Workflow-Management im Web

Diplomarbeit an der Universität Ulm

Fakultät für Informatik

Vorgelegt von:

Ralf Kießling

1. Gutachter: Prof. Dr. Peter Dadam

2. Gutachter: Dr. Manfred Reichert

2001

Kurzfassung

Die Bedeutung des Web als Basis für die Entwicklung von Applikationen wird immer größer. Zahlreiche Vorteile, wie die Verfügbarkeit auf vielen Systemplattformen oder standardisierte Clients, sprechen für Web-basierte Lösungen. Daher bieten sich Web-Clients auch als Benutzerschnittstelle für Workflow-Management-Systeme (WfMS) an. Welche speziellen Anforderungen WfMS besitzen, und wie diese mit den besonderen Charakteristika von Web-Applikationen vereint weden können, wird im Rahmen dieser Arbeit geklärt. Außerdem werden Lösungsansätze für web-basierte WfMS aufgezeigt und bewertet. Ein Vergleich von Web-Anbindungen in existierenden WfMS, und eine Diskussion über die Einsatzmöglichkeiten von Web-Clients schließen die Arbeit ab.

Danksagung

Iucundi acti labores
(lat.) Erfreulich sind geleistete Arbeiten

In diesem Sinne möchte ich mich bei

- Herrn Prof. Dr. Peter Dadam
- Herrn Dr. Manfred Reichert
- Herrn Dipl.-Inform. Thomas Fries
- Herrn Dr. Thomas Bauer
- Herrn Cand.-Inform. Udo Martschat
- meinen Eltern
- meiner Freundin Christina
- und meinem Sohn Leo

für Ihre Unterstützung und die sehr gute Betreuung meiner Diplomarbeit bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Infolge der zunehmenden Globalisierung der Märkte werden die Unternehmen mit einer immer dynamischeren Umwelt konfrontiert. Der verschärfte Wettbewerb, Kostendruck, kurzlebige Trends, hohe Qualitätsanforderungen und permanente technische Innovationen erfordern eine immer schnellere Anpassung der Unternehmen an neue Rahmenbedingungen. Sie haben im Regelfall Änderungen der Organisationsstruktur und der betrieblichen Abläufe zur Folge. Die *Geschäftsprozeß-Optimierung (GPO)* und das *Geschäftsprozeß-Reegineering (GPR)* stellen Werkzeuge und Methoden zur Planung und Simulation bereit [HS95].

Das Problem besteht nun darin, diese Planungen effizient und effektiv in die Praxis umzusetzen. In diesem Zusammenhang stellt die Implementierung der neuen Abläufe in die bestehende EDV-Infrastruktur eine besondere Herausforderung dar. Zwar bieten viele Hersteller Standardlösungen für Branchensoftware in Bereichen wie *E-Commerce*, *Business-To-Customer (B2C)*, *Business-To-Business (B2B)*, *Enterprise Resource Planing (ERP)* und *Supply Chain Management (SCM)* an, doch hinsichtlich Flexibilität, Fehlersicherheit und Konfigurierbarkeit existieren gravierende Mängel. Abläufe sind in der Regel fest in den Anwendungsprogrammen codiert und können nur begrenzt durch Parameter beeinflußt werden. Im Gegensatz dazu erlaubt individuell programmierte Anwendungssoftware (mit fest implementierter Ablauflogik) direkte Veränderungen im Quellcode, allerdings sind hierfür Spezialisten, Neuinstallationen und vor allem aufwendige Tests erforderlich, was insgesamt zu hohen Entwicklungs- und Wartungskosten führt. Fehlende Standards und proprietäre Schnittstellen erschweren zudem die Interoperabilität der Systeme untereinander.

Die dezentrale Ausführung der Anwendungsprogramme in unterschiedlichen Bereichen des Unternehmens führt zu weiteren Problemen in Bezug auf Sicherheit, Stabilität und Konsistenz. Deshalb sind hier flexible, leicht anpaßbare, plattformübergreifende und vor allem zuverlässige Softwarelösungen gefragt, um den Ablauf der Geschäftsprozesse adäquat zu unterstützen. *Workflow Management Systeme* [JBS99, RD00, HM99] bieten hierfür einen vielversprechenden Ansatz.

1.1 Workflow-Management

Die Denkweise im Zusammenhang mit betrieblichen Abläufen innerhalb eines Unternehmens wird von dem zentralen Begriff des *Geschäftsprozesses (GP)* geprägt [Sch98a]. Geschäftsprozesse bezeichnen größere Einheiten von Abläufen in einer Organisation. Sie sind durch einen wesentlichen Beitrag zum Geschäftserfolg, eine klar abgegrenzte Themenstellung, eine Strategie zur Zielerreichung und einen meßbaren Nutzen gekennzeichnet [JBS99, Här97].

Workflows (WF) stellen die elektronische Repräsentation von Geschäftsprozessen oder Teilen davon dar. Die automatisierte Steuerung von konkreten Geschäftsfällen bzw. Prozessen, der Austausch von Dokumenten und Informationen, sowie die Zuordnung von Aufgaben an potentielle Bearbeiter auf der Grundlage definierter Regeln wird als *Workflow-Management (WfM)* bezeichnet [Wor95]. Die Definition, Verwaltung und Ausführung der Workflows wird mit Hilfe spezieller Softwaresysteme durchgeführt. Sie werden *Workflow-Management-Systemen (WfMS)* genannt.

„Echte“ WfMS realisieren die Trennung von Ablauflogik und Applikationscode (vgl. Abbildung 1.1). Sie ermöglichen somit die Modellierung der Geschäftsprozese auf einer implementierungneutralen Ebene [DR00]. Systemnahe Probleme wie Synchronisation, Persistenz oder Fehlerbehandlungen werden vom WfMS, ähnlich wie bei einem *Datenbank-Management-System (DBMS)* [Dad96], behandelt. Der Anwendungsentwickler kann sich deshalb auf seine wesentlichen Aufgaben, die Umsetzung der Geschäftsprozesse, konzentrieren.

Viele WfMS bieten Schnittstellen zum Austausch von Prozeßmodellen und zur Interoperabilität mit anderen WfMS [Mar01]. Dies spielt vor allem im Kontext von *E-Business* und unternehmensübergreifenden Abläufen, wie sie im Bereich *B2B* und *B2C* vorkommen, eine große Rolle [Gar99b].

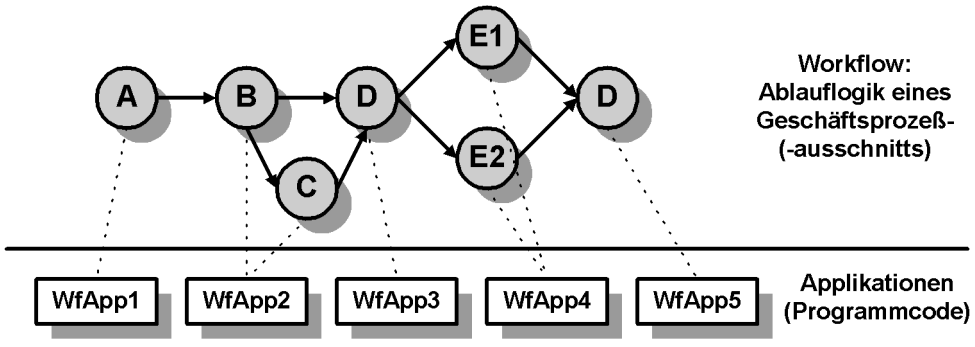

Abbildung 1.1: Trennung von Ablauflogik und Applikationscode

1.2 World Wide Web

Aus einem vom US-Verteidigungsministerium in den 60er Jahren initiierten Forschungsprojekt enstand innerhalb der letzten Jahrzehnte eines der bedeutendsten Computernetzwerke - das *Internet* [Tan98]. Es verbindet durch seine Protokolle weltweit Rechner und Netzwerke und ermöglicht Millionen von Anwendern den Zugang zu seinen Diensten. Einen der populärsten Internetdienste stellt das *World Wide Web* (kurz: *WWW* oder *Web*) dar.

Die Geschichte des WWW begann 1989 am CERN, dem europäischen Zentrum für Atomphysik [Tan98]. Die Wissenschaftler suchten nach einem Informationssystem, um ihre Forschungsdaten bequem und schnell austauschen zu können. Aufgrund der internationalen Zusammensetzung des Forschungsteams und der unterschiedlichen Struktur der Daten (formatierte Texte, Bilder, usw.) stellten die bestehenden Kommunikationsmittel nur eine unbefriedigende Lösung dar. Außerdem wollte man Hypertextfunktionalität einbauen, die es erlauben sollte, in einem Dokument Verweise auf andere Dokumente einzubauen. Diese Verweise sollten auch auf Dokumente zeigen, welche auf anderen Rechnern gespeichert waren, so daß eine einfache Navigation zwischen den Inhalten verschiedener Rechner möglich wird.

Zur Beschreibung der Dokumentinhalte entwickelte man die *Hypertext Markup Language (HTML)* und spezifizierte das *Hypertext Transfer Protocol (HTTP)* zu deren Übertragung [Wor01e, Wor01l]. Es folgten bald erste (Web-)Server, welche HTTP unterstützten, und Client-Programme mit einer (graphischen) Benutzeroberfläche (*Web-Browser*) zur Darstellung der Inhalte. Die ansprechende Präsentation und die leichte Navigation mit Hyperlinks sorgten für die schnelle Verbreitung und die große Popularität des WWW in den folgenden Jahren.

In den letzten Jahren hat das WWW, neben der Nutzung als Informationssystem, eine immer größere Bedeutung als Basis für Anwendungen bekommen. Die Web-Server bieten, neben der Grundfunktionalität als Server für HTTP-Anfragen, eine Plattform für sog. *Web-Applikationen* , welche auf dem Web-Server ablaufen und die Web-Clients als Bedienerschnittstelle nutzen. Die übertragenen Inhalte sind in der Regel für jede Applikation, jeden Benutzer und jeden Zugriff individuell zugeschnitten, und können nicht mehr wie die bisherigen Dokumente statisch auf einem Speichermedium abgelegt werden. Stattdessen müssen sie zur Laufzeit dynamisch vom Web-Server generiert werden. Die Hersteller der Web-Server haben unterschiedliche Verfahren für die Erstellung dynamischer Inhalte entwickelt. Diese Verfahren werden in Kapitel 4 näher betrachtet und bewertet. Die Vielzahl unterschiedlicher Arten von Clients mit verschiedenen Anforderungen und Möglichkeiten (vgl. Kapitel 5) stellt zudem neue Herausforderungen für die Web-Server dar. Der Einsatz von Web-Applikationen in einem heterogenen Umfeld, zum Beispiel im *Intranet* (internes Netz eines Unternehmens), im *Extranet* (unternehmensübergreifendendes Netz mehrerer Geschäftspartner) oder im Internet, stellt unterschiedliche Anforderungen an die jeweils bereitgestellte Funktionalität oder Sicherheitsmechanismen.

1.3 Aufgabenstellung und Motivation

Die Technologien des Internets wurden für kurz andauernde Interaktionen (z.B. Auskunftsdienste, Suchmaschinen) entwickelt. Eine Unterstützung für lang andauernde, prozeßorientierte Anwendungen, wie sie zum Beispiel WfMS bieten, gibt es in der Regel nicht [Wor98b]. Genau hier liegt die Herausforderung, diese beiden Technologien miteinander zu verbinden, um die Workflow-Funktionalität über eine einheitliche und weit verbreitete Schnittstelle vielen Anwendern zugänglich zu machen.

Ziel dieser Diplomarbeit ist es, Verfahren und Technologien zu untersuchen und zu bewerten, welche für eine Web-Anbindung von WfMS geignet sind. Dazu sollen sowohl die Anforderungen von WfMS als auch Charakteristika von Web-Applikationen betrachtet werden. Außerdem sollen Lösungen für Web-basierte Workflow-Anwendungen vorgestellt und bewertet werden. Ein ausgewählter Lösungsansatz wird durch einer Beispielimplementierung vertieft. Abgerundet wird die Arbeit durch einen Überblick über die Web-Schnittstellen bestehender WfMS und eine Diskussion über die Einsatzmöglichkeiten von Web-basierten Workflow-Anwendungen.

1.4 Gliederung

Kapitel 2 beschäftigt sich mit Workflow-Management-Systemen, ihren speziellen Anforderungen und Standards aus dem Worflow-Management-Bereich. Besondere Bedeutung erfahren hier die Schnittstellen, welche ein WfMS zur Interaktion mit Klienten oder anderen Softwaresystemen (z.B. Web-Servern oder anderen WfMS) bereitstellt.

In Kapitel 3 werden wichtige Grundlagen zum besseren Verständnis Web-basierter Applikationen vermittelt. Es werden Protokolle, Web-Inhalte und Architekturkonzepte vorgestellt. Abschließend werden die Zusammenhänge an zwei Beispielen verdeutlicht und die Vor- und Nachteile von Web-basierten Applikationen unter Beachtung der speziellen WfMS-Anforderungen zusammengefaßt.

In Kapitel 4 werden Möglichkeiten aufgezeigt, wie WfMS mit Hilfe von Web-Servern mit ihren Clients kommunizieren können. Einen besonderen Schwerpunkt bildet die Anbindung der Web-Server durch Komponenten oder Programmierschnittstellen an das WfMS. Außerdem werden Techniken zur effizienten Erstellung Web-basierter Workflow-Applikationen diskutiert.

Die Aufgaben und Möglichkeiten von Workflow-Clients werden in Kapitel 5 vorgestellt. Neben der Darstellung von Workflow-relevanten Inhalten bieten sie auch begrenzte Möglichkeiten zur Implementierung von Applikationslogik. Die Vorteile, Nachteile und Grenzen der Client-seitigen Programmausführung werden ebenfalls besprochen.

In Kapitel 6 wird eine Implementierungsmöglichkeit für ein Web-basierte WfMS detailliert vorgestellt. Dabei kommen Verfahren zum Einsatz, welche in den Kapiteln 4 und 5 besprochen wurden. Die Vielfalt der Web-Clients (z.B. Web-Browser, Mobiltele-

fon, usw.), mit teilweise unterschiedlichen Anforderungen und Möglichkeiten, macht die Erstellung von Anwendungen für mehrere Clients sehr komplex. Anhand einer Client-unabhängigen Formularbeschreibung soll ein Verfahren zur einfachen Anwendungserstellung vorgestellt werden.

Das Kapitel 7 beschäftigt sich mit ausgewählten Vertretern Web-basierter WfMS. Es werden sowohl prototypische Systeme aus der Workflow-Forschung als auch kommerzielle Systeme vorgestellt und bewertet.

In Kapitel 8 werden die Vor- und Nachteile von Web-Clients im Vergleich zu konventionellen Workflow-Clients diskutiert, und die daraus resultierenden Einsatzzwecke vorgestellt.

Abschließend wird in Kapitel 9 eine Zusammenfassung der in der Arbeit gewonnenen Erkenntnisse erstellt, und es werden Perspektiven für die zukünftige Entwicklung von Web-basierten WfMS aufgezeigt.

Kapitel 2

Workflow-Management-Systeme

Workflow-Management und die mit ihm in Zusammenhang stehenden Begriffe werden oft mit unterschiedlichem Verständnis gebraucht. Deshalb sollen zuerst grundlegende Begriffe geklärt und durch eine Abgrenzung gegen andere verwandte Thematiken ein besseres Verständnis geschaffen werden. Die unterschiedlichen Ansätze für die Implementierung von WfMS werden in Abschnitt 2.3 vorgestellt. Eine Analyse der grundlegenden Anforderungen an WfMS wird in Abschnitt 2.4 durchgeführt. Sie dient im weiteren Verlauf der Arbeit auch zur Bewertung der unterschiedlichen Lösungungsalternativen für Web-basierte WfMS. Abschließend werden Standards aus dem WfMS-Bereich vorgestellt und beurteilt.

2.1 Begriffe

Wie in Kapitel 1 erwähnt, dienen WfMS der elektronischen Unterstützung von Geschäftsprozessen. Um Geschäftsprozesse bzw. Ausschnitte davon durch Workflows abbilden zu können, benutzen WfMS ein *Workflow-Metamodell* [vdA98, Obe96, EN93]. Es besteht in der Regel aus einer *Workflow-Sprache*, welche die Beschreibung der Geschäftsprozesse unter Beachtung verschiedener Aspekte, wie der Abfolge von Einzeltätigkeiten (sog. *Workflow-Aktivitäten*), der Zuordnung von Bearbeitern zu diesen Tätigkeiten oder der Verknüpfung von Prozeßschritten mit Anwendungen, erlaubt (siehe Abschnitt 2.4.1.1). Mit Hilfe des Metamodells und darauf basierenden *Modellierungswerkzeugen* lassen sich für jeden zu unterstützenden Prozeßtyp konkrete *Workflow-Modelle (Prozeßvorlagen)* erstellen (*Buildtime*). Aus diesen Vorlagen können zur Ausführungszeit elektronisch unterstützte (Geschäfts-)Prozesse erzeugt werden (vgl. Abbildung 2.1). Sie werden *Workflow-Instanzen* genannt und durch das WfMS während ihrer kompletten Lebensdauer kontrolliert und verwaltet (*Runtime*).

