DIPLOMARBEIT

THIN CLIENTS IN JWAM

Holger Bohlmann
Schulenbeksweg 7
20535 Hamburg

November 2000

Erstbetreuung: Prof. Dr. Heinz Züllighoven
Zweitbetreuung: Prof. Dr. Winfried Lamersdorf

Fachbereich Informatik
Arbeitsbereich Softwaretechnik
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

Erklärung:

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

Hamburg, den

Holger Bohlmann
Schulenbeksweg 7
20535 Hamburg
Tel.: 040 / 29823357 o. 0170 / 9300240
E-Mail: 3bohlman@informatik.uni-hamburg.de
Matrikel-Nr: 4627320

Betreuung:
Prof. Dr. Heinz Züllighoven (Erstbetreuer)
Prof. Dr. Winfried Lamersdorf (Zweitbetreuer)

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

Prof. Dr. Winfried Lamersdorf
Arbeitsbereich Verteilte Systeme (VSYS)
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

My name is Wolf. I solve problems. I heard, you have a problem?

(Harvey Keitel als Winston Wolf in „Pulp Fiction“)

1 Einleitung

„Ein Anwendungssystem soll möglichst von überall her nutzbar sein.“

Ein Anwendungssystem ist dazu da, einen ganzen Aufgabenbereich abzudecken, wie zum Beispiel die Warenwirtschaft eines Buchhändlers. Eine Vielzahl von Benutzern erledigt mit einzelnen Anwendungen des Anwendungssystems die unterschiedlichen Aufgaben. Solche Systeme sind Client/Server-Systeme (kurz C/S-Systeme): Es gibt eine zentrale Datenhaltung, auf die verschiedene Anwendungen zugreifen. Diese Anwendungen, wie z.B. das Bestellwesen oder die Lagerhaltung, können selbst über mehrere Rechner verteilt sein. Jeder Benutzer verfügt auf seinem Rechner, dem Benutzerrechner, über eine Software, mit der eine Anwendung ausgeführt werden kann.

Bei einer Client/Server-Architektur ist die Software für das gesamte Anwendungssystem auf mehrere Rechner verteilt: Ein Teil befindet sich auf einem Server und bietet für die Clients zentral den Dienst an, die Aufgaben des Anwendungssystemes zu erledigen. Durch die Aufteilung in Client und Server wird das Anwendungssystem ‚geschnitten‘. Für diese Aufteilung gibt es viele technische und fachliche Schnittmöglichkeiten, wobei ein Anwendungssystem auch mehrfach geschnitten werden kann. Die Präsentation einer Anwendung auf dem Bildschirm muß aber mindestens auf dem Benutzerrechner vorhanden sein, schließlich müssen Informationen dort angezeigt, eingegeben und manipuliert werden.

Wie kann ein solches System konkret aussehen? Bleiben wir beim Beispiel der Warenwirtschaft eines Buchhändlers. Innerhalb seiner Organisation kann für die Benutzerrechner eine einheitliche Infrastruktur vorherrschen: Gleiche Rechner mit einem gleichen Betriebssystem, wobei die Rechner über ein schnelles lokales Netz angeschlossen sind. Die Systeme werden zentral verwaltet, auf den Benutzerrechnern kann die Software der einzelnen Anwendungen ohne Probleme aufgespielt und gewartet werden. Die Anzahl der Benutzerrechner ist überschaubar. Es gibt einen Server, der für die Anwendungen auf den Benutzerrechnern die zentralen Aufgaben des Warenwirtschaftssystemes erledigt.

Doch nun möchte der Buchhändler sein Anwendungssystem öffnen, ein Zugang über das Internet soll möglich sein. Bestellungen sollen online abgewickelt werden können. Weiter möchte der Buchhändler wichtigen Kunden einen direkten Zugang zu seinem System bieten können, damit diese auch über das Internet mit dem Buchhändler Geschäfte abwickeln können. Für beide Fälle müssen Anwendungen entwickelt werden, die auch über das Internet zu bedienen sind. Der Zugang muß mit vorhandenen Technologien wie HTML, Webservern oder Java-Applets realisiert werden, damit von einem beliebigen Benutzerrechner nun auf das Anwendungssystem zugegriffen werden kann. Eine Einschränkung der Zugangsmöglichkeit bedeutet den Verlust von Benutzerkreisen und damit Kunden.

Für die Bestellung von Büchern über das Internet macht es Sinn, eine eigene Benutzerschnittstelle zu entwickeln, falls es für diesen Vorgang noch kein geeignetes Softwarewerkzeug gibt. Anderen wichtigen Kunden möchte der Buchhändler einen Zugriff über bewährte Anwendungen geben, was aber viele Probleme aufwirft: Läuft die Anwendung

überhaupt außerhalb der Organisation des Buchhändlers auf anderen Rechnern, ist die Verbindung schnell genug oder wie kann eine neue Version auf den entfernten Benutzerrechner aufgespielt werden?

Eine vorhandene Anwendungssoftware kann meist nicht ‚einfach so' verteilt werden, da sie für eine spezielle, lokale Infrastruktur entwickelt wurde. Die Software muß neu entwickelt, auf das Anwendungssystem muß über eine neue Schnittstelle zugegriffen werden können. Einen Ausweg aus dieser Situation versprechen **Thin Clients**: Auf dem Benutzerrechner wird nicht die Software einer Anwendung verteilt, sondern es wird nur die Präsentation einer Anwendung dargestellt. Da sich weniger Software auf dem Benutzerrechner befindet, wird hier der Begriff ‚Thin' verwendet. Im Gegensatz zu Hostsystemen oder der Verteilung von Zeichenbefehlen wie bei einem X-System [Barton 94], wird nicht die reine grafische Bildschirmdarstellung der Präsentation auf den Benutzerrechner gebracht, sondern es werden die in einer Anwendung verwendeten Oberflächenelemente auf den Benutzerrechner verteilt: die GUI der Anwendung.

Ein solcher Schnitt zwischen Benutzerrechner und Server bringt Vorteile: Da nur die Präsentation einer Anwendung auf dem Benutzerrechner abläuft, kann diese Anwendung nun einfacher von überall her benutzt werden. Die Anforderungen an Benutzerrechner und die Netzanbindung sinken. Das bisher Entwickelte kann dabei erhalten bleiben, da es auch durchaus möglich ist, existierende Anwendungen als Thin Clients zu realisieren, ohne daß es zu Änderungen am Anwendungscode kommen muß. Anwendungen selbst können nach dem WAM-Ansatz entwickelt werden [Züllighoven 98]. Für die Konstruktion von Anwendungen steht das JWAM Framework zur Verfügung [JWAM], mit dessen Hilfe sich Anwendungen nach dem WAM-Ansatz in Java entwickeln lassen.

In dieser Diplomarbeit stelle ich zwei Technologien vor, die es ermöglichen, einen Thin Client zu konstruieren. Eine Technologie ist der ‚Ultra Light Client', kurz ULC [Canoo 00]. Dort wird eine GUI-Bibliothek zur Verfügung gestellt, die von einer Anwendung benutzt wird und mit der sich dann diese Anwendung als Thin Client auf beliebige Benutzerrechner verteilen läßt. ULC läßt sich als Technologie in das JWAM Framework integrieren, so daß Anwendungen in JWAM auch als Thin Client verteilt werden können. Um das zu ermöglichen, sind aber auch Änderungen und Erweiterungen am Framework selbst notwendig, die ich in der Diplomarbeit erläutere.

Die zweite Technologie, um Thin Clients zu realisieren, ist eine Eigenentwicklung auf Basis des JWAM Frameworks. Da dieser Thin Client auf dem JWAM Framework aufbaut, können nur Anwendungen, die mit JWAM entwickelt wurden, als Thin Clients verteilt werden. Beide Thin Clients haben eine Gemeinsamkeit: Ohne Änderungen in der Anwendungslogik können die Anwendungen verteilt werden. Bestehende Anwendungen können als Thin Clients verteilt werden, und damit ist das gesamte Anwendungssystem dann von überall her nutzbar.

1.1 Überblick

Die Einleitung sollte schon einen Überblick über die Diplomarbeit geben, jedoch stelle ich hier kurz noch die Kapitel der Diplomarbeit und ihren Zusammenhang vor. An den Anfängen der Kapitel befindet sich jeweils noch eine kleine Erläuterung.

Kapitel 2 dient der Begriffsbestimmung und Motivation. Ich beschreibe grundlegend , was eine Client/Server-Verteilung ausmacht und was für Anforderungen an ein solches System gestellt werden können. Dabei betrachte ich nur das System auf den Rechnern der Benutzers und eines Serverrechners, der diese Rechner bedient. Ich stelle konkrete Systeme dafür vor, wobei ich Stärken und Schwächen aufzeige.

In **Kapitel 3** stelle ich dann ein Anwendungssystem vor, bei dem verschiedene Technologien aus Kapitel 2 verwendet wurden: den Helpdesk. Ich zeige auf, warum diese Technologien verwendet wurden und welchen spezifischen Nutzen diese für die Benutzer des Anwendungssystemes haben, wobei ich auch den JWAM Thin Client vorstelle.

In **Kapitel 4** gehe ich dann auf die grundlegende Architektur eines Thin Clients ein. Ich betrachte, wie gut ein Thin Client die Anforderungen eines Client/Server-Systems erfüllen kann. Weiter stelle ich den Ultra Light Client (ULC) vor und beschreibe, wie ich diese Technologie in das JWAM Framework integriert habe.

Die technische Konstruktion des JWAM Thin Clients beschreibe ich in **Kapitel 5**, wobei ich auch einige ungelöste Problem der Integration von ULC und JWAM aus Kapitel 4 aufgreife und die Lösung der technischen Probleme darstelle.

In **Kapitel 6** stelle ich dann die Ergebnisse vergleichend dar und bewerte die Möglichkeiten eines Thin Clients.

1.2 Konventionen & Softwareversionen

Die in dieser Arbeit verwendeten Interaktions- und Klassendiagramme lehnen sich an die UML-Spezifikation [JBR 99] an. Sofern es Abweichungen dazu gibt, sind diese in den Diagrammen erläutert. Wird Softwarecode beschrieben oder sich im Text auf Klassen oder Methoden bezogen, so wird der Zeichensatz `Courier New` zur Markierung verwendet. Methoden werden im Text weiter mit einer Klammer gekennzeichnet, z.B. `add()`. Erstmalig im Text genannte, wichtige Namen und Begriffe werden im laufenden Text durch eine fette Schreibweise gekennzeichnet, wie z.B. **Frontends**. Redewendungen, die in einem besonderen Zusammenhang verwendet werden, werden durch ‚Apostrophe' gekennzeichnet. Wörter wurden *kursiv geschrieben*, wenn diese besonders betont werden sollen.

Das vorgestellte Anwendungssystem Helpdesk und der JWAM Thin Client sind auf der JWAM Version 1.5 aufgebaut. Die Integration von ULC in JWAM beruht auf der JWAM Version 1.3.

Diese Diplomarbeit basiert auf der alten Rechtschreibung.

1.3 Danksagung

Für die freundliche Unterstützung zur Erstellung dieser Diplomarbeit danke allen, die mir dabei geholfen haben. Ich danke Prof. Dr. Heinz Züllighoven für die Erstbetreuung dieser Arbeit und für seine Anregungen und Hinweise. Bei Prof. Dr. Winfried Lamersdorf möchte ich mich für die Zweitbetreuung bedanken.

Dank auch insbesondere an Henning Wolf für seine Hilfe uns sein Feedback und an Martin Lippert für seine guten Einfälle, sowie an die anderen Mitglieder der JWAM-Architekturgruppe: Stefan Roock, meinen Namensvetter Holger Breitling, Andreas Havenstein, Robert F. Beeger und all die anderen, die ab und zu mal dabei waren, insbesondere Christian Beis. Besonderen Dank natürlich auch an Michael Otto und Norbert Schuler für die Zusammenarbeit in der Arbeitsgruppe Helpdesk, sowie an Marco Zühlke. Großen Dank auch an Bruno Schäffer, Elisabeth Maier, Guido Hächler, Thomas Ernst und all die anderen Mitarbeiter aus dem AEC Basel (bzw. jetzt bei Canoo Engineering) für die Unterstützung in Sachen ULC und für den keinen Einblick ins Schwitzerdütsch.

2 C/S-Konstruktion und Verteilungsprobleme

In diesem Kapitel erläutere ich allgemeine Probleme bei der Client/Server-Verteilung von Software, um einen Überblick zu verschaffen. Ich betrachte dabei ausschließlich die Verteilung zwischen dem Rechner des Benutzers und dem dahinter liegenden Server. Einen Schwerpunkt bei dieser Betrachtung wird die Problematik des Softwaredeployments sein: Welcher Teil einer Anwendung muß auf dem Benutzerrechner ablaufen und wie wird dieser Teil auf den Benutzerrechner installiert? Weiter stelle ich einige Forderungen auf, die eine Software erfüllen muß, damit sie optimal auf Benutzerrechner verteilt werden kann. Anhand dieser Forderungen stelle ich dann existierende Lösungen auf Benutzerrechnern vor, zeige Stärken und Schwächen auf und vergleiche sie abschließend miteinander.

2.1 C/S-Schnittmöglichkeiten

Ein Anwendungssystem besteht aus mehreren Anwendungen, mit denen Benutzer von ihren Rechnern aus, den Benutzerrechnern, verschiedene Aufgaben erfüllen. Eine laufende Anwendung eines Benutzers arbeitet auf einem Datenbestand, den sie mit anderen laufenden Anwendungen weiterer Benutzer teilen muß. Das Anwendungssystem hat somit einen Datenbestand (eine Ressource), der von allen Anwendungen gemeinsamen genutzt wird. Ein solches Anwendungssystem ist daher immer als C/S-System realisiert: Es muß mindestens einen zentralen Server geben, der einen zentralen Zugriff auf diesen gemeinsamen Datenbestand ermöglicht. Er bietet diesen Zugriff als Dienst an, der von den Clients genutzt wird, was beispielsweise direkt die Software der Anwendungen sein kann, die auf den Benutzerrechnern installiert ist. Damit besteht ein Anwendungssystem auch aus mehren Teilen von Anwendungen: Es gibt einmal eine Serveranwendung, die einen Dienst für Clientanwendungen anbietet. Die Anwendungsteile laufen auf unterschiedlichen Rechnern ab.

Aus der Sicht des Softwareentwicklers stellt sich die Frage, wie die Anwendungen eines Anwendungssystem geschnitten werden können, um eine sinnvolle und effektive Verteilung zu ermöglichen. Welcher Teil eines Anwendungssystems befindet sich auf einem Server, welcher Teil befindet sich auf den Benutzerrechner?

Ein Anwendungssystem läßt sich dazu grob in drei logische Einheiten einteilen [CTRC 95]: **Presentation Logic**, **Application Logic** und **Data Management**. Das Data Management ist für die persistente Sicherung und die Bereitstellung der Daten zuständig (z.B. eine RDBMS). Die Application-Logic-Schicht bearbeitet diese Daten und besitzt fachliches Wissen der Anwendungen. Die Präsentationsschicht ist die Darstellungsebene, von der sich zumindest ein Teil auf den Benutzerrechner befinden *muß*, nämlich die GUI. Weiter dient die Präsentationsschicht der Aufbereitung von Daten aus der Applikationslogik zwecks Darstellung. Der Teil der Anwendung auf dem Benutzerrechner wird auch als **Frontend** bezeichnet. Es ist der Teil der Anwendung, mit der ein Benutzer ‚am Ende' arbeitet.

Diese logischen Einheiten eines Anwendungssystemes können sich auf unterschiedlichen Rechnern befinden, was unterschiedliche Schnittmöglichkeiten bietet. Auch Schnitte durch die logischen Einheiten sind denkbar, da dieses durchaus mehrere Bestandteile umfassen können:

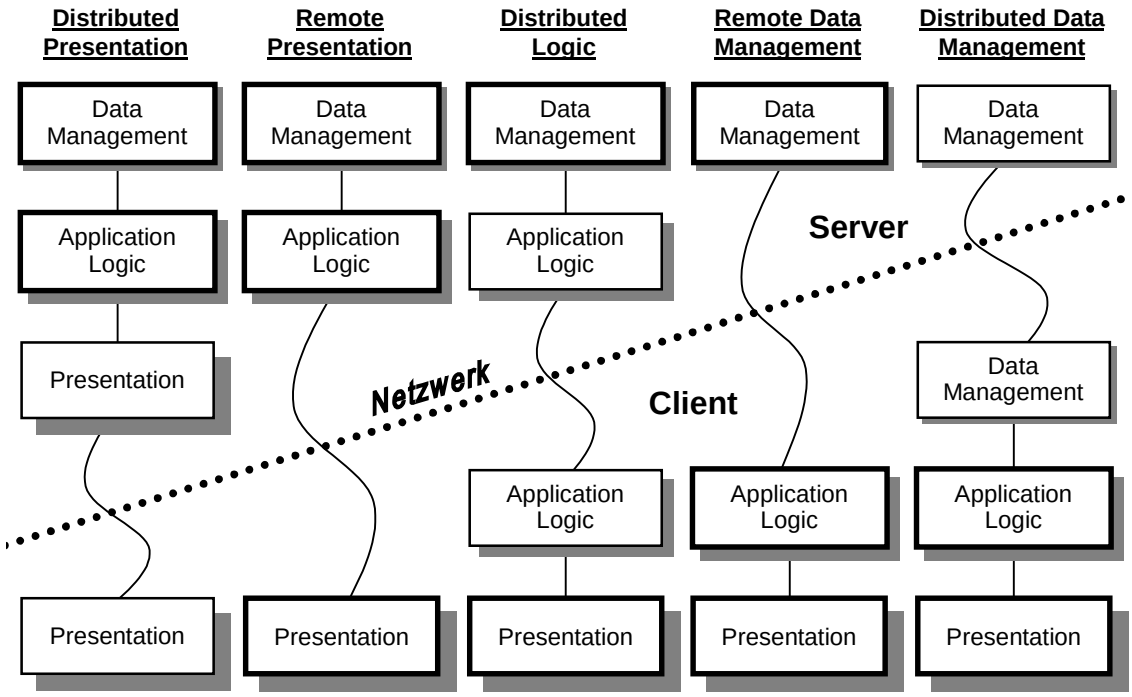

Abbildung 2-1: Schnitte zwischen logischen Einheiten.
Quelle: Aus [CTRC 95], Original Gartner Group

Ein Anwendungssystem kann auch mehrere solcher Schnitte aufweisen, wozu ich aber später noch kommen werden. Jede Schnittart ist unterschiedlich motiviert und bringt verschiedene Vor- und Nachteile mit sich:

- **Distributed Data Management:** Eine Schnittmöglichkeit, die beispielsweise bei verteilten Datenbanksystemen angewandt wird. Durch die Verteilung der Datenspeicherung wird hauptsächlich eine Lastverteilung durchgeführt, was bei stark ausgelasteten Datenbanken sinnvoll ist. [Bell/Grimson 92]

- **Remote Data Management:** Bei diesem Schnitt wird zwischen dem Datenmanagement und der Applikationslogik geschnitten. Ein Beispiel wäre eine Applikation, die auf eine entfernte Datenbank zugreift. Dieser Schnitt wird bei vielen existierenden Systemen angesetzt, bei denen direkt auf einer Datenbank gearbeitet wird[1]. Auf dem Client werden komplett die Daten der Anwendung verarbeitet, während der Server das reine Management der Datenhaltung regelt. Diese Schnittart wird häufig verwendet, denn bei C/S-Anwendungen müssen Daten zentral in einer Datenbanken gehalten werden. Beispielhaft sind hier die oft verwendeten relationalen Datenbanken, die über die einheitliche Datenbanksprache SQL (Standard Query Language) zu bedienen sind [Orfali 99]. An SQL-Datenbanken können SQL-Befehle direkt von einer Anwendung aus gesendet und deren Resultate verarbeitet

[1] Diese Schnittart wird oft bei dem ‚Two-Tier'-Ansatz verwendet. Siehe Kapitel 2.1.2

werden. Ein Zugang zur Datenbank ist somit leicht möglich, so z.B. bei JDBC für die Programmiersprache Java. [Reese 00][Orfali 99]

- **Distributed Logic:** Hier wird in der Applikationslogik selbst geschnitten, die sich damit auf unterschiedlichen Rechnern befinden kann. Der Schnitt teilt die Applikationslogik beispielsweise so auf, daß auf einem Server nicht nur ein Dienst zum Zugriff auf reine Daten angeboten wird, wie bei Remote Data Management, sondern weitergehende Dienste bereitgestellt werden. Diese Dienste besitzen Wissen über das Anwendungssystem selbst und bieten einen fachlichen Umgang mit dem Anwendungssystem an. Durch einen solchen Schnitt kann ein Anwendungssystem so zerteilt werden, daß Aufgaben, die in unterschiedlichen Teilen eines Anwendungssystemen immer wieder vorkommen, an einer Stelle zentral durch einen **Application Server** angeboten werden. Softwaretechnisch gesehen bedeutet eine verteilte Applikationslogik auch eine Abstraktion von der Datenbank, auf der nicht mehr von anderen Teilen der Anwendung aus direkt zugegriffen wird. Vorteile sind eine bessere Wartung und Skalierbarkeit der Software [Orfali 99].

- **Remote Presentation:** Bei dieser Aufteilung befindet sich keinerlei Wissen über die eigentliche Anwendung in Form von Software auf dem Client, nur der Umgang mit den Oberflächenelementen und deren Handhabung ist auf dem Client als Software vorhanden. Ein Beispiel wäre hier eine sehr leichtgewichtige[2] Anwendung, die sich nur um die Aufbereitung und Darstellung der Daten kümmert, die sie von einem Server erfragt.

- **Distributed Presentation:** Bei diesem Schnitt befindet sich nur die reine Präsentation in Form einer GUI auf dem Benutzerrechner, der gesamte Rest der Anwendung läuft auf einem Server ab. Auf dem Benutzerrechner ist kein anwendungsspezifischer Code installiert, der Benutzerrechner ist hier nur für die Anzeige und Verwaltung von allgemeinen, grafischen Elementen verantwortlich. Diese Form der Verteilung findet man auch bei Internet-anwendungen vor, die über einen HTML-Browser benutzt werden. Hostsysteme, die über Terminals benutzt werden, arbeiten ebenfalls nach diesem Prinzip.

Als Schnittmöglichkeiten zwischen Benutzerrechner und Server kommen nur die letzten vier der eben genannten Optionen in Frage. Die Datenhaltung befindet sich bei C/S-Systemen zentral auf einem Server. Die Schnittmöglichkeit des Distributed Data Management ist nur für den Fall interessant, wenn die Datenhaltung zwecks Skalierung und Steigerung der Performance auf mehrere Rechner verteilt wird. Der Benutzerrechner kommt dabei nicht ins Spiel. Auf die anderen Systeme und zugehörige Beispiele werde ich noch eingehen.

2.1.1 Fat Client und Fat Servers

Zwischen Präsentation und Datenhaltung befindet sich der Softwarecode der Anwendung, der sich bei einer C/S-Anwendung zu einem Teil auf dem Server, zu anderem Teil auch auf dem Client befinden kann.

[2] Leichtgewichtig im Sinne von ‚wenig Softwarecode vorhanden'

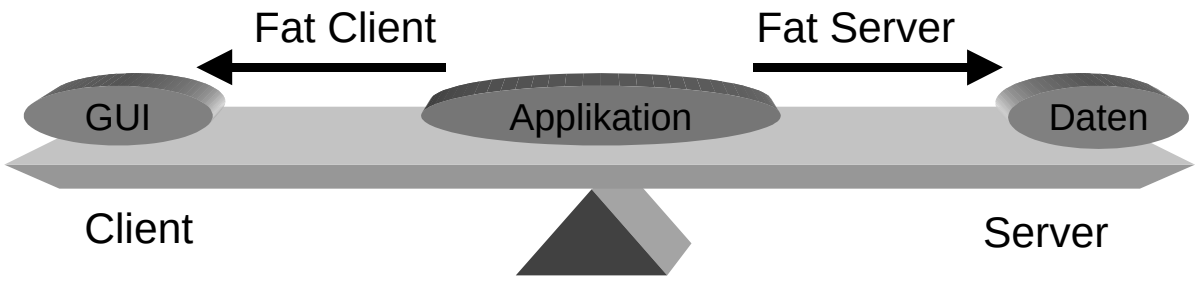

Abbildung 2-2: Fat und Thin Client

Je nach Menge des Applikationscodes auf einem Rechner spricht man von einem **Fat Client** und einem **Thin Server**, analog dazu **Thin Client** oder **Fat Server** [Orfali 99]. Je mehr Softwarecode vorhanden ist, desto mehr Aufgaben muß der entsprechende Rechner bewältigen. Bei einem Fat Server, auf dem ein Großteil der Anwendung läuft, kommt es daher bei vielen Clients schneller zu einem Lastproblem als bei einem Thin Server. Durch eine Verteilung der Aufgaben über mehrere Rechner kann dieses Problem gelöst werden.

Je ‚fetter' der Server ist, desto anwendungsfachlicher kann die Schnittstelle zu der Anwendung auf dem Client formuliert werden, was zu einer Reduktion des Datenverkehrs zwischen Client und Server führt. Arbeitet ein Fat Client beispielsweise direkt auf einer Datenbank und müssen durch eine Aktion des Benutzers mehrere Tabellen verändert werden, so kommt es über das Netz zu mehreren Abfragen an die Datenbank. Ein Fat Server hingegen kann dem Client die Bearbeitung der Anfrage durch einen Dienst zur Verfügung stellen. Es kommt nur zu einem Aufruf dieses Dienstes, der Netzwerkverkehr wird minimiert.

Je mehr Softwarecode sich auf dem Client befindet, desto öfter müssen Änderungen an der Anwendung auch auf dem Client nachgezogen werden. Dient der Server zum Beispiel nur der reinen Datenhaltung wie bei der Schnittart "Remote Data Management" und kommt es zu Änderungen an der Datenstruktur, so muß die Software auf den Benutzerrechnern daran angepaßt werden. Sofern der Softwarecode auf dem Client nicht automatisch aktualisiert wird, kann das zu einem Problem werden. Es ist unproblematisch, ein Update auf einem Serverrechner durchzuführen, da es nur einen oder wenige Server gibt, während die Anzahl der Benutzerrechner oft ein vielfaches davon ist. Bei Internet-Anwendungen geht die Zahl der möglichen Benutzerrechner sogar in die Millionen. Die maximale Anzahl wird theoretisch nur durch die Größe der IP-Nummernraumes begrenzt.

Natürlich arbeiten nicht alle diese Rechner gleichzeitig mit einem Server, aber eine Internet-Anwendung sollte von jedem dieser Rechner aufgerufen werden können. In diesem Fall ergeben sich besondere Anforderungen: Auf dem Client-Rechnern muß eine Standardsoftware vorhanden sein, die keinen Applikationscode enthält, um nicht in ein Update-Problem zu kommen. Andererseits muß ein Server viele Rechner gleichzeitig bedienen und kann daher nicht viel Applikationscode ausführen. Auf diesen Widerspruch werde ich später noch eingehen.

2.1.2 Mehrfachschnitt: 2-,3- und N-Tier

Schnitte durch die unterschiedlichen Schichten können nicht nur einmal sondern auch mehrfach gezogen werden. Je nach Anzahl der Schnitte lassen sich C/S-Anwendungen in drei Kategorien aufteilen [Orfali 99]:

- **2-Tier:** Es gibt nur einen Schnitt und es existieren nur zwei Schichten: die auf dem Benutzerrechner und die auf dem Server. Eine klassische Aufteilung wäre hier eine komplette Anwendung auf dem Benutzerrechner (Fat Client), die mit einer zentralen Datenbank (Thin Server)[3] zusammenarbeitet. Die zweischichtige Aufteilung bringt viele Nachteile mit sich, obwohl sie in existierenden Anwendungssystemen noch häufig vorkommt[4]: Sie ist schwer zu administrieren, skaliert schlecht und eine Anbindung außerhalb des bisherigen Bereiches des Anwendung (z.B. aus einem Unternehmen ins Internet) ist schwer möglich. Dagegen steht die schnelle Entwicklung einer solchen Anwendung, die aber durch die mangelnde Möglichkeit der Wiederverwendung bald an ihre Grenzen stößt.

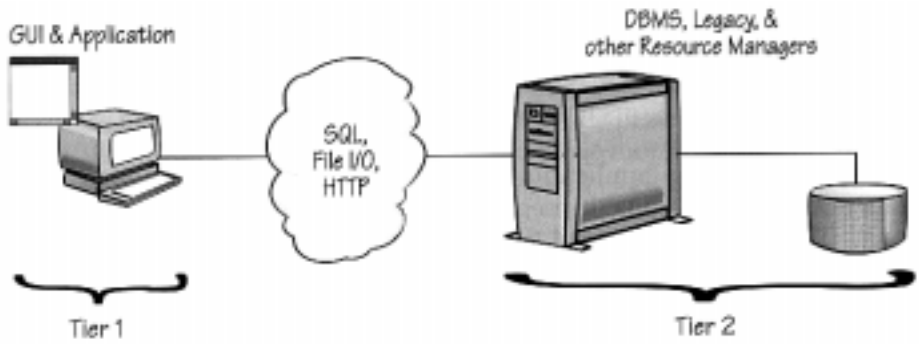

Abbildung 2-3: 2-Tier C/S-Architekturen, Quelle [Orfali 99]

- **3-Tier:** Eine bessere Architektur verspricht die Aufteilung in drei Schichten durch Einführung einer Applikationsschicht. Dazu wird das Anwendungssystem genau wie bei dem vorigen Modell (Vergleiche Abbildung 2-2 und 2-4) geschnitten: Es gibt einen Datenbankserver, einen Applikationsserver und einen Benutzerrechner, der nur für die Präsentation zuständig ist. Ein Schnitt erfolgt entlang der Schichtgrenzen. Diese Aufteilung bringt im Vergleich zur 2-Tier Architektur einige Vorteile mit sich [Orfali 99]:
  - Bessere Administration der Benutzerrechner durch Thin Clients mit Standardsoftware
  - Größere Skalierung durch Lastverteilung in der mittleren Schicht möglich
  - Bessere Wiederverwertbarkeit durch Abstraktion
  - Anbindung an Altsysteme durch Kapselung von Anwendungslogik in der mittleren Schicht
  - Mögliche Anbindung des Anwendungssystems an unterschiedliche Frontends

  Der Entwicklungsaufwand für ein 3-Tier Anwendungssystem ist größer als für ein 2-Tier System. Daher sind schnelle, prototypische Lösungen schlechter zu realisieren.

[3] Eine Datenbank ist unter dem Gesichtspunkt des darin enthalten Wissens über die Anwendung ‚dünn'. Die Datenbank an sich kann aber durchaus ein großes Stück Software sein.
[4] Eine Untersuchung der Gartner Group hat ermittelt, daß im Jahr 1998 ca. 2/3 aller existierenden C/S-Anwendungen auf dem 2-Tier Modell basieren [Orfali 99].

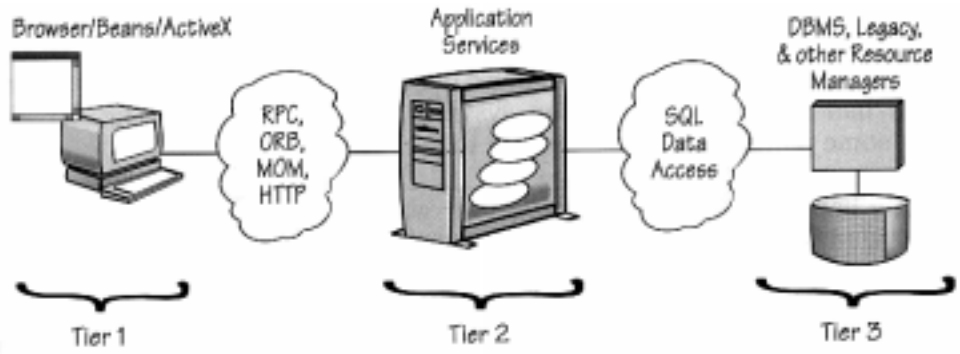

Abbildung 2-4: 3-Tier C/S-Architekutren, Quelle [Orfali 99]

- **N-Tier:** Betrachtet man ein gesamtes Anwendungssystem, so können dort abgeschlossene Aufgabengebiete gefunden werden. Diese können in einer Komponente gekapselt werden, die als Dienst eine Schnittstelle zur Bearbeitung ihres Aufgabengebietes bereitstellt. Diese Komponenten können auf unterschiedliche Applikationsserver verteilt werden (siehe "Distributed Logik"). Hier spricht man dann von einer N-Tier Architektur[5]. Durch die Aufteilung einer Anwendung in mehrere fachliche Komponenten erhofft man sich eine bessere softwaretechnische Konstruktion. Komponenten können einzeln entwickelt, geändert oder hinzugefügt werden, ohne das gesamte System verändern zu müssen. Weiter können Komponenten schrittweise entwickelt werden. Der Grad der Wiederverwendung ist noch weiter erhöht.

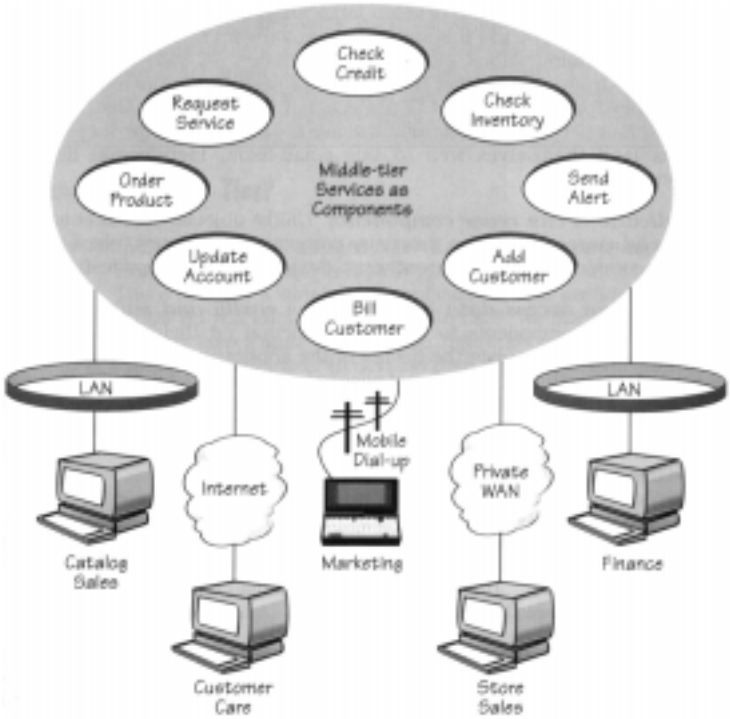

Abbildung 2-5: N-Tier C/S-Architekturen, Quelle [Orfali 99]

Während die 2-Tier Architektur schnell an ihre Grenzen stößt, bietet die 3-Tier oder N-Tier Architektur eine größere Flexibilität und Skalierbarkeit. Durch die Entwicklung von fachlichen Komponenten bei der N-Tier Architektur erreicht man softwaretechnisch einen hohen Grad der

[5] N-Tier ist auch unter dem Begriff Multi-Tier bekannt

Abstraktion und Wiederverwendung. Solche Komponenten werden als verteilte Objekte oder Gruppen von Objekten zur Verfügung gestellt, die über ihre Schnittstelle einen entsprechenden Dienst anbieten [Singh 99]. Die Implementierung der Komponenten ist dem Benutzer verborgen, ein Zugriff findet immer über eine definierte Schnittstelle statt. Ein ähnlicher Ansatz aus der WAM-Welt sind **fachliche Services**, die auch für das Anwendungsbeispiel in dieser Diplomarbeit benutzt werden [Otto/Schuler 00].

