7-2).

Da die Clientanwendung hier lediglich eine IDL-Schnittstelle verwendet, ist sie in der Wahl der Programmiersprache unabhängig und nicht durch die Nutzung von JTAPI auf Java festgelegt. Lediglich der lokal ablaufende Telephony-Server nutzt über eine entsprechende Java-Implementierung die CTI-Basisfunktionalität. Es ist jedoch bei diesem Ansatz die Verwendung einer anderen CTI-Middleware ebenfalls möglich. Um eine derartige Lösung möglichst universell einsetzen zu können, bieten sich hier natürlich weit verbreitete Standards wie TAPI oder TSAPI an. Ein Einsatz in der Praxis könnte jedoch ebenso die Nutzung einer herstellerspezifischen Schnittstelle zu einer Telefonanlage erfordern.

Weiterhin ist hier ein Ausbau zu einer third-party Lösung denkbar. In diesem Fall würde der Telephony-Server nicht auf dem lokalen Rechner laufen, sondern zentral und beispielsweise direkt mit dem Switch oder einem CTI-Server kommunizieren.

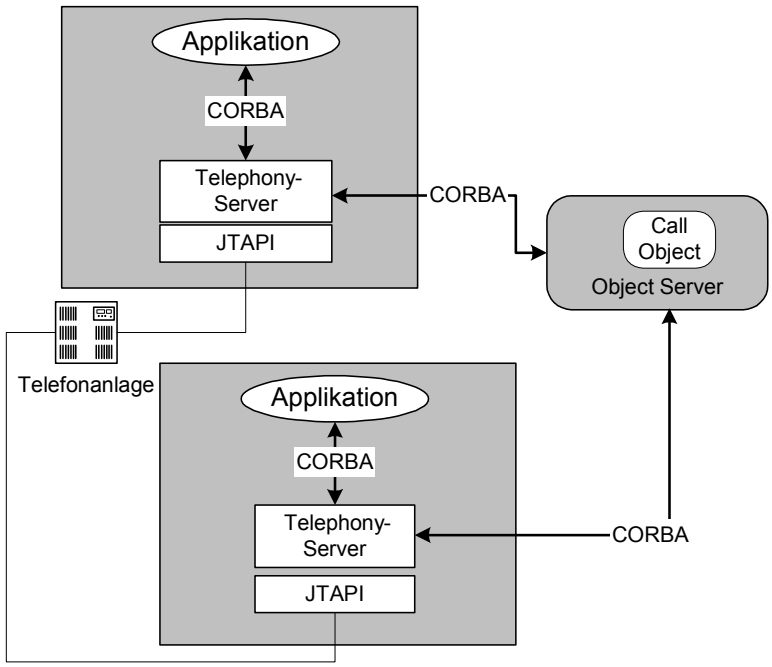


Abbildung 7-2 CORBA-Framework

Nachteil dieser Lösung ist, dass ein Teil der CTI-Funktionalität, die bereits in Spezifikationen wie JTAPI enthalten ist, nachgebildet werden muss. So müssen Ereignisse wie etwa ein neuer Anruf (ALERT) der Clientanwendung mitgeteilt werden. Für diesen Zweck bietet sich der Event-Service von CORBA an, wobei der Telephony-Server gemäß dem *push-model* der Anwendung derartige Ereignisse mitteilt (vgl. 5.2.3.1). Ähnliches gilt auch für die verschiedenen Zustände sowie deren Übergänge, die in JTAPI durch ein eigenes Objekt repräsentiert werden. Diese müssen durch das Framework auf den eigenen Zustandsautomaten abgebildet werden. Es ist jedoch zu berücksichtigen, dass die dort spezifizierten Zustände lediglich die CTI-Grundfunktionalität berücksichtigen und eventuell für die Implementierung mächtigerer Anwendungsdienste nicht ausreichen. In diesem Fall ist eine Erweiterung ohnehin notwendig. Gleiches gilt auch für die Implementierung eines CORBA-Objekts, das mit einem Anruf verknüpft wird. Dieser Mechanismus ist ebenfalls in JTAPI bereits spezifiziert, jedoch für alle Anforderungen eines Systems für Call Center nicht unbedingt ausreichend.

Die Konzeption eines CORBA-Frameworks erscheint deshalb sinnvoll und wird im Folgenden als bevorzugte Variante weiter verfolgt.

7.2 Anrufobjekt

Zentraler Bestandteil des Designs sowie der Implementierung des Systems ist die Integration eines objektorientierten verteilten Ansatzes. Einem derartigen Ansatz folgend wird der Anruf analog als zentrales *Call Object* modelliert, dessen Spezifikation ja schon durchgeführt wurde. Dies bedeutet, dass die verteilte CTI-Anwendung dem Anruf strukturierte Daten zuordnen kann. Aufgabe des Systems ist nun, diese Informationen demjenigen Client zur Verfügung zu stellen, der den aktuellen Anruf erhält. Dazu kann es auf bereits vorhandene Mechanismen des CTI-Basissystems zurückgreifen sowie diese um eigene Funktionalität erweitern, die wiederum mit Hilfe der vorhandenen Middleware umgesetzt werden kann. Neben der konkreten Definition eines solchen Objektes sind deshalb mehrere Ansätze zu untersuchen, wie die Übertragung dieser Information realisiert werden kann. Dazu werden in den nächsten Abschnitten einige Lösungsalternativen aufgezeigt und gegenübergestellt.

7.2.1 Call Object im Objektmodell

Im allgemeinen, für die Implementierung konzipierten Objektmodell wird ein Anruf im Wesentlichen durch zwei Klassen repräsentiert, die in Abbildung 7-3 dargestellt sind. Die Basisklasse CallObject enthält alle Daten, die unabhängig von individuellen Anforderungen einem Anruf zugeordnet werden können. Diese Daten beinhalten im Wesentlichen Informationen, die das zugrunde liegende CTI-System liefert. Folgende Attribute sollten dabei enthalten sein:

- CallID
  Ein für den aktuellen Anruf eindeutiger Bezeichner. Diese Information liefert jedes CTI-System und verwendet diesen auch für alle internen Prozesse, wie etwa das Weiterleiten von Ereignissen durch das lokale Netzwerk analog zu einem Anruf.
- Calling Number
  Nummer des Anrufers, die abhängig vom jeweiligen Telefonnetz als ANI oder auch CLID übergeben wird. Diese Information ist ein wichtiges Hilfsmittel zur automatischen Identifikation des Anrufers.
- Dialed Number
  Die Nummer, die der Anrufer gewählt hat. Diese Information erhält das System über den DNIS-Dienst und kann verwendet werden, um unterschiedliche Anwendungen in einem Call Center anhand unterschiedlich gewählter Nummern auszuwählen, die jedoch letztendlich an die gleichen Sprachcomputer bzw. Arbeitsplätze geleitet werden.
- Timestamp
  Zeitpunkt, an dem der Anruf das Call Center erreicht hat. Diese Information wird in der Regel von jedem System geliefert.
- Queue Time
  Zeit, die der Anrufer in der Warteschlange verbracht hat. Diese Information ist im Grunde für statistische Auswertungen gedacht aber für die Arbeit eines Mitarbeiters von hoher Bedeutung, da lange Wartezeiten für den Anrufer im Allgemeinen als sehr negativ empfunden werden.

Neben diesen Daten liefern die unterschiedlichen CTI-Systeme noch eine Reihe von individuellen Informationen, deren Integration in das Anrufobjekt gegebenenfalls sinnvoll sein kann. Beispiel solcher Daten sind ein eindeutiger Identifikator der letzten Telefonanlage oder statistische Informationen des zentralen CTI-Servers. Für die konkreten Betrachtungen reichen allerdings die angegebenen Attribute für die Basisklasse aus.

Davon abgeleitet wird nun ein Objekt definiert, das alle individuellen Anforderungen berücksichtigt, die sich aus den Abläufen eines Call Centers ergeben. Die hier definierten Daten werden in der Regel eine Kundennummer umfassen, die den Anrufer entsprechend identifiziert. Diese Kundennummer kann entweder implizit über seine mitgelieferte Telefonnummer ermittelt oder explizit durch Eingabe an einem Sprachcomputer bzw. durch einen Mitarbeiter erfasst werden. Darüber hinaus sind jedoch eine Vielzahl von Informationen denkbar, die eine Clientanwendung für den Aufbau eines Begrüßungsbildschirms (*screen pop*) sowie weitere Funktionen benötigt. Die folgende Aufzählung enthält typische Beispiele für derartige Informationen:

- Name und Vorname des Anrufers
  Diese Information stammt aus den Kundenstammdaten der lokalen Datenbank und muss durch die erste Instanz, die den Anruf entgegennimmt, anhand der Telefonnummer oder anderer Identifikationsmöglichkeiten zugeordnet werden.
- Kunden-ID
  Um Zugriff auf weitere Kundeninformationen zu haben, wird eine entsprechende ID mit übergeben. Derartige Informationen werden somit nicht direkt im Rahmen des Begrüßungsbildschirms sondern erst während des Gesprächs angezeigt, sofern es die jeweilige Situation verlangt.

- DB-ID der aktuellen Kontakthistorie
  Analog den Kundeninformationen können über diese ID alle Informationen bzgl. des Gesprächs in einer Kontakthistorie hinterlegt werden.

- Legitimationsstatus
  Falls die Prozesse des Call Centers eine Legitimation des Kunden erfordern, so ist es sinnvoll, diese Information ebenfalls direkt im Anrufobjekt zu hinterlegen. Somit sieht der Mitarbeiter sofort, welche Aktionen er für den Anrufer durchführen darf bzw. welche Informationen er ihm mitteilen kann. Die Legitimation erhält dieser durch Eingabe einer PIN entweder an einem Sprachcomputer oder aber direkt bei einem Mitarbeiter.

- Informationen zu Produkten des Kunden
  Lassen sich einem Kunden wenige oder sogar nur ein konkretes Produkt zuordnen, etwa eine Versicherung, ein Auto oder ein Konto, so kann die Anwendung auf diese Information reagieren und zum Beispiel nur mehr auf dieses Produkt zugeschnittene Informationen anzeigen.

- Informationen für eine Weiterleitung
  Im Falle einer Weiterleitung ergeben sich weitere Möglichkeiten, das Anrufobjekt zu nutzen, da sich hier der neue Mitarbeiter möglichst schnell ein Bild von der aktuellen Situation machen muss. Dies ist besonders wichtig im Falle eines *blind transfer*, der zwar für die Auslastung eines Call Centers sehr günstig ist, bei dem sich die Mitarbeiter allerdings nicht mehr vor der Weiterleitung abstimmen können. In diesem Falle müssen ausreichende Information mit dem Anruf übergeben werden, damit eine nahtlose Übergabe möglich ist. Typische Informationen sind hier der Name des letzten Mitarbeiters sowie ein Textfeld, in dem der weiterleitende Bearbeiter individuelle Hinweise geben kann.

Da diese Informationen sehr stark von den individuellen Anforderungen des Call Centers und damit der Funktionalität seiner CTI-Anwendungen abhängen, kann der Inhalt dieses abgeleiteten Objekts hier nur exemplarisch festgelegt werden. Es ist deshalb bei der weiteren Konzeption darauf zu achten, wie eine individuelle Anpassung dieses Objekts praktisch durchzuführen ist.

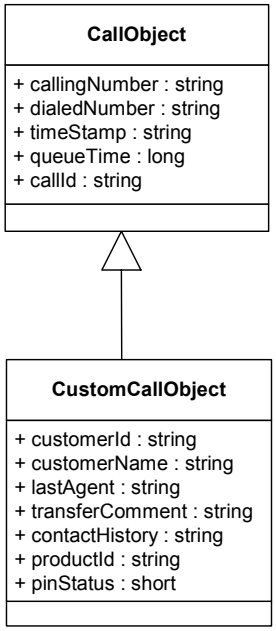


Abbildung 7-3 Objektmodell des Call Object

7.2.2 Datentransfer

Das derartig spezifizierte Anrufobjekt muss nun fest mit einem Telefonanruf verbunden und dementsprechend parallel zu diesem an diejenigen Arbeitsplätze bzw. deren CTI-Anwendung transportiert werden, denen der Telefonanruf zugestellt wird. Das bedeutet, dass die in modernen CTI-Systemen übliche Verknüpfung von Anruf und Daten mit mehr Funktionalität angereichert wird und dazu die mächtigeren Kommunikationsmechanismen moderner Middleware eingesetzt werden.

Dazu sind vorab nochmals die Anforderungen zusammenzustellen, die an den Datentransfer gestellt werden müssen. Als Erweiterung der bereits üblichen Mechanismen muss gewährleistet sein, dass Informationen zu einem Anruf sowohl strukturiert als auch typisiert definiert werden können und dann auch dementsprechend übertragen werden. Die Übertragung einer generischen Zeichenkette, die dann von unterschiedlichen Anwendungsprogrammen gemäß einer gemeinsamen Spezifikation ausgelesen und interpretiert wird, reicht nicht aus. Die Informationen, wie sie im vorangegangenen Kapitel für ein Anrufobjekt definiert wurden, müssen also vollständig übertragen werden und es muss der jeweiligen Anwendung ein komfortabler Zugriff auf diese Informationen ermöglicht werden.

Außerdem muss die Übertragung dieser Informationen den Vorgaben, die CTI-Systeme im Hinblick auf Zuverlässigkeit und Performance machen, nicht nachstehen. Die Informationen zu einem Anruf sollten im Idealfall gleichzeitig mit diesem an einem Arbeitsplatz verfügbar sein, so dass sich der Benutzer während der kurzen Zeit, ehe er das Gespräch entgegennimmt, ein Bild über den Inhalt dieses Anrufs machen kann.

Was die Sicherheit der Datenübertragung angeht, so kann man davon ausgehen, dass Client und Server über ein lokales und damit prinzipiell relativ sicheres Netz kommunizieren. Auch im Falle eines verteilten Netzes wird die Anbindung der Arbeitsplätze an die zentralen Datenbankserver in erster Linie über private Leitungen erfolgen. Dennoch kann auch hier die Anforderung nach Verschlüsselung der übertragenen Informationen auftreten, wenn es sich zum Beispiel um größere Unternehmen handelt oder wenn der Anwendungsbereich des Call Centers dies verlangt. Die Anwendung des Telefonbankings ist hier ein derartiges Beispiel. Hier sind Aktionen wie das Durchführen einer Überweisung mit so hohen Anforderungen an Zuverlässigkeit und Sicherheit geknüpft, dass sensible Daten wie etwa die PIN-Nummer, die zur Legitimation an einem Sprachcomputer eingegeben wird, unbedingt verschlüsselt übertragen und auch gespeichert werden muss.

Das bedeutet, es sind Vorkehrungen zu treffen, die ein ausreichendes Maß an Ausfallsicherheit sowie Zuverlässigkeit der Übertragung garantieren. Wie hoch genau jedoch hier die Ansprüche zu stellen sind, muss anhand der konzipierten Lösungen noch diskutiert werden.

7.2.2.1 Nutzung eines gemeinsamen Objektservers

Eine erste Variante ist die Verwaltung der Objektdaten durch einen zentralen Server, an den sich alle beteiligten Partner wenden, um Information des Anrufobjekts abzurufen bzw. zu verändern. Das Call Object ist also ein Objekt des verteilten Systems und der Zugriff auf die jeweiligen Informationen erfolgt dementsprechend mit den Kommunikationsmechanismen der Middleware. Im konkreten Fall wird das Objekt also als CORBA-Objekt spezifiziert und seine Schnittstelle entsprechend in IDL definiert. Die Informationen, die das Objekt bereitstellt, entsprechen dabei den in Abbildung 7-3 beschriebenen Strukturen. Ist ein Anruf im CTI-System aktiv, so existiert eine aktuelle Instanz dieser Objektklasse, auf die alle betroffenen Clients zugreifen können. Die an dem Anruf beteiligten Clientanwendungen können so individuelle Daten setzen oder lesen. Die Verwaltung übernimmt dabei ein entsprechender Objektserver. Bei einer Weiterleitung eines Anrufs wird über die Mechanismen der CTI-Middleware eine Objektreferenz auf das Call Object verschickt. Somit kann ein neuer Partner bzw. die von ihm genutzte Anwendung ebenfalls auf die konkrete Objektinstanz mit all ihren aktuellen Informationen zugreifen (vgl. Abbildung 7-4). Die Objektinstanz kann somit auch die Synchronisation der Zugriffe gewährleisten, wie sie ja für den Fall einer Weiterleitung sowie einer Konferenz gefordert werden.

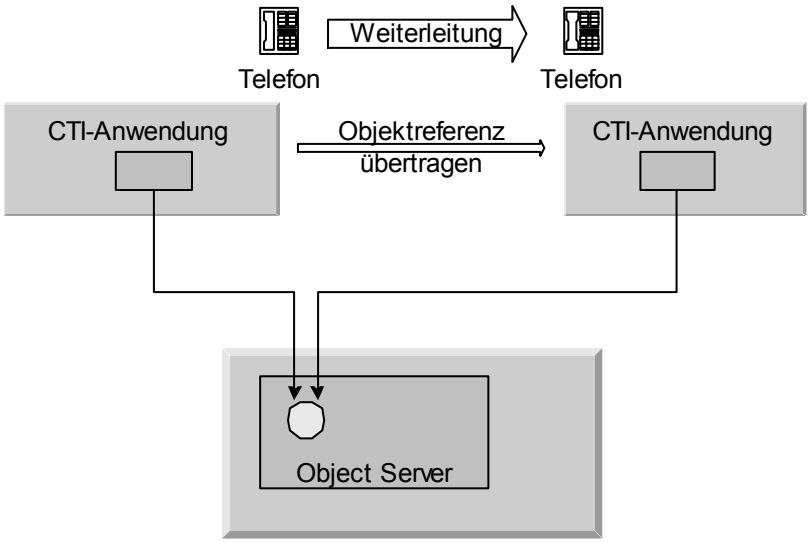


Abbildung 7-4 Datentransfer über gemeinsamen Objektserver

Nun gibt es eine Reihe von Möglichkeiten, wie dieses Objekt verwaltet werden kann. So ist es möglich, für jede Instanz des Objekts einen eigenen Server zu starten, der als Prozess während der Dauer des Anrufs besteht. CORBA unterstützt diesen Ansatz durch die Strategie *unshared server*, in der bei jedem ersten Methodenaufruf ein neuer Serverprozess gestartet wird. Vorteil dieser Lösung ist der hohe Replikationsgrad der Server und damit die hohe Zuverlässigkeit des Systems. Laufen nicht alle Prozesse auf einer Maschine, so kann mit dieser Strategie eine sehr hohe Ausfallsicherheit erreicht werden. Der damit verbundene Nachteil liegt in dem relativ hohen Bedarf an Ressourcen, den die Erzeugung immer neuer Prozesse mit sich bringt. Alternative dazu ist die Nutzung einer zentralen Factory, also eines Servers, der auf Anfrage ein neues Call Object erzeugt und während der Dauer des Anrufs entsprechend verwaltet. Da sich die Dauer eines solchen Anrufs nur im Bereich von wenigen Minuten bewegt und man in einem größeren Call Center mit teilweise sehr hohem Anrufaufkommen rechnen muss, ist dieser Effekt nicht zu vernachlässigen.

Es ist deshalb auch ein Kompromiss dieser beiden Ansätze als dritte Alternative zu betrachten, bei der eine feste Anzahl von Objektservern gestartet wird, die dann abwechselnd neu benötigte Anrufobjekte erzeugen und verwalten. Für die Auswahl des nächsten zuständigen Servers bietet sich eine einfache Strategie wie etwa das Verfahren *round robin* an, bei dem nach einer linearen Reihenfolge für jedes neue Objekt der nächste Server ausgewählt wird und am Ende der Liste mit dem ersten von neuem begonnen wird.[1] Somit vermeidet man das Problem, mit nur einem Objektserver ein *single point of failure* zu haben und kann dennoch ausreichend viele Objekte innerhalb eines Prozesses effizient verwalten.

Dieser Ansatz eines zentral verwalteten Objekts berücksichtigt dabei auch mögliche Erweiterungen bzgl. der Funktionalität des Anrufobjekts. Dieses dient ja in erster Linie der vorübergehenden Speicherung aller wesentlichen, zu einem Anruf gehörenden Daten. Es ist jedoch hier denkbar, diese um Funktionen zu erweitern. Als Beispiel wäre die Speicherung statistischer Daten eines Anrufs in einer Datenbank in diesem Zusammenhang eine sinnvolle Anwendung. Die konkrete Implementierung des CORBA-Objekts könnte hier jeden Zugriff auf die Anruferdaten und damit den Weg des Anrufs durch das Call Center genau verfolgen und eine für spätere Auswertungen sinnvolle Historie schreiben. Solche Dienste sind in der Regel Bestandteil von CTI-Basissystemen, allerdings mit dem Unterschied, dass hier anwendungsspezifische Daten integriert werden können, was den Nutzen hinsichtlich einer späteren Analyse der Geschäftsprozesse enorm erhöht.

[1] Die CORBA-Implementierung VisiBroker von Inprise unterstützt dieses Verfahren bereits.

7.2.2.2 Kopien der Objektdaten verschicken – value objects

Ein häufig auftretender Nachteil von verteilten objektorientierten Anwendungen ist, dass hier eine hohe Interaktion der Objekte untereinander einen erhöhten Kommunikationsaufwand zur Folge hat. Viele Designmuster der objektorientierten Entwicklung haben jedoch eine enge Zusammenarbeit mehrerer Objekte zur Folge, die sich dann eine komplexere Gesamtaufgabe teilen. Bildet man diese Muster auf die Objektstruktur einer verteilten Anwendung ab, so ist mit einem eventuell sehr hohen Kommunikationsaufwand zu rechnen.

Auf den konkreten Anwendungsfall bezogen bedeutet dies, dass jedes Lesen oder Schreiben eines Attributs des zentralen Serverobjektes eine entsprechende Kommunikation über das Netzwerk zur Folge hat. Was die Häufigkeit derartiger Zugriffe angeht, so lässt sich für die Basisklasse sagen, dass die hier angegebenen Attribute lediglich einmal gesetzt und dann von allen beteiligten Anwendungen nur mehr ausgelesen werden. Die Anzahl der dadurch auftretenden Zugriffe wird also nicht besonders hoch sein. Allerdings benötigt jeder Client, dem ein Anruf zugestellt wird, in der Regel die Mehrzahl, wenn nicht alle dieser Attribute, so dass dies immer eine ganze Reihe von *Get*-Aufrufen nach sich zieht. Nicht so leicht abzuschätzen ist dies für den anwendungsspezifischen Teil des Objekts, in dem ja jede beteiligte Clientanwendung spezifische Informationen hinterlegen kann. So kommen hier im Falle einer Weiterleitung entsprechende Aufrufe hinzu, um etwa den Namen des letzten Bearbeiters, einen individuellen Hinweis für den neuen Teilnehmer oder Verweise auf aktuell angefragte Produkte einzutragen. Jedoch wird sich auch hier die Gesamtanzahl der Zugriffe in einem überschaubaren Rahmen bewegen, da für umfangreichere Informationen lediglich Verweise gespeichert werden, über die dann in der zentralen Datenbank alle weiteren Daten abgerufen werden können.

Ein allgemeiner Lösungsansatz zur Reduktion dieses Kommunikationsaufwands ist der Austausch sog. *value objects* zwischen Client und Server, die aktuelle Kopien der Daten des Serverobjekts darstellen. Derartige Objekte definieren sich lediglich über ihre Daten und haben im Gegensatz zu Serverobjekten keinen aktuellen Zustand. Der Client kommuniziert also über die Schnittstelle eines Servers, über die er dieses Datenobjekt austauscht (vgl. Abbildung 7-5). Im konkreten Falle bietet der Server dazu zwei Dienste an:

- GetCallObject
  Liefert ein Datenobjekt mit der aktuellen Kopie der Daten, die zum aktuellen Anruf gehören. Als Parameter wird dazu ein den Anruf eindeutig bezeichnender Identifikator übergeben.
- SetCallObject
  Die eventuell veränderten Daten werden wieder an den Server übermittelt, der damit sein zentrales Anrufobjekt abgleicht.

Änderungen an den Attributen des Objekts werden also nicht synchron sondern asynchron auf Seiten des Clients durchgeführt und dann gesammelt zum Server übertragen. Da diese hier nicht mehr direkt auf ein Serverobjekt mittels einer Objektreferenz zugreifen sondern immer mit einem Server kommunizieren, muss zur Identifikation des Anrufobjekts ein eigener Parameter übergeben werden. Dies kann zum Beispiel die vom CTI-System eindeutig vergebene CallId sein. In diesem Falle müsste diese Information dann allerdings parallel dazu über den Datentransfer des CTI-Systems übertragen werden. Da es sich aber in beiden Fällen um Zeichenketten handelt, ist auch diese Variante ohne Probleme möglich.

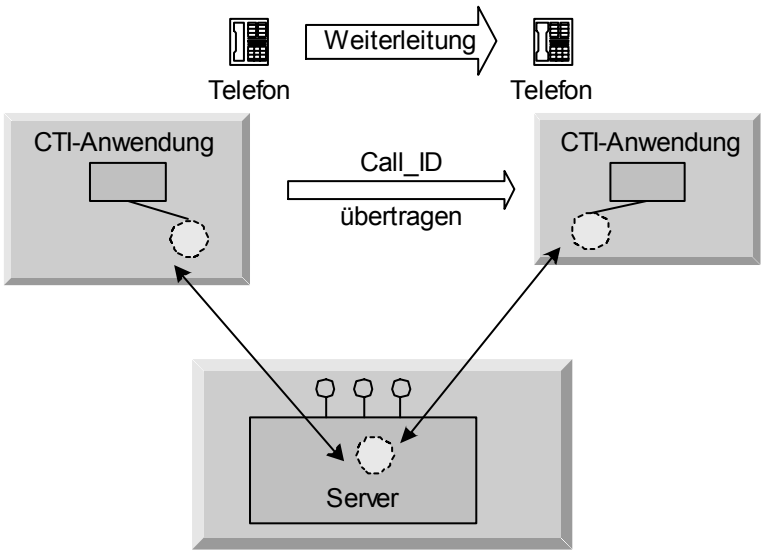


Abbildung 7-5 Kommunikation über value objects

Bei dieser Variante der Kommunikation zwischen Client und Server muss ein Anrufobjekt nicht unbedingt ein verteiltes Objekt sein, das mit einer IDL-Schnittstelle Zugriff auf seine Attribute erlaubt. Vielmehr ist hier auch die Implementierung eines bzw. mehrerer Server möglich, die als Manager alle Daten der aktuell vorhandenen Anrufe verwalten. Diese Server haben dann eine eigene IDL-Schnittstelle, die im Wesentlichen die zwei dargestellten Methoden zum Lesen und Schreiben der gesamten Daten eines Anrufobjekts anbieten. Mit dieser Variante existiert das Anrufobjekt dann im Grunde nur mehr virtuell und vor allem zustandslos in Form der jeweils aktuellen Informationen. Man wählt diesen Ansatz, wenn Objekte ausschließlich als Sammlung von Informationen identifiziert werden und keine bzw. nur sehr wenig Funktionalität sowie Zustände besitzen (vgl. dazu auch [SUN00]).

Der Einsatz von value objects lässt sich jedoch auch mit dem im vorherigen Kapitel dargestellten Ansatz kombinieren, in dem das CallObject als konkrete, verteilte Instanz während des Anrufs existiert. In diesem Falle enthalten diese verteilten Objekte die Methoden, um Kopien ihrer internen Attribute auszugeben bzw. neu zu setzen. Parameter dieser Dienste sind dabei eben beschriebene value objects.

Ein Nachteil der Verwendung solcher Datenobjekte liegt in der aufwändigeren Synchronisation der Zugriffe auf die Attribute des Anrufobjekts, die in diesem Falle nicht mehr synchron stattfinden. Der jeweilige Server muss also lokal das Objekt nach einem Aufruf von *getCallObject* sperren und nach *setCallObject* wieder freigeben.

Für eine Bewertung dieser Alternative kann man davon ausgehen, dass sich im Normalfall die Anzahl der Zugriffe auf das Anrufobjekt in Grenzen halten wird. Die Anwendung wird dabei für den Aufbau des Begrüßungsbildschirms auf einmal alle Informationen auslesen und vor einer Weiterleitung die entsprechend geänderten Werte setzen. Inwieweit sich also der Einsatz solcher Datenobjekte lohnt, hängt stark von dem Umfang der Informationen ab, die im individuellen Teil des Anrufobjekts jeweils definiert werden. Dass jedoch gerade für das Lesen und Anzeigen dieser Informationen der Zeitfaktor besonders kritisch ist, spricht wiederum dafür, die Informationen als Kopie in einem Schritt zu übertragen.

7.2.2.3 Nutzung der CTI-Middleware

Als letzte Variante besteht noch die Möglichkeit, den Datentransfer nicht als Funktionalität des Frameworks neu zu implementieren sondern dazu auf die Mechanismen des CTI-Basissystems zurückzugreifen. Die Anforderungen, die eingangs dieses Abschnitts an die Übertragung gestellt wurden, schließen allerdings einfache Mechanismen, die lediglich einen unstrukturierten Datenbereich mit einem Anruf verbin-

den, hier aus. Neuere CTI-Ansätze und insbesondere die JTAPI-Spezifikation sieht hier allerdings vor, ein beliebiges Java-Objekt mit einem Anruf zu verknüpfen. Damit wäre im Prinzip eine Verknüpfung auch beliebig strukturierter Daten und ein komfortabler Zugriff auf diese Informationen möglich. Allerdings hängt die Umsetzung dieser Funktionalität stark von der jeweiligen JTAPI-Implementierung ab. Somit kann dieses doch sehr wesentliche Kriterium des Frameworks nicht unabhängig konzipiert werden und lässt sich nur mit Hilfe einer bestimmten JTAPI-Software konkret realisieren. Da hier neben der reinen Übertragung auch noch eine Reihe von Anforderungen an Sicherheit und Performance gestellt wurden, scheint die Wahl einer geeigneten Implementierung nicht ohne Probleme zu sein. Durch die Realisierung mittels einer Middleware wie CORBA erzielt man hier eine größere Unabhängigkeit von einzelnen JTAPI-Implementierungen bzw. verallgemeinert sogar von unterschiedlichen CTI-Middleware-Produkten und kann bei Bedarf durch eine geeignete Implementierung diesen wichtigen Mechanismus in Richtung Sicherheit, Zuverlässigkeit und Leistungsfähigkeit geeignet optimieren.

7.2.2.4 Vergleich der Alternativen

Tabelle 7-1 zeigt die angesprochenen Alternativen nochmals im Überblick. Die erste Variante mit eigenständigen Objektservern erscheint vor allem vor dem Hintergrund eines hohen Anrufvolumens des hohen Ressourcenverbrauchs wegen für ungeeignet, die Verwendung einer Factory deshalb sinnvoller. Der Einsatz von value objects ist eine sinnvolle Alternative, allerdings ist die effizientere Kommunikation bei dem eher geringen Datenvolumen, das mit eine Anruf verknüpft wird, hier nicht so entscheidend. Die letzte Variante der Nutzung einer CTI-Middleware scheidet durch die damit verbundene Abhängigkeit zu bestimmten Lösungen aus, die im Grunde vermieden werden sollte.

Lösungsansatz	Merkmale	Vorteile	Nachteile
(A) Zentraler Objektserver pro Anrufobjekt	Jede Instanz eines Anrufobjekts wird durch einen eigenen Server verwaltet	Hohe Ausfallsicherheit	Hoher Verbrauch an Ressourcen
(B) Objectfactory	Mehrere Anrufobjekte werden durch einen Server verwaltet	Effiziente Verwaltung auch einer hohen Anzahl von Objekten	Geringere Ausfallsicherheit Bei nur einem Server „single point of failure“
(C) Einsatz von value objects	Anrufobjekt wird als reine Datensammlung zwischen Client und Server übertragen	Geringere und effizientere Kommunikation	Aufwändigere Synchronisation
(D) CTI-Middleware	Anrufobjekt wird über Mechanismen der CTI-Middleware mit einem Anruf verknüpft	Keine Neuimplementierung notwendig	Abhängigkeit von CTI-Middleware

Tabelle 7-1 Alternativen eines zentralen Anrufobjekts

Im Folgenden wird also der Ansatz einer zentralen Objectfactory weiter verfolgt, die mehrere Anrufobjekte verwaltet.

Abschließend sollte der ausgewählte Lösungsansatz nochmals den eingangs dargestellten Anforderungen gegenüber gestellt werden. Die erste Forderung, strukturierte Daten übertragen zu können, wird durch die Definition eines Anrufobjekts und dem Einsatz einer Middleware wie CORBA ausreichend befriedigt. Inwieweit die Definition derartiger Daten flexibel gestaltet werden kann, muss jedoch im anschließenden Abschnitt noch näher betrachtet werden. Auch der einfache Zugriff auf diese Informationen durch eine Anwendung wird durch den Einsatz der Middleware und den dort generierten Zugriffsmechanismen gewährleistet. Was die Zuverlässigkeit und Skalierbarkeit angeht, so wurde mit der ausgewählten Lösung ein Kompromiss gewählt, der den Anforderungen der meisten Call Center genügt. Eine Verschlüsselung der Daten wurde dabei in diesem Ansatz noch nicht berücksichtigt, da sie nur in besonderen Anwendungen

erforderlich ist. Hier bietet allerdings CORBA ebenfalls ausreichende Mechanismen, die im Bedarfsfall eingesetzt werden können.

7.2.3 Flexible Definition des Objekts

Die Konzeption des Anrufobjekts und die Betrachtung der hierfür in Frage kommenden Informationen hat gezeigt, dass die Aufteilung in einen systemspezifischen und anwendungsspezifischen Teil sinnvoll ist und eine wesentliche Ergänzung zu bestehenden CTI-Systemen darstellt. Wichtiger Aspekt dabei ist jedoch, dass diese spezifischen Informationen für die konkrete Anwendung individuell und ohne allzu großen Aufwand angepasst werden können. Es ist deshalb noch zu klären, wie eine konkrete Anpassung innerhalb des Frameworks aussehen kann.

In einem typischen Anwendungsfall, bei dem eine Call-Center-Anwendung etwa für eine Bank erstellt wird, möchte ein Entwickler dem Anrufobjekt etwa Informationen über das Wertpapierdepot eines Kunden hinzufügen, da diese wesentlich für den Aufbau des Begrüßungsbildschirms sind. Seine Anforderung besteht also darin, das vorhandene Framework zu konfigurieren bzw. zu erweitern, so dass seiner Anwendung bei Eingang eines Anrufes diese zusätzlichen Daten zur Verfügung stehen, ohne dass die übrige Funktionalität des Systems beeinflusst wird. Er würde demnach das eben dargestellte, abgeleitete Anrufobjekt um genau diese Informationen erweitern. Konkret bedeutet dies, die Schnittstelle des verteilten Objektes um genau diese neuen Attribute zu ergänzen.

Hinter einem CORBA-Objekt mit einer in IDL definierten Schnittstelle verbirgt sich jedoch immer eine konkrete Implementierung, auf die ein Objektserver zugreift, um Anfragen von Clients bedienen zu können. Verändert man demnach die Schnittstelle eines Objekts, so ist auch die damit verbundene Implementierung anzupassen und zumindest neu zu übersetzen. Eine individuelle Definition des angepassten Datenobjekts für einen Anruf benötigt demnach einen Eingriff durch den Entwickler in die bestehende Implementierung des Serverobjekts und damit in das vorhandene Framework. Dabei existieren nun einige grundlegende Alternativen, die in diesem Zusammenhang möglich sind. Deren Auswirkungen auf die Einsatzmöglichkeiten eines Frameworks in der Praxis werden nun etwas genauer betrachtet und diskutiert.

Die im Folgenden beschriebenen Alternativen zeigen einige Beispiele auf, wie individuell definierte Daten mit einem Anruf verknüpft werden können. Für jeden der angesprochenen Ansätze werden die Stellen erläutert, an denen der Entwickler einer konkreten Anwendung Anpassungen bzw. Erweiterungen vornehmen muss. Nicht dargestellt sind dabei Anpassungen der jeweiligen Anwendung, die ja in jedem Falle durchzuführen sind.

7.2.3.1 Verwendung generischer Strukturen

Eine erste Lösung des angesprochenen Problems bietet die Verwendung generischer und damit allgemein gültiger Datenstrukturen, die einmal fest definiert werden und dann flexibel genutzt werden können. Eine solche Datenstruktur sind sog. *name value pairs*. Die Schnittstelle des Anrufobjekts enthält eine fest definierte bzw. unbegrenzte Menge von Daten, denen noch kein Typ, allerdings ein Name zugeordnet wird. Der Client erhält die gesamte Menge dieser Daten mit einem Anruf und kann über bereitgestellte Set- und Get-Methoden, die als Parameter den Namen des gewünschten Werts benötigen, einzelne Daten abfragen bzw. neu setzen. Wie in Abbildung 7-6 dargestellt, muss hier der Entwickler nur diese Struktur anpassen, um neue, für seinen Anwendungsfall spezifische Datensätze hinzuzufügen. Er erweitert dazu die Liste um neue Einträge, die ein eindeutiges Schlüsselwort als Identifikator bekommen. Die Anwendung kann daraufhin mit diesem Schlüsselwort die jeweiligen Datensätze lesen.

Diese Lösung lässt sich mit CORBA z.B. durch Verwendung des Datentyps *any* sowie darauf aufbauender Sequenzen realisieren. Andere, sehr häufig angewandte Alternative ist die Übertragung aller Informationen als Zeichenketten des Typs *string*, die von der Anwendung dann in die entsprechenden Formate konvertiert werden. Kombinieren lässt sich dieser Ansatz im Prinzip mit beiden Varianten, den Zugriff auf das Anrufobjekt zu implementieren. Bei der Verwendung von Objektservern und Zugriff auf einzelne Attribute (Variante A) wird hier die Schnittstelle des Objekts um eine allgemeine Methode *getAttribute* erweitert,

die als Parameter ein Schlüsselwort erhält und aus der lokalen Liste dann den zugehörigen Wert zurück liefert. Der Client kann hier über die Objektreferenz diese Methode dann direkt nutzen. Im Fall der Variante C enthalten die value objects die gesamte Liste der name-value Paare und das Framework bietet auf der Seite des Clients die entsprechenden Zugriffsmethoden an.

Der Vorteil dieser Variante liegt in der flexiblen Definition neuer Daten durch den Nutzer des Systems, der dazu lediglich eine definierte Schnittstelle kennen und keinen Eingriff in die Implementierung des Frameworks vornehmen muss. Es müssen hier also insbesondere auch keine CORBA-Kenntnisse vorausgesetzt werden. Die CTI-Middleware von Nabnasset (jetzt Quintus), die ebenfalls CORBA einsetzt, bietet einen derartigen Ansatz. Nachteil einer solchen Implementierung ist jedoch, dass die Daten nur sehr eingeschränkt strukturiert werden können. Komplexer aufgebaute Daten wie etwa verschachtelte Strukturen oder Listen können nicht definiert werden. Außerdem bietet dieser Ansatz keine Überprüfung der einzelnen Typen.

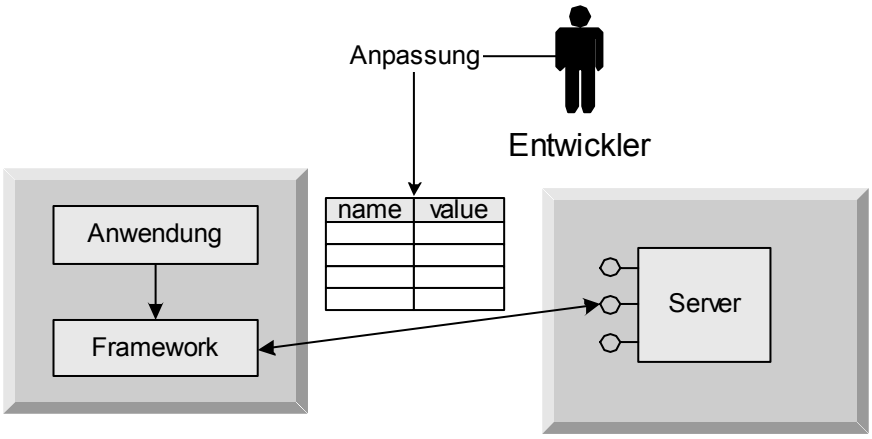


Abbildung 7-6 Anpassung durch name-value Datensätze

7.2.3.2 XML als Austauschformat

Eine andere Variante, frei definierbare Daten auszutauschen, ist die Verwendung von XML als Austauschformat. Dieses wird derzeit in sehr vielen Anwendungsfällen als allgemein verwendbares Format unterschiedlichster Anwendungen insbesondere vor dem Hintergrund von eBusiness stark diskutiert.

Bezogen auf den Austausch der Informationen eines Anrufobjekts würden hier die Daten in einer XML-Datei mit entsprechenden Identifikatoren (*tags*) versehen und gesammelt zum Client übertragen. Im Prinzip ähnelt dieser Ansatz dem eben dargestellten mit der Verwendung von name-value Paaren. Allerdings bietet XML mehr Möglichkeiten, kontrolliert auf strukturierte Daten zugreifen zu können. So ist es möglich, neben flachen auch komplexere und verschachtelte Datenstrukturen zu spezifizieren. Außerdem erlaubt XML, über die Definition eines Dokumenttyps (*document type definition DTD)* eine gültige Struktur der Daten vorzugeben, so dass die beteiligten Kommunikationspartner XML-Dateien auf das korrekte Format ihres Inhalts überprüfen können. Allerdings werden auch hier die Informationen keiner Typüberprüfung unterzogen.

Diese Variante ergibt also vom Ansatz her keine wesentlichen Änderungen, bietet aber gegenüber der eben dargestellten Lösung mehr Funktionalität.

7.2.3.3 Individuelle Anpassung der Serverobjekte

Verlässt man den Ansatz allgemein gültiger Datenstrukturen, so muss man Anpassungen des Frameworks an bestimmten Stellen in Kauf nehmen. Erweitert man die Attribute der IDL-Schnittstelle des Anrufobjekts, so muss die Implementierung dieser Schnittstelle angepasst werden, worauf dem Client dann dafür entsprechende Set- und Get-Methoden zur Verfügung stehen. Sofern sich hinter diesen Methoden keine weitere Funktionalität wie etwa Konsistenzprüfungen verbirgt, hält sich der Aufwand für diese Anpassungen in Grenzen. Man erhält dadurch eine entsprechend flexible Lösung, da auch komplizierte Strukturen oder Objektreferenzen berücksichtigt werden können. Außerdem kann das Objekt in diesem Fall in seiner Funktionalität ebenfalls erweitert werden, beispielsweise um genau eben angesprochene Methoden individuell zu implementieren, etwa um die Formate einzelner Daten zu überprüfen oder zu konvertieren. Allerdings ist dazu die Implementierung bzw. Anpassung der entsprechenden Objektserver notwendig, die für den Nutzer des Anrufobjekts Grundkenntnisse der CORBA-Technologie voraussetzt.

An welchen Stellen nun Anpassungen vorzunehmen sind, hängt bei diesem Ansatz von der ausgewählten Variante der Anrufobjekte ab. Die Verwendung von Datenobjekten in Variante C bedeutet, dass der Entwickler neben dem Server auch die Stelle des Frameworks anpassen muss, die auf Seiten des Clients die Daten entgegennimmt und aufbereitet (vgl. Abbildung 7-7).

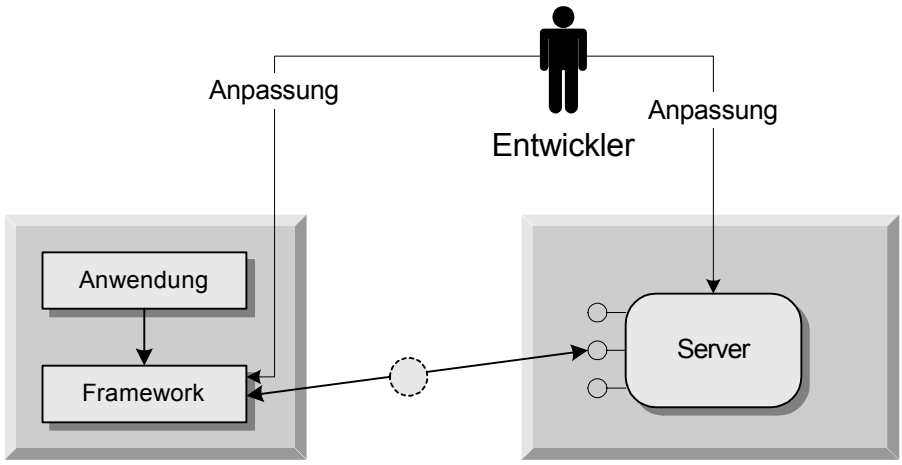


Abbildung 7-7 Anpassungen bei Factory mit value-objects

In Kombination mit Variante A bzw. B nutzt die Anwendung direkt über die Objektreferenz die entsprechend erweiterte Schnittstelle des Serverobjekts. Die Komponenten des Frameworks bleiben hier auf dieser Seite unverändert, da ja weiterhin eine CORBA-Objektreferenz vom gleichen Objekttyp an die Anwendung übergeben wird (vgl. Abbildung 7-8).