Die einzelnen Tätigkeiten eines WF-Modells bzw. einer WF-Instanz werden als *Aktivitäten* bezeichnet. Man unterscheidet Aktivitäten mit und ohne Benutzerinteraktion. Aktivitäten ohne Interaktion können vom WfMS automatisch gestartet werden, wenn die

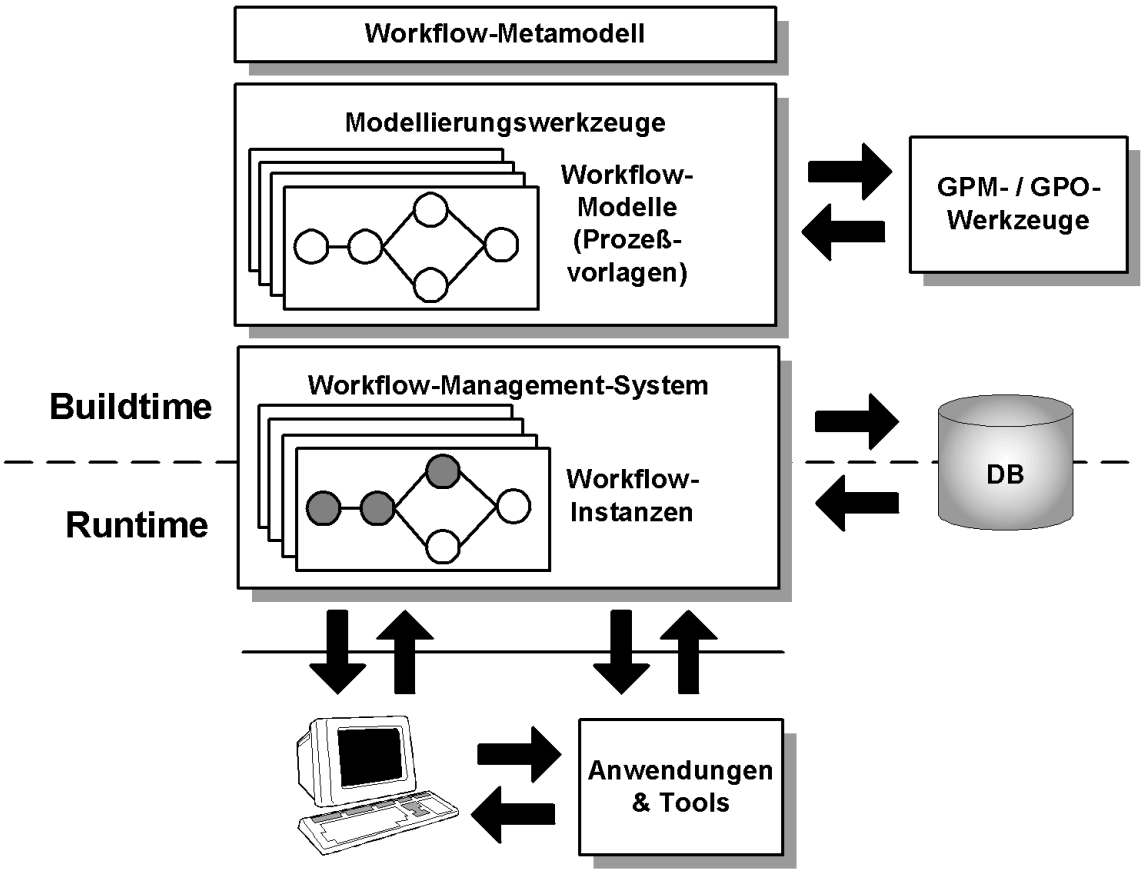

Abbildung 2.1: Komponenten eines WfMS (vgl. [LR00])

Bedingungen für ihre Ausführung erfüllt sind. Interaktive Aktivitäten werden dagegen nach bestimmten Kriterien in *Arbeitslisten (Worklists)* zusammengestellt. Der Anwender kann einzelne Arbeitslisteneinträge (*Workitems*) auswählen, woraufhin das festgelegte Anwendungsprogramm zur Bearbeitung der Aktivität (z.B. eine Bildschirmmaske) gestartet wird.

Weitere Begriffe werden im Rahmen der Anforderungsanalyse in Kapitel 2.4 geklärt. Außerdem wird auf [Wor99b], [DR00] und [JBS99] verwiesen.

2.2 Abgrenzung

Workflow-Management hat sich in den letzten Jahren zu einem populären Gebiet innerhalb der Softwarebranche entwickelt. Viele Anbieter werben mit Workflow-Funktionalität, erfüllen aber wesentliche Merkmale von WfMS nicht. Deshalb soll hier eine kurze Abgrenzung zu ähnlichen vorgangsorientierten Softwarelösungen gegeben werden.

2.2.1 Geschäftsprozeßmodellierungs-Werkzeuge

Geschäftsprozeßmodellierungs-Werkzeuge (GPM-Werkzeuge) unterstützen den Anwender bei der Erfassung, Analyse und Dokumentation betrieblicher Abläufe (nicht nur der rechnergestützten Ausschnitte) [Rei00]. Sie erlauben Analysen und Simulationen anhand festgelegter Wahrscheinlichkeiten, sind jedoch nicht in der Lage Workflows in der Realität auszuführen. Trotzdem bieten sie sich als Vorstufe zur WF-Modellierung an. Hierzu liefern sie Export-Schnittstellen für gängige WF-Beschreibungssprachen. Beispiele für GPM-Werkzeuge sind das *ARIS-Toolset* [Sch98a] und *Bonapart* [Int01].

2.2.2 Groupware-Systeme

Groupware-Systeme erlauben die gemeinsame Bearbeitung einer Aufgabe oder eines Vorgangs durch mehrere Anwender [Wul97, Gre88]. Die Speicherung von Dokumenten, Zeichnungen oder Bildern erfolgt meist in einer zentralen Datenbank, und der Informationsaustausch zwischen den Benutzern wird über Mail-Systeme realisiert. Es gibt zwar Versuche, Workflow-Funktionalität durch das „Umherreichen“ von Dokumenten nach bestimmten Regeln nachzuempfinden, allerdings fehlt hier eine zentrale Steuerung sowie die Trennung von Kontroll- und Datenfluß (vgl. Kapitel 2.4). Bekanntester Vertreter von Groupware-Systemen ist Lotus Notes [IBM01a].

2.2.3 Dokumenten-Management-Systeme

Zentrales Element von *Dokumenten-Management-Systemen (DMS)* ist der Lebenszyklus eines Dokuments, welcher mit der Erstellung beginnt und sich über die Bearbeitung, die Verteilung und die Speicherung fortsetzt [GSSZ99]. Daher lassen sich mit DMS auch keine komplexen Geschäftsprozesse modellieren, sondern lediglich einfache Dokumentenumläufe (z.B. Rundschreiben). Beispiele für DMS sind *ITA* von *SER Systeme* und *Lifelink* von *OpenText* [Ver00].

2.2.4 Projekt-Management-Systeme

Ziel von *Projekt-Management-Systemen (PMS)* ist die Unterstützung des Anwenders bei der Planung und Überwachung von Projekten. Sie erlauben die Modellierung von zeitlichen Abhängigkeiten zwischen Vorgängen, die Verwaltung von Ressourcen, die Vergabe von Prioritäten sowie die Berechnung von (Fertigstellungs-)Terminen und frühesten / spätesten Anfangs- und Endzeitpunkten für Vorgänge (d.h. einzelne Projekttätigkeiten) [Rei00]. Die Durchführung eines Projektes wird im Gegensatz zu einem WfMS nicht vom PMS gesteuert, sondern es dient lediglich dem Projektleiter als Planungs- und Überwachungsinstrument. Ein sehr populäres PMS ist zum Beispiel Microsoft Project [Stu99].

2.3 Kategorien

Für die Realisierung von WfMS gibt es im wesentlichen drei unterschiedliche Kategorien von Implementierungsansätzen. Sie werden in folgenden Abschnitten vorgestellt.

2.3.1 Formularorientierte WfMS

Formularorientierten WfMS liegt die Idee eines Formularsatzes mit unterschiedlichen Durchschlägen zu Grunde [Rei00]. Ein Formular enthält Felder, welche Werte aus bereits zuvor bearbeiteten Formularen enthalten können oder neue hinzufügen. Eine zentrale Ablaufsteuerung gibt es nicht, sondern lediglich einfache „Routing-Regeln“ (z.B. durch Skript beschrieben), welche im Formular enthalten sind und für die Übertragung (z.B. per E-Mail) zum jeweils nächsten Bearbeiter sorgen. Vertreter dieser Kategorie von Systemen sind z.B. *Lotus Notes* [IBM01a] und *Microsoft Exchange* ,[Mic01a].

2.3.2 Dokumentenorientierte WfMS

Dokumentenorientierte WfMS orientieren sich am Modell einer elektronischen Umlaufmappe, welche das Hinzufügen, Bearbeiten oder Entnehmen von Dokumenten erlaubt. Der Ablauf der Bearbeitungsschritte ist in der Mappe festgelegt, wodurch es bezüglich der Ablaufsteuerung dieselben Probleme wie bei formularorientierten Systemen gibt. Die parallele Ausführung von Prozeßschritten ist besonders problematisch, da auf verschiedenen Kopien der Umlaufmappe gearbeitet wird und eine Zusammenführung zweier Mappen (nach der parallelen Bearbeitung) hauptsächlich manuell erfolgen muß. Vertreter dieser Kategorie von Systemen sind z.B. *ProMInanD* [IAB96] und *SER Floware* [Ple96].

2.3.3 Prozeßorientierte WfMS

Prozeßorientierte WfMS (auch *Production Workflow Management System* [LR00] genannt) trennen Ablauflogik und Applikationscode konsequent voneinander [Rei00]. Sie besitzen eine (zentrale) Steuerungskomponente, welche den Kontroll- und Datenfluß zwischen den einzelnen Applikationen steuert und überwacht sowie Werkzeuge zur Administration bereitstellt. Diese Kategorie von WfMS, welche eine neutrale Realisierungsplattform für prozeßorientierte Anwendungen bietet, wird im Rahmen dieser Arbeit schwerpunktmäßig behandelt. Für die Unterstützung unternehmensweiter und -übergreifendender Workflows besitzt sie das mit Abstand größte Potential. Vertreter dieser Kategorie von Systemen sind z.B. *IBM MQ Series Workflow* und *Staffware* [Mar01].

2.4 Anforderungen

Der unternehmensweite Einsatz von WfMS und die Steuerung komplexer Geschäftsprozesse stellen hohe Anforderungen an die Software. Daher ist eine genaue Analyse der funktionalen und nichtfunktionalen Anforderungen, wie sie im Rahmen des Software-Engineering betrieben wird [Som92], auch für WfMS wichtig. Funktionale Anforderungen beschreiben die angebotenen Dienste und die Reaktion des Systems auf Eingaben, während nichtfunktionale Anforderungen Bedingungen an die Funktionsausführung stellen, zum Beispiel hinsichtlich Aspekten wie Interoperabilität mit anderen Systemen, Sicherheit oder Verfügbarkeit [JBS99].

2.4.1 Funktionale Anforderungen

Die funktionalen Anforderungen lassen sich im wesentlichen in die Anforderungen an die Ausdrucksmächtigkeit des Workflow-Metamodells sowohl an die Funktionalität des WfMS und seiner Workflow-Clients unterteilen.

2.4.1.1 Workflow-Metamodell

Das *Workflow-Metamodell* eines WfMS dient der Beschreibung der Geschäftsprozesse und bestimmt damit zu großen Teilen die Funktionalität des WfMS. Daher ist eine genaue Betrachtung des Modells unter verschiedenen Aspekten (vgl. Abbildung 2.2) wichtig [Jab95]:

- *Was muß ausgeführt werden? (Funktionsaspekt)*
  Der Funktionsaspekt definiert die logische Verarbeitungseinheiten eines Workflows. Die Einheiten können aus Aktivitäten, welche durch Workflow-Applikationen realisiert werden, oder aus Kompositionen anderer (Sub-)Workflows bestehen. Die Workflow-Sprache sollte Konstrukte zur Beschreibung dieser Einheiten anbieten.

- *Wann soll etwas ausgeführt werden? (Verhaltensaspekt)*
  Der Verhaltensaspekt eines Workflow-Modells beschreibt die Abfolge der Aktivitäten in einem Workflow. Die Workflow-Sprache bietet Konstrukte wie Sequenzen sowie parallele oder bedingte Verzweigungen zur Modellierung des Kontrolflusses zwischen den Verarbeitungseinheiten an. Das WfMS „führt“ den Benutzer während der Workflow-Ausführung auf dem durch den Kontrollfluß spezifizierten Pfad durch den Workflow.

- *Wie fließen die Daten? (Informationsaspekt)*
  Die Datenflüsse von Workflows werden durch den Informationsaspekt abgebildet. Er ermöglicht eine Strukturierung nach WF-Kontrolldaten (z.B. Status eines WF), Workflow-relevanten Daten (d.h. für die WF-Steuerung benötigte Daten, z.B. zur

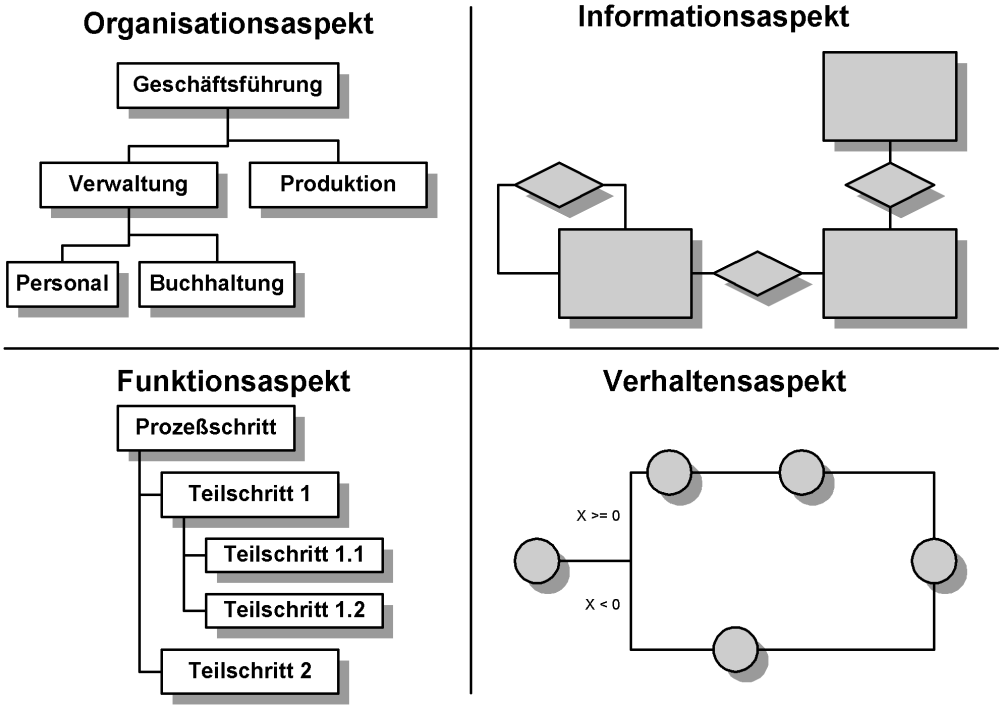

Abbildung 2.2: Aspekte eines Workflow-Meta-Modells [DR00]

Auswertung von Verzweigungsprädikaten) und Produktionsdaten (Anwendungsdaten). Benutzerdefinierte Datentypen und durch externe Anwendundungssysteme verwaltete Daten erschweren die Modellierung des Datenflusses.

- *Wer führt etwas aus? (Organisationsaspekt)*
  Der Organisationsaspekt spiegelt die Aufbauorganisation eines Unternehmens (bzw. Ausschnitte davon) wieder, indem er die Hierarchien, Organisationseinheiten und Stellen abbildet. Darüber hinaus erlaubt er die Definition von Gruppen und Rollen, um Ausführungsrechte für Prozesse und Prozeßschritte festlegen zu können. Außerdem können spezielle Kompetenzen oder Stellvertreterregelungen (z.B. bei Urlaub) spezifiziert werden.

- *Wie wird etwas ausgeführt? (Operativer Aspekt)*
  Der Operationsaspekt beschreibt die Programme (*Workflow-Applikationen*), welche zur Ausführung von Arbeitsschritten zur Verfügung stehen. Dabei kann es sich um Anwendungsprogramme, Skripte, Fomulare in Web-Clients oder die Integration bestehender Datenbank- oder ERP-Systeme (sog. *Legacy Systems*) handeln.

2.4.1.2 Funktionaliät der WfMS-Clients

Der hohe Interaktionsgrad von WfMS stellt viele Anforderungen an die Funktionalität der Workflow-Clients. Neben der Unterteilung zwischen Buildtime- und Runtime-Clients (vgl. Abbildung 2.1) können drei weitere Arten von Schnittstellen unterschieden werden: Entwickler-, Anwender- und Administratorschnittstellen. Ihre Aufgaben und Anforderungen werden im folgenden erläutert:

- *Entwicklerschnittstelle*
  Die Entwicklerschnittstelle gehört zu den Buildtime-Clients und bietet Modellierungskomponenten zur Umsetzung der Geschäftsprozesses in elektronisch repräsentierte Prozeßvorlagen. Beim Entwurf wird der Entwickler durch (grafische) Editoren und Werkzeuge zur Überprüfung der Korrektheit der Modelle unterstützt [Mar01]. Außerdem werden häufig Import- und Export-Schnittstellen zu anderen WfMS oder GPM-Tools angeboten. Entwicklerschnittstellen erfordern meist eine sehr umfangreiche graphische Benutzeroberfläche (*Graphical User Interface, GUI*) und Zugriffe auf Dateisysteme oder Datenbanken.