2.2 Anforderungen an einem C/S-Schnitt

Wie eine C/S-Anwendung im Detail auch geschnitten sein mag, es muß immer ein Teil auf dem Benutzerrechner vorhanden sein. Doch wie soll das Frontend für den Benutzer gestaltet sein? Welche grundsätzlichen Anforderungen gibt es an ein Frontend? Eine verteilte Anwendung soll mehrere Merkmale erfüllen: hohe Verfügbarkeit, Sicherheit der Übertragungen und Authentifikationsmechanismen, Skalierbarkeit, Performance, einfache Benutzbarkeit, Transparenz der entfernten Ressourcen [Singh 99]. Für den Schnitt zwischen Benutzerrechner und Server betrachte ich einige dieser Kriterien, die für die Konstruktion der Software relevant sind, und füge solche Kriterien hinzu, die ich für diesen Schnitt als wichtig erachte.

2.2.1 Softwaredeployment

Eine wichtige Anforderung an die Software des Anwendungssystem lautet: Auf den Benutzerrechnern muß die Software zur Benutzung eines Anwendungssystemes einfach zu installieren sein und ein Update der Software muß ebenso einfach möglich sein. Um das zu realisieren, muß man die Konfiguration eines Benutzerrechners genauer betrachten:

- **Lage des Rechners für die Installation (Lokalität):** Der Benutzerrechner, auf dem die Anwendung läuft, kann im Einflußbereich eines Administrators liegen, wie in einem lokalem Netzwerk, oder außerhalb, wie im Internet. Liegt der Benutzerrechner in einem lokalen Netzwerk, so kann sich der Administrator um eine Installation von Software kümmern und Updates aufspielen. Die aktuelle Software für Benutzerrechner kann sogar zentral über ein verteiltes Filesystem zur Verfügung gestellt werden. Eine Fernwartung der Benutzerrechner ist durch Administrationstools möglich. Sofern aber ein Anwendungssystem über das Internet zu bedienen ist, kann es keine zentrale Administration der Benutzerrechner mehr geben. Software muß dann durch Herunterladen von HTTP- oder FTP-Servern auf Benutzerrechnern gebracht und dort installiert werden. Updates müssen entsprechend über eine erneute Installation auf den Rechner gebracht werden.

  Ein weiterer Punkt, der auch von der Lokalität des Benutzerrechners abhängt, ist die Erreichbarkeit des Benutzerrechners vom Server aus und umgekehrt. Zwischen Benutzerrechner und Server kann eine **Firewall** liegen [Orfali 99], die zur Erhöhung der Sicherheit nur bestimmte Arten der Kommunikation zuläßt. Firewalls werden in Verbindungen von internen Netzen zu öffentlichen Netzen (Internet) eingesetzt, um eine sicherheitskritische Kommunikation zu verhindern. Das technische Protokoll zwischen Benutzerrechner und Server sollte möglichst von Firewalls nicht eingeschränkt werden. Ein Beispiel hierzu ist das HTTP-Protokoll [FGMFB 97], welches in der Regel nicht von Firewalls behindert wird [Orfali 99]. WWW-Browsern könnten sonst wegen einer Firewall

nicht mehr benutzt werden, ein wichtiges Internet-Werkzeug wäre damit nicht mehr zu bedienen. Über dieses Protokoll können aber nicht nur HTML-Seiten übertragen werden, es kann auch dazu benutzt werden, um andere Daten auszutauschen und damit restriktive Firewalls zu umgehen. Dieses Verfahren wird **Tunneling** genannt (siehe z.B. Servlet-Tunneling in [Hunter/Crawford 98]).

- **Betriebssystem:** Auf jedem Rechner kann ein unterschiedliches Betriebssystem installiert sein. In einem abgeschotteten Bereich, z.B. einem Unternehmen, kann man zwar Standards formulieren. Soll aber eine Anwendung über das Internet funktionieren, können solche Standards nur durch den Ausschluß von Benutzerkreisen geschaffen werden. Eine Lösung ist dafür beispielsweise eine Java-Anwendungen oder eine WWW-Schnittstelle eines C/S-Anwendungssystemen, womit dieses über einen WWW-Browser zu bedienen ist[6]. Für alle Betriebssysteme von Desktop-Rechnern existieren mittlerweile WWW-Browser und virtuelle Maschinen für Java (Java VM), teilweise aber in unterschiedlichen Versionen [Javasoft 00].

- **Ausstattung:** Auf den Benutzerrechnern können verschiedene Komponenten installiert sein, auf die sich ein Frontend eventuell verläßt. Das kann bei betriebssystemspezifischen Anwendungen auftreten, aber auch bei WWW-Browsern oder Java-Applikationen: Es können unterschiedliche Plugins oder Pakete installiert sein. Muß für eine Software ein spezielles Paket vorhanden sein, so sollte dieses einfach auf den Benutzerrechner zu installieren sein, wie es z.B. bei Browser-Plugins der Fall ist.

Die Software zur Bedienung eines Anwendungssystemes sollte möglichst unabhängig von diesen drei Konfigurationskriterien eines Benutzerrechner sein. Erfüllt eine Software diese Kriterien, kann sie einfach auf beliebige Benutzerrechner verteilt werden.

Ein Beispiel dazu ist ein Anwendungssystem, das über einen WWW-Browser von dem Benutzerrechner aus bedient wird. Ein solcher Browser kann als Standardsoftware angesehen werden, der auf jedem Benutzerrechner vorhanden sein sollte. Mit Standardsoftware lassen sich viele verschiedene Aufgaben erfüllen, sie ist nicht speziell für ein Anwendungssystem entwickelt. Ein Administrator ‚vor Ort' kann sich um die Installation und Wartung dieser Standardsoftware kümmern. Sofern ein Browser auf dem Benutzerrechner installiert ist und keine speziellen Eigenschaften eines spezifischen Browsers genutzt werden (siehe Kapitel 2.3.1), kann damit das Anwendungssystem von überall her benutzt werden. Anwendungsspezifische Software muß erst gar nicht auf dem Benutzerrechner installiert werden. Dieses Kriterium der Konfiguration des Benutzerrechners hat in diesem Beispiel keine Bedeutung.

2.2.2 Schnittstellenänderungen

Eine Anwendung auf einem Client kommuniziert mit dem Server über eine Softwareschnittstelle und mittels eines Protokolls. Die dafür notwendige Software wird **Middleware** genannt. Sie bietet die Möglichkeit, Schnittstellen zwischen Rechnern zu definieren

[6] Durch Unterschiede in den aktuellen Versionen von virtuellen Maschinen (VM) auf den konkreten Systemen sind diese mehr oder mindert inkompatibel zu der Spezifikation. Auf allen Systemen sind nicht die gleichen Versionen von Java verfügbar [Javasoft 00]. Gleiches gilt für WWW-Browser.

und implementiert ein technisches Protokoll zum Informationsaustausch [Orfali 99]. Welche Methoden die Schnittstelle anbietet und was für Informationen zwischen Benutzerrechner und Server ausgetauscht werden können, wird durch die Anwendung auf dem Server bestimmt.

Kommt es zu Änderungen an der Serveranwendung und damit zu Änderungen an der Schnittstelle zu den Benutzerrechnern, müssen die Anwendungen auf den Benutzerrechnern eventuell an diese Änderungen angepaßt werden. Informationen können beispielsweise unter Umständen nicht mehr wie bisher abgefragt werden oder einzelne Methoden werden nicht mehr angeboten.

Damit müssen die Anwendungen auf den Benutzerrechnern auch an die neue Schnittstelle angepaßt und eine neue Software aufgespielt werden. Eine Serveranwendung ist schwieriger zu entwickeln, da durch eventuelle Veränderungen der Schnittstellen zu den Benutzerrechnern auch gleich die Anwendung auf den Benutzerrechnern angepaßt werden müssen. Kommt es nur zu einer Erweiterung der Schnittstelle, können unter Umständen die Anwendungen auf den Benutzerrechnern weiter mit der veränderten Serveranwendung arbeiten. Dieses Verhalten muß natürlich von der Middleware unterstützt werden.

Diese Abhängigkeit vermeidet man, wenn zwischen Benutzerrechner und Server Schnittstellen definiert sind, die nicht von der eigentlichen Anwendung abhängig sind, sondern generisch für alle möglichen Anwendungen funktionieren. Als Beispiel sei hier wieder der Fall des WWW-Browser genannt: Über Formulare (Forms) mit den dazu passenden standardisierten Formularelementen und einer CGI-Schnittstelle auf dem Server gibt es viele verschiedene Internetanwendungen, die auf der gleichen Technologie aufsetzten.

Um gegen Schnittstellenänderungen gefeit zu sein, bieten sich zwei Möglichkeiten an: Entweder ist der Softwarecode auf den Benutzerrechnern durch automatische Mechanismen immer aktuell, womit eine Schnittstellenänderung für den Benutzerrechner ohne weitere Konsequenzen beleibt, oder die Schnittstelle ist generisch, so daß eine Anwendung auf dem Benutzerrechner nicht verändert werden muß, wenn es zu Änderungen an der Serveranwendung kommt. Allerdings ist auch bei der automatischen Aktualisierung des Softwarecodes auf den Benutzerrechnern darauf zu achten, daß dieser Code an die Änderungen angepaßt werden muß. Der dafür nötige Aufwand sollte möglichst gering sein.

2.2.3 Skalierung, Lastverteilung & Performance

Die Anzahl der Anwender und der Benutzerrechner eines C/S-Anwendungssystemes ist nicht zwangsläufig konstant. Das ist für ein C/S-Anwendungssystem dann problematisch, wenn die Anzahl wächst. Die Anwendung auf dem Server muß dann mit der wachsenden Anzahl von Anwendern skalieren, das Hinzufügen von weiteren Benutzerrechnern sollte problemlos möglich sein. Es dürfen möglichst keine **Bottlenecks** vorhanden sein: Teile einer Serveranwendung, die von jedem aktiven Benutzerrechner gebraucht werden, die aber nicht mit einer wachsenden Anzahl von Benutzerrechnern entsprechend skalieren können. Diese Engpässe führen zu einem Verlust der Performance des gesamten Systems, weswegen Skalierung und Performance hier auch zusammen betrachtet werden.

Für die Gesamtperformance ist es sinnvoll, den Schnitt zwischen Benutzerrechner und Server so zu gestalten, daß die Anfragen über das Netz möglichst schnell beantwortet werden und die Übertragung möglichst wenig Zeit in Anspruch nimmt. Damit bleibt das System auch dann performant, wenn zwischen Benutzerrechner und Server nur ein langsames Netz verfügbar ist.

Sofern ein Netz am Benutzerrechner ausreichend schnell ist, zum Beispiel bei einer lokalen Ethernet-Verbindung mit 100 MBaud/s, ist nur die Antwortzeit des Servers von Bedeutung, nicht aber die Übertragungszeit. Im Internet hingegen kann die Geschwindigkeit einer Verbindung zwischen zwei beliebigen Rechnern sehr stark schwanken. Ein Benutzerrechner kann beispielsweise nur über eine Modemverbindung mit einigen dutzenden Kilobaud pro Sekunde mit dem Internet verbunden sein, was sich dann bei vielen Anfragen mit hohen Datenmengen schnell unangenehm bemerkbar machen würde.

2.2.4 Transparenz

Netzverbindungen sind prinzipiell fehlerbehaftet, so daß man in einer Anwendung bei entfernten Zugriffen mit Fehlern rechnen und dementsprechend eine Fehlerbehandlung durchführen muß. Je nach Art des Fehler kann die Anwendung eine erneute Übertragung probieren oder aber dem Benutzer melden, daß es Verbindungsprobleme gab und eine weitere Behandlung nicht möglich ist, da der Server nicht mehr zu erreichen ist.

Kommen in einer Anwendung Serverzugriffe häufig und an verschiedenen Stellen vor, so wird die Fehlerbehandlung umso aufwendiger. Ein allgemeines Fehlermanagement, das automatisch reagiert, wie z.B. der Versuch, die Verbindung wiederherzustellen und/oder die Benachrichtigung des Benutzers, würde die Konstruktion der Anwendung vereinfachen.

Wie ein Fehler zu bewerten ist, hängst stark von der einzelnen Situation ab. An kritischen Stellen im Ablauf einer Anwendung, z.B. bei einem Abschluß eines längeren Vorganges, möchte man unbedingt sicherstellen, daß eine Übertragung funktioniert hat. Im Fehlerfall sollte gegebenenfalls mittels einer eigenen Fehlerbehandlung darauf reagiert werden. Die Anwendung würde dann beispielsweise nach Rücksprache mit dem Benutzer abgebrochen oder zurückgesetzt werden, da ein sicherer Fortgang nicht mehr gewährleistet ist. Vom Benutzer eingegebene Daten könnten gesichert werden, so daß eine erneute Eingabe nicht mehr notwendig ist, falls die Verbindung wieder einwandfrei funktionieren sollte. Eine andere Situation wäre beispielsweise die Abfrage einer Übersichtstabelle zwecks Anzeige auf dem Benutzerrechner. Hier sollte ein Übertragungsfehler weniger schwerwiegende Auswirkungen haben. Es reicht unter Umständen eine einfache Benachrichtigung aus, daß die Tabelle nicht aktualisiert werden konnte. Eine Aktualisierung kann später durch den Benutzer über eine eigene Aktion, z.B. durch das Drücken eines Refresh-Buttons erfolgen. Der Ablauf der Anwendung sollte nicht unterbrochen werden.

Ein eigenes Fehlermanagement sollte also optional sein, ansonsten sollte es für den Anwendungsentwickler transparent sein[7], damit er sein Hauptaugenmerk auf die Entwicklung der Anwendung legen kann und nicht auf Entwicklungsprobleme mit der Middleware. Eine totale

[7] Transparent im Sinne von ‚nicht sichtbar‘

Verteilungstransparenz ist keine Lösung. Da bei verteilten Zugriffen immer Fehler entstehen können, sollte in der Anwendung auch Wissen darüber vorhanden sein, daß auf eine entfernte, verteilte Ressource zugegriffen wird.

Ein weiterer Punkt ist die Programmierung der Schnittstellen einer Anwendung zwischen Benutzerrechner und Server, die auch möglichst transparent sein muß. Der Softwarecode der Anwendung sollte möglichst wenig mit einer konkreten Middleware verbunden sein, da sonst die Gefahr besteht, zu sehr von der Middleware und damit von technischen Gegebenheiten abhängig zu sein. Die Middleware sollte so reichhaltige Möglichkeiten bieten, daß eine Anwendung nicht an die technischen Notwendigkeiten der Middleware angepaßt werden muß. Ansonsten besteht die Gefahr, daß das Protokoll zwischen Benutzerrechner und Server an die technischen Gegebenheiten der Middleware angepaßt werden muß. Außer über Fehler durch die Netzverbindung sollte der Anwendungsentwickler nichts über die konkrete technische Verteilung erfahren müssen, sondern Schnittstellen rein fachlich formulieren können.

2.3 Ansätze für C/S-System

Sofern die Software einer Anwendung auf den Benutzerrechner ohne Zutun des Benutzer aufgespielt werden kann, ist das Problem des Softwaredeployments und der Schnittstellen-änderung gelöst. Ob eine gute Skalierung, Performance und Transparenz erreicht werden kann, hängt jedoch von der konkreten Realisierung des Systems ab. In dem folgenden Kapitel möchte ich Systeme vorstellen, die das Problem des Deployments angehen oder durch ihre Konstruktion erst gar nicht besitzen: Es handelt sich um Thin Clients als Frontends, die entweder keinen anwendungsspezifischen Code besitzen oder den entsprechenden Softwarecode automatisch auf den Benutzerrechner bringen können. Die anderen, oben genannten Punkte werden unterschiedlich gut erfüllt, worauf ich dann im Einzelfall eingehen werde.

2.3.1 WWW-Browser

WWW-Anwendungen sind von überall auf der Welt abzurufen und bieten daher einen größtmöglichen Verteilungsgrad. Auf dem Benutzerrechner muß nur ein Standardbrowser (z.B. Netscape Navigator oder Internet Explorer) installiert sein. Die Frage nach dem Betriebssystem ist nur insofern interessant, als der vorhandene Browser auch alle für die Anwendung geforderten Eigenschaften besitzt (Ausstattungsmerkmale Browser-Plugins oder Version der Java VM). Welche Möglichkeiten es gibt, eine Applikation mit Hilfe eines WWW-Browsers und der darin integrierten Java VM zu bedienen, möchte ich in den folgenden Unterkapiteln erläutern.

2.3.1.1 Dynamisch erzeugte HTML-Seiten

Webserver bieten nicht nur die Möglichkeit, HTML-Seiten als statische Informationen für WWW-Browser anzubieten, sondern diese Seiten zur Laufzeit zu erzeugen. Diese Erzeugung kann durch eine beliebige Anwendung geschehen, die mit einem Webserver verbunden ist. Hierzu werden oft Skriptsprachen (Perl, PHP, TCL, ...) eingesetzt. Die Anwendungen werden über das Common Gateway Interface (CGI) vom Webserver aufgerufen, sofern eine Seite generiert werden soll [Münz 98].

Auf dem Benutzerrechner wird über den WWW-Browser nur die Oberfläche einer Anwendung (Präsentation) dargestellte, die Bearbeitung der Daten findet vollständig auf dem Webserver statt. Auf den HTML-Seiten können Informationen angezeigt oder mittels Formularen eingegeben werden, die dann über einen Absenden-Button wieder an den Server geschickt werden.

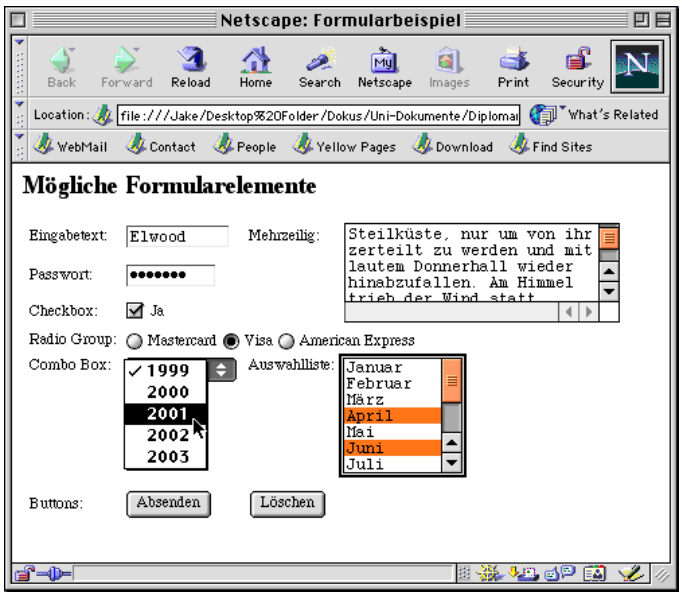

Abbildung 2-6: HTML-Formular-Elemente

In Abbildung 2-6 sind alle möglichen Formularelemente aufgezeigt. Zwar gibt es eine reichhaltige Auswahl an Elementen, aber es bleiben *Formular*elemente. D.h., die einzige Interaktion die hier möglich ist, ist das Eingeben und Abschicken von Formularen. Ansonsten können Daten in fast beliebiger Form präsentiert werden. Für die Bearbeitung und Aufbereitung der Informationen ist einzig und allein der Webserver zuständig, was bei vielen gleichzeitigen Anfragen über das Internet zu einem Performance-Problemen führen kann. Eine Lösung ist das Verteilen der HTTP-Anfragen an mehrere Server über einen Load-Balancing-Mechanismus. Da der HTML-Browser hier keine Anforderungen außer der Darstellung von HTML-Elementen erfüllen muß, ist die Gefahr einer Inkompatibilität gering. Die HTML-Elemente sind zwar spezifiziert [W3C], jedoch gibt es teilweise kleine Unterschiede und Erweiterungen in aktuellen Browsern (Internet Explorer & Netscape) [Münz 98]. Der Umgang mit Standardelementen ist aber einheitlich.

Basiert eine C/S-Anwendung im wesentlichen auf dem Ausfüllen von Formularen, so sind dynamisch erzeugte HTML-Seiten sicherlich ein einfacher Weg, die Applikation für den Benutzer verteilt zu realisieren. Sobald aber ein höherer Grad an Interaktion erforderlich ist, sei es nur, daß durch eine Auswahl aus einem Menü die Datenanzeige aktualisiert werden muß, stößt dieses Verfahren schnell an seine Grenzen. Eine HTML-Seite kann nur dann neu dargestellt werden, wenn eine Anfrage an den Server gesendet und eine neue Seite generiert wird. Das Auswählen eines Menüpunktes mit Aktualisierung der Datenanzeige muß daher zu einer Anfrage an den Server führen, wobei die HTML-Seite dort komplett neu erzeugt werden muß.

Sofern zur Generierung von HTML-Seiten auf dem Server Skriptsprachen eingesetzt werden, die dann meist direkt auf Datenbanken aufsetzten, ist der Abstraktions- und Transparenzgrad bei der Anwendungsentwicklung gering. Mittlerweile ist auch mit Java-Servlets und Java Server

Pages (JSP) eine Lösung mit Java etabliert [Hunter/Crawford 98], so daß beliebige andere Anwendungen in Java einfach integriert werden können.

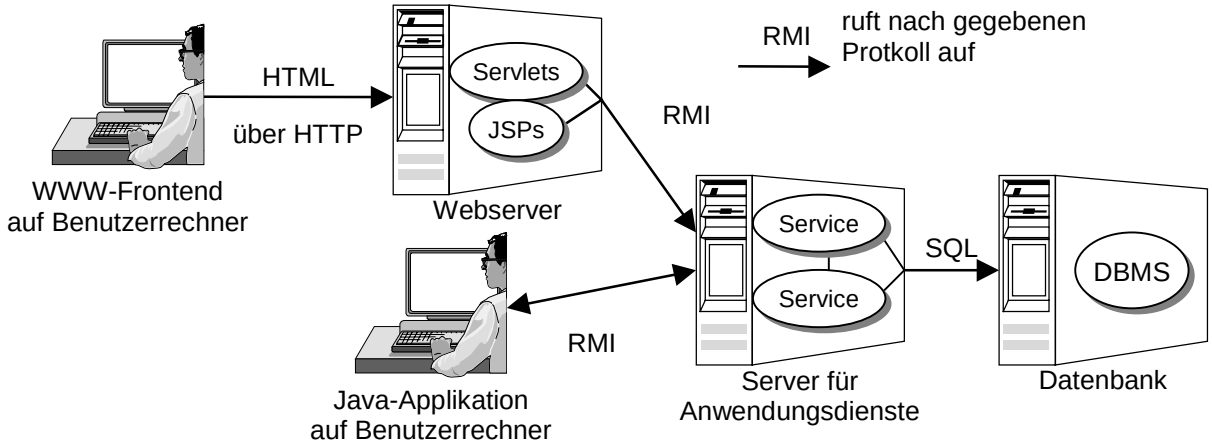

Abbildung 2-7: WWW & N-Tier. Die Anwendungsdienste können sich auch untereinander benutzen

Ein Beispiel für die Anbindung anderer Java-Anwendungen ist ein N-Tier C/S-System, bei dem ein Server Dienste für einen Bereich einer Anwendung zur Verfügung stellt und WWW-Browser auf den Benutzerrechnern über einen Webserver darauf zugreifen. Bei einer solchen Lösung kann recht einfach auf die existierenden Schnittstellen des Servers aufgesetzten werden, wobei für die Webschnittstelle ein eigenes Frontend entwickelt wird. Je nach Art der Anwendung wird es aber unterschiedlich schwer, den Umgang mit einer Anwendung auf eine formularbasierte Sichtweise umzustellen. Auf diesen Punkt werde ich noch ausführlicher im Beispiel im nächsten Kapitel eingehen (siehe auch [Otto/Schuler 00]).

2.3.1.2 Java Skript und Dynamic HTML

Java Skript[8] ist eine Skriptsprache, die auf den WWW-Browser des Benutzerrechners geladen und ausgeführt wird. Die Sprache wurde von Netscape für den eigenen WWW-Browser eingeführt, hat sich aber mittlerweile zu einem allgemeinen Standard weiterentwickelt [Flanagan 98]. Java Skript bietet die Möglichkeit, auf Ereignisse des Benutzers zu reagieren, die an HTML-Elementen festgemacht werden können: Mausbewegungen über die Elemente, Veränderungen oder Aktivierung von Formularelementen sowie zeitliche Ereignisse. Dadurch ist es mit Java Skript möglich, z.B. Formulare vor dem Abschicken auf syntaktische Korrektheit zu prüfen und damit Anfragen an den Server zu vermeiden. Andere typische Beispiele wären das automatische Ausfüllen von Feldern (Summe von Beträgen bei einer Rechnungseingabe) oder das Verändern von Bildern, wenn sich der Mauszeiger über sie hinweg bewegt.

[8] Der Name ist etwas irreführend: Java Skript hat außer einer syntaktischen Ähnlichkeit nichts mit der Programmiersprache Java zu tun.

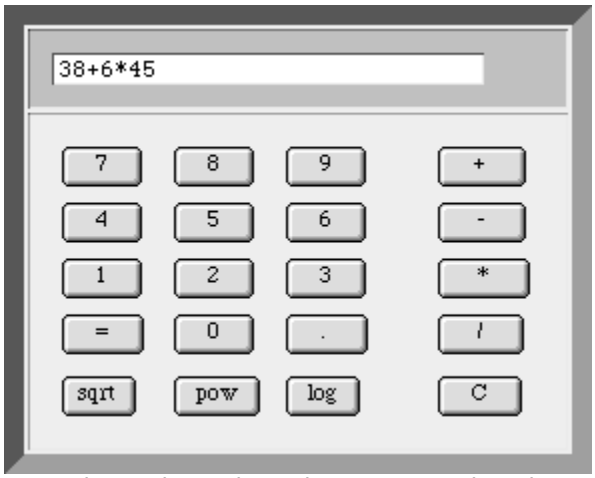

Abbildung 2-8: Ein Taschenrechner als HTML-Formular. Die Funktionalität wird durch Java Skript realisiert (Beispiel aus Self-HTML [Münz 98]), eine Übermittlung von Daten zum WWW-Server findet hier nicht statt

Mit Java Skript lassen sich einige Mängel einer reinen Formulareingabe beheben und die Eingabe interaktiver gestalten. Eine vollständige Anwendung kann damit nicht auf einem WWW-Browser ablaufen. Es können nur Anfragen an einen Webserver verschickt werden, die eine HTML-Seite zurückliefen. Ein Zugriff auf Daten eines Server, die dann beispielsweise mittels einer Java-Skript-Anwendung ausgewertet und dargestellt werden, ist nicht möglich. Ein weiteres Problem sind die gestiegenen Anforderungen an den Browser, der nun neben der Anzeige von HTML auch die Sprache Java Skript verstehen und ausführen muß. Der unterschiedliche Entwicklungsstand der Implementierung von Java Skript in den Browsern führt bei komplexeren Anwendungen zu Problemen, sofern sie für bestimmte Versionen von Java Skript entwickelt wurden. Die Anwendungen sind dann nicht mehr uneingeschränkt auf allen Browsern lauffähig (siehe [Münz 98] und weiter [Netscape 00][Microsoft 00]).

Allerdings wird von den Herstellern gängiger Browser (Netscape und Microsoft) betont, daß sie sich mit den Sprachinterpretern ihrer Browser am Standard für Internet-Skriptsprachen, ECMA-262, orientieren [Münz 98]. Das ECMA-Komitee, dem verschiedene Softwarehersteller, unter anderem auch Microsoft und Netscape, angehören, ist bemüht, im Bereich der Skriptsprachen einen allgemeingültigen Standard zu definieren [ECMA 99].

Weiter geht die Entwicklung bei Dynamic-HTML[9], kurz DHTML: Genau wie bei Java Skript wird hier auch Software auf den WWW-Browser geladen, jedoch bietet DHTML eine noch reichhaltigere Funktionalität, die sogar einen Zugriff auf externe Daten ermöglicht [Münz 98]. Mit DHTML soll es möglich sein, HTML-Seiten soweit zu ‚dynamisieren', so daß sich der Browser Seiten darstellt, sondern wie eine Anwendung vollkommen interaktiv reagieren kann. Auf alle Elemente einer HTML-Seite wird dazu über ein sogenanntes Document Object Model, kurz DOM, zugegriffen. Die einzelnen HTML-Elemente sind in diesem Modell hierarchisch angeordnet, wobei auch auf das Fenster des Browsers mit seinen Maus- und Tastaturereignissen zugegriffen werden kan. Auf Ereignisse kann reagiert werden, um Attribute an den HTML-Elementen oder am Browser zu verändern.

[9] Nicht zu verwechseln mit dynamischen, auf dem Server erzeugten HTML-Seiten

Leider steigen auch mit der Komplexität der Sprache die Anforderungen an Browser, die schlecht erfüllt sind: Sprachstandards sind in den Versionen der Browser unterschiedlich gut unterstützt und außerdem gib es zwischen den beiden gängigen WWW-Browsern (Netscape Navigator und Internet Explorer) auch noch grundverschiedene Implementierungen, so daß man bei DHTML von browserspezifischer Programmierung sprechen kann. Sowohl bei Java Skript als auch bei DHTML bleibt die softwaretechnische Einbindung der Systeme schwierig: WWW-Anwendungen müssen sehr auf die Möglichkeiten von HTML zurecht geschnitten und die Dienste des Servers auf diese Möglichkeiten angepaßt werden.

2.3.1.3 Applets

Die Programmiersprache Java bietet die Möglichkeit, Applikationen zu entwickeln, die über das Internet verteilt werden können: sogenannte Applets [Flanagan 99B]. Der Softwarecode einer Anwendung wird dabei einfach von einem Webserver heruntergeladen und auf dem Benutzerrechner zum Laufen gebracht. Ein Applet ist in einer HTML-Seite integriert[10], wobei die Angabe der Quelle des Applet-Codes als Parameter in der HTML-Seite anzugeben ist. Um ein Applet ablaufen zu lassen, muß auf dem Benutzerrechner eine Java VM installiert sein. Eine VM ist ein Bestandteil der gängigen WWW-Browser, oder aber der Browser benutzt eine Installation von Java auf dem Benutzerrechner.

Applets bieten eigentlich die vollen Möglichkeiten der Programmiersprache Java, jedoch gibt es für Applets ein strenges Sicherheitskonzept: Da sich der Softwarecode einer Anwendung ohne Aufwand herunterladen läßt, indem der Benutzer sich einfach eine HTML-Seite anschaut, werden die Möglichkeiten eines Applets sehr stark beschnitten: Das Applet läuft in einer sogenannten **Sandbox** ab, einer sicheren Umgebung, die von sich aus keine Möglichkeiten bietet, Informationen über dem Benutzerrechner oder über die Dateien auf dem Benutzerrechner zu erlangen. Auch Netzverbindungen sind eingeschränkt, so daß Applets nur eine Verbindung zu dem Server aufnehmen können, von dem sie auch heruntergeladen worden sind.

Falls Applets mehr Möglichkeiten auf dem Benutzerrechner haben sollen, müssen sie signiert sein, d.h. die Herkunft des Softwarecodes ist über Verschlüsselungsmechanismen und eine Zertifizierungsstelle eindeutig nachzuvollziehen. Einem signierten Applet kann der Benutzer mehr Berechtigungen geben, indem er für dieses Applet einzelne, weiterführende Aktionen erlaubt. Somit ist sichergestellt, daß der Benutzer über Aktionen des Applets informiert ist und sie nur durch seine explizite Zustimmung erlaubt sind. Sollen signierte Applets in einem Browser ablaufen, muß die Zertifizierungsstelle dem Browser auch bekannt sein. Der Dienst der Zertifizierung ist nicht kostenlos und außerdem sind die Sicherheitsmodelle der gängigen beiden Browsern nicht einheitlich. Ein signiertes Applets ist daher nicht ohne weiteres auf einen Benutzerrechner zu bringen.

[10] Applets können auf dem Benutzerrechner auch eigene Fenster öffnen

Abbildung 2-9: Ein Applet zur Depotverwaltung, Quelle Consors AG

Allgemein erfüllen Applets durch ihre automatische Codeverteilung die Forderung nach einfachem Deployment, womit gleichzeitig das Problem der Schnittstellenänderung gelöst ist. Der Softwarecode ist nicht fest auf dem Benutzerrechner installiert, so daß eine neue Version der Anwendung auf dem Benutzerrechner automatisch durch das Laden des Softwarecodes aufgespielt wird. Da auf dem Benutzerrechner eine beliebige Anwendung ablaufen kann, hängen die Möglichkeiten der Performance und der Skalierung von der Anwendung selbst ab. Je nach Anwendungsfall kann der Entwickler entscheiden, welche Aufgaben im Applet und welche Aufgaben auf einem Server ausgeführt werden. Weiter lassen sich Mechanismen zum Load-Balancing in das Applet integrieren.

Unsignierte Applets können zwar nur mit dem Server kommunizieren, von dem sie geladen wurden, allerdings können Applets dazu eine beliebige Middleware verwenden, z.B. Socket-Kommunikation, RMI oder CORBA. Die Verwendung von Sockets ist die schnellste Kommunikationsform, bietet allerdings kaum eine softwaretechnische Transparenz. Der in Java integrierte Verteilungsmechanismus RMI [Farley 98] oder CORBA [Orfali 99] bieten bessere softwaretechnische Möglichkeiten, sind aber langsamer. Firewalls sind für alle technischen Kommunikationsprotokolle ein Hindernis. Über ein Tunneling durch das HTTP-Protokoll [Hunter/Crawford 98] kann Abhilfe geschaffen werden, siehe dazu auch 2.2.1.

Leider ist die technische Infrastruktur zur Ausführung von Applets in WWW-Browsern nicht sehr befriedigend implementiert: Die unterschiedliche Güte der Implementierung der Java VM führt zu zahlreichen Problemen, die mit der Komplexität des Applets zunehmen. Die Konstruktion einer größeren Anwendung wird durch diese Gegebenheiten erschwert. Weiter wird in den gängigen Browsern der Softwarecode von Applets immer wieder vom Webserver geladen. Ein geeignetes Caching würde Abhilfe schaffen und die Benutzung von größeren Applikationen erleichtern.

2.3.2 Verteilungssysteme auf grafischer Ebene

Unter Distributed Presentation (siehe 2.1) fällt auch eine Klasse von Systemen, bei denen auf Benutzerrechner nur die grafische Präsentation einer auf dem Server laufenden Anwendung übertragen wird. Auf dem Benutzerrechner ist eine **Display-Engine** installiert, die von einer entsprechenden Software auf dem Server mit der grafischen Ausgabe der Anwendung beliefert wird. Diese wird auf dem Benutzerrechner ausgegeben, während Ereignisse, die beispielsweise von der Tastatur oder der Maus kommen, zum Server übertragen und an die Anwendung weitergeleitet werden.

Auf dem Benutzerrechner ist kein applikationsspezifischer Code vorhanden. Alle Anwendungen sind auf dem Server installiert und laufen dort ab. Das Softwaredeployment-Problem stellt sich also gar nicht, womit auch das Problem der Schnittstellenänderung bei solchen Systemen nicht vorkommt. Über den Verteilungsaspekt zwischen Benutzerrechner und Server muß sich der Anwendungsentwickler bei beiden Systemen keine Gedanken machen. In den folgenden Unterkapiteln möchte ich zwei solcher Systeme kurz vorstellen.

2.3.2.1 Das X/Motif-System

Aus dem Unix-Bereich kommt eine Lösung zur Verteilung von Anwendungen auf grafischer Ebene: das X-System [Barton 94]. Es besteht aus einer Display-Engine auf dem Benutzerrechner, die **X-Server** genannt wird. Der Begriff ‚Server' wird hier verwendet, weil die Display-Engine einen Dienst anbietet, nämlich das Darstellen einer grafischen Präsentation auf dem Benutzerrechner. Auf einem zentralen Serverrechner laufen die Anwendungen für die Benutzerrechner. Die Anwendungen nutzen dabei den X-Server auf dem Benutzerrechner als Diensterbringer, um ihre Präsentation darzustellen. Daher heißen die Anwendungen im Sprachgebrauch von ‚X' auch X-Client-Programme, da sie über das X-Protokoll dem X-Server Anweisungen zum Zeichnen geben. Der X-Server bearbeitet auch die Ereignisse von Eingabegeräten (Maus, Tastatur, ...), welche er über das X-Protokoll an die X-Client-Programme zur Verarbeitung weiterleitet.