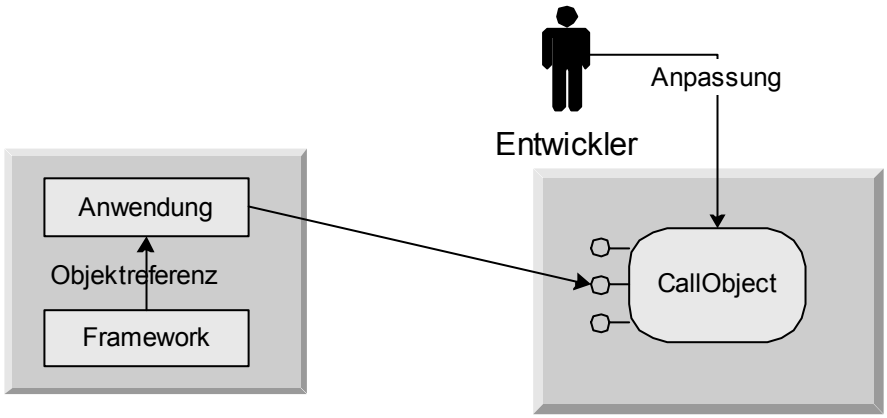


Abbildung 7-8 Anpassungen bei zentralem Anrufobjekt

7.2.3.4 Weitere Alternativen

Eine sehr aufwändige Lösung wäre die Konzeption eines eigenen Compilers, der für alle individuell definierten Attribute die Get-Set-Methoden auf Seiten des Servers automatisch realisiert. In diesem Fall könnten individuelle Datenstrukturen für ein Anrufobjekt definiert werden, ohne eine entsprechende Serverimplementierung anpassen zu müssen. Es wäre jedoch weiterhin eine Neuübersetzung des Servers notwendig. Die Komplexität eines derartigen Ansatzes erscheint allerdings gerade bei komplizierteren Datenstrukturen zu hoch.

7.2.3.5 Bewertung

Um abschließend eine Bewertung dieser unterschiedlichen Ansätze vornehmen zu können, muss noch einmal das allgemeine Ziel betrachtet werden, das mit dem Framework verfolgt werden soll. Es geht nämlich an dieser Stelle konkret um die Frage, ob eher ein flexibles Basissystem für Call-Center-Anwendungen entworfen werden soll, das im konkreten Einsatz jedoch individuellen Anpassungen unterworfen wird, oder ob es zu einer vollständigen CTI-Middleware mit entsprechend erweiterter Funktionalität ausgebaut werden soll.

Führt man sich nochmals die in der Einleitung definierten Anforderungen sowie in 6.1 zusammengefassten Charakteristika eines Frameworks vor Augen, so sind hier individuelle Anpassungen des Basissystems durch einen Entwickler durchaus vorgesehen. Der große Vorteil eines solchen Ansatzes liegt ja darin, dass das System zwar die wesentlichen Designvorgaben enthält, aber dadurch die Flexibilität bei Erweiterungen und Anpassungen nicht wesentlich beeinträchtigt wird. Man muss hier allerdings festhalten, dass der Entwickler einer Client-Anwendung im Call Center für die Verwendung und Anpassung eines solchen Frameworks neben Kenntnissen aus dem Bereich CTI und Telekommunikation auch Know-how im Umfeld verteilter Systeme haben muss, um insbesondere eine Middleware wie CORBA nutzen zu können. Dies ist jedoch in einem gewissen Sinne eine konsequente Weiterführung des CTI-Ansatzes, der auf einer höheren Ebene zwangsläufig zum Aufbau verteilter Anwendungen und Systeme führt. Somit bewegen sich moderne Anwendungen in Call Centern in der Regel immer im Bereich unternehmensweiter, verteilter Systeme und müssen sich daher ohnehin mit der Thematik etwa einer Middleware beschäftigen. Deshalb können die dadurch in den Vordergrund tretenden Technologien und Konzepte durchaus auch für die Anwendung des Frameworks vorausgesetzt werden.

Eine Weiterführung dieses Gedankens ist es, CORBA als Architektur nicht nur für das Framework sondern allgemein für das bereits bestehende System des Call Centers und damit auch den Client vorauszusetzen. Die Vorteile dieser Technologie zeigen sich ja besonders, wenn sie nicht nur lokal sondern in vielen

Bereichen eines Unternehmens eingesetzt werden. Dann existieren für wichtige Elemente in der Regel schon sog. *business objects* wie etwa Kunde, Adresse usw. Ein abgeleitetes Call-Objekt könnte in diesem Falle seine notwendigen Informationen selbst über derartige Objekte beziehen, wodurch ein äußerst flexibles System aufgebaut würde.

Zusammenfassend lässt sich also sagen, dass eine Erweiterung des Serverobjektes inklusive der hier notwendigen Anpassungen die flexibelste Alternative darstellt. Da die Anforderungen eines Call Centers besonders im Hinblick auf die Integration bestehender Systeme in der Regel äußerst unterschiedlich und individuell ausfallen, überwiegen hier die Vorteile. Dass dabei für die Nutzung gewisse Kenntnisse über CORBA vorausgesetzt werden müssen, ist akzeptabel. Die dargestellten Alternativen haben jedoch auch gezeigt, dass das System ebenfalls etwa durch die Verwendung von Formaten wie XML zu einer eigenständigen Softwarelösung umgebaut werden kann.

7.2.4 Übergabe der Objektreferenz

Der Austausch von Anrufdaten über verteilte Objekte erfordert, dass eine Objektreferenz zwischen den Systemen beider Teilnehmer übergeben wird. Da eine Objektreferenz z.B. in CORBA als String übergeben werden kann, existieren hier mehrere Alternativen, wie diese Information ausgetauscht werden kann.

Eine erste Möglichkeit besteht darin, die Methoden der CTI-Middleware, wie etwa *getApplicationData* sowie *setApplicationData* von JTAPI, zu verwenden. Sie erlauben es, ein beliebiges Java-Objekt, also auch ein String-Objekt, mit einem Anruf zu verbinden, das dann automatisch zum Beispiel bei einer Weiterleitung dem neuen Teilnehmer zur Verfügung gestellt wird. Voraussetzung dafür ist jedoch, dass die verwendete CTI-Implementierung diese Methoden unterstützt. Dies kann jedoch derzeit noch nicht bei allen verfügbaren Produkten sichergestellt werden. Denn gerade wenn diese auf bereits bestehende Systeme wie etwa TSAPI aufsetzen, bieten sie in der Regel diese Dienste, die ja einen Austausch beliebiger Java-Objekte voraussetzt, nicht immer an. Die Abhängigkeit des Frameworks von einzelnen CTI-Implementierungen ist deshalb ein Nachteil dieser Lösung.

Alternativ dazu besteht die Möglichkeit, eine eigene Kommunikation zwischen Komponenten der jeweiligen Framework-Instanzen aufzubauen, um die Referenzen auszutauschen. Dabei wäre eine Socket-Verbindung ebenso möglich wie die Kommunikation über CORBA. In diesem Falle müsste das Framework ein Serverobjekt bereitstellen, das neue Objektreferenzen entgegennimmt. Vorteil dabei ist, dass durch die direkte Kommunikation ein schneller Datenaustausch gewährleistet werden kann, der gerade im Falle des Anrufobjekts besonders wichtig ist. In beiden Fällen muss jedoch die Adressierung des jeweiligen Kommunikationspartners gelöst werden. Jede Client-Instanz des Frameworks müsste dabei Zugriff auf ein aktuelles Verzeichnis haben, so dass aus einer Telefonnummer die richtige IP-Adresse oder der Name des Objektservers ermittelt werden kann. Um dies auf Seiten des Clients zu vermeiden, kann diese Funktionalität jedoch auch in die zentrale Object-Factory verlegt werden. Da hier das Anrufobjekt selbst schon verwaltet wird, reicht es aus, lediglich alle Teilnehmer eines Anrufs zu registrieren und ihnen auf Anfrage eine Referenz auf das jeweilige Objekt zu übergeben. Damit wird die Client-Seite des Frameworks von dieser Aufgabe entlastet und die gesamte Implementierung weniger komplex. Allerdings erhöht sich dadurch die Last der Object-Factory, die nun jeden Teilnehmer registrieren und das entsprechende Anrufobjekt zuordnen muss. Dies könnte einen Engpass bei sehr großen Systemen darstellen.

In Tabelle 7-2 sind die drei Varianten nochmals gegenübergestellt. Zusammengefasst erscheint die Verwendung der zentralen Factory als sinnvollste Variante. Neben der relativ einfachen Implementierung bleibt das System damit bei dieser wichtigen Funktion unabhängig von der gewählten CTI-Middleware. Eventuell auftretende Engpässe bzgl. Performance müssen bei einer konkreten Realisierung besonders beachtet werden.

Lösungsansatz	Merkmale	Vorteile	Nachteile
Methoden der CTI-Middleware verwenden	Objektreferenz wird als String-Objekt direkt an Anruf gebunden	Einfache Implementierung	Abhängigkeit von CTI-Implementierung
Direkte Kommunikation	Individuelle Kommunikation über Sockets oder CORBA	Direkter und damit schneller Datenaustausch	Aufwändige Implementierung Zentrale Adressierung notwendig
Verwendung der Object-Factory	Registrierung aller Teilnehmer eines Anrufs in Factory	Unabhängigkeit von CTI-Middleware Einfache Implementierung	Höhere Last für Object-Factory

Tabelle 7-2 Alternativen für Austausch der Objektreferenz

7.2.5 Verwendung der Object-Factory

Aus den dargestellten Konzepten, was die Verwendung von Anrufobjekten im Framework angeht, ergibt sich nun zusammenfassend folgendes Bild für die zentrale Object-Factory. Sie verwaltet, wie die Abbildung 7-9 darstellt, alle Anrufobjekte des Systems und ordnet diese in einer internen Tabelle den verschiedenen Anrufteilnehmern zu. Dabei ist eine eindeutige Identifizierung der Teilnehmer notwendig, idealerweise über die in der Regel eindeutige Telefonnummer aber auch die IP-Adresse des jeweiligen Systems oder einen Identifikator des Anrufs. Mit der Factory kommuniziert die Komponente Call Control, die für den Client von der Factory die entsprechende Objektreferenz erzeugt bzw. abruft. Daraufhin kann die Anwendung selbst direkt mit dem Objekt kommunizieren, also Informationen abrufen oder neu setzen.

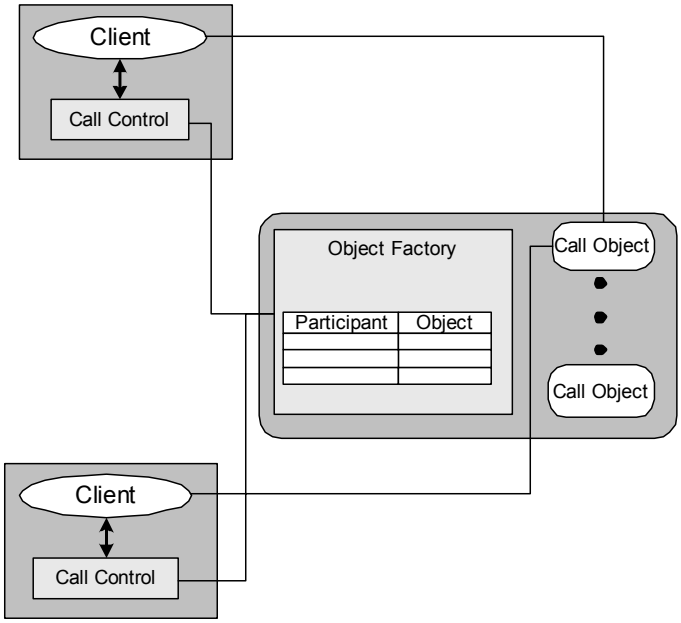


Abbildung 7-9 Verwendung der Object-Factory

Die Factory kümmert sich darüber hinaus auch um das Löschen der Objekte. Sobald alle Clients ihre Referenz auf ein Objekt freigegeben haben, kann dieses entfernt werden. Zusätzlich dazu ist ein Timeout-Mechanismus vorzusehen, damit im Rahmen einer Garbage Collection nicht mehr benötigte Objekte auch ohne Aktion der Clients wieder entfernt werden.

7.2.6 Synchronisation des Zugriffs

Da der Zugriff auf Attribute des Anrufobjekts durch direkten Zugriff des Clients auf dieses Objekt realisiert wird, bietet es sich an, die Synchronisation auch in diesem zu realisieren. Die Schnittstelle der Basisklasse enthält demnach zwei Methoden *occupyCallObject* sowie *releaseCallObject,* über die ein exklusiver Zugriff auf alle Attribute angefordert werden kann. Parameter dieser Methoden ist die jeweilige Telefonnummer der Anwendung, die die Sperre auf die Ressourcen anfordert bzw. wieder freigibt. Versucht eine Anwendung, ein bereits gesperrtes Objekt für sich zu reservieren, so wird eine *ObjectOccupiedException* geworfen. Inwieweit nun der unerlaubte Zugriff auf Attribute unterbunden wird, bleibt dabei prinzipiell der jeweiligen Implementierung überlassen. Im Grunde reicht jedoch für die Prozesse eines Call Centers eine sehr einfache Umsetzung aus, bei der lediglich die Exception erzeugt wird und daraufhin davon ausgegangen werden kann, dass sich Clients nur gutmütig verhalten und sich selbst an diese Synchronisation halten. Da man davon ausgehen kann, dass alle Clients des Call Centers die gleiche Software verwenden, ist eine solche kooperative Synchronisation in der Regel leicht einzuhalten. Sollte eine Implementierung dennoch eine restriktivere Kontrolle für nötig halten, so ist darauf zu achten, dass die dafür notwendige Funktionalität im Basisobjekt derartig implementiert wird, dass sie ohne größeren Aufwand im abgeleiteten Objekt für individuell definierte Attribute zu verwenden ist. Ansonsten ist der Ansatz gefährdet, dass die Anpassung dieses Objekts leicht durchzuführen sein muss.

In jedem Falle ist bei der Implementierung jedoch darauf zu achten, dass die beiden Synchronisationsmethoden nicht nebenläufig von mehreren Clients ausgeführt werden können.

7.2.7 Schnittstellen

Die aus diesem Konzept resultierende Schnittstelle für den Zugriff auf bzw. die Verwaltung des Anrufobjekts ist abschließend in Abbildung 7-10 nochmals dargestellt.

```
interface CallObject
    {
        attribute string callingNumber;
        attribute string dialedNumber;
        attribute string timeStamp;
        attribute long queueTime;
        attribute string switchId;
        attribute string callId;

        void occupyCallObject(in string extension) raises(ObjectOccupiedException);
        void releaseCallObject(in string extension);

    };
interface CustomCallObject:CallObject
    {
        attribute string customerId;
        attribute string customerName;
        attribute string lastAgent;
        attribute string transferComment;
        attribute string contactHistory;
        attribute string productId;
        attribute short  pinStatus;
    };
interface CallObjectFactory
{

    CustomCallObject createCallObject();
//   CustomCallObject createCustomCallObject();
    void addMember(in CallObject callObject, in string destination);
    CallObject getCallObject(in string extension);
    CustomCallObject getCustomCallObject(in string extension);
    void deleteCallObject(in CallObject co);

};
```

Abbildung 7-10 Schnittstellen für die Verwaltung des Anrufobjekts

7.3 Telefonsteuerung

Neben der Verknüpfung individueller Daten mit einem Anruf ist die Steuerung des Telefonsystems die zweite wichtige Funktionalität des Systems. Die Aufgabe dieser Komponente besteht also darin, die in der Spezifikation festgelegte Funktionalität einem Client zur Verfügung zu stellen. Sie stellt demnach auch die wesentliche Schnittstelle des Frameworks nach außen dar. Wichtiges Kriterium dabei ist es, Details des zugrunde liegenden Telefonsystems soweit wie möglich zu verbergen. Dazu gehört auch, dass komplexere Abläufe entsprechend komfortabel unterstützt werden.

Als zentrale Komponente kommuniziert sie, wie in Abbildung 7-11 dargestellt, sowohl mit dem Objektserver als auch mit dem State Manager, der den jeweils lokalen Zustandsautomaten verwaltet. Wie nachfolgend noch genauer erläutert wird, greift die Implementierung dabei auf einen JTAPI-Provider zurück, der die benötigte Telefonfunktionalität bereitstellen muss. Darüber hinaus ist innerhalb dieser Komponente ein Observerobjekt zu implementieren, das, nachdem es bei dem Provider angemeldet wurde, alle JTAPI-Ereignisse übermittelt bekommt und daraus die für den State Manager relevanten Eingaben erzeugen muss.

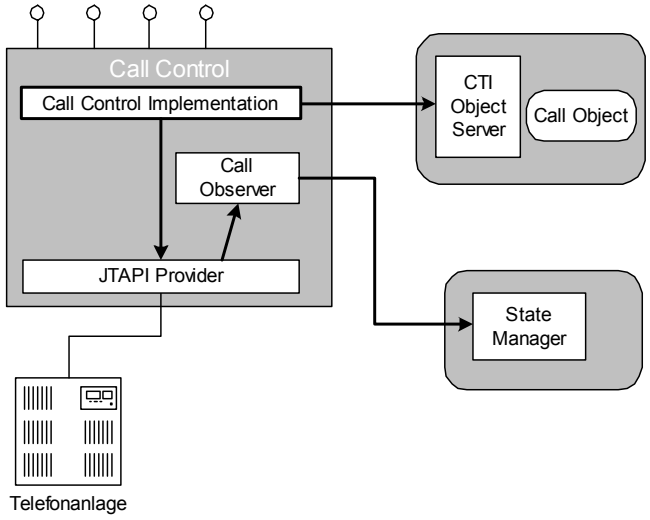


Abbildung 7-11 Call Control Komponente

7.3.1 Schnittstelle

Die Schnittstelle leitet sich ziemlich direkt aus den in der Spezifikation angegebenen Use Cases ab. Diese legen eine Aufteilung in Basisfunktionen sowie erweiterte Dienste nahe (vgl. Abbildung 7-12). Als CORBA-Schnittstelle lässt sich dies vorzugsweise durch den Vererbungsmechanismus umsetzen. Die Basisversion der Schnittstelle enthält demnach alle Grundfunktionen wie *answer* oder *dial*, die erweiterte Version komplexere Dienste zur Weiterleitung, Konferenz, administrative Funktionen sowie die Autorisierung eines Anrufers. Der Übersichtlichkeit wegen an dieser Stelle noch nicht dargestellt, für eine Schnittstelle aber natürlich notwendig, ist die Definition der möglichen Fehlerfälle und daraus resultierenden Ausnahmen, die das System an den Client meldet. Dies wird etwas später in einem eigenen Kapitel noch behandelt.

```
Interface BasicTelCtrl
    {
        Object getCallObject();
        void answer();
        void dial(in string number);
        void hold();
        void reConnect();
        void hangup();
    };
```

```
interface TelCtrl:BasicTelCtrl
    {
        void initTransfer(in string dest);
        void completeTransfer();
        void cancelTransfer();

        void initConference();
        void completeConference();
        void cancelConference();

        void returnCall();
        void authorizeCaller();

        void login();
        void logout();
        void showStatus();
        void showQueue();
    };
```

Abbildung 7-12 Schnittstelle CallControl

Wie man der Schnittstelle entnehmen kann, beziehen sich alle Methoden nicht nur auf einen aktuellen Zustand des Telefonsystems sondern beinhalten auch keine Details bzgl. mehrerer Telefonleitungen. Moderne Telefonanlagen bieten jedoch dem Benutzer über geeignete Endgeräte häufig den Zugriff auf mehrere Verbindungen an, die getrennt voneinander aufgebaut werden und dann beliebig für Weiterleitungen oder Konferenzen verwendet werden können (vgl. 2.5.2). Auch JTAPI unterstützt die Verwendung mehrerer Verbindungen, auf die sich einzelne Ereignisse und Operationen immer beziehen. Eine Erweiterung der Schnittstelle in diese Richtung ist demnach dadurch zu realisieren, dass allen Operationen ein weiterer Parameter *Line* hinzugefügt wird. Methoden wie *Dial* sowie Ereignisse wie etwa *Alert* geben dann als Rückgabewert bzw. weitere Information ebenfalls immer an, auf welche Verbindung sie sich gerade beziehen. Allerdings wird die Bedienung solch einer Schnittstelle durch den Client dadurch wesentlich komplizierter, da er sich nun auch um die Verwaltung mehrerer Verbindungen mit eigenen Zustandsautomaten kümmern muss. Da jedoch eines der Ziele dieses Frameworks eine Schnittstelle zum Telefonsystem ist, die dem Anwender die Kontrolle komplizierter Abläufe abnimmt, wird hier bewusst auf die Unterstützung mehrerer Verbindungen verzichtet. Gerade in Call Centern, in denen es in erster Linie auf Effizienz der Mitarbeiter ankommt, ist eine kompliziertere Bedienung, auch wenn sie mehr Flexibilität bietet, eher fehl am Platz.

Weiterhin ist bzgl. der Schnittstelle zu erwähnen, dass hier eine klare Trennung zwischen Weiterleitung und Konferenz getroffen wird. Aus Sicht einer Telefonanlage ist dies eigentlich nicht notwendig. Hier kann in der Regel erst später, sobald 2 oder mehrere Verbindungen zur Verfügung stehen, entschieden werden, ob ein Gespräch weitergeleitet oder aber eine Konferenz aufgebaut wird. Dies ist eine funktionale Einschränkung, ermöglicht dem Framework jedoch, klar zwischen beiden Prozessen zum Beispiel im Hinblick auf eine notwendige Synchronisation des Anrufobjekts zu unterscheiden

7.3.2 Anbindung an JTAPI

Um die eben dargestellte Schnittstelle implementieren zu können, muss die Komponente über einen JTAPI-Provider Zugriff auf das Telefonsystem erhalten. Dazu muss ein JTAPI-Provider mit einer geeigneten Implementierung initialisiert werden. Über diesen Provider erhält die Komponente dann Zugriff auf notwendige Objekte wie etwa *Connection* oder *CallCenterCall*. Diese Objekte werden zentral verwaltet, so dass alle Funktionen der Schnittstelle jederzeit darauf zugreifen können. Darüber hinaus muss die Komponente *Call Control* alle wichtigen Ereignisse des JTAPI-Systems verarbeiten. Dazu müssen Implementierungen für Observerobjekte bereitgestellt werden, die dann entsprechend registriert werden können. Für Ereignisse des Providers ist dies ein Objekt vom Typ *ProviderObserver*, für alle anrufbezogenen Ereignisse ein Objekt vom Typ *CallObserver*. Wichtige Aufgabe dabei ist es, für das Framework relevante Ereignisse herauszufiltern und dann als Eingabe für den Zustandsautomaten an die Komponente *StateManager* weiterzuleiten. Daneben können auch diese Provider Objekte wie *Call* oder *Connection* initialisieren, etwa wenn ein neuer Anruf an das System durchgestellt wird.

7.3.3 Mögliche Erweiterungen

Abschließend sei an dieser Stelle noch auf mögliche Erweiterungen hingewiesen, die vom jeweiligen Telefonsystem bzw. den Abläufen im Call Center abhängen und somit nicht allgemeingültig zu spezifizieren sind. Hierzu gehören beispielsweise die Steuerung sowie die Abfrage eines individuellen Anrufbeantworters, der jedem Telefonarbeitsplatz zugeordnet ist. Weitere Funktionen wären das Umleiten aller Gespräche auf eine individuelle Nummer oder einen vordefinierten Vertreter, individuelle Ansagen des Anrufbeantworters oder Einstellungen des lokalen Telefons wie etwa die Lautstärke des Hörers. Derartige Anpassungen sind jedoch durch Ableitung der dargestellten Schnittstelle möglich.

Ebenfalls nicht zu vernachlässigen ist eine ausreichende Fehlerbehandlung. Dem Client müssen alle Fehlersituationen gemeldet werden, wobei verschiedene Stufen der Fehlerbeschreibung unterstützt werden sollten, aus denen mehr oder weniger detaillierte Informationen hervorgehen.

7.4 Zustandsautomat

Die *StateMachine* verwaltet als globaler Zustandsautomat alle für die Nutzung von CTI in Frage kommenden Zustände. Diese umfassen elementare, durch die Telefonie vorgegebene Zustände wie *Online, Offline, Dialing* etc. Wie aus der Modellierung der Abläufe mit Hilfe von Petrinetzen hervorgegangen ist, benötigt das Framework jedoch einen erweiterten Zustandsautomaten (vgl. 6.3), der darüber hinaus auch Zwischenzustände etwa bei einer Weiterleitung unterstützt. Wie in Abbildung 7-13 dargestellt, kommuniziert dazu eine eigene Komponente *State Manager* direkt mit der Komponente zur Anrufsteuerung. Dadurch ist gewährleistet, dass auch der Zustandsautomat unabhängig von der jeweiligen CTI-Implementierung bleibt. Als eigenständige Komponente wird der Zustandsautomat direkt von einer oder mehreren Client-Anwendungen genutzt, die über eine definierte Schnittstelle den aktuellen Status des Systems abrufen können. Darüber hinaus besteht, wie nachfolgend noch erläutert wird, die Möglichkeit, direkt mittels einen Ereignisdienstes über neue Zustände informiert zu werden.

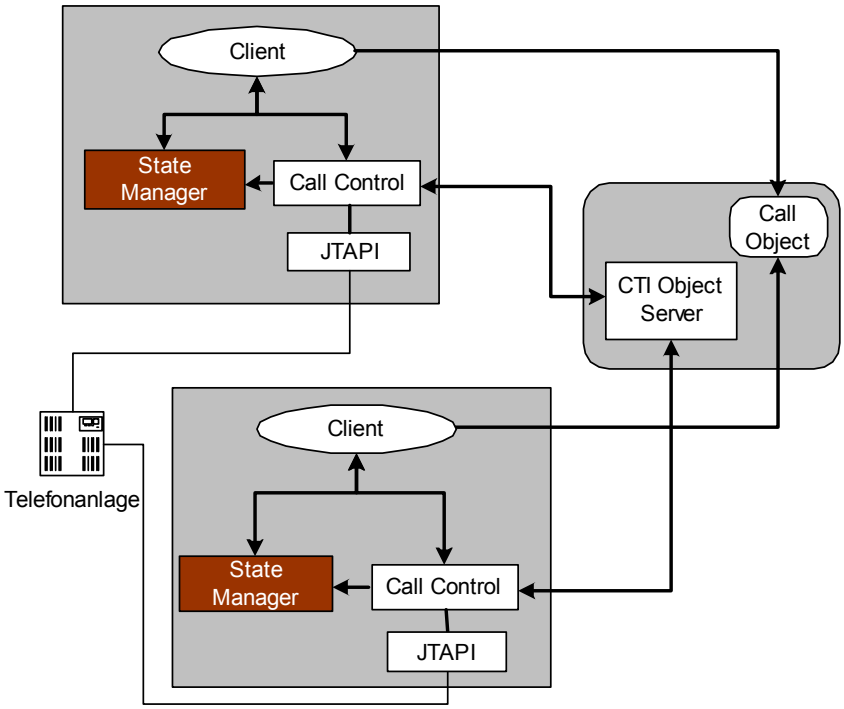


Abbildung 7-13 State Manager Komponente

7.4.1 Schnittstelle

Die Schnittstelle des *StateMgr* ist relativ einfach, da sie lediglich eine Methode zum Abruf des aktuellen Zustandes enthält sowie einen Dienst, um dem Automaten eine neue Eingabe zu übergeben (vgl. Abbildung 7-14).

```
Interface StateMgr
    {
        short getStatus();
        void processInput(in short input);
    };
```

Abbildung 7-14 Schnittstelle des StateMgr

7.4.2 Zustandsübergänge

Die Menge der Zustände, die die Implementierung des Automaten unterstützen muss, ergibt sich aus der Spezifikation und ist nachfolgend noch einmal zusammengefasst:

Q={OFFLINE, RINGING,CALLING, ONLINE,
ONHOLD,CALLING_TRANSFER,
CALLING_CONFERENCE,TRANSFER_PREPARED,
CONFERENCE_PREPARED}

Der Anfangszustand ist dabei OFFLINE.

Die Übergangsrelation ist in Tabelle 7-3 dargestellt.

Zustand	Eingabe	Folgezustand
OFFLINE	INPUT_ALERT	RINGING
OFFLINE	INPUT_CALLING	CALLING
OFFLINE	INPUT_ANSWERED	OFFLINE
OFFLINE	INPUT_WAIT	OFFLINE_WAITING
OFFLINE_WAITING	INPUT_ALERT	RINGING
OFFLINE_WAITING	INPUT_STOP_WAITING	OFFLINE
RINGING	INPUT_ACCEPT	ONLINE
RINGING	INPUT_HANGUP	OFFLINE
CALLING	INPUT_ANSWERED	ONLINE
ONLINE	INPUT_HOLD	ONHOLD
ONHOLD	INPUT_RECONNECT	ONLINE
ONLINE	INPUT_HANGUP	OFFLINE
ONLINE	INPUT_INIT_TRANSFER	CALLING_TRANSFER
ONLINE	INPUT_INIT_CONFERENCE	CALLING_CONFERENCE
CALLING_TRANSFER	INPUT_ANSWERED	TRANSFER_PREPARED
CALLING_TRANSFER	INPUT_CANCEL_TRANSFER	ONLINE
CALLING_CONFERENCE	INPUT_ANSWERED	CONFERENCE_PREPARED
CALLING_CONFERENCE	INPUT_CANCEL_CONFERENCE	ONLINE
TRANSFER_PREPARED	INPUT_TRANSFERRED	OFFLINE
TRANSFER_PREPARED	INPUT_CANCEL_TRANSFER	ONLINE
CONFERENCE_PREPARED	INPUT_CONFERENCED	ONLINE
CONFERENCE_PREPARED	INPUT_CANCEL_TRANSFER	ONLINE
ONLINE	INPUT_BEGIN_WORK	ONLINE_WORKING
ONLINE_WORKING	INPUT_END_WORK	ONLINE

Tabelle 7-3 Übergangsrelation des Zustandsautomaten

7.4.3 Interaktion mit JTAPI-Basissystem

Wie eingangs erwähnt, erfolgt die Interaktion mit dem Zustandsautomaten durch die Komponente *Call Control*, die wiederum einerseits durch ein Observerobjekt Ereignisse des JTAPI-Systems beobachtet und über ihre Schnittstelle auch die Anfragen der Clientanwendung auswerten kann. Beide Arten von Ereignissen wandelt sie dann in entsprechende Eingaben für den Zustandsautomaten um. Unter welchen Voraussetzungen welche Eingaben generiert werden müssen, zeigt Tabelle 7-4. Hier ist zuerst einmal das JTAPI-Ereignis relevant. Verwendet man dazu ein Observerobjekt vom Typ *Call Observer*, so bekommt man alle Ereignisse eines Anrufs, der im JTAPI-System immer aus zwei oder mehreren Verbindungen besteht. Die Information, ob es sich nun um die lokale oder eine der entfernten Verbindungen handelt, muss daher ebenfalls berücksichtigt werden, etwa um zwischen INPUT_ALERT und INPUT_CALLING zu unterscheiden. Durch die reine Interpretation der Ereignisse lassen sich im Grunde alle Eingaben bzgl. der Grundfunktionalität ableiten. Bei komplizierteren Vorgängen wie etwa der Weiterleitung oder der Konferenz geht das nicht mehr, da das Telefonsystem hierzu keine ausreichenden Informationen liefert. Hierzu ist dann auch der jeweilige Aufruf eines Dienstes durch den Client entscheidend.

JTAPI-Ereignis	Verbindung	Weitere Voraussetzungen	Aufruf durch Client	Eingabe für Automat
ConnAlertingEv	Local			INPUT_ALERT
ConnAlertingEv	Remote			INPUT_CALLING
ConnConnectedEv	Remote	Verbindung ist im Zustand *Alerting*		INPUT_ANSWERED
ConnConnectedEv	Local	Verbindung ist im Zustand *Alerting*		INPUT_ACCEPT
ConnDisconnectedEv	Local			INPUT_HANGUP
CallCtlTermConnHeldEv	Local			INPUT_HOLD
CallCtlTermConnTalkingEv	Local			INPUT_RECONNECT
ConnDisconnectedEv	Local			INPUT_HANGUP
			initTransfer	INPUT_INIT_TRANSFER
			initConference	INPUT_INIT_CONFERENCE
			cancelTranser	INPUT_CANCEL_TRANSFER
			cancelConference	INPUT_CANCEL_CONFERENCE
ConnDisconnectedEv	Local_1 Local_2	System hat Transfer über zweiten Anruf eingeleitet		INPUT_TRANSFERRED
TermConnCreatedEv	Neue Verbindungen der Konferenz	System hat Konferenz über zweiten Anruf eingeleitet		INPUT_CONFERENCED

Tabelle 7-4 Eingaben für Zustandsautomaten

Abschließend sei an dieser Stelle noch erwähnt, dass die dargestellte Herleitung der Eingaben für den Zustandsautomaten nur eine der Möglichkeiten ist. JTAPI liefert dazu eine ganze Reihe von Informationen zu Verbindungen, Terminals und logischen Anrufen, die zur Identifikation des aktuellen Systemzustands genutzt werden können. Details dazu sind jedoch nicht Inhalt der Konzeption und bleiben den konkreten Implementierungen überlassen.

Entscheidend ist an dieser Stelle jedoch der Beleg, dass ein aus der Modellierung hervorgegangener Zustandsautomat auch konkret auf Basis einer CTI-Middleware implementiert werden kann.

7.4.4 CORBA Ereignisse

Über die Schnittstelle des zentralen Zustandsautomaten, genauer über die Methode *getStatus,* kann der Client jederzeit den aktuellen Zustand abfragen, muss dies aber aktiv selbst durchführen. Nun ist jedoch die Reaktion auf asynchron auftretende Ereignisse ein wesentliches Merkmal einer CTI-Lösung. Das Framework stellt deshalb einen weiteren Mechanismus bereit, so dass mit Hilfe des Event Service von COR-

BA derartige Ereignisse automatisch an die Applikation übermittelt werden. Es wird dabei das push-Modell verwendet (vgl. 5.2.3.1), bei dem das Framework in der Rolle des Erzeugers immer dann ein CORBA-Ereignis initiiert, sobald sich der Zustand des Automaten verändert hat. Die Komponente des Zustandsautomaten muss daher einen push-Supplier implementieren, die dazu notwendigen Schritte sind in Abbildung 7-15 dargestellt. Nachdem ein Event Channel erzeugt wurde, erhält die Komponente über ein Administrationsobjekt einen Proxy für den Consumer und verbindet sich mit diesem. Daraufhin kann sie, wenn sich der Zustand des Automaten ändert, über diesen Proxy das Ereignis an den Client leiten.

Die Clientanwendung muss daraufhin lediglich einen Verbraucher mit einer entsprechenden Methode implementieren, der auf einkommende Ereignisse reagiert. Analog zum Erzeuger erhält auch der Client über ein Administrationsobjekt einen Stellvertreter für den Erzeuger (ProxyPushSupplier) und verbindet sich mit diesem. Sie muss dann eine Methode *push* bereitstellen, die durch das CORBA-System aufgerufen wird und in der sie dann entsprechend reagieren kann (vgl. Abbildung 7-16).

```
// get an event channel
EventChannel channel = EventChannelHelper.narrow(
              orb.resolveinitialreferences("EventService"));

// get admin objct
SupplierAdmin suppliers = channel.for_suppliers();

// get a proxy consumer
ProxyPushConsumer consumer = suppliers.obtainpushconsumer();

// connect to consumer
consumer.connectpushsupplier(this());

// method to push event to consumer
public void pushEvent() {
        try {
            org.omg.CORBA.Any msg = orb.createany();
            msg.insert_string("ACF Event");
            _consumer.push(msg);
        } catch (org.omg.CosEventComm.Disconnected e) {
            e.printStackTrace();
        }
    }
```

Abbildung 7-15 Implementierung des Ereignis-Erzeugers

```
// obtain an event channel
EventChannel channel = EventChannelHelper.narrow(
                  orb.resolveinitialreferences("EventService"));

// get admin object
ConsumerAdmin consumerAdm = channel.for_consumers();

// get a proxy supplier
ProxyPushSupplier supplier = consumerAdm.obtain_push_supplier();

// connect to supplier
supplier.connect_push_consumer(_this());

// method being called when a new event is being pushed
public void push(org.omg.CORBA.Any data) throws Disconnected {
     // react to this event
     // update GUI for example
     [ ……]
 }
```

Abbildung 7-16 Implementierung des Ereignis-Verbrauchers

Eine Definition und Übertragung weiterer Ereignisse wäre als Erweiterung noch denkbar, ist im Grunde jedoch nicht notwendig, da der Automat bereits sehr aufwändig ist und auch Zwischenzustände wie etwa ALERTING oder CALLING bereits beinhaltet.

7.5 Architektur im Detail

Wendet man die in den einzelnen Abschnitten erarbeiteten Lösungsalternativen auf die eingangs dargestellte Basisarchitektur an, so ergibt sich die endgültige Systemarchitektur des Frameworks, wie sie in Abbildung 7-17 dargestellt ist. In ihr interagieren die einzelnen Komponenten wie folgt:

Client – Call Control

Diese Interaktion ist die wohl zentralste des Frameworks, in der eine Clientanwendung alle Dienste zur Telefonsteuerung nutzt.

Client – State Manager

Der Zugriff auf den Zustandsautomaten durch den Client ist rein lesend und dient dazu, den gerade aktuellen Zustand des lokalen Systems abfragen zu können. Bei Einsatz des Ereignisdienstes von CORBA wird der Client über eine derartige Veränderung automatisch informiert.

Call Control – State Manager

Alle Zustandsänderungen werden über die Komponente Call Control initiiert, die ausgehend von den Ereignissen des JTAPI-Systems die jeweiligen Eingaben für den Automaten generiert.

Call Control – JTAPI

Mit Hilfe einer geeigneten JTAPI-Implementierung implementiert die Telefonsteuerung alle angebotenen Dienste.

Call Control – Call Object Factory

Auf Anfrage ihres Clients kommuniziert die Komponente *Call Control* mit der zentralen Factory, um ein neues Anrufobjekt zu erzeugen bzw. eine bestehende Referenz zu erhalten.

Client – Call Object

Ein Client kommuniziert direkt mit dem für den aktuellen Anruf gültigen Objekt, dessen Referenz er über die Anrufsteuerung erhalten hat.

JTAPI – JTAPI

Die Verbindung zweier lokaler JTAPI-Implementierungen erfolgt proprietär über die jeweils eingesetzte Telefonanlage bzw. ein eigenes Protokoll im Rahmen einer CTI-Anbindung dieser Anlage.

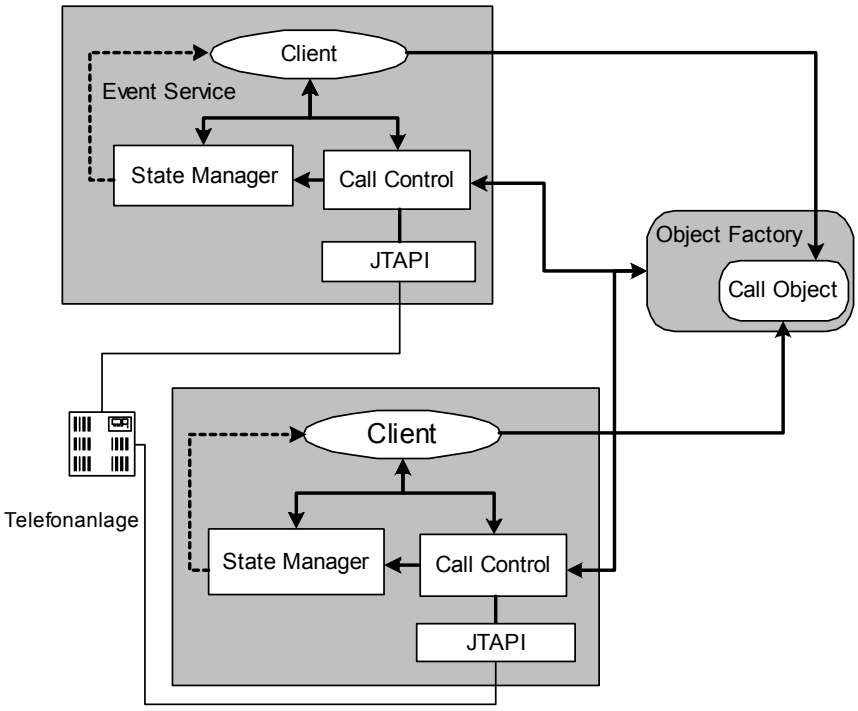


Abbildung 7-17 Systemarchitektur des Frameworks

7.6 Objektmodell

Nachdem die einzelnen Teilkomponenten des Systems grundlegend konzipiert wurden, besteht der nächste Verfeinerungsschritt nun in der Definition eines Objektmodells, das als Basis für eine Implementierung dient. Dabei bleibt die gewählte Aufteilung in die wesentlichen Grundbausteine des Systems, also die Telefonsteuerung, die Verwaltung des Anrufobjekts sowie den erweiterten Zustandsautomaten, bestehen. Diese Komponenten werden nun genauer definiert, wobei insbesondere auch das Zusammenspiel mit CORBA auf der einen Seite sowie mit JTAPI auf der anderen Seite berücksichtigt werden muss. Im Falle von CORBA wird dabei auch bereits die vordefinierte Abbildung von IDL-Schnittstellen auf Java-Klassen (*Java Mapping*) mit betrachtet. Dabei werden IDL-Schnittstellen auf zwei Java-Interfaces abgebildet, ein *signature interface*, das den selben Namen wie die IDL-Schnittstelle trägt sowie ein *operations interface*, das zusätzlich das Suffix *Operations* besitzt. Letzteres beinhaltet dabei die Definition der aus der IDL-Spezifikation abgeleiteten Java-Methoden und muss durch eine geeignete Serverklasse implementiert werden. Die Signaturschnittstelle enthält, wie der Name bereits andeutet, lediglich die Definition des Schnittstellennamens, was dazu genutzt wird, dass man sie auch in weiteren IDL-Definitionen verwenden kann. Sie wird dabei von der jeweiligen Operationsschnittstelle abgeleitet.

7.6.1 Call Control

Abbildung 7-18 zeigt die Klassen der Telefonsteuerung. Die Schnittstelle nach außen wird durch die beiden CORBA-Schnittstellen *BasicTelCtrl* sowie *TelCtrl* definiert, die sich gemäß dem Java-Mapping als jeweils zwei Schnittstellen widerspiegeln[2]. Implementiert werden diese durch die zwei Klassen *BasicTelCtrl_Impl* sowie *TelCtrl_Impl,* die ebenfalls voneinander abgeleitet werden. Hier steckt also die wesentliche Funktionalität dieser Komponente. Um die Ereignisse des JTAPI-Systems zu verarbeiteten, existieren

[2] Der Übersichtlichkeit halber wird die Signaturschnittstelle ohne weitere Methoden nur mittels des Schnittstellensymbols dargestellt, während die Operationsschnittstelle ausführlich als Klasse mit dem Stereotyp „Interface“ dargestellt wird.

zwei Observerklassen die, wie bereits angesprochen, als solche registriert werden müssen und eine Implementierung der Methoden *callChangedEvent* sowie *providerChangedEvent* zur Verfügung stellen.

Ebenfalls dargestellt ist die Interaktion dieser Komponente mit den anderen CORBA-Objekten. Daraus erkennt man, dass sowohl die Implementierungsklassen sowie die beiden Observerklassen Eingaben an den Zustandsautomaten weiterleiten.