- *Anwenderschnittstelle*
  Die Anwenderschnittstelle (oft auch *Standard-Client* genannt) zählt zu den Runtime-Clients und ermöglicht die Arbeitslistenverwaltung (Darstellung von Workitems mit Attributen, Selektion, Aktualisierung, usw.), das Ausführen von Arbeitsschritten durch den Aufruf definierter Workflow-Applikationen, die Verwaltung von Aktivitäteninstanzen (Statusabfragen, Suspendieren, usw.) sowie die Kontrolle und Überwachung von Prozeßinstanzen (Instantiieren, Beenden, usw.). Die Arbeitslistenverwaltung stellt besonders hohe Anforderungen an den Client, weil er die Arbeitsliste für den Benutzer immer auf einem aktuellen Stand halten muß (*Worklist Update*) .

  Prinzipiell gibt es hierzu zwei unterschiedliche Verfahren. Beim *Polling* fragt der Client in regelmäßigen Abständen beim WfMS-Server nach Änderungen und holt sich ggf. eine neue Version der Arbeitsliste. Die andere Möglichkeit besteht darin, daß der Client vom WfMS über Veränderungen (aktiv) benachrichtigt wird. Welches Verfahren für Web-basierte WfMS besser geeignet ist, wird in Kapitel 5 untersucht.

  Neben der Arbeitslistenverwaltung stellt der Aufruf der Workflow-Applikationen die zweite Hauptaufgabe der Anwenderschnittstelle dar. Dabei müssen der Applikation beim Aufruf Informationen zu Eingabeparametern oder zur Verbindung mit dem WfMS übergeben werden. Bei Beendigung der Applikation müssen ggf. die Ausgabeparamter wieder entgegengenommen werden.

- *Administratorschnittstelle*
  Die Administratorschnittstelle gehört ebenfalls zu den Runtime-Clients. Sie unterstützt den Administrator bei der Konfiguration (Benutzer- und Organisations-

verwaltung, Verwaltung der Workflow-Applikationen), der Überwachung (Erstellen und Analyse von Statistiken, Process Mining) und der Steuerung des WfMS (z.B. Hoch-/Herunterfahren des Systems) [AL98]. Zur einfacheren Überwachung werden Prozesse oft graphisch visualisiert oder der Administrator erhält (aktive) Benachrichtigungen über mögliche Ausnahmesituationen.

2.4.1.3 Weitere funktionale Anforderungen

Die Bedarf an Funktionalität geht in der Praxis oft über die bisher genannten Punkte hinaus. Für den praktischen Einsatz von WfMS ebenfalls wichtige, von heutigen WfMS aber nicht zufriedenstellend gelöste Punkte sind [DR00, KBB98]:

- *Ad-Hoc-Änderungen*, welche das Einfügen oder Löschen von Aktivitäten in bereits laufenden Prozeßinstanzen erlauben. In vielen Anwendungsbereichen (z.B. Krankenhaus, Engineering Bereich) kann es immer wieder zu außerplanmäßigen Änderungen an Workflows kommen [RD98].
- *Schema-Evolutionen*, die eine Änderung der Prozeßvorlagen unterstützen und die es bei Bedarf auch erlauben, die Änderungen nach bestimmten Regeln auf bereits laufende Prozeßinstanzen zu propagieren [CCPP98, JH98].
- *Temporale Aspekte*, welche die Überwachung von Terminen bzw. Fristen (*Deadlines*) sowie von zeitlichen Minimal- und Maximalabständen zwischen Aktivitäten erlauben [AAH98, Gri97].

Bei allen genannten Punkten ist der Erhalt der Konsistenz des Workflows ein vorrangiger Gesichtspunkt. Um diese zu gewährleisten, sind umfassendes theoretisches Wissen und aufwendige Analysen nötig. Hersteller kommerzieller WfMS verzichten deshalb oft auf die Implementierung dieser fortschrittlichen Funktionen [Mar01].

2.4.2 Nichtfunktionale Anforderungen

Nachdem wichtige Funktionalitäten eines WfMS beschrieben worden sind, sollen in diesem Abschnitt nichtfunktionale Anforderungen ermittelt werden. Sie sind für den Einsatz von WfMS im Zusammenhang mit bestehenden EDV-Lösungen sowie für die Akzeptanz bei Anwendern von großer Bedeutung.

2.4.2.1 Offenheit

Da es sich bei WfMS um Softwaresysteme handelt, welche unternehmensweit auf verschiedenen Hardwareplattformen, mit unterschiedlichen Kommunikationsmechanismen

und einer Vielzahl von Workflow-Clients eingesetzt werden, hat der Begriff der Offenheit eine besondere Bedeutung. Offene Systeme zeichnen sich dabei durch Aspekte wie *Portierbarkeit*, *Interoperabilität*, *Erweiterbarkeit* und *Skalierbarkeit* aus [Ste96].

Die Hardware- und Softwareheterogenität in vielen Unternehmen(-sbereichen) erschwert die Implementierung von einfachen Softwarelösungen. Deshalb sind für den unternehmensweiten Einsatz von WfMS eine Unabhängigkeit von der Hardware- und Betriebssystemplattform oder zumindest eine leichte Portierbarkeit von großer Bedeutung. Das Verständnis von Portierbarkeit geht allerdings über die Forderung nach lediglich geringfügigen Änderungen am Quellcode hinaus. Datenformate sollen auch nach der Portierung Bestand haben, und die Anwender sollen sich ohne Schulungen auf dem neuen System zurecht finden.

Interoperabilität bezeichnet die Fähigkeit einer Systemkomponente, sich aufgrund genormter Schnittstellen in ein Gesamtsystem integrieren zu lassen und mit ihm Daten auszutauschen. Die Interoperabilität zwischen verschiedenen WfMS [HPS+00] ist im Zusammenhang mit E-Business von entscheidender Bedeutung [Gar99b, Gar99a]. Ein Beispiel ist die Koordination der Geschäftsprozesse eines Produzenten und seines Logistikpartners. Standardisierte Kommunikationsmechanismen erleichtern die Kommunikation zwischen den einzelnen WfMS-Komponenten. In diesem Zusammenhang hat sich der Begriff der *Middleware* etabliert, welcher eine anwendungsneutrale Systemsoftware bezeichnet, die zwischen Anwendungungsprogrammen und Betriebssystem, Datenbanksystem und anderen Ressourcen-Managern läuft [JBS99]. Sie stellt Infrastrukturdienste (z.B. Kommunikations- und Transaktionsdienste) zur Verfügung, auf deren Basis Applikationen ortstransparent entwickelt und betrieben werden können.

In engem Zusammenhang zur Interoperabilität steht die *Erweiterbarkeit*. Offen zugängliche Anwendungsprogrammierschnittstellen (*Application Programming Interface, API*) oder ein komponentenorientierter Entwurf [GT00] erleichtern die Erweiterung bzw. Anpassung des WfMS durch den Entwickler.

Die Fähigkeit, Anwendungen auf Rechnern unterschiedlicher Leistungsklassen einsetzen zu können, wird als *Skalierbarkeit* bezeichnet. Für den unternehmensweiten Einsatz von WfMS spielt sie eine entscheidende Rolle. Sollen hunderte oder gar tausende von Workflow-Instanzen und -Benutzern unterstützt werden können, ist aus Performanzsicht (z.B. wegen Antwortzeitgarantien) und Ressourcengründen eine skalierbare Workflow-Architektur unerläßlich [Bau01]. Aber auch auf Klientenseite sind skalierbare Lösungen für unterschiedliche Leistungsanforderungen wichtig. Deshalb sind in der Praxis, neben „normalen“ Workflow-Clients, auch leichtgewichtige Clients (zum Beispiel für *Personal Digital Assistants (PDA)* und Mobiltelefone) gefragt.

2.4.2.2 Zuverlässigkeit

Verteile Systeme mit Client-/Server-Architektur (vgl. Kapitel 3.3) bieten einerseits zahlreiche Vorteile gegenüber monolithischen Architekturen, andererseits entstehen aber

auch Probleme, welche im verteilten Fall schwieriger zu lösen sind. Zuverlässigkeitsgarantien und fehlertolertantes Verhalten zählen hierzu. Zur Unterstützung unternehmensweiter Prozesse müssen beim Entwurf eines WfMS folgende Aspekte berücksichtigt werden:

- Mechanismen zur Fehlererkennung, Fehlerbehandlung und zum Weiterführen des Systems nach einem Client-/Server-Absturz [Web98]
- Transaktions- und Recovery-Konzepte für lang andauernde und geschachtelte Transaktionen (siehe [Dad96])

2.4.2.3 Mehrbenutzerbetrieb und Sicherheit

Da an die Ausführung von Workflows mehrere Benutzer mit teilweise unterschiedlichen Kompetenzen beteiligt sein können, ist die *Authentifizierung* jedes Anwenders gegenüber dem System ein zwingender Punkt. Außerdem muß es Mechanismen zur *Authorisierung* beim Zugriff auf Daten, auf Funktionen zur Steuerung von Prozeßinstanzen und auf andere Funktionen des Systems (z.B. *Ad-hoc-Änderungen*) geben. Außerdem muß die sichere Kommunikation zwischen Workflow-Server und -Client gewährleistet sein. Hier bieten sich verschlüsselte Kommunikationswege an [Web98].

Durch den Einsatz von WfMS in vielen Bereichen eines Unternehmens fließen zwangsläufig auch sicherheitsrelevante Daten durch das System, welche nur von bestimmten Benutzern gelesen oder bearbeitet werden dürfen. Aspekte wie Interoperabilität mit WfMS anderer Unternehmen oder die Kommunikation über fremde Netze verschärfen die Sicherheitsanforderungen noch.

2.4.2.4 Software-Ergonomie

Ein wesentlicher Faktor für die Akzeptanz eines WfMS beim Benutzer ist die *Software-Ergonomie* [Bal01]. Die Bedienung des Systems und dessen Funktionalität muß an die Anforderungen des Benutzers angepaßt sein.

Die Gestaltung der Benutzerschnittstellen ist deshalb von besonderer Bedeutung. Idealerweise fügt sich das WfMS in bereits bestehende Anwendungssoftware ein oder besitzt ein gewohntes *Look-And-Feel* (z.B. Darstellung im Web-Browser). Eine integrierte Darstellung in einem (Browser-)Fenster ist dabei ebenfalls von Vorteil, um den Benutzer nicht mit vielen Fenstern zu verwirren (kein „Fensterterror“).

Verschiedene Benutzerschnittstellen (vgl. Abschnitt 2.4.1.2) für unterschiedlichen Anwendergruppen (z.B. „normale“ Anwender, Entwickler, Administratoren, usw.) sorgen für die Bereitstellung der jeweils geforderten Funktionalität. Es macht zum Beispiel wenig Sinn, den „normalen“ Workflow-Anwender mit systemnahen Details, wie der Auslastung eines WfMS-Servers oder der Verwaltung von Prozeß-Vorlagen, zu belasten.

2.4.2.5 Sitzungssemantik

Der Begriff Sitzungssemantik hat im Zusammenhang mit vielen Anwendungen eine wichtige Bedeutung. Er bezeichnet eine Einheit von Aktionen, welche in einem gemeinsamen Kontext ablaufen. Eine *Sitzung (session)* beginnt typischerweise mit der einmaligen Authentifizierung des Benuters gegenüber dem WfMS und erstreckt sich über die Bearbeitung mehrerer Aktivitäten bis zum Abmelden. In der Umgebung der Sitzung werden Einstellungen wie die Sortierreihenfolge der Arbeitsliste oder die Verbindung zum WfMS „gespeichert“.

2.4.3 Zusammenfassung der Anforderungen

Die wesentliche Anforderung an ein WfMS ist es, die funktionalen Komponenten des Systemkerns über geeignete Schnittstellen zur Verfügung zu stellen (vgl. Abbildung 2.3) und dabei die nichfunktionalen Anforderungen aus Abschnitt 2.4.2 zu beachten.

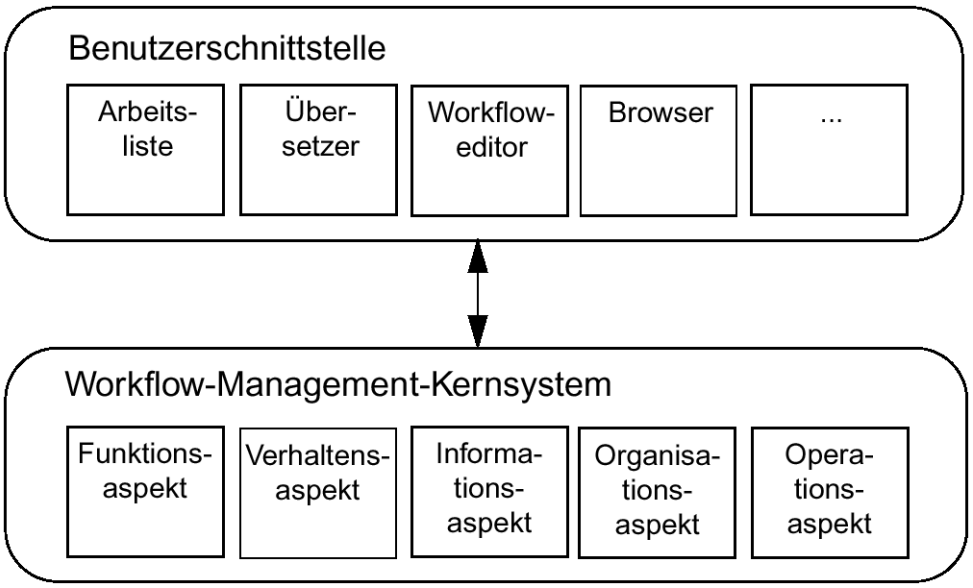

Abbildung 2.3: Funktionale Komponenten eines WfMS [JBS99]

Da die Funktionalität des WfMS durch das Workflow-Meta-Modell bereits vorgegeben ist, werden im Rahmen dieser Arbeit vor allem Möglichkeiten zur Realisierung von Workflow-Clients, unter Beachtung der Anforderungen aus Abschnitt 2.4.1.2 und 2.4.2, betrachtet.

2.5 Standards

Die wachsende Bedeutung von Workflow-Management-Systemen und das zunehmende Interesse auf Seiten der Softwarehersteller und Anwender machten eine Standardisierung in diesem Bereich erforderlich. Zur Zeit gibt es zwei wichtige Organisationen, welche sich um eine Standardisierung im Bereich Workflow-Management bemühen. Dies sind zum einen die *Workflow Management Coalition (WfMC)*, zum anderen eine Arbeitsgruppe der *Object Management Group (OMG)*. Die von ihnen entwickelten Referenzmodelle werden im folgenden kurz vorgestellt.

2.5.1 Workflow Management Coalition

Die WfMC [Fis00, Wor01a] ist eine von WfMS-Anbietern und -Anwendern im Jahr 1993 gegründete Organisation. Sie hat es sich zur Aufgabe gemacht, eine einheitliche Terminologie für Workflow-Management und Standards für die Interoperabilität von WfMS zu schaffen. Deshalb hat sie 1996 ein Referenzmodell für eine WfMS-Architektur spezifiziert [Wor95]. Ziel dieses Modells ist es nicht, den Herstellern Vorschriften für die (interne) Implementierung ihrer Systeme zu machen, sondern lediglich saubere Schnittstellendefinitionen zu liefern. Darum soll die Interoperabilität von WfMS gefördert und auf diese Weise für eine größere Verbreitung von WfMS gesorgt werden [JBS99].