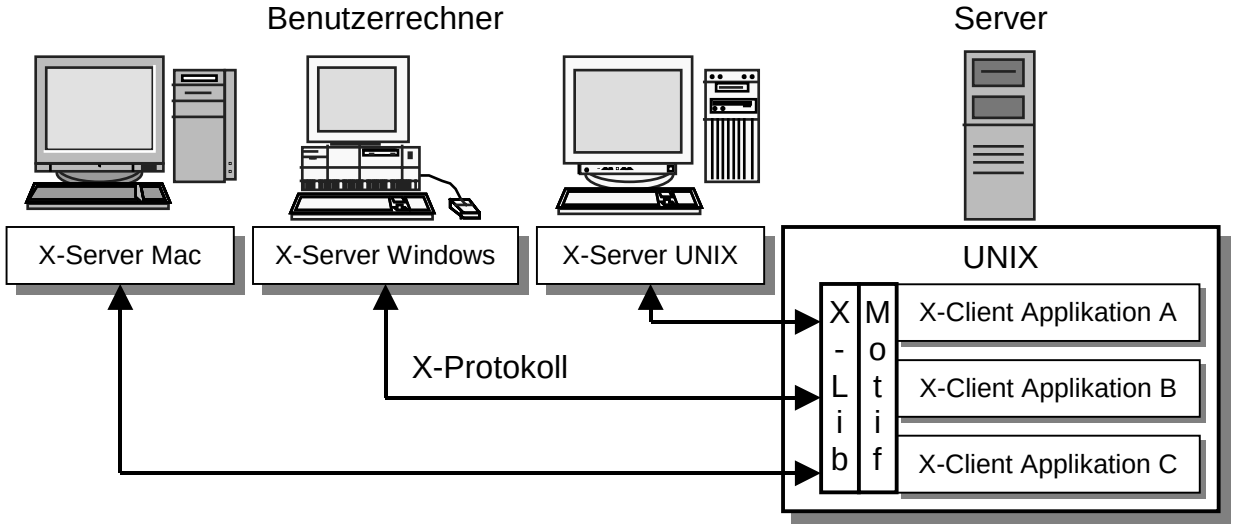

Abbildung 2-10: Eine X-Umgebung

Das X-Protokoll arbeitet auf einer elementaren Ebene: Es kennt Fenster als rechteckige, beliebig verschieb- und überlappbare Bereiche, in die grafische Grundelemente, wie Texte, Linien oder Füllflächen gezeichnet werden können. Die Ereignisse der Eingabegeräte werden ebenfalls auf dieser niedrigen Ebene über das X-Protokoll übertragen. Der X-Server übernimmt die Verwaltung der Fenster, das Zeichnen der Grafik und die Verwaltung der Eingabegeräte. Der X-Server abstrahiert dabei von der Hardware der Ein- und Ausgabegeräte. Er bietet über Metainformationen seine Einstellungsparameter wie Bildschirmgröße, Farbtiefe oder installierte Schriftarten an.

Da sich auf der niedrigen Ebene des X-Protokolls schlecht X-Client-Programme entwickeln lassen, existieren Bibliotheken wie X-Toolkit und Motif, die dem Anwendungsentwickler eine Oberflächenverwaltung auf Basis von Widgets anbieten. Die X-Lib-Bibliothek implementiert die Funktionen des X-Protokolls, auf der dann X-Toolkit und Motif aufsetzen.

Auf den Benutzerrechnern muß für eine X-Umgebung ein X-Server installiert sein, wobei es für alle gängigen Betriebssysteme entsprechende Produkte gibt. Die Anwendungen auf dem Server laufen dagegen in der Regel nur auf Unix-Systemen. Das Starten der Anwendung erfolgt daher auch über eine Terminalemulation, über die sich der Benutzer mit dem Server verbindet. Die gestarteten Anwendungen sind eigenständige Prozesse, deren grafische Ausgabe dann durch das X-System umgelenkt wird. Die Ausgabe kann auch über einen X-Server auf demselben Rechner geschehen, womit sich eine Anwendung dann wie eine lokale GUI-Applikation verhält.

Die Anwendung wird gegen Widgets aus den Bibliotheken programmiert, wobei der Verteilungsaspekt dabei keine Bedeutung hat. Die Verteilung der Oberfläche erfolgt transparent. Da der Anwendungsentwickler direkt Widgets aus seiner Anwendung benutzt, ist hier eine sehr starke Verbindung zu der verwendeten Technologie gegeben.

Unter dem Gesichtspunkt der Performance erweist sich ein X-System als problematisch: Da das X-Protokoll recht elementar ist, müssen alle Ereignisse oder Zeichenbefehle auf einer niedrigen Ebene übertragen werden. Jede Änderung an Oberflächenelementen führt zu möglicherweise vielen Zeichenbefehlen, die über das Netz übertragen werden müssen. Wird beispielsweise durch den Benutzer in einem Eingabefeld ein Text eingegeben, müssen die einzelnen Tastaturereignisse vom X-Server an die X-Client-Anwendung gesendet werden, die wiederum über neue Zeichenbefehle die Ausgabe aktualisieren muß. Daher findet schon bei kleinsten Ereignissen ein Netzverkehr statt. Daher sind die Anforderungen an das Netz in bezug auf die Latenzzeit und Bandbreite hoch, da eine Reaktion auf Ereignisse des Benutzers auch möglichst schnell erfolgen muß. Eine Verteilung über das Internet oder langsame Modem-verbindungen ist daher problematisch. Durch den hohen Kommunikationsbedarf entsteht auch auf dem Server ein weiteres Problem: Die X-Client-Anwendungen müssen schon auf kleinste Benutzeraktionen reagieren. Daher verbrauchen diese Anwendungen dauerhaft Rechenzeit, so daß auf einem Server bei vielen laufenden Anwendungen schnell ein Lastproblem auftreten kann.

2.3.2.2 Citrixs MetaFrame & ICA

Eine Lösung für die grafische Verteilung mit einem Windows-System ist Citrixs Independent Computing Architecture (ICA) [Citrix 00]. Das System setzt auf einem Windows NT-Server auf,

der Mehrbenutzerbetrieb unterstützt. Die Software MetaFrame stellt für verschiedene Benutzerrechner die Anwendungen auf einem Server zur Verfügung, indem die Grafikausgabe über das sogenannte Thinwire-Protokoll auf die Benutzerrechner verteilt wird. Auf den Benutzerrechnern muß eine entsprechende Display-Engine, in diesem Fall ein ICA-Client installiert sein. Damit kann eine Anwendung wie bei dem X-System von einem beliebigen Benutzerrechner bedient werden, sofern für das Betriebssystem des Benutzerrechners ein ICA-Client verfügbar ist.

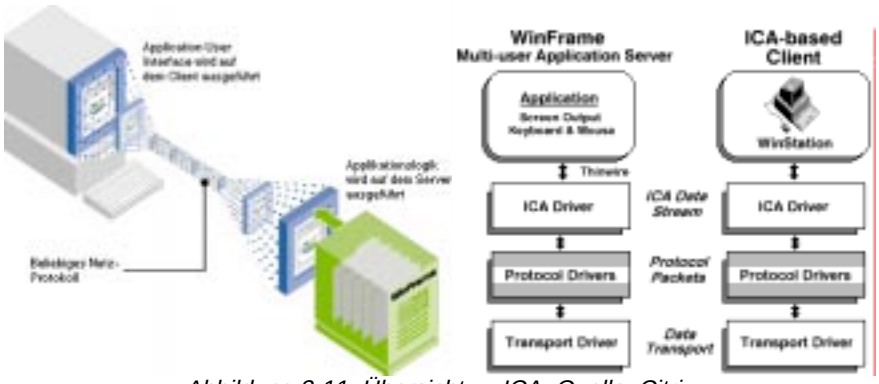

Abbildung 2-11: Übersicht zu ICA, Quelle: Citrix

Zur grafischen Darstellung von Anwendungen wird nicht der Treiber für die übliche Bildschirmausgabe verwendet, sondern ein eigener ICA-Treiber, der die grafische Ausgabe der Applikation in das Thinwire-Protokoll umsetzt Dieser Mechanismus wird ebenfalls für die Eingabegeräte (Maus, Tastatur) benutzt. Im Vergleich zu dem X-System kann diese Lösung damit für alle möglichen Anwendungen genutzt werden, da gegen keine Bibliothek programmiert werden muß, sondern die Verteilung über Treiber auf Betriebssystemebene funktioniert.

Über die Erweiterung **NFuse** können Windows-Applikationen auch über Browser aus dem Internet bedient werden. NFuse bietet die Möglichkeit, über einen Webserver einen persönlichen Desktop für einen Benutzer zu erzeugen. Dieser Desktop wird mittels eines HTML-Browser bedient. Der Desktop bietet vor allem die Möglichkeit an, Programme zu starten, die dann über das Thinwire-Protokoll verteilt werden. Auf dem Benutzerrechner wird dazu ein ICA-Client als Java-Applet gestartet, das als Display-Engine für die Applikation auf dem Server fungiert. Auf Ressourcen des Benutzerrechner (z.B. das Dateisystem oder einen Drucker) kann zugegriffen werden, da es sich um ein signiertes Applet handelt, dem der Benutzer weitergehende Zugriffsrechte übertragen kann.

Bei dem gesamten System hat man sich sehr viel Gedanken über eine performante Übertragung gemacht, so daß die Bedienung auch über eine Modemverbindung noch möglich ist. Aber wie bei dem X- System zieht jede Aktion des Benutzers auch Netzverkehr nach sich, da auf dem Server die Präsentation der Applikation neu erstellt werden muß. Um die Rechenlast bei vielen Benutzern zu verteilen, bietet das Produkt auch eine Unterstützung zum Betreiben einer ‚Serverfarm', in der Applikationen auf mehrere Server verteilt werden. Eine Verteilung von Applikationen ist daher wie bei dem X-System mit hohen Anforderungen an den Server verbunden. Dafür lassen sich aber beliebige Applikationen auch über langsame Netzverbindungen verteilen, die Transparenz bei der Programmierung ist vollkommen gewährleistet.

2.3.3 Der Thin Client Rechner

Die grafische Verteilung von Anwendungen eignet sich für eine besondere Form des Benutzerrechners: den Thin Client Rechner. Dahinter steckt die Idee, daß den Benutzern nur noch Rechner zur Verfügung gestellt werden, die ihre gesamte Konfiguration von einem zentralen Server beziehen. Betriebssystem, Anwendungen und auch eigene Benutzerdaten werden auf diesem Server verwaltet. Der Thin Client Rechner muß daher keine Festplatte haben. Ein Beispiel für ein solch abgespecktes System ist ein X-Terminal, auf dem nur ein X-Server läuft, der die grafische Ausgabe übernimmt.

Da auf einem Thin Client Rechner keine anwendungsspezifische Software oder ein Betriebssystem installiert werden muß, ist für diesen Rechner keine administrative Wartung mehr nötig. Der Einsatz solcher Rechner ist gerade für größere Organisationen und Unternehmen interessant, da sich der Administrationsaufwand stark vermindern läßt. Anwendungen müssen nur auf einem zentralen Server installiert werden, die Benutzerrechner erhalten damit automatisch eine aktuelle Version.

So basieren z.B. die Thin Client Rechner von Neoware oder NCD auf der Lösung von Citrix ICA [Neoware 00][NCD 00]. Auf dem Server laufen beliebige Applikationen unter Windows, während auf dem Benutzerrechner nur ein ICA Client unter Windows CE installiert ist. Das Boot-System kann auf einem Flash-Memory gehalten werden, eine Administration findet zentral von einem Server statt. Zugangsrechte und Benutzerrechte werden ebenfalls zentral administriert.

Für einen Einsatz außerhalb einer Organisation ist ein solcher Rechner aber nicht geeignet und auch nicht konstruiert. Falls auf einem Benutzerrechner aufwendige Aufgaben erledigt werden sollen (z.B. Softwareentwicklung), ist ein Einsatz eines normalen Desktop-Rechners sinnvoll, da ansonsten der zentralen Server mit solchen Aufgaben schnell überlastet wäre.

2.4 Zusammenfassung & Ausblick

In 2.2. habe ich einige Forderungen an ein C/S-Anwendungssystem aufgestellt: einfaches Softwaredeployment, Möglichkeit einer Schnittstellenänderung, ausreichende Geschwindigkeit und möglichst transparente Programmierung. In 2.3. habe ich existierende Technologien vorgestellt und anhand dieser Kriterien überprüft. Obwohl eine Bewertung solcher Technologien auch sehr von der konkreten Realisierung abhängig ist, möchte ich die Technologien anhand meiner Kriterien vergleichend bewerten. In der folgenden Tabelle bedeutet ein ‚+‘ eine ausreichend gute Erfüllung der Kriterien, ein ‚-‘ eine mangelhafte und ein ‚o‘ eine durchschnittliche, jeweils noch mit Abstufungen:

		Dynamisches HTML	Java-Skript	DHTML	Applets [A]			X-System	ICA NFuse
					RMI	Socket	Corba		
Software-deployment	Lokalität	+	+	+	+/o[B]			+/o[B]	+/o[B]
	Betriebssystem	+	+	+	+/o			+/o	+/o
	Ausstattung	+	o[C]	-[C]	+/o			+	+/o
Schnittstellenänderung		+	+	+	(+)[D]			+	+
Geschwin-digkeit	Skalierung	+	+	+	+			-	o
	Lastverteilung	-	o	+	+			-	o
	Performance	+	+	+	+/o	+	+/o	-	o
Trans-parenz	Fehlerman-agement	-	-	-	(-)[E]			-	-/o
	Wissen über Middleware	-	-	-	+	-	o	+	+

[A] Applets aufgeschlüsselt nach der Art der Middleware, über die sie mit dem Server kommunizieren. Zu der Middleware sind im folgenden Text noch weitere Erläuterungen vorhanden

[B] Mit +/o bewertet, da im Netz Probleme mit Firewalls auftreten können.

[C] Unterschiedliche Implementierung in Browsern können zu Problemen führen.

[D] Änderungen an der Schnittstelle der Serveranwendung müssen am Applet nachgezogen werden. Von einem Webserver geladene Applets sind unproblematisch, der Softwarecode wird automatisch dem Benutzerrechner zur Verfügung gestellt.

[E] Ein Fehlermanagement kann nur explizit programmiert werden, ist aber möglich

Alle Technologien lassen prinzipiell eine Verteilung über das Internet zu und bieten auf verschiedenen Benutzerrechnern lauffähige Frontends an. HTML-Anwendungen (Dynamisches HTML, Java Skript, DHTML) sind durch die Bedienung mittels eines Browser im Internet sehr leicht zu verteilen, jedoch ist die Entwicklung solcher Anwendungen mit einigen Einschränkungen verbunden. Bei der Anwendungsentwicklung muß man sich an den technischen Gegebenheiten von HTML oder den Skriptsprachen orientieren. WWW-Frontends müssen für Anwendungssysteme speziell entwickelt werden, sofern noch keine WWW-Schnittstelle vorhanden war.

Die Verteilungssysteme auf grafischer Ebene weisen in diesem Punkt die besten Eigenschaften auf, da beliebige Anwendungen einfach verteilt werden können. Für den Entwicklungsprozeß der Anwendung ist die spätere Verteilung auf die Benutzerrechner nicht von Bedeutung. Jedoch haben grafische Verteilungssystem große Schwächen bei der Skalierung.

Eine weitere Möglichkeit stellen Applets dar, die mehr Funktionalität bieten als reine HTML-Anwendungen und auch performanter als grafische Verteilungssysteme sind. Da sie auf Java basieren, ist vom Prinzip her eine Unabhängigkeit vom Benutzerrechner gegeben. Updates werden automatisch verteilt, da der Softwarecode immer von einem Webserver geladen wird. Das Problem des Softwaredeployments stellt sich daher nicht. Eine weiteres Problem bei Applets ist die eingeschränkte Benutzbarkeit aufgrund des Sicherheitskonzeptes. Sofern in einer Anwendung nicht auf lokale Ressourcen des Benutzerrechners zugegriffen werden muß, ist diese Einschränkung aber nicht von Bedeutung.

2.4.1 Ausblick: Thin Clients

In Abbildung 2-12 sind zwei unterschiedliche Frontends eines Systems dargestellt: die Kalenderverwaltung von Netscape. Beide Systeme bieten zwar die gleiche Funktionalität, jedoch verfügt die eigenständige Applikation über die vollen Möglichkeiten einer GUI. So können Termine direkt durch Anklicken eingetragen werden und auch die Darstellung von Terminen ist übersichtlicher als in der HTML-Lösung, die solche Möglichkeiten nur schwer bieten kann. Der Umgang mit der HTML-Lösung ist eingeschränkter, während bei einer GUI die Mittel weitaus größer und vielfältiger sein können.

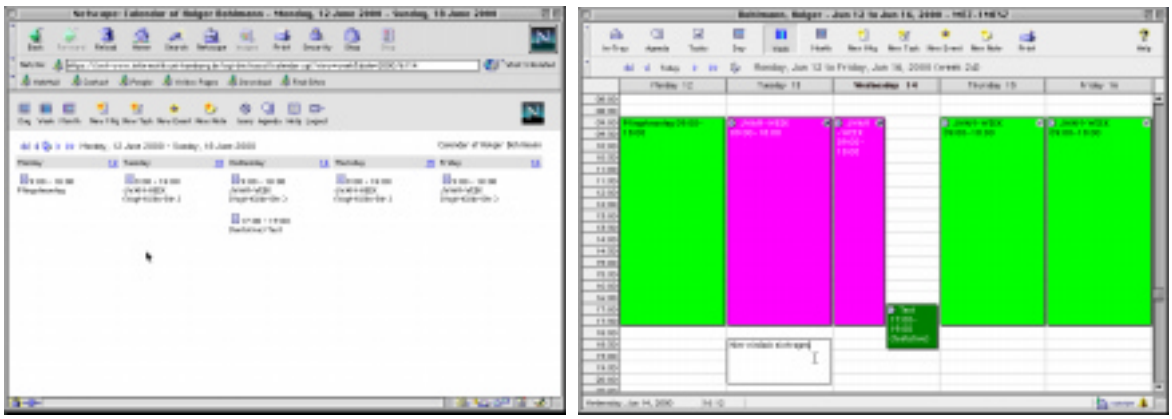

Abbildung 2-12: Links Netscape Calender als HTML-Variante im Browser
Rechts: Netscape Calender als Applikation

Eine Implementierung des Kalenders als Applet könnte die Vorteile beider Systeme zusammenbringen: eine komplette GUI und eine unproblematische Verteilung und Benutzung über das Internet. Das Sicherheitskonzept für Applets ist für diese Anwendung nicht weiter von Bedeutung, da bis auf die Druckmöglichkeit keine lokalen Ressourcen benötigt werden. Der Umfang einer solchen Applikation ist recht groß, so daß auch viel Softwarecode übertragen werden müßte. Leider beherrschen die gängigen Browser z.Z. kein effektives Caching was Java-Klassen betrifft, so daß der Softwarecode immer wieder geladen werden müßte. Da die Qualität der virtuellen Maschinen für Java unterschiedlich ist (siehe 2.3.1.3), wächst mit der Größe der Anwendung auch die Gefahr, daß ein solches Applet wegen Inkompatibilität nicht mehr überall lauffähig ist.

Eine andere Lösungsmöglichkeit wäre die Realisierung einer Anwendung als Thin Client mit der Hilfe von Java-Applets. Auf dem Benutzerrechner ist nur die GUI vorhanden. Auf einem Server läuft die eigentliche Anwendung, Änderungen an der GUI und Reaktionen von der GUI werden über ein Protokoll durch das Netz übertragen. Auf dem Benutzerrechner muß nur eine kleine Applikation geladen werden, während auf dem Server die eigentliche Anwendung läuft.

In den folgenden Kapiteln möchte ich dazu passende Technologien vorstellen: einmal eine allgemeine Lösung zur Verteilung einer GUI und einmal eine Lösung für das JWAM Framework. Beide Systeme sind in Java entwickelt. Bei beiden wird die GUI einer Anwendung auf den Benutzerrechner verteilt, so daß der Benutzerrechner auch die Verwaltung der GUI übernimmt. Im Vergleich zu grafischen Verteilungssystemen ist damit auf dem Benutzerrechner mehr Funktionalität vorhanden. Die Kommunikation zu dem Server ist wesentlich abstrakter, da

Ereignisse von und an Widgets der GUI übertragen werden und nicht grafische Zeichenbefehle oder Tastaturereignisse. Das Darstellung der GUI auf dem Bildschirm und die Abfrage von Benutzerereignissen findet komplett auf dem Benutzerrechner statt. Die Verteilung der GUI ist generisch, so daß die GUI einer beliebigen Anwendungen auf Benutzerrechner übertragen werden kann.

3 Prototyp Helpdesk

In diesem Kapitel stelle ich eine Anwendung vor, die von der Art her viele Frontends anbieten soll: den Helpdesk. Es handelt sich dabei um ein Hilfesystem, in das Anwender Anfragen zu technischen Problemen stellen können, die von Experten für bestimmte Themen beantwortet werden. Dieses Anwendungssystem ist auch Teil einer anderen Diplomarbeit [Otto/Schuler 00] und wurde für diese beiden Diplomarbeiten von den jeweiligen Autoren entwickelt. Dabei wurde nach dem WAM-Ansatz vorgegangen [Züllighoven 98]. Ich werde zuerst auf die Anforderungsermittlung eingehen und die daraus resultierenden Ideen für die Realisierung näher erläutern. Danach betrachte ich die technische Realisierung des Helpdesk genauer, wobei ich den Schwerpunkt der Betrachtung auf die Frontends für das Anwendungssystem setzten werde.

3.1 Anforderungsermittlung

Der hier vorgestellt Helpdesk wurde anhand der Anforderungen im Arbeitsbereich Softwaretechnik der Universität Hamburg nach dem WAM-Ansatz entwickelt. Die Akteure sind Angestellte des Arbeitsbereiches, wobei hier zwischen Anwendern und Administratoren oder Experten für bestimmte Themen unterschieden wird. Experten und Administratoren kennen sich in bestimmten Themen gut aus, wobei ein Administrator auch explizit für die Installation, Wartung und Pflege von Hard- und Software zuständig ist. Die Ergebnisse lassen sich aber auch auf andere Organisationen übertragen, da der Umgang mit einem Helpdesk gleich oder sehr ähnlich sein sollte.

Bei dem Helpdesk handelt es sich um ein Hilfesystem zu einem oder mehreren Problemfeldern, hier **Kategorien** genannt. Beispiele für Kategorien sind Probleme mit Rechnern (Kategorie Rechneradministration) oder Probleme bei der Benutzung von Anwendungssoftware (Kategorie Anwendungssoftware). Das Hilfesystem deckt mit seinen Kategorien einen ganzen Bereich ab, im folgenden auch **Projekt** genannt. Das Hilfesystem steht für eine Anzahl von Anwendern (Angestellte des Arbeitsbereiches) zur Verfügung, die mit Hardware- oder Softwaresystemen in einer Organisationsstruktur (Arbeitsbereich Softwaretechnik) arbeiten. Falls technische Probleme auftreten, soll sich der Anwender an das Hilfesystem mit einer Anfrage zu seinem Problem wenden können. Für die verschiedenen Kategorien sind Administratoren oder Experten zuständig, die dem Anwender bei der Lösung des Problems helfen können. Diese Personen werden hier **Problemlöser** genannt. Da ein Anwender nicht unbedingt eine richtige Zuordnung zu den Kategorien und damit zum zuständigen Problemlöser finden kann, gibt es noch einen Projekt-Problemlöser, an den sich Anwender zuerst wenden können. Dieser kann die Anfragen der Anwender dann an den passenden Problemlöser weiterleiten. Der Projekt-Problemlöser kennt sich mit allen Kategorieren und den Problemlösern des Projektes aus.

Ein solcher Helpdesk kann auch in anderen Situationen verwendet werden, beispielsweise als Hilfesystem für ein Softwareprodukt. Die Anwender sind die Kunden, die dieses Produkt nutzen und sich mit Problemen an den Hersteller werden. Als Kategorien kommen Teile des Produktes in Frage oder verschiedene Versionen auf unterschiedlichen Betriebssystemen. Für diese Kategorien sind eventuell mehrere Problemlöser zuständig, falls eine Person mit den

auflaufenden Anfragen überlastet wäre. Der Projekt-Problemlöser kümmert dann sich um eine Verteilung der Anfragen.

3.1.1 Szenarien

Für das Hilfesystem wurden folgende fünf Szenarien durch Interviews mit allen Akteuren ermittelt, die den *bisherigen* Ablauf schildern, wie mit Problemen von Anwendern umgegangen wird.

- **Szenario 1: Anwender hat ein Problem.** Der Anwender hat ein technisches Problem. Um dem Problemlöser nun dieses Problem zu schildern und nach einer Lösung zu fragen, kann der Anwender den Problemlöser auf drei Wegen kontaktieren: Er sucht ihn direkt in seinem Büro auf, er ruft ihn an oder er schickt ihm eine Email. Die Dringlichkeit entscheidet nicht unbedingt über den Kommunikationsweg, sondern auch die räumliche Erreichbarkeit. In der Regel wird der Problemlöser bei dringenden Problemen nicht über E-Mail kontaktiert, sondern angerufen oder aufgesucht. Eventuell kommen dann über andere Kommunikationswege Rückfragen zu den Problemen.

- **Szenario 2: Anwender nimmt eine Anfrage zurück.** Anfragen werden auch wieder zurückgenommen, falls das Problem schon ohne den Problemlöser erledigt werden konnte. Dieses Szenario tritt insbesondere bei einer Anfrage über E-Mail auf, da solche Anfragen länger dauern können und der Anwender das Problem schon selbst gelöst haben könnte.

- **Szenario 3: Der Problemlöser beantwortet eine Anfrage (löst das Problem).** Die Beantwortung der Anfrage oder die Lösung des Problems erfolgt bei der direkten Kommunikation entweder sofort oder mit einer zeitlichen Verzögerung bei Anfragen über E-Mail. Kann ein Problem im direkten Gespräch nicht sofort gelöst werden, so wird die Lösung des Problems (oder Anweisungen zur Lösung des Problems) per E-Mail mitgeteilt.

- **Szenario 4: Der Problemlöser stellt ein allgemeines Problem fest.** Wenn der Problemlöser viele Anfragen zu demselben Problem bekommt, so schickt der Problemlöser eine Rundmail, die an alle Anwender geht. Dort ist in rein textueller Form beschrieben, wie das Problem aussieht und wie es zu lösen ist. Vermutet der Problemlöser, daß gewisse Probleme auftreten werden, z.B. nach Installation einer neuen Software oder durch Wartungsarbeiten, so werden ebenfalls alle Anwender über eine Rundmail informiert.

- **Szenario 5: Weiterleitung des Problems.** Stellt der Problemlöser fest, daß er für ein Problem nicht zuständig ist oder das Problem aufgrund anderen Gegebenheiten nicht bearbeiten kann, so wird dieses Problem an einen zuständigen Problemlöser weitergegeben. Dem Problemlöser ist bekannt, an wen er sich wenden kann.

Aus diesen Szenarien und weiteren Ideen aus den Interviews konnten schon einige Punkte ermittelt werden, die für das spätere Anwendungssystem Helpdesk wichtig sind und unterstützt werden sollten:

- Inwieweit ein Problem bearbeitet wird, konnte bisher nicht immer nachvollzogen werden. Ein Hilfesystem sollte eine solche Option anbieten.

- Anfragen haben verschiedene Bearbeitungszustände, die häufig wechseln. So können Anfragen weitergeleitet, immer wieder Rückfragen gestellt und Rückfragen beantwortet werden.

- Eine gute Kommunikationsmöglichkeit zwischen Problemlöser und Anwender sollte gewährleistet sein. Anfragen dürfen nicht vergessen werden, wie es bei Telefonanfragen unter Umständen manchmal vorkommt.

- Das Hilfesystem sollte von allen möglichen Arbeitsplätzen aus zu bedienen sein, ohne daß die technische Ausstattung des Arbeitsplatzes eine Rolle spielt. Das Frontend für das Hilfesystem sollte möglichst wenig Anforderungen an den Benutzerrechner und an die Verbindungsart zum Server stellen. Die oder der Problemlöser sollen das Hilfesystem von überall her bedienen können, da sie durch ihre Aufgabe bedingt nicht immer an ihrem Arbeitsplatz zu erreichen sind. Im Vergleich zu dem Beispiel aus der Einleitung ist somit die Benutzbarkeit von überall her auch hier ein wichtiger Punkt, wobei er hier aus anderen Gründen von Bedeutung ist.

Ein weiterer Bestandteil für das Hilfesystem sollte noch eine Wissensbasis sein, in der allgemeine Anfragen gespeichert werden können.

3.1.2 Zustandsmodell für Anfragen

Aus den Szenarien und Ablaufvisionen lassen sich für Anfragen folgende Zustände ermitteln:

- **Neu:** Der Anwender hat ein Problem und schickt es an das Hilfesystem.
- **Weitergeleitet:** Hat der Problemlöser die Anfrage gelesen und hält sich nicht für zuständig, so schickt er sie an einen anderen Problemlöser weiter.
- **In Bearbeitung:** Der Problemlöser hat eine Anfrage gesichtet und hält sich für zuständig. Er übernimmt die Anfrage.
- **Rückfrage gestellt:** Hat der Problemlöser noch weitere Fragen, stellt er eine Rückfrage.
- **Rückfrage beantwortet:** Die Rückfrage durch den Anwender wurde vom Problemlöser beantwortet. Eine erneute Rückfrage kann wieder gestellt werden.
- **Gelöst:** Ist ein Anfrage durch den Problemlöser letztendlich beantwortet, also dem Anwender eine Problemlösung geschrieben, so geht diese in den Zustand ‚Gelöst' über.
- **Archiviert:** Wenn eine Anfrage schon vor längerer Zeit gelöst wurde und das Problem auch für andere Anwender nicht mehr aktuell ist, kann es archiviert werden.

Sofern eine Anfrage durch einen Anwender ins System gestellt wurde, wird sie durch Aktionen der Problemlöser in ihrem Zustand immer wieder verändert. Hinzu kommt noch die Möglichkeit, daß ein Anwender jederzeit eine Anfrage ergänzen kann, was auch am Zustand der Anfrage abzulesen ist.

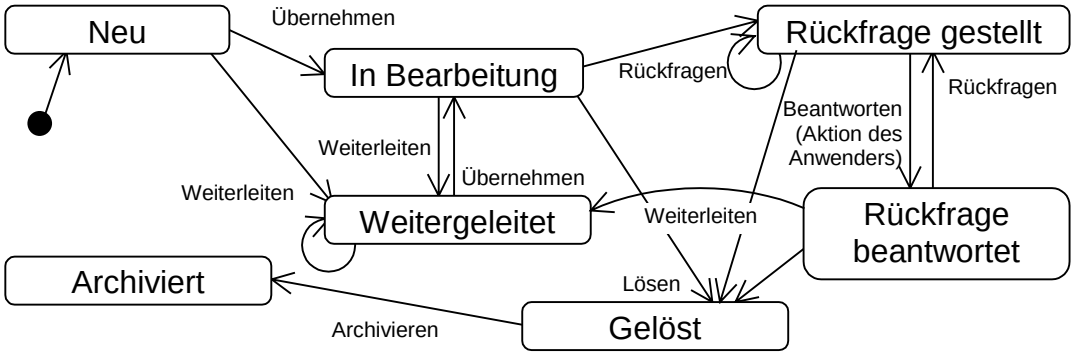

Abbildung 3-1: Zustandsübergänge von Anfragen

Die Zustände sind sowohl für den Anwender, als auch für den Problemlöser interessant. Der Anwender möchte über den Bearbeitungszustand informiert sein, während der Problemlöser über die aktuellen Anfragen auf dem Laufenden gehalten werden möchte. Die Häufigkeit der Zugriffe kann sich dabei stark unterscheiden. Während der Anwender den Zustand seiner Anfragen in unregelmäßigen Zeitabständen abfragt, möchte der Problemlöser über jede Änderung eventuell sofort informiert werden.

3.1.3 Visionen & Frontends

Um den Umgang mit dem Helpdesk besser zu verstehen, wurden Ablauf- und Werkzeugvisionen entwickelt [Züllighoven 98]. Die Ablaufvisionen beschreiben Handlungen von Anwendern und Problemlösern mit dem zukünftigen Hilfesystem anhand konkreter Aktionen, wie z.B. dem Eingeben einer neuen Anfrage. Die Werkzeugvisionen beschreiben das Aussehen der Werkzeuge, mit denen die Abläufe durchgeführt werden. Aus den Ablaufvisionen wurde klar, daß Anwender und Problemlöser unterschiedliche Aktionen ausführen. Diese Abläufe konnten nicht in einem Werkzeug vereinigt werden, da für Anwender und Problemlöser die Anfragen in unterschiedlicher Art und Weise präsentiert werden müssen. Jedoch gibt es einige Gemeinsamkeiten in der Präsentation, so daß die Werkzeuge ähnlich aussehen.

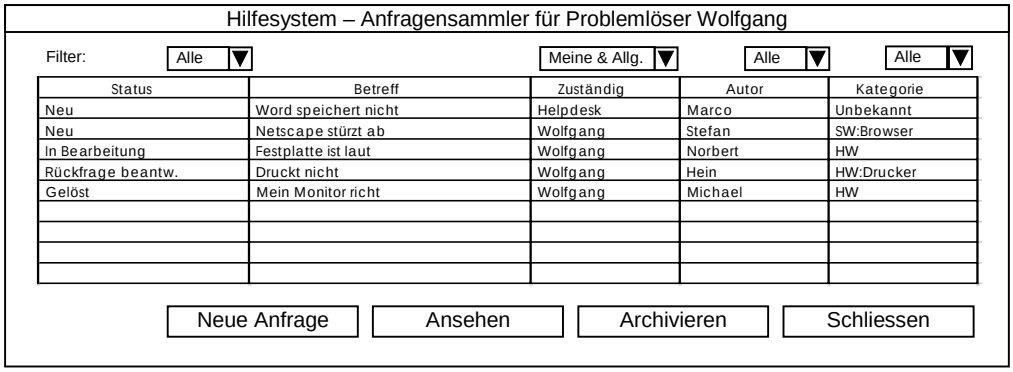

Abbildung 3-2: Werkzeugvision Anfragesammler für Problemlöser

Bei Werkzeugvisionen steht die Frage im Vordergrund, welche Funktionalität das zu entwerfende Werkzeug haben soll. Eine andere Frage ist, wie sich die Funktionalität an der Oberfläche präsentiert [Züllighoven 98]. Unterschiedliche Frontend können sich deshalb in der Präsentation unterschieden, jedoch sollten sie alle die gleiche Funktionalität anbieten, um die Abläufe für den Anwender und Problemlöser zu unterstützen.

Für das Hilfesystem wurde die Anforderung ermittelt, daß es von überall her zu bedienen sein sollte. Die technischen Gegebenheiten auf dem Rechner eines Anwenders oder Problemlösers sollten dabei keine Rolle spielen (einfaches Softwaredeployment). Ein weiterer Punkt war die Erreichbarkeit: Problemlöser möchten eventuell über jede Änderung sofort informiert werden, was für Anwender nicht unbedingt der Fall ist.