Auf die Schnittstelle der Objectfactory bzw. des Anrufobjekts selbst greift hauptsächlich die Klasse *BasicTelCtrlImpl* zu, um dem Benutzer Zugriff auf das Anrufobjekt zu ermöglichen. Daneben hat jedoch auch das Observerobjekt *CC_CallObserver* Zugriff auf diese CORBA-Schnittstellen. So kann bei einem neu eintreffenden Anruf der Observer direkt bei Eingang des entsprechenden Ereignisses das zugehörige Anrufobjekt anfordern. Es steht dann der Clientanwendung bereits zur Verfügung, wenn ihr der Anruf signalisiert wird. Eine derartige Optimierung bietet sich besonders im Call Center an, da bei höherem Anrufaufkommen die Zeit, die zur Verfügung steht, um wesentliche Anruf- und Kundendaten anzuzeigen, meistens sehr knapp ist.

Das Hauptprogramm dieser Komponente liegt in der Klasse *CallControlServer*, in der die *main*-Methode für den Serverprozess implementiert ist und die CORBA-Objekte erzeugt und am System angemeldet werden.

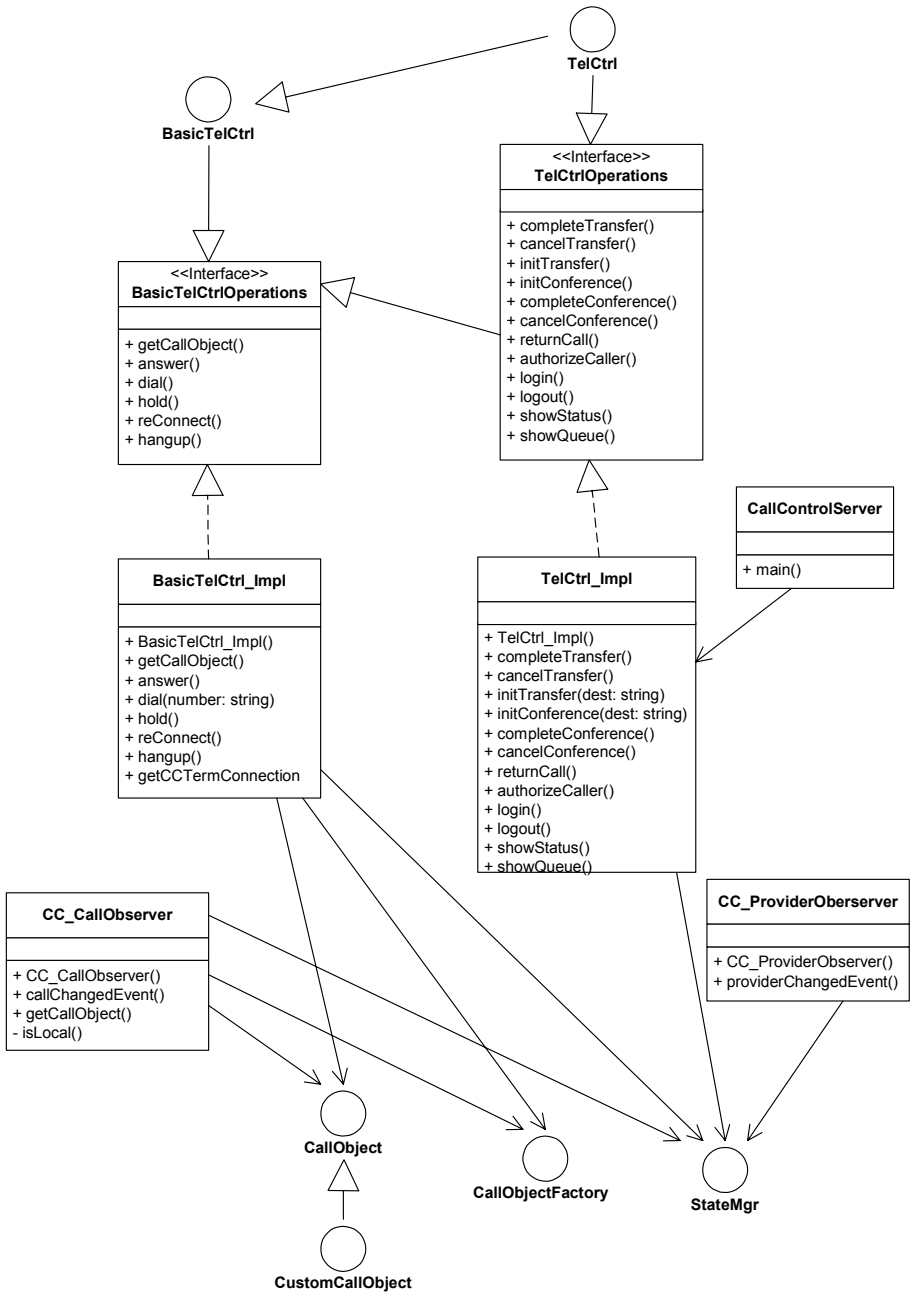


Abbildung 7-18 Objektmodell der Komponente Call Control

7.6.2 Call Object Server

Die Verwaltung des Anrufobjektes sowie die damit verbundene Factory ergibt das in Abbildung 7-19 dargestellte Objektmodell. Die Schnittstellen für das allgemeine und individuelle Anrufobjekt sind sehr einfach, da sie im Wesentlichen nur aus den diversen Attributen bestehen. Lediglich das Basisobjekt enthält, wie bereits erläutert, zwei Methoden, um den Zugriff auf die Informationen für einen Benutzer zu sperren bzw. wieder freizugeben. Damit ist auch die Implementierung dieser Schnittstellen vor allem im Falle des abgeleiteten Objekts sehr einfach, da lediglich die Attribute über die generierten Get- und Set-Methoden übernommen bzw. zurückgegeben werden müssen. Die Namenskonvention von CORBA sieht für diese Methoden den gleichen Namen wie das entsprechende Attribut vor. Somit ist auch gewährleistet, dass eine individuelle Anpassung dieses Objekts durch Hinzunahme weiterer Attribute keinen größeren Ein-

griff in die Implementierung des Frameworks nach sich zieht und auch ohne weitere Detailkenntnisse erfolgen kann.

Verwaltet wird das Anrufobjekt durch die zentrale Factory, die je nach Anfrage neue Instanzen erzeugt und als CORBA-Objekte anmeldet bzw. bei Bedarf auch wieder löscht. Über die Methode *addMember* kann ein neuer Client als zugriffsberechtigt für ein bestimmtes Objekt eingetragen werden und dann selbst auf dieses zugreifen. Identifiziert wird er dabei zum Beispiel über seine im System eindeutige Telefonnummer.

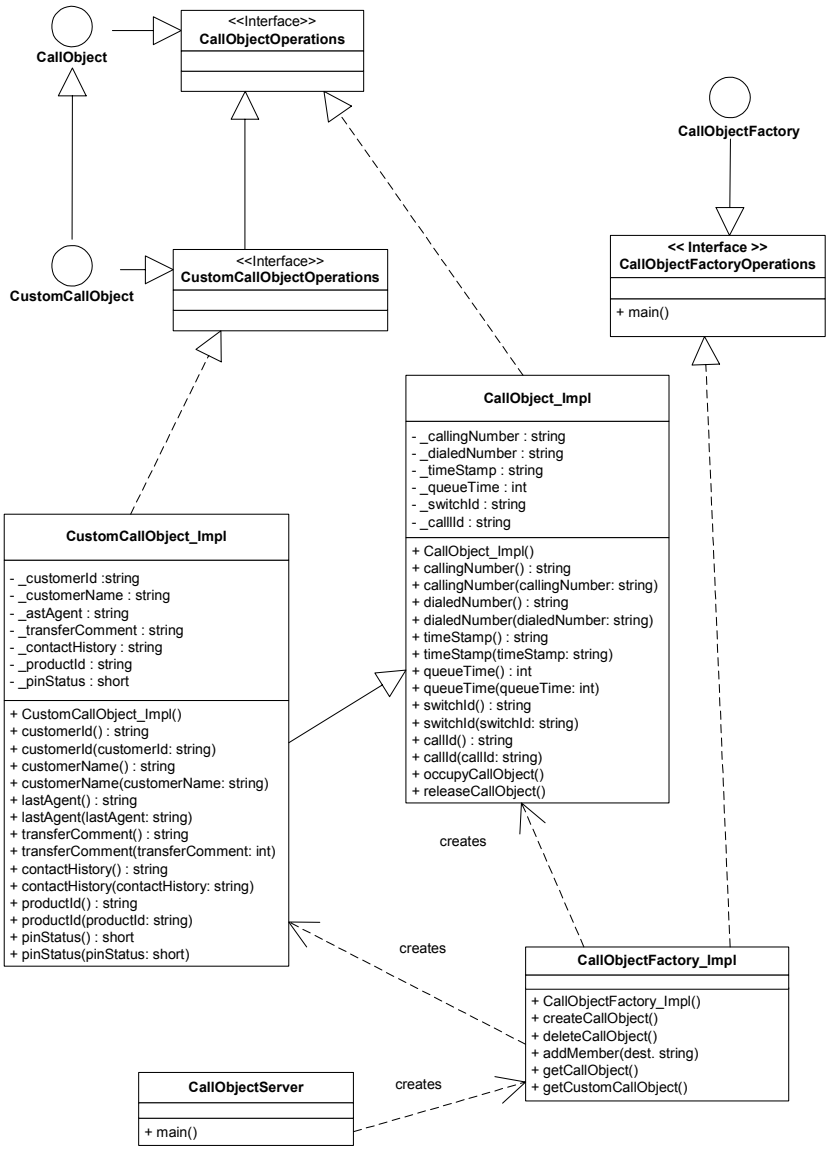


Abbildung 7-19 Objektmodell des zentralen Objektservers

7.6.3 State Machine

Der eigentliche Zustandsautomat wird idealerweise in einer getrennten Klasse implementiert, auf die das CORBA-Objekt auf zwei Arten zugreift, nämlich einmal um neue Eingaben zu übergeben oder um auf Anfrage von außen den aktuellen Zustand abzurufen (Abbildung 7-20). Darüber hinaus kommuniziert sie im Falle eines Ereignisdienstes von CORBA mit dem entsprechenden Supplier-Objekt, dessen Methode *pushEvent* immer aufgerufen wird, sobald sich der Zustand des Automaten verändert hat. Diese Überprüfung findet immer dann statt, wenn über *processInput* eine neue Eingabe verarbeitet wurde. Der Zustands-

automat selbst beinhaltet die in Tabelle 7-3 definierte Übergangsrelation als mehrdimensionales Array, über die der jeweils aktuelle Zustand nach jedem Aufruf der Methode *update* ermittelt wird. Dieser Zustand kann über *getStatus* abgefragt werden.

Instanziert und registriert werden auch hier alle Objekte durch eine eigene Serverklasse.

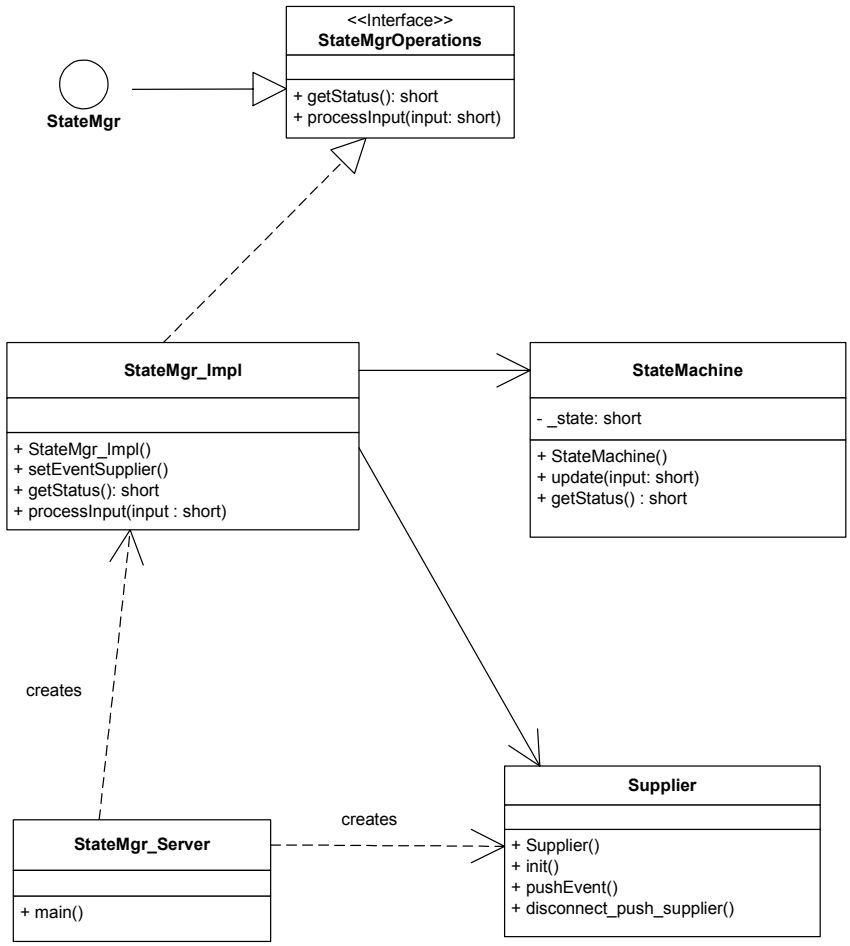


Abbildung 7-20 Objektmodell des Zustandsautomaten

7.7 Zusammenspiel der Komponenten

Um nun zu gewährleisten, dass auf Basis des entworfenen Objektmodells auch die in der Spezifikation definierte Funktionalität des Frameworks realisiert werden kann, werden nun ausgewählte und für die Gesamtfunktionalität repräsentative Abläufe durch dynamische Diagramme nachgebildet. Dadurch wird das Zusammenspiel der einzelnen Komponenten genauer definiert und gleichzeitig überprüft, inwieweit das statische Modell den Anforderungen Rechnung trägt.

Der Übersichtlichkeit wegen und um den Umfang dieses Abschnittes im Rahmen zu halten, werden nur einige ausgewählte Abläufe genauer definiert und aus den Ergebnissen auf die Umsetzung der restlichen Funktionalität geschlossen.

7.7.1 Anrufobjekt

Erster und sehr wesentlicher Ablauf ist die Verwaltung bzw. der Zugriff auf das gemeinsame Anrufobjekt. Abbildung 7-21 zeigt das zugehörige Kollaborationsdiagramm mit den einzelnen Interaktionen der beteiligten Objekte. Ausgangspunkt ist eine Clientanwendung A, die über die Telefonsteuerung ein Anrufobjekt anfordert (Nachricht 1). Ist, wie im Beispiel dargestellt, noch kein derartiges Objekt vorhanden, wird eine neue Instanz über die zentrale Factory erzeugt (2,3). Daraufhin belegt der Client direkt Attribute des Objekts mit individuellen Werten (4,5). Im Falle einer Weiterleitung (6) definiert die Telefonsteuerung dann Client B als weiteren Teilnehmer des Anrufs (7). Dieser ruft nun wiederum das Anrufobjekt ab (8), wobei nun die Telefonsteuerung von der Factory das bereits bestehende Objekt anfordert (9). Client B kann nun wiederum direkt die Werte der Attribute des Objekts auslesen und seinem Benutzer anzeigen (10,11).

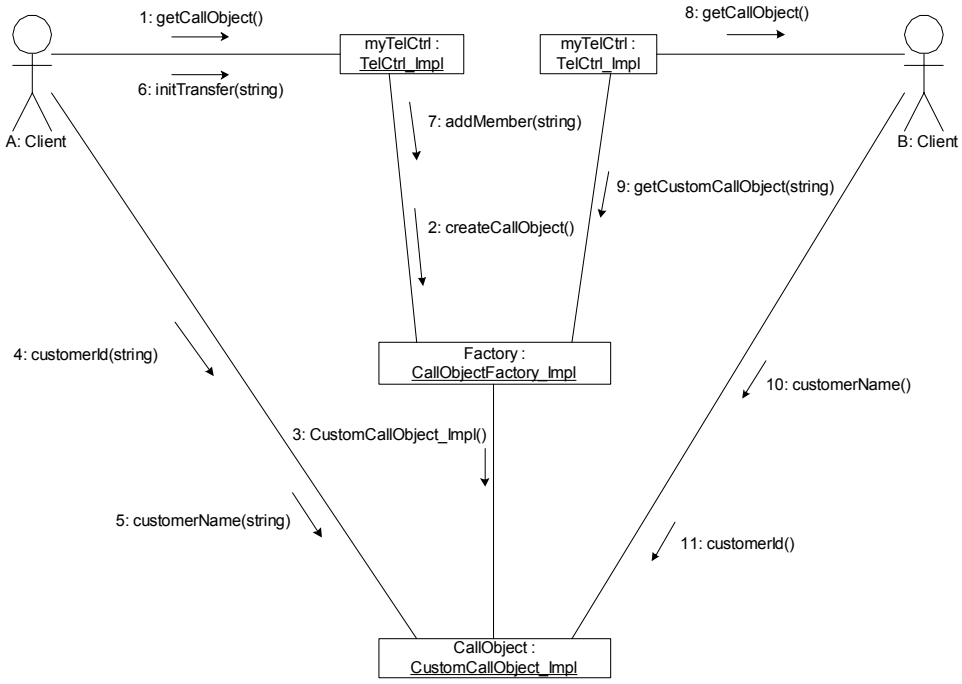


Abbildung 7-21 Zugriff auf gemeinsames Anrufobjekt

7.7.2 Synchronisierter Zugriff auf Anrufobjekt

Nachdem der allgemeine Zugriff auf ein gemeinsames Objekt für den Fall einer Weiterleitung betrachtet wurde, ist das Beispiel eines synchronisierten Zugriffs eine weitere Alternative. Da hier der zeitliche Ablauf stärker im Vordergrund steht, bietet sich zur Darstellung eher ein Sequenzdiagramm an, wie es in Abbildung 7-22 dargestellt ist. Als Akteure existieren nun drei Clientanwendungen, von denen Client A als Initiator das gemeinsame Anrufobjekt wiederum über die zentrale Factory erzeugt. Client B und C haben daraufhin ebenfalls Zugriff auf dieses Objekt, vorausgesetzt sie wurden als gültige Teilnehmer des Anrufs zum Beispiel im Falle einer Konferenz eingetragen. Diese Zwischenschritte, an denen wie bereits gezeigt die Telefonsteuerung beteiligt ist, sind der Übersichtlichkeit wegen hier nicht mehr dargestellt. Nachdem alle drei Parteien Zugriff auf das Anrufobjekt haben, belegt Client A diese Ressource durch Aufruf der Methode *occupyCallObject*. Daraufhin kann dieser Client durch direkten Aufruf einzelner Set-Methoden die Informationen neu belegen. Währenddessen scheitern alle weiteren Aufrufe der anderen Clients, was durch die Ausnahme *ObjectOccupiedException* angezeigt wird. Sie nehmen daher von einer Abänderung der Daten des Objekts Abstand und warten, bis Client A wieder freigibt. Ist dies der Fall (*releaseCallObject* durch Client A), sperrt hier Client B das Objekt, während Client C weiterhin warten muss.

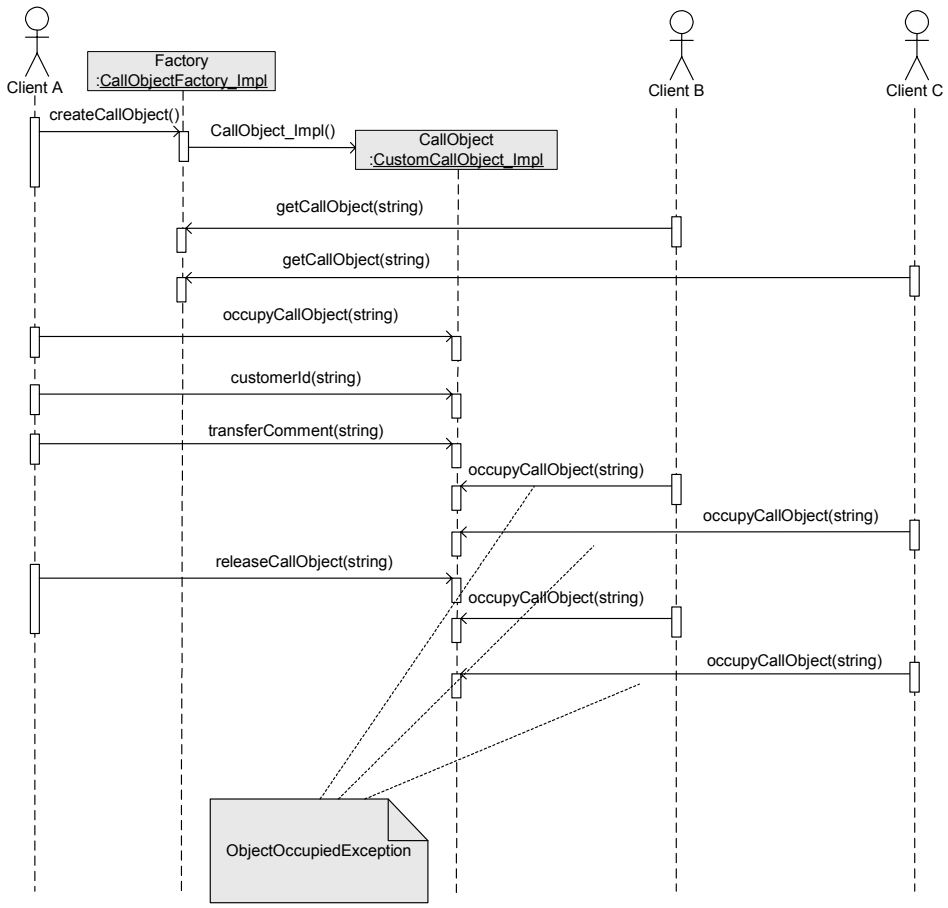


Abbildung 7-22 Synchronisierter Zugriff auf Anrufobjekt

7.7.3 Anruf durchführen

Als zentraler Prozess eines Telefonsystems ist der Vorgang, einen Anruf durchzuführen, sehr geeignet, um das statische Objektmodell auf seine Tauglichkeit zu prüfen. Auch hier bietet sich wiederum ein Sequenzdiagramm zur Darstellung an, wie es in Abbildung 7-23 dargestellt ist. Akteure sind dabei zwei Clients sowie als Vermittler das Telefonsystem selbst. Der Ablauf wird initiiert, indem Client A über seine Telefonsteuerung eine Telefonnummer wählt. Diese erzeugt daraufhin einen JTAPI-Call und veranlasst über die Methode *connect* das System, die Verbindung herzustellen. Das Telefonsystem bzw. die JTAPI-Implementierung meldet diesen Vorgang durch zwei Ereignisse an die Observerklassen der beiden Clients, die wiederum daraus eine neue Eingabe für den Zustandsautomaten generieren. Die Clientanwendung fragt diesen Zustand am StateManager ab und gibt beispielsweise dem Benutzer eine Meldung. In diesem Beispiel nicht dargestellt ist dabei die Weiterleitung des Ereignisses über den CORBA-Eventservice. Nimmt Client B das Gespräch über die Methode *answer* der Telefonsteuerung an, so wickelt diese es über eine der lokal gespeicherten Objekte vom Typ *TerminalConnection* ab. Daraufhin meldet das Telefonsystem wiederum an beide Observerobjekte, dass nun eine Verbindung zustande gekommen ist, diese leiten es wiederum an die jeweiligen Zustandsautomaten weiter und beide Clientanwendungen können den aktuellen Zustand ablesen.

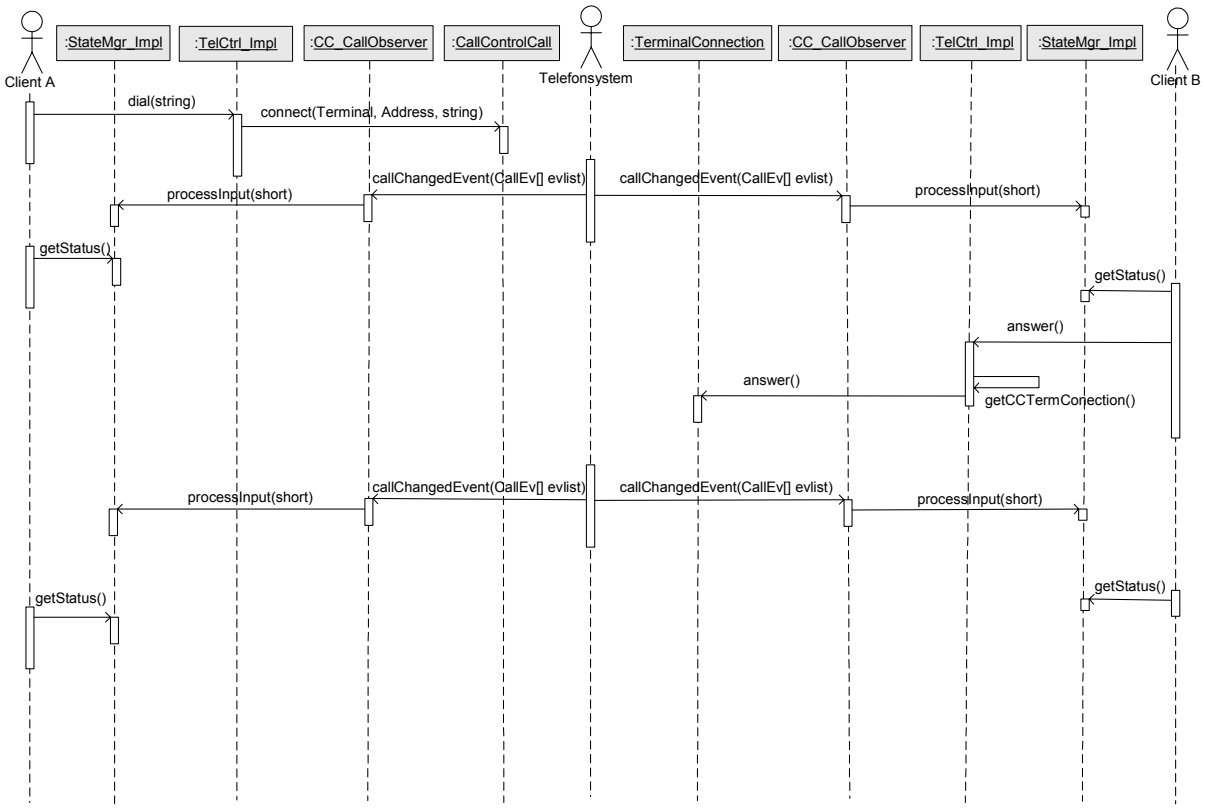


Abbildung 7-23 Anruf durchführen

7.7.4 Weiterleitung

Als Erweiterung des einfachen Anrufs ist der Vorgang einer Weiterleitung ein Ablauf, der repräsentativ für komplexere Vorgänge ist, die das Framework unterstützen muss. Das zugehörige Sequenzdiagramm ist in

Abbildung 7-24 dargestellt. Da die Menge aller hierbei betroffenen Objekte für eine übersichtliche Darstellung zu umfangreich ist, werden nur die wichtigsten Schritte der Weiterleitung skizziert. Es wird weiterhin angenommen, dass Client A bereits eine Telefonverbindung zu einem anderen Teilnehmer aufgebaut hat.

Der Client initiiert die Weiterleitung über seine lokale Telefonsteuerung, die daraufhin ein neues JTAPI-Anrufobjekt erzeugt und über dies mit der Methode *consult* eine Verbindung zum Client B aufbaut. Weiterhin wird diese Aktion als entsprechende Eingabe auch dem Zustandsautomaten gemeldet. Das JTAPI-System führt daraufhin die intern notwendigen Aktionen aus und meldet dies als Ereignis beiden Parteien. Dadurch verändern sich die Zustände der jeweiligen Automaten, was beide Client B im Beispiel durch aktive Abfrage (*getStatus)* zur Kenntnis nimmt. Nimmt nun Client B das Gespräch an (*answer)*, so erfährt dies Client A ebenfalls durch ein erneutes Ereignis bzw. den veränderten Zustand seines Automaten. Zu diesem Zeitpunkt sind die Teilnehmer A und B miteinander verbunden, der dritte Partner ist auf Halten gestellt. Client A kann nun die Weiterleitung durchführen, was seine lokale Telefonsteuerung durch einen Aufruf der Methode *transfer* umsetzt. Daraufhin ist Client B mit dem Gesprächspartner von A verbunden und das JTAPI-System meldet wiederum allen Parteien den veränderten Zustand durch die entsprechenden Ereignisse.

Eine Weiterleitung eines Telefongesprächs kann also wie hier skizziert durch das Framework abgewickelt werden, wobei erwähnt sei, dass die dargestellte Umsetzung, insbesondere die Verwendung der Methode *consult* nicht als zwingend für eine Implementierung betrachtet werden sollte, sondern eine der vielen Möglichkeiten darstellt, einen Transfer zu realisieren. Die Interaktion der Komponenten des Frameworks wird dabei jedoch nicht beeinflusst.

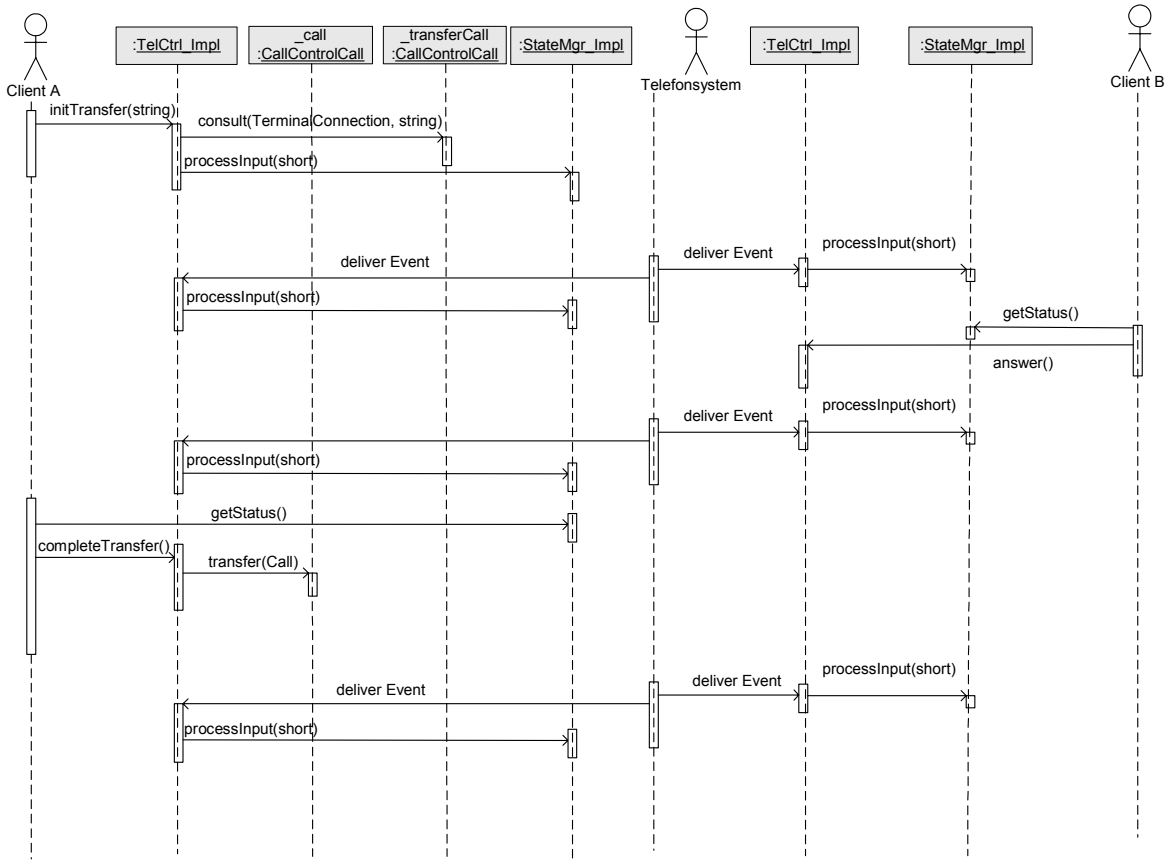


Abbildung 7-24 Weiterleitung

7.7.5 Konferenz

Dritter grundlegender Ablauf ist die Telefonkonferenz zwischen drei oder mehreren Parteien. Da auch dies eine wesentliche Funktionalität des Frameworks darstellt, ist eine mögliche Umsetzung in einer Implementierung hier zu betrachten. Abbildung 7-25 zeigt anhand eines Kollaborationsdiagramms die wichtigsten Schritte, die dazu unternommen werden müssen. Auch hier gilt wieder die Voraussetzung, dass zwischen Client A und Client B bereits eine Telefonverbindung besteht, die sich in den Objektinstanzen *currentCall* beider Parteien widerspiegelt. Client A initiiert nun eine Konferenz über seine lokale Telefonsteuerung (1), die daraufhin einen neues Anrufobjekt erzeugt und über dieses eine Verbindung zu Client C herzustellen versucht (2,3,4). JTAPI meldet diesen neuen Anruf als Ereignis der Telefonsteuerung von Client C, die daraufhin den lokalen Zustandsautomaten aktualisiert (5,6). Sobald die Anwendung diese Veränderung registriert hat (7) und der Benutzer sich entschließt den Anruf anzunehmen, wird die Verbindung zwischen A und C hergestellt (8). Auch hier läuft dann wieder der übliche Austausch von Ereignisinformationen zwischen dem Telefonsystem und allen Observerobjekten der Telefonsteuerung ab, der in der Grafik allerdings nicht mehr dargestellt ist. Sobald sich Benutzer A dann entschließt die Konferenz zu etablieren, ruft die Anwendung dazu die Methode *completeConference* (9) auf, wodurch das Framework veranlasst wird, seine beiden existierenden Anrufobjekte (*_currentCall* sowie *_conferenceCall*) zu einer Konferenz zusammenzulegen (10).

Da die Parteien auch im Falle einer Konferenz Informationen über das gemeinsame Anrufobjekt austauschen und hier auch primär die Funktion einer Synchronisation gefragt ist, ist dieser Ablauf im Zusammenhang mit den Abschnitten 7.7.1 sowie 7.7.2 zu sehen, in denen diese Aspekte angesprochen wurden.

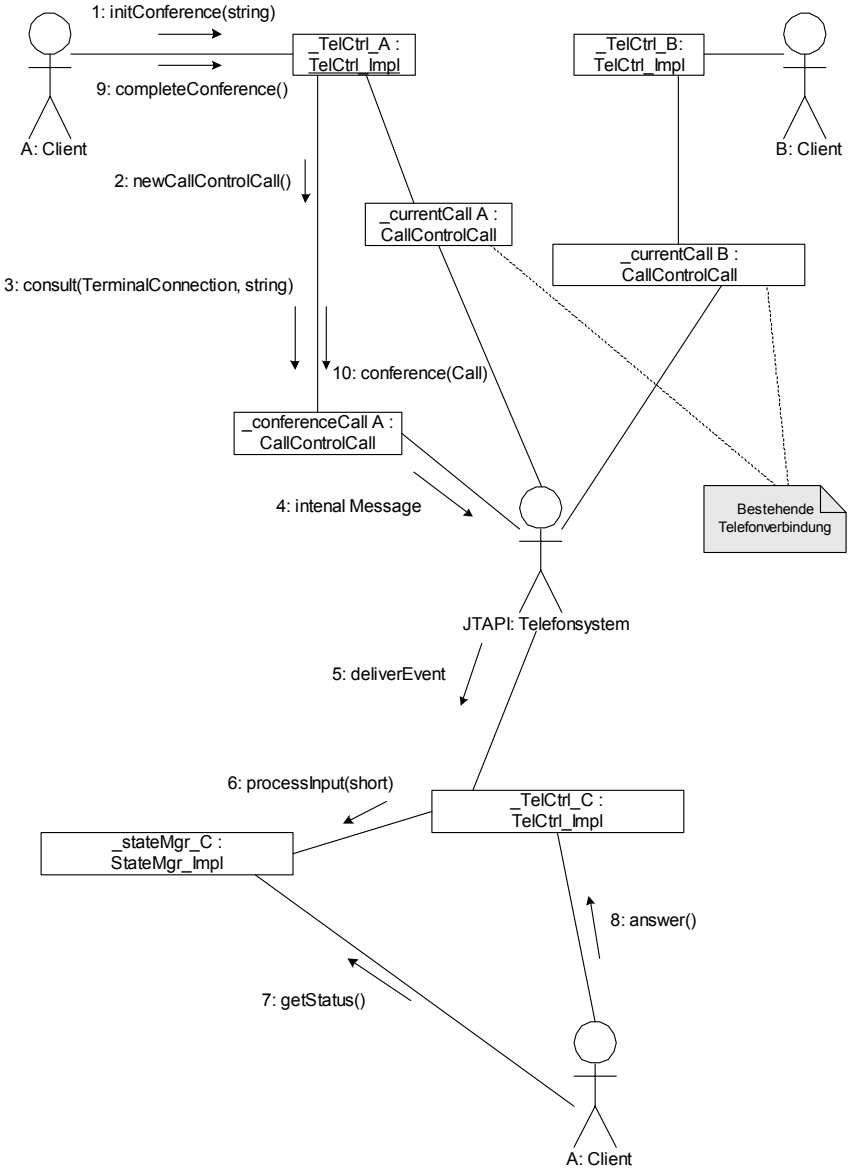


Abbildung 7-25 Konferenz

7.7.6 Ereignisse

Abbildung 7-26 zeigt abschließend noch das Zusammenspiel der beteiligten Komponenten, wenn ein Ereignisdienst für die Anbindung des Zustandsautomaten an den Client verwendet wird. Das Observerobjekt wird auch hier durch das Telefonsystem (genauer den JTAPI-Provider) über ein neues Ereignis dadurch informiert, dass seine Methode *callChangedEvent* aufgerufen wird (1). Es wertet die übergebenen Informationen aus und generiert die richtige Eingabe, die es an den Zustandsmanager übergibt (2). Dieser bringt daraufhin seinen internen Zustandsautomaten auf den neuesten Stand und ruft, sofern sich dieser verändert hat, die Methode *pushEvent* seines lokalen Supplier-Objekts auf (3). Dieses übergibt diesen Aufruf an ein erzeugtes Proxy-Objekt weiter, worauf der Ereignisdienst veranlasst wird, alle Informationen an alle registrierten Consumer-Objekte weiterzuleiten (4). Es ruft also im konkreten Fall die Methode *push* des Consumers eines Clients auf. In dieser Methode wird dann individuell auf dieses Ereignis reagiert, zum Beispiel indem das Ereignis optisch an der Oberfläche signalisiert wird (5,6,7).

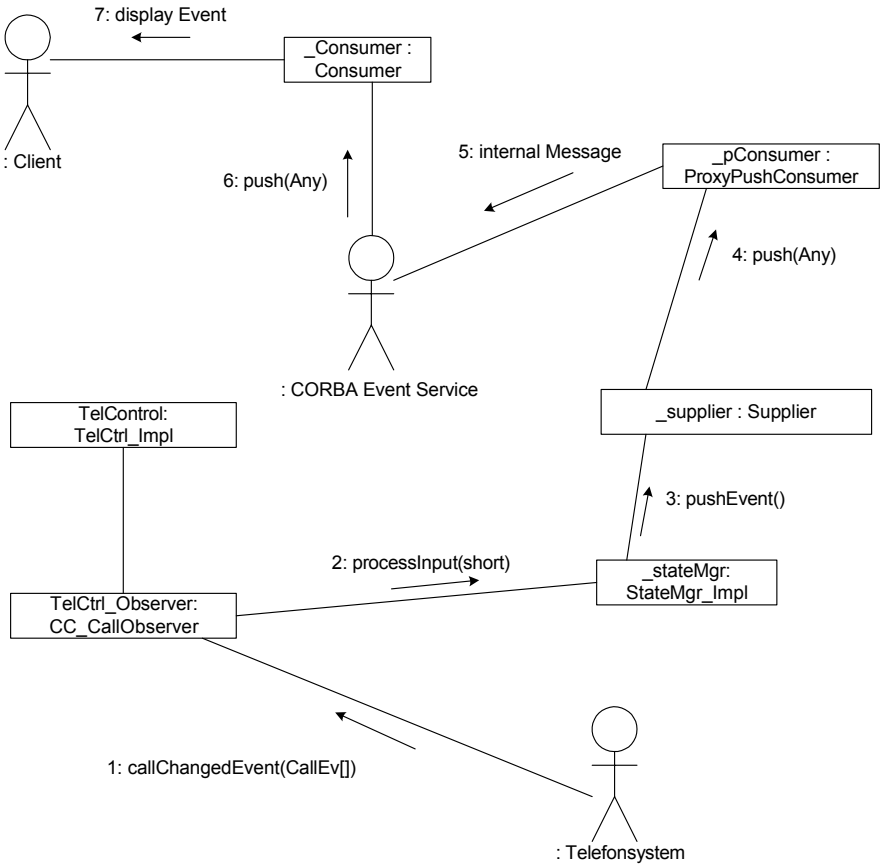


Abbildung 7-26 Zustandsübergang mit Ereignisdienst

7.7.7 Zusammenfassung

In diesem Abschnitt wurde das entworfene Objektmodell den Anforderungen der Spezifikation gegenübergestellt, indem wesentliche und für die gesamte Funktionalität repräsentative Abläufe exemplarisch umgesetzt wurden. Dabei wurden die Verwendung eines gemeinsamen Anrufobjekts sowie der synchronisierte Zugriff darauf betrachtet sowie wesentliche Dienste bzgl. der Telefonsteuerung, nämlich das Durchführen eines Anrufs, einer Weiterleitung sowie einer Konferenz. In allen Szenarien spielt dabei auch immer die Interaktion mit einem lokalen Zustandsautomaten eine Rolle, so dass auch diese Komponenten nicht außer Acht gelassen wurden. Der Spezialfall einer Übermittlung einer Zustandsänderung mittels des Ereignisdienstes wurde darüber hinaus nochmals gesondert behandelt.

Als Ergebnis kann man festhalten, dass anhand des konzipierten Objektmodells alle Abläufe umgesetzt werden können und somit die Tragfähigkeit des Konzepts grundsätzlich angenommen werden kann. Insbesondere aus der Kombination der einzelnen Szenarien (etwa das Durchführen einer Konferenz sowie der synchronisierte Zugriff auf das Anrufobjekt) lassen den Schluss zu, dass sich der gesamte Funktionsbereich des Frameworks abdecken lässt. Somit stellt das Objektmodell eine richtige Basis für die Realisierung der spezifizierten Funktionalität dar.

7.8 Fehlerhandling

Als letzter Aspekt der Konzeption wird das Fehlerhandling behandelt und damit betrachtet, wie das Framework Ausnahmen (*Exceptions)* und Fehler an die Clientanwendung meldet. Auch hier wird das Prinzip verfolgt, dem Anwender eine möglichst komfortable und einfache Schnittstelle zur Verfügung zu stellen,

die die zum Teil sehr komplexen Fehlersituationen eines Telefonsystems weitgehend verbirgt. Somit sind nachfolgend aufgeführte Fehlermeldungen auch eher allgemein gehalten. Ausnahme hier ist die Komponente der Telefonsteuerung, die für alle Dienste auch die Fehler des zugrunde liegenden CTI-Systems wiedergeben kann, damit im Zweifelsfall eine ausreichend detaillierte Fehleranalyse möglich ist.

Technisch wird das Fehlerhandling durch die Definition von Exceptions in den IDL-Schnittstellen der einzelnen Komponenten realisiert. Durch die Abbildung dieser Schnittstellen auf Programmiersprachen wie Java werden diese dann mit den dort vorhandenen Mechanismen umgesetzt. Im Falle von Java bedeutet dies, dass jede Methode einer Schnittstelle eine Java-Exception mit genau dem gleichen Namen hervorrufen kann, die von einer Clientanwendung entsprechend behandelt werden muss.

Nachfolgend nun die Aufzählung der einzelnen Exceptions, die in den Schnittstellen der unterschiedlichen Komponenten definiert werden.

TelCtrl

`InvalidStateException`
Zeigt an, dass die aufgerufene Methode im derzeitigen Zustand des Systems nicht möglich ist. Beispiel dafür wäre der Aufruf von *Hangup* im Zustand *Offline*.