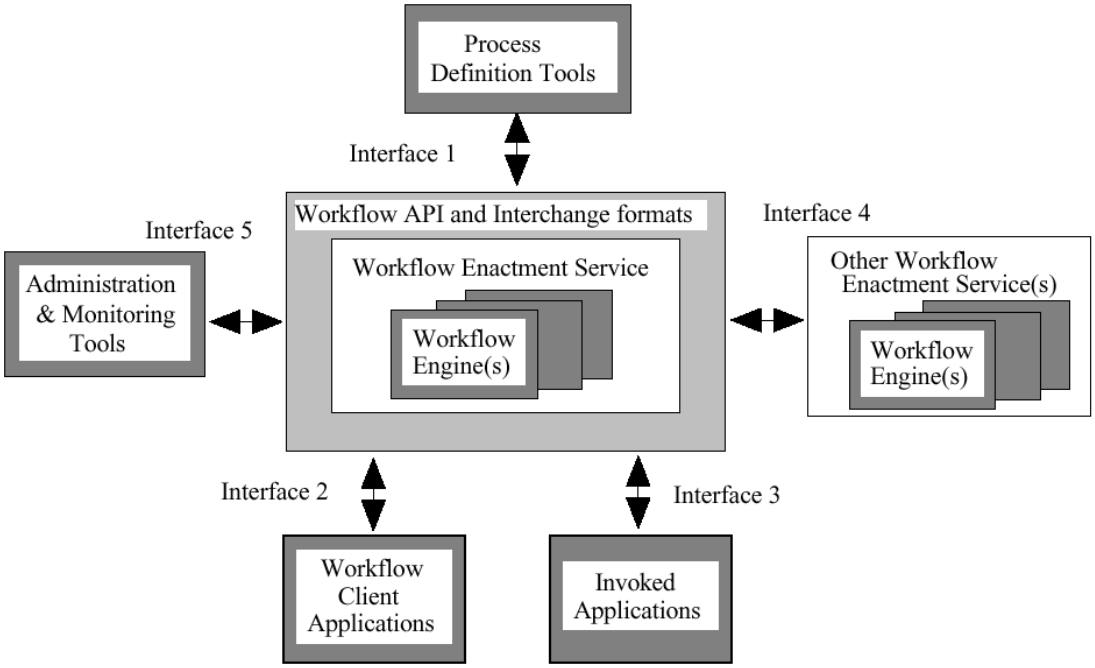

Abbildung 2.4: Schnittstellen des WfMC-Referenzmodells

Zentrales Element der WfMC-Referenzarchitektur (siehe Abbildung 2.4) ist eine Komponente zur Abwicklung und Koordination laufender Workflows (*Workflow Enactment Service*) [JBS99]. Die Realisierung durch eine oder mehrere Workflow-Engines sowie die interne Struktur der Komponente werden nicht vorgeschrieben. Für die Interaktion mit Benutzern und anderen Systemen wurde das *Workflow Application Interface (WAPI)* definiert [Wor98d]. Es besteht aus fünf verschiedenen Schnittstellen, deren Aufgaben im folgenden erläutert wird.

2.5.1.1 Interface 1

Über die Schnittstelle 1 können *Prozeß-Modellierungswerkzeuge (Process Definition Tools)* Workflow-Modelle mit dem *Workflow-Ausführungsdienst (Workflow Enactment Service)* austauschen. Dazu müssen die Modelle in einer standardisierten Workflow-Sprache (sog. *Workflow Process Definition Language (WPDL)*) beschrieben sein [Wor99a]. Die Schnittstelle 1 soll den Austausch von Workflow-Modellen zwischen unterschiedlichen WfMS-Produkten und GPM-Werkzeugen ermöglichen.

2.5.1.2 Interface 2

Die Schittstelle 2 dient der Interaktion zwischen Workflow-Clients und dem *Workflow Enactment Service* [Wor98e]. Insbesondere bietet sie Funktionen für die Abfrage und Verwaltung der Benutzerarbeitslisten. Workflow Clients können diese Arbeitslisten in geeigneter Form darstellen und die Auswahl von *Workitems* über diese Schnittstelle unterstützen.

2.5.1.3 Interface 3

Die Schnittstelle 3 dient dem Austausch von Workflow-relevanten Daten (Parameterdaten) zwischen WfMS und aufgerufenen Anwendungsprogrammen (*Invoked Applications*). Sie wurde ursprünglich als eigenständige Schnittstelle mit Referenzmodell geplant [Wor95], ist jedoch mittlerweile mit der Schnittstelle 2 zusammengefaßt worden [Wor98e].

2.5.1.4 Interface 4

Für die Interoperabilität mit anderen WfMS wurde die Schnittstelle 4 definiert. Sie ermöglicht die Ausführung von (Sub-)Workflows auf anderen WfMS. Für den Daten- und Kontrollaustausch wurde vor kurzem die XML-basierte Auszeichnungssprache *Wf-XML* (vgl. [Wor00a, Ben01]) definiert. Die Standardisierungsbemühungen der WfMC konzentrieren sich in der jüngsten Vergangenheit vor allem auf die Erweiterung dieser Schnittstelle.

2.5.1.5 Interface 5

Über die Schnittstelle 5 sollen Administrations- und Monitorwerkzeuge Informationen vom *Workflow Enactment Service* erfragen können und die Verwaltung von Workflows ermöglichen [Wor98c].

2.5.1.6 Erweiterungen

In den letzten Jahren gab es einige Erweiterungen der WfMC-Referenzarchitektur. Zunächst als Proposal veröffentlicht [Wor98a], gab es Bestrebungen, ein einheitliches *Komponentenmodell (Common Object Model)* für die WfMC-Referenzarchitektur zu definieren. In der Spezifikation 2.0 der Schnittstelle 2/3 [Wor98e] wurden Anbindungen für die Komponentenmodelle von Microsoft und der OMG definiert.

Außerdem gab es Überlegungen zu Event-Modellen [Wor99c], welche zum Beispiel bei der Notifikation über Änderungen in der Arbeitsliste benutzt werden können. Allerdings sind diese Erweiterungen bisher nur als Vorschläge veröffentlicht und bisher noch nicht Teil eines neuen Standards.

2.5.2 Object Management Group

Die *Object Management Group* ist das größte internationale Konsortium der Softwareindustrie [SBMW98]. Ihr Ziel ist die Standardisierung einer gemeinsamen Architektur für unterschiedliche Hardwareplattformen und Betriebssysteme. Diese wird als *Object Management Architecture (OMA)* bezeichnet. Seit 1995 arbeitet die OMG daran, ihre Referenzarchitektur um eine Komponente für Workflow-Unterstützung zu erweitern. Die OMA und die Workflow-Technologie richten sich an denselben Interessentenkreis, nämlich große, verteilte Organisationen mit heterogenen IT-Plattformen. Zuerst wollte man das WfMC-Referenzmodell in die OMA integrieren, allerdings erkannte man 1996, daß dies nicht gelingen wird, und gründete deshalb 1997 eine eigene Workflow-Arbeitsgruppe. Ihre Arbeit und die Entstehung der *Workflow Management Facility* werden im Anschluß an die Beschreibung des OMA-Modells vorgestellt.

2.5.2.1 Object Management Architecture

Die OMA bildet eine Referenzarchitektur für die Entwicklung verteilter objektorientierter Anwendungen [Zah00]. Die zentrale Kommunikationsplattform bildet der *Object Request Broker (ORB)* (vgl. Abbildung 2.5). Er verbindet die unterschiedlichen OMA-Komponenten und sorgt für die Unabhängigkeit von Betriebssystemen und Hardwareplattformen. Dieser zentrale Teil der OMA, welcher aus dem ORB und dessen Schnittstellen besteht, wird als *Common Object Request Broker Architecture (CORBA)* bezeichnet [Obj01, JBS99].

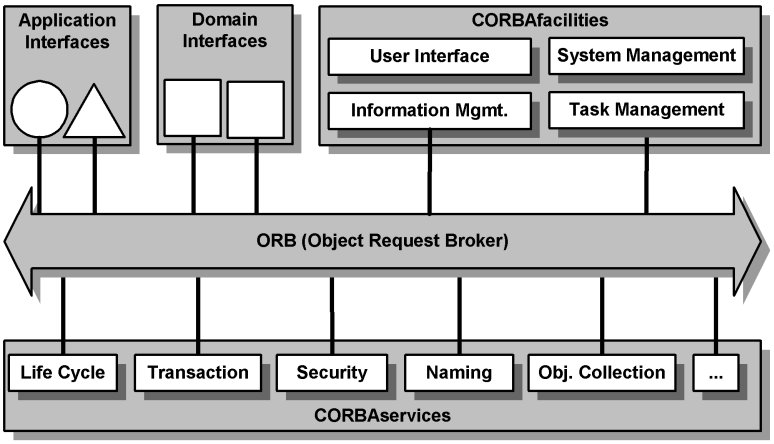

Abbildung 2.5: OMA-Architektur

Basis der OMA ist das *Core Object Model* der OMG. Es beinhaltet grundlegende Konzepte der Objektorientierung, wie Vererbung, Kapselung und Polymorphie. Die Objekttypen sowie ihre Operationen (Methoden) werden als *Interface* bezeichnet und durch die *Interface Definition Language (IDL)* beschreiben. Die IDL ermöglicht die vollständige Trennung von Schnittstellenbeschreibung und Implementierung, so daß die Objekte in fast jeder Programmiersprache (z.B. C, C++, Java, Smalltalk, usw.) implementiert werden können.

In den meisten objektorientierten Programmiersprachen existieren Objekte lediglich auf Quellcode-Ebene, um Prinzipien wie Kapselung oder Codewiederverwendung zu unterstützen [Kee01]. Nach der Übersetzung durch den Compiler sind diese Objekte allerdings von außen nicht mehr sichtbar. Im Gegensatz dazu existieren die Objekte des *OMG Core Object Model* nicht nur auf Programmiersprachenebene, sondern sie sind auch zur Laufzeit des Programms von außen sichtbar. Ein (Client-)Objekt kann sich eine Referenz auf ein anderes (Server-)Objekt holen, und über den ORB eine Operation dieses Objekts aufrufen [JBS99]. Dieser Operationsaufruf hat die Charakteristik eines entfernten synchronen Dienstaufrufs, vergleichbar mit einem *Remote Procedure Call (RPC)* [Blo92, Web98].

Neben dem ORB existieren die *CORBAservices*, welche grundlegende Dienste für die Entwicklung verteilter Applikationen bereitstellen (vgl. Abbildung 2.5).

Die *CORBAfacilities* sind auf einer höheren Abstraktionsebene als die *CORBAservices* angesiedelt. Sie unterstützen die Anwendungsentwicklung durch die Bereitstellung häufig verwendeter Funktionen. So gibt es zum Beispiel *Facilities* zur Unterstützung bei der Erstellung von Benutzerschnittstellen (*User Interface*), zur Verwaltung von Daten (*Information Management*), zur Administration verteilter Objekte (*System Management*) und zur Bearbeitung von Aufgaben (*Task Management*). Die *Workflow Facility* selbst ist Teil der *Task Management Facility*.

Die *Domain Interfaces* stellen spezialisierte Dienste zu bestimmten Anwedungsgebieten bereit. Verschiedene Arbeitsgruppen entwickeln genormte Dienste in den Bereichen *Business Object*, *Electronic Commerce*, *CORBAfinance*, *CORBAmanufacturing*, *CORBAmed* und *CORBAtel*. Neben den genormten *Domain Interfaces* existieren noch anwendungsspezifische, nicht standardisierte *Application Interfaces*.

2.5.2.2 OMG Workflow Management Facility

Im Gegensatz zur WfMC stellt Workflow-Management bei der OMA nur einen Teilbereich der Standardisierungsbemühungen dar. Deshalb war die Integration als eine *Common Facility* in die OMA-Gesamtarchitektur ein wichtiger Entwicklungsaspekt [Obj00, SBMW98, Sch98b, Sch98c]. Im Rahmen einer Ausschreibung der Arbeitsgruppe wurden drei Vorschläge zur Spezifikation eingereicht. Letztendlich wurde mit „jFlow“ der Vorschlag eines Firmenkonsortiums, dem vor allem WfMC-Mitglieder angehörten, ausgewählt und verabschiedet. Er beinhaltet folgende Punkte [JBS99, SBMW98, Sch98b]:

- Verwaltung von Workflow-Schemata
- Unterstützung verteilter Workflow-Objekte
- Verwaltungsfunktionen für Workflow-Objekte
- Konformität zum OMG-Objektmodell
- Verwendung vorhandener OMA-Komponenten
- Integration von Geschäftsobjekten (*Business Objects*)
- Keine Vorwegnahme der Implementierung

2.6 Zusammenfassung und Ausblick

WfMS unterstützen den Anwender bei der elektronischen Umsetzung seiner Geschäftsprozesse. Dies stellt eine große Herausforderung an WfMS dar, woraus hohe Anforderungen an die Systeme selbst resultieren (vgl. Abschnitt 2.4).

Die Standardisierungsbemühungen der WfMC und der OMG hinken der Realität allerdings hinterher. Die OMG betrachtet nur einseitig „ihre“ OMA-Architektur und die WfMC arbeitet in den letzten Jahren fast ausschließlich in Richtung Interoperabilität zwischen WfMC (Schnittstelle 4). Andere wichtige Bereiche, wie etwa die Integration von Workflow-Management in Web-basierte Anwendungen oder die Erweiterung der WfMS-Funktionalität (vgl. Abschnitt 2.4.1.3), bleiben dagegen weitgehend unberücksichtigt.

Kapitel 3

Grundlagen von Web-Applikationen

In diesem Kapitel soll ein kompakter Überblick zu Web-basierten Applikationen gegeben werden. Das Internet bildet die bedeutendste Kommunikationsplattform für den Datenaustausch im Web. Deshalb werden in Abschnitt 3.1 die wichtigsten Protokolle kurz vorgestellt. Anschließend werden in Abschnitt 3.2 aktuelle Dokumentformate für Web-Inhalte und in Abschnitt 3.3 Architekturmodelle für Web-basierte Applikationen besprochen. An zwei praktischen Beispielen aus dem E-Commerce- und dem Workflow-Bereich werden die Charakterista von Web-Applikationen an verdeutlicht. Abschließend erfolgt in Abschnitt 3.5 eine Bewertung unter Betrachtung der speziellen Anforderungen von Web-basierten WfMS.

3.1 Protokolle

Kommunikationssysteme sind in der Regel nicht monolithisch aufgebaut, sondern bestehen aus einem Schichtenmodell mit unterschiedlichen Protokollen [Web98]. Das wichtigste Netzwerkarchitekturmodell für offene Kommunikationssysteme ist das 1977 von der *International Standards Organization (ISO)* veröffentlichte *ISO-OSI-Referenzmodell* [Int77]. Es unterscheidet sieben Protokollebenen. Die unteren Ebenen sind für die Netzwerkkommunikation und die oberen für Anwendungen konzipiert. Dazwischen befinden sich Ebenen, welche eine Kommunikation mit Sitzungsemantik oder die Darstellung von Inhalten übernehmen. Eine detaillierte Beschreibung des *ISO-OSI-Referenzmodells* findet sich zum Beispiel in [Tan98].

Obwohl das *ISO-OSI-Referenzmodell* erst viel später als das Internet begründet wurde, erlaubt es doch ein besseres Verständnis der Aufgaben der einzelnen Internetprotokolle (vgl. Abbildung 3.1). Gemäß der primär verwendeten Protokolle wird die Internet-Architektur *TCP/IP-Referenzmodell* genannt [Tan98]. Es unterscheidet im Gegensatz zum *ISO-OSI-Referenzmodell* nur drei Schichten, deren Funktion in den folgenden Abschnitten erläutert wird.

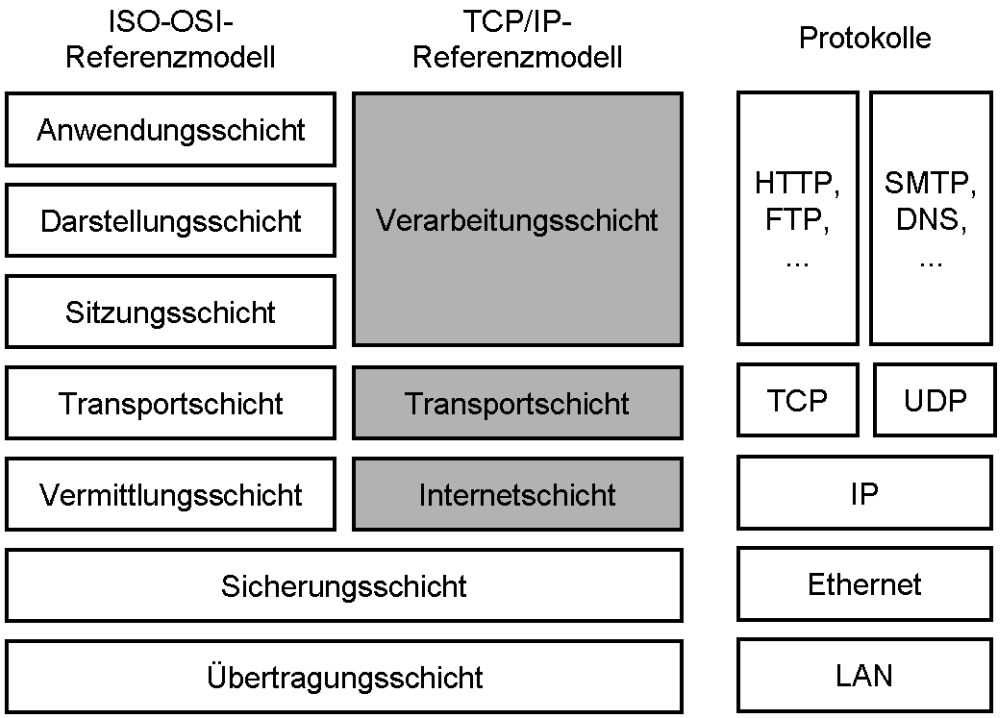

Abbildung 3.1: ISO-OSI- und TCP/IP-Referenz-Modell im Vergleich

3.1.1 Internetschicht

Über der physikalischen Schicht und der Sicherungsschicht des *ISO/OSI-Referenzmodells* befindet sich im *TCP/IP-Modell* die Internetschicht [Web98]. Sie hat die Funktion einer verbindungslosen Vermittlungsschicht und benutzt hierzu das *Internet Protocol (IP)*. IP ermöglicht es, Pakete von einem zu jedem beliebigen anderen Rechner im Netz zu versenden. Die Adressierung erfolgt über weltweit eindeutige 32-Bit lange Adressen, welche der Lesbarkeit halber als vier durch Punkte getrennte Zahlen dargestellt werden (z.B. 134.60.0.1) [Web98]. Die Paketübertragung und deren Wegwahl (*Routing*) ist die Hauptaufgabe der Internetschicht. Bezüglich der Reihenfolge oder einem Verlust von Paketen gibt IP jedoch keine Garantien. Dies sicherzustellen, ist die Aufgabe eines Protokolls in einer höheren Schicht.