Diese Anforderungen führten dazu, daß für das Hilfesystem mehrere Frontends als Prototypen konstruiert wurden. Bei jedem Frontend wurden unterschiedliche Schwerpunkte gesetzt: von wem und wie es am Besten zu benutzen ist (Anwender oder Problemlöser) und inwieweit es von technischen Gegebenheiten unabhängig ist. Anhand der unterschiedlichen Frontends konnte auch untersucht werden, welche Vor- und Nachteile die verwendeten Technologien für die Konstruktion von Werkzeugen haben.

3.2 Der Helpdesk

Die prototypische Implementierung des Helpdesks ist eine Lösung mit unterschiedlichen Frontends. Alle Frontends basieren auf einem fachlichen Service [Otto/Schuler 00]. Ein fachlicher Service bietet für einen Anwendungsbereich, in diesem Fall das Hilfesystem, spezifische Dienste an (siehe auch Distributed Logic in 2.1). Für das Hilfesystem ist das der Umgang mit Anfragen und eine Benutzerverwaltung. Bei dem Helpdesk handelt es sich um eine N-Tier-Architektur.

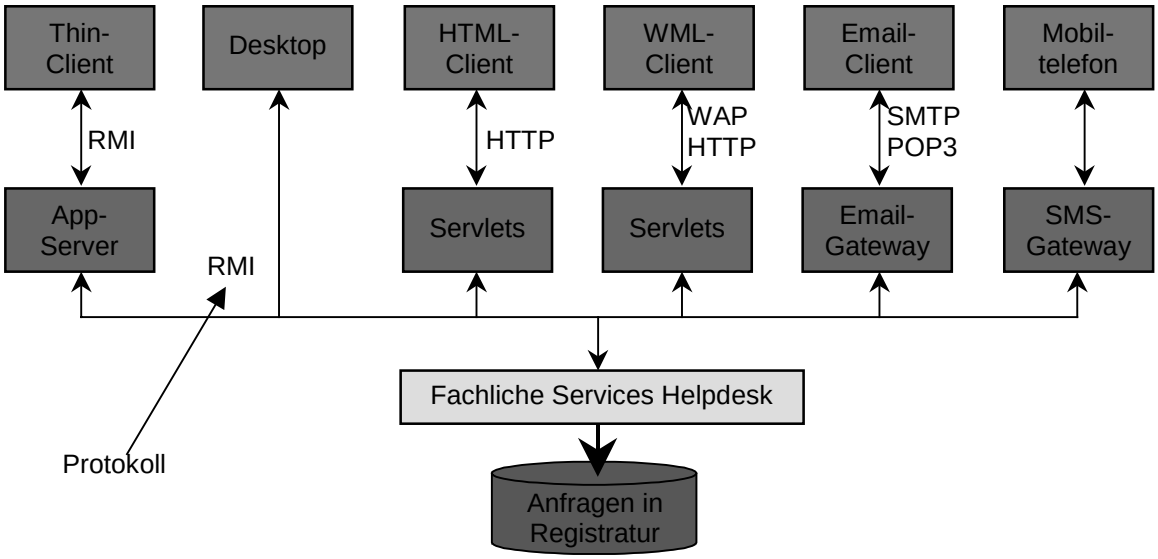

Abbildung 3-3: Infrastruktur Helpdesk

Für die Konstruktion des Helpdesks wurde das JWAM Framework verwendet [JWAM]. Die einzelnen Teile des Anwendungssystems Helpdesk nutzen unterschiedlich viele Komponenten aus dem Framework. In den folgenden Unterkapiteln möchte ich auf die einzelnen Bestandteile des Helpdesks näher eingehen, insbesondere auf die Frontends. Für die einzelnen Frontends wird dabei oft der Begriff **Top** verwendet, z.B. Desktop, Webtop, Waptop und Mailtop. Ein Frontend ist der Teil eines Systems, der aus der Sicht der Benutzers ‚oben' liegt und dem Benutzer eine Bedienung des ‚darunterliegenden' Systems ermöglicht.

3.2.1 Fachliche Services

Der Helpdesk basiert auf zwei fachlichen Services: einem Anfragen-Service und einem Benutzer-Service. Die fachliche Funktionalität, wie mit einer Anfrage umgegangen wird, befindet sich in dem Anfragen-Service. Dadurch wird vermieden, daß diese fachliche Funktionalität in den verschiedenen Frontends mehrfach implementiert wird. Die einzelnen Frontends haben nur die Aufgabe, die Funktionalität eines oder mehrere Services an ihrer Oberfläche zu präsentieren. Gleiches gilt für den Benutzerservice, der für die Verwaltung der Benutzer zuständig ist.

Die fachlichen Services sind in diesem Beispiel als RMI-Objekte realisiert. Die einzelnen Teile des Helpdesks rufen über RMI die Methoden an den Services auf [Farley 98]. Andere technische Realisierungen sind möglich, da die fachlichen Services zu diesem Thema keine speziellen Anforderungen stellen[11]. Als Persistenzmedium wird eine Registratur verwendet, die als Komponente aus dem JWAM Framework stammt [Havenstein 99]. Auch hier kann eine andere technische Realisierung benutzt werden, z.B. eine relationale Datenbank. Eine Implementierung, die eine relationale Datenbank benutzt, ist ebenfalls vorhanden. Die Klienten des fachlichen Service erfahren nichts über das Persistenzmedium. Es ist für den fachlichen Umgang mit dem Service nicht von Bedeutung. Eine ausführliche Beschreibung von fachlichen Services kann in der Diplomarbeit von Michael Otto und Norbert Schuler nachgelesen werden.

3.2.2 WAM-Werkzeuge (Desktop)

Anhand der Ist- und der Sollbeschreibungen zu dem Hilfesystem (Szenarien, Glossar, Ablaufvisionen, Werkzeugvisionen) wurden klassische WAM-Werkzeuge konstruiert. Diese sind mit Hilfe des JWAM Frameworks implementiert, wobei die Werkzeugkonstruktion intensiv genutzt wurde. Die Werkzeuge werden standardmäßig mit dem Desktop aus dem JWAM Framework gestartet [Lippert 99].

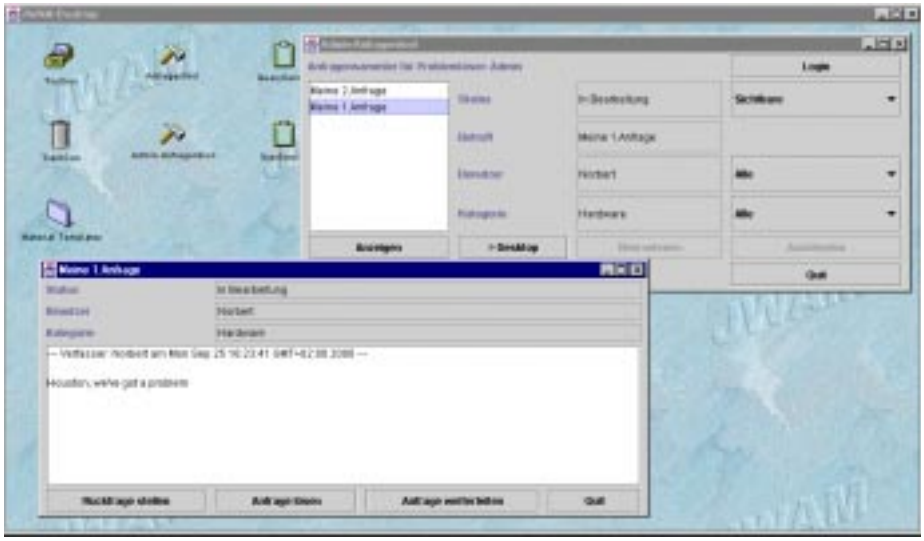

Abbildung 3-4: WAM-Werkzeuge auf dem Desktop

[11] Es existiert auch eine Lösung unter Verwendung von Enterprise Java Beans (EJB) [Otto/Schuler 00].

Diese WAM-Werkzeuge stellen die volle Funktionalität des Helpdesk zur Verfügung. Für Anwender und Problemlöser existieren zwei unterschiedliche Werkzeuge, die unterschiedliche Aktionen anbieten, aber vom Aufbau her ähnlich sind. Zustandsänderungen am Helpdesk werden sofort sichtbar, z.B. wenn eine neue Anfrage eingestellt wurde. Die Werkzeuge melden sich bei den beiden fachlichen Services an und werden über Änderungen informiert[12]. Diese Eigenschaft ist insbesondere für Problemlöser interessant.

Das Frontend für WAM-Werkzeuge muß auf einem Rechner installiert sein. Der Softwarecode der Werkzeuge und der des JWAM Frameworks muß sich auf dem Benutzerrechner befinden. Außerdem muß eine aktuelle Version einer Java VM vorhanden sein (mindestens JDK 1.2). Somit ist dieses Frontend, was das Softwaredeployment betrifft, nicht uneingeschränkt benutzbar. Es muß immer eine Installation der Anwendungssoftware auf dem Benutzerrechner vorgenommen werden oder der Softwarecode der gesamten Anwendung ist über ein verteiltes Filesystem zugänglich. Werden die fachlichen Services verändert und kommt es zu Änderungen an der Schnittstelle, so müssen auch die lokalen Installationen auf den Benutzerrechnern geändert werden.

3.2.3 Thin Client

Ein weiteres Frontend für den Helpdesk ist der schon im vorigen Kapitel angesprochene Thin Client. Bei diesem Frontend läuft nur die GUI eines Werkzeuges auf dem Benutzerrechner, während das Werkzeug selbst auf einem zentralen Server ausgeführt wird (Applikationsserver) Es muß für den Thin Client keine Veränderung an der Anwendungslogik der Werkzeuge vorgenommen werden. Da Werkzeuge in JWAM auf die GUI über eine Abstraktionsebene[13] zugreifen, kann die bisherige, lokale GUI-Konstruktion durch eine verteilte GUI-Konstruktion ohne Änderungen an der Anwendungslogik ersetzt werden.

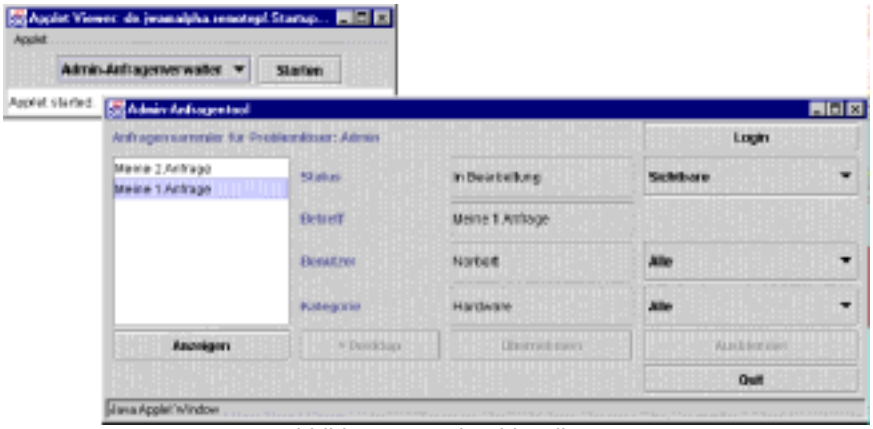

Abbildung 3-5: Ein Thin Client

Da bei diesem Frontend nur die GUI eines klassischen WAM-Werkzeuges verteilt wird, ist der Umgang exakt der gleiche wie bei einem lokalen, auf dem Benutzerrechner ablaufenden WAM-

[12] Für die bisherigen Implementierungen wurde der Message Broker aus dem JWAM Framework verwendet, der eine Verteilung von Nachrichten über Rechner ermöglicht [FLLRW 98][JWAM]
[13] IAF/PF Trennung, siehe 5.1.1

Werkzeug. Daher ist zwischen den beiden letzten Abbildungen (3-5 und 3-4) auch kein Unterschied an den Werkzeugen zu erkennen, wohl aber an der Umgebung in der die Werkzeuge ablaufen. Der Thin Client wird auf dem Benutzerrechner als Java-Applet gestartet. Dieses Applet ist mit dem Server[14] verbunden und erfragt beim Start am Server eine Liste der verfügbaren Werkzeuge. Wird ein angebotenes Werkzeug auf dem Server gestartet, so wird die GUI dieses Werkzeuges auf dem Benutzerrechner aufgebaut. Die GUI läuft innerhalb desselben Applets ab. Auf dem Server läuft eine virtuelle Maschine, in der eventuell mehrere Benutzer mit ihren Werkzeugen arbeiten. Anwendungsspezifische Software ist in der Regel nicht auf dem Benutzerrechner installiert, jedoch gibt es dazu einige Ausnahmen (siehe 5.5.2). Sofern dieser anwendungsspezifische Softwarecode auf den Benutzerrechner gebracht werden muß, geschieht dieses automatisch. Anwendungsspezifischer Code und der Softwarecode des Thin Clients selbst werden vom Server bezogen, so daß keine Installation auf den Benutzerrechner vorgenommen werden muß (Einfaches Softwaredeployment).

Um eine Anwendung als Thin Client zu verteilen, braucht der Anwendungsentwickler für dieses Frontend keine spezielle Entwicklungsarbeit zu leisten (Transparenz). Man entwickelt seine Werkzeuge und kann sie dann ohne Änderung an der Anwendungslogik selbst auf alle möglichen Benutzerrechner verteilen. Der Helpdesk ist damit von überall her zu benutzen. Der Benutzerrechner muß nur über eine VM für Java verfügen. Durch die Benutzung von Java-Applets kann der Thin Client auch von einem Browser aus aufgerufen werden.[15]

3.2.4 WWW Frontend (Webtop)

Der Webtop ist ein HTML-Frontend: Informationen aus dem Helpdesk werden über dynamisch erzeugte HTML-Seiten dargestellt. Anfragen an den Helpdesk finden über HTML-Formulare statt. Dieses Frontend läßt sich einfach über einen HTML-Browser bedienen, eine weitere Installation von Software ist nicht nötig.

[14] Dazu wird RMI als Middleware verwendet

[15] Dazu muß mindestens das JDK 1.2 vorhanden sein. Sofern ein Browser das nicht unterstützt, kann über ein Java-Plugin auf die lokale VM eines aktuellen JDKs zugegriffen werden.

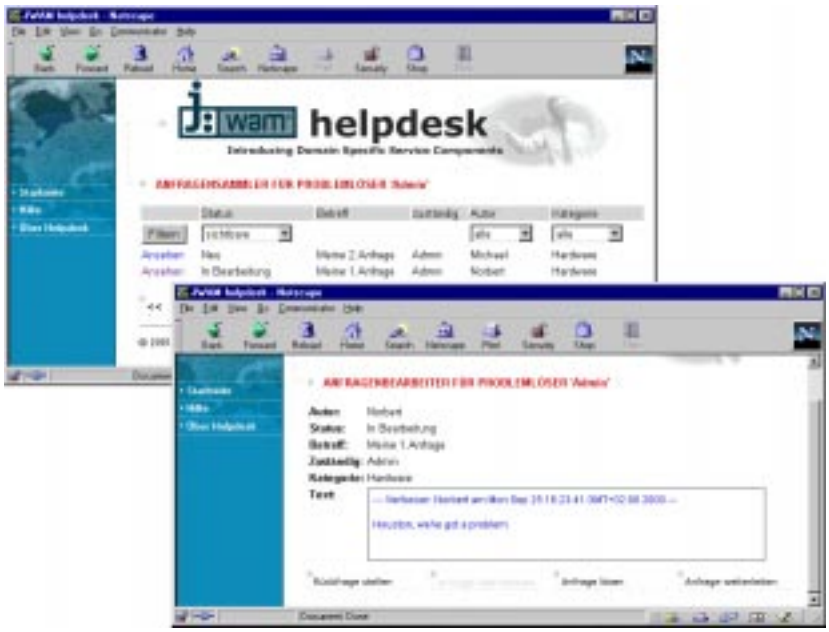

Abbildung 3-6: Webtop

Auch dieses Frontend bietet die volle Funktionalität im Umgang mit dem Helpdesk (vergleiche dazu die Abbildungen 3-4 und 3-5). Allerdings ist eine direkte Rückmeldung von Ereignissen an den Benutzerrechner nicht möglich. Änderungen an bisherigen Anfragen und neue Anfragen werden erst dann sichtbar, wenn die Übersichtsseite neu geladen wird. Durch einen Steuercode in der HTML-Seite wird aber das Aktualisieren der Seite nach einem kurzen Zeitintervall erzwungen, so daß immer ein aktueller Zustand der Anfragen sichtbar ist.

Dieses Frontend für den Helpdesk mußte komplett neu entwickelt werden. Aus der Konstruktion der WAM-Werkzeuge konnte keine Funktionalität für die Oberflächengestaltung verwendet werden. Allerdings wurde durch die Benutzung der fachlichen Services keine fachliche Funktionalität für die unterschiedlichen Frontends doppelt entwickelt. Durch die Verwendung von Java Servlets [Hunter/Crawford 98] konnte auf die fachlichen Services einfach zugegriffen werden, da beide Teil in Java entwickelt wurden.

Bei der Konstruktion dieses Frontend zeigte sich, daß die technischen Gegebenheiten von HTML die Konstruktion komplizierter gestalteten. Wählt der Benutzer bei einem der Filter im Anfragensammler einen neuen Eintrag aus, so muß der Server durch Absenden eines HTML-Formulars davon benachrichtigt werden. Durch eine solche Aktion wird die HTML-Seite komplett neu zusammengesetzt und zum Benutzerrechner übermittelt. Das Response-Request Verhalten von HTML bietet keine anderen Konstruktionsmöglichkeit. Das Generieren von HTML-Seiten ist dabei aufwendig. In der HTML-Seite müssen zahlreiche HTML-Steuercodes mit den eigentlichen Informationen vermengt werden, um eine, wie im obigen Beispiel annehmbare Präsentation zu erhalten.

Das HTML-Frontend ist durch die reine Darstellung von HTML besser von einem beliebigen Benutzerrechner zu erreichen als der Thin Client. Damit eignet es sich besonders für den Anwender des Helpdesk, da dieses Frontend die geringsten technischen Anforderungen stellt.

3.2.5 Waptop, Mailtop & SMS

Eine Anbindung an ein Mobiltelefon bietet optimale Erreichbarkeit. Über eine SMS-Schnittstelle (Short Message Service) können neue Anfragen als SMS-Kurznachricht an ein Mobiltelefon versendet werden. Eine weitere Zugriffsart bieten Mobiltelefone an, die WAP (Wireless Application Protocol) unterstützen: Über die Beschreibungssprache WML (Wireless Markup Language) können ähnlich wie bei HTML Daten auf dem Mobiltelefon dargestellt werden, das als Browser die WML-Seiten anzeigt [WAP]. Beim Wap-Frontend (Waptop) des Helpdesks können Anfragen in der Übersicht angezeigt und durch Auswahl en détail betrachtet werden.

Abbildung 3-7: WAP-Schnittstelle

Die SMS-Anbindung bietet nur eine Funktionalität: Benachrichtigung eines Problemlösers. Das SMS-Gateway meldet sich dazu bei dem Anfragen-Service an, um über Änderungen informiert zu werden. Falls eine neue Anfrage eingegeben wird, werden alle relevanten Informationen dieser Anfrage als SMS versendet. Die technische Begrenzung ist hier groß: So können die Nachrichten nur eine maximale Länge von 160 Zeichen haben.

Ein Mobiltelefon mit WAP-Funktionalität bietet mehr Möglichkeiten als eine SMS-Benachrichtigung: Informationen können als Text und Grafik in prinzipiell beliebiger Länge dargestellt werden. Aber auch hier gibt es durch die Größe des Displays und die Geschwindigkeit der Verbindung[16] technische Begrenzungen.

WML ist an HTML angelehnt und bietet ähnliche Möglichkeiten, unter anderem auch das Verknüpfen von Seiten über Links. Für den Helpdesk werden die WML-Seiten wie HTML-Seiten über Java-Servlets erzeugt. Zur Konstruktion des WML-Frontends konnten Teile des HTML-Frontends wiederverwendet werden, da sich nur der Inhalt der Präsentation verändert hat, nicht aber die grundlegende Art der Präsentation (vergleiche dazu den Unterschied zwischen Webtop und Desktop).

Die Eingabe von Texten über die Tastatur eines Mobiltelefons ist mühselig, so daß sich das Waptop für den Helpdesk auf die Anzeige von Informationen beschränkt. Ein Problemlöser kann

[16] Üblich sind 9600 Baud in GSM-Netzen

mit einem WAP-Mobiltelefon von jedem beliebigen Ort aus Zugriff auf den Helpdesk bekommen, um sich über den Stand von Anfragen genauer zu informieren.

Ein weiteres Frontend für den Helpdesk mit voller Funktionalität wäre ein Mailtop, der aber bislang nicht realisiert wurde. Die Analyse der Umgebung des bisherigen Hilfesystems ergab, daß Anwender einem Problemlöser Anfragen auch per E-Mail zusenden. Der Helpdesk sollte auch eine solche Möglichkeit unterstützen, um eine bestehende Infrastruktur einzubinden. Neue Anfragen würden über E-Mail an eine zentrale Stelle geschickt werden, Änderungen am Zustand der Anfragen oder Rückfragen per E-Mail dem Anwender zugestellt. Ein Mailtop ist für Problemlöser und Anwender prinzipiell interaktiv. Änderungen am Zustand von Anfragen werden durch das sofortige Versenden von E-Mails allen Beteiligten mitgeteilt.

3.3 Frontends und Benutzer

Für Anwender und Problemlöser gibt es beim Helpdesk verschiedene Frontends, die für sie und ihre Arbeitssituation unterschiedlich geeignet sind. Ein wichtiger Punkt ist dabei immer die gute Erreichbarkeit des Helpdesks: Er soll von überall her zu bedienen und zu benutzen sein. Für den Anwender ist daher der Webtop geeignet: Der Helpdesk ist über einen HTML-Browser zu erreichen und stellt damit an den Benutzerrechner wenig Anforderungen. Problemlöser können über Frontends fürs Handy (Waptop, SMS-Benachrichtigung) jederzeit über den aktuellen Stand informiert werden. Für sie kommt auch der Desktop in Frage, der auf ihrem Arbeitsplatz installiert ist. Im Vergleich zu der Anzahl der Arbeitsplätze der Anwender sind das wenige Benutzerrechner, auf denen ein solches System gepflegt werden muß.

Der Thin Client schließt dabei eine Lücke in den Frontends zum Helpdesk: Ein Problemlöser kann sein Werkzeug nun von überall her benutzen, er muß nicht mehr auf eine lokale Installation auf seinem Benutzerrechner zurückgreifen. Anwender hingegen erhalten durch den Thin Client ein Zugriff auf WAM-Werkzeuge, ohne sie auf ihren Benutzerrechner installieren zu müssen. Sie bedienen dieses Werkzeug über eine GUI und werden im Gegensatz zum Webtop auch über Änderungen an den Anfragen sofort informiert.

So kann möglicherweise die Konstruktion eines Webtops entfallen. Zwar benötigt ein Thin Client höhere technische Voraussetzungen als ein Webtop, was die Möglichkeiten der Verbreitung einschränkt. Dafür reicht aber die Konstruktion von WAM-Werkzeugen für den Desktop aus. Sie werden später ohne Änderungen in der Anwendungslogik als Thin Clients verfügbar sein.

4 Thin Client Konstruktion

In diesem Kapitel beschreibe ich zuerst die grundlegende Architektur eines Thin Clients, indem ich die allgemeine Konzeption aufzeige. Danach stelle ich zwei Technologien vor: den Ultra Light Client (ULC) und den JWAM Thin Client. Den JWAM Thin Client beschreibe ich hier nur überblicksmäßig. In Kapitel 5 gehe ich im Detail auf die Konstruktion ein. Anschließend greife ich die in 2.2 genannten Anforderungen für einen C/S-Schnitt auf und betrachte damit die beiden Architekturen. Abschließend werde ich die Integration von ULC in das JWAM Framework beschreiben und einige konzeptuelle Probleme darstellen, deren Lösung ich erst im nächsten Kapitel angehen werde.

4.1 Grundlegende Architektur eines Thin Clients

Ist eine Anwendung als Thin Client realisiert, läuft die Oberfläche (GUI) auf dem Benutzerrechner und der Rest der Anwendung auf einem Server ab. Im folgenden gehe ich auf drei grundlegende Punkte ein, die eine Architektur eines Thin Clients ausmacht: Was muß getan werden, um eine GUI auf einen Benutzerrechner zu bringen, wie sieht ein geeignetes Kommunikationsprotokoll aus und wie laufen die Anwendungen für Benutzerrechner auf einem zentralen Server ab.

4.1.1 Trennung der Oberflächenelemente

Der Schnitt bei einem Thin Client geht durch die Oberflächenelemente einer Anwendung hindurch. Die Anwendung benutzt direkt diese Element, auch Widgets genannt. Jedes unterschiedliche Widget (Buttons, Auswahllisten, Eingabefelder) bietet über eine Schnittstelle einen passenden Umgang an. Die Anwendung besitzt somit Wissen über die Widgets, da sie diese direkt benutzen muß[17]. In objektorientierten Sprachen sind diese Widgets als einzelne Objekte realisiert: GUI-Objekte.

Für einen Schnitt zwischen Benutzerrechner und Server müssen die GUI-Objekte aufgeteilt werden: Die Präsentation der GUI-Objekte befindet sich auf dem Benutzerrechner, während auf dem Server ein Teil zur Ansteuerung der GUI-Objekte durch die Anwendung verbleiben muß. Dieser Teil wird durch ein Proxy-Objekt realisiert:

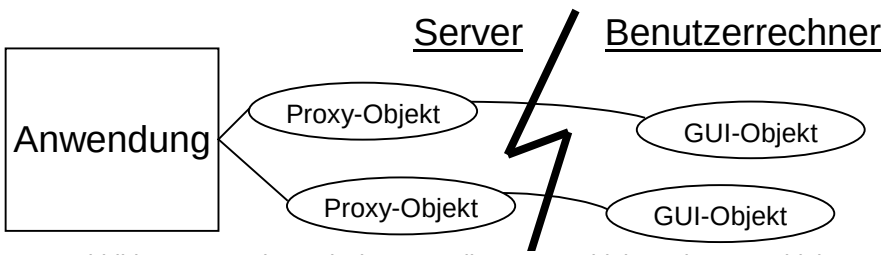

Abbildung 4-1: Schematische Darstellung GUI-Objekt und Proxy-Objekt

[17] Bei ULC und dem JWAM Thin Client wird durch unterschiedliche Methoden versucht, diese Abhängigkeit so gering wie möglich zu halten.

Die Anwendung benutzt die Proxy-Objekte, um die GUI zu manipulieren. Die GUI-Objekte bleiben der Anwendung verborgen, sie greift nie direkt auf diese Objekte zu. Wird eine Methode an einem Proxy-Objekt aufgerufen, so wird dieser Aufruf über ein Protokoll an das korrespondierende GUI-Objekt übertragen und gegebenenfalls ein Wert zurückgeben, der vom Benutzerrechner zum Server übermittelt werden muß.

Soll beispielsweise in einem Texteingabefeld durch die Anwendung ein Text angezeigt werden, wird über das Proxy-Objekt der entsprechende Text an das GUI-Objekt weitergereicht. Das GUI-Objekt auf dem Benutzerrechner ist damit für das Anzeigen des Textes auf dem Bildschirm zuständig. Ändert der Benutzer den Text in einem solchen Texteingabefeld, findet die Bearbeitung dieses Vorganges (das Zeichnen der Grafik) ausschließlich auf dem Benutzerrechner statt. Die GUI-Objekte auf dem Benutzerrechner erledigen die grundlegenden Aufgaben einer grafischen Benutzeroberfläche.

Da die GUI-Objekte immer über die Proxy-Objekte manipuliert werden, kann der komplette Zustand der Oberfläche auch in den Proxy-Objekten gespeichert werden, sofern man die aktuellen Veränderungen an der Oberfläche durch den Benutzer nicht berücksichtigt. Die Anwendung auf dem Benutzerrechner kann als zustandslos betrachtet werden, da sich der Zustand der GUI in den Proxy-Objekten auf dem Server befindet. Somit ist es prinzipiell möglich, die GUI nach einem Verbindungsabbruch oder einem Absturz der Anwendung auf dem Benutzerrechner zu rekonstruieren.

4.1.2 Kommunikationsprotokoll

Für die Kommunikation zwischen GUI-Objekt und dem dazugehörigen Proxy-Objekt muß ein Protokoll entwickelt werden. Eine simple Lösung wäre, diese beiden Objekte einfach als verteilte Objekte zu betrachten und ein geeignetes technisches Protokoll für entfernte Methodenaufrufe zu nutzen, beispielsweise RMI oder CORBA [Orfali 99]. Betrachtet man aber die Art, wie zwischen beiden Objekten Informationen ausgetauscht werden, kann die Kommunikation optimiert werden, so daß trotz einer möglicherweise langsamen Verbindung zwischen Benutzerrechner und Server die Anwendung flüssig zu bedienen bleibt.

Eine Anwendung manipuliert die Oberflächenelemente über Proxy-Objekte. Eine Manipulation eines Oberflächenelementes kann durchaus verzögert auf den Benutzerrechner übertragen werden, womit eine optimale Ausnutzung des Kommunikationskanals möglich ist. Werden die Daten durch einen nebenläufigen Prozeß versendet, wird die Anwendung nicht blockiert, solange der Kommunikationskanal blockiert ist. Änderungen an Oberflächenelementen werden solange zwischengespeichert, bis der Kommunikationskanal wieder frei wird. Die zwischengespeicherten Daten werden dann so schnell wie möglich übertragen.

Meistens wird nicht nur ein einzelnes Oberflächenelement durch irgendeine Aktion verändert, sondern mehrere Elementen hintereinander. Diese Manipulationen können daher auf dem Server gesammelt und dann als Block verschickt werden. Sofern sichergestellt ist, daß die Daten spätestens nach Bruchteilen einer Sekunde versendet werden, wird der Benutzer diese Verzögerung nicht bemerken. Durch das Gruppieren der Daten muß nicht für jede einzelne

Manipulation eine Kommunikation initialisiert werden, der Aufwand der Kommunikation verringert sich.

Sofern der Zustand von Oberflächenelementen abgefragt wird, muß die Serveranwendung auf eine Antwort des Benutzerrechners warten, um weiter fortfahren zu können. Speichern die Proxy-Objekte die an den GUI-Objekten einmal ermittelten Werte, kann bei einer erneuten Abfrage dieser zwischengespeicherte Wert zurückgegeben werden. Damit entfällt eine Kommunikation zum Benutzerrechner, falls in einer Anwendung ein Wert mehrfach hintereinander abgefragt wird.

Löst der Benutzer eine Aktion an der Oberfläche aus (z.B. durch Drücken eines Buttons), muß darauf reagiert werden. In einem synchronen Übertragungsmodell kommt der Kontrollfluß an die GUI erst dann zurück, wenn die Aktion übertragen, auf dem Server bearbeitet und die GUI aktualisiert wurde. Solange der Vorgang bearbeitet wird, ist die GUI blockiert. Es können keine Aktionen an ihr ausgeführt werden. So kann beispielsweise nicht in einer Tabelle geblättert oder an anderer Stelle irgendetwas eingegeben werden.

Eine Abhilfe schafft auch hier eine asynchrone Übertragung der Daten durch einen nebenläufigen Prozeß. Die Oberfläche kann weiter vom Benutzer bedient werden, solange die Aktion auf dem Server bearbeitet wird. Dieses Verhalten birgt aber eine Gefahr in sich: Während eine Aktion übertragen oder bearbeitet wird, kann der Benutzer die GUI verändern. Abfragen von Oberflächenelementen durch die Anwendung können somit unterschiedliche Werte zurück liefern (siehe dazu auch 5.5).

4.1.3 Der Applikationsserver

Die Anwendungen eines Thin Clients laufen auf einem Server ab, der damit zu einem Applikationsserver wird. Der Server bietet als Dienst Anwendungen (Applikationen) an, auf die über einen Thin Client zugegriffen wird. Damit läuft bei vielen aktiven Benutzern auf einem Server durchaus eine große Anzahl von Anwendungen ab, die Speicherressourcen und Rechenzeit verbrauchen. Die Anwendungen verbrauchen zumindest immer nur dann Rechenzeit, wenn an der Oberfläche durch den Benutzer ein Ereignis ausgelöst wurde.

Jede Anwendung könnte als eigener Betriebssystemprozeß realisiert werden, womit die Anwendungen nebenläufig agieren würden. Jeder Betriebssystemprozeß erzeugt aber einen Overhead. So müßte beispielsweise bei Java-Anwendungen für jeden Betriebssystemprozeß eine VM gestartet werden. Da die Anwendungen bei einem Thin Client Proxy-Objekte benutzen, die durch eine Umgebung bereitgestellt werden müssen, kann diese Umgebung die Anwendungen auch in eigenen Prozessen ablaufen lassen, ohne das Betriebssystemprozesse gestartet werden müssen.

Dazu bietet beispielsweise die Programmiersprache Java eine Unterstützung an: In einer VM können mehrere nebenläufige Prozesse ablaufen, Threads genannt, die auf den Objekten einer VM arbeiten [Oaks/Wong 97]. Durch das Abwickeln von Anwendungen in Threads statt in eigenen Betriebssystemprozessen ist weniger Verwaltungsaufwand auf dem Server nötig. Die laufenden Anwendungen werden prinzipiell kleiner gehalten, womit mehr Anwendungen auf

einem Server laufen können als bei Betriebssystemprozessen. Außerdem können bei dieser Konstruktion die Anwendungen prinzipiell auf gemeinsame Objekte zugreifen. Es kann durch diese gemeinsame Nutzung ein Synergie-Effekt eintreten. Da allerdings nebenläufige Threads benutzt werden, muß bei der Anwendungsentwicklung darauf geachtet werden, daß es durch die Nebenläufigkeit zu Problemen kommen kann. In 5.2 werde ich für den JWAM Thin Client beispielhaft auf diese Problematik eingehen.

4.1.4 Vergleich zu einer grafischen Verteilung

Die Architektur eines Thin Client ähnelt der von Verteilungssystemen auf grafischer Basis. Jedoch gibt es einige Unterschiede, durch die ein Thin Client eine besser Verteilung erzielt:

- Da bei einem Thin Client auf dem Benutzerrechner eine GUI abläuft, wird das Zeichnen der GUI komplett auf dem Benutzerrechner erledigt. Im Vergleich zu einer grafischen Verteilung ist dieses performanter, da nicht für jede Änderung an der GUI eine Kommunikation mit einer Serveranwendung stattfinden muß. Ändert beispielsweise der Benutzer bei einem grafischen Verteilungssystem den Text in einem Eingabefeld, so muß über eine Kommunikation mit der Serveranwendung die Darstellung bei jeder Tastatureingabe aktualisiert werden.

- Da bei einem Thin Client im Vergleich zu einer grafischen Verteilung keine Zeichenkommandos oder Ereignisse von Eingabegeräten zwischen Benutzerrechner und Server versendet werden, ist die Kommunikation abstrakter und damit optimaler. Soll beispielsweise ein neuer Text gezeichnet werden, wird nur dieser Text an das GUI-Objekt auf dem Benutzerrechner übermittelt und nicht eine Vielzahl von Zeichenkommandos.

- Bei grafischen Verteilungssystemen laufen die Anwendungen in der Regel als Betriebssystemprozesse ab. Bei Thin Clients ist es möglich, daß die einzelnen Anwendungen als Prozesse (Threads) innerhalb eines Betriebssystemprozesses ausgeführt werden, wodurch weniger Ressourcen in Anspruch genommen werden.

4.2 Thin Client Technologien

Im folgenden möchte ich zwei Technologien vorstellen, mit denen sich Thin Clients realisieren lassen: den Ultra Light Client, kurz ULC [Canoo 00][MTZ 98] und den JWAM Thin Client. Auf den JWAM Thin Client werde ich in Kapitel 5 noch genau eingehen. Ich werde in dem folgenden Abschnitt nur einen kurzen Überblick geben.