`CTISystemException (string name, string message)`
Gibt die Fehlermeldung des CTI-Systems wieder, die der Aufruf eines Dienstes hervorgerufen hat. Diese Exception enthält weitere Informationen anhand der beiden Parameter *name* sowie *message*, über die die Originalmeldung eindeutig identifiziert werden kann.

`ServerNotAvailableException`
Da die Telefonsteuerung Zugriff auf die zentrale Objekt-Factory haben muss um einen Anruf mit Informationen zu verknüpfen bzw. diese auszulesen, meldet die Komponente einen Fehler, sobald die Factory nicht verfügbar ist.

StateManager

`IllegalInputException`
Zeigt an, dass die Eingabe im aktuellen Zustand des Automaten ungültig ist. Dies ist im Grunde nur für die Telefonsteuerung relevant, die Eingaben für den Zustandsautomaten generiert.

ObjectFactory und CallObject

`ObjectOccupiedException`
Wird erzeugt, sobald ein Client versucht, ein bereits belegtes Objekt für einen Zugriff zu sperren. Anhand dieser Exception können sich also mehrere Anwendungen bzgl. eines gemeinsamen Anrufobjekts synchronisieren.

7.9 Einsatz von UML

Dass UML eine geeignete Beschreibungsform für objektorientierte Systeme darstellt, ist hinreichend bekannt und wird auch durch den vorliegenden Entwurf bestätigt. An dieser Stelle sollen jedoch der Einsatz sowie die damit verbundenen Methoden des objektorientierten Entwurfs analog zu Petrinetzen an einigen ausgewählten Stellen vor dem Hintergrund der allgemeinen Zielsetzung diskutiert werden.

Wie der Entwurf zeigt, stellen statische Objektmodelle auf Basis von UML ein sehr geeignetes Beschreibungsmittel in diesem Zusammenhang dar, insbesondere da hier mehrere Aspekte berücksichtigt werden können. So können neben Architektur und Design auch die Einbettung in eine bereits vorhandene Umgebung über vorgegebene Schnittstellen exakt definiert werden. Darüber hinaus ist die Möglichkeit, an ausgewählten Stellen technische Details zu integrieren und sich an anderen Stellen nur auf das Modell konzentrieren zu können, für die hier vorliegende Art von Anwendungen hilfreich, die sowohl von Mo-

dellen wie Zustandsänderungen und Telefonabläufen als auch von vorhandenen Technologien wesentlich beeinflusst werden.

Zur Notation der Use Cases muss man anmerken, dass diese zur Definition von Funktionalität eine nur eingeschränkte Aussagekraft besitzen und daher eher für fachliche Abstimmungen mit Anwendern geeignet erscheinen. Für Ansprüche wie in dieser Arbeit sind die Strukturierungsmöglichkeiten zwar hilfreich aber nicht ausreichend und müssen daher zum Beispiel um textliche Beschreibungen ergänzt werden.

Hervorzuheben sind dagegen die dynamischen Modelle als Sequenz- oder Kollaborationsdiagramme. Diese sind gerade für den hier vorliegenden Anwendungsbereich telefonbasierter Software äußerst hilfreich. Dabei trägt die Kombination von statischen und dynamischen Modellen sehr viel zur Verifikation des Ansatzes bei, wenn an ausgewählten Szenarios das Zusammenspiel wesentlicher Komponenten geprüft wird. Zur Verifikation trägt auch positiv bei, dass UML-Modelle in der Regel sehr leicht und automatisch auf Programmiersprachen wie Java abgebildet werden können und so durch einfache Prototypen sehr schnell eine weitere Kontrolle möglich ist. Darin liegt ein Vorteil etwa im Vergleich zu Petrinetzen. Da jedoch Petrinetze gerade im Hinblick auf Zustände in verteilten Systemen mächtigere Möglichkeiten bieten, erscheint die Kombination beider Ansätze dennoch sinnvoll.

Insgesamt ist also festzuhalten, dass UML und hier die statischen Klassenmodelle sowie die dynamischen Sequenz- und Kollaborationsdiagramme für die Entwicklung CTI-basierter, verteilter Systeme zu empfehlen ist. Dabei ist die Möglichkeit, technologische Details und plattformunabhängige Modelle zu kombinieren, hier besonders interessant. Diesen Aspekt greift unter anderem der Ansatz der Model Driven Architecture der Object Management Group in [OMG01] auf.

Damit wird auch an dieser Stelle der Mehrwert, den der Einsatz etablierter Methoden der verteilten Anwendungsentwicklung angewendet auf das Thema CTI mit sich bringt, bestätigt.

7.10 Zusammenfassung

In diesem Kapitel wurde vor dem Hintergrund der Spezifikation ein Konzept für ein objektorientiertes Framework entworfen. Die Konzeption wurde dabei auf Basis der ausgewählten Technologien, insbesondere CORBA sowie JTAPI, durchgeführt. Wesentliches Designziel dabei war, eine Schnittstelle bereitzustellen, die Details des CTI-Systems weitgehend vor dem Anwender verbirgt und dadurch auch prinzipiell neutral vom jeweils eingesetzten Basissystem bleibt.

Erster Schritt dabei war der Entwurf einer verteilten Architektur, die sich auf drei wesentliche Komponenten stützt, eine komfortable Telefonsteuerung, ein erweiterter Zustandsautomat sowie eine Factory für anrufbezogene Objekte. Diese Komponenten wurden daraufhin konkreter betrachtet und spezielle Aspekte, etwa die Nutzung eines gemeinsamen Anrufobjekts oder die Übergabe einer Objektreferenz genauer diskutiert. Im Anschluss daran wurde ein Objektmodell entworfen, dass die Funktionalität des Frameworks weiter verfeinert und auch die Einbindung der CORBA- bzw. JTAPI-Technologie berücksichtigt.
Um die Tragfähigkeit des Objektmodells sicherzustellen, wurden wesentliche Abläufe wie das Durchführen eines Anrufs oder einer Weiterleitung anhand dynamischer UML-Diagramme modelliert und dadurch gezeigt, dass das entworfene Objektmodell als Basis für eine entsprechende Implementierung geeignet ist.

Abschließend wurde das allgemeine Fehlerhandling erläutert, in dem definiert wurde, welche Fehlersituationen durch das Framework an die Clientanwendung gemeldet werden sowie der Einsatz von UML bewertet.

Der vorliegende Entwurf zeigt, wie viel Potential in einem Konzept steckt, das CTI aus dem Blickwinkel der verteilten Anwendungen betrachtet. So ist beispielsweise die Verbindung eines verteilten Objekts mit einem Anruf und die darauf aufbauenden Möglichkeiten für die Anwendungsentwicklung ein Beleg für den Mehrwert, den so eine Vorgehensweise mit sich bringt.

7.10.1 Ausblick

Einige Aspekte, die im Rahmen dieser Konzeption noch nicht angesprochen wurden jedoch durchaus interessant sind, seien hier nochmals im Rahmen eines kurzen Ausblicks zusammengefasst.

Erster Punkt wäre hier die Bewertung der Architektur im Hinblick auf aktuelle Komponententechnologien wie etwa EJB. Diese Alternative wurde ja bereits im Rahmen der Bewertung von Middlewaretechnologien betrachtet. Vor dem Hintergrund des nun fertiggestellten Konzepts kann man hierzu festhalten, dass eine Implementierung auf Basis eines EJB-konformen Applicationservers ebenso denkbar ist. Das Objektmodell könnte dabei ohne größere Veränderungen übernommen werden, die einzelnen Komponenten würden als session-beans implementiert werden. Lediglich die Schnittstellen nach außen müssten gemäß der Spezifikation definiert werden, allerdings mit der gleichen Funktionalität.

Abschließend sei noch erwähnt, dass das Konzept zwar eine individuelle Kontrolle über das Telefonsystem nicht zum Ziel hat, diese Möglichkeit allerdings im Rahmen einer Erweiterung durchaus möglich ist. JTAPI bietet eine Reihe von Funktionen, um Abläufe im Telefonsystem oder aber den lokalen Apparat selbst individuell zu beeinflussen. Um so etwas auch im Rahmen des Frameworks anbieten zu können, würde sich eine weitere Variante der Schnittstelle der Telefonsteuerung anbieten, die abgeleitet von der bisherigen Definition individuelle Funktionen zur Verfügung stellen könnte.

8 IMPLEMENTIERUNG EINES PROTOTYPEN

Ausgehend von den bisherigen Überlegungen sowie der vorgestellten Architektur und des Konzepts sollen nun die wesentlichen Ansätze anhand einer konkreten Implementierung eines Prototypen überprüft und untersucht werden.

Inhalt dieses Abschnitts ist demnach die Dokumentation einer exemplarischen Implementierung eines Frameworks für Call Center.

8.1 Entwicklungsumgebung

Als Basis für die Implementierung wird eine Entwicklungsumgebung gewählt, die den derzeit aktuellen Stand der Technik darstellt und auch für eine Produktivumgebung eingesetzt werden könnte. Einzige Ausnahme stellt das JTAPI-System dar, da hier, wie später noch ausführlich erläutert wird, eine Simulation anstelle eines echten CTI-Systems gewählt wird.

8.1.1 Client-/Server-Systeme

Sowohl für Client als auch für Server wird Windows 2000 als Betriebssystem gewählt, da man hier gerade im Rahmen eines Prototypen flexibel bleibt, vor allem was die Verfügbarkeit von Rechnern angeht. Als Hardwareausstattung wird ebenfalls der aktuelle Standard vorausgesetzt, also ein PC mit Pentium-Prozessor, mindestens 600 MHz sowie 128 MB Arbeitsspeicher. Die Maschinen müssen über ein TCP/IP-Netz miteinander verbunden sein, wobei hier von einem 10 Mbit Ethernet ausgegangen wird.

Für den Client, der über eine graphische Oberfläche verfügt, wird ein 17“ Farbbildschirm vorausgesetzt.

8.1.2 Programmiersprache

Als Programmiersprache wird für alle Komponenten Java in der Version 2 gewählt. Um eine möglichst große Kompatibilität zu wahren, werden nur die in der Java-Entwicklungsumgebung standardmäßig angebotenen Bibliotheken verwendet. Für Komponenten mit graphischer Benutzeroberfläche wird Swing eingesetzt.

Als Entwicklungsumgebung für Java kommt das Werkzeug Visual Cafe in der Version 4 zum Einsatz. Dies erleichtert vor allem im Hinblick auf graphische Oberflächen die Entwicklung und gewährleistet weiterhin bei ausschließlicher Verwendung der Standardbibliotheken volle Kompatibilität mit der Java-Laufzeitumgebung.

8.1.3 JTAPI

Was die Auswahl eines für den Prototypen geeigneten JTAPI-Systems angeht, so sind hier mehrere Aspekte zu berücksichtigen. Für die Implementierung entscheidend ist, welche der spezifizierten Funktionalität die Implementierung umsetzt. Hier wurde ja in den vorangegangenen Abschnitten bereits geklärt, dass das Framework die Funktionalität des Basispakets (*javax.jtapi)* sowie die in dem Paket CallControl und CallCenter enthaltenen Dienste benötigt. Als Version wird JTAPI 1.2 vorausgesetzt, da ein neueres Release (1.3) derzeit zwar in Arbeit, allerdings noch nicht verfügbar ist.

Als Anbieter von JTAPI-Implementierungen kommen in erster Linie die Hersteller von Telefonanlagen oder aber reine CTI-Softwarehersteller in Frage. Von Seiten der Hardwarehersteller wurden zwei Produkte für den Einsatz untersucht, eine Implementierung von Lucent inklusive Simulator einer PBX sowie das Produkt CallPath der Firma IBM. Hierbei war zu erkennen, dass letzteres als echte CTI-Middleware auch im Hinblick auf JTAPI mehr Funktionalität bietet. So unterstützt diese etwa die Operationen *Get-/SetApplicationData*, wenn auch nur für einfache Typen wie *String, Int* oder *Long*. Die im Rahmen dieser

Arbeit verfügbare Lösung von Lucent bietet dagegen derzeit keine Unterstützung für diesen Mechanismus und integriert lediglich den von ihren Switch-Produkten angebotenen Dienst, der sehr eingeschränkte Daten über ISDN direkt mit einem Anruf verknüpft. Problem von CallPath ist jedoch, dass hierfür eine der unterstützten Telefonanlagen zur Verfügung stehen muss und es sich hier in der Regel um größere Anlagen handelt. Hier ist die Verwendung eines Simulators von Vorteil, wenn auch die Variante von Lucent sehr eng an die eigene Hardware angelehnt ist.

Unter anderem auch aus diesen Gründen, die eine Entwicklung von CTI-Systemen durch die eingeschränkten Testmöglichkeiten erschweren, hat eine Entwicklungsabteilung von IBM (IBM Haifa Research Lab) ein Entwicklungs- und Testsystem JtapiChat entwickelt. Dieses System beruht auf der Idee, dass eine Telefonanlage vollständig durch ein sog. Chat-System simuliert werden kann, bei dem sich Teilnehmer eines Anrufs nicht über das Telefon sondern über textbasierte Nachrichten unterhalten können. Auf Basis von Java RMI implementiert JtapiChat alle wesentlichen Funktionen einer Telefonanlage inklusive Weiterleitung und Konferenz und bietet in der letzten Version eine zu JTAPI 1.2 konforme Schnittstelle an.

Der große Vorteil einer solchen Lösung liegt in der Unabhängigkeit von spezieller und im Falle leistungsfähiger Telefonanlagen auch sehr aufwändiger Hardware. Das erleichtert zum einen den Aufbau einer Testumgebung und stellt zum anderen auch sicher, dass neben der reinen JTAPI-Implementierung keine individuellen Erweiterungen bestimmter Hersteller verwendet werden. Damit ist auch die Unabhängigkeit des Frameworks von bestimmten Systemen gewährleistet. Allerdings bringt ein derartiges System auch eine Reihe von Einschränkungen mit sich. So wird das CallCenter-Paket mit Funktionen wie An- bzw. Abmelden von Benutzern an einer ACD nicht unterstützt und auch eher hardwarenahe Funktionen wie etwa das Parken eines Anrufs wurden nicht implementiert. Für die Implementierung des Prototypen erscheint dieses Testsystem dennoch am geeignetsten, da es in erster Linie auf die Umsetzung des Konzepts und nicht so sehr auf den Einsatz in einer realen Umgebung angkommt. Da sich das Framework gegenüber der Telefonanlage einer standardisierten Schnittstelle bedient, besteht hier auch kein Kompatibilitätsrisiko. Ein funktionierender Prototyp sollte in dieser Hinsicht also auch jederzeit in einer Umgebung eines Call Centers ablaufen können.

8.1.4 CORBA

Wie auch schon im Falle von JTAPI wird für CORBA eine Implementierung benötigt, die konform zur CORBA-Spezifikation der OMG in der Version 2.3 ist. Die Auswahl einer konkreten CORBA-Implementierung hängt dabei von mehreren Faktoren ab:

- Verbreitungsgrad und Verfügbarkeit
- Implementierung der benötigten CORBA-Services wie etwa Eventing
- Allgemeine Bewertung des Produkts hinsichtlich Stabilität und Leistungsfähigkeit

Diese Faktoren führen derzeit zu den drei gängigsten Produkten am Markt, die als Alternative zu Verfügung stehen. Zum einen VisiBroker für Java/C++ der Firma Borland, Orbix von Iona oder WebLogic von BEA.

Prinzipiell ist allerdings auf eine möglichst große Unabhängigkeit des realisierten Systems von einzelnen CORBA-Implementierungen zu achten. Auf die Nutzung proprietärer Lösungen sollte daher weitgehend verzichtet werden.

Keine der drei Alternativen weist so gravierende Vor- bzw. Nachteile auf, dass sie hinsichtlich der Anforderungen des Frameworks an CORBA nicht in Frage käme. Für den Prototypen wird deshalb die Implementierung VisiBroker in der Version 4.0 der Firma Borland gewählt, hauptsächlich aus dem Grund, weil hier bereits konkrete Erfahrungen in einigen Projekten vorliegen und eine vollwertige Version dieser Software für die Entwicklung des Prototypen zur Verfügung steht.

8.2 Implementierung

Nachdem die Basistechnologien für den Prototypen zur Verfügung stehen, kann der Prototyp implementiert werden. Die wichtigsten Details der Implementierung sowie seine Funktionalität sind Inhalt dieses Abschnittes. Eine Übersicht aller Klassen und Schnittstellen findet sich im Anhang.

8.2.1 Funktionalität und Abgrenzungen

Anspruch der Implementierung eines Prototypen ist es, einen repräsentativen Ausschnitt der in der Spezifikation definierten Funktionalität umzusetzen. Sie ist also zuerst einmal den Anwendungsfällen aus dem Abschnitt 6.4 gegenüberzustellen. Tabelle 8-1 zeigt, welche Use Cases in der Implementierung berücksichtigt werden. So sind alle Basisdienste sowie das Management einer Konferenz und einer Weiterleitung umgesetzt. Der Spezialfall einer direkten Weiterleitung, der ja lediglich eine Zusammenfassung der anderen Dienste darstellt, wird dagegen ausgespart ebenso wie das Zurückgeben eines Anrufs. Auch stellt die Autorisierung über einen Sprachcomputer im Sinne der Telefonfunktionalität keine echte neue Anforderung dar. Hier ist jedoch eine weitere Anwendung notwendig, die den Status eines Anrufs nach erfolgreicher Legitimation entsprechend verändert und das Gespräch dann wieder zurückgibt. Was die Administration angeht, so sind ein An- bzw. Abmelden eines Benutzer an einer ACD für den Prototypen bzw. dem verwendeten Basissystem nicht relevant. Auch kann das JTAPI-System keine Statusinformationen ausgeben, wie es die letzten Anwendungsfälle erfordern würden. Diese Funktionalität ist zwar im Prototypen implementiert, kann aber aufgrund der Einschränkungen des Basissystems nicht getestet werden.

Kategorie	Dienst	Implementiert
Basisdienste	Telefonnummer wählen	✓
	Anruf annehmen	✓
	Gespräch halten	✓
	Gespräch wieder aufnehmen	✓
	Gespräch beenden	✓
Weiterleitung	Weiterleitung einleiten	✓
	Weiterleitung abschließen	✓
	Weiterleitung abbrechen	✓
	Gespräch direkt weiterleiten	--
	Gespräch zurückgeben	--
Konferenz	Konferenz einleiten	✓
	Konferenz abbrechen	✓
	Konferenz herstellen	✓
Legitimation	Anrufer autorisieren	--
Administration	Anmelden	--
	Abmelden	--
Monitoring	Status anzeigen	(✓)
	Warteschlange anzeigen	(✓)

Tabelle 8-1 Funktionalität des Prototypen

Neben der Funktionalität des Frameworks nach außen sind die ebenfalls in der Spezifikation festgelegten Anforderungen an einen zentralen Zustandsautomaten sowie die Verknüpfung von Informationen mit einem Anruf noch für den Prototypen zu berücksichtigen. In diesen Bereichen sind im Grunde keine Einschränkungen bzgl. der Vorgaben nötig, das heißt der Zustandsautomat bietet die geforderte Schnittstelle und berücksichtigt alle definierten Zustände und auch die Definition eines Anrufobjekts folgt ziemlich genau den Vorgaben.

Anforderungen an Ausfallsicherheit und Stabilität sind vor dem Hintergrund einer prototypischen Implementierung im Rahmen eines *proof of concept* zu sehen, halten also den Anforderungen einer realen Implementierung in der Umgebung eines Call Centers naturgemäß nicht stand. So wird die Verfügbarkeit aller Komponenten sowie der notwendigen Basissysteme für den Betrieb vorausgesetzt und Abweichungen lediglich als entsprechende Fehlermeldung ausgegeben, aber nicht direkt behandelt.

8.2.2 Architektur

Die Architektur der Implementierung besteht analog zur Konzeption aus einem Client, der lokal mit zwei CORBA-Servern kommuniziert (vgl. Abbildung 8-1), dem *CallManager* sowie dem *StateManager*. Darüber hinaus existiert systemweit eine zentrale Objectfactory, die alle Anrufobjekte instanziert und verwaltet. Die Architektur ist also im Grunde mit dem konzipierten Ansatz identisch bis auf die Verbindung zwischen JTAPI-Instanzen, die hier nicht über eine Telefonanlage erfolgt.

Da über CORBA der Zugriff prinzipiell auf alle Objektserver im System möglich ist, muss die Clientanwendung zwischen lokalen und entfernten Servern unterscheiden können. Um hier die Verwendung eines Nameservers und die damit verbundene Administration zu umgehen, wird eine einfache Namenskonvention verwendet, anhand derer lokale und entfernte Server immer unterschieden werden können.

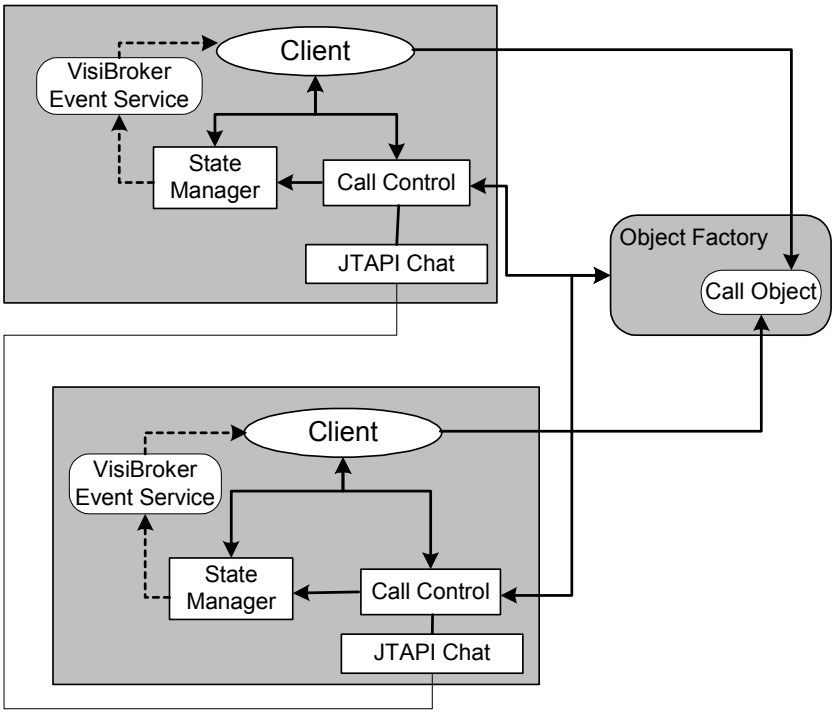


Abbildung 8-1 Architektur des Prototypen

8.2.3 Verwendung von CORBA

Was die Verwendung von CORBA, genauer gesagt die Implementierung VisiBroker angeht, so gibt es im Rahmen des Prototypen einige wichtige Details, die an dieser Stellen angesprochen werden.

Alle Objektserver werden genau einmal manuell gestartet und sind dann verfügbar. Ein Starten mehrerer Instanzen des gleichen Servers aus Gründen der Skalierbarkeit ist für die lokalen Server nicht sinnvoll, für die zentrale Factory wird dieser Aspekt später nochmals betrachtet. Auch ein automatisches Starten durch Aktivierungsprozesse von CORBA (*activation daemon)* ist für die Funktionalität des Prototypen nicht ausschlaggebend. Insgesamt werden folgende POA-Einstellungen verwendet:

Thread policy	ORB_CTRL_MODEL
Lifespan policy	TRANSIENT
Implicit Activation policy	NO_IMPLICIT_ACTIVATION
Request processing policy	USE_ACTIVE_OBJECT_MAP_ONLY

Für die Lokalisierung der einzelnen Server wird zusammen mit der eben angesprochenen Namenskonvention das Tool *SmartAgent* von VisiBroker verwendet, das dem Entwickler die Lokalisierung eines Objektservers innerhalb eines Netzes abnimmt.

Für die Implementierung der Serverschnittstellen generiert VisiBroker aus der IDL-Definition eine Skeleton-Klasse *<interface_name>POA*. Die Implementierung selbst kann von dieser Klasse abgeleitet werden und muss die in IDL definierten Methoden implementieren. Dieser für den Entwickler sehr unkomplizierte Weg funktioniert allerdings nicht mehr, wenn die Implementierung selbst von einer anderen Klasse abstammen muss, da Java keine Mehrfachvererbung unterstützt. In diesem Falle bietet VisiBroker den sog. *Tie-Mechanismus* an. Dabei muss die Implementierungsklasse ein generiertes Java-Interface *<interface_name>Operations* implementieren und wird dann über eine weitere Wrapper-Klasse dem CORBA-System als Servant übergeben. Für den Prototypen wird dieser Mechanismus für die Komponenten *CallControl* sowie *CallData* verwendet, da hier jeweils zwei voneinander abgeleitete Schnittstellen und dementsprechend auch Implementierungen eingesetzt werden.

8.2.4 ACF Call Manager

Die Implementierung dieser Komponente konzentriert sich, wie in der Konzeption bereits angedeutet, auf die Umsetzung der Telefonfunktionen mittels JTAPI sowie das Entgegennehmen von Ereignissen des Telefonsystems und die daraus resultierende Eingabe für den Zustandsautomaten. Die Implementierung enthält dabei auch zwei Klassen, die voneinander abgeleitet einmal die Basisdienste sowie einmal die erweiterten Dienste beinhalten. Sie enthält also eine ziemlich genaue Umsetzung des Objektmodells sowie der dynamischen Abläufe aus dem Konzeptabschnitt.

Für die Umsetzung der Telefonfunktionen greift die Implementierung auf die grundlegenden JTAPI-Objekte wie *Call*, *Connection* oder *TerminalConnection* zurück, die als lokale Membervariablen gehalten werden. Abbildung 8-2 zeigt die wichtigsten Ausschnitte der Implementierung des Dienstes *dial*. Als erster Schritt wird dabei ein Objekt des Typs *CallControlCall* erzeugt. Besteht zu diesem Zeitpunkt bereits ein Objekt mit Informationen zu diesem Anruf, so wird der Factory der neue Teilnehmer mitgeteilt. Damit ist die später noch genauer beschriebene Verknüpfung von Daten mit einem Anruf in der Factory möglich. Durch den Aufruf *connect* wird schließlich JTAPI veranlasst, die Verbindung herzustellen.

Da die JTAPI-Implementierung Kommunikationspartner anhand ihrer IP-Adresse identifiziert, verwendet auch der Prototyp IP-Nummern anstatt Telefonnummern.

```
Public void dial(String number) {
     //
     //   Create the telephone call object
     //
     try {
         _call = (CallControlCall) _jtapiProvider.createCall();
         System.out.println("Created call " + _call);
     } catch (Exception excp) {
         // Handle exceptions
     }

     // add destination as member for call data
     if (_callObject != null) {
         factory.addMember(callObject,number);
        }

     //
     //    Place the telephone call.
     //

     try {
         _connections = _call.connect( _terminal, _address, number );
     } catch (Exception excp) {
         // Handle exceptions
     }
}
```

Abbildung 8-2 Codebeispiel „Anruf durchführen“

Auch die zweite Aufgabe, nämlich das Überwachen von Ereignissen anhand eines Observerobjekts, ist in Abbildung 8-3 in Ausschnitten dargestellt. In der Methode *callChangedEvent*, die bei jedem neuen JTAPI-Ereignis aufgerufen wird, werden diese analysiert und ausgewertet. Dabei werden als erster Schritt die Verbindung sowie die Adresse, auf die sich dieses Ereignis bezieht, ermittelt. Im Beispiel dargestellt ist die Reaktion auf das JTAPI-Event *ConnAlertingEv*. Anhand der ermittelten JTAPI-Adresse kann bestimmt werden, ob es sich um ein lokales oder entferntes Ereignis handelt. Im ersten Fall wird die Eingabe INPUT_ALERT an den Zustandsautomaten geschickt, da hier ein Anruf lokal anliegt. Bezieht sich das Ereignis allerdings auf die Gegenstelle, so bedeutet dies, dass dort ein Anruf anliegt, das lokale System also gerade eine Verbindung aufbaut. Die hier korrekte Eingabe ist demnach INPUT_CALLING. In dieser Art werden auch alle übrigen Eingaben anhand der Ereignisse ermittelt, wobei einige der Eingaben auch direkt aus der Anrufsteuerung heraus an den Zustandsautomaten weitergeleitet werden.

```
Public void callChangedEvent(CallEv[] evlist) {...

[ … ]

  for (int I = 0; I < evlist.length; I++) {
          // monitor Connection Events
          CallEv event = evlist[I];
          […]
          if (event instanceof ConnEv) {

              try {
                  connection = ((ConnEv)event).getConnection();
                  addr = connection.getAddress();
                  AdressName = addr.getName();
              } catch (Exception excp) {
                  // handle exception
              }
               switch (event.getID()) {
                   case ConnAlertingEv.ID:
                       System.out.println(msg + "ALERTING");

                       if (isLocal(addr)) {
                           // send ALERT signal to state machine
                           try {
                               sm.processInput(StateMachine.INPUTALERT);
                           } catch (IllegalInputException e) {
                               System.out.println("Illegal input");
                           }
                           // check if call object is available
                           _callObject = CustomCallObjectHelper.narrow(
                                 factory.getCallObject(extension));
                       } else {
                           try {
                               sm.processInput(StateMachine.INPUTCALLING);
                               } catch (IllegalInputException e) {
                               System.out.println("Illegal input");
                           }
                       }
                   break;
                   [ … ]
                   }
          }
  }
}
```

Abbildung 8-3 Codebeispiel Oberserverobjekt

8.2.5 ACF State Manager

Die Implementierung des State Managers besteht aus der Umsetzung der IDL-Schnittstelle, die Eingaben für den Zustandsautomaten entgegennimmt bzw. den gerade aktuellen Zustand ausgibt sowie einer Klasse *StateMachine,* die den eigentlichen Zustandsautomaten enthält. Sie umfasst die in 7.4.2 definierte Relation als mehrdimensionales Array sowie eine Methode *update*, die anhand einer Eingabe den neuen Zustand ermittelt.

Ist ein CORBA-Ereignisdienst wie etwa der Event Service von VisiBroker installiert, so gibt die Komponente jede Veränderung des Zustandautomaten als neues Ereignis an alle registrierten Clients, die als Verbraucher auftreten, weiter. Der Code dazu entspricht dabei ziemlich genau dem in 7.4.4 dargestellten Beispiel.

8.2.6 ACF Call Object

Der Prototyp beinhaltet eine Call Factory, die die Verwaltung von Anrufobjekten an zentraler Stelle übernimmt und implementiert damit eine der im Konzeptionsabschnitt beschriebenen Alternativen. Aufgabe dieser Komponente ist es also, auf Anfrage eines Clients (konkret einer Komponente Call Manager) ein neues Anrufobjekt zu erzeugen und als CORBA-Objekt zu registrieren, eine bestehende Objektreferenz zurückzugeben bzw. ein nicht mehr benötigtes Objekt zu löschen. Für den Zugriff auf ein bereits bestehendes Objekt wird nachfolgend näher beschriebene Strategie verwendet, die eine Übergabe einer Objektreferenz zwischen mehreren Clients über die zentrale Factory ermöglicht. Standardmäßig sollten nicht mehr benötigte Objekte durch den Client selbstständig gelöscht werden. Darüber hinaus enthält die Factory jedoch auch einen einfachen Mechanismus zur Bereinigung (garbage collection), der in zyklischen Abständen nicht mehr referenzierte Objekte automatisch löscht.

Daneben wird jedes Anrufobjekt ebenfalls als CORBA-Objekt realisiert. Dabei teilt sich die Implementierung analog der spezifizierten Schnittstelle in eine Basisversion sowie eine individuell erweiterte Variante auf. Die Basisversion übernimmt dabei neben dem Bereitstellen von Standardinformationen die nachfolgend noch genauer beschriebene Synchronisation von Zugriffen. Die Implementierung des erweiterten Anrufobjekts gestaltet sich daher äußerst einfach, da die Methoden lediglich das Setzen bzw. Auslesen von Membervariablen implementieren müssen. Damit ist auch die Forderung nach einer für den Entwickler einfachen Erweiterung des Anrufobjekts erfüllt.

8.2.6.1 CallObject Protokoll

Für die Übergabe von Objektreferenzen zwischen zwei Clients wird die in 7.2.4 beschriebene Alternative unter Verwendung einer Factory implementiert. Die Möglichkeit, die Objektreferenz direkt per JTAPI an einen Anruf zu knüpfen, scheidet durch die Restriktionen des Testsystems aus. Die Implementierung einer direkten Kommunikation zwischen beiden Clients erscheint vor allem wegen der bereits angesprochenen Adressierungsthematik als zu umständlich im Rahmen dieses Prototypen.

Die Factory verwaltet also eine interne Tabelle, die ein Anrufobjekt einem oder mehreren Teilnehmern zuordnet. Ideal wäre hier die Verwendung einer eindeutigen Call_ID des Telefonsystems, durch die ein Anruf und alle seine Teilnehmer eindeutig identifiziert werden können. Die verwendete JTAPI-Version scheint dies allerdings nicht zu unterstützen und so haben die einzelnen Instanzen des Anrufobjekts bei allen Teilnehmern keinen übergreifenden Identifikator. Die Implementierung verwendet deshalb die Durchwahl des Teilnehmers bzw. alternativ dazu auch die IP-Nummer des Clients als Identifikator und weist sie einem Anrufobjekt zu. Dazu ist es allerdings notwendig, dass die Komponente *Call Manager* des jeweiligen Clients diese Information der Factory mitteilt. Zu diesem Zweck enthält die Schnittstelle eine Methode *addMember*, die ein Anrufobjekt sowie eine Telefon-/IP-Nummer als Parameter enthält. Verwaltet wird die Tabelle in der Factory als Hash-Tabelle, die als Basisobjekt direkt von Java unterstützt wird.

8.2.6.2 Synchronisation

Um die Synchronisation des Zugriffs auf ein Anrufobjekt umzusetzen, werden die beiden Methoden *occupyCallObject* sowie *releaseCallObject* implementiert. Dadurch wird in einer Variablen *_callLock* des Objekts gespeichert, ob und wer dieses gerade für einen Zugriff gesperrt hat. Wichtig dabei ist, dass ein nebenläufiger Aufruf dieser Methoden vermieden wird, da sonst eine korrekte Synchronisation nicht gewährleistet werden kann. Bei praktisch gleichzeitigem Aufruf von *occupyCallObject* kann es nämlich in 2 Threads passieren, dass in beiden Fällen die Variable *_callLock* noch nicht gesetzt ist und beide Aufrufe diese nun neu besetzen. Damit erhalten beide Aufrufe eine positive Meldung, obwohl nur einer gültig ist.

Vermieden wird dieser Fall dadurch, dass CORBA so konfiguriert wird, dass ein Anrufobjekt immer nur genau in einem Prozess ausgeführt wird. Weiterhin ist dafür zu sorgen, dass entweder parallel Zugriffe nicht in mehreren Threads ausgeführt werden oder aber die Zugriffsmethoden mittels der Synchronisationsmechanismen von Java (*synchronize*) entsprechend geschützt werden.

8.2.7 Testclient

Als Beispiel einer Clientanwendung wurde ein Java-Programm mit einer graphischen Benutzeroberfläche entwickelt, das eine Steuerung aller individuellen Funktionen ermöglicht. Ziel dabei war es, die Dienste der Prototyp-Implementierung möglichst direkt und getrennt voneinander aufrufen zu können. Diese Anwendung ist demnach nicht als Beispiel einer typischen Call-Center-Anwendung zu verstehen, da hier in der Regel die typischen Geschäftsprozesse individueller unterstützt werden.

Abbildung 8-4 zeigt die Oberfläche der Clientanwendung, die in drei Bereiche aufgeteilt ist. Im oberen Teil des Fensters kann die Information, die mit einem Anruf verknüpft wird, abgerufen bzw. geändert werden. Dabei können über die dargestellten Eingabemöglichkeiten einzelne Attribute des Anrufobjekts, die Kundennummer, der Kundenname oder aber ein individueller Kommentar gesetzt werden. Die Nummer des Anrufers kann nicht verändert werden und dient lediglich der Information. Über die Get- und Set-Buttons wird das Lesen bzw. Setzen dieser Werte im Anrufobjekt veranlasst. Die Buttons *Occupy* sowie *Release* dienen der Synchronisation und sperren das Objekt bzw. geben es wieder frei.

Der zweite Bereich der Oberfläche dient der Anrufsteuerung und spiegelt alle im Framework angebotenen Dienste wider. Im Eingabefeld *Destination* kann eine Telefonnummer bzw. für die Testumgebung auch eine IP-Nummer eines Teilnehmers eingegeben werden, zu dem ein neuer Anruf, eine Weiterleitung oder aber eine Konferenz aufgebaut werden soll. Die einzelnen Buttons in diesem Bereich initiieren den Aufruf des jeweiligen Dienstes der Anrufsteuerung, so dass damit das Verhalten des Frameworks schrittweise untersucht werden kann.

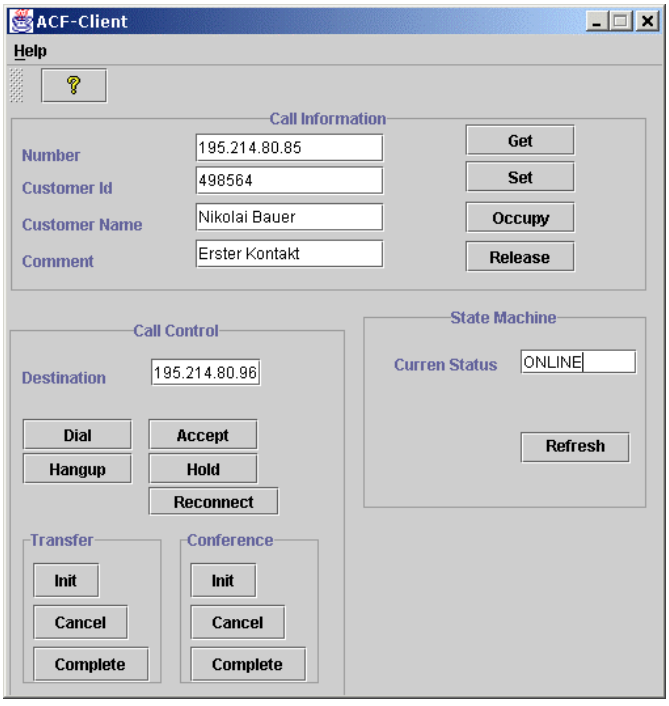


Abbildung 8-4 Oberfläche eines Testclients

Der Bereich *State Machine* ist an den lokalen Zustandsautomaten gekoppelt und gibt den aktuellen Zustand wieder. Ist der Client über den Event Service von CORBA angebunden, so erfolgt die Aktualisierung die-

ser Information automatisch. Ist dies nicht der Fall, so kann die Abfrage auch jederzeit über den Button *Refresh* initiiert werden.

Abbildung 8-5 zeigt, wie derartige Ereignisse grundsätzlich verarbeitet werden. Die Methode *push* der Klasse *Consumer*, die als Verbraucher des CORBA-Ereignisdienstes registriert wurde, wird aufgerufen, sobald sich der Zustand des Systems geändert hat und initiiert dadurch einen Update der Oberfläche. Dieser veranlasst das Hauptfenster, den aktuellen Status des Automaten über CORBA zu ermitteln und dies in einem Textfeld anzuzeigen.

Die Implementierung des Clients stützt sich im Wesentlichen auf Bibliothek *Swing* sowie die darin enthaltenen Java-Komponenten. Die Oberfläche wurde mittels der Entwicklungsumgebung (*form designer)* von Visual Cafe gestaltet, die daraus Java Code erzeugt.

```
============================== Consumer Klasse ==================================
=================================================================================
public class Consumer extends PushConsumerPOA {

    org.omg.CORBA.ORB _orb = null;
    MainFrame _parent = null;

[…]

     public void push(org.omg.CORBA.Any data) throws Disconnected {
         System.out.println("Receiving event");

         // updatem status in main window
         _parent.updateStatus() ;

     }
}
============================== Main Frame des Clients ===========================

public class MainFrame extends javax.swing.Jframe
{
 […]
    public void updateStatus() {
        // to do: code goes here.
     State s = new State(_state.getStatus());
     // display current state
     JtextFieldStatus.setText(s.toString());
    }
}
```

Abbildung 8-5 Ereignisverarbeitung im Testclient

9 BEWERTUNG

In den vorangegangenen Kapiteln wurde ein konkretes Konzept für die Umsetzung eines fortschrittlichen Frameworks entworfen sowie ein Großteil der Funktionalität in einer prototypischen Implementierung umgesetzt. Ziel dieses Abschnittes ist es nun, das Ergebnis einer genaueren Bewertung zu unterziehen.

Für die Bewertung werden mehrere Aspekte in Betracht gezogen. Da der konzeptionelle Ansatz des Frameworks im Vordergrund steht, ist eine Analyse des hier erarbeiteten Ergebnisses vor dem Hintergrund der eingangs gestellten Anforderungen der erste Bereich der Evaluierung. Daraufhin wird der Frage nachgegangen, wie die Nutzung eines derartigen Frameworks durch einen Entwickler aussieht und damit ein möglicher Einsatz in der Praxis betrachtet. Abschließend werden Anforderungen an Skalierbarkeit und Performance diskutiert.

9.1 Konzept

Die ursprünglichen Anforderungen an Konzept und Architektur bestanden darin, einen allgemeinen, von bestehenden Systemen weitgehend unabhängigen und auf einer formalen Basis aufsetzenden Ansatz zu definieren. Das daraus resultierende Framework hat den Anspruch, auch komplexe Telefonfunktionen zu unterstützen, einen allgemeinen Zustandsautomaten zu liefern und eine komfortable und mächtige Verknüpfung von Informationen mit einem Anruf zu realisieren.

Alle drei Aspekte werden im Konzept angesprochen und auch umgesetzt. Durch die Einschalung wesentlicher Telefonfunktionen wird eine Unabhängigkeit von konkreten CTI-Lösungen erreicht und eine Schnittstelle angeboten, die sich stark an den Geschäftsprozessen eines Call Centers orientiert und Abläufe wie Weiterleitung und Konferenz besonders komfortabel unterstützt. Dazu gehört auch der an diese Abläufe angepasste Zustandsautomat, der laufend mit den Abläufen des Frameworks synchronisiert wird. Den funktional größten Mehrwert bietet schließlich die Verknüpfung eines Anrufs mit einem eigenständigen Serverobjekt, da dadurch auch komplexer strukturierte Informationen abgebildet werden können und als CORBA-Objekte flexibel auch unternehmensweit genutzt werden können.

Der Einsatz von CORBA und konkret die Implementierung der lokalen Komponenten Telefonsteuerung und Zustandsautomat als eigenständige CORBA-Server sind ein weiteres wichtiges Merkmal des Frameworks und ermöglichen ebenfalls einen sehr flexiblen Einsatz. So ist der Zugriff auf diese Komponenten von mehreren Anwendungen im Grunde zu jedem Zeitpunkt und auch über Rechnergrenzen hinweg möglich. Damit lässt sich zum Beispiel neben den üblichen CTI-Anwendungen an den Arbeitsplätzen der Mitarbeiter auch eine zentrale Administrationsanwendung aufsetzen, die von einer Stelle aus sämtliche Telefonarbeitsplätze überwachen und gegebenenfalls auch steuern kann. Auch bietet der eigenständige StateManager den Vorteil, dass jederzeit und von mehreren Anwendungen der lokale Zustand abgefragt werden kann, auch wenn die Anwendungen zu einem späteren Zeitpunkt, etwa wenn bereits ein Anruf aktiv ist, gestartet werden.

Verglichen mit bestehenden Lösungen aus dem Bereich der CTI-Middleware stellt das Konzept den Anspruch einer unternehmensweit verteilten Anwendung mehr in den Vordergrund, da hier die einzelnen Komponenten sehr eigenständig sind und vor allem hinsichtlich den Abläufen der Telefonanlage wesentlich stärker abstrahiert wird.

Betrachtet man das Konzept im Hinblick auf mögliche Erweiterungen, so wären hier in erster Linie weitere Dienste der Telefonsteuerung zu erwähnen. Obwohl eines der Designprinzipien ist, kompliziertere CTI-Details vor dem Anwender zu verbergen, kann es in Spezialfällen notwendig sein, an dieser Stelle weitere Funktionen anzubieten. Dazu gehört zum Beispiel die Unterstützung mehrerer, paralleler Telefonleitungen. Was diesbezüglich das Konzept angeht, so würde das keine größeren Änderungen nach sich ziehen, sondern lediglich eine Erweiterung der Dienstschnittstelle bedeuten. Es wäre an dieser Stelle sogar denkbar, dem Entwickler parallel dazu einen direkten Zugriff auf JTAPI zu ermöglichen. Die Erweiterungsmöglichkeiten sind also gegeben.