3.1.2 Transportschicht

Die Transportschicht ist über der Internetschicht angesiedelt. Sie ermöglicht die Kommunikation zweier Rechner auf Prozeßebene über spezielle Kommunikationsendpunkte (*Sockets*) [Web98]. Im *TCP/IP-Modell* existieren zwei Protokolle für die Transportschicht.

Das *Transmission Control Protocoll (TCP)* erlaubt eine zuverlässige verbindungsorientierte Kommunikation, indem es einen Bytestrom in Nachrichten zerlegt und an die Internetschicht weiterleitet. Am Zielrechner werden die Pakete wieder zu einem Ausgabestrom zusammengesetzt [Tan98]. TCP übernimmt die Flußsteuerung, die Sortierung der Pakete und das Nachfordern verloren gegangener Nachrichten.

Neben TCP gibt es ein zweites Protokoll auf Ebene der Transportschicht. *UDP (User Datagram Protocol)* ist ein unzuverlässiges verbindungsloses Protokoll, welches hauptsächlich für einmalige Abfragen und Anwendungen mit einer eigenen Paketsteuerung eingesetzt wird. Es ist im Vergleich zu TCP deutlich schneller, weil kein Verbindungsaufbau initiiert werden muß.

3.1.3 Verarbeitungsschicht

Im Gegensatz zum *ISO-OSI-Referenzmodell* besitzt das *TCP/IP-Modell* keine explizite Sitzungs- und Darstellungsschicht, so daß die Verarbeitungsschicht direkt über der Transportschicht angeordnet ist. Wird von Anwendungen eine Sitzungssemantik gewünscht, müssen sie diese selbst implementieren. Auf welche Weise dies bei Webbasierten Anwendungen geschieht, wird in Abschnitt 3.4.1 an einem Beispiel gezeigt.

Die Protokolle der Vermittlungsschicht basieren in der Regel auf TCP bzw. UDP. Sie ermöglichen Anwendungen für vielfätige Einsatzzwecke. Es existieren u.a. spezielle Protokolle

- zur Übertragung von E-Mails (SMTP)
- zum Transfer von Dateien (FTP)
- für das Arbeiten an einem virtuellen Terminal (TELNET)
- zur Ermittlung von Rechnernamen/-adressen (DNS)und
- zur Kommunikation im WWW (HTTP).

Für Web-basierte Applikationen ist das *Hypertext Transfer Protocol (HTTP)* von besonderer Bedeutung. Es unterstützt verteilte, hypermediale Informationssysteme wie das WWW [Wor01l]. Es ist generisch, zustandslos und kann für viele Einsatzzwecke auf Anwendungsebene eingesetzt werden. Welche Besonderheiten und Probleme im Zusammenhang mit Web-Applikationen bestehen, wird in Abschnitt 3.4 gezeigt.

Die Kommunikation über HTTP läuft streng nach dem *Request-Response-Mechanismus* ab (vgl. Abbildung 3.2). Der Client (z.B. ein Web-Browser) schickt eine *Anfrage (Request)* an den Server (1). Zur Adressierung der Web-Inhalte werden sog. *Universal Resource Locator (URL)* eingesetzt [Wor01g]. Sie bestehen aus dem Namen des Web-Servers und dem Pfad der angeforderten Seite in seiner Verzeichnisstruktur.

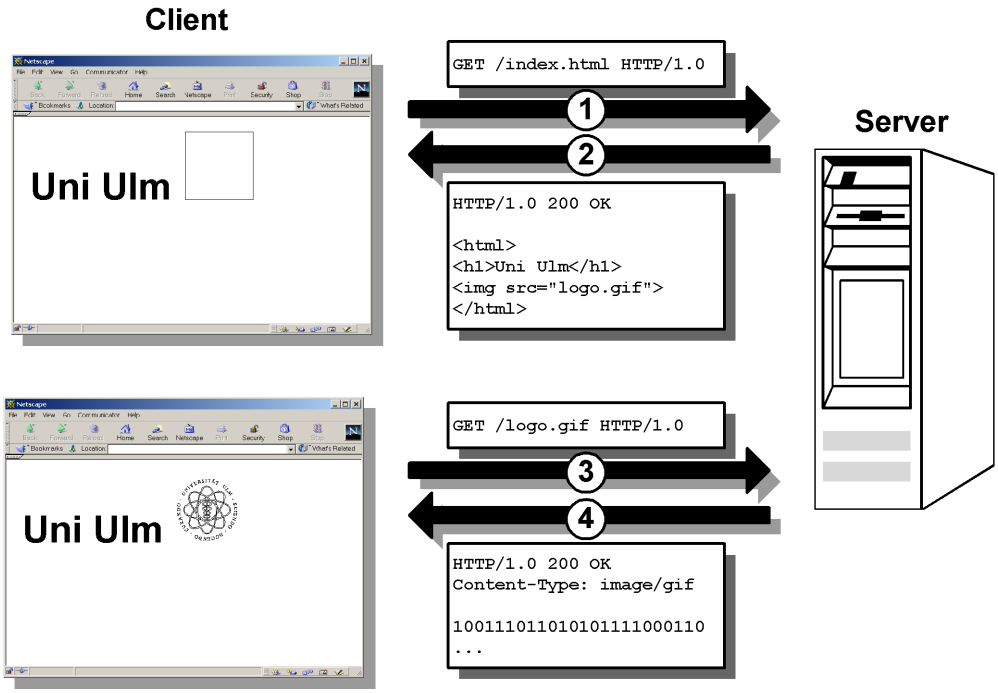

Abbildung 3.2: Beispiel-Interaktion zwischen Web-Client und Web-Server

Die Anfrage beinhaltet außer der URL zum Beispiel Authentifizierungsdaten oder Informationen über den Client (Hersteller, unterstützte Datenformate, usw.). Der Server schickt dem Client in einer *Anwort (Response)* die angeforderten Daten (2). Dies ist im Beispiel aus Abbildung 3.2 eine HTML-Seite (vgl. Abschnitt 3.2.1). Sind in der Seite weitere Objekte (z.B. Bilder, Formatvorlagen, ...) enthalten, muß der Client diese einzeln vom Server nachfordern, bis er von ihm alle nötigen Informationen zur Darstellung der Seite erhalten hat (vgl. (3) und (4)).

3.2 Inhalte

Wie bereits in Abschnitt 1.2 erwähnt, wurde das WWW für die Publikation von Forschungsdaten konzipiert. Deshalb ist der Begriff des Dokuments von zentraler Bedeutung. Dokumente lassen sich in drei wesentliche Komponenten aufgliedern (vgl. Abbildung 3.3) [BM98]:

- Struktur (Gliederung, Kapitel, ...)
- Inhalt (Texte, Bilder, Videos, ...)
- Darstellung (Schriftart, Schriftgröße, Farbe, tabellarisch, ...)

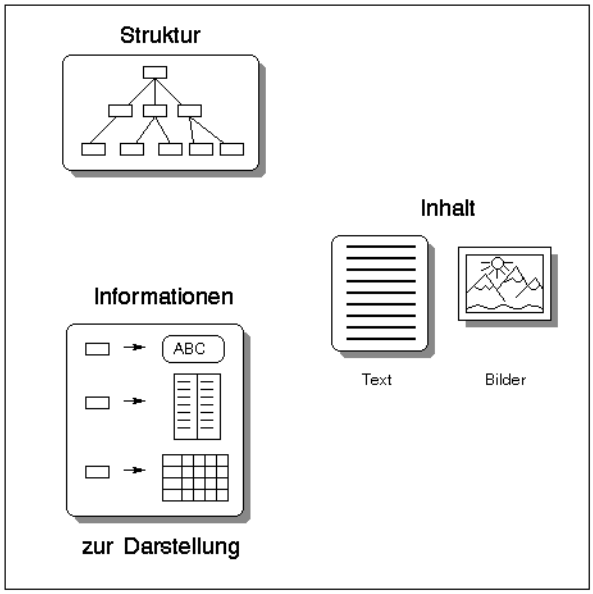

Abbildung 3.3: Komponenten eines Dokumentes [BM98]

Für die Beschreibung von Dokumenten bzw. Daten im allgemeinen wurden *Metasprachen* entwickelt [GS00]. Metasprachen erlauben durch die Festlegung einer bestimmten Syntax und einer Grammatik (z.B. *Document Type Definition (DTD)*) auch die Definition abgeleiteter Sprachen. Diese können als Grundlage für die Beschreibung von Dokumenten bzw. Daten(formaten) dienen. Eine der bekanntesten Metasprachen ist die 1986 von der ISO definierte *Standard Generalized Markup Language (SGML)* [Gol90]. Da SGML allerdings sehr komplex ist, erlangte es selbst für Web-Inhalte keine große Bedeutung. SGML diente jedoch als Grundlage, für die beiden von SGML abgeleiteten, und im Web weit verbreiteten Sprachen *HyperText Markup Language (HTML)* [Wor01e] und *Extensible Markup Language (XML)* [Wor01d, BM98].

3.2.1 HyperText Markup Language

HTML war die erste Beschreibungssprache für Dokumente im Web. Sie wurde 1989 am europäischen Zentrum für Kernforschung (CERN) konzipiert und anschließend vom *World Wide Web Consortium (W3C)* gepflegt und weiterentwickelt [Wor01e]. HTML ist in SGML formuliert. Die Sprache ermöglicht die Formatierung von Texten, die Integration unterschiedicher Inhalte (Text, Bilder, ...) und die Verknüpfung mit anderen Dokumenten durch *Hyperlinks*. Die Dokumente bestehen dabei aus geschachtelten Elementen, welche durch sog. *Tags* geöffnet (z.B. `<body>`) und geschlossen werden (z.B. `</body>`). Viele Elemente können um Parameter erweitert werden. Zum Beispiel setzt die Anweisung `<body bgcolor="#000000">` eine schwarze Hintergrundfarbe für die Seite.

Es gibt Elemente, welche immer paarweise auftreten und geschlossen werden müssen (z.B. <h1>...</h1>), und andere die auch für sich alleine stehen können (z.B. <p> oder <hr>). Die Struktur eines HTML-Dokuments soll an folgendem Beispiel verdeutlicht werden:

HTML-Quellcode	**Kommentar**
`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">`	Version, Sprache
`<html>`	Dokumentbeginn
`  <head>`	Beginn Kopfteil
`    <title>Testseite</title>`	Dokumenttitel
`    ...`	
`  </head>`	Ende Kopfteil
`  <body>`	Beginn Hauptteil
`    <h1>Einleitung</h1>`	Überschrift
`    <font face="Arial", size="5">`	Formatierung
`      Die Bedeutung von HTML ...`	Text
`    </font>`	
`    ...`	
`  </body>`	Ende Hauptteil
`</html>`	Dokumentende

Jedes HTML-Dokument wird von den <html>...</html> Tags eingeschlossen. Zur Untergliederung und zur Formatierung des Inhalts existieren weitere Elemente, die in den folgenden Abschnitten kurz vorgestellt werden.

3.2.1.1 Formatierung

HTML erlaubt es, Texte durch Verwendung folgender Elemente zu gestalten (Beispielausschnitt aus dem zur Verfügung stehenden Sprachumfang von HTML):

- Gliederung
  - Kopfteil (Header)
    (z.B. <head> <title>Beispielseite</title> </head>)
  - Textkörper
    (z.B. <body>Hier steht der Text im Dokument...</body>)
  - Überschriften
    (z.B. <h1> Kapitel 1 </h1> <h2> Abschnitt 1.1 </h2>)
  - Absätze
    (z.B. Absatz 1 <p> Absatz 2 <p> Absatz 3)

  - Listen
    (z.B. `<ul> <li>Eintrag1</li><li>Eintrag2</li> </ul>`)
  - Tabellen
    (z.B. `<table> <tr> <td>Feld1</td> </tr> </table>`)
  - Kommentare
    (z.B. `<!-- Dies ist ein Kommentar -->`)

- Textgestaltung
  - Schriftart und -größe
    (z.B. `<font face="Arial", size="5"> ... </font>`)
  - Schrifteffekte
    (z.B. `<b>fett</b> <i>kursiv</i> <u>unterstrichen</u>`)

- Verweise (Hyperlinks)
  (z.B. `<a href="next.html">Weiter</a>`)

Ausführlichere Informationen zu HTML und weiteren Formatierungselementen finden sich in [Wor01e, MN98].

3.2.1.2 Formatvorlagen

Formatvorlagen haben die Aufgabe, den Anwender bei der Textgestaltung zu unterstützen. Dazu bieten sie die Möglichkeit, einheitliche Formatierungen für Überschriften, hervorgehobenen Text, Abbildungsbeschriftungen, usw. festzulegen. Dies ermöglicht die Trennung von Inhalt und Format. Für HTML wurde eine spezielle Sprache zur Definition solcher Formatvorlagen (*Cascading Style-Sheets (CSS)*) entwickelt [Wor01b]. Sie erlaubt es, existierende HTML-Elemente an eigene Erfordernisse anzupassen [MN98]. Die Formatdefinition für eine große Überschrift könnte zum Beispiel folgendermaßen aussehen:

```
h1 { font-size: 48pt;
     font-family: Helvetica,Arial;
     color: #FF0000;
     letter-spacing: 5mm;
     word-spacing: 10mm;
}
```

Die Definition kann entweder im `<head>`-Abschnitt einer HTML-Datei oder in einer separaten CSS-Datei erfolgen. Die Beschreibung in einer separaten Datei ermöglicht die Wiederverwendung in anderen Seiten, so daß die kompletten Inhalte eines Web-Servers ein einheitliches Format (z.B. *Corporate Identity*) besitzen können. Außerdem wird die Modifizierbarkeit der Darstellung durch die Verwendung von CSS deutlich erleichtert.

3.2.1.3 Formulare

Neben den Befehlen zur Textgestaltung stellt HTML auch spezielle Befehle zur Erstellung von Formularen bereit (`<form> ... </form>`). Formulare können aus Textfeldern, mehrzeiligen Eingabefeldern, Listen, Kombo-, Radio- und Checkboxen sowie Buttons bestehen. Sie werden wie eine normale HTML-Seite vom Client geladen (*Request*) und im Browser dargestellt (vgl. (1) und (2) in Abbildung 3.4).

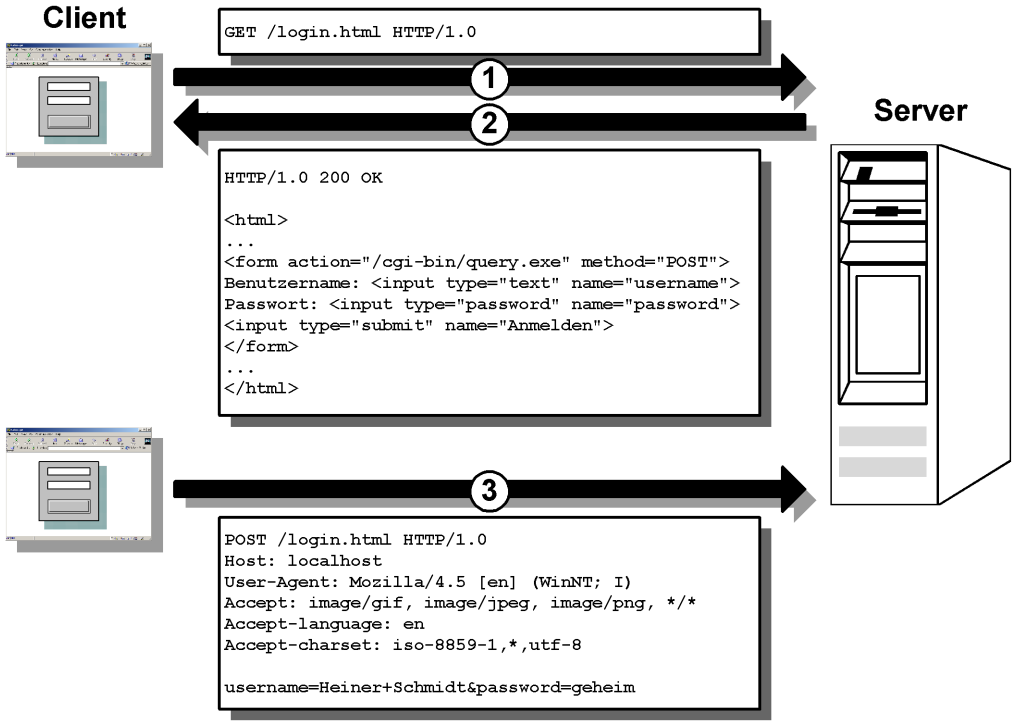

Abbildung 3.4: Beispiel für die Übertragung eines HTML-Formulars

Dann kann der Anwender den Inhalt bearbeiten, indem er Daten in Eingabefelder einträgt oder Elemente aus Listen auswählt. Nachdem der Anwender seine Bearbeitung abgeschlosssen hat, drückt er den `Submit`-Knopf (`<input type="submit">`) und veranlaßt damit die Übertragung der Formulardaten an den Server oder deren Versand per E-Mail. Für die Übertragung zum Server gibt es zwei Methoden:

- Bei der Übertragung durch `Post` (`<form action="post">`) werden werden die Daten des augefüllten Formulars codiert (vgl. (3) `username=Heiner+Schmidt&password=geheim`) und in einer HTTP-Nachricht zum Server gesendet.
- Die `Get`-Variante (`<form action="get">`) fügt die Parameter an die URL der angeforderten Seite an und sendet die Informationen auf diese Weise zum Server: `"...login.html?username=Heiner+Schmidt?password=geheim"`

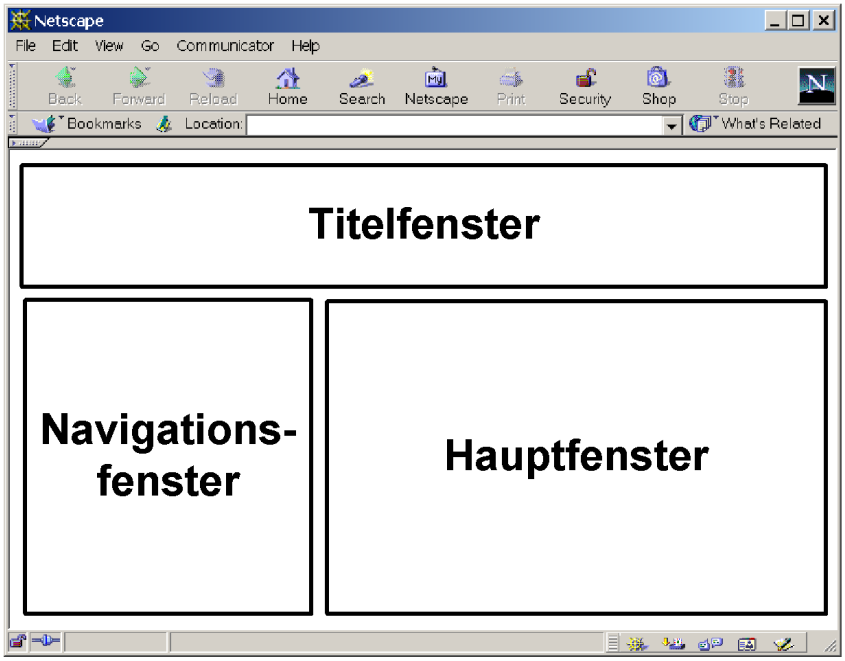

Abbildung 3.5: Beispiel eines durch Frames untergliederten Browserfensters

3.2.1.4 Frames

Eine Untergliederung des Browserfensters in mehrere Teilfenster ist durch sog. *Frames* möglich. In Abbildung 3.5 ist ein Beispiel für eine Unterteilung dargestellt. Die zugehörige Definition in HTML hat folgende Form:

```
<html>
<head></head>
<frameset rows="30%,70%">
  <frame src="titel.html" name="Titelfenster">
  <frameset cols="40%,60%">
    <frame src="navigation.html" name="Navigationsfenster">
    <frame src="main.html" name="Hauptfenster">
  </frameset>
</frameset>
<body> Ihr Browser kann keine Frames darstellen! </body>
</html>
```

Bei Hyperlinks kann festgelegt werden, in welchem Fenster der Inhalt des Links angezeigt werden soll. Die Links im Navigationsfenster können zum Beispiel einen Parameter `<a target="Hauptfenster">` enthalten, so daß alle über das Navigationsfenster ausgewählten Seiten im Hauptfenster dargestellt werden.

3.2.1.5 Dynamic HTML

Unter dem „Marketing-Begriff“ *Dynamic HTML (DHTML)* werden eine Reihe von Technologien zusammengefaßt, welche das bisher statische HTML um dynamische Elemente im Browser erweitern [HL01, Goo98]. Welche Elemente dies sind und welche Möglichkeiten sie eröffnen, wird in Abschnitt detaillierter 5.2.1 vorgestellt.

3.2.2 Extensible Markup Language

Die *Extensible Markup Language (XML)* ist eine vom W3C definierte Metasprache [Wor01d, BM98]. Sie ist (wie HTML) von SGML [Gol90] abgeleitet und kann als eine vereinfachte Variante von SGML angesehen werden. XML hat in den letzten Jahren stark an Bedeutung gewonnen, weil es durch die strikte Trennung von Inhalt und Format ein geeignetes Datenformat zum Austausch und zur Speicherung von Informationen darstellt. Eine XML-Datei zur Speicherung von Adreßdaten könnte zum Beispiel folgendes Format haben:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="adressen_table.xsl"?>
<adressbuch>
  <eintrag>
    <nachname>Maier</nachname>
    <vorname>Heinz</vorname>
    <strasse>Gartenweg 7</strasse>
    <plz>89081</plz>
    <ort>Ulm</ort>
    <telefon>0731-9123456</telefon>
  </eintrag>
  <eintrag>
    <nachname>Schmidt</nachname>
    <vorname>Karl</vorname>
    <telefon>07304-432421</telefon>
    <email>karl.schmidt@informatik.uni-ulm.de</email>
    <email>karl@schmidt.de</email>
  </eintrag>
  <eintrag>
    <nachname>Zimmermann</nachname>
    <vorname>Stefan</vorname>
    <telefon>0171-234890234</telefon>
  </eintrag>
</adressbuch>
```

Ein wesentliches Kriterium beim Austauch von Daten ist die Einhaltung eines definierten Datenformats. Für XML wurde eine spezielle Sprache zur Beschreibung solcher Formate entwickelt.

3.2.2.1 Document Type Definition

Durch eine *Document Type Definition (DTD)* werden der strukturelle Aufbau und die logischen Elemente einer Klasse von Dokumenten (Dokumenttyp) beschrieben [BM98]. Im Gegensatz zur HTML-Sprache, bei der es nur eine einzige DTD für alle Dokumente gibt, kann bei XML für jedes Dokument eine eigene DTD angegeben werden. Für das XML-Adreßbuch ergibt sich folgende DTD:

```
<?xml version="1.0"?>
<!ELEMENT adressbuch (eintrag+)>
<!ELEMENT eintrag (nachname,vorname,strasse?, plz?, ort?,
                   telefon+, email*)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT vorname  (#PCDATA)>
<!ELEMENT strasse  (#PCDATA)>
...
```

Das Adreßbuch besteht aus mehreren Einträgen, wobei jeder Eintrag eine Reihe von weiteren Elementen enthält. Manche davon müssen zwingend vorkommen (Nachname, Vorname, Telefon), andere sind optional (Straße, PLZ, Ort, E-Mail). Wie häufig die jeweiligen Elemente vorkommen dürfen bzw. müssen wird ebenfalls in der DTD festgelegt. Die Syntax (z.B. ? oder *) orientiert sich hier an regulären Ausdrücken (siehe [SSP99]). Die Elemente (Nachname, Vorname, usw.) bestehen aus beliebigem Text (*Parsed Character Data, PCDATA*) und dürfen selbst keine weiteren (Unter-)Elemente enthalten.

Viele XML-Editoren (z.B. *XML-Spy*) oder XML-Parser sind in der Lage, XML-Dokumente auf die Konformität zu einer bestimmten DTD zu prüfen. Allerdings lassen sich mittels DTD die Typen von Dokumenten nur „oberflächlich“ spezifizieren [Bal01]. Die genaue Angabe eines präzisen Datenformats (z.B. anstatt beliebiger Zeichenketten bei einer PLZ nur Ziffern) ist nicht möglich. Außerdem ist eine DTD selbst kein XML-Dokument, d.h. sie kann mit XML-Werkzeugen nicht bearbeitet werden. *XML-Schemata* beseitigen diese Mängel und bieten wesentlich mächtigere Konstrukte zur Spezifikation von Struktur, Inhalt und Semantik von XML-Dokumenten [Wor01k].

3.2.2.2 Extensible Style Language

Zur Transformation von XML-Dokumenten in andere Formate wurde die Sprache *Extensible Style Language (XSL)* entwickelt [Wor01i, BM98]. Sie bietet die Möglichkeit XML-Dokumente in ein anderes (XML-)Format zu überführen (*Extensible Style Language Transformation (XSLT)*). Mit XSL ist es zum Beispiel möglich, ein XML-Dokument nach HTML umzuwandeln.

Eine XSL-Vorlage zur Transformation des Adreßbuches in eine HTML-Tabelle könnte folgendermaßen aussehen:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
    <table>
      <tr>
        <th>Name</th> <th>Strasse</th> <th>Ort</th>
        <th>Telefon</th> <th>E-Mail</th>
      </tr>
      <xsl:for-each select="adressbuch/eintrag">
      <tr>
        <td><xsl:value-of select="nachname"/>
        ...
        <td><xsl:value-of select="plz"/>
            <xsl:value-of select="ort"/></td>
        <td>
          <xsl:for-each select="telefon">
            <xsl:value-of select="."/>
          </xsl:for-each>
        </td>
        <td>
        <xsl:for-each select="email">
          <xsl:value-of select="."/>
        </xsl:for-each>
        </td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template> </xsl:stylesheet>
```

Der einfache Aufbau von XML einerseits und die vielfältigen Möglichkeit zur Beschreibung komplexer Datenstrukturen andererseits machen XML zu einem idealen Format für die Speicherung und den Austausch unterschiedlicher Daten in heterogenen Umgebungen. Mit XSL besteht zudem die Möglichkeit, die Daten einfach in ein anderes Format zu überführen. Daher ist die Bedeutung von XML in den letzten Jahren enorm gewachsen, und es sind zahlreiche XML-Applikationen entstanden.

Einige Beispiele dafür sind:

- *Extensible HyperText Markup Language (XHTML)*
  In HTML gibt es zahlreiche Sprachkonstrukte (z.B. Kommentare, <p>, <hr>, usw.), welche nicht „XML-konform“ sind. XHTML bietet die Möglichkeit, HTML-Dokumente als XML-Dokument zu formulieren [Wor00b] und damit die Vorteile von XML (z.B. DTD, XSL) nutzen zu können.

- *Simple Object Access Protocol (SOAP)*
  SOAP bietet, ähnlich wie RPC [Blo92], die Möglichkeit eines entfernten Dienstaufrufs. Die Kommunikation zwischen den beiden beteiligten Rechner läuft dabei in Form von XML-Dokumenten ab [Wor01h]

- *Simple Workflow Access Protocol (SWAP)*
  SWAP wurde entwickelt, um die Interoperabilität zwischen verschiedenen WfMS und den von ihnen unterstützten Anwendungen zu ermöglichen [SWA01, BK99, HPS+00]. SWAP definiert dabei keine API, sondern ein XML-basiertes Nachrichtenprotokoll [Ben01].

- *Wf-XML*
  Wf-XML soll die Interoperabilität von WfMS fördern, indem es XML-Dokumente als Nachrichten zwischen den beteiligen WfMS benutzt [Wor00a, HPS+00]. Der einfache Aufbau der Nachrichten und die Implementierungsunabhängigkeit von XML (Unabhängigkeit von Programmiersprache, Middleware oder Betriebssystem) erleichtern die Unterstützung durch viele WfMS [Ben01]. Wf-XML stellt eine Erweiterung von SWAP dar und beseitigt eine Mängel.

- *VoiceXML*
  Das Hauptziel von *VoiceXML* ist es, die Vorteile von Web-basierten Applikationen und Inhalten für interaktive, sprachgesteuerte Anwendungen (z.B. für Mobiltelefone mit *VoiceXML*-Funktionalität) bereitzustellen [Voi00]. *VoiceXML* unterstützt durch eine XML-basierte Beschreibungssprache die Erstellung von Sprachdialogen.

3.2.3 Wireless Markup Language

Die *Wireless Markup Language (WML)* ist eine von XML abgeleitete Seitenbeschreibungssprache, welche speziell für mobile Endgeräte (z.B. Mobiltelefone) entwickelt wurde [WAP01a, WAP01b]. Sie soll den Abruf von Informationen aus dem WWW und die Interaktion mit Web-Servern ermöglichen. Da mobile Endgeräte meist nur über beschränkte Eingabe- und Darstellungsmöglichkeiten sowie eine Netzanbindung mit geringer Bandbreite verfügen, wurde WML als Alternative zu komplexen HTML-Seiten mit vielen Bildern und Skripten spezifiziert. WML unterstützt die Darstellung auf kleinen Displays dadurch, daß sich ein WML-Dokument in mehrere kleine Seiten (*cards*) aufteilen