4.2.1 ULC

ULC ist eine GUI-Bibliothek, die von einer Java-Anwendung benutzt wird. In der folgenden Abbildung 4-2 ist der Schnitt zwischen Benutzerrechner und Server für ULC exemplarisch aufgezeichnet (Presentation Server), wobei vergleichend eine grafische Verteilung (Display Server) und ein Fat Client dargestellt sind. Das Anwendungssystem wird hier in drei Schichten aufgeteilt (vergleiche dazu 2.1): Der Business Layer ist ein zentraler Dienst des Anwendungssystems. Auf diesem setzen einzelne Anwendungen auf, deren Logik sich im

Application Layer befindet, während im Presentation Layer die Darstellung für den Benutzerrechner erledigt wird.

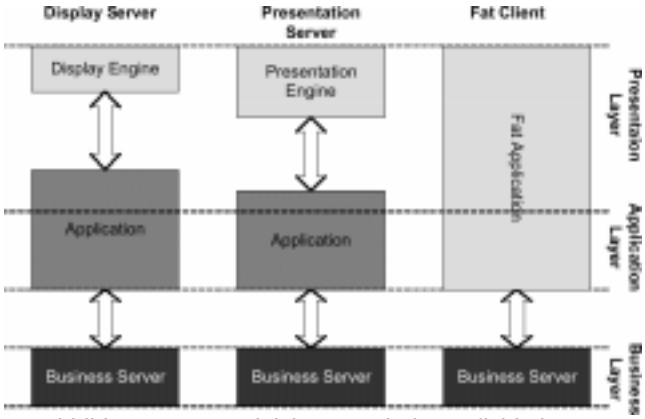

Abbildung 4-2: Vergleich von Schnittmöglichkeiten zu ULC (mittlere Darstellung), Quelle: Canoo Engineering AG

Ein Fat Client befindet sich komplett auf dem Benutzerrechner und setzt mit seiner Anwendungs- und Präsentationslogik auf dem Business Layer auf. Bei einer grafischen Verteilung befindet sich nur eine Anzeigemaschine (Display Engine) auf dem Benutzerrechner. Ein großer Teil der Präsentation aus dem Presentation Layer ist ein Bestandteil der Anwendung. So befinden sich z.B. bei einem X-System alle Oberflächenelemente auf dem Server und sind Teil der Anwendung auf diesem Rechner.

Bei ULC befinden sich nun die Oberflächenelemente auf dem Benutzerrechner, die durch die **Presentation Engine** verwaltet und angezeigt werden. Der Begriff Presentation Engine ist hier an den Begriff Display Engine aus der grafische Verteilung angelehnt: Auf dem Benutzerrechner bietet die Presentation Engine die Darstellung einer Präsentation als Dienst an, der von der Anwendung als Klient benutzt wird. Da auf dem Benutzerrechner die Oberfläche einer Anwendung abläuft, befindet sich ein größerer Teil aus dem Presentation Layer auf dem Benutzerrechner als bei einer grafischen Verteilung.

Bei ULC heißt die Presentation Engine **UI Engine** und kann als Java-Applikation oder als unsigniertes Java-Applet in einem Browser ausgeführt werden, wobei das JDK 1.2 vorausgesetzt werden muß. Auf dem Benutzerrechner wird nie anwendungsspezifischer Code installiert (vergleiche dazu später den JWAM Thin Client in 4.2.2), so daß eine UI Engine als Java-Applikation einmal auf einem Benutzerrechner installiert werden kann, die dann die GUIs aller möglichen ULC-Anwendungen auf einem Benutzerrechner darstellt.

4.2.1.1 Halbobjekte

Bei ULC wird die Aufteilung der Oberflächenelemente in GUI-Objekte und dazugehörige Proxy-Objekte durch sogenannte **Halbobjekte** realisiert (siehe Abbildung 4-3). Auf dem Benutzerrechner befinden sich die **UI Half Objects**, die GUI-Objekte. Dabei handelt es sich um Widgets aus der Swing-Bibliothek, die auf dem Benutzerrechner verwendet werden.

Auf dem Server benutzt die Anwendung sogenannte **Faceless Half Objects**, die Proxy-Objekte. Sie bieten von der Schnittstelle her einen Umgang an, als ob es sich um

Oberflächenelemente handeln würde. Sie werden ‚gesichtslos' genannt, da sich ihre Präsentation auf dem Benutzerrechner befindet. Die Schnittstelle der Proxy-Objekte auf dem Server ist an Swing orientiert, so daß eine Umstellung einer Swing-Anwendung auf eine ULC-Anwendung problemlos möglich ist. Es müssen nur an stelle der Swing-Widgets die ULC-Widgets verwendet werden.

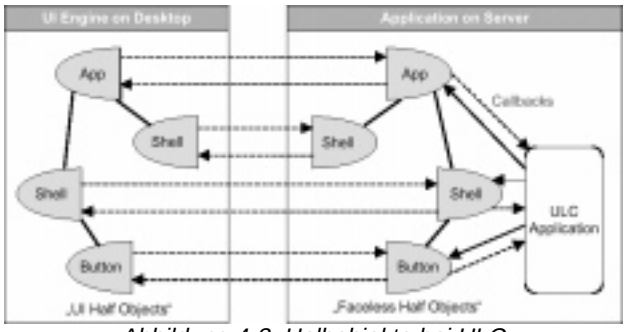

Abbildung 4-3: Halbobjekte bei ULC
Quelle: Canoo Engineering AG

Die Elemente sind in einer Hierarchie angeordnet, wie es bei GUIs üblich ist. In einem Verwaltungsobjekt für die laufende Applikation (App) können sich mehrere Fenster (Shell) befinden. In diesen Fenstern sind dann die Oberflächenelemente angeordnet, in der Abbildung beispielsweise ein Button.

Die Anwendung nutzt die Faceless Half Objects auf dem Server und wird über Aktionen an der Oberfläche durch einen Callback-Mechanismus informiert[18]. Die Halbobjekte auf Benutzerrechner und Server sind miteinander verbunden, wobei Informationen in beide Richtungen ausgetauscht werden.

4.2.1.2 Das Half-Object-Protokoll

Zwischen den Halbobjekten ist das sogenannte Half-Object-Protokoll etabliert. Über dieses Protokoll werden die Manipulationen von Oberflächenelementen und Abfragen über den Zustand der Elemente zwischen Benutzerrechner und Server übertragen. Die Kommunikation geschieht asynchron: Daten werden über nebenläufige Versandprozesse verschickt. Der Ablauf der GUI und des Anwendungsprogrammes wird möglichst nicht durch die Kommunikation behindert.

Für das Protokoll ist es nicht von Bedeutung, welche Middleware zum Verbinden von Benutzerrechner und Server verwendet wird. ULC bietet die Möglichkeit, das Protokoll auf eine beliebige Middleware aufzusetzen. Bislang ist eine Kommunikation über Sockets, SSL, HTTP und CORBA in ULC realisiert [Canoo 00]. Welches Protokoll verwendet wird, hängt auch von der Infrastruktur der Netzverbindung ab. Unter Umständen können Firewalls zwischen Benutzerrechner und Server nur bestimmte Kommunikationsprotokolle zulassen (siehe auch 2.3.1.3 zur Kommunikation zwischen Applets und Serveranwendung).

[18] Realisierung über Listener-Objekte. Diese Konstruktion wird auch in Swing verwendet [ELW 98]

Das Half-Object-Protokoll ist für niedrige Bandbreiten und hohe Latenzzeiten optimiert. So wird beispielsweise die Technik des **Lazy Loading** angewendet: Ein Tabelleneintrag kann unter Umständen viele hundert Einträge haben, die alle auf dem Benutzerrechner zur Darstellung bekannt sein müssen. Da der Benutzer aber immer nur einen Ausschnitt aus einer Tabelle sieht, werden auch nur die Daten dieses Ausschnittes vom Benutzerrechner angefordert und übertragen. Will der Benutzer andere Teile der Tabelle sehen, werden diese nach Bedarf (Lazy Loading) vom Benutzerrechner angefordert. Solange die Information über die neuen Tabellenteile noch nicht vorhanden sind, werden einfach leere Einträge angezeigt, die beim Eintreffen der Informationen automatisch mit den richtigen Daten gefüllt werden. Die GUI bleibt aufgrund des asynchronen Kommunikationsmodells weiter zu bedienen.

4.2.1.3 Optimierungsstrategie

Um den Kommunikationsaufwand weiter zu verringern, kann bei ULC auch ein Teil des Anwendungswissens auf den Benutzerrechner übertragen werden. Dieses geschieht in Form von Validatoren und Aktivatoren.

Ob ein Oberflächenelement aktiviert ist, d.h. durch den Benutzer zu bedienen, hängt oft vom Zustand der Oberfläche insgesamt ab. Ein Button zum Löschen eines Tabelleneintrags ist beispielsweise nur dann aktiviert, wenn die Tabelle überhaupt einen Eintrag besitzt und ein Element angewählt ist. Ob dieser Button dann aktiviert wird oder nicht, muß die Anwendungslogik auf dem Server immer dann entscheiden, wenn die Elemente einer Tabelle verändert werden. Außerdem muß die Serveranwendung die Selektierung in der Tabelle überwachen und gegebenenfalls darauf reagieren. Um diese Form der Kommunikation zu verhindern, gibt es das Konzept des Aktivators: Ein Oberflächenelement reagiert direkt auf eine Zustandsänderung eines anderen Oberflächenelementes durch Aktivierung oder Deaktivierung. Diese Reaktion findet vollständig auf dem Benutzerrechner statt, es muß keine Kommunikation mit dem Server stattfinden.

Validatoren bieten die syntaktische Überprüfung von Eingabefeldern an. Diese syntaktische Überprüfung findet ebenfalls auf dem Benutzerrechner statt, es muß keine Kommunikation mit der Serveranwendung initiiert werden. Ansonsten müßte die Serveranwendung jede Änderung in einem Eingabefeld überwachen, den Eingabetext überprüfen und entsprechend über die Oberfläche auf falsche Eingaben reagieren, indem beispielsweise der Eingabetext eine andere Farbe bekommt.

Bei Aktivatoren und Validatoren handelt es sich um spezielle Eigenschaften von ULC. Wird eine Anwendung, deren GUI bislang auf Swing basierte, auf ULC umgestellt, so können zusätzlich diese beiden Optimierungsstrategien verwendet werden. In diesem Fall kann eine Anwendung, die ULC benutzt, allerdings nicht ohne weiteres wieder auf Swing oder eine ähnliche GUI-Bibliothek umgestellt werden. Der Austausch der GUI, bislang durch die Ähnlichkeit der Schnittstellen zwischen Swing und ULC leicht möglich, gestaltet sich damit komplizierter.

4.2.2 Der JWAM Thin Client

Der JWAM Thin Client ist eine Technologie, mit der sich Anwendungen als Thin Clients verteilen lassen, die mit dem JWAM Framework entwickelt worden sind [JWAM]. Der JWAM Thin Client baut dabei auf das WAM-Entwurfsmuster der Trennung von Handhabung und Präsentation auf (siehe dazu auch 5.1.1, [Züllighoven 98][BLRSZ 99]), das eine Abstraktion über verwendete GUI-Bibliotheken liefert, die von einer Anwendung (einem Werkzeug) benutzt werden.

Dieser Abstraktionsmechanismus teilt eine GUI in zwei Teile: Es gibt einmal Präsentationsformen, die GUI-Objekte aus einer konkreten GUI-Bibliothek benutzen. Die Präsentationsformen bieten dabei ein standardisiertes Verhalten an, das über Interaktionsformen definiert ist. Auf Präsentationsformen wird über die Schnittstelle einer Interaktionsformen zugegriffen, die jeweils eine bestimmte Handhabung mit einer Präsentationsform anbietet. Ein Beispiel für eine Interaktionsform ist der Umgang mit einer Präsentationsform, die sich aktivieren läßt (z.B. ein Button), oder der Umgang mit einer Präsentationsform, die eine Selektierung eines Elementes ermöglicht (z.B. eine Auswahlliste). Eine Präsentationsform kann auch mehrere Interaktionsformen implementieren, wie beispielsweise bei einer Liste, deren Elemente einmal ausgewählt und einmal durch Doppelklick aktiviert werden können.

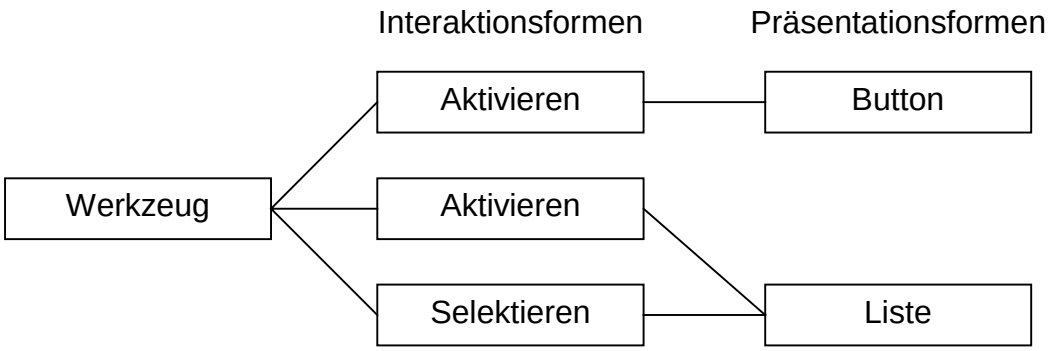

Abbildung 4-4 : Schematische Darstellung der Nutzung der Komponenten

Durch die Trennung von Interaktion und Präsentation arbeitet das Werkzeug nicht direkt mit den Widgets einer GUI, sondern immer mit allgemein gehaltenen Interaktionsformen. Für eine GUI-Bibliothek müssen eigene Präsentationsformen entwickelt werden, die dann gegen anderen Präsentationsformen anderer GUI-Bibliotheken ausgetauscht werden können. Die Anwendung selbst muß dabei nicht an eine neue GUI angepaßt werden.

Diese Trennung macht sich der JWAM Thin Client zu nutze: Die Präsentationsformen laufen auf dem Benutzerrechner ab, während spezielle Interaktionsformobjekte als Proxy-Objekte fungieren. Diese Interaktionsformobjekte implementieren ihre jeweilige Interaktionsform, so daß das Werkzeug wie gehabt auf die GUI zugreifen kann. Auf dem Benutzerrechner können

beliebige Präsentationsformen verteilt werden, sofern diese Swing oder AWT benutzen. Anwendungsspezifische Präsentationsformen[19] können ebenso verteilt werden.

Als technisches Protokoll zwischen Benutzerrechner und Server wird RMI benutzt, wobei die in 4.1.2 genannten Optimierungsmöglichkeiten implementiert sind. Die Kommunikation ist asynchron. Auf dem Benutzerrechner läuft ein Java-Applet ab, wobei der Softwarecode komplett vom Server geladen wird. Der Code der Präsentationsformen wird damit ebenfalls auf diese Art verteilt, so daß auf dem Benutzerrechner keine anwendungsspezifische Software vorinstalliert werden muß. Der Code von Präsentationsformen wird automatisch und ohne das Zutun des Benutzers auf den Benutzerrechner gebracht.

4.3 Anforderungen an ULC und JWAM Thin Client

In 2.2 habe ich vier Anforderungen aufgezählt, die für einen optimalen Schnitt einer Anwendung zwischen Benutzerrechner und Server wichtig sind und die von einer Technologie für eine solche Verteilung erfüllt werden sollten. In 2.3 und 2.4 habe ich schon einige Technologien anhand dieser Anforderungen untersucht. In diesem Kapitel betrachte ich, wie der JWAM Thin Client und ULC diese Anforderungen erfüllen. In jedem der folgenden Unterkapitel überprüfe ich zuerst ULC und dann den JWAM Thin Client anhand einer Anforderung, die ich auch in der Reihenfolge aus 2.2 hier darstelle.

4.3.1 Softwaredeployment

Um die GUI einer ULC-Anwendung auf dem Benutzerrechner darzustellen, muß nur eine UI Engine vorhanden sein. Diese kann vorinstalliert sein oder der Softwarecode wird durch die Konstruktion als Java-Applet über einen Browser geladen. Anwendungsspezifischer Code wird nicht verteilt, so daß kein Deployment-Problem auftritt. Der Benutzerrechner muß mit dem JDK 1.2 ausgestattet sein, was momentan nicht alle Betriebssysteme bieten [Javasoft 00]. Da eine Kommunikation zwischen Benutzerrechner und Server auch über das HTTP-Protokoll möglich ist, können Probleme mit Firewalls zwischen den beiden Rechnern umgangen werden. Die Anforderung eines optimalen Softwaredeployments ist für ULC gut erfüllt.

Der JWAM Thin Client ermöglicht die Benutzung anwendungsspezifischer Oberflächenelemente. Der Softwarecode dieser Elemente wird durch den JWAM Thin Client automatisch geladen, da dieser als Applet konstruiert ist. Es tritt somit kein Deployment-Problem auf, auch wenn sich hier anwendungsspezifischer Code auf dem Benutzerrechner befindet. Auch beim JWAM Thin Client muß ein JDK 1.2 auf dem Benutzerrechner installiert sein. Da RMI zwischen Benutzerrechner und Server als Middleware benutzt wird und sowohl Verbindungen vom Benutzerrechner zum Server als auch umgekehrt aufgebaut werden, können Firewalls die Kommunikation verhindern. Die Anforderung eines optimalen Softwaredeployments sind daher für den JWAM Thin Client weniger gut erfüllt.

[19] Es kann bei JWAM-Anwendungen durchaus vorkommen, daß vom Anwendungsentwickler auch eigene Präsentationsformen entwickelt werden.

4.3.2 Schnittstellenänderungen

Da nur die GUI bei ULC über ein generisches Protokoll auf den Benutzerrechner verteilt wird, gibt es keine anwendungsspezifische Schnittstelle zwischen Benutzerrechner und Server und damit keine Probleme bei Schnittstellenänderungen. Sofern das Protokoll zwischen Benutzerrechner und Server durch eine neue Version von ULC verändert wird, muß eine vorinstallierte UI Engine auf dem Benutzerrechner angepaßt werden.

Für den JWAM Thin Client gilt das eben genannte genauso, wobei es aber keine vorinstallierte Version durch die Applet-Konstruktion geben kann und daher dieser Punkt entfällt.

4.3.3 Skalierung & Performance

Bei ULC wird die GUI auf dem Benutzerrechner verwaltet, so das elementare Aufgaben wie das Zeichnen der Oberfläche oder die Positionierung eines Cursors auf dem Benutzerrechner erledigt werden. Durch Aktivatoren oder Validatoren werden weitere Aufgaben an den Benutzerrechner delegiert, deren Bearbeitung dort schnell erledigt werden kann. Es erfolgt nur eine Anfrage an den Server, wenn es zu anwendungsfachlichen Änderungen an der Oberfläche durch eine Aktion des Benutzers kommen muß. Somit wird die Last auf dem Server reduziert. Sofern doch zu viele Anwendungen gleichzeitig auf einem Server ablaufen, kann einfach ein weiterer Server mit den gleichen Anwendungen hinzugefügt werden, mit dem dann zukünftig ein Teil der Benutzer arbeitet soll. Eine Skalierung ist somit einfach möglich. Die Kommunikation zwischen Benutzerrechner und Server ist stark optimiert, so daß geringe Anforderungen an Bandbreite und Latenzzeit gestellt werden. So kann eine Kommunikation auf der Ebene von Sockets aufgebaut werden, womit eine schnellstmögliche Bearbeitung sichergestellt ist. Eine ULC-Anwendung bleibt somit auch dann flüssig zu bedienen, wenn zwischen Benutzerrechner und Server nur eine langsame Netzverbindung vorhanden ist.

Auch beim JWAM Thin Client werden die elementaren Aufgaben der GUI auf dem Benutzerrechner erledigt. Da anwendungsspezifische Präsentationsformen auf dem Benutzerrechner ablaufen, können auf dem Benutzerrechner schon spezielle Aufgaben der Anwendung erledigt werden, wie beispielsweise eine beliebige syntaktische Überprüfung einer Eingabe auf Korrektheit. Der Server wird damit auch beim JWAM Thin Client entlastet, es kommt nur zu Anfragen an den Server, wenn es anwendungsfachlich notwendig ist. Das System skaliert ebenso wie ULC durch Hinzufügen eines neuen Rechners. Die Kommunikation zwischen Benutzerrechner und Server ist ebenfalls optimiert. Da RMI als Middleware genutzt wird, gibt es aber einen Verwaltungs-Overhead, so daß im Vergleich zu ULC sich eine langsame Netzverbindung schneller bemerkbar macht.

4.3.4 Transparenz

ULC verhält sich gegenüber Fehlern genauso transparent wie jede andere Verteilungstechnologie: Kommt es zu Übertragungsfehlern, gehen Informationen verloren. Bricht die Verbindung zwischen Benutzerrechner und Server ab, so kann die Anwendung nicht restauriert werden, indem die GUI auf dem Benutzerrechner später wiederhergestellt wird. Es

gibt kein automatisches Fehlermanagement. ULC ist, was die Transparenz von der Middleware betrifft, besser ausgestattet. Die für den Anwendungsentwickler angebotenen Oberflächenelemente bei ULC sind von der Schnittstelle her an die Swing-Bibliothek angelehnt, die als Standard für die Entwicklung einer GUI in Java existiert [ELW 98]. Ein Austauschen von Swing und ULC ist leicht möglich, der Anwendungsentwickler muß seine Software kaum anpassen, sofern nicht erweiterte Konzepte von ULC wie Validatoren oder Aktivatoren genutzt werden sollen. Das technische Verteilungsprotokoll ist für den Anwendungsentwickler nicht von Bedeutung.

Der JWAM Thin Client benutzt einen Abstraktionsmechanismus über die GUI: die Trennung von Interaktion und Präsentation. Ein Austausch einer lokalen gegen eine verteile GUI ist hiermit möglich, da die Abhängigkeit zu einer konkreten GUI beseitigt ist. Somit läßt sich in JWAM eine existierende Anwendung ohne irgendeine Änderungen am Anwendungscode als Thin Client verteilen. Was die Transparenz gegenüber Fehlern betrifft, so verhält sich der JWAM Thin Client genauso wie ULC. Jedoch ist es prinzipiell möglich, eine GUI auf dem Benutzerrechner wieder anzuzeigen, wenn die Verbindung unterbrochen wurde (siehe 5.5.3). Wenn diese Eigenschaft realisiert würde, kann man von einer Fehlertransparenz des JWAM Thin Client sprechen.

4.4 Integration von ULC in JWAM

ULC an sich ist ‚nur' eine GUI-Bibliothek, die von einer Anwendung aus genutzt wird. Mit Hilfe der ULC-Bibliothek soll es möglich sein, eine bestehende Anwendung als Thin Client auf Benutzerrechnern zu verteilen, wie auch Anwendungen, die mit dem JWAM Framework nach dem WAM-Ansatz entwickelt wurden. Eine solche Integration von ULC in JWAM habe ich im Rahmen eines Praktikums bei der UBS Basel durchführen können [Bohlmann 98]. Dabei habe ich das JWAM Framework in der Version 1.3 verwendet, sowie eine frühe Testversion von ULC.

4.4.1 Einbettung der GUI-Bibliothek von ULC

Nach dem WAM-Entwurfsmuster der Trennung von Handhabung und Präsentation (siehe dazu auch 4.2.2. oder 5.1.1, [Züllighoven 98][BLRSZ 99]) konnte die ULC-Bibliothek in JWAM eingebettet werden. Es mußten für die Widgets aus ULC Präsentationsformen entwickelt werden, wobei die Strukturähnlichkeit von ULC und Swing hilfreich war. Widgets aus Swing sind vom Umgang her ähnlich zu verwenden wie Widgets aus dem AWT-Toolkit [Flanagan 99B], für das in JWAM Präsentationsformen vorhanden waren. Diese Präsentationsformen konnten als Grundlage für die Konstruktion von ULC-Präsentationsformen verwendet werden.

Die GUI einer Anwendung wird in JWAM in einem sogenannten GUI-Kontext verwaltet, in dem auch Wissen über die verwendete GUI-Bibliothek enthalten ist. Für die Anbindung von ULC mußte daher auch ein eigener Kontext entwickelt werden. Durch die Trennung von Handhabung und Präsentation konnte ULC als neue GUI-Bibliothek einfach eingebunden werden. Die Werkzeuge mußten nicht verändert werden, damit sie ULC-Präsentationsformen nutzen konnten[20]. Eine Ausnahme bilden dabei die GUI-Klassen: In einer solchen Klasse wird

[20] Bislang wurde Präsentationsformen für Swing und AWT benutzt

die GUI eines Werkzeuges erstellt, wobei Objekte von Präsentationsformen erzeugt und zu einer Oberfläche zusammengefügt werden. Statt der bisherigen Präsentationsformen müssen nun ULC-Präsentationsformen erzeugt werden, was aber durch eine einfache Umbenennung von Klassennamen erfolgen kann.

4.4.2 Offene Punkte der Konstruktion

Um ULC in JWAM nutzen zu können, muß nicht nur die GUI angebunden werden, sondern auch das Umfeld für die Anwendungen verändert werden. Die Konstruktion von ULC setzt voraus, daß auf einem Server mehrere Anwendungen in einer Java VM ablaufen. Jede Anwendung eines Benutzerrechners läuft in einem eigenständigen, nebenläufigen Prozeß. Das JWAM Framework ist aber so konstruiert, daß in einer Java VM genau eine Anwendung abläuft. Diese Annahme ist auch sinnvoll, da auf einem Benutzerrechner bisher eine Anwendung in einer Java VM lief, die von genau einem Benutzer bedient wurde.

Das JWAM Framework ist so konzipiert, daß nur ein nebenläufiger Prozeß (Thread) mit den Objekten der Anwendung arbeiten kann. Ein Thread kann beispielsweise durch ein Ereignis eines Oberflächenelementes oder eine Nachricht von einem anderen Rechner gestartet werden. Innerhalb eines solchen Threads wird das Ereignis durch die Anwendung bearbeitet. Konkurrieren zwei Threads um eine Anwendung, so muß ein Thread solange warten, bis der andere abgearbeitet wurde. Die Ereignisse werden somit serialisiert. Bei ULC existiert auf dem Server pro laufender Anwendung ein eigener Thread, der die Bearbeitung von Ereignissen für eine Anwendung übernimmt. Das JWAM Framework muß dahingehend erweitert werden, daß es mehrere, gleichzeitig aktive Anwendungen erlaubt, auf denen jeweils ein Thread arbeitet.

Ein weiterer Problemfall in diesem Zusammenhang ist das Entwurfsmuster Singleton [GHJV 94], welches im JWAM Framework verwendet wird. Ein Singleton ist eine Klasse, die nur die Erzeugung eines Objektes erlaubt. Soll ein weiteres Objekt erzeugt werden, wird das bereits erzeugte Objekt zurückgegeben. Globale Informationen werden über Singletons zur Verfügung gestellt, wie z.B. Informationen über den Benutzer einer Anwendung,.

Laufen auf einem Server in einer Java VM mehrere Anwendungen ab, dann müssen diese globalen Informationen nicht einmalig im gesamten System vorhanden sein, sondern einmal pro laufender Anwendung. Jede Anwendung kann beispielsweise durch einen unterschiedlichen Benutzer bedient werden, was ein Singleton zu Benutzerinformationen entsprechend berücksichtigen müßte. Bei einem Zugriff auf das Singleton muß erkannt werden, welche gerade aktive Anwendung Informationen abfragen möchte. Dementsprechend wird dann die passende Information zurückgeliefert.

Im folgenden Kapitel beschreibe ich die Konstruktion des JWAM Thin Client, für den ich die eben genannten, offenen Punkte gelöst habe, da sie auch für den JWAM Thin Client von Bedeutung sind. Mit Hilfe dieser Erweiterungen ist es dann möglich, ULC in JWAM vollständig zu integrieren. Bei der ersten Einbindung von ULC habe ich diese Probleme nur durch Zwischenlösungen bewältigt.

5 Der JWAM Thin Client

In diesem Kapitel gehe ich auf den technischen Aufbau des JWAM Thin Clients ein. Zuerst erläutere ich einige Konstruktionsprinzipien aus dem JWAM Framework (Version 1.5), welche für die Entwicklung eines Thin Clients von Bedeutung sind. Danach zeige ich, an welchen Stellen das JWAM Framework erweitert werden muß, um Anwendungen als Thin Clients laufen zu lassen. Zum Abschluß erläutere ich die genaue Konstruktion eines Thin Clients und stelle noch offene Punkte der Konstruktion dar.

5.1 Übersicht der Komponenten in JWAM

Um einen Thin Client mit dem JWAM Framework zu realisieren, mußte ich die GUI-Konstruktion erweitern. Da in JWAM eine Abstraktion von der GUI-Bibliothek möglich ist, kann ohne Änderung an den Werkzeugen eine neue GUI integriert werden, wie z.B. eine GUI für einen Thin Client (siehe auch 4.4 zur Integration der ULC-Bibliothek). Weiter habe ich, wie schon im vorigen Kapitel angedeutet, auch einige Änderungen am JWAM Framework selbst durchführen müssen. Um diese Änderungen besser erläutern zu können, stelle ich in diesem Unterkapitel ausgewählte Komponenten aus dem JWAM Framework unter dem Gesichtspunkt der Erweiterung durch einen Thin Client genauer dar. Für weitergehende Informationen über das JWAM Framework sei auf entsprechende Literatur verwiesen [JWAM].

5.1.1 Grundlagen der GUI-Konstruktion

Die Anbindung der GUI an das Werkzeuge und damit an die Anwendung kann im JWAM Framework auf zwei Arten geschehen: Entweder wird direkt mit GUI-Elementen aus einer GUI-Bibliothek gearbeitet (z.B. Swing) oder es wird über eine Abstraktionsschicht auf diese GUI-Elemente zugegriffen. Für die Konstruktion eines Thin Client ist nur die letzte Möglichkeit interessant, nämlich die Abstraktion von der GUI über Interaktions- und Präsentationsformen (siehe [BLRSZ 99] und zu aktuellen Realisierung [JWAM]). Mit Hilfe dieser Abstraktionsschicht konnte auch die ULC-Bibliothek leicht in JWAM eingebunden werden.

Die Elemente einer GUI sind nach ihrem Umgang über Interaktionsformen (Kurzform IAF) klassifiziert. So können beispielsweise aus einer Liste einmal Elemente ausgewählt oder durch Doppelklick aktiviert werden. Ein Button hingegen besitzt nur eine Interaktionsform, nämlich die eben genannte Aktivierung. Auf die GUI-Elemente wird von einem Werkzeug[21] nur über die Schnittstelle der Interaktionsform zugegriffen. Die Interaktionsformen sind in Interfaces definiert und bieten je nach Art der Interaktion einen entsprechenden Umgang an. So sind z.B. die Interaktionsformen für eine Liste, Selektierung und Aktivierung, in den Interfaces `ifSingleSelection` und `ifActivator` definiert. Die Interaktionsformen bilden eine

[21] Ob es sich hierbei um ein Werkzeug handelt, das nach dem WAM-Ansatz in Funktions- und Interaktionskomponente (FP und IP) aufgeteilt ist, oder aber um ein Werkzeug, das beide Komponenten in einem Werkzeug-Objekt beinhaltet (Mono-Tool), ist für die weitere Betrachtung unwichtig.

Hierarchie und erben letztlich von dem Interface `ifInterface`, in dem der allgemeine Umgang mit GUI-Elementen definiert ist.

Die Präsentationsformen (Kurzform PF) implementieren die entsprechenden Interfaces und erben meistens von den konkreten GUI-Elementen einer Bibliothek. Sie erweitern somit die GUI-Elemente um den Umgang mit Interaktionsformen[22]. Alternativ können Präsentationsformen auch GUI-Elemente benutzen. In den Werkzeugen wird zwar eine Referenz auf die konkreten Präsentationsformen gehalten, aber nur über die Schnittstelle der Interaktionsform darauf zugegriffen. Der Zugriff auf die GUI aus einem Werkzeug hinaus ist durch die Verwendung der Interaktionsschnittstellen von der konkreten GUI-Bibliothek getrennt. Somit kann eine GUI-Bibliothek durch eine andere ersetzt werden, ohne daß die Werkzeuge selbst verändert werden müssen.

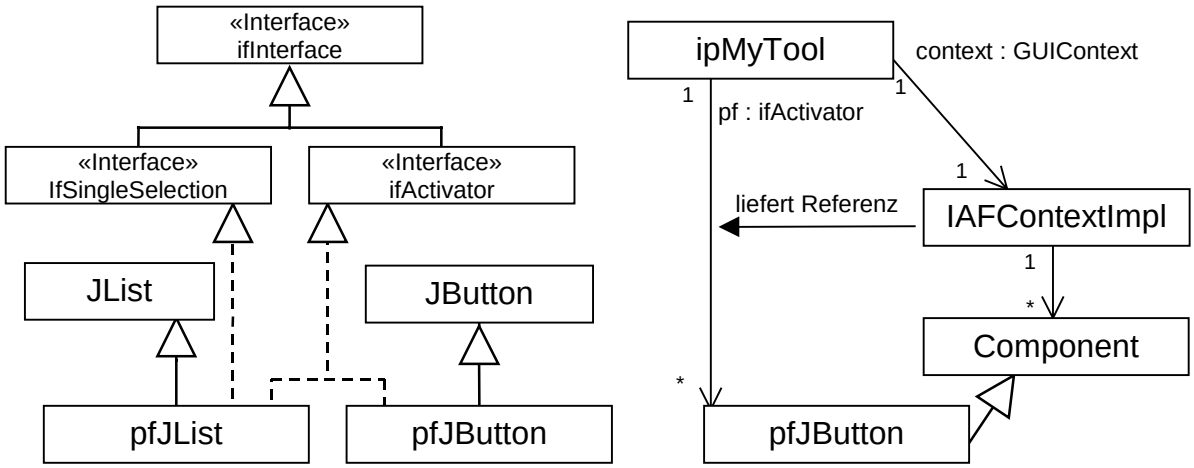

Abbildung 5-1: Vererbungsbeziehungen IAF-PF & Benutzt-Beziehungen zwischen den GUI-Objekten (hier mit einer IP)

Die GUI eines Werkzeuges wird in der Regel in einer GUI-Klasse implementiert. In dem Konstruktor einer solchen Klasse werden alle GUI-Objekte erzeugt und entsprechend positioniert. Die GUI-Editoren der meisten Java-Entwicklungsumgebungen erzeugen eine solche GUI-Klasse (Beispiel JBuilder [Inprise 00]) Um GUI-Elemente über Editoren zusammenzustellen, müssen diese als Java-Beans realisiert sein, damit alle Attribute der Elemente, wie Farbe oder Texteigenschaften, über eine einheitliche Schnittstelle verändert werden können (siehe [Englander 97]). Präsentationsformen erfüllen ebenfalls diese Anforderungen. Die Entwicklung einer GUI-Klasse ‚von Hand', d.h. durch das Schreiben von Softwarecode, ist ebenfalls möglich.

Die GUI für ein Werkzeug wird immer über einen GUI-Kontext bereitgestellt. Dieser Kontext wird dem Werkzeug übergeben und bietet in der allgemeinsten Form (Interface `GUIContext`) Funktionen zum Aktivieren/Deaktivieren der GUI und zum Einbetten von Sub-Werkzeugen, deren GUI in der GUI des Kontext-Werkzeuges eingegliedert ist. Der `IAFContextImpl`[23] ist ein spezialisierter Kontext, der Methoden zum Auffinden von Interaktionsformen anbietet. Der

[22] Seit Version 1.4 sind die Interaktionsformen als Interfaces definiert.
[23] Im Interface `IAFContext` ist der Umgang definiert, der in `IAFContextImpl` implementiert wird.