9.2 Variabilitätsanalyse

Ein hartes Kriterium zur Bewertung von Softwarearchitekturen ist die Variabilitätsanalyse, die das Verhalten des Systems bei Veränderungen der Umgebung untersucht.

In [Siedersl00] werden solche möglichen Änderungen in 3 Gruppen eingeteilt, nämlich in solche, die die Anwendung selbst betreffen (A-Änderungen), solche, die die eingesetzte Technik betreffen (T-Änderungen) und solche, die die externe Repräsentation betreffen, also zum Beispiel die Benutzerschnittstellen. Dieser Ansatz erscheint auch für eine Bewertung des konzipierten Frameworks sinnvoll, allerdings mit einigen Einschränkungen und Erweiterungen, die vorab kurz erläutert werden sollen. Hintergrund der Einteilung ist der Ansatz von Quasar, der zum Ziel hat, eine Verfeinerung der allgemein üblichen Schichtenarchitektur ganz allgemein für Softwaresysteme zu definieren und dabei mit Hilfe generischer Programmierung und neutraler Formate die Qualität zu verbessern. Das vorliegende Framework stützt sich jedoch auf ein bereits entworfenes Modell, das die fachlichen Funktionen sowie die Anforderungen an die Technologie zu einem großen Teil bereits definiert. Somit kann man bei der Bewertung der Architektur auf diese Grundlagen zurückgreifen und deshalb einige der Variabilitätskriterien mit einem anderen Gewicht betrachten. Deshalb werden die nachfolgend diskutierten Änderungen in diesen Bereichen auch hinsichtlich der Wahrscheinlichkeit bewertet, mit der sie eintreten. Die Grundannahme dabei ist, dass der Grad der Variabilität durch die Spezifikation, in die bereits fachliche Erkenntnisse einfließen, entscheidend beeinflusst wird. Damit soll an dieser Stelle nochmals hervorgehoben werden, dass für diese Arbeit neben den allgemeinen Grundsätzen aus dem Bereich Software Engineering die fachlich orientierte Modellbildung für eine bestimmte Anwendungsdomäne, im konkreten Fall Call Center, als notwendig und für die Gesamtqualität entscheidend angesehen wird. Einige interessante Aspekte zu diesem Thema gibt [Glass02]. Dazu gehören auch die in Abschnitt 5.4 gewonnenen Erkenntnisse bzgl. Unabhängigkeit von eingesetzten Technologien. Auch hier sollte berücksichtigt werden, dass nicht jede denkbare Änderung im Gesamtzusammenhang auch möglich oder zumindest sinnvoll ist.

Analog zu den möglichen Typen von Änderungen teilt [Siedersl00] Software in Kategorien ein, je nachdem wie abhängig oder unabhängig sie von Anwendung und Technik ist. Tabelle 9-1 gibt einen Überblick über diese Kategorien und zeigt, wie man die Elemente des Frameworks hier zuordnen kann. Dazu sei noch erwähnt, dass die graphische Oberfläche kein Teil des Frameworks ist, aber hier ebenfalls berücksichtigt wird, da sie ein wesentlicher Bestandteil eines konkreten Einsatzes darstellt.

Kategorie	Beschreibung	Komponenten im Framework
0-Software	Basissoftware, die ein abstraktes Konzept wie etwa einen Zustandsautomaten implementiert	Zustandsautomat in der Komponente *StateMachine*
A-Software	Wird durch Anwendungslogik bestimmt	Implementierung der Telefonsteuerung in Komponente *CallControl*
T-Software	Wird durch Technologie bestimmt	CTI-Anbindung in Komponente *CallControl*
AT-Software	Wird sowohl durch Technologie als auch Anwendungslogik bestimmt	Übergang von Telefonfunktionen und CTI-Anbindung in Komponente CallControl
R-Software	Repräsentiert fachliche Objekte nach außen	Anrufobjekt Graphische Benutzeroberfläche

Tabelle 9-1 Einteilung der Komponenten des Frameworks

Diese Einteilung soll nun als Basis für eine kurze Analyse des entworfenen Frameworks hinsichtlich Variabilität dienen.

Betrachtet man die einzelnen Kategorien, so lassen sich folgende Änderungen identifizieren, die im Anschluss kurz hinsichtlich ihrer Auswirkungen diskutiert werden:

1) T-Änderungen
   a) CTI-System
   b) Middleware
   c) Programmiersprache
   d) Betriebssystem
2) A-Änderungen
   a) Neue Geschäftsprozesse
3) R-Änderungen
   a) Veränderte Information, die mit einem Anruf verknüpft wird
   b) Graphische Benutzeroberfläche

1) Veränderung an Technologien

Das Framework hängt, wie in Kapitel 5 untersucht wurde, neben Betriebssystem und Programmiersprache vor allem vom Einsatz einer geeigneten Middleware sowie einer CTI-Basissoftware ab. In beiden Fällen hat die Untersuchung ergeben, dass eine Reihe von verfügbaren Technologien für den Einsatz geeignet ist. Die Frage ist nun, welche Änderungen des Frameworks hier zu erwarten sind.

1a) Austausch der CTI-Software

Tauscht man die CTI-Software aus, so bedeutet dies, dass die Implementierung der Schnittstelle CallControl sowie die Interaktion des Zustandsautomaten mit dem CTI-System verändert bzw. neu implementiert werden müssen. Wichtig dabei ist, dass die Schnittstelle für den Client in beiden Fällen unverändert bleibt. Auch der Zustandsautomat bleibt davon unberührt, da die Umsetzung auf Ereignisse innerhalb der Komponente CallControl erfolgt.

1b) Austausch der Middleware

Ein Austausch der Middleware etwa von CORBA auf Java RMI oder EJB ist weitgehend problemlos, da die IDL-Schnittstellen allgemein genug spezifiziert sind. Eine derartige Änderung bedeutet aber die Einbettung der bestehenden Implementierung der Serverkomponenten in eine neue Umgebung, was in der Regel eine Neuübersetzung sowie Integration in das neue Middlewaresystem bedeutet. Dies trifft allerdings hier im Gegensatz zur CTI-Technologie auch auf die Clientanwendungen zu, die zwar unveränderte Schnittstellen benutzen, sie aber über die Mechanismen der neuen Middleware ansprechen müssen.

1c) Wechsel der Programmiersprache

Ein vollständiger Wechsel der Sprache etwa bedingt durch eine neue Middleware zieht natürlich die Neuimplementierung aller Komponenten mit sich und ist daher eher ein Extermfall was die Variabilität angeht. Dabei hängt es von der eingesetzten Middleware ab, inwieweit bestehende Komponenten weiterverwendet werden können. Im Falle von CORBA trifft dies für unterstützte Sprachen zu. Hier ist also die Variabilität nicht unbedingt durch die Systemarchitektur sondern durch die definierten Anforderungen an Technologien wie eine Middleware gegeben.

1d) Wechsel des Betriebssystems

Diese Änderung ist im Prinzip unkritisch, sofern der Wechsel des Betriebssystems nicht den Austausch weiterer Technologien zur Folge hat. Insofern lässt sich dieser Punkt also auf die eben dargestellten Punkte 1a) bis 1c) abbilden. In jedem Fall sind die Änderungen jedoch nicht auf konkrete Stellen isoliert sondern ziehen sich durch alle Komponenten des Frameworks. Aber auch hier ist die Wahrscheinlichkeit eines Wechsels in der Regel nicht besonders hoch, da neue Betriebssysteme einen enormen Verwaltungsaufwand mit sich bringen.

2a) Veränderte Geschäftsprozesse

Änderungen in der Anwendungslogik ergeben sich, wenn neue Abläufe oder Geschäftsprozesse unterstützt werden müssen. Dabei sind grundsätzlich zwei Fälle zu unterscheiden. Handelt es sich dabei um

Erweiterungen, die nicht unmittelbar die Integration mit der Telefonanlage berühren, so sind diese als relativ unproblematisch einzuschätzen, da sie durch Implementierung neuer Komponenten realisiert werden können. Ein Beispiel wäre die Erweiterung um Funktionen eines Workflowsystems. Durch den komponentenorientierten Ansatz sowie die Verwendung einer Middleware können solche Funktionen realisiert werden, ohne Änderungen an der Architektur vornehmen zu müssen. Die eventuell notwendige Interaktion mit den bestehenden Elementen (Anrufsteuerung und Zugriff auf Anrufobjekt) erfolgt weiterhin über die vorhandenen Schnittstellen.

Kritischer sind Änderungen, die die spezifizierte Funktionalität des Frameworks betreffen, also etwa den Zustandsautomaten oder die Anrufsteuerung. Hier ist nämlich denkbar, dass ein neuer Ablauf, der neue Funktionen und Zustände mit sich bringt, zum Beispiel nur über eine ganz bestimmte CTI-Software realisiert werden kann. In diesem Fall würden also in der Komponente Call Control AT-Änderungen anfallen, was die Variabilität des Ansatzes entscheidend einschränkt. Diesem Aspekt könnte man nur begegnen, indem man hier einen generischen Ansatz wählt. Dieser Fall wird jedoch dadurch unwahrscheinlich, da eine sorgfältige und fachlich fundierte Spezifikation Basis des Systems ist, die relativ allgemeingültig die notwendigen Funktionen definiert. Da sich die definierten Funktionen eng an den Möglichkeiten einer Telefonanlage orientieren, ist die Variabilität an dieser Stelle eher gering. Dennoch liegt hier eine potentielle Schwachstelle der Architektur des Frameworks, die durch die Qualität des Anwendungsmodells ausgeglichen werden muss.

3a) Änderungen der Informationen des Anrufobjekts

Diese Änderungen sind mit einer sehr hohen Wahrscheinlichkeit zu bewerten, da die Informationen bzgl. eines Anrufs in einem Call Center entscheidend von Art und Qualität des Anrufers, des verwendeten IT-Systems und der unterstützten Geschäftsprozesse abhängen. Dementsprechend wurde bei der Konzeption an dieser Stelle auch großen Wert auf Flexibilität gelegt. Ein Vergleich verschiedener Varianten in Abschnitt 7.2 hat hier gezeigt, dass die Lösung auf Basis von CORBA mit einer expliziten Definition individueller Informationen durch Ableitung des Anrufobjekts die flexibelste Variante darstellt. Voraussetzung dafür sind allerdings Grundkenntnisse in der CORBA-Technologie. Neben den Serverobjekten wird eine Änderung der Informationen natürlich noch Anpassungen an der graphischen Benutzeroberfläche nach sich ziehen, um die Daten auch anzeigen zu können. Aber auch diese sind was das Framework angeht als unkritisch einzustufen.

Insgesamt sind also die notwendigen Änderungen am Framework gering und dabei auf ganz konkrete Stellen isoliert.

3b) Änderungen der graphischen Benutzeroberfläche

Ähnlich zu den Informationen eines Anrufs ist die Anpassung der graphischen Benutzeroberfläche mit hoher Wahrscheinlichkeit zu erwarten. Die Änderungen betreffen jedoch das Framework selbst nicht und sind daher unkritisch.

Art der Änderung	Beschreibung	Auswirkungen	Wahrscheinlichkeit
T	CTI-Basissystem	Gering Isoliert	Mittel
	Middleware	Mittel Nicht isoliert	Gering
	Programmiersprache	Hoch Nicht isoliert	Gering
	Betriebssystem	Abhängig von unterstützten Technologien	Gering
A	Telefonie unabhängige Funktionen	Gering Isoliert	Hoch
	Von Telefonie abhängige Funktionen	Hoch Isoliert	Gering
R	Änderungen am Anrufobjekt	Gering Isoliert	Hoch
	Änderungen an graphischer Benutzeroberfläche	Gering Isoliert	Hoch

Tabelle 9-2 Variabilität des Frameworks

Zusammengefasst kann man also festhalten, dass für sich alleine genommen das Framework hinsichtlich der Variabilität einige Kritikpunkte aufzuweisen hat, da nicht alle Veränderungen ohne größere Auswirkungen abgefangen werden können. Betrachtet man jedoch zusätzlich die Wahrscheinlichkeit, mit der einzelne Änderungen auftreten, wobei hier die Untersuchungen bzgl. Technologien sowie die fachliche Spezifikation berücksichtigt werden müssen, so ist das gesamte Verhalten des Frameworks hinsichtlich Veränderungen ausreichend stabil.

9.3 Spektrum und Grenzen der Einsetzbarkeit

Durch die enge Verankerung des Frameworks mit den Abläufen eines Telefonsystems liegt das Spektrum der Anwendungen, für die das System geeignet erscheint, klar im Bereich der klassischen CTI-Anwendungen. Durch den von konkreten Anwendungen abstrahierten Ansatz deckt das Anwendungsgebiet hier allerdings den gesamten derzeit üblichen Bereich ab. So sind Helpdesksysteme oder Telefonbanking ebenso wie Anwendungen automatischer Sprachcomputer (IVR) denkbar. Durch die sehr offenen Schnittstellen und den Einsatz einer standardisierten Middleware eignet sich das System dabei insbesondere für kleinere und sehr dynamische Systeme, etwa für informelle Call Center in normaler Büroumgebung, in denen mehrere Mitarbeiter nach Bedarf ein virtuelles Call Center bilden. Hier ist der verteilte Charakter des Systems ein entscheidender Vorteil. Ebenso interessant ist das System in Kombination mit neuen Technologien wie etwa VoIP sowie die Kopplung von Call Center und Internet.

Klare Grenzen zu verwandten Systemen kann man einerseits in Richtung Groupware-Anwendungen und andererseits in Richtung Workflowsysteme ziehen. Für den ersten Bereich fehlen weitere Funktionen für das gemeinsame Arbeiten an verteilten Ressourcen, im zweiten Fall in erster Linie die Modellierungsmöglichkeit individueller Workflows. Es bestehen aber zu beiden Bereichen durchaus Schnittstellen und Gemeinsamkeiten. Da eines der Grundkonzepte und somit Stärke des Systems die Anbindung an bestehende Systeme ist, wäre in diesen Fällen eine Integration vorhandener Lösungen sinnvoll.

9.4 Beispielhafte Nutzung durch Entwickler

Neben der Bewertung des reinen Konzepts ist der Aspekt der konkreten Anwendung im Falle eines Frameworks ein entscheidendes Kriterium. Zu prüfen ist dabei, inwieweit die Nutzung dem Entwickler einer konkreten Applikation in einem Call Center Vorteile bringt.

9.4.1 Nutzung von Telefonfunktionen

Die Nutzung von Telefonfunktionen ist als einer der Kernbestandteile ein weiterer Bewertungspunkt, was den praktischen Einsatz des Frameworks angeht. Tabelle 9-3 zeigt anhand einiger Beispiele, welche Aktionen bei der direkten Nutzung von JTAPI im Gegensatz zur Nutzung der Frameworkdienste notwendig sind.

Direkte Verwendung von JTAPI	Verwendung des ACF
System initialisieren	
JTAPI-Provider initialisieren sowie Basisobjekte wie *Terminal* und *Address* ermitteln sowie ein geeignetes Observerobjekt registrieren.	CORBA-Server *CallControl* starten
Anruf einleiten	
Erzeugen der notwendigen Objekte wie *Call* oder *Connection* und Herstellen der Verbindung über die Methode *call.connect.*	Aufruf von *CallControl.dial*
Anruf annehmen	
Ereignis über ein Observerobjekt empfangen, zugehörige Objekte wie zum Beispiel die TerminalConnection ermitteln und Anruf über *TerminalConnection.answer* annehmen.	Aufruf von *CallControl.answer*
Weiterleitung	
Zweites Anrufobjekt erzeugen und zum Beispiel über *CallControlCall.consult* eine Verbindung zu neuem Partner herstellen. Die Weiterleitung selbst kann dann über *CallControlCall.transfer* durchgeführt werden.	Aufruf von *CallControl.initTransfer* sowie *CallControl.completeTransfer*

Tabelle 9-3 Vergleich JTAPI und Framework

Bei einfachen Diensten wie MakeCall oder AcceptCall ist der Unterschied noch nicht besonders gravierend, da hier JTAPI praktisch analoge Dienste anbietet. Dies ist jedoch auch nicht anders zu erwarten gewesen, da JTAPI selbst ja bereits eine neuartige, objektorientierte und damit sehr komfortable Schnittstelle zu einem Telefonsystem bietet. Die direkte Verwendung dieser Schnittstelle wurde deshalb ja auch als Konzeptalternative in Betracht gezogen. Der Entwickler benötigt jedoch auch hier bereits Kenntnisse über die Konzepte und das Modell, das hinter JTAPI steht, um die Verwaltung der notwendigen Basisobjekte sowie die Implementierung zumindest eines einfachen Observerobjektes realisieren zu können.

Die Gegenüberstellung zeigt allerdings auch, dass bei komplexeren Funktionen wie etwa der Weiterleitung die Realisierung auch über JTAPI schwieriger wird und entsprechend mehr Wissen über das JTAPI-Modell voraussetzt. Die dargestellte Variante über einen sog. *Consultation Call* ist dabei nur eine der möglichen Realisierungen. Man kann also durchaus erkennen, dass die Benutzung von Telefonfunktionen durch das Framework nochmals vereinfacht wird und vor allem keine Detailkenntnisse über das zugrunde liegende CTI-System vorausgesetzt werden.

Für die korrekte Durchführung von Telefonabläufen ist neben der Nutzung von CTI-Diensten noch die Berücksichtigung des jeweils aktuellen Status des lokalen Systems wichtig und deshalb auch in diesem Zusammenhang von Bedeutung. Der aktuelle Status lässt sich direkt unter JTAPI aus den verschiedenen Objekten *Call, Connection* sowie *TerminalConnection* ablesen, die alle einen eigenen Zustandsautomaten integriert haben (vgl. 0). Die für eine Anwendung individuelle Zustände, etwa die Zwischenzustände bei einer Weiterleitung, müssen jedoch selbst gehalten und mit den Ereignissen von JTAPI synchronisiert werden. Genau dies nimmt das Framework dem Entwickler ab, indem es einen auf die Abläufe eines Call Centers zugeschnittenen Automaten definiert und automatisch durch Verarbeitung relevanter JTAPI-Ereignisse auf dem aktuellen Stand hält. Der Zugriff auf den aktuellen Status ist dabei sehr einfach durch Aufruf einer Methode des CORBA-Objekts zu realisieren. Auch der automatische Abgleich einer Clientanwendung mit dem Zustandsautomaten durch Einsatz des Ereignisdienstes von CORBA ist relativ einfach umzusetzen, wie die Implementierung des Prototyps zeigt. Somit trägt auch die Bereitstellung eines erweiterten

Automaten zur vereinfachten Benutzung der CTI-Funktionalität bei, vor allem für die Entwicklung umfangreicherer Anwendungen, bei denen die CTI-Anbindung nur einer von vielen Aspekten ist und somit tiefere Detailkenntnisse des CTI-Systems nicht unbedingt vorausgesetzt werden können.

9.4.2 Nutzung des Anrufobjekts

Neben der Kontrolle über das Telefonsystem ist die verbesserte Verknüpfung eines Anrufs mit Benutzerinformationen ein weiteres Ziel des Frameworks. Betrachtet man nun dahingehend das Konzept bzw. die Implementierung des Prototypen, so zeigt sich auch hier, dass die Verwendung eines CORBA-Objekts sich als äußerst flexibel und komfortabel erweist. Die Clientanwendung kann über eine Objekreferenz, die sie durch einen Dienst der zentralen Factory erhält, auf die Daten des Anrufs durch Get- und Set-Methoden wie auf lokale Information zugreifen. Dabei können alle in CORBA definierten Datentypen verwendet werden und somit sind aufwändige Formatdefinitionen und entsprechende Prüfungen nicht notwendig. Ein weiterer Vorteil liegt darin, dass das Anrufobjekt als CORBA-Objekt über seine Objektreferenz praktisch beliebig im System zu nutzen ist.

Weiterhin erlaubt die Implementierung der Anrufdaten als eigenständiges CORBA-Objekt die Erweiterung des System um zusätzliche Dienste wie etwa die Synchronisation des Zugriffs auf seine Informationen. Hier sind eine Reihe von zusätzlichen, eventuell sinnvollen Möglichkeiten denkbar, etwa der Abgleich solcher Informationen über CORBA mit Informationen, die sich in Datenbanken oder Legacy-Systemen befinden. Hier liegt also in jedem Fall eine der großen Stärken des Frameworks, das durch den konsequenten Einsatz von CORBA viel Flexibilität und damit viele Möglichkeiten der Erweiterung mit sich bringt. Das bedeutet vor allem auch sehr viel Potential, was die Erweiterungsmöglichkeiten und den Einsatz in großen Systemen angeht.

Für die Entwicklung von CTI-Anwendungen auf Basis dieses Frameworks ist neben der Nutzung wie bereits angesprochen vor allem die individuelle Anpassung ein entscheidender Aspekt. Hier hat das Konzept und vor allem der Prototyp gezeigt, dass dies sehr flexibel und unkompliziert durchgeführt werden kann und vor allem keine tieferen CORBA-Kenntnisse voraussetzt. Die Definition eigener Inhalte erfolgt dabei in der vom Basisinterface abgeleiteten Schnittstelle *CustomCallObject*, wobei hier alle Datentypen von CORBA verwendet werden können. Eine Beschränkung auf einfache Typen besteht dabei grundsätzlich nicht und so können auch mächtigere Konstrukte wie Sequenzen oder Strukturen verwendet werden. Auch die Anpassung der Implementierung gestaltet sich relativ einfach, da der Entwickler lediglich die aus der IDL-Datei generierten Get- und Set-Methoden, die immer den selben Namen wie das Attribut besitzen, zu programmieren hat.

Insgesamt ist festzuhalten, dass das Framework eine Reihe von Möglichkeiten zur Verfügung stellt, die es einer CTI-Anwendung erlauben, sehr flexibel individuelle Informationen mit einem Anruf zu verbinden und durchgängig in einem objektorientierten Ansatz zu integrieren. Auf Basis von CORBA können diese Daten über den reinen CTI-Bereich hinaus auch unternehmensweit im Zusammenspiel mit anderen Systemen gesehen werden.

9.4.3 Fazit

Als Fazit kann man ziehen, dass das Framework dem Anspruch, die Nutzung von CTI-Funktionalität für die Entwicklung von Anwendungen im Call-Center einfacher zu gestalten, durchaus gerecht wird. Durch das Einschalen der wesentlichen Telefonfunktionen und die Kombination mit einem eigenständigen, den Abläufen in einem Call Center angepassten Zustandsautomaten werden komplizierte Details einer CTI-Middleware vor dem Benutzer weitgehend verborgen. Er wird dadurch allerdings auch was die individuelle Nutzung des Telefonsystems angeht in gewissem Maße eingeschränkt, was jedoch durchaus beabsichtigt ist. Zielgruppe des Frameworks sind daher nicht die Speziallösungen aus dem CTI-Bereich sondern die unternehmensweiten Anwendungen, die über CTI ein Telefonsystem mit integrieren.

Gezeigt hat sich auch, dass durch die Verwendung von eigenständigen CORBA-Servern für zentrale wie auch lokale Komponenten eine Steigerung der Flexibilität erzielt werden konnte.

9.5 Leistungsmessung

Ein besonderer Aspekt bei der praktischen Nutzung des Frameworks ist die Performance des Systems vor allem bei hoher Last, die sich im Falle eines Call Centers durch eine hohe Anzahl von Clientanwendungen sowie ein hohes Anrufvolumen widerspiegelt. Ausreichende Systemleistung vor allem der Basissysteme ist gerade hier besonders wichtig, wie eingangs in dieser Arbeit erläutert wurde (vgl. 3.3), da durch den typischen Ablauf eines Telefongesprächs entsprechend kurze Antwortzeiten von der CTI-Anwendung erwartet werden. Somit ist eine Einordnung des Frameworks in dieser Hinsicht ein wichtiger Aspekt.

Erster Ansatz wäre also, eine Implementierung in einer konkreten Produktivumgebung auf das Verhalten bei steigender Last hin zu testen. Erstes Problem dabei ist jedoch, eine repräsentative Testumgebung zu realisieren. Größere Call Center optimieren in der Regel alle beteiligten Komponenten, also Telefonanlage und Hardware der CTI-Server, sehr individuell und abgestimmt auf die jeweiligen Anforderungen. Ergebnisse von Tests in einer derartigen Umgebung sind demnach nicht ohne weiteres übertragbar. Ein weiteres Problem ist, dass die Ergebnisse eines derartigen Gesamttests sehr stark von der Qualität und Performance der JTAPI- bzw. CORBA-Implementierung abhängen. Was CORBA-Implementierungen angeht, so ist hier bereits eine Vielzahl von unterschiedlichsten Performance-Tests durchgeführt worden. Auch der Einsatz in konkreten Projekten ([iSYS98]) hat bereits gezeigt, dass CORBA durchaus den Anforderungen einer Call-Center-Umgebung gewachsen ist. Im Falle von JTAPI ist man hier sehr stark auf die Umsetzung durch einzelne Hersteller angewiesen und damit darauf, wie gut sie die Anbindung an die eigene Hardware implementieren können. Die im Prototypen verwendete Implementierung greift zum Beispiel auf RMI zurück, was teilweise zu Leistungseinbußen führt. Das Framework selbst erhebt jedoch den Anspruch, prinzipiell unabhängig von derartigen Implementierungen realisiert werden zu können, und daher erscheint ein Test des gesamten Systems eher ungeeignet. Nicht zuletzt ist ein Perfomance-Test unter realen Bedingungen auch äußerst schwierig zu realisieren, da hier im Grunde in einem konkreten Call Center mit vergleichbarer Größe getestet werden müsste.

Es wird deshalb ein alternativer Ansatz verfolgt, nämlich die Identifikation der bei hoher Last kritischen Stellen des Frameworks, die daraufhin durch isolierte Lasttests konkret untersucht werden können.

9.5.1 Skalierbarkeit des Frameworks

Betrachtet man die einzelnen Komponenten bzgl. Skalierbarkeit, so sind die lokalen Server, die pro Client nur jeweils einmal vorhanden sind, unkritisch, da sie von einer steigenden Gesamtanzahl von Clients nicht betroffen sind. Auch ein steigendes Anrufvolumen bedeutet für einen einzelnen Client und somit diese Komponenten keine steigende Last. Auch reicht die Interprozesskommunikation völlig für die notwendige Interaktion mit einer Clientanwendung aus, es wäre sogar noch ein Einbinden der Komponenten in einen Prozess denkbar.

Kritisch ist jedoch die Verwaltung der Anrufobjekte über eine zentrale Factory, da hier der Zugriff über das Netz erfolgt. Auch steigt hier die Last mit der Anzahl der Clients und der Anzahl der aktiven Anrufe im System. Außerdem ist eine schnelle Verfügbarkeit der Daten essenziell, da diese in der Regel während der kurzen Phase zwischen Anrufen und Abheben angezeigt werden sollen. Diese Komponente ist also der potentielle Engpass, der die Gesamtleistung des Frameworks beeinträchtigen kann.

Bei steigender Last bietet es sich in diesem Falle also an, mehrere Instanzen der Factory zu betreiben. Dabei muss allerdings geregelt werden, dass sich zwei Teilnehmer eines Anrufs auch an einen gemeinsamen Server wenden. Ansonsten müssten die Instanzen der Factory gegenseitig Informationen über die jeweiligen Objekte und die damit verbundenen Anrufparteien bei Bedarf austauschen, was ebenfalls relativ einfach zu implementieren ist.

Als Fazit kann man ziehen, dass die Skalierbarkeit des Frameworks zum größten Teil an der zentralen Factory hängt und deshalb an dieser Stelle Leistungsmessungen interessant sind.

9.5.2 Last eines Call Centers

Bevor Leistungsmessungen durchgeführt werden ist es sinnvoll, anhand realer Call Center die Grenzwerte zu bestimmen, innerhalb derer sich die Lasttests bewegen sollten. Dabei sind in erster Linie folgende Werte interessant: die Zahl der Mitarbeiter und damit Anzahl der Clients, das Anrufvolumen sowie die durchschnittliche Anrufdauer. Die Ermittlung konkreter Werte aus der Praxis erweist sich dabei als relativ schwierig, in erster Linie weil Unternehmen diese Werte in der Regel nicht veröffentlichen. Außerdem ist die Kapazitätsplanung bei Call Centern nicht immer einfach, da die Auslastung nicht bei 100 % liegen sollte, da sonst der Servicelevel (also die Anzahl der Anrufe, die in einer bestimmten Zeit angenommen werden) zu niedrig ist und sich Warteschlangen bilden. Für das betrachtete System kann aber hier durchaus von voller Auslastung aller Clients ausgegangen werden. [Bittner00] kommt auf eine durchschnittliche Anzahl von 50 Mitarbeitern pro Call Center in Deutschland, eine Zahl, die sich auch mit den Ergebnissen eigener Recherchen deckt. Maximale Werte liegen hier bei etwa 500, zum Beispiel bei rein auf das Telemarketing ausgerichteten Unternehmen. Eine Skalierung des Systems von 10 bis 500 Clients ergibt also einen realistischen Querschnitt. So können 500 voll ausgelastete Clients bei einer Gesprächsdauer von 5 Minuten in 8 Stunden mehr als eine Million Anrufe verarbeiten.

Darüber hinaus gibt es natürlich auch Systeme mit Volumina in Spitzenzeiten von mehreren 10.000 Anrufen. Man findet solche Szenarien häufig bei Aktionen in Verbindung mit anderen Medien, etwa dem Fernsehen. Hier können bei Umfragen etc. durchaus solche Spitzenlasten erreicht werden, deren Großteil dann jedoch von IVR-Systemen abgedeckt wird. Solche Szenarien sind allerdings primär kein Einsatzgebiet für das vorgestellte Framework. Um die volle Funktionalität des Frameworks, etwa die vielseitigen Informationen zu einem Anruf oder die Unterstützung einer Konferenz und Weiterleitung nutzen zu können, sind Anwendungsfälle mit so hohen Anrufvolumen eher ungeeignet. Vielmehr stehen hier kleinere Systeme oder aber auch herkömmliche Büroumgebungen, die erst durch CTI bei Bedarf die Aufgaben eines Call Centers erfüllen, im Vordergrund. Deshalb ist eine so hohe Skalierung eher im Rahmen eines Ausblicks und nicht primär für die Bewertung des dargestellten Ansatzes interessant.

9.5.3 Leistungsmessungen

Um die Leistungsfähigkeit der Factory und damit der Verwaltung der Anrufobjekte zu beurteilen, sind zwei wichtige Aspekte interessant. Zum einen, wie viele Anrufobjekte gleichzeitig verwaltet werden müssen und zum anderen, wie schnell der Zugriff auf die dort gespeicherten Informationen erfolgt.

Ein typisches Szenario, das den Zugriff eines Clients simuliert, ist folgender Ablauf:

```
Anrufobjekt über Factory erzeugen
Daten setzen
Neuen Teilnehmer für Objekt eintragen
SLEEP
Objekt nochmals über Factory abfragen
Daten auslesen
Objekt löschen
```

Damit wird prinzipiell ein Telefonanruf in einem Call Center simuliert, bei dem eine erste Instanz die wesentlichen Informationen im Objekt einträgt und zum Beispiel nach einer Weiterleitung die Daten wieder abgefragt werden. Dabei ist die Wartezeit zwischen den beiden Abschnitten ein wichtiger Parameter, da dadurch die Anzahl der parallel zu verwaltenden Objekte bestimmt wird. Dabei ist die Tatsache, dass hier das Setzen und Abfragen der Informationen innerhalb eines Ablaufes erfolgt, eine Vereinfachung, die jedoch die auf die Factory erzeugte Last nicht wesentlich beeinflusst.

Um relevante Leistungsmessungen durchführen zu können, wird dieser Ablauf innerhalb eines Testclients implementiert. Die Last wird dabei durch parallel ablaufende Threads erzeugt, die den eben beschriebenen Ablauf beinhalten (vgl. Abbildung 9-1). Durch Einstellen folgender Parameter kann das Verhalten des Testclients beeinflusst und damit unterschiedliche Szenarien simuliert werden:

- N: Anzahl der Threads
- Z: Anzahl der Abläufe (Zyklen) pro Threads
- W: Wartezeit zwischen beiden Teilen des Ablaufs
- D: Umfang der im Objekt gespeicherten Daten
- R: Maximale Abweichung der Wartezeit

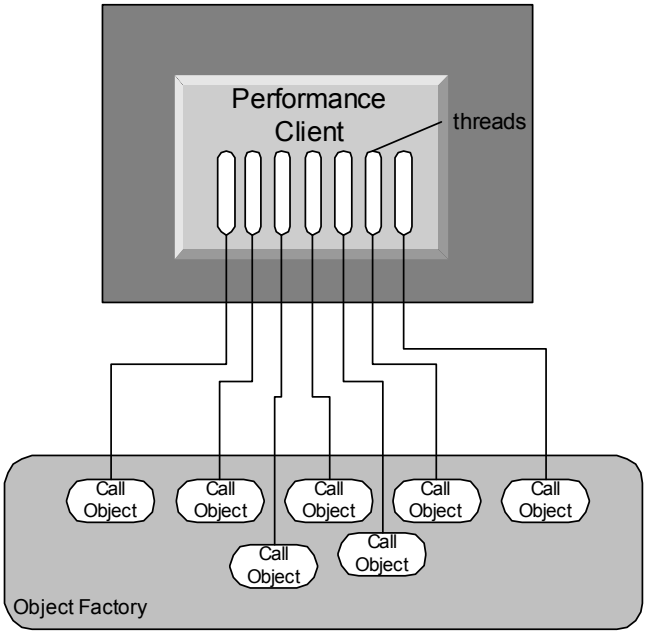


Abbildung 9-1 Architektur für Performance-Messungen

Um eine etwas gleichmäßigere Auslastung des Clientrechners zu gewährleisten, wartet jeder einzelne Thread eine zufällige Zeitspanne von bis zu drei Sekunden, ehe er startet. Der letzte Parameter dient dazu, die Wartezeiten zwischen den beiden Zyklen innerhalb eines Intervalls variieren zu können. Jeder Thread addiert bzw. subtrahiert diesen Wert multipliziert mit einem Zufallswert zwischen 0 und 1 zur eingestellten Wartezeit.

Um eine hohe Last auch mit relativ kurzen Laufzeiten der Threads zu ermöglichen ist es notwendig, dass der Start dieser möglichst gleichzeitig erfolgt, damit eine hohe Nebenläufigkeit erreicht wird, ehe sich die ersten Threads bereits wieder beenden. Im Falle von nur einem Clientrechner reicht dazu das sequentielle Starten nach einer manuellen Eingabe sowie die zufällig ausgewählte Verzögerung aus. Bei Einsatz von mehreren Rechnern muss dazu ein Synchronisationsmechanismus implementiert werden. Für den Performance-Client wurde dazu die Synchronisation über eine gemeinsame Datei gewählt. Dazu wird vor jedem Testlauf ein gesonderter Thread gestartet, der in kurzen Abständen ($^1/_{10}$ Sekunde) prüft, ob eine bestimmte Datei vorhanden ist. Erst wenn dies nicht mehr der Fall ist, werden die eigentlichen Threads gestartet. Somit können verschiedene Clients auch auf mehreren Rechnern, die sich in dieser Warteschleife befinden, durch Löschen einer zentralen Datei ziemlich gleichzeitig gestartet werden.

Während jedes Durchlaufs eines Threads wird an folgenden Punkten die Zeit ermittelt:

Time 1 start
```
Anrufobjekt über Factory erzeugen
Daten setzen
neuen Teilnehmer für Objekt eintragen
```
Time 1 stop
```
SLEEP
```
Time 2 start
```
Objekt nochmals über Factory abfragen
Daten auslesen
Objekt löschen
```
Time 2 stop

Daraus ergeben sich 3 Zeiten, die für jeden Durchlauf eines Threads gespeichert werden, die Dauer des ersten sowie zweiten Teils und die Gesamtdauer eines Durchlaufs. Die eingestellte Wartezeit zwischen beiden Teilen fließt in die Messung nicht mit ein.

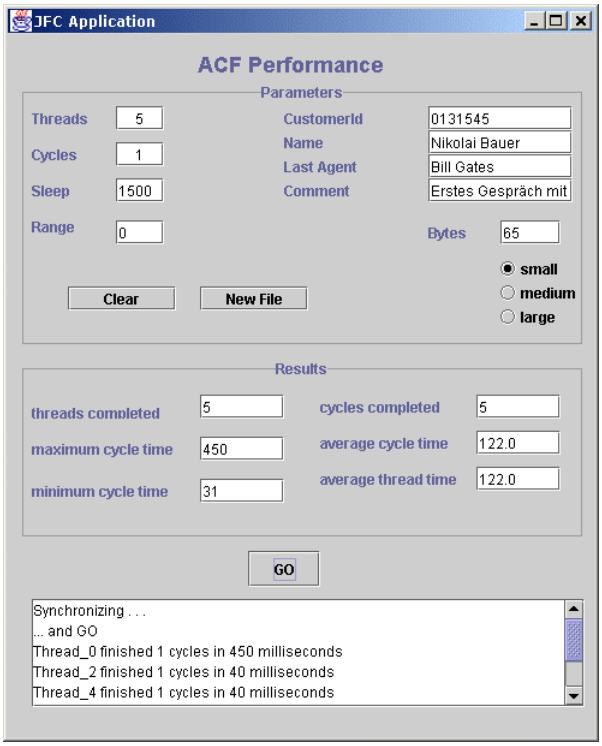


Abbildung 9-2 Testclient für Performance-Messungen

Innerhalb der Factory werden ebenfalls Messungen durchgeführt, allerdings stehen hier keine Zeitmessungen sondern der Verbrauch an Ressourcen im Vordergrund. So ermittelt die Implementierung bei jedem Erzeugen eines Objekts die aktuelle Anzahl verwalteter Objekte sowie den erreichten Maximalwert. Außerdem wird die Schnittstelle der Factory um eine Methode *report()* erweitert, die eine Speicherung folgender drei Messwerte veranlasst:

Anzahl der verwalteten Objekte
Speicherbedarf der Java Virtual Machine
Freier Speicher

Die einzelnen Messwerte sowohl des Testclients als auch der Factory werden in ein Logfile geschrieben und können so nachträglich ausgewertet werden. Abbildung 9-2 zeigt die graphische Benutzeroberfläche des Testclients, über die alle Parameter und damit verschiedene Szenarien gesteuert werden können.

9.5.3.1 Testumgebung

In einer konkreten Testumgebung werden drei der eben beschriebenen Testclients mit einem zentralen Factory-Server verbunden, um in verschiedenen Szenarien Lastprofile erzeugen zu können, die denen eines realen Call Centers nahe kommen. Dazu werden PC-Workstations verschiedener Ausstattung mit Prozessoren zwischen 400 und 733 MHz verwendet. Verbunden sind diese über ein 100Base-T Netzwerk, wobei allen Rechnern durch eine zentrale Switch-Komponente die volle Bandbreite zur Verfügung steht (vgl. Abbildung 9-3). Da auch Call Center in der Regel eine sehr gute Netzwerkinfrastruktur aufweisen, kann diese Umgebung durchaus als repräsentativ eingeschätzt werden. Als Betriebssystem wird auf allen Rechnern Windows 2000 eingesetzt. Alle Clients haben Zugriff auf das Dateisystem des Servers und schreiben dort ebenso wie die Factory ihre Log-Dateien, so dass diese an zentraler Stelle gesammelt und ausgewertet werden können. Über dieses Dateisystem läuft auch die Synchronisation anhand einer gemeinsamen Datei.

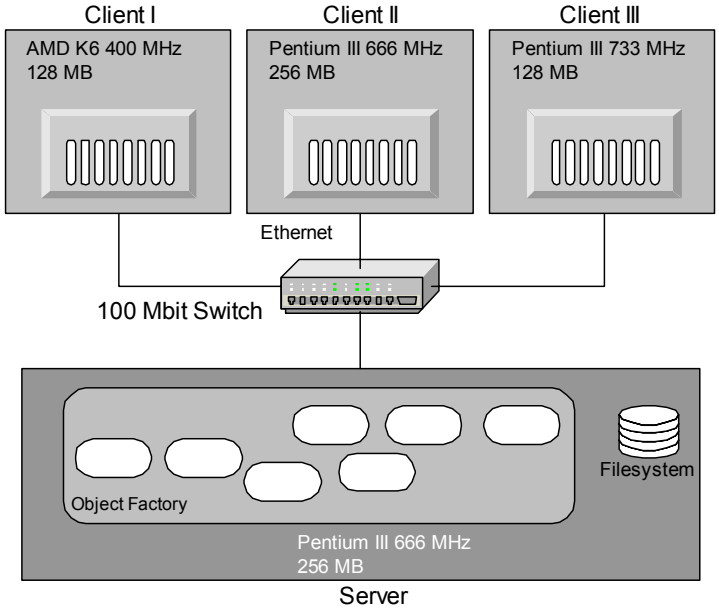


Abbildung 9-3 Testaufbau für Performance-Messungen

9.5.3.2 Antwortzeitverhalten

Der erste Teil der Tests beschäftigt sich mit dem Antwortzeitverhalten des Systems bei steigender Last. In den nachfolgend beschriebenen drei Szenarien kommuniziert dabei eine steigende Anzahl von Client-Threads mit einer zentralen Factory, wobei durch Variation der Wartezeiten und Zugriffzyklen unterschiedliche Lastprofile erzeugt werden. Gemessen wird jeweils die Zeit, die für das Anlegen des Objekts sowie das erneute Auslesen der Informationen benötigt wird.

Szenario I:

In einem ersten Test wird versucht, eine maximale Last durch geringe Wartezeit und hohe Parallelität der Client-Threads zu erzeugen. Ziel ist hier also kein für ein Call Center repräsentatives Verhalten. Mit einer Wartezeit von 1,5 Sekunden werden gleichzeitig auf allen Clientrechnern die angegebene Anzahl von Threads gestartet, wobei genau ein Zyklus durchlaufen wurde. Es wird dadurch also eine Last von 15 bis 425 parallelen Clients erzeugt. Das Ergebnis zeigt, dass bis zu einer Last von 100 Threads pro Client die Gesamtzeit für Anlegen und Auslesen der Daten unter einer halben Sekunde liegt. Ab 125 Threads steigt dann die Antwortzeit relativ schnell auf Spitzenwerte um zwei Sekunden an, was in erster Linie an der Performance der zentralen Factory aber auch an der zunehmenden Auslastung der Clientrechner liegt (vgl. Abbildung 9-4).

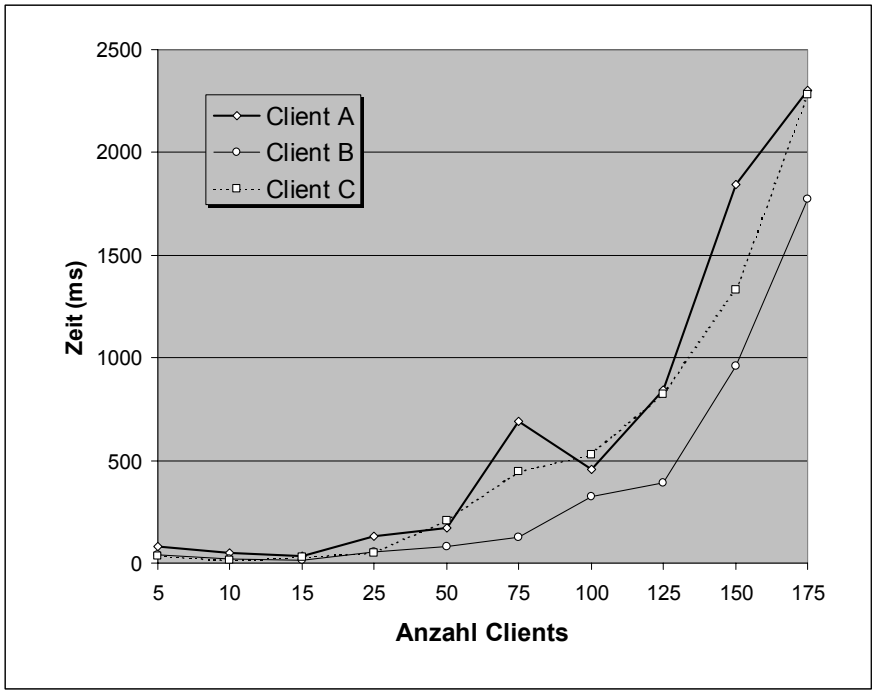


Abbildung 9-4 Antwortzeiten bei steigender Last

Die nachfolgende Auswertung vergleicht für Client B nochmals die Zeiten, die für das Anlegen bzw. Auslesen des Anrufobjekts benötigt werden. Man sieht hier, dass auch bei Anzahl von 150 Threads, also insgesamt 450 Clients, das Lesen der Informationen noch innerhalb 600 Millisekunden liegt, also durchaus noch ausreichend etwa für den Aufbau eines Begrüßungsbildschirms. Die signifikant längeren Zeiten für das Lesen von Objekten bei hoher Clientanzahl kann man auf die Zeit zurückführen, die die Factory zum Auffinden der korrekten Objektreferenz benötigt, da hier kein optimierter Suchalgorithmus implementiert wurde.