Kontext besitzt Wissen über die verwendete GUI-Bibliothek (siehe Abbildung 5-1: Der Kontext kennt die GUI-Objekte unter dem Typ `Component`). Für jede GUI-Bibliothek muß es prinzipiell einen eigenen Kontext geben[24]. Der Zugriff auf den Kontext ist wieder über Schnittstellen möglich, so daß von einer konkreten Implementierung abstrahiert werden kann.

```
// Erzeugt einen IAF-Kontext für eine GUI
IAFContext context;
context = new IAFContextImpl(new guiToolSwing());
```

In dem obigen Beispiel wird ein Objekt der Klasse `IAFContextImpl` erzeugt, das einen Zugriff auf ein `guiToolSwing`-Objekt (Objekt einer GUI-Klasse) bietet. Dieser Kontext wird dann von der Interaktionskomponente eines Werkzeuges benutzt, um eine Referenz auf die GUI-Objekte zu bekommen. Jede Interaktionsform einer Präsentationsform ist mit einem Namen versehen, der die Interaktion eindeutig identifiziert. Der Name der Interaktionsform wird wie ein Attribut an der jeweiligen Präsentationsform gesetzt.

```
// Sucht PF mit Interaktionsform Aktivator namens “okbutton”
ifActivator okButton = (ifActivator)context.interactionForm(
                       ifActivator.class, "okbutton");
```

Da der GUI-Kontext die enthaltene GUI-Bibliothek kennt und auch Wissen über den prinzipiellen Aufbau einer GUI hat, können die GUI-Elemente traversiert und nach dem passenden Element gesucht werden. Bei einer GUI mit Swing oder AWT ist dieses für den passenden Kontext recht einfach möglich. Die Elemente bilden untereinander eine Hierarchie, auf die über Methoden an den GUI-Objekten zugegriffen werden kann[25]. Da die Präsentationsformen die Interfaces der Interaktionsformen implementieren, kann an den Objekten direkt abgefragt werden, ob sie überhaupt eine gesuchte Interaktion implementieren. Über die Methode `ifName()`, die jede Präsentationsform aus `ifInterface` implementieren muß, kann der Name für eine gegebenen Interaktion ermittelt und verglichen werden.

Durch das Auffinden der Interaktionsformen über einen Namen und die Verwendung eines GUI-Kontextes ist die Anbindung der GUI an das Werkzeug soweit abstrahiert, daß im Werkzeug über die konkrete GUI keine Annahmen gemacht werden müssen. Der Zugriff auf die GUI-Elemente findet dann über die abstrakte Schnittstelle der Interaktionsform statt. Jetzt muß noch für die Rückmeldung der GUI-Elemente (z.B. durch das Drücken eines Knopfes) eine Reaktionsform entwickelt werden, die nicht von den Mechanismen der speziellen GUI-Bibliothek abhängig ist: Commands.

5.1.2 Benachrichtigung via Commands

Die Rückkopplung von Ereignissen der GUI zu einem Werkzeug findet über einen eigenen Benachrichtigungsmechanismus in JWAM statt: durch Commands. Ein Kommando hat genau

[24] Da die GUI-Elemente aus Swing von den Klassen aus AWT erben, kann der gleiche Kontext verwendet werden.
[25] Die ULC-Widget unterstützen beispielsweise nicht eine solche Vorgehensweise!

einen Sender und einen Empfänger, in diesem Fall das GUI-Objekt, welches das Werkzeug benachrichtigt[26]. Die jeweiligen Interaktionsformen bieten Methoden zum An- und Abmelden von Commands an. Für die unterschiedlichen Arten der Interaktion gibt es auch unterschiedliche Commands. An einem Aktivator, z.B. einem Button, meldet man sich mit `cmdActivate` an, um über die Aktivierung informiert zu werden. Für eine Selektierung eines Elementes aus einer Auswahlliste (`ifSingeSelection`) wäre dies ein `cmdSelect`. Die Kommandos bieten an ihrer Schnittstelle unterschiedliche Methoden an. So kann z.B. an einem `cmdSelect` der Index der selektierten Elements abgefragt werden. Alle Commands erben von der Oberklasse `cmdObject`. Im folgenden Beispiel wird an einer Interaktionsform (`okButton`) ein Activate-Command angemeldet:

```
okButton.attachActivateCommand(new cmdActivate()
    {
      public void doExecute ()
      {
        buttonOKPressed();
      }
    });
```

Die Klassen der Commands sind immer abstrakt. In einer konkreten Klasse muß die `doExecute()`-Methode überschrieben werden, die aufgerufen wird, sobald das Ereignis ausgelöst wird. In dem obigen Beispiel wird mit dem Aufruf einer Methode auf das Ereignis reagiert. Ein Exemplar dieser Klasse, hier eine anonyme Inner-Class [Flanagan 99A], wird an die Präsentationsform übergeben. Das Verwenden von anonymen Klassen ist hier typisch, wird aber später bei der Konstruktion des Thin Clients noch zu einem Problem führen.

Die Präsentationsformen selbst melden sich über die Methoden der GUI-Bibliothek für die passenden Ereignisse an. Bei einem Button aus der Swing-Bibliothek beispielsweise funktioniert die Anmeldung über ein Listener-Interface, wobei dort auch eine Methode implementiert werden muß, die bei Auslösung des Ereignisses aufgerufen wird. In der Präsentationsform wird dann das entsprechende Command ausgeführt und damit das Werkzeug informiert. Dieser Aufruf wird, um bei dem Beispiel des Swing-Button zu bleiben, durch einen Thread erledigt. Da durchaus noch andere Threads aktiv sein können, entsteht ein Problem durch nebenläufige Änderungen, das aber durch den Synchronizer gelöst wird.

5.1.3 Synchronizer

Innerhalb des JWAM Frameworks gibt es vier verschiedene Benachrichtigungsmechanismen: Requests, Events, Commands und Messages [JWAM]. Während die ersten beiden Mechanismen nur innerhalb des Kontrollflusses im Framework ausgelöst werden können, können Commands und Messages von ‚außerhalb' in das System hineinkommen. Commands werden ausgelöst, wenn ein GUI-Element aktiviert wurde. Messages dienen unter anderem der Kommunikation zwischen laufenden Anwendungssystemen mit JWAM und können über Rechnergrenzen

[26] Siehe dazu Command-Pattern [GHJV 94]

verschickt werden. Für diese Messages registriert sich ein Werkzeug an einem Nachrichtenverteiler (Message Broker), der auch die Benachrichtigung der Klienten übernimmt.

Bei beiden Mechanismen wird durch die technische Realisierung ein Thread gestartet, der die Ausführung des Ereignis übernimmt. Wird beispielsweise Swing als GUI-Bibliothek verwendet und eine Aktion an der Oberfläche ausgelöst, so wird durch Swing ein Thread gestartet. In diesem Thread wird dann die angemeldete Methode der Präsentationsform aufgerufen und letztlich auch das Command bearbeitet. Interessiert sich ein Werkzeug für eine Message, die von einem anderen Rechner kommen kann, und wird für die technische Realisierung eines Nachrichtenverteilers für Messages[27] RMI verwendet, so wird zur Bearbeitung der Message durch die RMI-Architektur ein Thread gestartet [Farley 98]. Der weitere Kontrollfluß findet dann innerhalb dieses Threads statt. So wäre es möglich, daß zwei Threads zur gleichen Zeit aktiv sind und damit zwei Kontrollflüsse auf den Werkzeugen und allen abhängigen Objekten arbeiten. Die Objekte des Frameworks und der Anwendungen müßten alle synchronisiert werden, was aufwendig wäre und im Falle einer falschen Synchronisierung zu schwer nachvollziehbaren Fehler führen kann.

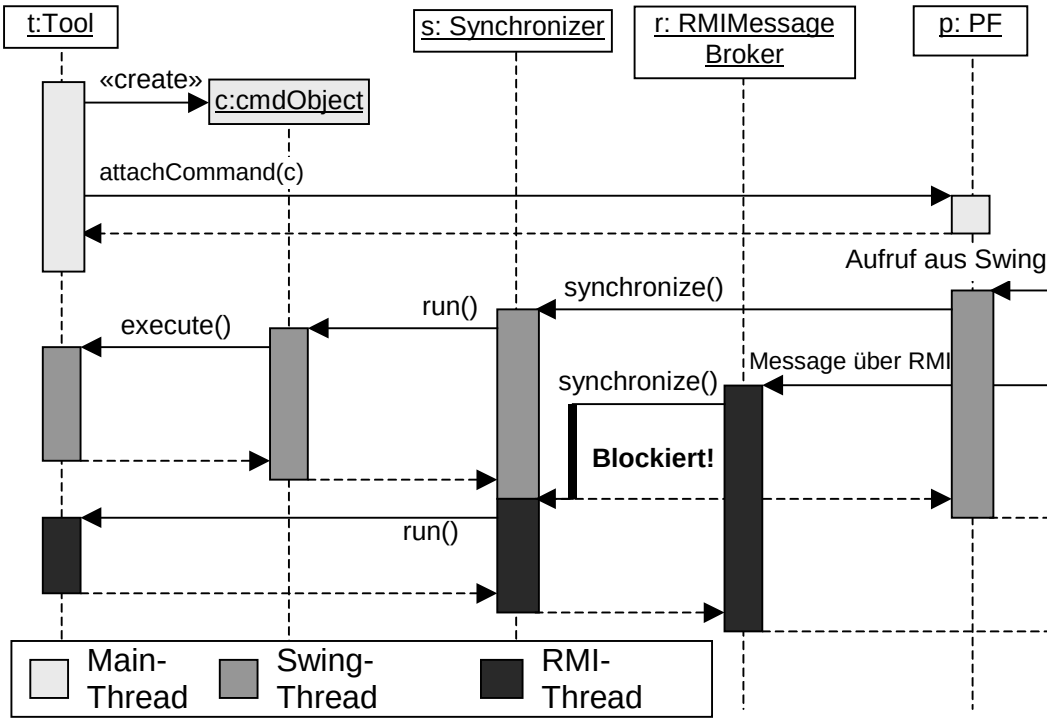

Abbildung 5-2: Kontrollfluß über den Synchronizer

Um das zu vermeiden, werden Messages und Commands über einen Synchronizer geleitet, der nur eine serielle Bearbeitung der Ereignisse zuläßt (siehe Abbildung 5-2). An einer zentralen Stelle werden über das `synchronized`-Schlüsselwort Threads geblockt, bevor weitere Aktionen mit Werkzeugen ausgeführt werden. Innerhalb dieses Bereiches kann nur ein Thread zur gleichen Zeit aktiv sein. Da es sich um ein Einzelplatzsystem handelt und die

[27] Der `RMIMessageBroker` [JWAM]

Wahrscheinlichkeit einer Blockade recht gering ist, ist der Verlust an Performance als minimal anzusehen.

5.1.4 Singleton-Konstruktion

Das Singleton-Pattern dient dazu, nur die Konstruktion eines Exemplars einer Klasse zuzulassen [GHJV 94]. Mit diesem Konstruktionsmuster können globale Informationen in einem Objekt abgelegt werden, die ansonsten über Klassenmethoden bereitgestellt werden müßten. Über Klassenmethoden lassen sich nämlich keine Vererbungsbeziehungen definieren, wohl aber über Objektmethoden. Auf das Objekt wird über eine `instance()`-Methode zugegriffen. Der Konstruktor der Klasse ist `protected`, so daß eine versehentliche und damit fehlerhafte Erzeugung ausgeschlossen ist.

```
// Environment ist ein Singleton
User user = Environment.instance().currentUser();
```

Von diesem Konstruktionsmechanismus wird im JWAM Framework oft Gebrauch gemacht, denn an vielen Stellen müssen Informationen global zur Verfügung gestellt werden. Auch innerhalb der Werkzeuge findet ein Zugriff auf Singletons statt. Singletons werden nicht nur von Klassen innerhalb des Frameworks verwendet. Beispiele für Singletons sind das `Environment` (Werkzeug- und Materialverwaltung), Fabriken für Fachwerte [JWAM][BLZ 99] oder das `UserManagement` (Benutzerverwaltung). Sofern nur eine Anwendung in einer VM aktiv ist, können globale Informationen in Singletons gehalten werden. Bei mehreren Anwendungen in einer VM wie bei einem Thin Client sind die globalen Informationen eventuell für jede Anwendung unterschiedlich, so daß es zu Änderungen an der bisherigen Singleton-Konstruktion kommen muß (siehe auch 4.4.2)

5.2 Vom Einzelplatzsystem zum Applikationsserver

Eine Anwendung in JWAM läuft innerhalb einer Java VM frei von nebenläufigen Threads ab. Durch den Synchronizer ist garantiert, daß auf die Objekte der Werkzeuge gleichzeitig nur von einem Thread aus zugegriffen wird. In einer klassischen JWAM-Anwendung, die aus mehreren Werkzeugen bestehen kann, existiert es nur einen aktiven Benutzer.

Bedingt durch die Konstruktion eines Thin Client sollen aber innerhalb der Java VM auf dem Server mehrere Anwendungen mit JWAM laufen, die alle unterschiedliche aktive Benutzer haben können und nebenläufig agieren sollen (siehe auch 4.4.2). Das gesamte System ist ein Applikationsserver geworden, auf dem für viele verschiedene Benutzerrechner die Applikationen wie Prozesse nebeneinander ablaufen. Nebeneinander ablaufen bedeutet in Java, daß mehrere Threads innerhalb einer Java VM zeitgleich aktiv sind. Mehrere Benutzer bedeutet für das JWAM Framework, daß Werte aus einer Systemumgebung (z.B. das `Environment`) , die bislang über Singletons verfügbar waren, nun pro Benutzer vorhanden sein müssen und nicht mehr einmalig in der gesamten Java VM.

Da ein Benutzer mit seinen aktiven Werkzeugen nur Aktionen nacheinander ausführen kann, ist es sinnvoll, wie bei einem Einzelplatzsystem nur einen aktiven Thread pro Benutzer zu haben. Für jeden Benutzer gibt es also einen **Benutzerprozeß**, der aus einem Thread besteht und auf den Objekten (Werkzeugen, Materialien, ...) des entsprechenden Benutzers arbeitet. Somit entfällt auch weiterhin die Notwendigkeit, sich bei der Konstruktion von Werkzeugen mit Nebenläufigkeitsproblemen zu beschäftigen. Welche Technik dazu eingesetzt wird, beschreibe ich in den folgenden Unterkapiteln.

5.2.1 Vom Synchronizer zum Dispatcher

In einem JWAM-Applikationsserver für Thin Clients laufen mehrere Benutzerprozesse nebenläufig ab. Ein Benutzerprozeß wird durch einen Thread realisiert. Genau wie bei einem Einzelplatzsystem wird ein Benutzerprozeß durch Ereignisse von außen aktiviert: Durch eine Interaktion an der GUI wird ein Command an ein Werkzeug geschickt oder ein Werkzeug erreicht eine Message von einem RMI-Message-Broker. Nach dem WAM-Ansatz sind die Werkzeuge als reaktives System realisiert [Züllighoven 98]. Auf Ereignisse wird durch die Werkzeuge reagiert, in der Regel mit einer Aktualisierung der GUI. Bis zum nächsten Ereignis passiert nichts, der Benutzerprozeß kann somit deaktiviert werden (‚ruhen‘), bis ein neues Ereignis eintrifft.

Ein Benutzerprozeß wird durch ein Exemplar der Klasse `UserProcess` repräsentiert. Es handelt sich dabei um einen Thread, der gestartet wird, sobald ein Benutzer mit einer Anwendung arbeitet. Ereignisse wie Commands und Messages erfüllen das Interface `Runnable` und werden durch den Aufruf von `run()` ausgeführt, indem eine vorher definierte Methode aus der Anwendung aufgerufen wird[28]. Ein Benutzerprozeß besteht aus einer Warteschlange, in die Ereignisse über die Methode `takeEvent()` eingefügt werden und nach dem FIFO Prinzip abgearbeitet werden. Somit wird nur ein Ereignis zur Zeit ausgeführt.

```
// Public-Schnittstelle der Klasse UserProcess
public class UserProcess extends Thread
{
  // Reiht ein Ereignis in die Warteschlange ein
  public void takeEvent(Runnable event)

  // Sind Events in der Warteschlange?
  // Hat der Benutzerprozeß etwas zu tun?
  public boolean isWorking()

  // Gibt das Ereignis zurück, welches gerade bearbeitet
  // wird oder als nächstes bearbeitet werden soll
  public Runnable currentEvent()

  // Nummer des aktuellen Events
  public int eventNumber()
```

[28] Siehe dazu 5.1.2: Ein Aufruf der Methode `run()` bei Commands führt letztlich zum Aufruf der Methode `doExecute()`, in der die Anwendung eine Einschubmethode implementiert.

```
    // Beendet nach Bearbeitung des letzten Events den
    // Benutzerprozeß
    public void exitUserProcess()

    // Nötige Einschub-Methode für Thread-Klasse.
    // Endlosschleife zum Abarbeiten der eingehenden Events,
    // wird durch exitUserProcess() unterbrochen
    public void run()
  }
```

Ereignisse erhalten vom Benutzerprozeß noch eine laufende Nummer zugeteilt (siehe `eventNumber()`), was später für ein Caching noch von Bedeutung sein wird. Ist kein Ereignis mehr eingetragen, geht der Thread in einen Wartezustand über, bis ein neues Ereignis über `takeEvent()` eingetragen wird. Solange ein Benutzerprozeß kein Ereignis zu bearbeiten hat, belegt er auch keine Ressourcen des Prozessors.

Ereignisse können, je nach technischer Implementierung, über einen Thread ausgeführt werden, z.B. ein Swing- oder ein RMI-Thread für Commands bzw. für Messages. Daneben steht ein Benutzerprozeß, der auch als Thread realisiert ist. Ein Ereignis in einem anderen Thread muß also zur Aktivierung des Benutzerprozesses führen, der dann das Ereignis bearbeitet.

Messages und Commands werden bislang zentral über den Synchronizer in eine serielle Reihenfolge zur Bearbeitung gebracht, damit in einem JWAM Einzelplatzsystem zur gleichen Zeit nur ein Thread auf den Werkzeugen arbeitet. In einem Mehrbenutzersystem muß nun dieser Synchronizer die Aufgaben an die Benutzerprozesse verteilen. Er ist zu einem **Dispatcher** geworden, der Ereignisse verteilt. Die Ereignisse werden über `takeEvent()` bei dem Benutzerprozess eingetragen und nacheinander abgearbeitet. Die ursprüngliche Arbeit des Synchronizers, die Serialisierung der Ereignisse, wird nun durch den Benutzerprozeß selbst erledigt.

Damit die eintreffenden Ereignisse passend verteilt werden können, muß an ihnen erkennbar sein, zu welchem Benutzerprozeß sie gehören. Für Commands und Messages meldet sich der Empfänger bei den entsprechenden Komponenten an, in der Regel während der Initialisierungsphase der Werkzeuge. Die Initialisierung der Werkzeuge wird durch den Benutzerprozess durchgeführt. Bei einer späteren Anmeldung aus einem Werkzeug heraus ist der passende Benutzerprozeß auch der aktive Thread. Dieser gerade aktive Thread (und damit der Benutzerprozeß) wird bei einer Anmeldung von Commands oder Messages ermittelt und seine Identifikation in den Commands oder Messages gespeichert.

Die Verwaltung der Benutzerprozesse wird durch den `ProcessManager` erledigt, wobei Prozesse über eine ID (Fachwert `dvIdentificator`) identifiziert werden. Benutzerprozesse werden beim `ProcessManager` angemeldet. Der `ProcessManager` kann anhand des aktiven Threads den Benutzerprozeß ermitteln und seine ID zurück liefern, sowie über eine gegebene ID dann die Referenz auf den passenden Benutzerprozeß wieder herausgeben. Der `ProcessManager` ist als Singleton realisiert, er kann in einer VM nur einmal vorkommen.

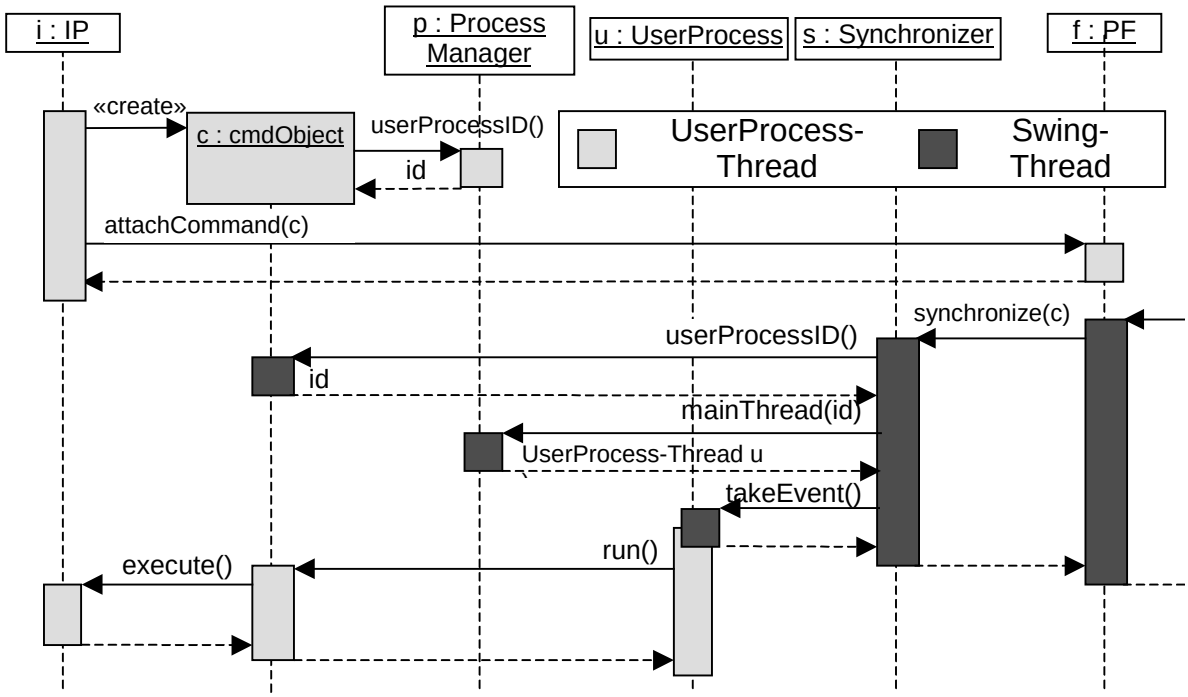

Abbildung 5-3: Interaktionsdiagramm für die Anmeldung und Ausführung eines Commands

Soll dann ein Command oder eine Message an das Werkzeug weitergeleitet werden, so wird es dem Synchronizer hingehalten. Dieser kann den passenden Benutzerprozeß am Command- oder Message-Objekt ermitteln. An dem Benutzerprozeß wird das Ereignis über `takeEvent()` eingetragen. Der ursprüngliche Thread hat damit seine Aufgabe erledigt und wird schließlich beendet. Der Thread des Benutzerprozesses übernimmt dann die Bearbeitung des Ereignisses.

Im Mehrbenutzerbetrieb ist es somit sichergestellt, daß immer der passende Benutzerprozeß aktiv ist, wenn innerhalb eines Werkzeuges Code ausgeführt wird. Die Benutzerprozesse innerhalb eines JWAM Systems können störungsfrei vollkommen nebenläufig arbeiten, sofern sie nicht auf gemeinsame Objekte oder Klassenmethoden zugreifen.

5.2.2 Der Singleton-Manager

In dem JWAM Framework wird das Entwurfsmuster Singleton [GHJV 94] dazu verwendet, Informationen global an jeder Stelle im Programmcode zur Verfügung zu stellen. Da es nun mehrere Benutzerprozesse geben kann, muß es pro Benutzerprozeß ein Singleton geben, da beispielsweise Umgebungsinformationen in einem Singleton für jeden Benutzer unterschiedlich sind (siehe auch 4.4.2). Informationen müssen somit über **Prozeßsingletons** verfügbar sein. Andere Singletons, wie z.B. die Fabriken von Fachwerten, können von allen Benutzerprozessen benutzt werden und bleiben als Singletons erhalten. Der Umgang der Singletons nach außen sollte sich durch den Mehrbenutzerbetrieb nicht ändern, da ansonsten der Programmcode einer Anwendung geändert werden muß.

Über den `ProcessManager` kann die ID des Benutzerprozesses ermittelt werden, der gerade lebendig ist. Die Prozeßsingletons werden dazu in einem Singleton-Manager verwaltet[29], der anhand der ID eine passende Zuordnung durchführt. Der Singleton-Manager hält für jede angemeldete Klasse von Prozeßsingletons eine Liste vor. In dieser Liste ist für jeden Benutzerprozeß ein dazugehöriges Exemplar des jeweiligen Prozeßsingletons enthalten. Prozeßsingletons müssen in ihrer `instance()`-Methode diesen Singleton-Manager verwenden. Ein Prozeßsingleton ist genauso zu benutzen wie ein Singleton, die Schnittstelle ist identisch. Dazu ein Beispiel aus der Klasse `Environment`:

```
// Gibt ein Exemplar (Singleton) des Environments zurück
public static synchronized Environment instance()
{
  SingletonManager sm = SingletonManager.instance();

  // Ist noch kein Exemplar dieser Klasse für den
  // aktiven Benutzerprozeß registriert?
  if(!sm.existsInstance(Environment.class))
  {
   // Exemplar für diese Klasse und Prozeß registrieren
   sm.registerInstance(Environment.class, new Environment());
  }
  // Exemplar zurückgeben
  return (Environment)sm.instance(Environment.class);
}
```

Für den Mehrbenutzerbetrieb müssen alle Singletons im JWAM Framework angepaßt werden. Bei Singletons, die zu Prozeßsingletons werden, muß nur die `instance()`-Methode nach obiger Art umgeschrieben werden. Um Prozeßsingletons und Singletons besser unterscheiden zu können, habe ich die Interfaces `Singleton` und `ProcessSingleton` eingeführt, die aber keine Methoden definieren und nur der Markierung dienen.

Aber auch Singletons müssen neben der Implementierung der Schnittstelle `Singleton` weiter angepaßt werden. Da auf sie von verschiedenen Benutzerprozessen nebenläufig zugegriffen wird, sind die Methoden durch das Schlüsselwort `synchronized` gegen nebenläufige Änderungen zu schützen, um ein inkonsistentes Verhalten zu vermeiden. Zu beachten ist hierbei, daß auch die `instance()`-Methode eines Prozeßsingletons synchronisiert ist, da auf diese Klassenmethode ebenfalls von verschiedenen Benutzerprozessen zugegriffen wird. Alle Klassenmethoden müssen synchronisiert werden, sofern ein nebenläufiger Zugriff zu Problemen führen kann.

Somit muß man sich für einige Klassen im JWAM Framework Gedanken über Nebenläufigkeit machen, allerdings ist dieses Problem auf einen kleinen Bereich beschränkt: Singletons und Klassenmethoden. Ein Beispiel dafür sind Fabriken (siehe Fabrikmuster [GHJV 94]) für Fachwerte [Müller 99][JWAM]: Sie sind als Singletons realisiert, die erzeugten Fachwerte werden also von allen Benutzerprozessen gemeinsam verwendet, was den Umfang der einzelnen Werkzeugen in verschiedenen Benutzerprozessen reduziert. Fachwerte bieten über ihre Fabriken die Möglichkeit an, aus Stringrepräsentationen einen Fachwert zu erzeugen. Die Repräsentation

[29] Selbst ein Singleton

kann auch auf ihre Gültigkeit geprüft werden, wobei im Fehlerfall in der Fabrik eine genauere Fehlerbeschreibung abgelegt wird. Diese kann nach Aufruf der Gültigkeitsprüfung abgefragt werden.

Für den Mehrbenutzerbetrieb schlägt dieses Verhalten fehlt, da aus verschiedenen Prozessen nebenher Repräsentationen auf ihre Gültigkeit überprüft werden können. Diese Prozesse überschreiben sich im Fehlerfall die eingetragene Beschreibung eventuell gegenseitig. Um dieses Problem zu umgehen, müssen die Fehlerbeschreibung pro Benutzerprozeß gespeichert werden, ähnlich wie bei der Konstruktion der Prozeßsingletons. Dazu bietet der `ProcessManager` Methoden an, um Objekte unter einem Namen und für eine Klasse zu registrieren und abzufragen. Je nach aktuellem Benutzerprozeß werden die Objekte gesichert und auch für den aktuellen Benutzerprozeß zurückgegeben.

Ein weiterer Fehlerfall kann durch zwei Threads auftreten, die mit gemeinsamen Ressourcen arbeiten: **Deadlocks**. Damit ein Deadlock in Java überhaupt auftreten kann, müßten sich zwei synchronisierte Abschnitte gegenseitig aufrufen oder Ressourcen nacheinander gesperrt werden (siehe [Oaks/Wong 97]). Wenn Klassenmethoden oder Methoden von Singletons synchronisiert werden, sollten Ressourcen nur einmal komplett durch das `synchronzied`-Schlüsselwort gesperrt werden. Die Sperrung einer Ressource innerhalb der Sperrung einer anderen Ressource ist potenziell gefährlich und sollte vermieden werden. Da bei der Konstruktion des JWAM Frameworks bewußt zyklische Benutzt-Beziehungen vermieden wurden, kann der Fehlerfall nicht auftreten, daß sich synchronisierte Methoden von unterschiedlichen Klassen gegenseitig aufrufen. Deadlocks sind bei einer nebenläufigen Programmierung zwar nie vollkommen auszuschließen, jedoch ist das Fehlerrisiko für den JWAM Thin Client minimal.

5.3 Entwurf des JWAM Thin Client

Die Einführung von Benutzerprozessen ermöglicht es, daß in einer Java VM mehrere Benutzer mit dem JWAM Framework auf Werkzeugen arbeiten können. Damit ist die Grundlage für einen Applikationsserver geschaffen, der Thin Clients mit laufenden Applikationen bedienen kann. Ein solcher Thin Client kann mit ULC realisiert werden, wie in 4.4 dargestellt. Die offenen Punkte aus 4.4 sind damit gelöst. In den folgenden Kapiteln stelle ich noch eine weitere Technologie vor, die wie ULC eine Verteilung der Oberfläche auf den Benutzerrechner ermöglicht: den JWAM Thin Client. Durch eine Erweiterung des Frameworks ist es möglich, Präsentationsformen auf dem Benutzerrechner ausführen zu lassen, während der Rest der Anwendung auf dem Server abläuft.

5.3.1 Schnitt durch PF/IAF

Bei der Konstruktion eines Thin Client mit ULC befinden sich Interaktionsformen und Präsentationsformen auf dem Server. Die Präsentationsformen erben von den GUI-Elementen aus der GUI-Bibliothek von ULC. Der Schnitt zwischen Benutzerrechner und Server ging nicht durch Klassen aus JWAM.

Der Schnitt für den JWAM Thin Client setzt eine Ebene tiefer an: Die Interaktionsformen befinden sich auf dem Server, während die Präsentationsformen auf dem Benutzerrechner liegen. Damit werden alle Aufgaben, die innerhalb der Präsentationsformen erfüllt werden, auf dem Benutzerrechner durchgeführt. Ein Beispiel ist das Überprüfen eines Fachwertes [JWAM]: Präsentationsformen, die Fachwerte anzeigen, überprüfen die Eingabe auf syntaktische Korrektheit. Dieses wird schon auf dem Benutzerrechner durchgeführt, eine Kommunikation mit der Serveranwendung muß nicht stattfinden. Das entspricht dem Konzept der Validatoren in ULC, die solche Aufgaben zwecks Kommunikationsreduzierung auf dem Benutzerrechner erfüllen. Durch die Verteilung der Präsentationsformen wird ein solches Verhalten automatisch ermöglicht.

Die Werkzeuge haben bislang über die Schnittstelle einer Interaktionsform auf eine Präsentationsform zugegriffen, wozu sie eine Referenz auf das Objekte besaßen. Beim JWAM Thin Client greifen die Werkzeuge auf Stellvertreterobjekte zu, welche die Schnittstellen von Interaktionsformen erfüllen und die Aufrufe dann an die Präsentationsformen weiterleiten. Die Konstruktion entspricht den Halbobjekten in ULC: Auf dem Server befinden sich Proxy-Objekte, in diesem Fall die Interaktionsformobjekte, die auf die eigentlichen GUI-Objekte auf dem Benutzerrechner zugreifen, in diesem Fall die Präsentationsform. Die Methodenaufrufe werden unverändert weitergeleitet, da es sich um Aufrufe über die gleiche Schnittstelle an den Präsentationsform handelt.

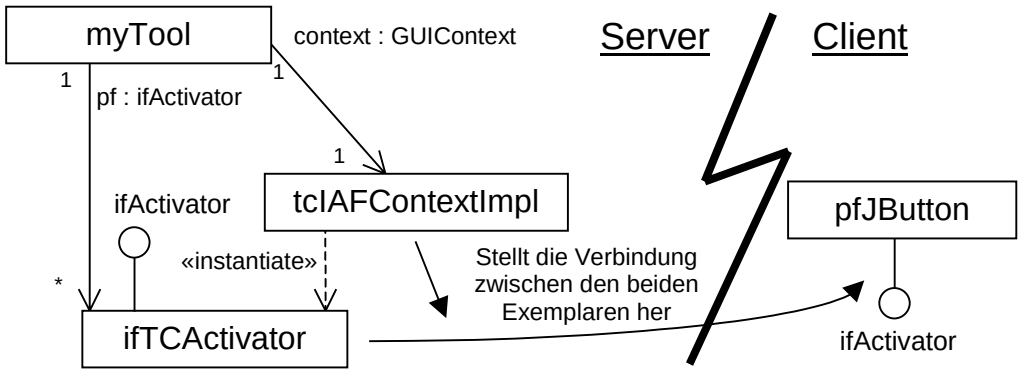

Abbildung 5-4: Zusammenhang IF/PF/Kontext

Die Erzeugung der Interaktionsformobjekte übernimmt ein spezieller Kontext: der `tcIAFContextImpl`[30]. Dieser Kontext erfüllt die Schnittstelle `IAFContext` . Er wird bei Werkzeugen anstatt des `IAFContextImpl` verwendet und ist vom Verhalten her für das Werkzeug identisch. Fragt ein Werkzeug nach einer Interaktionsform für eine Präsentationsform, so erzeugt der Kontext ein passendes Interaktionsformobjekt und verbindet es mit der Präsentationsform auf dem Benutzerrechner.

5.3.2 Kommunikation über Client- & Serverobjekte

Für die Verbindung von Interaktion und Präsentation und auch für die weitere Kommunikation zwischen den beiden Objekten sind zwei weitere Objekte zuständig: Exemplare der Klasse

[30] Das Kürzel ‚tc' steht für ‚Thin Client'

`tcServerImpl` (im folgenden Serverobjekt genannt) und `tcClientImpl` (im folgenden Clientobjekt genannt). Beide erfüllen jeweils die Schnittstellen `tcServer` und `tcClient`, die definieren, was für Methoden die Objekte für den verteilen Zugriff anbieten. Die Schnittstelle für den lokalen Zugriff auf die Objekte ist jeweils in `tcServerLocal` bzw. in `tcClientLocal` definiert. Die vorgestellten Implementierungen benutzen RMI als Middleware für die Verteilung. Andere Implementierung sind denkbar, es müßten dafür entsprechende andere Implementierungsklassen entwickelt werden.

Auf dem Server ist pro Benutzerprozeß ein Serverobjekt als Prozeßsingleton vorhanden. Auf dieses greifen sowohl die speziellen Kontexte der Werkzeuge als auch die Interaktionsformobjekte zu. Auf dem Benutzerrechner existieren mehrere Clientobjekte, für jeden Kontext ein Clientobjekt. Da jedes Werkzeug über einen Kontext verfügt, existiert somit für jedes Werkzeug ein Clientobjekt. Das Clientobjekt übernimmt die Verwaltung der Oberfläche eines Werkzeuges.

5.3.2.1 Start eines Werkzeuges

Der Start eines Werkzeuges und die Initiierung eines Benutzerprozesses geht vom Benutzer auf dem Benutzerrechner aus. Auf diesem Rechner läuft ein Java-Applet, das nur die Aufgabe hat, die verfügbaren Werkzeuge anzuzeigen und den Start der Werkzeuge zu veranlassen.

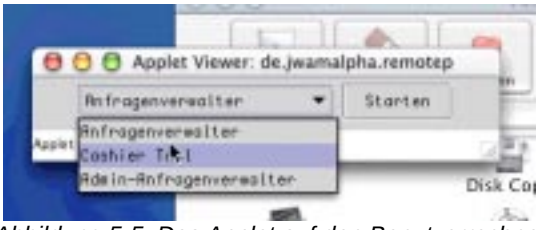

Abbildung 5-5: Das Applet auf den Benutzerrechner

Auf dem Server ist, sofern noch kein Werkzeug gestartet wurde, als verteilt zugreifbares Objekt nur ein Listenerobjekt aktiv. Dieses Listenerobjekt hat die Aufgabe, eine Liste der verfügbaren Werkzeuge herauszugeben und ein solches Werkzeug zu starten. Außerdem werden an diesem Listenerobjekt beim Start der Serveranwendung alle verfügbaren Werkzeuge angemeldet. Die Vererbungsstruktur der Klasse des Listener-Objektes ist mit der Klasse der Server- bzw. Clientobjekte identisch: Es gibt eine Implementierungsklasse `tcListenerImpl`, wobei die verteilten Methoden in `tcListener` und die lokal aufzurufenden Methoden in `tcListenerLocal` definiert sind. Eine andere Implementierung kann auch hier durch eine weitere Implementierungsklasse konstruiert werden.

Das Java-Applet fragt an diesem Listenerobjekt ab, welche Werkzeuge vorhanden sind und gibt diese Liste an der Oberfläche aus. Wird nun ein Werkzeug ausgewählt und gestartet, so erzeugt das Applet auf dem Benutzerrechner zuerst ein Clientobjekt, das später die GUI des Werkzeuges verwalten wird. Mit einer über RMI verteilten Referenz auf dieses Objekt und dem Namen des zu startenden Werkzeuges wird dann das Listenerobjekt angewiesen, einen neuen Benutzerprozeß mit dem passenden Werkzeug zu starten.

5.3.2.2 Aktivierung der GUI und dynamisches Class-Loading

Beim JWAM Thin Client befinden sich die Präsentationsformen auf dem Benutzerrechner. Um das zu ermöglichen, muß ein Clientobjekt die GUI erzeugen können. Das wird für den JWAM Thin Client dadurch erreicht, daß die Klasse, in der die GUI implementiert ist, auf den Benutzerrechner gebracht wird[31]. Dort wird dann ein Exemplar dieser Klasse erzeugt und die GUI auf dem Bildschirm angezeigt. Auf dem Server wird kein GUI-Objekt erzeugt.

Für das Laden einer Klasse wird der Mechanismus des dynamischen Class-Loading aus Java benutzt [Sun 99]. Das Java-Applet auf dem Benutzerrechner ist so konfiguriert, daß es seine Klassen vollständig vom Server lädt und somit auch die GUI-Klassen für die Werkzeuge. Damit befindet sich auf dem Benutzerrechner kein applikationsspezifischer Softwarecode, der zum Starten der Clientanwendung vorher installiert werden müßte. Das Problem des Softwaredeployments entfällt komplett. In 5.4.3 ist der Mechanismus des dynamischen Class-Loading noch genauer erläutert.

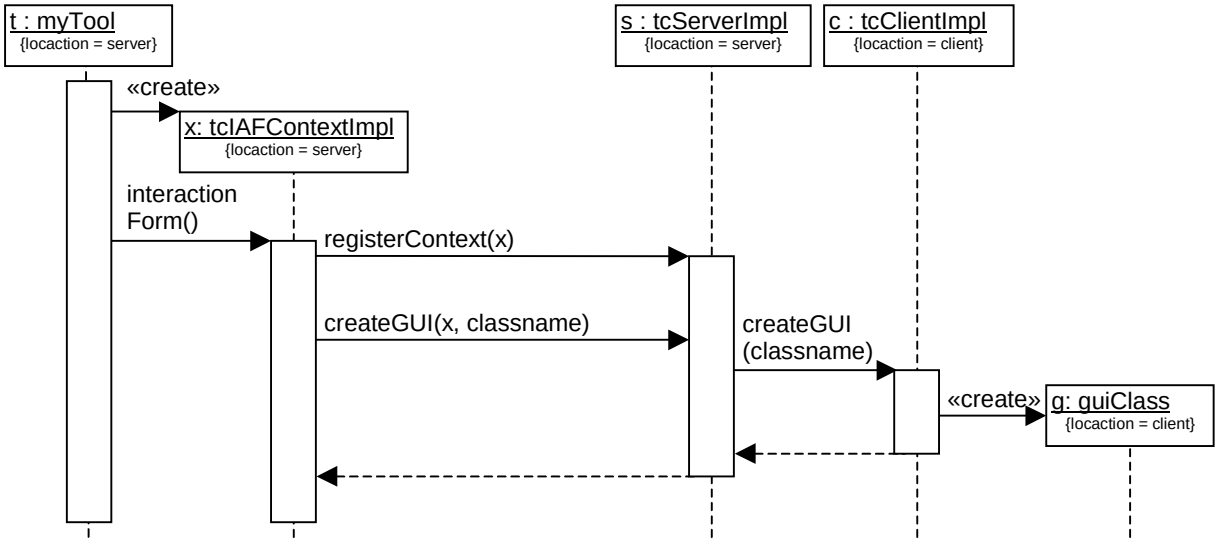

Abbildung 5-6: Erzeugung der GUI

Wird durch das Listenerobjekt ein Werkzeug gestartet, so erzeugt das Werkzeug für die Verwaltung der GUI einen speziellen Kontext (`tcIAFContext`). Dem Kontext wird die GUI-Klasse als Klassenobjekt übergeben und intern vermerkt. Findet eine Aktion am Kontext statt, wie z.B. das Ermitteln von Interaktionsformen, wird der ganze Prozeß der Verteilung und der Aktivierung der GUI in Gang gesetzt. Eine Kommunikation mit dem Benutzerrechner findet somit nur nach Bedarf statt.

Zuerst registriert sich der Kontext am Serverobjekt. Durch die Registrierung werden intern Verwaltungsobjekte erzeugt (siehe dazu 5.4.2 zu Caching) und die Verbindung zwischen Client- und Serverobjekt für diesen Kontext hergestellt. Client- und Serverobjekte kennen sich gegenseitig, da von beiden Seiten eine Kommunikation initiiert werden kann (siehe Commands).

[31] GUIs für Werkzeuge werden im JWAM Framework normalerweise durch Klassen konstruiert, für den JWAM Thin Client ist dieses nun zwingend notwendig.

An das Serverobjekt kommt aus dem Kontext heraus die Aufforderung zur Erzeugung der GUI. Dem Aufruf wird der Kontext mitgegeben, so daß das Serverobjekt entscheiden kann, an welches Clientobjekt die Aufforderung ergehen soll. Dort wird dann der Aufruf weitergeleitet, wobei nicht mehr das Klassenobjekt, sondern nur der Name der Klasse weitergegeben wird. Das Clientobjekt erzeugt anhand des Klassennamens ein Exemplar und macht die GUI sichtbar. Bevor ein Exemplar der GUI-Klasse erzeugt wird, wird über das dynamische Class-Loading die passende Klasse auf den Benutzerrechner geladen. Das Laden der Klasse wird von der Java VM durchgeführt.

5.3.2.3 Auffinden von Präsentationsformen

Nachdem eine GUI auf dem Benutzerrechner erzeugt wurde, muß zu einer Interaktion die passende Präsentation gefunden und ein Interaktionsformobjekt erzeugt werden. Die Verbindung von Präsentationsform und Interaktionsform ist im folgenden Interaktionsdiagramm genauer verdeutlicht:

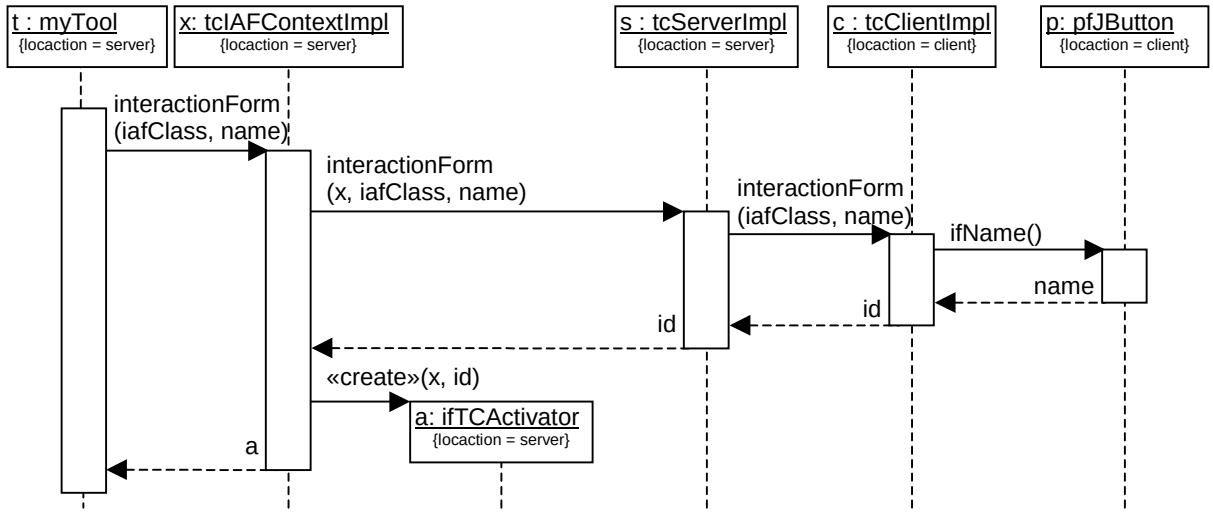

Abbildung 5-7: Auffinden einer Präsentationsform

Unter Angabe einer Klasse (für die Interaktionsform) und eines Namens fragt das Werkzeug nach einer Interaktionsform. Der Kontext wendet sich dann an das Serverobjekt, wobei er sich zwecks Identifikation wieder selber übergibt. Der Aufruf wird an das passende Clientobjekt weitergeleitet, welches dann die Suche nach der Präsentationsform übernimmt. Damit erfüllt das Clientobjekt auch eine Aufgabe eines GUI-Kontextes und besitzt deshalb Wissen über die verwendete GUI.

Die Identifikation der gefundenen Präsentationsformen findet über eindeutige Identifikatoren statt, wobei es sich um Fachwerte des Typs `dvIdentificator` aus dem JWAM Framework handelt. Das Clientobjekt erzeugt für jede Präsentationsform einen Identifikator und gibt diesen als Ergebnis der Suche nach Präsentationsformen zurück. Der Kontext auf dem Server erzeugt für die gefundene Präsentationsform nun ein Stellvertreterobjekt, in diesem konkreten Beispiel ein Exemplar der Klasse `ifTCActivator`. Dem Interaktionsformobjekt wird der Kontext und die Identifikation der Präsentationsform übergeben, womit später das passende Clientobjekt und die passende Präsentationsform gefunden werden kann.

5.3.2.4 Abwicklung eines Aufrufes an einer Präsentationsform

Das Werkzeug wendet sich über die Interaktionsform an die Präsentationsform. Dazu ruft es Methoden an dem Interaktionsformobjekt auf, die einfach an die passende Präsentationsform weitergeleitet werden. Die Interaktionsformobjekte leiten damit einfach nur ihre Methodenaufrufe weiter und liefern gegebenenfalls ein Ergebnis zurück. In den Interaktionsformobjekten wird bei einem Methodenaufruf der Aufruf für die Präsentationsform durch Exemplare der Klasse `MethodCall` erzeugt. Dabei wird neben der Identifikation der Präsentationsform, der Methodenname und gegebenenfalls Übergabeparameter bei der Konstruktion festgelegt[32]. Unter Angabe des passenden Kontext wird dann dieser Methodenaufruf über das Serverobjekt an das Clientobjekt weitergeleitet. Dazu ein Codebeispiel aus der Klasse `ifTCFillInText`:

```
/**
 * Setzt einen Text
 */
public void setValue (String value)
{
  _server.handleCallVoid(_context,
    new MethodCall(_pfID, "setValue", value, String.class));
}
```

Dieses Verhalten ist für die Methoden in den Interaktionsformobjekten typisch. Alle Methodenaufrufe werden über Exemplare der Klasse `MethodCall` kodiert. Die Parameter `_context` (Der Kontext, in dem die GUI eingebettet ist) und `_pfID` (Identifikation der Präsentationsform) wurden während der Konstruktion des Interaktionsformobjektes gesetzt und sind konstant. An dem Serverobjekt werden zwei Methoden zur Weiterleitung der Methodenaufrufe angeboten: `handleCallVoid()` und `handleCallReturn()`. Letzterer wird benutzt, falls ein Aufruf ein Ergebnis zurück liefert. Die beiden Methoden haben in bezug auf die später erläuterte Caching-Strategie einen etwas anderen Ablauf.

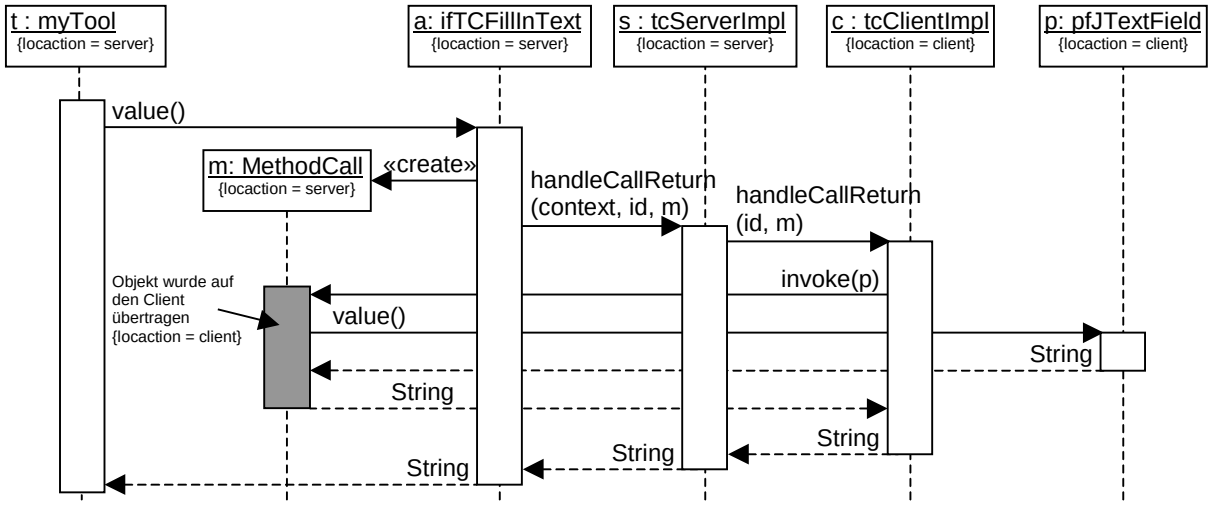

Abbildung 5-8: Aufruf einer Methode an der PF

[32] Neben dem Übergabeparameter selbst muß auch die Klasse (Typ) des Übergabeparameters angegeben werden. Dazu kann nicht einfach die Klasse (der dynamische Typ) des übergebenen Objektes genommen werden, mit dem später die Methode aufgerufen werden soll. Es muß der in der Methodendefinition angegeben Typ benutzt werden, damit auch die korrekte Methode aufgerufen wird. Ansonsten ist der Aufruf bei überladenen Methoden nicht mehr eindeutig.

Das erzeugte `MethodCall`-Objekt wird über das Serverobjekt an das passende Clientobjekt übertragen. Dieses ermittelt anhand der übergebenen Identifikation die Präsentationsform, an der die Methode aufgerufen werden soll. Diese Präsentationsform wird dem `MethodCall`-Objekt übergeben, das dann anhand der eingestellten Parameter die richtige Methode an der Präsentation aufruft. Das `MethodCall`-Objekt selbst ist serialisiert und auf den Benutzerrechner übertragen worden, weshalb es im obigen Diagramm gesondert markiert ist.

5.3.3 Verteilung der Commands

Bislang wurde nur der Fall betrachtet, das Aufrufe vom Server an Benutzerrechner weitergeleitet werden. Auf den Benutzerrechnern befindet sich aber eine Oberfläche, deren Elemente durch Benutzeraktionen Ereignisse auslösen können. Diese Ereignisse müssen an das Werkzeug übermittelt werden. Eine Rückkopplung über Änderungen an Oberflächenelementen an das Werkzeug findet im JWAM Framework über Commands statt.

Commands werden in den Werkzeugen typischerweise als Exemplare anonymer Inner-Classes erzeugt und über die Schnittstelle der Interaktionsform an eine Präsentationsform angemeldet (siehe 5.1.2). In der anonymen Inner-Class ist eine Callback-Methode implementiert, die durch das Aktivieren des GUI-Elementes aufgerufen wird. Ein solches Command-Objekt kann daher nicht wie ein `MethodCall`-Objekt serialisiert und zwischen Client- und Serverobjekt transportiert werden.

Werden Commands an einem Interaktionsformobjekt angemeldet, so wird diese Anmeldung wieder an das zentrale Serverobjekt weitergeleitet. Dort wird für jedes Command ein Identifikator vergeben. Das Command-Objekt verbleibt auf dem Server. Anhand dieses Identifikators kann später das Clientobjekt dem Serverobjekt mitteilen, welches Command es auszuführen hat. Das Command wird über den Synchronizer geleitet, da der Aufruf vom Clientobjekt in der hier erläuterten Implementierung (siehe auch 5.5.4) über RMI funktioniert und ein RMI-Thread statt eines Benutzerprozesses aktiv ist.

An die Präsentationsformen auf dem Benutzerrechner meldet sich das Clientobjekt selbst als Empfänger für Commands an. Problematisch ist dabei allerdings die Tatsache, daß es verschiedene Arten von Commands gibt, die durch unterschiedliche Klassen repräsentiert werden[33]. Das Clientobjekt muß also wissen, mit welcher Art von Command es sich bei einer Präsentationsform anmelden soll. Diese Information ist in dem Interaktionsformobjekt auf dem Server verfügbar, was auch als Parameter weitergeben wird. Dazu ein Beispiel aus der Klasse `ifTCActivator`:

```
// Fügt ein Activate-Command hinzu
public void attachActivateCommand (cmdActivate cmd)
{
  _activateCommandID = _server.attachCommand(_context, _pfID,
    cmd, "attachActivateCommand", "cmdActivate");
}
```

[33] Alle Commands erben von der Oberklasse `cmdObject`.

Dem Serverobjekt wird wieder der Kontext und der Identifikator für die Präsentationsform übergeben. Weiter wird dem Serverobjekt das eigentliche Command mitgeteilt. Wie bei einem Methodenaufruf über `MethodCall`, der später auch bei der Anmeldung an die Präsentationsform stattfinden wird, wird der Name der Methode und die Klasse des Aufrufparameters hier übergeben. Diese Parameter werden an das Clientobjekt weitergereicht, sowie die vom Serverobjekt für das Command vergebene Identifikation. Dazu ein Auszug aus dem Code des Clientobjektes:

```
/**
 * Meldet ein Command an einer PF an
 */
public void attachCommand(dvIdentificator pfID,
  String methodName, String cmdClassName,
  dvIdentificator cmdID) throws RemoteException
{
  Object iaf = _iafs.get(pfID);
  // Ermittelt Präsentationsform anhand der ID

  cmdObject cmd = CommandTransformer.instance().
    createCommand(cmdClassName, this);
  // Meldet sich als Empfänger für das Command an

  _cmdIDs.put(cmd, new dvIdentificator [] {cmdID, id});
  // Zu einem Command wird die ID des Commands
  // und der PF in einer Liste gespeichert

  (new MethodCall(pfID, methodName, cmd,
    cmd.getClass().getSuperclass())).invoke(iaf);
  // Aufruf zum Anmelden des Commands an der PF
}
```

Die Klasse `CommandTransformer` übernimmt zentral die Aufgabe, aus dem Namen des Commands (`cmdClassName`) ein passendes Command zu erzeugen. In der Methode `createCommand()` wird anhand des übergeben Namens das passende Command wie gehabt als anonyme Inner-Class erzeugt. Die Callback-Methode in dieser anonymen Inner-Class ruft eine weitere Callback-Methode in dem Clientobjekt auf (`executeCommand()`). Dazu wird das Clientobjekt bei `createCommand()` übergeben.

In der Klasse `CommandTransformer` steckt damit Wissen über alle Commands aus dem JWAM Framework. Da prinzipiell alle möglichen Commands über diesen Mechanismus angemeldet werden können, müssen in der Klasse `CommandTransformer` auch alle Eventualitäten an möglichen Command-Klassen abgedeckt werden.

In dem Codebeispiel wird abschließend das erzeugte Command an die Präsentationsform angemeldet. Die Klasse des Übergabeparameters ist hier die Superklasse des Commands: Wird in einem Werkzeug ein Command angemeldet, so muß das durch eine Implementierung einer abstrakten Command-Klasse geschehen (siehe 5.1.2).

Wird eine Aktion an der Oberfläche ausgelöst, so wird die `executeCommand()`-Methode des Clientobjektes gerufen. Von dort wird dann der Aufruf an den Server weitergeleitet, wobei über eine Liste (`_cmdIDs`) für ein Command der Identifikator des Commands auf dem Server und der Identifikator der Präsentationsform ermittelt werden kann. Über den Identifikator des Commands wird im Serverobjekt das entsprechende Command ermittelt und ausgeführt. Allerdings beinhalten die unterschiedlichen Commands auch unterschiedliche Informationen. In einem Command, das bei einer Selektierung ausgelöst wird, ist auch der selektierte Index enthalten. In einem anderen Command, welches bei Änderung eines Textes ausgelöst wird, steht der neue, veränderte Text. Diese Informationen können vom Werkzeug verarbeitet werden.

Commands sind aufgrund ihrer Erzeugung nicht serialisierbar, das Objekt und damit die darin enthaltenen Informationen, kann daher nicht einfach über Rechnergrenzen verteilt werden. Abhilfe dazu schafft wieder der `CommandTransformer`: Er bietet als Dienst die Möglichkeit, alle relevanten Informationen aus einem Command-Objekt in ein serialisierbares Objekt zu packen und ein gegebenes Command-Objekt mit diesen Informationen zu bestücken. Für diese Aufgabe benötigt der `CommandTransformer` wieder Wissen über alle Commands aus dem JWAM Framework.

Das Clientobjekt liest über den `CommandTransformer` alle von der Präsentationsform gesetzten Informationen aus. Diese werden an das Serverobjekt übergeben, welches an dem Command, das an dem Interaktionsformobjekt angemeldet wurde, alle übergeben Werte wieder setzt. Danach wird das Command letztlich ausgeführt.

5.4 Optimierungsstrategien

Die Geschwindigkeit des Kommunikationskanals zwischen Clientobjekt und Serverobjekt kann stark variieren, da ein Thin Client auch über das Internet mit seiner unterschiedlichen Verbindungsgüte benutzt wird. Der Benutzerrechner kann über eine schnelle lokale Verbindung oder nur über ein Modem am Server angebunden sein. Um den Kommunikationsaufwand zu verkleinern, sind im JWAM Thin Client weitere Optimierungsstrategien implementiert.

5.4.1 Asynchronität

Client- und Serverobjekt arbeiten nicht synchron. Das bedeutet, daß bei einem Aufruf einer Methode an dem anderen Objekt nicht auf die Beendigung der Methode gewartet wird. Der Kontrollfluß auf dem Server und vor allem auf dem Benutzerrechner wird damit nicht durch das Warten auf eine Kommunikation über ein möglicherweise langsames Netz behindert.

In einem Werkzeug werden beispielsweise an Präsentationsformen nacheinander neue Daten gesetzt. In einem synchronen Modell würde der Aufruf an einer Präsentationsform erst dann beendet werden, wenn der neue Inhalt eingetragen ist. Für den JWAM Thin Client würde das bedeuten, daß der Benutzerprozeß solange warten muß, bis die Präsentationsform auf dem Benutzerrechner mit dem neuen Inhalt gefüllt ist und der Kontrollfluß zurückkommt. Umgekehrt würde ein solches synchrones Vorgehen bedeuten, daß die Oberfläche auf dem Benutzerrechner blockiert ist, solange ein abgeschicktes Ereignis nicht beantwortet wurde. Die Oberfläche kann

während dieser Zeit nicht bedient werden. Es können beispielsweise keine Daten eingeben oder in einer Tabelle geblättert werden.

Deshalb ist für das Versenden der Aufrufe vom Serverobjekt an das Clientobjekt und umgekehrt jeweils ein eigener Thread zuständig: der **Versandprozeß**. Aufrufe werden in einem Puffer gesammelt, der eintragende Prozeß wird weiter nicht im Ablauf behindert. Der Versandprozeß kümmert sich um das Versenden der Methodenaufrufe, wobei er diese auch in einem Block verschicken kann. In einem Block zusammengefaßte Aufrufe vermindern den Kommunikationsaufwand, da nicht für jeden einzelnen Aufruf eine gesonderte Kommunikation initiiert werden muß.

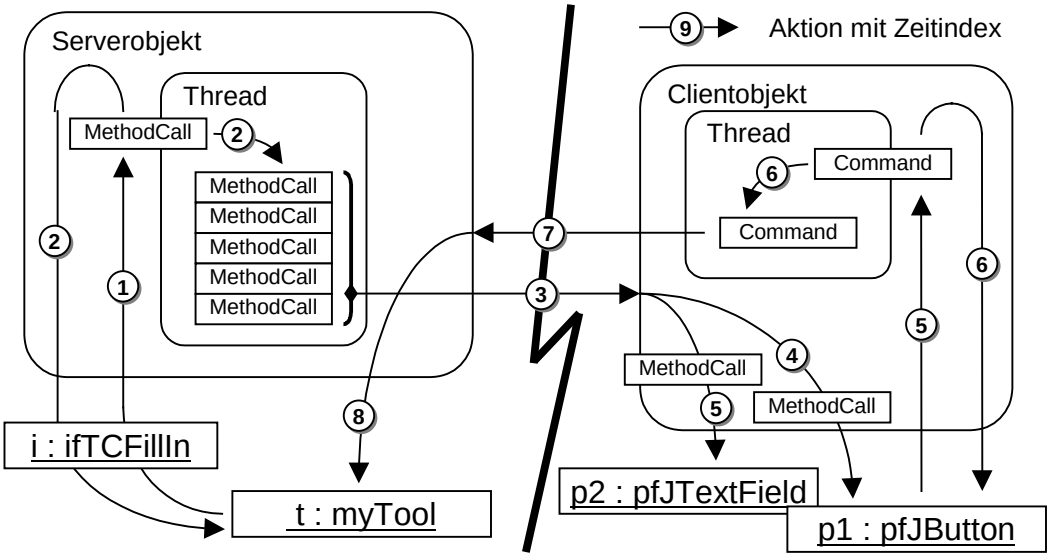

Abbildung 5-9: Ablauf von Methodenaufrufen

In Abbildung 5-9 ist der asynchrone Ablauf der Kommunikation zwischen Client- und Serverobjekt schematisch dargestellt. Ein Werkzeug will über ein Interaktionsformobjekt die Oberfläche manipulieren (1). Dazu wird vom Interaktionsformobjekt ein `MethodCall`-Objekt erzeugt, das an die Präsentationsform weitergereicht werden soll. Dieses Objekt wird vom Versandprozeß entgegengenommen, und in einem Puffer gesammelt (2). Nach dem Eintragen kann der Benutzerprozeß weiterarbeiten, der Aufruf an diesem Interaktionsformobjekt aus dem Werkzeug ist somit erledigt. Es können weitere Manipulationen dieser Art folgen (Wiederholung Ablauf 1&2), die letztlich zu Eintragungen im Puffer des Versandprozesses führen. Ist der Puffer voll oder schon eine gewisse Zeitspanne vergangen, in der die Oberfläche nicht mehr aktualisiert wurde, werden die `MethodCall`-Objekte aus dem Puffer en bloc übertragen (3).

Das Clientobjekt arbeitet dann die Aufrufe nacheinander ab (4 & 5): An den Präsentationsformen werden die Methoden aufgerufen, die in den `MethodCall`-Objekten angegeben sind und damit die Oberfläche aktualisiert. Währenddessen (5) kann der Benutzer an der Oberfläche ein Ereignis (Command) auslösen, z.B. durch das Drücken eines Buttons. Wie bei dem Serverobjekt wird dieses Ereignis zunächst zwischengespeichert, der Methodenaufruf durch das Command ist erledigt (6). Der Kontrollfluß kehrt an die Oberfläche zurück, es kommt zu keiner Blockade. Im Gegensatz zum Versandprozeß des Serverobjektes wird dieses Ereignis jedoch so schnell wie möglich abgeschickt (7), ein Sammeln von Ereignissen findet nicht statt.

Aktionen an der Oberfläche werden möglichst schnell übertragen und bearbeitet (8), ein Puffern der Ereignisse würde hier nur zu Verzögerungen führen.

Erwartet eine Anfrage aus dem Werkzeug ein Ergebnis, z.B. die Abfrage des selektierten Index aus einer Auswahlliste, wird der Puffermechanismus des Serverobjektes kurzzeitig aufgehoben. Der Puffer wird gezwungenermaßen abgearbeitet und damit die Oberfläche aktualisiert. Damit kann eine Abfrage an dem Oberflächenelement durchgeführt werde, ohne daß eventuell auf alte Daten zurückgegriffen wird. Ansonsten könnte ein Eintrag in dem Puffer nach der Abfrage neue Daten setzen, obwohl das Setzen der Daten vor der Abfrage stattgefunden hat. Trotzdem können noch Fehler auftreten, siehe dazu 5.5.1.

Softwarewerkzeuge sollten an der GUI möglichst schnell eine Reaktion auf die Aktionen des Benutzers anzeigen [Shneiderman 98]. Durch die Trennung von Werkzeug und GUI beim JWAM Thin Client kann es bei einer langsamen Verbindung immer wieder zu Verzögerungen kommen. Die Reaktion auf eine Aktion eines Benutzer muß nicht unmittelbar erfolgen, er kann sogar weiter mit der GUI arbeiten. Damit aber eine gewisse Rückkopplung gegeben ist, wird der Mauszeiger für dieses GUI-Element in den Wartezustand versetzt[34], bis die Anfrage auf dem Serverobjekt bearbeitet ist. Das Ende der Bearbeitung wird, wieder vollkommen asynchron, durch eine Mitteilung des Serverobjektes signalisiert, die auch über den Puffer des Versandprozesses verteilt wird.

5.4.2 Caching

Die Abfrage von Werten an Präsentationsformen behindert das asynchrone Verhalten: Der Puffer des Versandprozesses muß geleert werden. Es kommt in jedem Fall zu einer zeitraubenden Kommunikation, da der Benutzerprozeß auf ein Ergebnis warten muß. Um solche Abfragen möglichst selten durchführen zu müssen, werden die abzufragenden Werte in den Interaktionsformobjekten gecacht.

Immer wenn ein Wert abgefragt wird, wird das Ergebnis gespeichert. Bei einer erneuten Abfrage wird dann nicht mehr die entfernte Präsentationsform befragt, sondern das lokale gespeicherte Objekt zurückgegeben. Dazu ein Beispiel aus der Klasse `ifTCFillInText`:

```
public static String TEXT = "Text:ifTCFillInText";
/**
 * Gibt den eingetragenen Text zurück
 */
public String value ()
{
  // Cache abfragen, Identifikation über einen String
  String result = (String)cacheObject(TEXT);

  // Ist ein Objekt im Cache
  if(result == null)
  {
```

[34] Er zeigt dann z.B. eine Sanduhr an

```
      // Kein Objekt, PF fragen
      result = (String)_server.handleCallReturn(
                  _context, new MethodCall(_pfID, "value"));

      // Cache löschen
      unregisterAllCacheObjects();

      // Abgefragtes Objekt registrieren
      registerCacheObject(TEXT, result);
    }
    return result;  // Rückgabe
  }
```

Für das Caching bietet die Oberklasse `ifTCInterface` alle nötigen Methoden an. Objekte werden für das jeweilige Interaktionsformobjekt gespeichert, die Identifikation findet über einen Namen statt (`TEXT`). Es können durchaus mehrere Objekte gespeichert werden. Die Interaktion `ifSingleSelection` bietet beispielsweise mehrere Abfragearten zu dem selektieren Element an: nach Index, nach der Identifikation des Elementes oder nach der String-repräsentation. Jedes Objekt wird einzeln gespeichert.

Ein gespeichertes Objekt darf nicht für immer im Interaktionsformobjekt verbleiben. An der Oberfläche können die Daten aus der Präsentationsform durchaus vom Benutzer verändert werden, so daß gecachte Informationen in den Interaktionsformobjekten schnell veralten. Um das zu verhindern, besitzen die gespeicherten Objekte nur eine gewisse Lebenszeit: Der Cache ist nur für die Dauer der Bearbeitung eines Ereignisses im Benutzerprozeß aktiv.

Hat der Benutzer an der Oberfläche ein Ereignis ausgelöst, wird dieses über ein Command dem Werkzeug mitgeteilt. Das Werkzeug reagiert typischerweise auf das Ereignis, indem es den Zustand von Oberflächenelementen abfragt und neue Daten setzt. Innerhalb dieser Aktion kann es vorkommen, daß Oberflächenelemente mehrfach abgefragt werden, wobei dann das Caching in den Interaktionsformobjekten greift. Nachdem auf das Ereignis des Benutzers reagiert wurde, werden alle Einträge im Cache als veraltet markiert. Bis zur nächsten Aktion des Benutzers kann dieser an der Oberfläche völlig andere Daten eingetragen haben. Die Abfrage kann deshalb bei zukünftigen Ereignissen nicht mehr mit den alten, zwischengespeicherten Daten funktionieren.

Die Interaktionsformobjekte müssen für ein solches Verhalten wissen, wann ein Benutzer-prozeß beendet ist. Dafür kommt die Methode `eventNumber()` des Benutzerprozesses ins Spiel (siehe 5.2.1). Jede Aktion, die vom Benutzerprozeß bearbeitet wird wie z.B. Commands, bekommt eine eindeutige, fortlaufende Nummer. Dieses Nummer kann an dem Benutzerprozeß erfragt werden. Für die gespeicherten Objekte im Interaktionsformobjekt wird diese Nummer ebenfalls hinterlegt. Wird dann versucht, auf das gespeicherte Objekt aus einer anderen Aktion des Benutzerprozesses heraus zuzugreifen, so ist die aktuelle Ereignisnummer zu der gespeicherten nicht mehr identisch. Das gespeicherte Objekt kann gelöscht werden.

5.4.3 Klassenverteilung

Die folgende Strategie dient nicht der Kommunikationsreduktion. Sie ermöglicht ein einfaches Softwaredeployment, das bei dem JWAM Thin Client durch ein dynamisches Class-Loading gelöst wird.

Der verwendeten Softwarecode von Klassen des Thin Client wird vom Benutzerrechner zur Laufzeit nachgeladen[35]. Dazu benutzt der JWAM Thin Client die Möglichkeiten des Class-Loadings in Java. Auf dem Benutzerrechner müssen neben der Klasse des Applets und der Klasse des Clientobjektes, sowie weiteren zugehörigen Klassen, noch Klassen aus dem JWAM Framework geladen werden: Präsentationsformen, Commands, Interaktionsformen, Fachwerte, GUI-Klassen und einige weitere Hilfsklassen. Die verwendeten Klassen werden nicht sofort beim Start des Applets auf den Benutzerrechner geladen, sondern je nach Bedarf. So wird die Klasse einer Präsentationsform beispielsweise nur dann geladen, wenn sie in einer GUI benutzt und von der GUI-Klasse ein Exemplar erzeugt wird. Dann werden auch die von der Präsentationsform abhängigen Klassen der Interaktionsformen und Commands geladen.