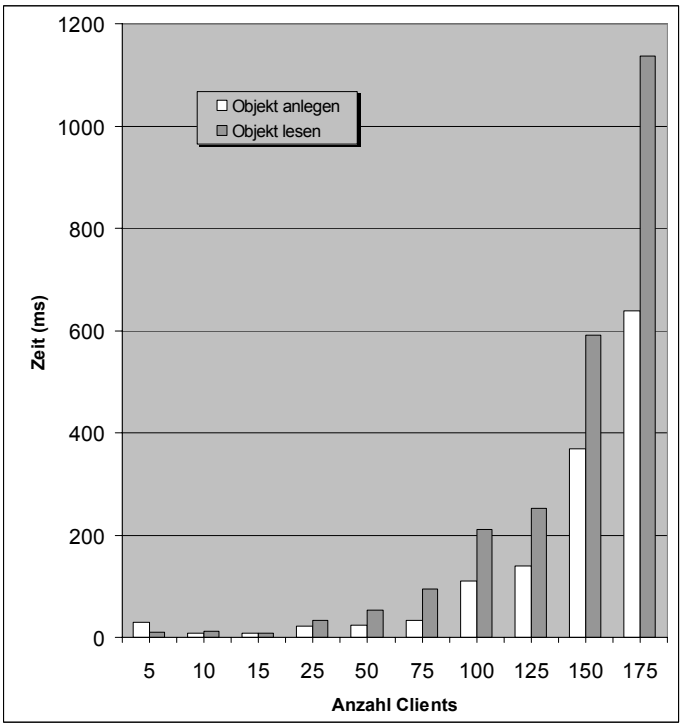


Abbildung 9-5 Vergleich Lesen/Schreiben der Objektdaten

Szenario II

Eine noch höhere Last kann erzeugt werden, indem jeder Thread nacheinander mehrere, im konkreten Beispiel 5 Zyklen durchläuft, wiederum mit einer Wartezeit von 1,5 Sekunden. Diese Zugriffe werden zwar sequentiell ausgeführt, insgesamt erhöht sich jedoch damit die Last auf die Factory um ein weiteres Stück, wie man auch den Antwortzeiten in Abbildung 9-6 entnehmen kann. Bis zu einer Anzahl von 75 Threads pro Client sind die Zeiten noch vergleichbar, steigen dann aber stärker an und erreichen ein Maximum von 4 Sekunden. In diesem Bereich kann der Server nicht mehr genügend Anfragen in nebenläufigen Threads abarbeiten, so dass hier Wartezeiten entstehen.

Damit sind auch die Performancegrenzen der Prototyp-Implementierung klar erkennbar, da Antwortzeiten über einer Sekunde für den Betrieb eines Call Centers in jedem Falle zu lang sind. Da sich die hier angegebenen Zeiten auf beide Teile des Zyklus beziehen, liegt die Grenze hier bei 100 bis 125 Threads pro Clientrechner. Das System stößt also bei einer Spitzenlast von etwa 300 Anrufen, die parallel bearbeitet werden müssen, an seine Grenzen.

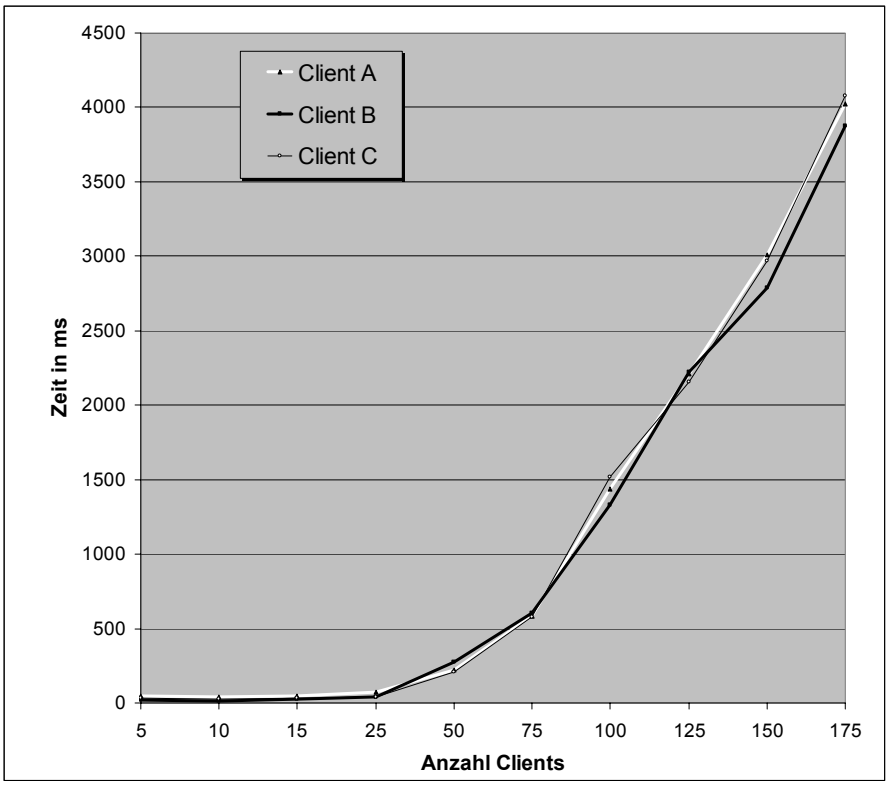


Abbildung 9-6 Antwortzeiten bei vielen Zugriffen

Szenario III

In den bisherigen Tests wurde die Last durch den möglichst gleichzeitigen Ablauf mehrerer Clients und entsprechend kurze Wartezeiten erzeugt, ein Verhalten, das in einem Call Center eher untypisch ist. Daher werden als Vergleich in diesem Szenario die Client-Threads mit einer Wartezeit von 30 Sekunden sowie einer individuellen Abweichung von bis zu 10 Sekunden gestartet. Die individuelle Abweichung wurde auf Basis der Funktion *rand* implementiert, die einen Zufallswert zwischen 0 und 1 liefert, mit dem die Abweichung multipliziert wurde und abwechselnd addiert bzw. subtrahiert wurde. Die Implementierung dieser Funktion liefert für die hier notwendigen Zwecke eine ausreichend gute Gleichverteilung.

Auch dies sind für einen realen Anruf noch relativ kurze Werte, die Last auf die Factory sinkt jedoch hier bereits durch die geringere Anzahl nebenläufiger Anfragen äußerst stark. Selbst bei 175 Threads liegen die

Antwortzeiten unter 100 Millisekunden. Bei einer gleichmäßigeren Verteilung der Anrufe ist die Implementierung also auch einer höheren Anzahl von Clients durchaus gewachsen.

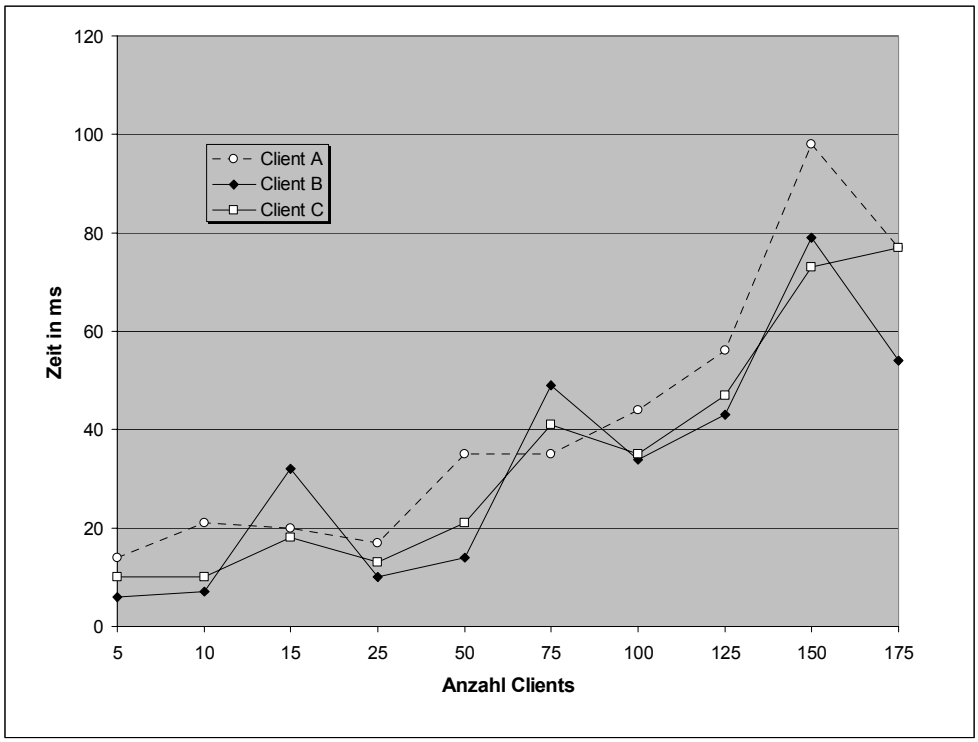


Abbildung 9-7 Antwortzeiten bei realistischen Wartezeiten

9.5.3.3 Ressourcenverbrauch

Neben den Antwortzeiten der Clients ist der Verbrauch an Ressourcen der Serverkomponenten ein wichtiges Indiz für die Skalierbarkeit verteilter Systeme. Im konkreten Falle ist also der benötigte Arbeitsspeicher der Factory in Abhängigkeit der Anzahl der Clients interessant. Dabei wird zum einen der Einfluss untersucht, den die Größe der Anrufdaten auf den gesamten Speicherverbrauch hat sowie in einem weiteren Test das Verhalten der Implementierung über einen längeren Zeitraum hinweg beobachtet.

Szenario I

Ziel dieses Tests ist es, eine steigende Anzahl von Objekten zu erzeugen, die die Factory verwalten muss. Dabei werden analog den bisherigen Tests die drei Clients in mehreren Durchläufen mit jeweils einer steigenden Anzahl von Threads gestartet. Jeder Thread durchläuft dabei 3 Zyklen, so dass möglichst viele Anrufobjekte gleichzeitig angelegt werden. Nach jedem Durchlauf wird die Report-Funktion der Factory aufgerufen, so dass der zeitliche Verlauf des Speicherverbrauchs sowie der Objektanzahl deutlich wird. Diese steigt, wie Abbildung 9-8 zeigt, auf Werte von über 500 (bei 3 Clients und jeweils 175 Threads). Der Speicherbedarf der Factory steigt insgesamt von 2000 Kilobyte auf etwa 13000 Kilobyte an. Dass der Anstieg dabei in Stufen erfolgt liegt an der Speicherverwaltung der Java Virtual Machine, die in Stufen neuen Speicher reserviert, sobald der intern belegte Speicher einen gewissen Schwellwert erreicht.

Dass der Speicherbedarf kontinuierlich ansteigt obwohl die Anzahl der Objekte zwischen den einzelnen Testläufen auf null zurück geht, hat zwei Gründe. Zum einen läuft die automatische Bereinigung des Speichers (*garbage collection)* bei Java asynchron innerhalb eines eigenen Threads. Durch nicht mehr benötigte Objekte frei gewordener Speicher wird also erst beim nächsten Durchlauf berücksichtigt. Außerdem läuft die Implementierung ja innerhalb eines CORBA-Laufzeitsystems, das selbst eine eigene Verwaltung seiner Ressourcen vornimmt und hier zum Beispiel Netzwerkverbindungen und Threads in Pools verwaltet und daher für eine eventuelle Wiederverwendung eine gewisse Zeit vorhält.

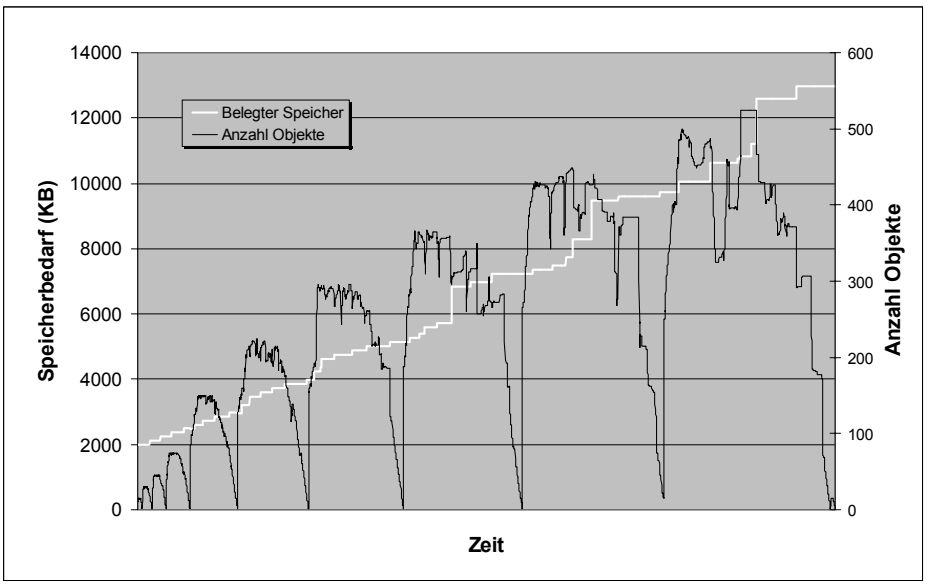


Abbildung 9-8 Speicherbedarf der Factory

Eine Wiederholung des Tests mit Nutzdaten von 256, 512 sowie 1024 Byte zeigt, dass der Umfang dieser Informationen nur einen geringen Einfluss auf den gesamten Speicherbedarf hat. Zwar steigt dieser im ersten Falle langsamer an, erreicht aber letztendlich einen ebenso hohen Maximalwert (vgl. Abbildung 9-9). Auch hier überdecken das CORBA-Laufzeitsystem sowie die Speicherverwaltung von Java die Auswirkungen, die durch Änderung der Nutzdaten hervorgerufen werden. Eine weitere Steigerung des Umfangs dieser Informationen würde hier sicher zu deutlich messbaren Effekten führen. Der Wert von 1024 Byte ist allerdings bereits sehr großzügig bemessen, wenn es um die Informationen geht, die in der Regel mit einem Anruf verknüpft werden. Tests mit noch höheren Werten wären daher eher unrealistisch.

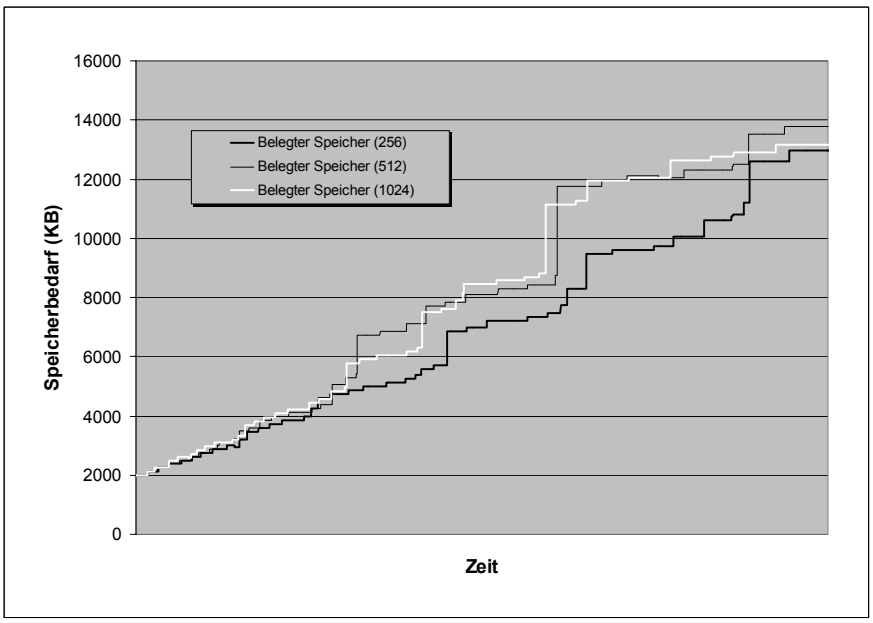


Abbildung 9-9 Vergleich des Speicherbedarfs

Szenario II

Auch wenn der Speicherbedarf der Factory selbst bei hoher Last im Grunde unkritisch ist, so steigt er während der Tests doch kontinuierlich an. Es ist daher abschließend noch zu untersuchen, wie sich der Prototyp in dieser Beziehung über einen längeren Zeitraum hinweg verhält, insbesondere wenn die Last

wieder zurückgenommen wird. Für diese Betrachtung wurde die Factory leicht modifiziert, so dass sie selbstständig alle 5 Sekunden ihre eigene report-Funktion aufruft, um so einen kontinuierlichen Verlauf der Speichernutzung zu erhalten. Weiterhin wird bei jedem Aufruf der garbage collector von Java angestoßen.

In einer ersten Phase wird maximale Last erzeugt, indem die einzelnen Clients zunehmend immer mehr Threads mit sehr kurzen Wartezeiten erzeugen. Daraufhin wird in einer zweiten Phase die Last reduziert und die Clients mit langen Wartezeiten und geringer Anzahl von Threads gestartet. Abschließend wird in einer letzten Phase ein Client nach dem anderen beendet.

Das Diagramm (vgl. Abbildung 9-10) zeigt zu Anfang das bereits bekannte Verhalten des Speicherbedarfs, der bei hoher Anzahl von Objekten auf über 5000 Kilobyte ansteigt. Zusätzlich ist jedoch noch der für die Anwendung freie Speicher dargestellt, der erwartungsgemäß stark ansteigt, sobald die Last zurückgenommen wird (bei 282 Sekunden). Daraufhin gibt das System einen Teil des belegten Speichers wieder frei und stabilisiert sich bei etwa 3500 Kilobyte. Verschiedene Durchläufe mit Objektanzahlen zwischen 10 und 100 ändern an dieser Situation im Grunde nichts. Erst in der letzten Phase, wenn durch das Beenden der Clients weitere CORBA-Ressourcen frei werden, sinkt der Speicherverbrauch nochmals um ca. 10 %.

Die Implementierung gibt also Ressourcen, die während einer Spitzenlast reserviert wurden, nach und nach wieder frei und verhält sich so erwartungsgemäß unkritisch. Im normalen Betrieb mit durchschnittlicher Last kann man hier von relativ konstantem Verbrauch ausgehen, da die Ressourcenverwaltung des CORBA-Systems die Anforderungen der Implementierung überdeckt.

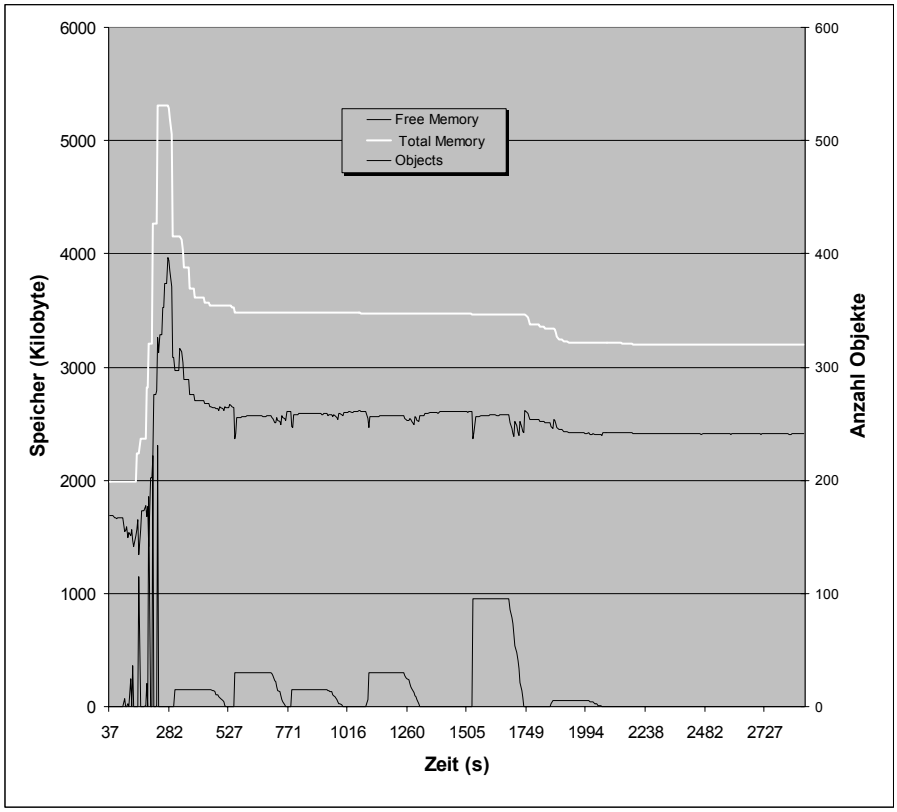


Abbildung 9-10 Langzeitbetrachtung der Factory

9.5.4 Zusammenfassung

Die Ergebnisse der verschiedenen Lastszenarien zeigen, dass die Implementierung einer zentralen, auf CORBA basierenden Factory den Anforderungen eines Call Centers gewachsen ist. Es können Spitzenlasten von über 300 parallelen Clients mit Antwortzeiten unter einer halben Sekunde abgearbeitet werden.

Bei gleichmäßiger Verteilung und realistischen Gesprächszeiten bleiben diese auch bei hohen Zahlen noch weit darunter. Auch der Speicherbedarf der Serverkomponente liegt ebenfalls in einem normalen Bereich.

Anzumerken ist dabei noch, dass die Testumgebung sicherlich optimal und wenig belastet war. Bei einem größeren Netzwerk mit insgesamt mehr Betrieb muss man an dieser Stelle mit gewissen Einbußen der Leistungsfähigkeit rechnen. Allerdings enthält die prototypische Implementierung auch keinerlei Optimierungen, etwa was die Zuordnung von Objektreferenzen auf Anrufteilnehmer angeht. Insgesamt lassen die Ergebnisse also den Schluss zu, dass mit einer derartigen Implementierung Call Center bis zu einer Größenordnung von 500 Clients bedient werden können. Für größere Installationen bzw. überdurchschnittlich hohe Last wäre der Einsatz von mehreren Instanzen zu empfehlen, die wie bereits angesprochen sich untereinander bzgl. der verwalteten Objektreferenzen bei Bedarf koordinieren.

Das neue Konzept, die Verknüpfung von Informationen zu einem Anruf außerhalb des CTI-Systems in einen zentralen CORBA-Server bzw. eigenständige CORBA-Objekte zu verlagern, erweist sich also auch in dieser Hinsicht als tragfähig.

9.6 Diskussion verwandter Arbeiten

Wie in Abschnitt 1.3 bereits dargestellt wurde, berührt die Arbeit mehrere Bereiche aus dem Umfeld der Telekommunikation, Call Center und CTI sowie der verteilten Systeme. Die nachfolgenden, wissenschaftlichen Arbeiten beinhalten interessante Aspekte zu einigen dieser Themen und werden in diesem Abschnitt kurz vor dem Hintergrund der Ergebnisse dieser Arbeit diskutiert.

Im Umfeld von Petrinetzen sind [Anisimov96] sowie [Anisimov99] erwähnenswert, da diese analog zur vorliegenden Arbeit versuchen, die Abläufe und Szenarien in Call Centern formal und mit Hilfe von Petrinetzen zu modellieren. Allerdings beschäftigt sich insbesondere [Anisimov99] intensiver mit dem Workflow, den ein Anruf innerhalb eines Call Centers auslöst und dessen Aktionen über vordefinierte Scripts spezifiziert werden. Es geht hier also in erster Linie um die Anrufbehandlung durch einen oder mehrere Mitarbeiter. Es wird dazu ein erweitertes Modell sog. *script-nets* entworfen, die das grundlegende Modell um Konzepte wie etwa *Macroplaces* erweitern. Im Vordergrund steht hier die Modellierung der Funktionalität einer bestimmten CTI-Anwendung während das vorliegende Framework versucht, sich auf die grundlegenden Telefonfunktionen zu konzentrieren und den dadurch ausgelösten Workflow der konkreten CTI-Applikation zu überlassen.

Nicht gezielt für ein Call Center sondern eher allgemein beschäftigt sich [Alst98] mit der Modellierung von Workflows mit Hilfe von Petrinetzen. Neben einigen, für die Modellierung von Workflows interessanten Ergänzungen wird hier auch die Analyse der Netze zur Überprüfung konkreter Modelle angesprochen und gezeigt, dass wichtige Kriterien einer korrekten Definition auch formal überprüft werden können. Insgesamt decken sich die Ergebnisse mit denen der vorliegenden Arbeit dahingehend, dass Petrinetze vor allem wegen der klar definierten Semantik und der formalen Analysemöglichkeiten als geeignetes Modellierungskonzept eingeschätzt werden. Jedoch ist ein Ergebnis dieser Arbeit, dass auch die einfache und Version von B/E-Netzen ohne Erweiterungen zur Modellierung ausreicht. Eine weitere Anwendung von Petrinetzen zur Modellierung von Workflows gibt [Woetzel93].

Die Anwendung von Petrinetzen für kollaborative Systeme untersucht [Ferraro97] im Zusammenhang mit Teamarbeit und den dazugehörenden Protokollen etwa bei einer Besprechung. Auch hier wird der Einsatz dieses Modells positiv eingeschätzt. Insgesamt belegen diese Arbeiten, dass die Verwendung von Petrinetzen im Umfeld von CTI und Call Centern viel Potential hat und zeigen ergänzend zu den Ergebnissen dieser Arbeit verwandte Einsatzgebiete auf.

Eine etwas andere und eher theoretische Sicht auf die Organisation und Abläufe eines Call Centers haben [Jongbloed01] und [Koole00]. Diese Arbeiten betrachten Call Center aus stochastischer Sicht und modellieren es unter anderem als Warteschlangensystem. Mittels hier verfügbarer Formeln wie etwa der Erlang-Formel werden dann Voraussagen zum Beispiel über die Auslastung des Call Centers getroffen. Diese Arbeiten grenzen sich demnach klar von der ingenieurwissenschaftlichen Motivation dieser Arbeit ab. Sie greifen auch eher den Bereich des Managements und der Verwaltung von Call Centern auf als die hier

konkret geforderten Softwarelösungen im Bereich CTI. Noch allgemeiner werden mathematische Modelle für diesen Bereich in [Koole02] dargestellt. Im Rahmen eines Ausblicks wäre in diesem Zusammenhang der Vergleich solch theoretischer Voraussagen mit Ergebnissen konkreter Performancetests auf Basis des konzipierten Frameworks.

Weitere verwandte Arbeiten, die abschließend noch erwähnt werden sollten, beschäftigen sich mit neuen Telekommunikationstechnologien. Hier ist zum Beispiel [Jain00] zu erwähnen, in der eine Java API JCC/JCAT zur Anrufsteuerung insbesondere vor dem Hintergrund paketvermittelter Netze (IP/ATM) vorgeschlagen wird. Da die Schnittstelle vom Aufbau her eng an JTAPI angelehnt ist, würde sich eine Nutzung in dem hier vorgeschlagenen Framework ohne großen Aufwand anbieten und würde das Einsatzgebiet auf Call Center mit neuartiger Kommunikationstechnologie erweitern. Gerade im Hinblick auf die Integration von Webanwendungen bietet der Einsatz von IP-Netzen hier viel Potential. Mit diesem Thema beschäftigt sich [Kuhlins01]. In dieser Arbeit wird die Kombination von Internet- und Telefonkommunikation untersucht und eine entsprechende Beispielimplementierung vorgestellt. Der Ansatz eines allgemeinen Frameworks für Call Center lässt sich auch auf das dort dargestellte Szenario sehr gut abbilden und durch das Konzept eines unabhängigen Anrufobjekts könnte die Integration beider Medien entscheidend vereinfacht werden.

9.7 Zusammenfassung

In diesem Abschnitt wurde das ausgearbeitete Framework hinsichtlich Konzept, Einsatz durch einen Entwickler sowie Verhalten und Last untersucht und bewertet. Ergebnis dabei ist, dass die Konzeption des auf JTAPI und CORBA aufbauenden Frameworks die zu Anfang gestellten Ziele eines allgemeinen Basisystems für CTI-Anwendungen umsetzen kann. Die Variabilitätsanalyse hat dies grundsätzlich bestätigt, wenn auch Einschränkungen festgestellt werden müssen, die durch die fachliche Modellierung ausgeglichen werden. Insgesamt setzt das Framework die Forderung nach einer Unabhängigkeit von konkreten CTI-Lösungen, einer komfortablen Schnittstelle für den Anwender, mächtigeren Zuständen sowie einer flexible Verknüpfung von Informationen mit einem Anruf um.

Die Vorteile eines derartigen Frameworks für den Entwickler liegen neben dem einfacheren Zugang auch zu komplexeren Telefonfunktionen (Weiterleitung, Konferenz) und dem Zugriff auf einen mächtigeren und auf die Prozesse eines Call Centers zugeschnittenen Zustandsautomaten in der flexiblen Definition von Informationen, die als CORBA-Objekte mit einem Anruf verknüpft werden. Hierdurch ergeben sich eine Reihe von Möglichkeiten, die die Integration der CTI-Anwendung in das Gesamtsystem eines Unternehmens erleichtern bzw. erst ermöglichen.

Eine prototypische Implementierung hat schließlich die Realisierbarkeit der ausgearbeiteten Ansätze gezeigt und insbesondere das Zusammenspiel eines CTI-Systems wie JTAPI mit einer CORBA-Middleware verifiziert. Die Untersuchung der zentralen Objektverwaltung dieses Prototypen hinsichtlich für Call Center typischen Lastszenarien hat anschließend ergeben, dass dieser Ansatz auf Basis einer aktuellen CORBA-Implementierung auch für größere Installationen durchaus geeignet und einsetzbar ist.

10 ZUSAMMENFASSUNG UND AUSBLICK

10.1 Inhalte der Arbeit

Die vorliegende Arbeit versucht, die Grundidee von CTI, nämlich die Integration von Computer und Telefonie, auf einem höheren Abstraktionslevel vor dem Hintergrund moderner Call Center und ihrer Anwendungen umzusetzen und auf ein verteiltes System abzubilden. Schwerpunkt ist dabei die Entwicklung eines Frameworks, das als Basis für fortschrittliche CTI-Anwendungen dienen soll und dem Entwickler funktional höherwertige Dienste für die Telefonsteuerung, die Verwaltung von Zuständen sowie die Verknüpfung von Informationen mit einem Anruf zur Verfügung stellt.

Neben einer Untersuchung der CTI-Technologie und Geschäftsprozesse in einem Call Center sowie einem Überblick über Konzepte und Basistechnologien, die für Design und Implementierung des Frameworks in Betracht kommen, wird die für diesen Ansatz notwendige Funktionalität spezifiziert, wobei für die Modellierung der verteilten Abläufe Petrinetze eingesetzt werden. Auf dieser Basis wird ein objektorientiertes Framework konzipiert, das im Wesentlichen aus drei eigenständigen CORBA-Servern besteht, die sowohl lokal auf dem Client als auch zentral über das Netzwerk angesprochen werden. Diese bieten dem Anwender über wohldefinierte Schnittstellen die Dienste bzgl. Telefonsteuerung, Zustandsautomat sowie Anrufobjekt. Die Anbindung an das Telefonsystem erfolgt dabei über die standardisierte Java-Schnittstelle JTAPI.

Im Anschluss an die Konzeption wird die praktische Umsetzung anhand einer prototypischen Implementierung überprüft. Dieser Prototyp basiert neben der CORBA-Implementierung VisiBroker der Firma Borland auf einem JTAPI-Simulationssystem der Firma IBM. Er ist abschließend Gegenstand einer allgemeinen Bewertung, die sowohl die konzeptionellen Aspekte als auch den Mehrwert für den Benutzer sowie das Verhalten unter Last berücksichtigt.

10.2 Ergebnisse

Wesentliches Ergebnis der Arbeit ist das entwickelte Basisframework, wodurch gezeigt werden konnte, dass innovative CTI-Anwendungen für Call Center auf Basis einer modernen, auf einer Middleware wie CORBA basierenden Architektur realisiert werden und somit wesentliche Aspekte verteilter Anwendungsentwicklung berücksichtigt werden können. Die Vorteile des darin enthaltenen Konzepts liegen zum einen in der Eigenständigkeit der einzelnen Komponenten, die so von unterschiedlichen CTI-Anwendungen genutzt werden können. Dem Anwender bietet das System damit neben einer komfortablen Telefonsteuerung sowie einem Zustandsautomaten die Verknüpfung eines Anrufs mit praktisch frei definierbaren und eigenständigen CORBA-Objekten und wendet so auch hier das Prinzip von CTI, das bisher in der Regel nur auf technischer Basis gelöst wird, auf das Gebiet der verteilten Anwendungen und Systeme an. Diese Verknüpfung von Informationen mit einem Anruf wird dabei unabhängig von konkreten CTI-Basissystemen in einer zentralen Factory gelöst. Darauf aufbauend können die Anrufobjekte als CORBA-Server selbst weiterführende Aufgaben wie etwa die Synchronisation des Zugriffs übernehmen. Darüber hinaus bietet der durchgängige Einsatz von CORBA eine Reihe von Möglichkeiten, dieses Framework in eine bestehende IT-Landschaft einzubetten und an bestehende Legacy-Systeme anzuschließen. Die für das Konzept notwendigen Basistechnologien wurden auf ihre Tauglichkeit untersucht und damit die Unabhängigkeit des Ansatzes dargelegt. Eine abschließende Variabilitätsanalyse hat gezeigt, dass das Framework für die dargestellte Anwendungsdomäne ausreichend allgemeingültig ist.

Ein weiteres Ergebnis stellen die Erkenntnisse dar, die aus der Anwendung verfügbarer Methoden auf diesem Spezialgebiet gewonnen werden konnten. Hier hat sich während der Modellierung neben UML vor allem auch der Einsatz von Petrinetzen als sehr geeignet für diesen Bereich erwiesen, nicht zuletzt durch die formalen Analysemöglichkeiten, die sehr effizient zu ausreichend mächtigen lokalen Zustandsautomaten führen.

Die Realisierung eines Prototypen, der einen Großteil der spezifizierten Funktionalität umsetzt, hat ergeben, dass sich die Konzeption auch hinsichtlich einer konkreten Umsetzung als tragfähig erweist. Der Aufwand für die Implementierung hielt sich dabei durchaus in Grenzen. Diverse Lastszenarien, die die Anforderungen von Call Centern mit bis zu 500 Arbeitsplätzen simulieren, haben schließlich gezeigt, dass die zentrale Factory und die dort verwalteten Anrufobjekte auch für größere Installationen geeignet sind. Eine ausreichend leistungsfähige JTAPI-Implementierung vorausgesetzt kann man also davon ausgehen, dass der in dieser Arbeit beschriebene Ansatz auch in der Praxis konkreter Call Center seinen Einsatz finden kann.

Abschließend wird dadurch die eingangs aufgestellte These unterstützt, dass ein allgemeineres Modell für CTI-Anwendungen aus der Sicht der verteilten Anwendungsentwicklung möglich und auch notwendig ist. Die entwickelten Vorgaben bzgl. Design und Architektur lassen sich dabei sehr allgemein auf alle Systeme anwenden, die sich mit der Interaktion zwischen IT- und Telekommunikationssystem befassen.

10.3 Ausblick

Nachdem anhand einer prototypischen Implementierung grundsätzlich die Machbarkeit eines konkreten Einsatzes des Frameworks gezeigt werden konnte, stehen hier weitere Untersuchungen bezüglich der Verwendung in der Praxis an. Dazu gehört unter anderem die Verwendung einer kommerziellen CTI-Middleware mit entsprechender JTAPI-Implementierung, die über eine Anbindung an ein Switch-System eines Call Centers verfügt.

Anwendungen innerhalb eines Call Centers und in diesem Zusammenhang auch die CTI-Technologie werden derzeit zunehmend in den allgemeineren Ansatz des *Customer Relationship Management* (CRM) integriert. Da die Ansätze dieser Arbeit unter anderem auch eine Integration in bestehende Systeme eines Unternehmens berücksichtigen, wäre die Einbettung des Frameworks in eine CRM-Umgebung ein interessanter Ansatz für weitere Forschungsarbeiten. Aber auch die Integration des Frameworks in die allgemeine IT-Landschaft des Unternehmens, etwa die Anbindung an bestehende ERP-Systeme oder vorhandene Datenbanken. Auch ein grundlegender Vergleich mit den Ansätzen von Workflowmanagementsystemen sowie Integrationsmöglichkeiten stellen ein interessantes Themengebiet dar.

Im Bereich der Modellierung verteilter Prozesse wären weitere Erfahrungen bzgl. des Einsatzes von Petrinetzen interessant. Auch wenn sich das Modell der B/E-Netze als ausreichend mächtig erwiesen hat, ist die Bewertung erweiterter, etwa zeitbehafteter Netze eine interessante Fragestellung.

Weiterer interessanter Aspekt ist die Kombination des Frameworks mit bestehenden Internet-Technologien mit dem Ziel, die Dienstleistungen eines Call Centers mit den über das WWW angebotenen Funktionen zu verknüpfen. Ansatz dabei wäre, Informationen über den Status eines Benutzers auf einer WWW-Seite mit einem Anruf zu verknüpfen und so dem Mitarbeiter im Call Center zur Verfügung stellen zu können. Damit könnte unter anderem die Initiierung eines Rückrufs über eine WWW-Seite realisiert werden, indem der Web-Server des Call Centers die Statusinformation über die Factory in einem Objekt speichert und über die Anrufsteuerung den Rückruf einleitet. Der Begrüßungsbildschirm des Mitarbeiters könnte dann alle Aktionen des Benutzers auf dem Web-Auftritt anzeigen. Interessante Fragen tauchen hier auf, wenn man das Framework vor dem Hintergrund aktueller Entwicklungen und Middleware-Ansätze im Internet betrachtet, wie sie derzeit im Rahmen von .NET von Microsoft oder ONE von Sun Microsystems vorgeschlagen werden. Inwieweit hier die Nutzung bzw. das Anbieten sog. Web-Services einen Mehrwert bringen kann, wäre einer von vielen zu untersuchenden Aspekten.

Als letzter Aspekt eines Ausblicks ist der Zusammenhang mit neuen Entwicklungen in der Telekommunikation zu erwähnen. So kommt die Technologie der IP-Telefonie in letzter Zeit gerade in Call Centern wieder ins Gespräch, da durch das IP-Protokoll eine sehr flexible Vernetzung von IT- und TK-Systemen möglich ist. Auch die mobile Kommunikation etwa über WAP bietet hier viel Potential. Dazu müsste untersucht werden, inwieweit über dieses Protokoll parallel zu einem Anruf Informationen zwischen dem Anrufer und dem System des Call Centers ausgetauscht werden können und welche Synergieeffekte dabei zu erwarten sind.

11 GLOSSAR

100Base-T	*100 Megabit Baseband Trunk*	LAN mit einer Übertragungsrate von 100 megabit pro Sekunde (Fast Ethernet)

- A -

„at most once" Semantik		Eigenschaft eines RPC-Systems, die anzeigt, dass eine Operation höchstens einmal aufgerufen wird.
ACD	*Automatic Call Distribution*	Komponenten einer Telefonanlage eines Call Centers, die die automatische Verteilung der Anrufe auf Mitarbeiter überehmen.
ACID	*Atomicity* *Consistency* *Isolation* *Durability*	Eigenschaften von Transaktionen
ADO	*ActiveX Data Objects*	
Aggregation		Beziehung zwischen Objekten, bei der eines ein Teil des anderen darstellt.
ANI	*Automatic Number Identification*	
ASCII	*American Standard Code for Information Interchange*	
Assoziation		Beziehung zwischen Objekten.
AWT	Advanced Windows Toolkit	GUI-Bibliothek von Java.

- B -

B/E-Netz	*Bedingungs/Ereignis-Netz*	Variante der Petrinetze
Begrüßungs-bildschirm		Wichtiger Bestandteil von Anwendungen im Call Center. Der Begrüßungsbildschirm enthält alle wichtigen Informationen des Anrufers, die der Mitarbeiter für die Begrüßung verwendet. Diese Informationen sollten bereits verfügbar sein, ehe das Gespräch entgegengenommen wird.
BOA	*Basic Object Adapter*	

Business Object		Ein Objekt, das eine bestimmte Funktionalität der Geschäftslogik innerhalb des Anwendungsbereichs darstellt.
	- C -	
Class library		Sammlung von Objektklassen, die in unterschiedlichen Anwendungen wiederverwendet werden können.
CLID	*Calling Line Identification*	
Composition		Variante der Aggregation, bei der das übergeordnete Objekt die Verantwortung für seine Teilobjekte übernimmt.
CORBA	*Common Object Request Broker*	
CRM	*Customer Relationship Management*	
CSCW	*Computer Supported Co-operative Work*	
CTI	*Computer Telephony Integration*	
	- D -	
Design pattern		Muster, das systematisch ein wiederkehrendes Designproblem definiert und beschreibt.
DSOM	*Distributed System Object Model*	Erweiterung von SOM bzgl. netzwerkweiter Kommunikation. Stellt damit eine CORBA-Implementierung dar.
DTD	*Document Type Definition*	
	- E -	
„exactly once“ Semantik		Eigenschaft eines RPC-Systems, die anzeigt, dass eine Operation genau einmal aufgerufen wird.
ECMA	*European Computer Manufacturers Association*	
ECTF	*Enterprise Computer Telephony Forum*	
EJB	*Enterprise Java Beans*	

Encapsulation		Paradigma der objektorientierten Entwicklung, dass Daten eines Objekts nicht direkt sondern nur über entsprechende Methoden des Objekts verändert werden.
ERP	*Enterprise Resource Planning*	
		- F -
FAQ	*Frequently asked question*	
Firewall		Komponente eines lokalen Netzwerkes, die den Datenaustausch mit der Außenwelt kontrolliert und damit unberechtigten Zugang vermeidet.
First party		Eigenschaft einer CTI-Anwendung, die anzeigt, dass die Applikation hier direkt mit der Telefonhardware interagiert.
Framework		Anwendungsgerüst mit dem Ziel, die Wiederverwendbarkeit in der objektorientierten Entwicklung zu steigern.
		- G -
garbage collection		Teil einer Speicherverwaltung, der nicht mehr benötigte Objekte aus dem Heap-Speicher entfernt.
		- H -
http tunneling		Technik, bei der Nachrichten zwischen Anwendungen über das http-Protokoll versendet werden, um die Blockierung durch Firewalls zu verhindern.
		- I -
		- J -
Inversion of control		Eigenschaft eines Frameworks, dass individueller Programmcode integriert und aus vordefinierten Abläufen heraus aufgerufen wird.
IT	*Information Technology*	
IVR	*Interactive Voice Response*	
JAVA		Von SUN entwickelte Programmiersprache.
JDK	*Java Development Kit*	

JFC	*Java Foundation Classes*	
JNDI	*Java Naming and Directory Interface*	
JTAPI	*Java Telephony API*	
	- K -	
	- L -	
Legacy system		Bereits bestehendes IT-System, das bei der Entwicklung neuer Anwendungen berücksichtigt werden muss.
Load balancing		
	- M -	
	- N -	
Network Computer		Ansatz eines PC ohne Festplatte, der nur verbunden mit dem Internet Zugriff auf Daten und Anwendungen erhält.
	- O -	
off-hook dialing		Übliche Version, einen Anruf einzuleiten. Dabei wird erst der Hörer des Telefons abgehoben und dann die Nummer gewählt.
OLE	*Object Linking and Embedding*	
OMG	*Object Management Group*	
on-hook dialing		Alternative Version, einen Anruf einzuleiten, wie es beispielsweise bei Mobiltelefonen üblich ist. Dabei wird die Nummer bereits gewählt, ehe abgehoben wird.
ORB	*Object Request Broker*	
	- P -	
P/T-Netz	*Prädikat-/Transitionsnetz*	Variante der Petrinetze.
PABX	*Private Access Branch Exchange*	Siehe PBX.

PBX	*Private Branch Exchange*	Englische Bezeichnung für eine Telefonanlage, die das Basissystem eines jeden Call Centers darstellt.
Petrinetz		Modell zur Beschreibung von parallelen und nebenläufigen Prozessen.
PIN	*Personal Identification Number*	
POA	*Portable Object Adapter*	
Polymorphismus		Paradigma der objektorientierten Programmierung
PSTN	*Public Switched Telephone Network*	

- Q -

- R -

Round robin		Strategie zur Verteilung von Anfragen, wobei linear immer der nächste Server ausgewählt wird.
RPC	*Remote Procedure Call*	
RUP	*Rational Unified Process*	

- S -

S/T-Netz	*Stellen/Transitions-Netz*	Variante eines Petrinetzes.
Screen pop		Englische Bezeichnung für Begrüßungsbildschirm.
Skalierbarkeit		Eigenschaft eines IT-Systems, sowohl in größeren als auch kleineren Installationen operieren zu können.
SOAP	*simple object access prototcol*	
SOM	*System Object Model*	Von IBM entwickelte Architektur, die eine gemeinsame Nutzung von Binärcode erlaubt.
SSL	*Secure Socket Layer*	
Stelle		Element eines Petrinetzes.
Swing		Aktuelle Version der GUI-Library von Java.
Switch		Englische Bezeichnung für die Telefonanlage eines Call Centers.

- T -

TAPI	*Telephone Application Programming Interface*	
Third party		Eigenschaft einer CTI-Anwendung, die anzeigt, dass die Applikation nicht direkt sondern über einen Server mit der Telefonhardware interagiert.
TSAPI	*Telephone Server Application Programming Interface*	
Transition		Teil eines Petrinetzes.
Touch Tone Dialing		Akustisches Wählverfahren

- U -

UML	*Unified Modeling Language*	
Use Case		Anwendungsfall, der eine bestimmte Funktion eines Softwaresystems beschreibt.

- V -

VDU	*Voice Data Unit*	Im Softwaresystem VESP Bezeichnung für Daten, die mit einem Anruf verknüpft werden.
Vererbung		Paradigma der objektorientierten Programmierung, bei dem eine Klasse Zugriff auf Methoden und Attribute einer übergeordneten Klasse erhält.
Versit		Industriekonsortium bestehend aus Apple, AT&T, IBM und Siemens.
VRU	*Voice Response Unit*	Siehe IVR.

- W -

WORA	*Write Once Run Anywhere*	Konzept von Java, dass ein Programm ohne Veränderung innerhalb eines virtuellen Laufzeitsystems auf verschiedenen Plattformen laufen kann.

- X -

XML	*Extensible Markup Language*	

XSL	*Extensible Stylesheet Language*
XSLT	*Extensible Stylesheet Language Transformations*

- Y -

- Z -

12 LITERATUR

[Albahari00] Ben Albahari. *A Comparative Overview of C#*. genamics.com/developer/csharp-comparative.html, August 2000.

[Alst98] W. M. P. van der Aalst. *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems and Computers, 8 (1), pp. 21-66, 1998.

[Anisimov96] N.A. Anisimov, K. Kishinski, A. Miloslavski, P.A. Postupalski. *Macroplaces in High Level Petri Nets: Application for Design Inbound Call Center*. In: Proceedings of the Int. Conference on Information System Analysis and Synthesis (ISAS'96), Orlando, FL, USA, pp. 153-160. July 1996.

[Anisimov99] Nikolay Anisimov, Konstantin Kishinski and Alec Miloslavski. *Formal Model, Language and Tools for Design Agent's Scenarios in Call Center Systems*. In: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences, 1999.

[Ari90] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice Hall International, 1990.

[Balzert96a] H. Balzert. *Lehrbuch der Software Technik, Software-Entwicklung*. Spektrum Akademischer Verlag, 1996.

[Balzert96b] Heide Balzert. *Objektorientierte Analyse*. Spektrum Akademischer Verlag, 1996.

[Bauer99] N. Bauer. *Verteilte Anwendungen im Call Center*. Computer Telephony, Heft 8, pp. 36-41, 1999.

[Bauer00a] N. Bauer. *Verteilte Anwendungen im Call Center (Teil 2)*. Computer Telephony, Heft 1-2, pp. 44-47, 2000.

[Bauer00b] N. Bauer, J. Hauptmann. *GUI Entwicklung im Web*. ItFokus, Heft 4, pp. 42-49, 2000.

[Bauer00c] N. Bauer, J. Hauptmann, P. Mandl, T. Weise. *Schaltzentrale*. iX Magazin für Professionelle Informationstechnik, Heft 1, pp. 116-126, 2000.

[Bayer97] M. Bayer. *CTI Solutions and Systems*. McGraw-Hill, 1997.

[Bern96] P. A. Bernstein. *Middleware: A Model for Distributed System Services*. Communications of the ACM, 39, no. 2, pp. 86-98, 1996.

[Bern93] A. J. Bernstein and P. M. Lewis. *Concurrency in Programming and Database Systems*. Jones and Bartlett Publishers Inc., 2nd Edition, 1993.

[Bern97] P. A. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers Inc., 1997.

[Bittner00] Susanne Bittner, Marc Schietinger, Jochen Schroth, Claudia Weinkopf. *Call Center – Entwicklungsstand und Perspektiven - Eine Literaturanalyse*. Projektbericht des Instituts Arbeit und Technik, Gelsenkirchen, 2000.

[Booch99] Grady Booch. *UML in Action*. Communications of the ACM, Vol. 42, No 10, October 1999.

[Borghoff98] Uwe M. Borghoff, Johann Schlichter. *Rechnergestützte Gruppenarbeit – Eine Einführung in Verteilte Anwendungen*. Springer Verlag, 2nd Edition, 1998.

[Born01] Achim Born, Jürgen Diercks. *Dienen und Verdienen*. iX Magazin für Professionelle Informationstechnik, Heft 7, pp. 104-113, 2001.

[Bowers00] John Bowers, David Martin. *Machinery in the New Factories: Interaction and Technology in a Bank 's Telephone Call Centre*. In: Proceedings of CSCW'00, December 2000.

[Budd91] Timothy Budd. *An Introduction To Object Oriented Programming*. Addison Wesley, 1991.

[Burgett95] Jeff L. Burgett, Sheryl J. Adam. *Applying Object-Oriented Software Engineering Methods to the Development of Call Center Software: A Case Study*. Addendum to the Proceedings of OOPSLA'95, 72-76, October 1995.

[Cockburn00] Alistair Cockburn. *Writing Effective Use Cases*. Addison Wesley 2000.

[Coul94] G. Coulouris, J. Dollimore, T. Kindberg. *Distributed Systems Concepts and Design*. Addison-Wesley, Queen Mary and Westfield College, University of London, 2nd Edition, 1994.

[Duden93] Duden. *Informatik*. Dudenverlag, 1993.

[Edwards97] J. Edwards and D. DeVoe. *3-Tier Client/Server At Work*. John Wiley & Sons Inc., 1997.

[Ellis90] Margaret A. Ellis, Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Addison Wesley, 1990.

[Fire95] D. G. Firesmith and E. M. Eykholt. *Dictionary of Object Technology: the Definitve Desk Reference*. SIGS Books Inc., New York, 1st Edition, 1995.

[Flanagan97] D. Flanagan. *JAVA in a Nutshell*. O´Reilly & Associates Inc., 2nd Edition, 1997.

[Farber98] D. Farber. *Distributed Call Centers: New Technology, New Approach*. www.tmcnet.com/articles/ctimag/0298/ccenter001.htm, Accessed 18 March 1998.

[Farley00] Jim Farley. *Microsoft .NET vs. J2EE: How Do They Stack Up?*. java.oreilly.com/news/farley_0800.html, August 2000.

[Fayad97] Mohamed E. Fayad, Douglas C. Schmidt. *Object-Oriented Application Framework*. Communications of the ACM, Vol. 40 No. 10, pp. 32-38, October 1997.

[Ferraro97] A. Ferraro and E.H. Rogers. *Petri Nets in the Evaluation of Collaborative Systems*. In: IEEE Systems, Man and Cybernetics Conference Proceedings, 1997.

[Foote88] Foote, Brian. *Designing to faciliate change with object-oriented frameworks*. Master's thesis, University of Illionois at Urbana-Champaign, Urbana, Illinois, 1988.

[Fowler98] S. Fowler. *GUI Design Handbook*. McGraw-Hill, 1998.

[Gamma95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[Genesys] Genesys Corporation. www.genesyslab.com. Accessed 20 August 2001.

[Glass02] Robert L. Glass. *Searching for the Holy Grail of Software Engineering*. Communications of the ACM, Volume 45, Number 5, pp. 15-16, 2002.

[Gray93] J. Gray and A. Reuter. *Transaction Processing: Conecepts and Techniques*. Morgan Kaufmann Publishers Inc., 1993.

[Hampe96] J. Felix Hampe. *Weitergehende CTI-Dienste*. Beitrag zur VoiceCom'96, Dezember 1996.

[Henn98] H. Henn, J. P. Kruse, V. Strawe. *Handbuch Call Center Management*. telepublic Verlag, Hannover, 1998.

[Hopcroft79] J. E. Hopcroft and J. D. Ullmann. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[IBM94] IBM Corporation, Object Technology Products Group. *The System Object Model (SOM) and the Component Object Model (COM)*. Austin, Texas, July 1994.

[ISM98] International Systems Group. *Middleware -- The essential component for enterprise client/server applications*. www.openvms.digital.com/openvms/whitepapers/middleware/-isgmidware.html, Accessed June 1998.

[iSYS98] iSYS Software GmbH. *Postbank Telefonbanking, DV-Feinkonzept*. März 1998.

[Jacobson99] I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[Jain00] Ravi Jain, Farooq M. Anjum, Paolo Missier, Subramanya Shastry. *Java Call Control, Coordination and Transactions*. IEEE Communications Magazine, pp. 108-114, January 2000.

[John88] R. E. Johnson and B. Foote. *Designing Reusable Classes*. Journal of Object-Oriented Programming, Volume 1, Number 2, pp. 22-35, 1988.

[John97a] R. E. Johnson, *Components, Frameworks, Patterns (Extended Abstract)*. University of Illinois, Urbana Illinois, 1997.

[John97b] R. E. Johnson. *Frameworks=(Components + Patterns)*. Communications of the ACM, Vol. 40, No. 10, pp. 39-42, 1997.

[Jongbloed01] Geurt Jongbloed and Geurt Koole. *Managing uncertainty in call centers using Poisson mixtures*. Applied Stochastic Models in Business and Industry, 17, pp. 307-318, 2001.

[Komm98] Kommunicate. *An overview of CTI: editorial for the British Computer Society*. www.kommunicate.co.uk/pages/info_pr/cti.htm, Accessed January 1998.

[Koole00] Ger Koole and Jerome Talim. *Exponential Approximation of Multi-Skill Call Centers Architecture*. In: Proceedings of QNETs 2000, Ilkley, pp. 23/1-10, 2000.

[Koole02] Ger Koole. *Call Center Mathematics, A scientific method for understanding and improving your contact center*. Vrije Universiteit, Division of Mathematics and Computer Science, www.cs.vu.nl/~koole/ccmath, Accessed June 2002.

[Kruchten98] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison Wesley, 1998.

[Kuhlins01] Stefan Kuhlins and Didier Gutacker. *Web-enabled Voice over IP Call Center, An Open Source Based Implementation*. In: IEEE International Conference on Networking (ICN'01), 2001.

[Lippman91] Stanley B. Lippmann. *C++ Primer*. Addison Wesley, 2nd Edition, 1991.

[Lucent97] Lucent Technologies. *Java Telephony API (JTAPI) Programmer's Reference.* 1997.

[Mandl97] P. Mandl. *Transaktionskonzepte im Netz- und Systemmanagement.* Dissertation, Technische Universität Dresden, 1997.

[Mandl00] P. Mandl and N. Bauer. *Rollenspiel.* IX Magazin für professionelle Informationstechnik, Heft 1, pp. 108-111, 2000.

[Mattsson99] M. Mattsson, J. Bosch, M. E. Fayad. *Framework Integration – Problems, Causes, Solutions.* Communications of the ACM, Vol. 42 No. 10, pp. 81-87, October 1999.

[Messer98] Burkhard Messer. *Workflow Management als Verbindung zwischen Call Center und Kundendatenbank.* Workshop "Soziotechnischer Zugang bei der Konstruktion, Einführung und Anwendung von WfMS", Universität Potsdam, Juni 1998.

[Minde99] A. Mindemann, C. Schubert, K.-H. Prümm. *CTI und Call Center.* Addison Wesley Longman, 1999.

[Mowbray95] Thomas J. Mowbray, Ron Zahavi. *The Essential CORBA: Systems Integration Using Distributed Objects.* John Wiley & Sons Inc., 1995.

[Microsoft00] Microsoft. *MSDN Library.* CD Set, July 2000.

[Newton99] Harry Newton. *Newton's Telecom Dictionary.* Miller Freeman, 1999.

[Novell95] Novell. *NetWare Telephony Services Release 2.21 Telephony Services Application Programming Interface (TSAPI).* 1995.

[Oester97] Bernd Oesterreich. *Objektorientierte Softwareentwicklung mit der Unified Modeling Language.* Oldenburg Verlag, 3. Auflage, 1997.

[Oldevik98] Jon Oldevik, Arne-Jørgen Berre. *UML-Based methodology for distributed systems.* In: Proceedings Second International Enterprise Distributed Object Computing Workshop (EDOC'98), November 1998.

[OMG97] Object Management Group. *CORBAservices: CommonObject Services Specification.* 1997.

[OMG98] Object Management Group. *The Common Object Request Broker: Architecture and Specification.* 1998.

[OMG00] Object Management Group. *OMG Unified Modeling Specification.* Version 1.3, 1st edition, March 2000.

[OMG01] Object Management Group, Architecture Board ORMSC. *Model Driven Architecture (MDA).* Document number ormsc/2001-07-01. July, 2001.

[Oracle01] ORACLE Corporation. www.oracle.com. Accessed 21. July 2001.

[Orfali96a] R. Orfali, D. Harkey, J. Edwards. *The Essential Client/Server Survival Guide.* John Wiley & Sons Inc., 2nd Edition, 1996.

[Orfali96b] R. Orfali. *The Essential Distributed Objects Survival Guide.* John Wiley & Sons Inc., 1st Edition, 1996.

[Orfali97] R. Orfali, D. Harkey, J. Edwards. *Instant CORBA.* John Wiley & Sons Inc., 1997.

[Orfali98] R. Orfali and D. Harkey. *Client/Server Programming With Java and CORBA*. John Wiley & Sons Inc., 2nd Edition, 1998.

[Petri62] C.A. Petri. *Kommunikation mit Automaten*. Schriften des Institutes für Instrumentelle Mathematik, Bonn 1962.

[Popien98] Claudia Lienhoff-Popien. *CORBA Kommunikation und Management*. Springer-Verlag, 1998.

[Redlich96] J.-P. Redlich. *Corba 2.0 Praktische Einführung, Für C++ Und Java*. Addison-Wesley, 1st Edition, Bonn 1996.

[Reason00] Karen Resaoner. *The Modernization of a Call Center*. In: Proceedings of SIGUUCS '00, pp. 270-273, November 2000.

[Reisig91] W. Reisig. *Petrinetze, Eine Einführung*. Springer-Verlag, 2. Auflage, 1991.

[Reisig98] W. Reisig. *Elements of Distributed Algorithms, Modeling and Analysis With Petri Nets*. Springer-Verlag, 1998.

[Resnick93] Paul Resnick. *Phone-Based CSCW: Tools and Trials*. ACM Transactions on Information Systems, Vol. 11, No. 4, pp. 401-424, October 1993.

[Rivest78] R. L. Rivest, A. Shamir, L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21, no. 2, pp. 120-126, 1978.

[Rumb93] James Rumbaugh, Muchael Blaha, William Premerlani, Frederick Eddy, William Lorensen. *Objektorientiertes Modellieren und Entwerfen*. Carl Hanser / Prentice Hall International, 1993.

[Rumb99] James Rumbaugh, I. Jacobson, Grady Boch. *The Unified Modeling Language, Reference Model*. Addison Wesley, 1999.

[Schob2000] Robert Schoblick. *CTI Middleware*. Funkschau, 8, pp. 43-45, 2000.

[Schob2000a] Robert Schoblick. *CTI von der Stange*. Funkschau, 9, pp. 43-45, 2000.

[Schob2001] Robert Schoblick. *Telefon-Komfort mit Datenbanken*. Funkschau, 23, pp. 34-37, 2001.

[Siebel01] Siebel Corporation. www.siebel.com. Accessed July 2002.

[Siedersl00] J. Siedersleben, E. Denert. *Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur*. Informatik Spektrum 23.4, pp. 247-257, 2000.

[Siegel98] J. Siegel. *CORBA and the OMA in Enterprise Computing*. Communications of the ACM, 41, no. 10, pp. 37-43, 1998.

[Sing98] S. Singhal and B. Nguyen. *The Java Factor*. Communications of the ACM, 41, no. 6, 34-37, 1998.

[Sommer96] Ian Sommerville. *Software Engineering*. Addison-Wesley, 5th Edition, 1996.

[Stam98] Mathias Stambach. *Web based Customer Service*. Diplomarbeit, Universität St. Gallen, August 1998.

[Strath94] C. R. Strathmeyer. *Client-Server Computing and the Call Center*. TeleProfessional Magazin, 1994.

[Strath96] C. R. Strathmeyer. *An introduction to computer telephony*. www.dialogic.com/company/whitepaper/carlieee.htm, Accessed January 1998.

[SUN98] Sun Microsystems. *The Java Telephony API, An Overview, Version 1.1*. java.sun.com/products/jtapi/jtapi-1.1/Overview.html, Accessed February 1998.

[SUN99] Sun Microsystems. *Enterprise Java Beans Specification, v1.1*. Final Release, Dezember 1999.

[SUN00] Sun Microsystems. *Designing Enterprise Applications With the Java™ 2 Platform, Enterprise Edition*. Version 1.0, Final Release, 2000.

[SUN01] Sun Microsystems. *Enterprise Java Beans Specification, Version 2.0*. Final Release, August 2001.

[Szyperski97] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1997.

[Taligent93] Taligent Inc. *Leveraging Object-Oriented Frameworks*. 1993.

[Taligent94] Taligent Inc. *Building Object-Oriented Frameworks*. 1994.

[Tan89] A. S. Tanenbaum. *Computer Networks*. Prentice Hall International Inc., 2nd Edition, 1989.

[Tan92] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall International Inc., 1992.

[Taylor95] David A. Taylor. *Business Engineering with Object Technology*. John Wiley & Sons Inc., 1995.

[Taylor98] David A. Taylor. *Object Technology: A Manager's Guide*. Addison-Wesley, 2nd Edition, 1998.

[TechCh01] techchannel.de. *Microsoft .NET versus Sun ONE*. www.techchannel.de/betriebssysteme/616/index.html. Februar 2001

[Tyma98] P. Tyma. *Why are we using JAVA again?*. Communications of the ACM, 41, no. 6, pp. 38-42, 1998.

[Vinoski98] S. Vinoski. *New Features for CORBA 3.0*. Communications of the ACM, 41, no. 10, pp. 44-52, 1998.

[Weise99] T. Weise et al. *Lightweight Workflows*. HMD Praxis Der Wirtschaftsinformatik, Heft 210, pp. 83-100, 1999.

[Weise00] Weise, P. Mandl. *Verteilte Transaktionsverarbeitung auf Basis von CORBA*, OBJEKT Spektrum, Heft 1, pp. 53-61, 2000.

[Woetzel93] Gerd Woetzel, Thomas Kreifelts. *The Use of Petri Nets for Modeling Workflow in the DOMINO System*. Workshop on CSCW, Petri Nets and Related Formalisms, Int. Conf. on Appl. and Theory of Petri Nets. June 21- 25, 1993.

[Vasters01] Vasters, Oellers, Javidi, Jung, Freiberger, DePetrillo. *.NET-Crashkurs*. Microsoft Press, 2001.

[Versit96a] Versit. *Computer Telephony Integration (CTI) EncyclopediaVolume 1: CTI Concepts and Terminology*. Version 1.0, 1996.

[Versit96b] Versit. *Computer Telephony Integration (CTI) EncyclopediaVolume 2: CTI Configurations and Landscapes*. Version 1.0, 1996.

[Versit96c] Versit. *Computer Telephony Integration (CTI) EncyclopediaVolume 3: Telephony Feature Set*. Version 1.0, 1996.

[Versit96d] Versit. *Computer Telephony Integration (CTI) EncyclopediaVolume 2: Call Flow Scenarios*. Version 1.0, 1996.

[Versit96e] Versit. *Computer Telephony Integration (CTI) EncyclopediaVolume 2: CTI Protocols*. Version 1.0, 1996.

[Versit96f] Versit. *Computer Telephony Integration (CTI) EncyclopediaVolume 2: Versit TSAPI*. Version 1.0, 1996.

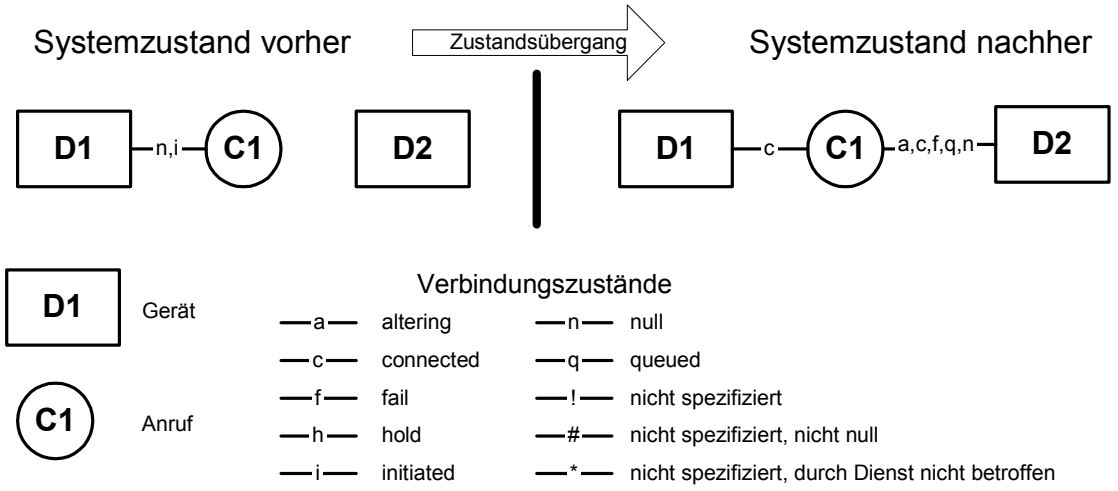
[Wetterau98] James Wetterau. *CTI in the Corporate Enterprise*. International Journal of Network Management, 8, pp. 235-243, 1998.

[Yourdon96] Edward Yourdon, Cal Argila. *Case studies in object oriented analysis and design*. Prentice Hall, 1996.

13 ANHANG

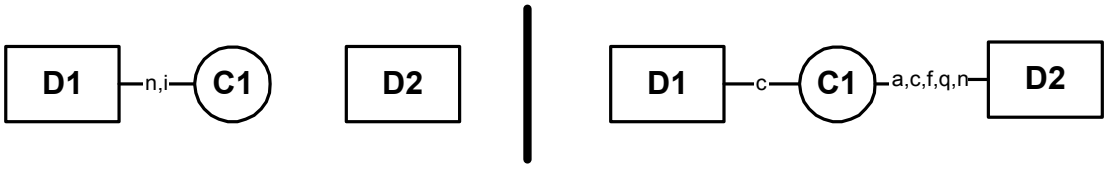
13.1 Telefonie-Prozesse

Nachfolgend werden nochmals alle für die Betrachtungen in der Arbeit grundlegenden Prozesse eines Telefonsystems kurz erläutert und anhand der eingeführten Notation spezifiziert. Die Bedeutung der Elemente der Notation erläutert nachfolgende Grafik nochmals.



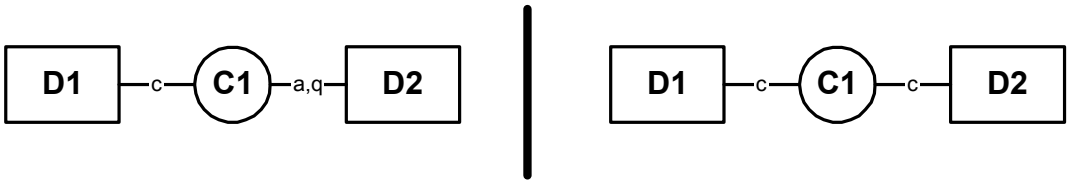
13.1.1 Anruf einleiten

Die Verbindung des Geräts D1 ist entweder nicht vorhanden (*null)* oder im Zustand *initiated*, je nachdem, ob direkt mit aufgelegtem Hörer gewählt werden kann oder ein Abheben zuerst notwendig ist. Den ersten Fall, den man auch als *off-hook dialing* bezeichnet, kennt man beispielsweise von Mobiltelefonen. Der Dienst ist beendet, sobald die Verbindung zwischen D1 und C1 den Zustand *connected* erreicht hat. Die Verbindung zu D2 kann zu diesem Zeitpunkt verschiedene Zustände besitzen. In den meisten Fällen wird dies *alerting* sein und das Gerät den Zustand entsprechend anzeigen (Klingeln). Es ist aber auch denkbar, dass die Verbindung direkt in den Zustand *connected* gelangt. In diesem Fall antwortet das Gerät D2 automatisch auf eingehende Anrufe (*auto answer).*



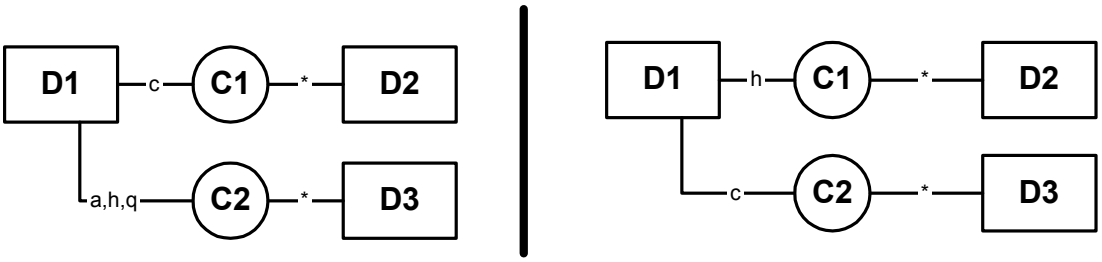
13.1.2 Anruf entgegennehmen

Dieser Dienst folgt in der Regel dem Initiieren eines Anrufs. Die Verbindung zu D2, die sich im Zustand *alerting* oder *queued* befindet geht in *connected* über, wodurch eine Kommunikation zwischen D1 und D2 möglich wird.



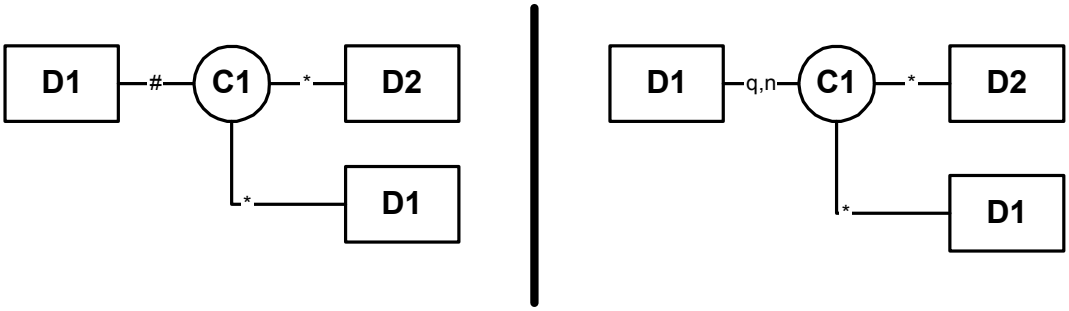
13.1.3 Makeln

Diesen Dienst bezeichnet man auch als *Alternate Call*. Dabei wechselt das Gerät D1 zwischen zwei Anrufen (C1, C2), wobei die erste Verbindung auf *hold* und die zweite auf *connected* wechselt.

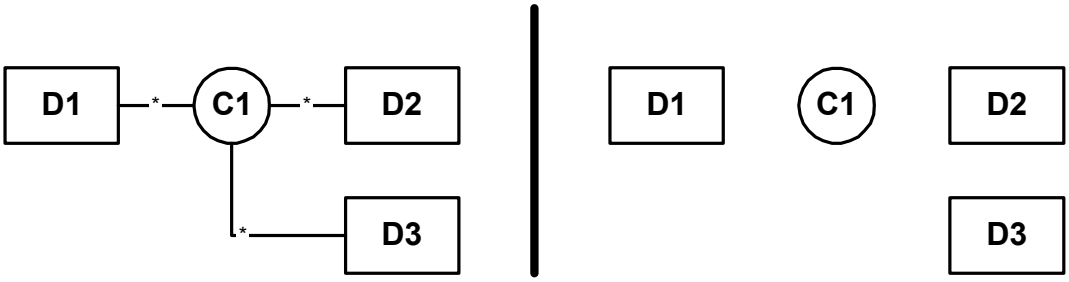


13.1.4 Anruf beenden

Das Beenden eines Anrufs bedeutet in der Regel das Beenden der entsprechenden Verbindung. Dies bewirkt ein Aufruf des Dienstes *clear connection*. Nach Ausführung dieses Dienstes geht die Verbindung zwischen D1 und C1 in den Zustand *null* über, während die anderen Verbindungen unberührt bleiben.

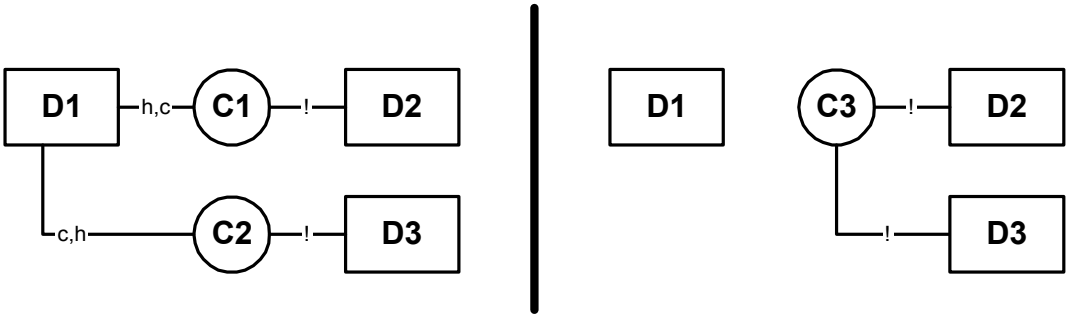


Eine andere Variante einen Anruf zu beenden geschieht mittels *clear call*, wodurch der gesamte Anruf gelöscht wird.

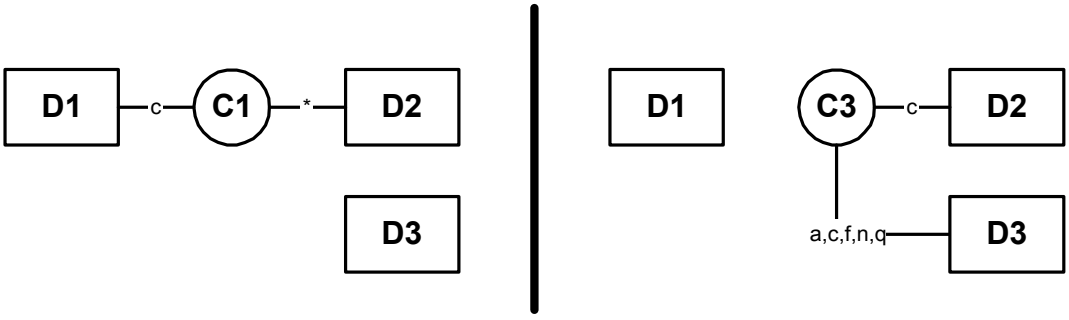


13.1.5 Weiterleitung

Dieser Dienst trennt Gerät D1 von den beiden Anrufen C1 und C2 und verbindet die Teilnehmer D2 sowie D3 mit einem neuen Anruf C3. Die Verbindung zwischen D1 und C1 bzw. C2 kann dabei den Zustand *hold* oder *connected* haben. Mit Hilfe dieses Services sind nun zwei unterschiedliche Arten der Weiterleitung möglich.



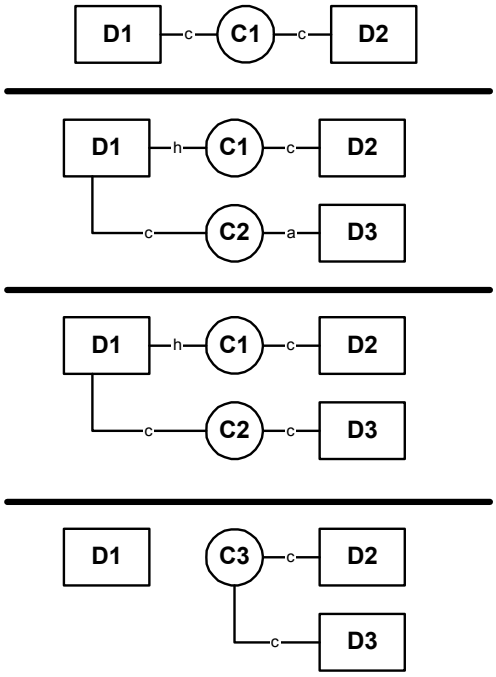
Die direkte Weiterleitung (*single step transfer)* ist nachfolgend skizziert. Die Verbindung zwischen D1 sowie C1 ist dabei im Zustand *connected*. Durch Aufruf des Dienstes für eine Weiterleitung wird direkt ein neuer Partner D3 mit D2 verbunden und die Verbindung zu D1 getrennt. Analog zu dem Fall eines neuen Anrufs kann die Verbindung D3 C3 nun mehrere Zustände annehmen. Der Normalfall ist auch hier, dass der neue Anruf am Gerät angezeigt wird (*alerting)* und nach Abnehmen des Hörers von D3 in den Zustand *connected* übergeht.



Diese Weiterleitung findet man in Call Centern relativ häufig, wenn beispielsweise an einen beliebigen Mitarbeiter einer bestimmten Gruppe oder aber an einen Sprachcomputer vermittelt wird. In diesem Fall ist eine Verbindung zwischen D1 und D3 unnötig. Ein weiterer Grund für den Einsatz dieses Verfahrens

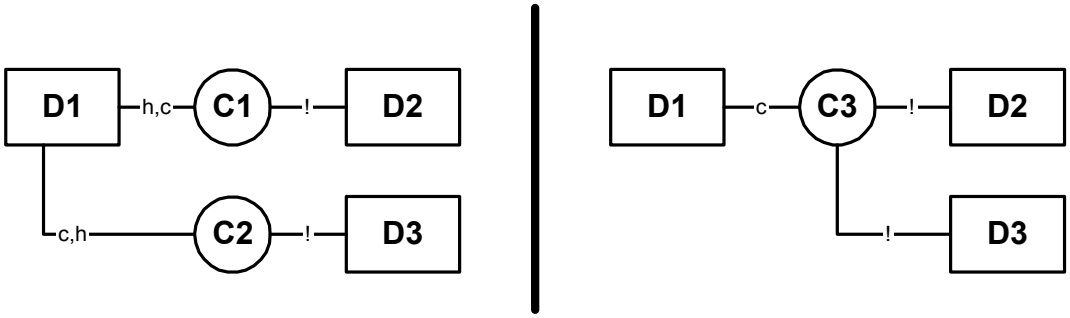
ist die in Call Centern so wichtige Frage der Effizienz. D1 ist hier sofort nach Aufruf des Dienstes wieder frei und der Mitarbeiter kann ein neues Gespräch annehmen.

Die Alternative dazu ist eine Weiterleitung mit Rückfrage. In diesem Falle wird als erster Schritt die Verbindung von D1 auf *hold* geschaltet. Daraufhin wird ein neuer Anruf initiiert und versucht, eine Verbindung mit einem neuen Teilnehmer D3 aufzubauen. Akzeptiert dieser den Anruf, ist eine Kommunikation zwischen D1 und D3 möglich. In der Regel wird dabei der Mitarbeiter an D3 über den bevorstehenden Transfer informiert. Erst nach dieser Rückfrage wird mittels des Dienstes *transfer call* ein neuer Anruf D3 geschaffen und D2 mit D3 verbunden.

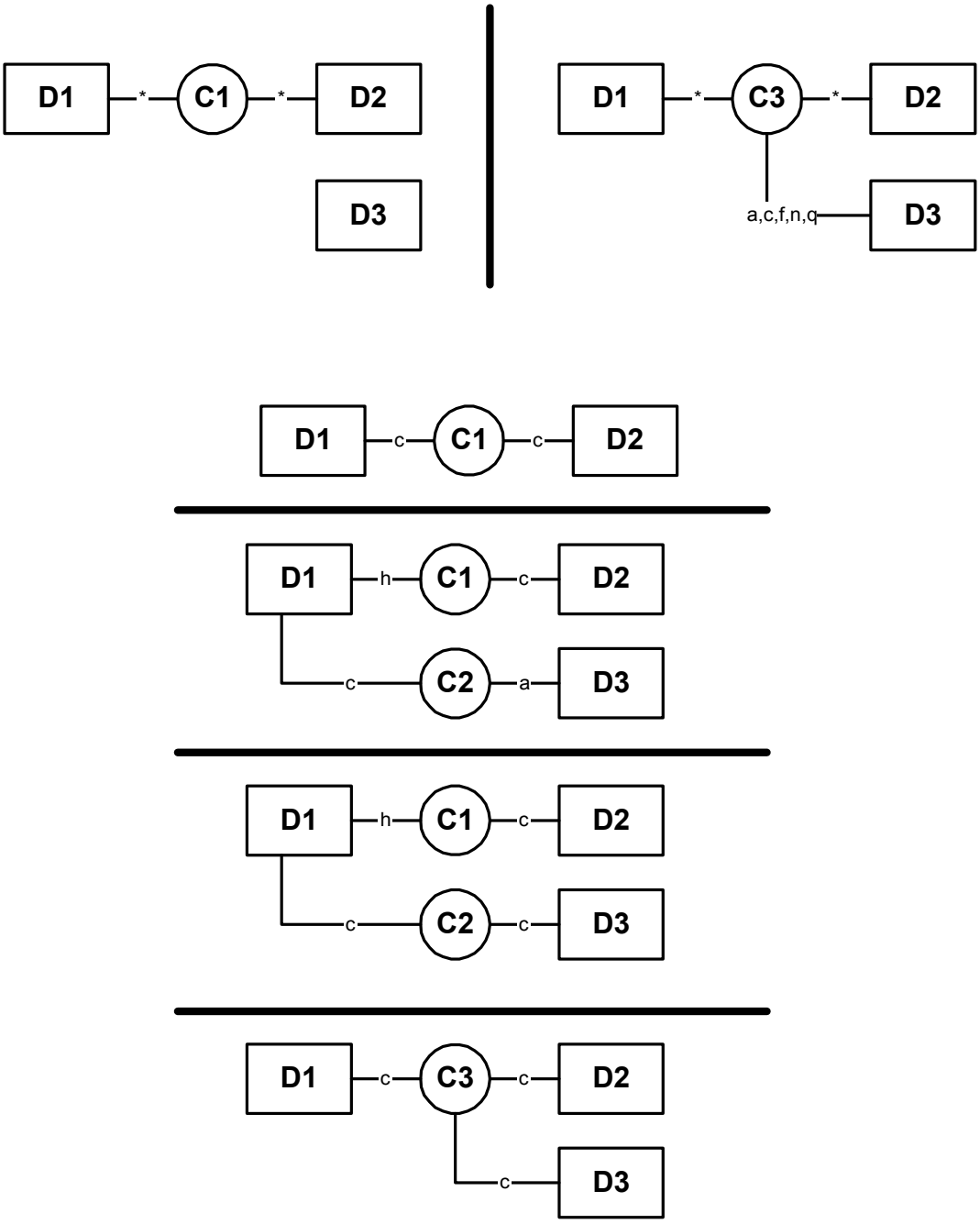


13.1.6 Konferenz

Der Unterschied im Vergleich zur Weiterleitung liegt bei der Konferenz in der weiterhin bestehenden Verbindung von D1 und dem neuen Anruf D3, die sich im Zustand *connected* befindet.

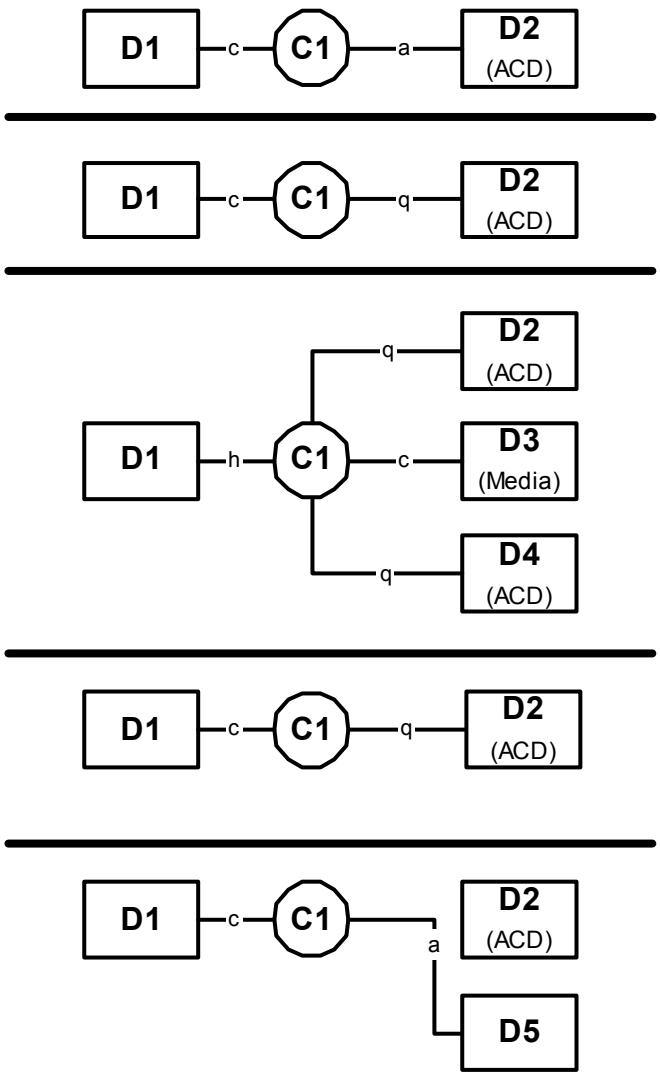


Analog zur Weiterleitung existiert auch für die Konferenz eine sog. *single step conference*, bei der der Zwischenschritt mit der Verbindung zwischen D1 und D3 übersprungen wird.



13.1.7 ACD Warteschlange

Ein besonders für Call Center typischer Ablauf ist das Einreihen eines Anrufs in eine Warteschlange durch das ACD-System. Dabei ist D1 anfangs nicht mit einem Endgerät eines Arbeitsplatzes sondern mit dem ACD-System selbst verbunden. Das System setzt den Anruf auf *queued* und versucht, einen geeigneten Teilnehmer zu finden. Dazu kann es, sofern nötig, weitere ACD-Systeme einschließen, die ebenfalls mit dem Anruf verbunden werden. Außerdem wird in der Regel ein spezieller Teilnehmer (D3) eingebunden, der dem Anrufer beispielsweise Musik vorspielt bzw. andere Informationen per Sprache übermittelt. Die Verbindung dieses Teilnehmers ist dementsprechend im Zustand *connected*. Wurde ein geeignetes Ziel (D5) ermittelt, so werden D4 und D3 von diesem Anruf getrennt und dieser weitergeleitet. Der Anruf liegt nun an D5 an und die entsprechende Verbindung ist im Zustand *alerting*.