Die Klassen müssen zum Laden auch bereitgestellt werden. Der für Applets angebotene Class-Loading-Mechanismus funktioniert über das HTTP-Protokoll [Sun 99]: Die Klassen müssen über einen Webserver bereitgestellt werden, von dem auch der HTML-Code des Applets geladen werden muß[36]. Das würde für den JWAM Thin Client bedeuten, daß neben dem eigentlichen Server für die Thin Clients noch ein Webserver bereitgestellt werden müßte. Dieser muß dann auch immer auf den aktuellen Code der Klassen verweisen, so daß von einem Benutzerrechner nicht falsche oder veraltete Klassen geladen werden.

Dieser notwendige Dienst wird von der Serveranwendung für den JWAM Thin Client erbracht: In der Klasse `ClassFileServerClasspath` ist ein minimaler Webserver implementiert, der nur die für den JWAM Thin Client notwendige Funktionalität erfüllt. Das Auffinden der Klassen funktioniert über den Classpath der Serveranwendung. In dem Classpath sind alle Lokationen angegeben, in denen sich die Klassen der Serveranwendung befinden. Dieser muß nach der passenden Klasse durchsucht werden. Der Code der Klasse kann sich entweder in einem File oder in einer Jar-Datei befinden. Sofern der Code der Klasse gefunden wurde, wird dieser gelesen und schließlich über das HTTP-Protokoll herausgegeben und damit auf den Benutzerrechner verteilt. Der notwendige HTML-Code für das Applet wird ebenfalls durch diese Klasse bereitgestellt. Die Anwendung auf dem Server bedient damit den JWAM Thin Client vollständig mit allen Diensten, es muß keine weitere Hilfsapplikation benutzt werden.

5.5 Offene Fragen

Der vorgestellte JWAM Thin Client ist vollständig funktionsfähig. Jedoch gibt es noch einige Probleme, die durch die grundlegende Art der Konstruktion eines Thin Clients bedingt sind, oder aber durch die konkrete Implementierung des JWAM Thin Client hervorgerufen werden.

[35] Eine Klasse laden heißt, daß der Softwarecode einer Klasse geladen wird.
[36] Sicherheitsbeschränkung für Applets: Eine Netzverbindung darf nur zu dem Rechner aufgenommen werden, von dem auch das Applet geladen wurde (siehe dazu auch Kapitel 2.3.1.3)

5.5.1 Asynchronität in der GUI

Das asynchrone Verhalten der GUI ist für den JWAM Thin Client gewollt. Bei ULC ist ein solches Verhalten noch ausgeprägter vorhanden. Dieses asynchrone Verhalten erfordert vom Benutzer einen anderen Umgang mit dem Werkzeug, als wenn es lokal vorhanden wäre. Die Reaktion auf seine Aktionen an der GUI kommt unter Umständen verzögert an. Der Hinweis mittels eines veränderten Mauszeigers, daß seine Aktion noch nicht bearbeitet wurde, kann nur ein Ersatz für eine Rückkopplung mittels einer aktualisierten GUI sein.

Das Verhalten des JWAM Thin Client ähnelt von der Verzögerung her dem einer Webanwendungen: Es wird ein Formular ausgefüllt und abgeschickt, wobei die Antwortzeit durch das Netz und die Auslastung des Servers bedingt unterschiedlich sein kann. Dabei ist es unbedeutend, ob es sich in den Augen des Benutzers um eine ‚kleine' Anfrage handelt, die von der Komplexität her sofort zu lösen sein müßte. Das eine Anfrage noch andauert, ist bei einem Browser an speziellen Oberflächenelementen sichtbar:

Abbildung 5-10: Benutzerrückkopplung im Browser (Beispiel Netscape)

Der Benutzer hat keine unmittelbare Rückkopplung. Sofern die Antwort länger auf sich warten läßt, versuchen Benutzer, die Anfrage nochmals auszuführen, abzubrechen oder gar andere Aktionen durchzuführen. So kann es zu unkontrollierten Folgeaktionen kommen, deren Ausführung ebenfalls durch Übertragungsprobleme oder zu lange Antwortzeiten des Servers verhindert werden kann.

Eine weitere Fehlerquelle für einen asynchronen Thin Client ist die verzögerte Übertragung. Innerhalb einer solchen Zeitspanne kann der Benutzer an der Oberfläche des Softwarewerkzeuges Einstellungen vornehmen, so daß spätere Abfragen des Servers nicht mehr den Zustand der Oberfläche vorfinden, der beim Aussenden des Ereignisses aktuell war.

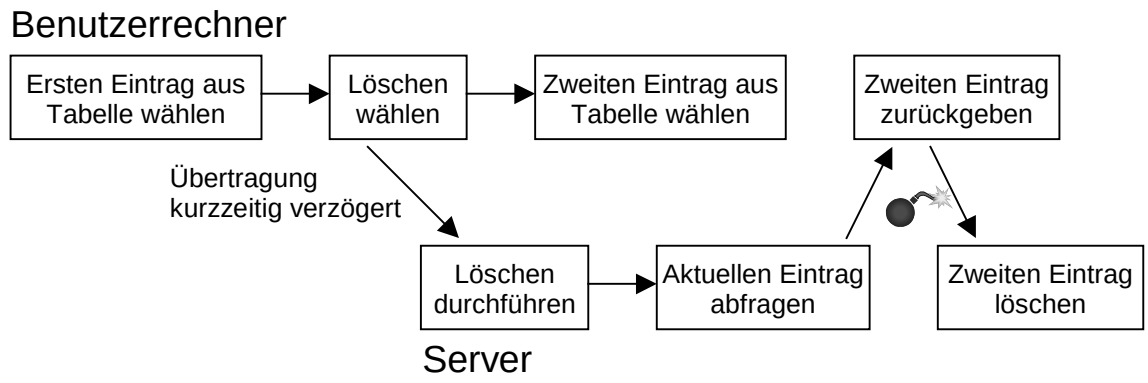

Abbildung 5-11: Fehler durch verzögerte Übertragung

Diese Art von Fehlerquelle in Abbildung 5-11 kann letztlich nur durch ein synchrones Modell behoben werden. Die Oberfläche müßte dazu gesperrt werden, solange ein Ereignis ausgelöst und

noch nicht bearbeitet wurde. Gerade bei einer langsamen Netzverbindung wird dann aber die Anwendung auf dem Benutzerrechner unhandlicher, das Softwarewerkzeug ist unter Umständen nicht mehr ‚flüssig' zu bedienen.

Eine Möglichkeit, solche Fehler zumindest teilweise zu verhindern, ist durch die speziellen Commands in JWAM gegeben. So übermittelt beispielsweise ein `cmdSelect`, das bei einer Selektierung eines Eintrages aus einer Liste ausgelöst wird, das selektierte Element. Wird diese Information ausgewertet, ist keine weitere Abfrage des Benutzerrechners mehr nötig und es kann zu keinen Fehlern aufgrund des asynchronen Verhaltens kommen. Ein solches Vorgehen erfordert die Disziplin des Anwendungsentwicklers, was aber nicht garantiert werden kann. Außerdem wird es in der zukünftigen Version von JWAM keine speziellen Commands mehr geben (siehe auch 5.5.4).

5.5.2 Abhängigkeiten der Klassen & Class-Loading

Der gesamte Softwarecode des Thin Client wird vom Server geladen. Bei einem einfachen Werkzeug handelt es sich dabei um ca. 70 Klassen, die insgesamt eine Größe von 180 KB haben. Geladen werden nicht nur Klassen aus dem JWAM Framework, sondern auch Klassen der verteilten Anwendung. In einer Anwendung können eigene Präsentationsformen und Fachwerte vorkommen, die auf den Benutzerrechner als Code verteilt werden.

Da die Klassen beim Start des Applets auf dem Benutzerrechner neu geladen werden, ist zwar das Problem des Softwaredeployments gelöst. Jedoch ist der Start des Applets damit verlangsamt, da alle benötigten Klassen *immer* geladen werden. In dem Prozeß einer Anwendungsentwicklung können zwar Präsentationsformen und Fachwerte immer wieder verändert werden, so daß diese Klassen auch erneut geladen werden müssen. Die Klassen aus dem JWAM Framework hingegen bleiben unverändert.

Daher bietet es sich an, auf dem Benutzerrechner zumindest die notwendigen Klassen aus dem JWAM Framework zu installieren. Leider hat sich eine solche Methode bei der Konstruktion als fehlerbehaftet erwiesen. Der Class-Loading-Mechanismus in Java konnte nicht mehr einwandfrei entscheiden, wann eine Klasse lokal vorhanden war und wann eine Klasse vom Server geladen werden mußte. Dieses Problem trat insbesondere dann auf, wenn die Oberklasse lokal und die Unterklasse auf dem Server vorhanden war, wie z.B. bei Fachwerten: Die Fachwerte der Anwendung werden vom Server geladen, während die Oberklasse aller Fachwerte aus dem Framework auf dem Benutzerrechner installiert ist. Die Installation einiger Klassen auf dem Benutzerrechner hat aber noch einen weiteren Nachteil: Sofern diese Klassen geändert werden, muß eine neue Installation vorgenommen werden, das Softwaredeployment wird dadurch komplexer.

Eine andere Möglichkeit für ein optimales Softwaredeployment wäre ein Caching der Klassen, das es für den Class-Loading-Mechanismus in Java nicht gibt: Der Class-Loader würde geladene Klassen lokal speichern und sie nur dann wieder vom Server laden, falls sich lokale und entfernte Klassen unterschieden. Für diese Konstruktion müßte ein neuer Class-Loader entwickelt werden, was in der Spezifikation von Java auch möglich und vorgesehen ist [Sun 99]. Würde für das Applet auf dem Benutzerrechner ein neuer Class-Loader eingesetzt werden, müßte das Applet

signiert sein. Der Zugriff auf das lokale Filesystem zwecks Caching der Klassen erfordert ebenfalls eine Signierung des Applets. Damit würde die Konstruktion des Applets komplexer (siehe 2.3.1.3).

Ein weiteres Problem ist die Abhängigkeit der Klassen untereinander. So befindet sich beispielsweise in dem Fachwert `dvThingDescription`, der ein Material beschreibt, eine Referenz auf die Klasse des beschriebenen Materials. Dieser Fachwert kann auf den Benutzerrechner übertragen werden, womit über die Referenz auch die Materialklasse und alle davon abhängigen Klassen übertragen werden. Da jede Klasse vom Server geladen wird, steigt damit die übertragene Datenmenge, obwohl nie ein Exemplar einer solchen Klasse erzeugt wird.

Solche Abhängigkeiten über Klassenobjekte können dadurch vermieden werden, daß nicht das Attribut mit dem Klassenobjekt serialisiert wird, sondern nur ein Attribut mit dem Namen der Klasse. Wird dann versucht, in einem serialisierten und wieder deserialisierten Objekt auf das Klassenobjekt zuzugreifen, wird anhand des Namens der Klasse das Klassenobjekt ermittelt und zurückgegeben. Würde dieses beispielsweise für den Fachwert `dvThingDescription` auf dem Benutzerrechner stattfinden, müßten dann allerdings auch die Materialklasse und alle abhängigen Klassen geladen werden. Bislang greift aber keine Klasse im JWAM Framework, die sich bei einem Thin Client auf dem Benutzerrechner befindet, auf diese Abfrage des Klassenobjektes zu. Somit bleibt dieses Problem nur für die aktuelle Version gelöst.

Da auf dem Benutzerrechner letztlich ein Teil des JWAM Frameworks abläuft, ist man auch an die Version des Java Development Kit (JDK) gebunden. Das JWAM Framework arbeitet nur mit dem JDK 1.2 zusammen, da in vielen Klassen Eigenschaften des JDK 1.2 benutzt werden, die in vorigen Versionen noch nicht vorhanden waren. Der Benutzerrechner muß dann auch eine entsprechende Version des JDK unterstützen, was den Benutzerkreis wieder einschränkt (siehe Verfügbarkeit des JDK auf unterschiedlichen Betriebssystemen [Javasoft 00]).

5.5.3 Programmiertransparenz

Werkzeuge müssen für einen Thin Client nicht speziell konstruiert werden, solange die Trennung von Interaktionsform und Präsentationsform konsequent verwendet wird. Sollen Werkzeuge als Thin Clients verfügbar sein, so muß nur ein anderer GUI-Kontext verwendet werden und das Startup der Werkzeuge angepaßt werden. Die eigentlichen Anwendungslogik wird nicht verändert.

Das JWAM Framework läßt dem Anwendungsentwickler die Option offen, eigene GUI-Kontexte zu verwenden. Der Desktop innerhalb des Frameworks verwendet beispielsweise einen eigenen GUI-Kontext, um an der Trennung von Interaktion und Präsentation vorbei auf die GUI zuzugreifen. Leider ist ein solches Verhalten für den Desktop und eventuell auch für andere Werkzeuge nötig, sofern spezielle Features der GUI genutzt werden müssen. Solche Werkzeuge (und damit auch der Desktop) können daher nicht mittels des JWAM Thin Clients auf Benutzerrechner verteilt werden.

Sollte es zu einem Fehlerfall in der Verbindung zwischen Benutzerrechner und Server kommen, hat das Werkzeug keine Möglichkeit, über diesen Fehler informiert zu werden. Dieses

Verhalten ist vollkommen transparent[37]. Wird beispielsweise die Verbindung dauerhaft getrennt, bleibt das Werkzeug auf dem Server weiter aktiv. Es hat keine Möglichkeit, auf den Verbindungsfehler zu reagieren.

Eine Lösung dieses Problems wäre die Option, daß Benutzerrechner bei Abbruch der Verbindung und Wiederaufbau die GUI rekonstruieren können. In den Interaktionsformobjekten kann der Zustand der Oberfläche gesichert werden, so daß man die GUI auf dem Benutzerrechner als zustandslos ansehen kann. Eine Rekonstruktion wird damit möglich.

In den Interaktionsformobjekten müßte das Caching so erweitert werden, daß zuletzt gesetzte Daten dauerhaft aufgehoben werden, um eine GUI wiederherzustellen. Weiter müßte der Verlauf der Anmeldung von Commands und Präsentationsformen gesichert werden, damit dieser wieder nachvollzogen werden kann. Durch eine solche Rekonstruktionsmöglichkeit ist dann eine vollkommene Transparenz in bezug auf Verbindungsfehler vertretbar: Kommt es zu einem Fehler, kann der alte Zustand wiederhergestellt werden, ohne daß das Werkzeug davon beeinträchtigt wird.

Eine andere Möglichkeit wäre, daß das Werkzeug über einen Fehler informiert wird, beispielsweise durch Runtime-Exceptions, und dann entsprechend reagieren kann. Bei klassischen Werkzeugen mit einer lokalen GUI können solche Verbindungsprobleme an dieser Stelle nicht auftreten. Ein Fehlerfall kann somit behandelt werden, allerdings ist damit die Transparenz in der Werkzeugkonstruktion nicht mehr vorhanden. Das Werkzeug hat damit spezielles Wissen über die benutzte GUI.

5.5.4 Erweiterungen der Konstruktion

Die Präsentationsformen einer Anwendung werden automatisch verteilt, so daß neue Präsentationsformen einfach eingebunden werden können. Sollte eine neue Interaktionsform hinzukommen, so muß für diese eine neue Klasse entwickelt werden, deren Exemplare dann die notwendigen Interaktionsformobjekte bilden. Die Konstruktion einer solchen Klasse beschränkt sich jedoch nur auf die Implementierung von ‚Durchreichemethoden': Aufrufe werden an das Serverobjekt weitergeleitet. Dabei muß auf ein Caching von Attributen geachtet werden.

Falls im Zuge einer neuen Interaktionsform ein neues Command entwickelt wird, muß der `CommandTransformer` um das Wissen über dieses neue Command erweitert werden. Die Einbindung beschränkt sich auf das Aus- bzw. Einlesen der Attributes des Commands und auf Erzeugung und Anbindung eines Interessenten an das Command. Der Aufwand bleibt auf wenige Zeilen Code beschränkt.

In der nächsten Version von JWAM (Version 1.6) wird es keine speziellen Command-Klassen mehr geben, sondern nur noch die Klasse `cmdObject`. Damit entfallen auch Änderungen am `CommandTransformer`. Diese Umstellung in JWAM 1.6 vereinfacht nebenbei die Konstruktion des Thin Clients und vermindert die Anzahl der zu übertragenen Klassen, da nicht

[37] Im Sinne von ‚nicht sichtbar'

mehr unterschiedliche Command-Klassen für die verschiedenen Präsentationsformen übertragen werden müssen.

Eine andere Möglichkeit der Erweiterung wäre der Austausch der Middleware. Dabei sind drei Punkte zu beachten:

- **Klassenverteilung:** RMI bietet ohne zusätzlichen Aufwand[38] eine einfache Möglichkeit, den Softwarecode der benutzten Klassen zu verteilen. Dieses Verhalten muß eventuell durch einen eigenen Class-Loader nachgebildet werden.

- **Kommunikation:** Die Kommunikation zwischen Client- und Serverobjekt findet hauptsächlich durch das Übermitteln von Methodenaufrufen statt, die als Objekte der Klasse `MethodCall` vorhanden sind. Diese müßten nur als Binärdaten übertragen werden, indem die Objekte serialisiert werden. Über RMI finden aber auch Methodenaufrufe vom Server- an das Clientobjekt statt und umgekehrt. Dazu muß ein geeigneter Mechanismus gefunden werden, eine Nutzung der `MethodCall`-Struktur bietet sich an.

- **Listener:** Zur Initialisierung der Kommunikation ist das Listenerobjekt zuständig. Es müßte ebenfalls auf eine neue Middleware angepaßt werden.

Die Änderungen bleiben auf die Klassen des Client-, des Server- und des Listenerobjekt beschränkt sowie die dazugehörigen Threads. Über Schnittstellen ist das Verhalten der ersten drei Objekte definiert, so daß ein Austausch von Implementationen möglich ist.

5.5.5 Trennung FK/IAK?

Werkzeuge sind nach WAM in eine Funktionskomponente und eine Interaktionskomponente aufgeteilt. In JWAM ist dieses prinzipiell so realisiert, daß es je Komponente eine Klasse gibt, in der die Aufgaben der Komponente implementiert sind. Das Werkzeug besteht dann aus je einem Exemplar dieser Klasse[39]. Da die Interaktionskomponente stark mit der GUI verknüpft ist, wäre ein erster Gedanke, nicht nur die GUI sondern auch die Interaktionskomponente auf dem Benutzerrechner ablaufen zu lassen. Dagegen sprechen aber einige Punkte:

- Funktions- und Interaktionskomponente eines konkreten Werkzeuges werden vom Anwender entwickelt, wozu eigene Klassen entwickelt werden, die von Oberklassen aus dem Framework erben. Die Interaktionskomponente ruft die Funktionskomponente direkt auf, so daß solche Aufrufe verteilt werden müßten. Damit wäre in dem Werkzeug ein direktes Wissen über die Verteilungstechnik vorhanden, das Werkzeug müßte daran angepaßt werden. Die Transparenz der Programmierung wäre nicht mehr gewährleistet.

- Zwischen Interaktions- und Funktionskomponente kann ein komplexes Zustandsmodell vorhanden sein. Die Wiederherstellung der Verbindung im Fehlerfall wäre komplexer.

[38] Von der Einrichtung eines Webservers einmal abgesehen, der aber hier schon integriert ist (5.4.3).
[39] Hinzu kommt noch eine Werkzeugklasse (`Tool`).

- Die Interaktionskomponente ist stärker mit dem Framework verbunden als die Präsentationsformen. Daher müßte mehr Softwarecode auf den Benutzerrechner gebracht werden. Die Anwendung auf dem Benutzerrechner wäre ein Fat Client.

- Das JWAM Framework bietet auch eine Möglichkeit, Werkzeuge monolithisch zu konstruieren: Funktion und Interaktion sind in einer Klasse vereint. Diese Art der Werkzeug-Konstruktion würde für den Thin Client ausscheiden, es wäre eine weitere Einschränkung vorhanden.

Diese teils massiven Hindernisse sprechen gegen einen Schnitt zwischen Funktion und Interaktion. Ein solcher Schnitt wäre einfach nicht sinnvoll.

5.6 Zusammenfassung

In diesem Kapitel habe ich dargestellt, was für Erweiterungen am JWAM Framework und in den Werkzeugen einer Anwendung durchgeführt werden müssen, um JWAM Thin Clients laufen zu lassen. Um einen Überblick über alle Punkte der notwendigen Erweiterungen zu geben, fasse ich diese hier nochmals checklistenmäßig zusammen, zuerst die Änderungen am Framework:

- Singletons, die Informationen für einen Benutzerprozeß bereitstellen, wie z.B. das `Environment`, müssen in Prozeßsingletons umgewandelt werden. Dazu muß die `instance()`-Methode angepaßt werden und die Klasse sollte das Interface `ProcessSingelton` implementieren.

- Die übriggeblieben Singletons sollten das Interface `Singleton` implementieren. Weiter müssen diese Klassen synchronisiert werden, so daß auf die Methoden nebenläufig zugegriffen werden kann.

- Falls ein Singleton nur einzelne Informationen je nach Benutzerprozeß bereitstellen sollte und es sich lohnt, die Singleton-Konstruktion aufrecht zu erhalten, wie z.B. bei der allgemeinen Fabrik für Fachwerte, so kann das Singleton erhalten bleiben. Die prozeßspezifischen Informationen müssen dazu über den `ProcessManager` je Benutzerprozeß gespeichert werden.

- Alle Klassenmethoden müssen, sofern es notwendig ist, synchronisiert werden.

Für den Helpdesk habe ich diese Änderungen soweit durchgeführt, daß die Werkzeuge des Helpdesks in einem solch modifizierten Framework als Thin Clients lauffähig sind. Bei der Entwicklung eines Werkzeuges, das mittels eines JWAM Thin Clients verteilt werden soll, müssen folgende Punkte beachtet werden:

- Die GUI muß in einer Klasse implementiert sein. Als GUI-Bibliothek kann nur Swing oder AWT benutzt werden.

- Der Zugriff auf die GUI darf nur über Interaktionsformen geschehen. Das Werkzeug darf keinen eigenen Kontext benutzen, es muß vielmehr den Kontext für Thin Clients verwenden.

- Anwendungsspezifische Präsentationsformen können benutzt werden. Sofern anwendungsspezifische Interaktionsformen oder Command eingeführt werden sollen, muß die Konstruktion des Thin Clients erweitert werden (siehe 5.5.4).

- Präsentationsformen oder Fachwerte, die von Präsentationsformen dargestellt werden, sollten möglichst nicht komplexe Klassenstrukturen nutzen, da der Softwarecode dieser Klassen dann auf den Benutzerrechner übertragen werden muß (siehe 5.5.2).

- Das Startup der Anwendung muß angepaßt werden.

Sofern alle diese Punkte und Einschränkungen erfüllt wurden, können die Werkzeuge einer Anwendung als JWAM Thin Client verteilt werden. Es kommt zu keinen Änderungen an der eigentlichen Anwendungslogik.

6 Abschluß

„Ein Anwendungssystem soll möglichst von überall her nutzbar sein“ ist die Anforderung aus der Einleitung der Diplomarbeit. Damit diese Anforderung erfüllt werden kann, muß der Zugang durch einen beliebigen Benutzerrechner über das Internet möglich sein. Es muß ein optimaler Schnitt durch das Anwendungssystem gefunden werden, so daß ein Teil des Systems unproblematisch auf einen beliebigen Benutzerrechner verteilt und von dort ein Zugriff auf das Anwendungssystem realisiert werden kann.

Ich habe dazu verschiedene Schnittmöglichkeiten aufgezeigt, wobei auch ein mehrfacher Schnitt nötig ist, um ein System optimal zu verteilen. Ich habe einige Technologien für den Schnitt zwischen Benutzerrechner und Server vorgestellt, die vor allem eine wichtige Anforderung erfüllen: ein einfaches Softwaredeployment. Sofern zum Ausführen einer Anwendung vorher anwendungsspezifischer Code explizit installiert werden muß, ist das ein großes Hindernis, um ein Anwendungssystem von überall her benutzen zu können. Weitere Punkte für einen optimalen Schnitt sind die Robustheit gegenüber Schnittstellenänderungen, Skalierung, Lastverteilung, Performance und Transparenz. Die vorgestellten Technologien erfüllen diese in unterschiedlicher Güte.

Bei allen Technologien, bis auf die Verteilungssysteme auf grafischer Basis, können bestehende Anwendungen nicht wiederverwendet werden: Soll beispielsweise eine Anwendungen über einen Browser verfügbar sein, muß eine WWW-Schnittstelle neu entwickelt werden. Günstigenfalls ist der Kern der Anwendungslogik nicht zu sehr mit einem Frontend verknüpft, so daß eine Schnittstelle für ein weiteres Frontend relativ einfach entwickelt werden kann, wie auch im Prototyp Helpdesk gezeigt (siehe [Otto/Schuler 00]).

Eine Lösung stellen Thin Clients dar, die am meisten den Verteilungssystemen auf grafischer Basis ähneln: Der Schnitt für die Verteilung geht durch die Präsentationslogik, so daß unter bestimmten Voraussetzungen bestehende Anwendungen ohne Änderungen an der Anwendungslogik einfach auf Benutzerrechner verteilt werden können. Da im Gegensatz zu Verteilungssystemen auf grafischer Basis auf dem Benutzerrechner die eigentliche GUI einer Anwendung ausgeführt wird, ist die gesamte Architektur performanter. Thin Clients bieten durch ihre grundlegende Konstruktion auch prinzipiell die Möglichkeit, daß die GUI auf dem Benutzerrechner rekonstruiert werden kann, falls die Kommunikation zwischen Benutzerrechner und Server abbricht.

Mit dem JWAM Thin Client habe ich zeigen können, daß sich bestehende Anwendungen, die mit dem JWAM Framework konstruiert wurden, ohne Änderungen an der Anwendungslogik als Thin Clients realisieren lassen. Das Framework bietet über den IAF/PF-Mechanismus eine Abstraktion von der verwendeten GUI-Bibliothek, so daß eine Verteilung der GUI einfach möglich ist. Für den JWAM Thin Client bedeutet das, daß ein Thin Client ‚mit eigenen Mitteln‘ aus dem JWAM Framework entwickelt werden konnte: Die Trennung von Interaktion und Präsentation ermöglicht es, zwischen diesen Komponenten den Schnitt für den Thin Client zu setzen. Ich habe auch gezeigt, daß sich durch diese Abstraktion von der GUI-Bibliothek ULC zur Realisierung eines Thin Clients in das JWAM Framework integrieren läßt.

Da die Anwendungen bei einem Thin Client nicht auf den Benutzerrechnern ablaufen, sondern zusammen auf einem Server, mußte das JWAM Framework sowohl für ULC als auch für den JWAM Thin Client an diese Gegebenheiten angepaßt werden. Aus Gründen der Performance ist es sinnvoll, die einzelnen Anwendungen in JWAM innerhalb einer Java VM ablaufen zu lassen. Da im Framework einige Konstrukte darauf beruhen, daß es nur eine Anwendung in einer VM gibt, mußten diese Stellen gezielt angepaßt werden. Diese Modifikationen haben ebenfalls keine Auswirkungen auf die Anwendungslogik, es muß keine Anpassung stattfinden.

Anhand des Prototyps Helpdesk habe ich demonstrieren können, daß die lokalen JWAM-Werkzeuge des Anwendungssystemes durch den JWAM Thin Client verteilt werden konnten. Durch die Architektur des JWAM Thin Client kann auf die WAM-Werkzeuge für den Helpdesk nun wie beim Webtop über das Internet zugegriffen werden. Die Werkzeuge bieten über ihre GUI einen vollwertigen Zugang zu dem Helpdesk, ohne daß es wie beim Webtop durch die Architektur zu Einschränkungen kommt: Beim Webtop ist keine Benachrichtigung des Benutzerrechners durch den Server möglich und durch die Verwendung von HTML gestaltet sich die Konstruktion aufwendiger.

Fraglich bleibt aber, ob es sich lohnt, statt einer Webanbindung einen Thin Client zu konstruieren. Gibt es für ein Anwendungssystem bislang noch keine JWAM-Anwendung oder keine Java-Anwendung, die mit dem JWAM Thin Client bzw. ULC verteilt werden kann, so kann es möglicherweise auch ausreichen, nur eine Webanbindung für einen Zugriff über das Internet zu entwickeln. Diese Entscheidung hängt von der Art der Anwendungen ab, die über das Internet genutzt werden soll. Ob ein Thin Client letztlich die Performance bei vielen gleichzeitig aktiven Benutzern bieten kann, die er verspricht, kann nur durch eine empirische Untersuchung festgestellt werden. Für den Helpdesk ließen sich leider keine sinnvollen Erfahrungsberichte erstellen.

Ich konnte leider ULC nicht neben dem hier vorgestellten JWAM Thin Client integrieren, da dieses Produkt erst seit kurzem zur Verfügung steht [Canoo 00]. Damit kann die Frage, ob ULC eine bessere Thin-Client-Technologie für das JWAM Framework ist als der JWAM Thin Client, nicht abschließend geklärt werden.

7 Literatur

[Barton 94] Robert Barton: **Die X/Motif Umgebung**, Springer Verlag 1994

[Bell/Grimson 92] David Bell, Jane Grimson: **Distributed Database Systems**, Addison-Wesley Publishing 1992, International Computer Science Series

[BLRSZ 99] Wolf-Gideon Bleek, Martin Lippert, Stefan Roock, Wolfgang Strunk, Heinz Züllighoven: **Frameworkbasierte Anwendungsentwicklung (Teil 3): Die Anbindung von Benutzungsoberflächen und Entwicklungsumgebungen an Frameworks**, ObjektSpektrum 3/99. 1999.

[BLZ 99] Bleek, W.-G., Lilienthal, C., Züllighoven, H.: **Frameworkbasierte Anwendungsentwicklung (Teil 4): Fachwerte**, ObjektSpektrum 5/99, September/Oktober 1999

[Bohlmann 98] Holger Bohlmann: **Einbindung der ULC-Infrastruktur in ein bestehendes Software-Entwicklungs-Framework (JWAM)**, Praktikumsbericht für die UBS Basel (Abteilung AEC), 1998

[Canoo 00] Canoo Engineering AG**: ULC Technical White Paper**, Basel 2000 (Schweiz), http://www.canoo.com

[Citrix 00] Citrix Systems**: Produktdokumentation zu WinFrame, MetaFrame und NFuse**, http://www.citrix.com

[CTRC 95] Computer Technology Research Corp**.: Client/Server Application Development – Tools and Techniques**, 1995

[ECMA 99] ECMA (Standardizing Information and Communication Systems): **ECMA 262 – Script Language Specification, 3rd Edition**, December 1999, http://www.ecma.ch

[ELW 98] Robert Eckstein, Marc Loy, Dave Wood : **Java Swing**, O'Reilly & Associates Inc., September 1998, First Edition

[Englander 97] Robert Englander: **Developing Java Beans**, O'Reilly & Associates Inc., 1997, First Edition

[Farley 98] Jim Farley: **Java – Distributed Computing**, First Edition, O'Reilly, 1998

[FGMFB 97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee: **Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068**, http://www.cis.ohio-state.edu/rfc/rfc2068.txt, Januar 1997

[Flanagan 98] David Flanagan: **Javascript: The Definitive Guide**, O'Reilly & Associates Inc., 1998

[Flanagan 99A] David Flanagan: **Java in a nutshell,** O'Reilly & Associates Inc., 1998

[Flanagan 99B] David Flanagan: **Java foundation classes in a nutshell,** O'Reilly & Associates Inc., 1998

[FLLRW 98] Niels Fricke, Carola Lilienthal, Martin Lippert, Stefan Roock, Henning Wolf: **Asynchrone Nachrichtenvermittlung für interaktive, kooperative Anwendungssysteme mit Java**. Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Universität Hamburg, 1998.

[GHJV 94] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Design Patterns**, Addison Wesley Longman, 1994

[Havenstein 99] Andreas Havenstein: **Unterstützung kooperativer Arbeit durch eine Softwareregistratur**, Studienarbeit am Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Hamburg, 1999.

[Hunter/Crawford 98] Jason Hunter, William Crawford**: Java Servlet Programming**, O'Reilly & Associates Inc., 1998

[Inprise 00] Inprise Corporation, **Produktinformationen zu JBuilder 4.0**, http://www.inprise.com/jbuilder/, November 2000

[Javasoft 00] Javasoft: **Java(TM) Platform Ports**, http://www.javasoft.com/cgi-bin/java-ports.cgi, November 2000

[JBR 99] Ivar Jacobsen, Grady Booch, James Rumbaugh: **The Unified Software Development Process**, Addison Wesley Publishing, Reading, Massachusetts, Januar 1999

[JWAM] **JWAM Dokumentation** und weitere Informationen, http://www.jwam.de, Apcon Workplace Solution

[Lippert 99] Martin Lippert: **Die Desktop-Metapher in Systemen nach dem Werkzeug- und Material-Ansatz**, Diplomarbeit, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Hamburg 1999

[Microsoft 00] Microsoft Online Workshop: **DHTML Dokumentation**, http://msdn.microsoft.com/workshop/author/default.asp

[MTZ 99] Silvano Maffeis, Fridtjof Toenniessen, Christian Zeidler: **Erfahrungen mit Java**, dpunkt-Verlag, Februar 1999

[Müller 99] Klaus Müller: **Konzeption und Umsetzung eines Fachwertkonzeptes**, Studienarbeit, Arbeitsbereich Softwaretechnik, Fachbereich Informatik, Universität Hamburg, 1999

[Münz 98] Stefan Münz: **SelfHTML**, eine Dokumentation zu HTML, Version 7.0, 1998, http://www.teamone.de/selfhtml/

[NCD 00] Network Computing Devices: **Produktinformationen zum NCD ThinStar**, http://www.ncd.com/products/hardware/, 2000

[Neoware 00] Neoware Systems Inc.: **Produktinformationen zur Neostation**, http://www.neoware.com/products.html, 2000

[Netscape 00] Netscape Developer Central: **DOM Spezifikationen**, http://developer.netscape.com/tech/dom/

[Oaks/Wong 97] Scott Oaks, Henry Wong: **Java Threads**, O'Reilly & Associates Inc., 1997, First Edition

[Orfali 99] Robert Orfali, Dan Harkey, Jeri Edwards: **Client/Server Survival Guide**, Third Edition, Wiley Computer Publishing, 1999

[Otto/Schuler 00] Michael Otto, Norbert Schuler: **Fachliche Services: Geschäftslogik als Dienstleistung für verschiedene Benutzungsschnittstellen-Typen**, Diplomarbeit, Universität Hamburg, 2000

[Reese 00] George Reese: **Database Programming with JDBC and Java**, O'Reilly & Associates, Second edition August 2000

[Shneiderman 98] Ben Shneiderman: „**Designing the user interface: strategies for effective human-computer interaction**". Third edition. Addision-Wesley, 1998.

[Singh 99] Harry Singh: **Progressing to distributed multiprocessing**, Prentice Hall PTR, 1999

[Sun 99] Sun Mircosystem: **Dynamic code downloading using RMI**, http://www.javasoft.com/products/jdk/1.2/docs/guide/rmi/codebase.html , Sun Microsystems 1999

[W3C] World Wide Web Consortium: **WWW-Spezifikationen,** http://www.w3c.org

[WAP] Wireless Application Protocol Forum: **Wireless Application Protocol White Paper**. http://www.wapforum.org, 2000.

[Züllighoven 98] Heinz Züllighoven et al.: **Das objektorientierte Konstruktions-handbuch**, 1. Auflage, dpunkt-Verlag, 1998