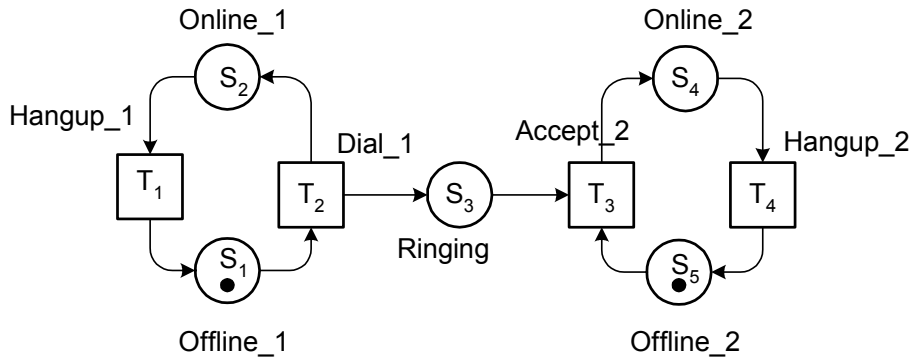


13.2 Analyse der Petri-Netze

Die in der Arbeit entworfenen Petri-Netze wurden mit dem Programm *Petri-Netze für Windows, Version 2.01* getestet und analysiert, um eventuelle Schwachstellen (Konflikte, Kontakte, Verklemmungen etc.) zu entdecken. Die detaillierten Ergebnisse der jeweiligen Analysen sind nachfolgend nochmals aufgeführt. Da die Software Stellen und Transitionen lediglich durchnummeriert, wurden die jeweiligen Grafiken der Übersichtlichkeit halber nochmals erweitert und jede Stelle bzw. Transition durch einen Bezeichner S_n bzw. T_n ergänzt.

Die angegebenen Netzmarkierungen geben dabei jeweils an, welche der Stellen gerade belegt sind und welche nicht. M = (1,0,0,0,1) bedeutet also, dass sowohl S_1 als auch S_5 gerade belegt sind. Die Notation M1--T1-->M2 gibt an, dass durch Schalten der Transition T_1 die Markierung von M1 nach M2 wechselt.

13.2.1 Anruf (Variante 1)



Analyse:

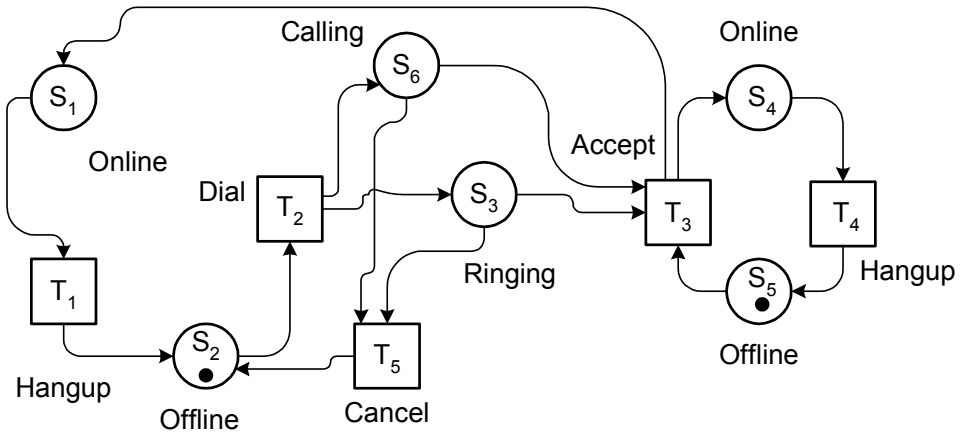
Allgemeine Netzdaten:
- Das Netz besteht aus 5 Stellen.
- Das Netz besteht aus 4 Transitionen.
- Der Analyse wurde die starke Schaltregel zugrundegelegt.

Zusammenfassung der Ergebnisse:
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
- Das Netz ist beschränkt: Die größte Stellenmarkierung ist 1, das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Das Netz ist konfliktfrei.
- Es treten 2 Kontaktsituationen auf.
- Der Erreichbarkeitsgraph ist stark zusammenhängend, das Netz ist daher reversibel.
- Das Netz ist lebendig.

Die Ergebnisse im Detail:

Knotenmenge (alle möglichen Netz-Markierungen):	Kantenmenge (Erreichbarkeiten):	Kontaktsituationen
M1 = (1,0,0,0,1) M2 = (0,1,1,0,1) M3 = (1,0,1,0,1) M4 = (0,1,0,1,0) M5 = (1,0,0,1,0) M6 = (0,1,0,0,1) M7 = (0,1,1,1,0) M8 = (1,0,1,1,0)	M1--T1-->M2 M2--T2-->M3 M2--T3-->M4 M3--T3-->M5 M4--T2-->M5 M4--T4-->M6 M5--T1-->M7 M5--T4-->M1 M6--T2-->M1 M7--T2-->M8 M7--T4-->M2 M8--T4-->M3	M3 bei T1 M8 bei T1

13.2.2 Anruf (Variante 2)



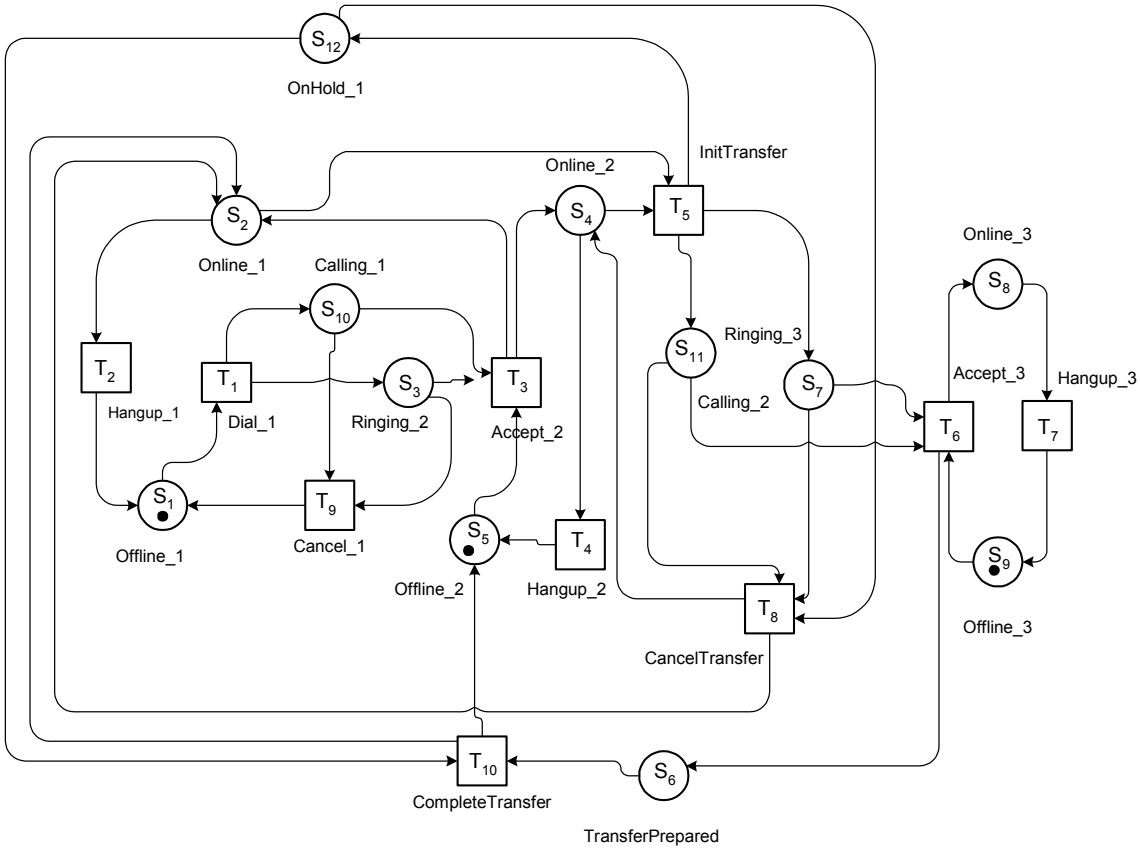
Allgemeine Netzdaten:
- Das Netz besteht aus 6 Stellen.
- Das Netz besteht aus 5 Transitionen.
- Der Analyse wurde die starke Schaltregel zugrundegelegt.

Zusammenfassung der Ergebnisse:
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
- Das Netz ist beschränkt: Die größte Stellenmarkierung ist 1, das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei einer Netz-Markierung treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Der Erreichbarkeitsgraph ist stark zusammenhängend, das Netz ist daher reversibel.
- Das Netz ist lebendig.

Die Ergebnisse im Detail:

Knotenmenge (alle möglichen Netz-Markierungen):	Kantenmenge (Erreichbar-keiten):	Konfliktsituationen:
M1 = (0,1,0,0,1,0) M2 = (0,0,1,0,1,1) M3 = (1,0,0,1,0,0) M4 = (0,1,0,1,0,0) M5 = (1,0,0,0,1,0) M6 = (0,0,1,1,0,1)	M1--T2-->M2 M2--T3-->M3 M2--T5-->M1 M3--T1-->M4 M3--T4-->M5 M4--T2-->M6 M4--T4-->M1 M5--T1-->M1 M6--T4-->M2 M6--T5-->M4	M2 bei T3 M2 bei T5

13.2.3 Weiterleitung



Allgemeine Netzdaten:
- Das Netz besteht aus 12 Stellen.
- Das Netz besteht aus 10 Transitionen.
- Der Analyse wurde die starke Schaltregel zugrundegelegt.

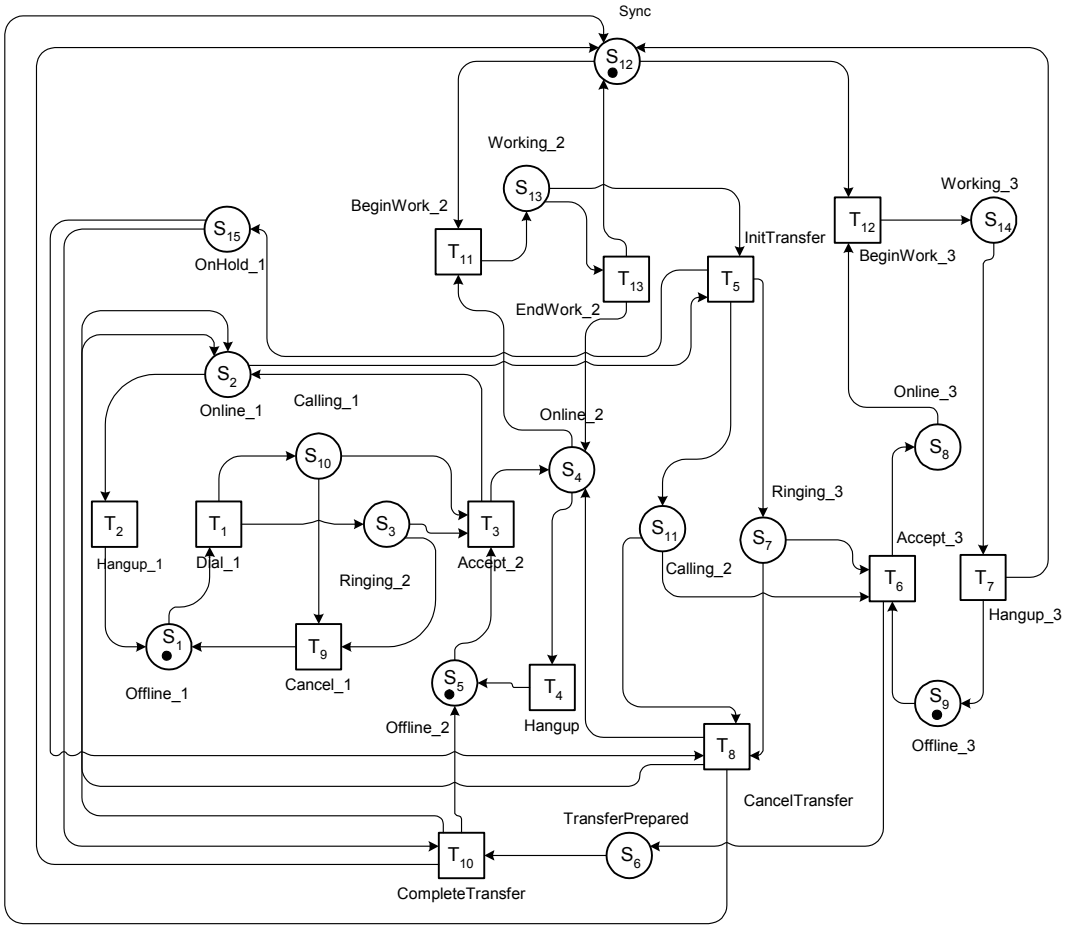
Zusammenfassung der Ergebnisse:
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
- Das Netz ist beschränkt: Die größte Stellenmarkierung ist 1, das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei 5 Netz-Markierungen treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Der Erreichbarkeitsgraph ist stark zusammenhängend, das Netz ist daher reversibel.
- Das Netz ist lebendig.

Die Ergebnisse im Detail:

Knotenmenge (alle möglichen Netz-Markierungen):	Kantenmenge (Erreichbarkeiten):	Konfliktsituationen:
M1 = (1,0,0,0,1,0,0,0,1,0,0,0) M2 = (0,0,1,0,1,0,0,0,1,1,0,0)	M1--T1-->M2 M2--T3-->M3	M2 bei T3 M2 bei T9

M3 = (0,1,0,1,0,0,0,0,1,0,0,0) M4 = (1,0,0,1,0,0,0,0,1,0,0,0) M5 = (0,1,0,0,1,0,0,0,1,0,0,0) M6 = (0,0,0,0,0,0,1,0,1,0,1,1) M7 = (0,0,1,1,0,0,0,0,1,1,0,0) M8 = (0,0,0,0,0,1,0,1,0,0,0,1) M9 = (0,0,0,0,0,1,0,0,1,0,0,1) M10 = (0,1,0,0,1,0,0,1,0,0,0,0) M11 = (1,0,0,0,1,0,0,1,0,0,0,0) M12 = (0,0,1,0,1,0,0,1,0,1,0,0) M13 = (0,1,0,1,0,0,0,1,0,0,0,0) M14 = (1,0,0,1,0,0,0,1,0,0,0,0) M15 = (0,0,0,0,0,0,1,1,0,0,1,1) M16 = (0,0,1,1,0,0,0,1,0,1,0,0)	M2--T9-->M1 M3--T2-->M4 M3--T4-->M5 M3--T5-->M6 M4--T1-->M7 M4--T4-->M1 M5--T2-->M1 M6--T6-->M8 M6--T8-->M3 M7--T4-->M2 M7--T9-->M4 M8--T7-->M9 M8--T10-->M10 M9--T10-->M5 M10--T2-->M11 M10--T7-->M5 M11--T1-->M12 M11--T7-->M1 M12--T3-->M13 M12--T7-->M2 M12--T9-->M11 M13--T2-->M14 M13--T4-->M10 M13--T5-->M15 M13--T7-->M3 M14--T1-->M16 M14--T4-->M11 M14--T7-->M4 M15--T7-->M6 M15--T8-->M13 M16--T4-->M12 M16--T7-->M7 M16--T9-->M14	M3 bei T2 M3 bei T4 M3 bei T5 M6 bei T6 M6 bei T8 M12 bei T3 M12 bei T9 M13 bei T2 M13 bei T4 M13 bei T5

13.2.4 Weiterleitung mit Synchronisation



Allgemeine Netzdaten:
- Das Netz besteht aus 15 Stellen.
- Das Netz besteht aus 13 Transitionen.
- Der Analyse wurde die starke Schaltregel zugrundegelegt.

Zusammenfassung der Ergebnisse:
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
- Das Netz ist beschränkt: Die größte Stellenmarkierung ist 1, das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei 12 Netz-Markierungen treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Der Erreichbarkeitsgraph ist stark zusammenhängend, das Netz ist daher reversibel.
- Das Netz ist lebendig.

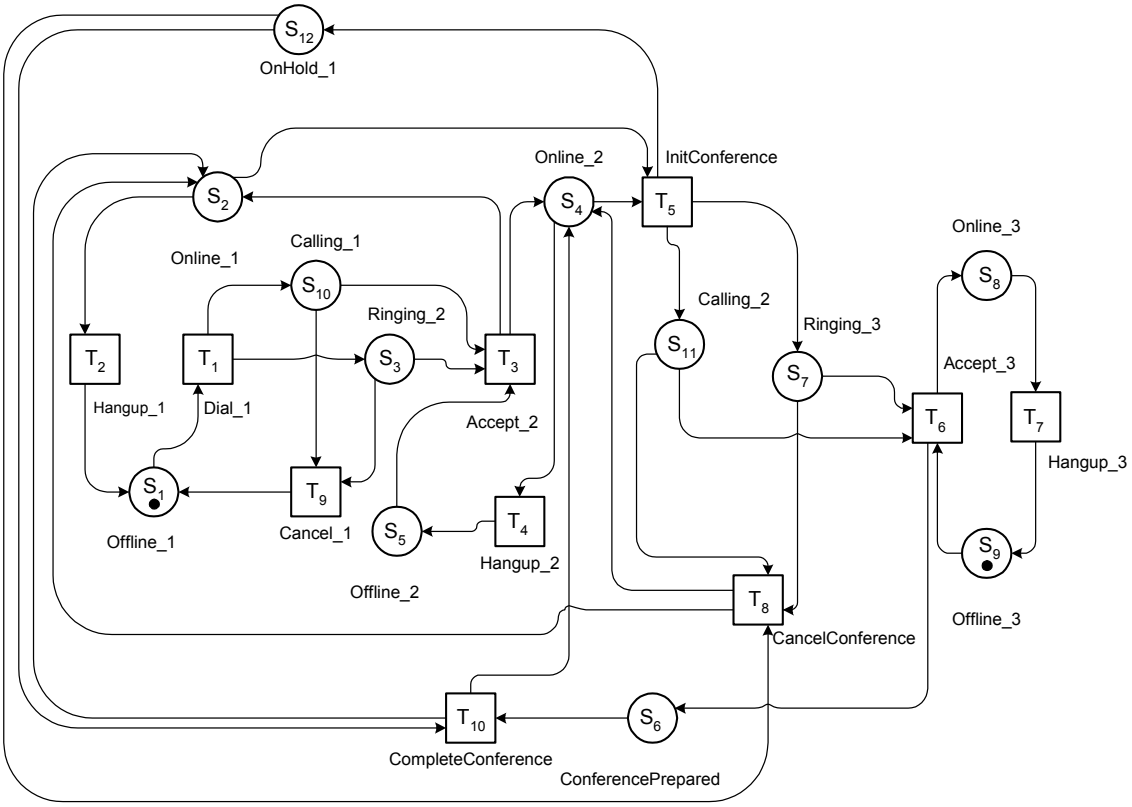
Die Ergebnisse im Detail:

Knotenmenge (alle möglichen Netz-Markierungen):	Kantenmenge (Erreichbar-keiten):	Konfliktsituationen:
M1 = (1,0,0,0,1,0,0,0,1,0,0,1,0,0,0) M2 = (0,0,1,0,1,0,0,0,1,1,0,1,0,0,0)	M1--T1-->M2 M2--T3-->M3	M2 bei T3 M2 bei T9

M3 = (0,1,0,1,0,0,0,0,1,0,0,1,0,0,0)	M2--T9-->M1	M3 bei T4
M4 = (1,0,0,1,0,0,0,0,1,0,0,1,0,0,0)	M3--T2-->M4	M3 bei T11
M5 = (0,1,0,0,1,0,0,0,1,0,0,1,0,0,0)	M3--T4-->M5	M4 bei T4
M6 = (0,1,0,0,0,0,0,0,1,0,0,0,1,0,0)	M3--T11-->M6	M4 bei T11
M7 = (0,0,1,1,0,0,0,0,1,1,0,1,0,0,0)	M4--T1-->M7	M6 bei T2
M8 = (1,0,0,0,0,0,0,0,1,0,0,0,1,0,0)	M4--T4-->M1	M6 bei T5
M9 = (0,0,0,0,0,0,1,0,1,0,1,0,0,0,1)	M4--T11-->M8	M6 bei T13
M10 = (0,0,1,0,0,0,0,0,1,1,0,0,1,0,0)	M5--T2-->M1	M7 bei T4
M11 = (0,0,0,0,0,1,0,1,0,0,0,0,0,0,1)	M6--T2-->M8	M7 bei T11
M12 = (0,1,0,0,1,0,0,1,0,0,0,1,0,0,0)	M6--T5-->M9	M9 bei T6
M13 = (1,0,0,0,1,0,0,1,0,0,0,1,0,0,0)	M6--T13-->M3	M9 bei T8
M14 = (0,1,0,0,1,0,0,0,0,0,0,0,0,1,0)	M7--T4-->M2	M15 bei T3
M15 = (0,0,1,0,1,0,0,1,0,1,0,1,0,0,0)	M7--T9-->M4	M15 bei T9
M16 = (1,0,0,0,1,0,0,0,0,0,0,0,0,1,0)	M7--T11-->M10	M17 bei T4
M17 = (0,1,0,1,0,0,0,1,0,0,0,1,0,0,0)	M8--T1-->M10	M17 bei T11
M18 = (0,0,1,0,1,0,0,0,0,1,0,0,0,1,0)	M8--T13-->M4	M17 bei T12
M19 = (1,0,0,1,0,0,0,1,0,0,0,1,0,0,0)	M9--T6-->M11	M18 bei T3
M20 = (0,1,0,0,0,0,0,1,0,0,0,0,1,0,0)	M9--T8-->M3	M18 bei T9
M21 = (0,1,0,1,0,0,0,0,0,0,0,0,0,1,0)	M10--T9-->M8	M19 bei T4
M22 = (0,0,1,1,0,0,0,1,0,1,0,1,0,0,0)	M10--T13-->M7	M19 bei T11
M23 = (1,0,0,0,0,0,0,1,0,0,0,0,1,0,0)	M11--T10-->M12	M19 bei T12
M24 = (1,0,0,1,0,0,0,0,0,0,0,0,0,1,0)	M12--T2-->M13	M20 bei T2
M25 = (0,0,0,0,0,0,1,1,0,0,1,0,0,0,1)	M12--T12-->M14	M20 bei T5
M26 = (0,0,1,0,0,0,0,1,0,1,0,0,1,0,0)	M13--T1-->M15	M20 bei T13
M27 = (0,0,1,1,0,0,0,0,0,1,0,0,0,1,0)	M13--T12-->M16	M22 bei T4
	M14--T2-->M16	M22 bei T11
	M14--T7-->M5	M22 bei T12
	M15--T3-->M17	
	M15--T9-->M13	
	M15--T12-->M18	
	M16--T1-->M18	
	M16--T7-->M1	
	M17--T2-->M19	
	M17--T4-->M12	
	M17--T11-->M20	
	M17--T12-->M21	
	M18--T3-->M21	
	M18--T7-->M2	
	M18--T9-->M16	
	M19--T1-->M22	
	M19--T4-->M13	
	M19--T11-->M23	
	M19--T12-->M24	
	M20--T2-->M23	
	M20--T5-->M25	
	M20--T13-->M17	
	M21--T2-->M24	
	M21--T4-->M14	
	M21--T7-->M3	
	M22--T4-->M15	
	M22--T9-->M19	
	M22--T11-->M26	
	M22--T12-->M27	
	M23--T1-->M26	
	M23--T13-->M19	
	M24--T1-->M27	

	M24--T4-->M16 M24--T7-->M4 M25--T8-->M17 M26--T9-->M23 M26--T13-->M22 M27--T4-->M18 M27--T7-->M7 M27--T9-->M24	

13.2.5 Konferenz



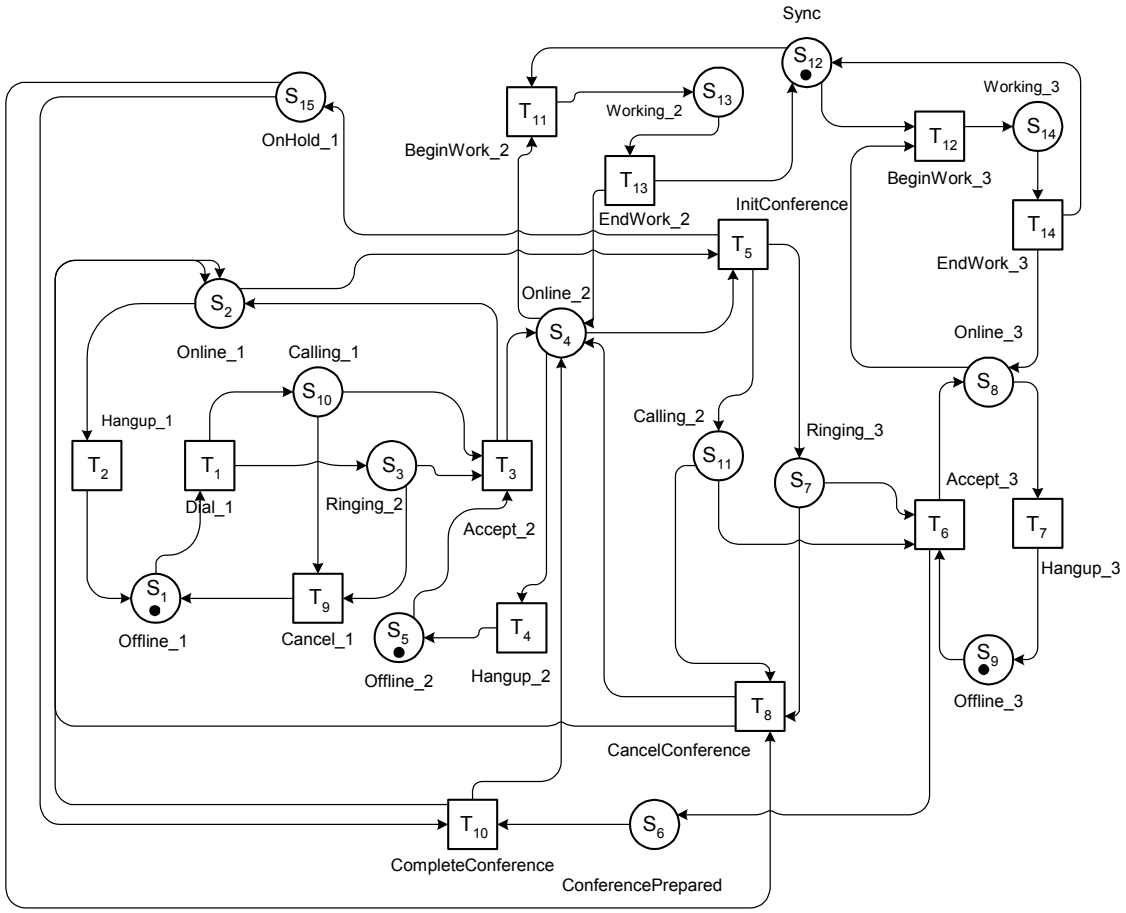
Allgemeine Netzdaten:
- Das Netz besteht aus 12 Stellen.
- Das Netz besteht aus 10 Transitionen.
- Der Analyse wurde die starke Schaltregel zugrundegelegt.

Zusammenfassung der Ergebnisse:
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
- Das Netz ist beschränkt: Die größte Stellenmarkierung ist 1, das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei 5 Netz-Markierungen treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Der Erreichbarkeitsgraph ist stark zusammenhängend,. das Netz ist daher reversibel.
- Das Netz ist lebendig.

Die Ergebnisse im Detail:

Knotenmenge (alle möglichen Netz-Markierungen):	Kantenmenge (Erreichbar-keiten):	Konfliktsituationen:
M1 = (1,0,0,0,1,0,0,0,1,0,0,0) M2 = (0,0,1,0,1,0,0,0,1,1,0,0) M3 = (0,1,0,1,0,0,0,0,1,0,0,0) M4 = (1,0,0,1,0,0,0,0,1,0,0,0) M5 = (0,1,0,0,1,0,0,0,1,0,0,0) M6 = (0,0,0,0,0,0,1,0,1,0,1,1) M7 = (0,0,1,1,0,0,0,0,1,1,0,0) M8 = (0,0,0,0,0,1,0,1,0,0,0,1) M9 = (0,0,0,0,0,1,0,0,1,0,0,1) M10 = (0,1,0,1,0,0,0,1,0,0,0,0) M11 = (1,0,0,1,0,0,0,1,0,0,0,0) M12 = (0,1,0,0,1,0,0,1,0,0,0,0) M13 = (0,0,0,0,0,0,1,1,0,0,1,1) M14 = (0,0,1,1,0,0,0,1,0,1,0,0) M15 = (1,0,0,0,1,0,0,1,0,0,0,0) M16 = (0,0,1,0,1,0,0,1,0,1,0,0)	M1--T1-->M2 M2--T3-->M3 M2--T9-->M1 M3--T2-->M4 M3--T4-->M5 M3--T5-->M6 M4--T1-->M7 M4--T4-->M1 M5--T2-->M1 M6--T6-->M8 M6--T8-->M3 M7--T4-->M2 M7--T9-->M4 M8--T7-->M9 M8--T10-->M10 M9--T10-->M3 M10--T2-->M11 M10--T4-->M12 M10--T5-->M13 M10--T7-->M3 M11--T1-->M14 M11--T4-->M15 M11--T7-->M4 M12--T2-->M15 M12--T7-->M5 M13--T7-->M6 M13--T8-->M10 M14--T4-->M16 M14--T7-->M7 M14--T9-->M11 M15--T1-->M16 M15--T7-->M1 M16--T3-->M10 M16--T7-->M2 M16--T9-->M15	M2 bei T3 M2 bei T9 M3 bei T2 M3 bei T4 M3 bei T5 M6 bei T6 M6 bei T8 M10 bei T2 M10 bei T4 M10 bei T5 M16 bei T3 M16 bei T9

13.2.6 Konferenz mit Synchronisation



Allgemeine Netzdaten:
- Das Netz besteht aus 15 Stellen.
- Das Netz besteht aus 14 Transitionen.
- Der Analyse wurde die starke Schaltregel zugrundegelegt.

Zusammenfassung der Ergebnisse:
- Berechnet wurde der Erreichbarkeitsgraph und dessen Kondensation.
- Das Netz ist beschränkt: Die größte Stellenmarkierung ist 1, das Netz ist daher sicher.
- Das Netz enthält keine toten (nicht schaltfähigen) Transitionen.
- Es gibt keine partiellen Verklemmungen.
- Es gibt keine totalen Verklemmungen (tote Markierungen).
- Bei 15 Netz-Markierungen treten Konflikte auf.
- Das Netz ist kontaktfrei.
- Der Erreichbarkeitsgraph ist stark zusammenhängend, das Netz ist daher reversibel.
- Das Netz ist lebendig.

Die Ergebnisse im Detail:

Knotenmenge (alle möglichen Netz-Markierungen):	Kantenmenge (Erreichbar-keiten):	Konfliktsituationen:
M1 = (1,0,0,0,1,0,0,0,1,0,0,1,0,0,0)	M1--T1-->M2	M2 bei T3
M2 = (0,0,1,0,1,0,0,0,1,1,0,1,0,0,0)	M2--T3-->M3	M2 bei T9
M3 = (0,1,0,1,0,0,0,0,1,0,0,1,0,0,0)	M2--T9-->M1	M3 bei T2

M4 = (1,0,0,1,0,0,0,0,1,0,0,1,0,0,0)	M3--T2-->M4	M3 bei T4
M5 = (0,1,0,0,1,0,0,0,1,0,0,1,0,0,0)	M3--T4-->M5	M3 bei T5
M6 = (0,0,0,0,0,0,1,0,1,0,1,1,0,0,1)	M3--T5-->M6	M3 bei T11
M7 = (0,1,0,0,0,0,0,0,1,0,0,0,1,0,0)	M3--T11-->M7	M4 bei T4
M8 = (0,0,1,1,0,0,0,0,1,1,0,1,0,0,0)	M4--T1-->M8	M4 bei T11
M9 = (1,0,0,0,0,0,0,0,1,0,0,0,1,0,0)	M4--T4-->M1	M6 bei T6
M10 = (0,0,0,0,0,1,0,1,0,0,0,1,0,0,1)	M4--T11-->M9	M6 bei T8
M11 = (0,0,1,0,0,0,0,0,1,1,0,0,1,0,0)	M5--T2-->M1	M8 bei T4
M12 = (0,0,0,0,0,1,0,0,1,0,0,1,0,0,1)	M6--T6-->M10	M8 bei T11
M13 = (0,1,0,1,0,0,0,1,0,0,0,1,0,0,0)	M6--T8-->M3	M10 bei T7
M14 = (0,0,0,0,0,1,0,0,0,0,0,0,0,1,1)	M7--T2-->M9	M10 bei T12
M15 = (1,0,0,1,0,0,0,1,0,0,0,1,0,0,0)	M7--T13-->M3	M13 bei T2
M16 = (0,1,0,0,1,0,0,1,0,0,0,1,0,0,0)	M8--T4-->M2	M13 bei T4
M17 = (0,0,0,0,0,0,1,1,0,0,1,1,0,0,1)	M8--T9-->M4	M13 bei T5
M18 = (0,1,0,0,0,0,0,1,0,0,0,0,1,0,0)	M8--T11-->M11	M13 bei T7
M19 = (0,1,0,1,0,0,0,0,0,0,0,0,0,1,0)	M9--T1-->M11	M13 bei T11
M20 = (0,0,1,1,0,0,0,1,0,1,0,1,0,0,0)	M9--T13-->M4	M13 bei T12
M21 = (1,0,0,0,1,0,0,1,0,0,0,1,0,0,0)	M10--T7-->M12	M15 bei T4
M22 = (1,0,0,0,0,0,0,1,0,0,0,0,1,0,0)	M10--T10-->M13	M15 bei T7
M23 = (1,0,0,1,0,0,0,0,0,0,0,0,0,1,0)	M10--T12-->M14	M15 bei T11
M24 = (0,1,0,0,1,0,0,0,0,0,0,0,0,1,0)	M11--T9-->M9	M15 bei T12
M25 = (0,0,0,0,0,0,1,0,0,0,1,0,0,1,1)	M11--T13-->M8	M16 bei T7
M26 = (0,0,1,0,1,0,0,1,0,1,0,1,0,0,0)	M12--T10-->M3	M16 bei T12
M27 = (0,0,1,0,0,0,0,1,0,1,0,0,1,0,0)	M13--T2-->M15	M17 bei T7
M28 = (0,0,1,1,0,0,0,0,0,1,0,0,0,1,0)	M13--T4-->M16	M17 bei T12
M29 = (1,0,0,0,1,0,0,0,0,0,0,0,0,1,0)	M13--T5-->M17	M19 bei T2
M30 = (0,0,1,0,1,0,0,0,0,1,0,0,0,1,0)	M13--T7-->M3	M19 bei T4
	M13--T11-->M18	M19 bei T5
	M13--T12-->M19	M20 bei T4
	M14--T10-->M19	M20 bei T7
	M14--T14-->M10	M20 bei T11
	M15--T1-->M20	M20 bei T12
	M15--T4-->M21	M21 bei T7
	M15--T7-->M4	M21 bei T12
	M15--T11-->M22	M26 bei T3
	M15--T12-->M23	M26 bei T7
	M16--T2-->M21	M26 bei T9
	M16--T7-->M5	M26 bei T12
	M16--T12-->M24	M30 bei T3
	M17--T7-->M6	M30 bei T9
	M17--T8-->M13	
	M17--T12-->M25	
	M18--T2-->M22	
	M18--T7-->M7	
	M18--T13-->M13	
	M19--T2-->M23	
	M19--T4-->M24	
	M19--T5-->M25	
	M19--T14-->M13	
	M20--T4-->M26	
	M20--T7-->M8	
	M20--T9-->M15	
	M20--T11-->M27	
	M20--T12-->M28	
	M21--T1-->M26	
	M21--T7-->M1	

	M21--T12-->M29 M22--T1-->M27 M22--T7-->M9 M22--T13-->M15 M23--T1-->M28 M23--T4-->M29 M23--T14-->M15 M24--T2-->M29 M24--T14-->M16 M25--T8-->M19 M25--T14-->M17 M26--T3-->M13 M26--T7-->M2 M26--T9-->M21 M26--T12-->M30 M27--T7-->M11 M27--T9-->M22 M27--T13-->M20 M28--T4-->M30 M28--T9-->M23 M28--T14-->M20 M29--T1-->M30 M29--T14-->M21 M30--T3-->M19 M30--T9-->M29 M30--T14-->M26	

13.3 Prototypische Implementierung

Die Implementierung des Prototypen gliedert sich, wie in Java üblich, in mehrere Packages, der Klassen nachfolgend aufgeführt sind. Dabei entspricht jede Klasse genau einer Datei. Bis auf die Implementierung des Testclients enthalten alle Komponenten noch eine in IDL spezifizierte Schnittstelle.

13.3.1 CallControl

Klasse	Beschreibung
BasicTelCtrl_Impl	Implementierung der grundlegenden Telefonfunktionen
CC_ProviderObserver	Observerobjekt für alle Ereignisse des JTAPI-Providers
CallControlServer	Server, der die Implementierungen instanziert und bei CORBA registriert
CC_CallObserver	Observerobjekt für alle Ereignisse eines JTAPI-Calls
TelCtrl_Impl	Implementierung erweiterter Telefonfunktionen

CORBA-Interfaces

Schnittstelle	Datei	Beschreibung
BasicTelCtrl	TelControl.idl	Definition der grundlegenden Telefondienste
TelCtrl	TelControl.idl	Definition erweiterter Telefondienste

Exceptions

Exception	Beschreibung
InvalidStateException	Ungültiger Zustand der Zustandsmaschine
CTISystemException	Fehlermeldung des JTAPI-Systems

13.3.2 CallData

Klasse	Beschreibung
CallObject_Impl	Implementierung der Zugangsmethoden zu Basisattributen des Anrufobjekts
CallObjectFactory_Impl	Implementierung der Zugangsmethoden zu individuellen Attributen des Anrufobjekts.
CallObjectServer	Server, der die Implementierung der Factory instanziert und bei CORBA registriert
CustomCallObject_Impl	Implementierung der Zugangsmethoden zu individuellen Attributen des Anrufobjekts.
Worker	Thread-Klasse für den asynchronen Aufruf der Report-Funktion. Wird für Langzeitbetrachtungen verwendet, bei denen die Factory in gleichmäßigen Abständen die Anzahl der verwalteten Objekte sowie die Speichernutzung aufzeichnet.

CORBA-Interface

Schnittstelle	Datei	Beschreibung
CallObject	CallData.idl	Definition der grundlegenden Attribute eines Anrufobjekts
CustomCallObject	CallData.idl	Definiert individuelle Attribute des Anrufobjekts
CallObjectFactory	CallData.idl	Schnittstelle der Factory

Exceptions

Exception	Beschreibung
ObjectOccupiedException	Zeigt an, dass das Objekt bereits zur Bearbeitung gesperrt ist

13.3.3 StateMgr

Klasse	Beschreibung
StateMgr_Server	Server, der die Implementierung des StateMgr instanziert und bei CORBA registriert
StateMgr_Impl	Implementierung der StateMgr-Schnittstelle
StateMachine	Beinhaltet den eigentlichen Zustandsautomaten
Event.Supplier	Erzeugerobjekt zur Übermittlung einer Zustandsänderung über den CORBA-EventService

CORBA-Interface

Schnittstelle	Datei	Beschreibung
StateMgr	StateMgr.idl	Schnittstelle des zentralen Zustandsautomaten

Exceptions

Exception	Beschreibung
IllegalInputExceptionn	Zeigt an, dass die Eingabe im aktuellen Zustand des Automaten ungültig ist

13.3.4 Client

Klasse	Beschreibung
Main	Hauptprogramm des Clients
MainFrame	Graphische Oberfläche des Clients mit gesamter GUI-Funktionalität
ConsumerServer	Serverklasse zur Instanzierung und Registrierung des Verbraucherobjekts
Consumer	Verbraucherklasse zur Verarbeitung eingehender CORBA-Ereignisse
State	Hilfsklasse für die Ausgabe von Zuständen in Textform

13.4 Testergebnisse

Die Zeitmessungen der drei unterschiedlichen Szenarien für Lasttests sind nachfolgend aufgeführt. Dabei gibt der Parameter „Wartezeit" die Pause zwischen dem Setzen und Abfragen der Daten an, der Parameter „Range" den Bereich, in dem sich die zufälligen Abweichungen bewegen sowie „Zyklen" die Anzahl der durchlaufenen Zyklen an.

Die Ergebnisse der Tests bzgl. des Speicherbedarfs der Factory sind zu umfangreich, als dass ein Aufführen aller einzelnen Messergebnisse an dieser Stelle sinnvoll erscheint.

13.4.1 Szenario I

Ziel: hohe Last durch starke Parallelität erzeugen
Wartezeit 1500
Range 0
Zyklen 1
3 Rechner

		Millisekunden		
Host	Anzahl Clients	Zeit 1	Zeit 2	Zeit gesamt
Io1	5	58	24	82
Vlad	5	30	10	40
Sam	5	24	14	38
Io1	10	25	25	50
Vlad	10	8	12	20
Sam	10	10	7	17
Io1	15	18	20	38
Vlad	15	8	8	17

Sam	15	10	18	29
Io1	25	42	90	132
Vlad	25	22	34	56
Sam	25	23	26	50
Io1	50	69	101	171
Vlad	50	24	54	79
Sam	50	81	128	210
Io1	75	254	436	690
Vlad	75	33	94	128
Sam	75	150	297	447
Io1	100	221	236	457
Vlad	100	111	211	323
Sam	100	193	334	527
Io1	125	365	479	845
Vlad	125	139	252	391
Sam	125	261	562	823
Io1	150	828	1013	1842
Vlad	150	369	592	961
Sam	150	505	826	1331
Io1	175	941	1360	2302
Vlad	175	638	1136	1774
Sam	175	1016	1264	2280

13.4.2 Szenario II

Ziel: hohe Last durch starke Parallelität und viele Zugriffe erzeugen
Wartezeit 1500
Range 0
Zyklen 5
3 Rechner

Host	Anzahl Clients	Zeit 1	Zeit 2	Zeit gesamt
Io1	5	24	21	45
Vlad	5	12	6	18
Sam	5	19	17	36
Io1	10	19	23	43
Vlad	10	9	2	12
Sam	10	14	16	30
Io1	15	20	25	45
Vlad	15	16	14	30
Sam	15	12	19	31
Io1	25	35	38	74
Vlad	25	16	20	37
Sam	25	17	20	37
Io1	50	89	130	220
Vlad	50	117	156	273
Sam	50	86	122	209
Io1	75	235	348	584
Vlad	75	232	370	603
Sam	75	214	365	579
Io1	100	589	851	1440
Vlad	100	527	802	1330

Sam	100	698	822	1521
Io1	125	1012	1197	2210
Vlad	125	914	1306	2221
Sam	125	880	1277	2157
Io1	150	1198	1811	3009
Vlad	150	1091	1698	2790
Sam	150	1176	1794	2970
Io1	175	1739	2284	4023
Vlad	175	1635	2239	3875
Sam	175	1777	2300	4078

13.4.3 Szenario III

Ziel: realistische Last durch höhere Wartezeit und stärkere Varianz
Wartezeit 30000
Range 10000
Zyklen 1
3 Rechner

Host	Anzahl Clients	Zeit 1	Zeit 2	Zeit gesamt
Io1	5	6	8	14
Vlad	5	4	2	6
Sam	5	4	6	10
Io1	10	8	13	21
Vlad	10	4	3	7
Sam	10	8	2	10
Io1	15	12	8	20
Vlad	15	28	4	32
Sam	15	10	7	18
Io1	25	8	9	17
Vlad	25	5	4	10
Sam	25	8	5	13
Io1	50	24	11	35
Vlad	50	8	6	14
Sam	50	11	9	21
Io1	75	19	16	35
Vlad	75	40	9	49
Sam	75	31	10	41
Io1	100	29	14	44
Vlad	100	28	6	34
Sam	100	18	17	35
Io1	125	41	14	56
Vlad	125	35	8	43
Sam	125	30	17	47
Io1	150	76	21	98
Vlad	150	65	13	79
Sam	150	44	28	73
Io1	175	45	32	77
Vlad	175	41	12	54
Sam	175	44	33	77

HSSS AdminTools (c) 2001, last visited: Thu Feb 13 10:21:13 GMT+01:00 2003