läßt. Zwischen diese Seiten kann dann schnell hin und her gewechselt werden. Hier ein einfaches Beispiel für ein WML-Dokument:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml" >
<wml>
  <card id="card1" title="Willkommen">
  <p> Herzlich Willkommen bei ... </p>
  </card>
  <card id="card2" title="Inhalt">
  Hier sehen Sie ...
  </card>
</wml>
```

Die Übertragung der WML-Dokumente erfolgt meistens über das *Wireless Application Protocol (WAP)*, welches vom *WAP-Forum* für den drahtlosen Datenverkehr entwickelt wurde [WAP01a].

3.2.4 Zusammenfassung

HTML ist momentan (noch) das „Standardformat“ für Dokumente im WWW, aber XML gewinnt zunehmend an Bedeutung. Insbesondere die Möglichkeit, Daten strukturiert und clientunabhängig speichern zu können, macht für den Einsatz auf Serverseite sehr interessant. Mit XSLT steht zudem eine leistungsfähige Technologie zur Weiterverarbeitung von XML zur Verfügung. Durch individuelle XSL-Vorlagen können die XML-Daten für die Darstellung auf den unterschiedlichen Web-Clients aufbereitet werden, ohne die Daten selbst verändern zu müssen. WML stellt eine „Übergangslösung“ dar bis die mobilen Endgeräte bessere Darstellungsmöglichkeiten bieten.

	HTML	**XML**	**XML + XSL**	**WML**
Struktur	○	+	+	-
Inhalt	○	+	+	○
Darstellung	+	○	+	–
Verbreitung	+	○	○	–

Erläuterung: + = gut, ○ = neutral, – = schlecht

Tabelle 3.1: Bewertung der Beschreibungssprachen

3.3 Architekturmodelle

Zum besseren Verständnis von Web-Applikation ist es sinnvoll, die Architektur von Software-Anwendungen genauer zu untersuchen. Es gibt zahlreiche Architekturmodelle, welche sich hinsichtlich Unterteilung in Applikationskomponenten, logische Schichten und Verteilung auf mehrere Recher unterscheiden. Eine Anwendung läßt sich im wesentlichen in folgende Applikationskomponenten aufgliedern:

- Präsentationslogik (*Presentation Logic*)
- Applikations- oder Geschäftslogik (*Business Logic*)
- Datenverwaltunglogik (*Data Logic*)

Aufgabe der *Presentation Logic* ist die Interaktion mit dem Benutzer durch Bildschirmausgaben und die Erfassung von Tastatur- und Mauseingaben. Die *Business Logic* stellt das eigentliche Programm dar, welches Berechnungen durchführt oder Entscheidungen trifft. Bei vielen Anwendungen ist diese Anwendungslogik im Programmcode hart „verdrahtet“, was rasche und fehlerfreie Änderungen deutlich erschwert. Bei WfMS wird diese Ablauflogik auf einer anderen konzeptuellen Ebene im WfMS realisiert, so daß der Client kein Wissen über den Gesamtablauf des Vorgangs benötigt. Die *Data Logic* realisiert „den physischen Zugriff auf die in Datenbank- und Dateisystem gespeicherten Daten“ [Dad96] und bietet Schnittstellen (z.B. SQL) für einen standardisierten Datenaustausch.

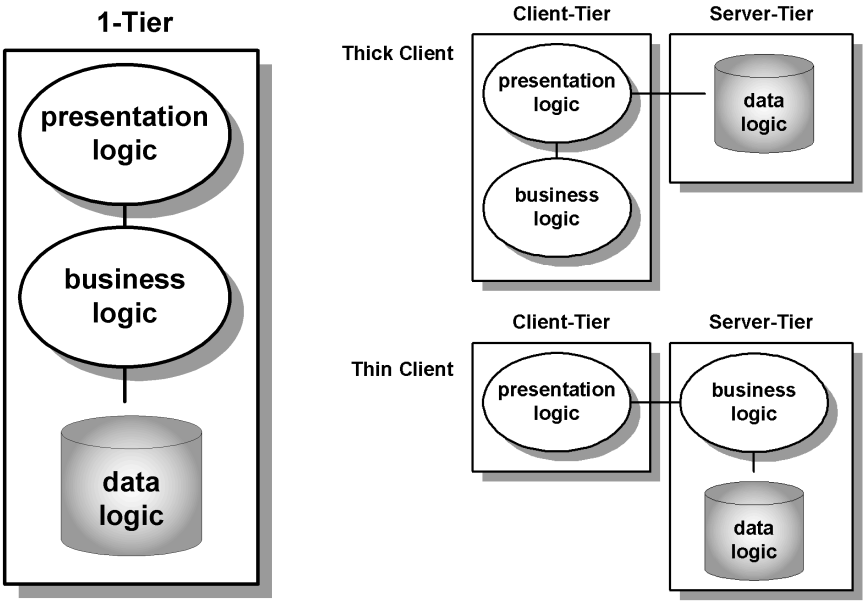

Abbildung 3.6: 1- und 2-Tier-Architektur

Die drei genannten Applikationsbestandteile können in unterschiedlichen logischen Schichten (*Tiers*) implementiert werden. Man spricht je nach Anzahl der Schichten von One-, Two-, Three- oder sogar Multi-Tier-Architekturen. Die Schichten selbst können alle auf einem Rechner realisiert oder über mehrere Rechner verteilt sein. Die wichtigsten Architekturmodelle werden in den folgenden Abschnitten kurz vorgestellt.

3.3.1 One-Tier-Architektur

Bei diesem Modell sind alle Applikationskomponenten in einer Schicht auf einem Rechner implementiert (vgl. Abbildung 3.6). Dieses entspricht der Vorstellung von einem Zentralrechner, auf dem die Daten erfaßt, verarbeitet und gespeichert werden.

3.3.2 Two-Tier-Architektur

Bei einer Two-Tier-Architektur (siehe Abbildung 3.6) sind Applikationskomponenten in zwei Schichten implementiert. Eine davon befindet sich auf dem Server und die andere auf dem Client. Deshalb spricht man auch von einer Client-/Server-Architektur. Der Server ist in der Regel für die Verwaltung der Daten zuständig, und der Client übernimmt die Aufgabe der Präsentation. Die *Business Logic* kann sowohl Client- als auch Serverseitig implementiert sein.

Da Web-Clients nur über eingeschränkte Möglichkeiten der Programmausführung verfügen (siehe Kapitel 5) und im wesentlichen der Darstellung und Benutzerinterkation dienen, werden sie auch als *Thin Client* bezeichnet. Im Gegensatz dazu stehen Applikationen, welche direkt mit der Datenbank kommunizieren können und die *Business Logic* (oder zumindest Teile davon) selber ausführen. Sie sind meist auf dem Client-Rechner lokal installiert und werden als *Thick Client* bezeichnet. Die Entscheidung zwischen *Thick* und *Thin Client* hängt von der Leistungsfähigkeit des Client-Rechners, dem Kommunikationsvolumen zwischen den Schichten und Sicherheitbestimmungen ab.

Die Kommunikation zwischen Client und Server läuft meist über eine sog. *Middleware* ab (vgl. Abschnitt 2.4.2.1). Sie ist eine standardisierte, anwendungsneutrale Systemsoftware, welche neben der reinen Übertragung von Informationen auch Namens-, Transaktions- oder Netzwerk-Dienste bereitstellt.

3.3.3 Three- und Multi-Tier-Architektur

Die *Two-Tier*-Architektur mit dem reinen *Client-/Server-Modell* hat in den letzten Jahren an Bedeutung verloren, weil es neben zahlreichen Vorteilen gegenüber *One-Tier*-Anwendungen auch eine Reihe von Nachteilen gibt (z.B. Installations- und Konfigurationsaufwand auf Client-Seite). Die *Business Logic* ist meistens ziemlich eng mit der *Presentation* und der *Data Logic* verzahnt, so daß sich Änderungen oder der Austausch

von Teilen der *Business Logic* schwierig gestalten. Dies hatte die Einführung einer dritten Schicht, dem *Middle Tier*, zur Folge (vgl. Abbildung 3.7).

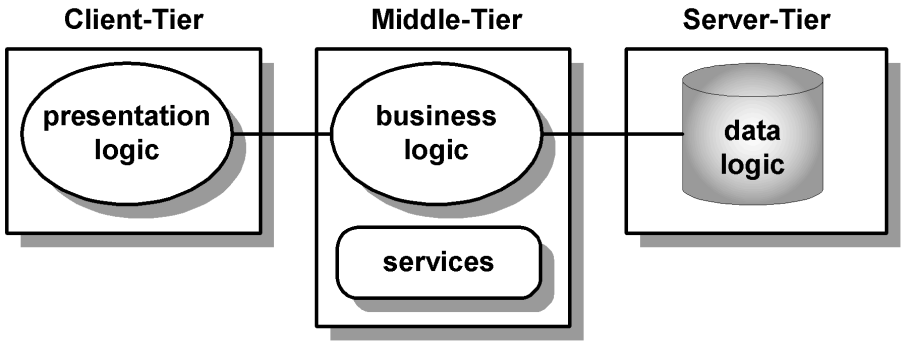

Abbildung 3.7: 3-Tier-Architektur

Die Einführung eines Middle-Tier bietet folgende Vorteile:

- höherer Abstraktionsgrad durch Trennung von *Presentation Logic*, *Business Logic* und *Data Logic*
- leichtere Austauschbarkeit der *Business Logic*
- *Middle*- und *Server*-Tier können auf mehrere Rechner verteilt werden und ermöglichen damit eine bessere Skalierbarkeit (vgl. Abschnitt 2.4.2.1)
- Ein Client benötigt weniger Funktionalität, da die *Business Logic* serverseitig ausgeführt wird
- geringerer Kommunikationsaufwand zwischen Client und Server

In bestimmten Fällen, etwa für die Realisierung eines Web-basiertes WfMS, bietet sich die Einführung einer vierten Schicht an (vgl. Abbildung 3.8). Die einzelnen Schichten haben dann folgende Aufgabe:

- *Client-Tier*: Hier findet die Präsentation der Workflow-Anwendungen (Arbeitslisten, Prozeßmonitor, usw.) statt. Es gibt unterschiedliche Clients wie Web-Browser, Handy-Displays oder Applikationen mit eigener GUI. Sie benötigen unterschiedlich aufbereitete Daten (z.B. in Form von HTML, WML, XML, usw.).
- *Web-Tier*: Diese Schicht stellt die Verbindung zwischen WfMS und Client dar. Sie interagiert mit dem WfMS, bietet Schnittstellen für unterschiedlichen Client-Typen und bereitet die Inhalte für diese dynamisch auf.

- *Business-Tier*: Hier ist das eigentliche WfMS implementiert. Es bietet Schnittstellen, über welche Komponenten des Web-Tier auf das System zugreifen können.
- *Server-Tier*: Das Server-Tier dient, wie in den bisher vorgestellten Architekturmodellen, der Verwaltung und Speicherung der Daten (*Data Logic*).

Die einzelnen Schichten haben eine klare Aufgabe und sind durch definierte Schnittstellen gekapselt. Dies ermöglicht zum Beispiel die einfache Austauschbarkeit der Datenbank im *Server-Tier* oder den Einsatz eines anderen Web-Servers im *Web-Tier*, ohne aufwendige Anpassungen in anderen Schichten vornehmen zu müssen.

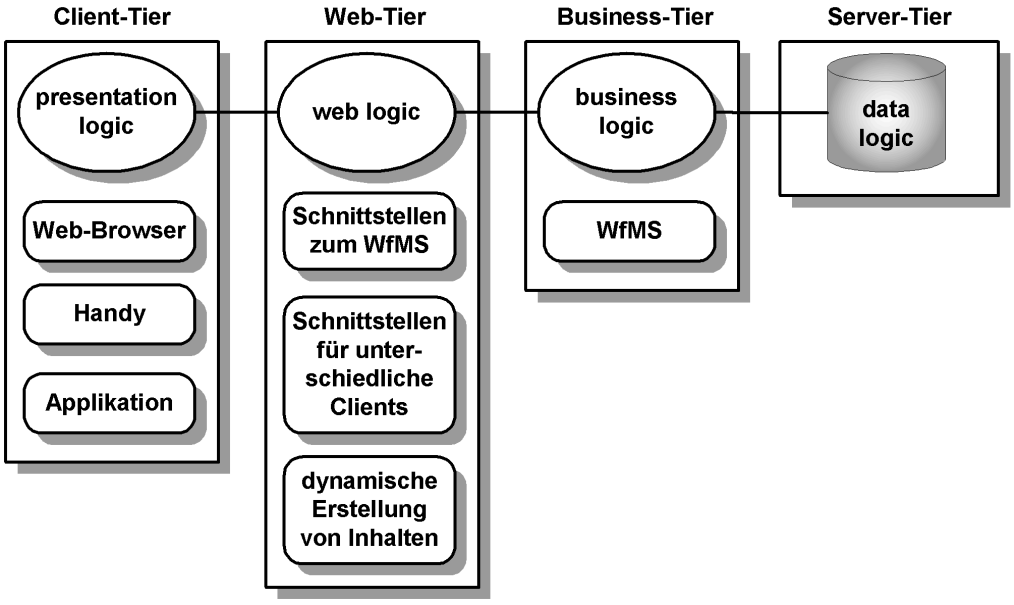

Abbildung 3.8: 4-Tier-Architektur eines Web-basierten WfMS

3.3.4 Bewertung

Die mehrschichtige Architektur (*Multi-Tier*) erfüllt die Anforderungen von WfMS aus Abschnitt 2.4 am besten.

Über die Schnittstellen des WfMS können durch das *Web-Tier* die unterschiedlichen Web-Clients angebunden werden. Am WfMS selbst müssen dabei keine Veränderungen vorgenommen werden.

Das *Web-Tier* nimmt die Anfragen der Web-Clients (incl. ihrer Parameter) entgegen und leitet an das WfMS. Die Ergebnisse werden anschließend für die verschiedenen Clientarten aufbereitet und über deren Protokolle (z.B. HTTP, WAP, usw.) versandt.

3.4 Beispiele für Web-Applikationen

In diesem Abschnitt wird anhand zweier einfacher Beispiele der exemplarische Ablauf zweier Web-Applikation aus dem Bereich *E-Commerce* und Workflow-Management demonstriert.

3.4.1 E-Commerce-Beispiel

Wie bereits in Abschnitt 3.1.3 erwähnt, ist HTTP ein zustandsloses Protokoll, d.h. der Server fällt nach der Bearbeitung einer Anfrage wieder in denselben Ausgangszustand zurück. Zwei serielle Anfragen stehen für den Server also in keinem direkten Zusammenhang zueinander. Für viele Einsatzzwecke ist es jedoch wichtig, Informationen auf dem Server zu speichern, z.B. wenn der Client ebenfalls keine Möglichkeit zur Speicherung besitzt oder wenn unnötige Datenübertragungen zwischen Client und Server vermieden werden sollen. Auf welche Weise dies funktioniert, soll an einem Beispiel aus dem E-Commerce-Bereich verdeutlicht werden.

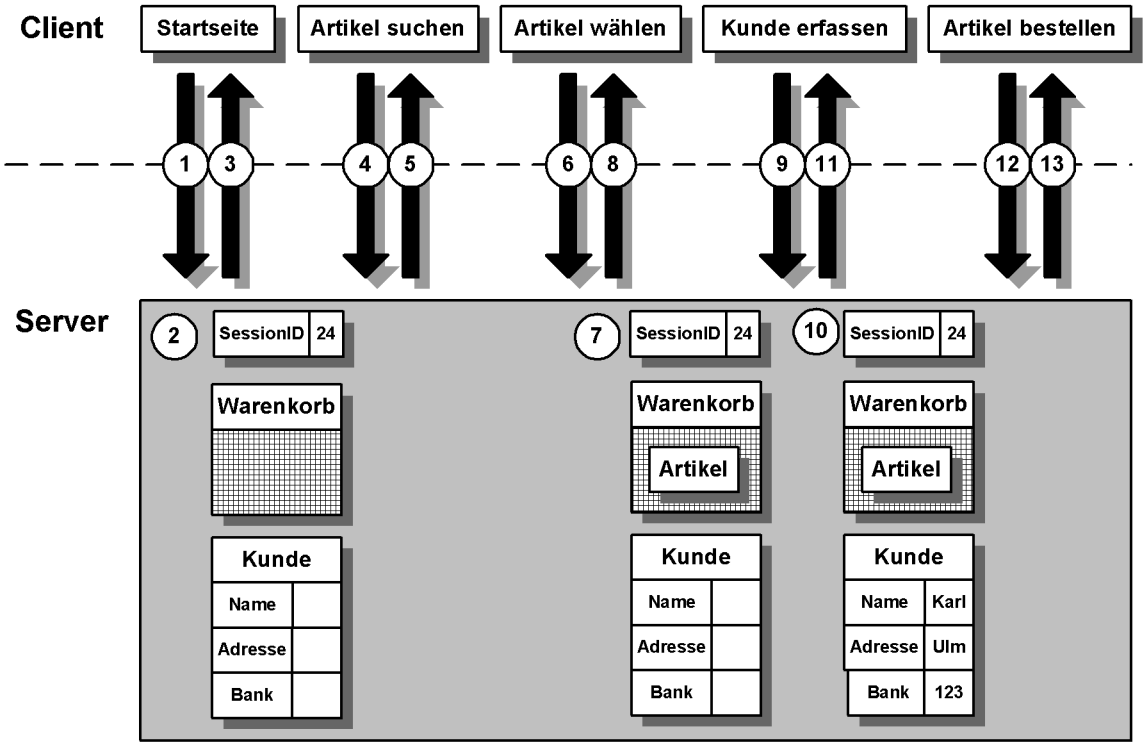

Abbildung 3.9: Beispiel einer Session aus dem E-Commerce-Bereich

Der Kunde ruft die Startseite eines „Online-Shops“ auf und bekommt vom Server eine *Session-ID* zugewiesen (vgl. Abbildung 3.9 (1)-(3)). Die *Session-ID* dient der Identifikati-

on des Clients bei den folgenden Anfragen beim Server. Außerdem werden Objekte zur Speicherung von Artikel (Warenkorb) und zum Hinterlegen von Kundendaten erzeugt. Die *Session-ID* wird bei allen Antworten des Servers in die Seite „miteingebaut“ und bei den Anfragen des Clients wieder zum Server zurückgeschickt. Auf diese Weise kann der Server bei jeder Client-Anfrage einen Bezug zwischen der Anfrage und der Session des Kunden herstellen. Die Gültigkeit der Session wird meist durch ein Timeout-Interval zeitlich begrenzt.

Das Einbinden der *Session-ID* kann auf folgende drei Arten geschehen:

- *URL Rewriting*
  In allen Links (`<a href="...">`), Formularen (`<form action="...">`) und Framedefinitionen (`<frame src="...">`) muß die Session-ID als Parameter angehängt werden. Dies kann durch spezielle Funktionen oder manuell durch Stringkonkakenation efolgen:

  ```
  newURL = oldURL + "\;jsessionid=a3d4d6e602a89c1abb7d"
  ```

  Bei der Realisierung mit Hilfe von Java Servlets (siehe Kapitel 4) bietet die Klasse `HttpServletResponse` des `javax.servlet.http`-Package eine Methode `encodeURL(java.lang.String url)` zum Einbinden der *Session-ID* in die Antwort-URL an.

- *Versteckte Eingabefelder (Hidden Input Fields)*
  Eine weitere Methode zur Übertragung der *Session-ID* bieten versteckte Eingabefelder in Formularen. Sie weden im Browser nicht dargestellt, aber ihr Inhalt wird beim Absenden des Formulars mit den anderen Formulardaten an den Server gesendet. Ein solches Eingabefeld könnte wie folgt aussehen:

  ```
  <input type="hidden" name="jsessionid"
         value="a3d4d6e602a89c1abb7d">
  ```

- *Cookies*
  Die dritte Möglichkeit eine Session-ID zu übertragen, stellen Cookies dar. Cookies sind kleine Datenpakete, welche vom Server an den Client geschickt und von diesem gespeichert werden. Dabei können Zugriffsberechtigungen für bestimmte Rechner oder Domains sowie deren Gültigkeitsdauer angegeben werden. Bei einer Anfrage sendet der Web-Browser die Informationen der Cookies zurück an den Server. Dieser kann dann zum Beispiel die *Session-ID* auslesen und somit der Anfrage die richtige Session zuordnen.

Die drei Verfahren haben dabei folgende Vor- bzw. Nachteile:

	URL Rewriting	**Versteckte Eingabefelder**	**Cookies**
Vorteil	mehrere verschiedene Sessions mit einem Browser möglich		keine dynamischen Anpassungen der Web-Inhalte nötig
Nachteil	alle Links müssen (dynamisch) um *Session-ID* erweitert werden	*Session-ID* muß (dynamisch) in ein verstecktes Formularfeld eingebunden werden, nur für formularbasierte Anwendungen geeignet	Cookie-Funktionalität muß im Browser eingeschaltet sein

Alle drei Verfahren ermöglichen es, die einzelnen Seitenaufrufe in einen gemeinsamen Kontext zu betrachten. So kann der Kunde im Laufe seines „virtuellen“ Einkaufs, Artikel suchen und diese in den Warenkorb legen, welche in seiner *Session* gespeichert ist ((4)-(8)). Möchte der Kunde die Artikel bestellen, muß er noch seine persönlichen Daten angeben ((9)-(11)). Dann kann er die Bestellung abschließen ((12)-(13)).

3.4.2 Workflow-Beispiel

Ein weiteres Beispiel zeigt Abbildung 3.10. Zuerst muß sich der Anwender dem WfMS gegenüber authentifizieren. Dazu fordert er über seinen Browser die Seite `login.html` an (1)-(2). Sie besteht aus einem Formular in welches er Benutzername und Passwort eintragen muß. Beim Absenden des Formulars werden die Benutzerinformationen an den Web-Server übertragen, und es wird eine Seite mit der Arbeitsliste des Anwenders angefordert (3). Bevor der Web-Server diese Seite generieren kann, muß er die Parameter (Benutzername, Passwort, usw.) aus der Anwort des Klienten auslesen und selbst eine Verbindung zum WfMS-Server aufbauen (4). Dann kann er die Arbeitsliste des Anwenders anfordern und dynamisch eine HTML-Seite generieren, welche zum Client geschickt wird (5). Damit sich der Client bei der weiteren Kommunikation nicht jedes Mal seine Benutzerinformationen mitschicken muß, speichert der Web-Server diese (und Informationen zur Verbindung mit dem WfMS) im Kontext der *Session*. Die Session hat solange Bestand, bis sich der Anwender ausloggt oder ein bestimmtes Zeitlimit erreicht ist. Welche Verfahren und Möglichkeiten es zur Verwaltung und Benutzung von Sessions gibt, wurde bereits in Abschnitt 3.4.1 vorgestellt.

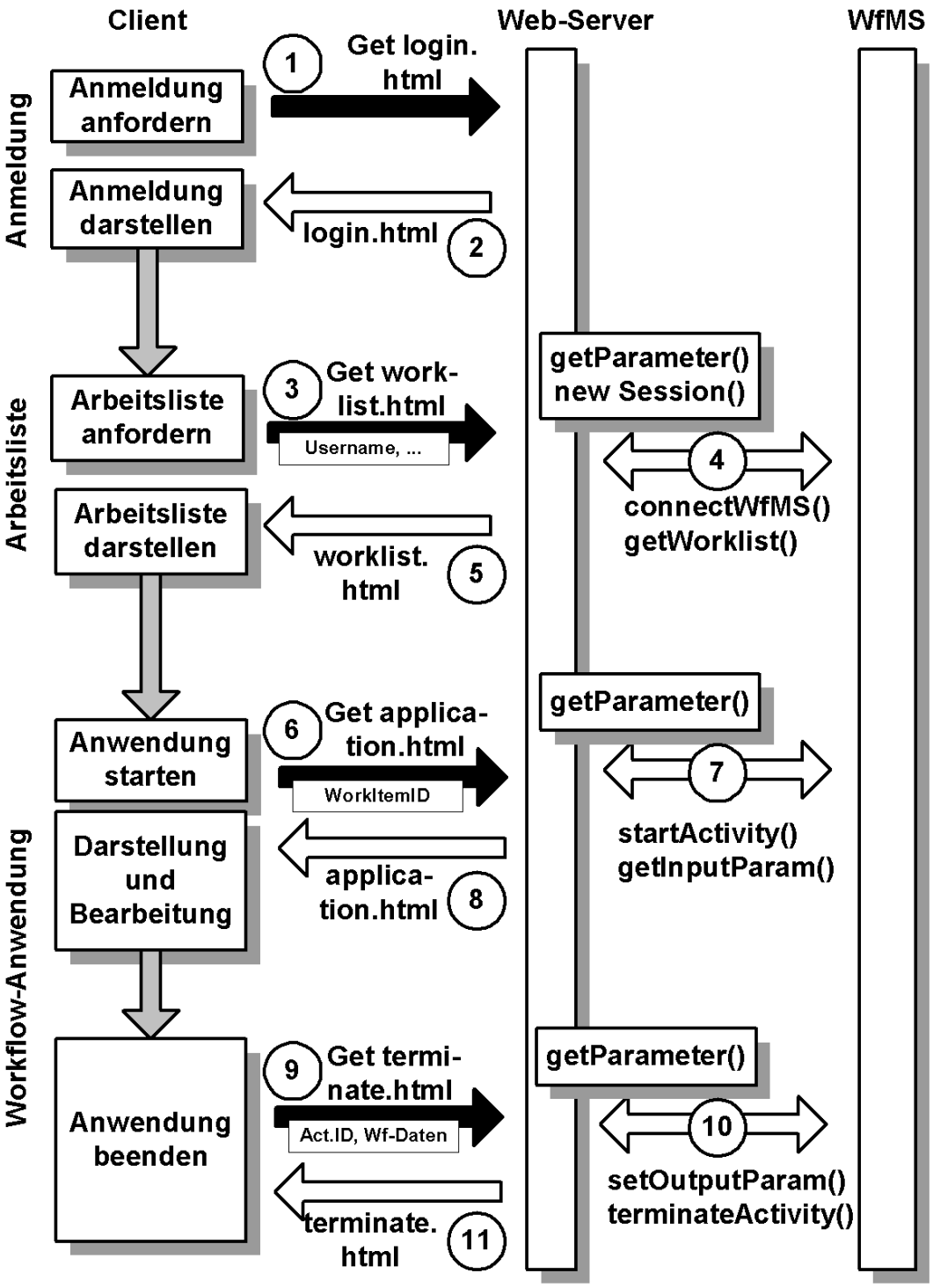

Abbildung 3.10: Beispiel einer Workflow-Sitzung

Nach Darstellung der Arbeitsliste kann der Anwender einen Arbeitslisteneintrag zur Bearbeitung auswählen. Er klickt dazu zum Beispiel auf einen Link, welcher die Seite mit der Workflow-Applikation anfordert (6). In diesem Link sind außerdem der Name bzw. die ID des Arbeitslisteneintrags als Parameter enthalten. Der Web-Server liest diese Parameter aus und startet die entsprechende Aktivität beim WfMS (7). Anschließend fragt der Web-Server die Eingabeparameter der Aktivität vom WfMS ab und erstellt daraus ein Formular in einer HTML-Seite (8). Der Anwender bearbeitet das Formular im Browser und schickt es dann an den Web-Server zurück (9). Dieser liest die Formulardaten, schreibt sie in den Ausgabecontainer der Aktivität und beendet diese (10).

Danach kann er erneut seine Arbeitsliste anfordern und das Arbeiten fortsetzen, oder er meldet sich ab und beendet damit die Workflow-Sitzung.

3.5 Zusammenfassung und Bewertung

Die Grundlagen von Web-Applikationen wurden in den vorangehenden Abschnitten vorgestellt und in Abschnitt 3.4 anhand zweier praktischer Beispiele verdeutlicht. In diesem Abschnitt sollen die Charakteristika von Web-Applikationen nochmals zusammengefaßt und im Hinblick auf die besonderen Anforderungen von Web-basierten WfMS (vgl. Abschnitt 2.4) bewertet werden.

Eine Unterscheidung zwischen „normalen“ Applikationen (z.B. „koventionelle“ Workflow-Clients) und Web-Applikationen (z.B. Web-basierten Workflow-Clients) läßt sich folgendermaßen durchführen:

	„Normale“ Applikationen	**Web-Applikationen**
GUI	verfügen über eine eigene GUI	können meist nur die GUI des Web-Clients nutzen (z.B. HTML-Elemente)
Verwaltung	werden als eigenständiger Prozeß vom Betriebssystem verwaltet	laufen im Kontext des Web-Clients ab
Zugriff	haben vollen Zugriff auf die Ressourcen des Betriebssystems	unterliegen den Sicherheitseinschränkungen des Web-Clients
Verbindungen	können direkt Verbindungen zu *Back-End-Systemen* (z.B. DBMS oder WfMS) aufbauen	können in der Regel keine direkten Verbindungen aufbauen

Nach dieser Differenzierung zwischen normalen und Web-basierten Applikationen sollen nun die generellen Vorteile von Web-Applikationen dargestellt werden:

- *Standardisierte Formate*
  Für die Übertragung (z.B. HTTP) und die Beschreibung von Inhalten (z.B. HTML) existieren standardisierte Formate, welche von (fast) allen Web-Clients unterstützt werden. Darüber hinaus gibt es spezielle Lösungen für Sicherheitsmechanismen (z.B. *Secure Socket Layer (SSL)*) und für die Anzeige spezieller Dokumenttypen (z.B. *Plug-In* für PDF, siehe Abschnitt 5.2.4), welche zwar nicht immer zum Funktionsumfang aller Clients gehören, jedoch nachrüstbar oder zumindest gut dokumentiert sind (z.B. für eine spätere Implementierung).

- *Universeller Zugriff*
  Sowohl das Internet als auch Web-Clients haben innerhalb der letzten Jahren eine sehr große Verbreitung erfahren. Ein Web-Browser zählt mittlerweile schon fast zur Standardausstattung eines PC. Im Gegensatz zu anderen Client-Programmen entfällt daher in der Regel die Notwendigkeit der Installation. Desweiteren ist der Anwender mit dem Umgang des Programms schon vertraut. Außerdem kann er seine Anwendungen von jedem vernetzten Arbeitsplatz in der Welt, egal ob im Büro, zu Hause oder vor Ort beim Kunden, nutzen.

- *Zentrale Verwaltung*
  Die Verwaltung der Applikationen (Konfiguration, Updates, usw.) erfolgt zentral auf dem Web-Server. Der Konfigurationsaufwand auf Clientseite entfällt oder er reduziert sich auf ein Minimum.

- *Plattformunabhängigkeit*
  Web-Clients sind für nahezu jede Computerplattform verfügbar. Seit einigen Jahren setzt sich zudem der Trend fort, den Zugang zum Web auch für andere Systemplattformen (z.B. Mobiltelefon, *Personal Digial Assistant, PDA*) zu ermöglichen.

- *Skalierbarkeit*
  Auf den ersten Blick mag die Ausrichtung auf einen zentralen Web-Server vielleicht als „Flaschenhals“ erscheinen. Allerdings existieren sehr gute Konzepte zur Skalierung (z.B. Replikation von Web-Servern, *Server Farm*), welche durch den Einsatz mehrschichtiger Architekturen (vgl. Abschnitt 3.3.3) noch verbessert werden können.

Neben den zahlreichen Vorteilen von Web-Applikationen existieren jedoch auch einige Probleme:

- *Programmausführung auf Clientseite*
  In seiner ursprünglichen Version sah HTML keine Möglichkeiten der clientseitigen Programmausführung vor [Wor01e]. Sämtliche Berechnungen, Eingabeüberprüfungen, usw. mußten vom Server durchgeführt werden. Dies führt zu einem

sehr hohen Kommunikationsvolumen, weil zuerst immer alle Daten an den Server übertragen werden müssen und das Ergebnis dann wieder an den Client zurückgeschickt wird. Erst mit der Einführung der Skriptsprache *Javascript* und der Fähigkeit der Clients, compilierte Programmbausteine (z.B. Applets und ActiveX-Steuerobjekte) ausführen zu können, wurde die Ausführung von Programmen auf Clientseite möglich. Allerdings bieten die angesprochenen Verfahren teilweise nur sehr begrenzte Möglichkeiten (z.B. aufgrund von Sicherheitseinschränkungen wie bei der *Java Virtual Machine (JVM)*) und stehen damit Clients, welche als „normale“ Applikation realisiert sind, nach. Inbesondere der Aufruf von externen Programmen (inkl. Parameterübergabe), welche nicht im Browser ablaufen stellt ein großes Problem dar. Allerdings sind diese externen Programme (z.B. ein Textverarbeitungsprogramm) gerade bei Workflow-Applikationen besonders wichtig.

- *Formularorientierung*
  Web-Applikationen mit HTML liegt die Idee eines Formulars (vgl. Abschnitt 3.2.1.3) zu Grunde. Web-basierten Workflows sollten sich deshalb formularorientiert (vgl. Abschnitt 2.3.1) darstellen lassen, wobei die Ablaufsteuerung durch das WfMS serverseitig abgewickelt wird.

- *Eingeschränkte GUI*
  Die (graphische) Benutzeroberfläche von Web-Applikationen ist durch die Seitenbeschreibungssprache (z.B. HTML) stark vorgegeben und kann nur in sehr begrenztem Umfang (z.B. durch Applets oder ActiveX-Steuerobjekte) angepaßt werden.

- *Client-Server-Interaktion und Updates*
  Durch den *Request-Response-Mechanismus* von HTTP ist ein festes Kommunikationsschema vorgegeben. Der Client fordert eine Seite an (*Request*), woraufhin der Server die Anfrage bearbeitet und die entsprechende Seite an den Client schickt (*Response*). Ein Problem entsteht, wenn der Server den Client (aktiv) über ein Ereignis benachrichtigen möchte (z.B. eine Veränderung in der Arbeitsliste). HTTP selbst unterstützt diese serverseitig initiierte Form der Kommunikation nicht. In Abschnitt 5.2.2 werden Verfahren für eine Lösung dieses Problems vorgestellt.
  Ein weiteres Problem in diesem Zusammenhang ist, daß Aktualisierungen (*Updates*) von Teilbereichen innerhalb eines HTML-Dokuments nicht möglich sind. Sollen zum Beispiel einzelne Einträge innerhalb einer Tabelle aktualisiert werden, so muß nicht nur die gesamte Tabelle, sondern das komplette Dokument neu übertragen werden. Dies erhöht das zu übertragende Datenvolumen enorm. Innerhalb von *Frames* (vgl. Abschnitt 3.2.1.4) sind solche Updates zwar möglich, allerdings bestehen immer viele Einschränkungen im Vergleich zu „normalen“ Applikationen.

- *Sitzungssemantik*
  Für Workflow-Anwendungen ist eine Sitzungssemantik von großer Bedeutung, weil nach der Anmeldung beim WfMS mehrere Aktionen im selben Kontext ablaufen. Da HTTP selbst keine Sitzungssemantik kennt, muß diese manuell implementiert werden (vgl. Beispiel in Abschnitt 3.4.2).

- *„Freie“ Navigation*
  HTML und die Funktionalität der Browser ermöglichen eine leichte Navigation zwischen verschiedenen Web-Inhalten und Servern. Dies kann über *Hyperlinks*, die direkte Eingabe einer *URL* oder durch Funktionen des Web-Browser (z.B. Zurückspringen zur letzten Seite) erfolgen. Für Web-Applikationen, bei denen die Einhaltung einer bestimmten Reihenfolge von Seitenaufrufen meist von sehr großer Bedeutung ist, bereitet diese „freie“ Navigation Probleme.

Wie die angesprochenen Vorteile für Web-basierte WfMS genutzt bzw. die Probleme gelöst werden können, wird in den folgenden beiden Kapiteln angesprochen. Kapitel 4 adressiert vor allem die Integration des WfMS in den Web-Server und der Bereitstellung der Applikationen für unterschiedliche Web-Clients, während sich Kapitel 5 vorrangig mit den Möglichkeiten der Darstellung und Programmierung im Web-Client befaßt. In Kapitel 8 werden die Stärken und Schwächen von Web-Clients im Vergleich zu konventionellen Workflow-Clients genauer diskutiert.

Kapitel 4

Web-Server

In den letzten Jahren entwickelte sich das Web weg von der reinen Plattform zur Präsentation von statischen Hypertextinhalten, hin zu einer Basis für Applikationen, welche das Web als Schnittstelle zum Anwender betrachten. Die „klassischen“ (HTTP-)Web-Server waren für die Unterstützung statischer Inhalte konzipiert, welche in der Regel auf einer Festplatte abgelegt waren. Web-Applikationen erfordern jedoch dynamisch generierte Inhalte, welche individuell auf die Applikationsparameter, den Anwender (*Personalisierung*) und den Client angepaßt werden können. Die Interoperabilität mit anderen *Back-End-Systemen* wie Datenbanken oder WfMS ist für die Abwicklung der durch die Web-Applikation realisierten Geschäftsprozesse ebenfalls von entscheidender Bedeutung.

Mit welchen Techniken Web-Server die Anforderungen hinsichtlich Web-Anbindung von WfMS (vgl. Abschnitt 2.4) unterstützen und mit den Charakeristika von Web-Applikationen (vgl. Abschnitt 3.5) verbinden, wird in den weiteren Abschnitten anhand folgender Kriterien untersucht:

- Interoperabilität
  - Zugriff auf WfMS-API bzw. -Komponenten
  - Kommunikationsmechanismen und *Middleware*-Unterstützung
- Dynamische Inhalte
  - Performanz bei der Generierung
  - Ausgabeformate (z.B. HTML, XML, Grafiken, usw.)
  - Unterstützung unterschiedlicher Web-Clientarten
- Anwendungsentwicklung
  - Aufwand zur Entwicklung bzw. Wartung
  - benötigtes (Fach-)Wissen (z.B. Programmiererfahrung)

- Architektur
  - Aufbau (monolithisch oder verteilt, z.B. Multi-Tier)
  - Skalier- und Verfügbarbarkeit
  - angebotene Zusatzdienste (z.B. Sitzungsunterstützung, Transaktionssicherheit, Authentifizierungsmechanismen, usw.)
- Offenheit
  - Einzelprodukt eines Herstellers oder offener Standard
  - Plattformunabhängigkeit

4.1 HTTP-Server

Zu der Grundfunktionalität eines jeden Web-Servers zählt die Unterstüzung des *Hypertext Transfer Protocol (HTTP)*. Der Web-Server arbeitet hier in ersten Linie als *File-Server*, indem er die HTTP-Anfragen interpretiert, die entsprechende Datei von der Festplatte liest und mittels HTTP an den Client überträgt.

Web-Applikationen erfordern jedoch, daß der Server selbst Aktionen durchführt (z.B. Durchsuchen einer Datenbank) und die Ergebnisse in geigneter Form (z.B. als HTML-Dokument) dem Client zugänglich macht. Für die Erweiterung der Web-Server um solche „Zusatzfunktionalitäten“ wurden diverse Technologien entwickelt. Die wichtigsten von ihnen werden in den folgenden Abschnitten vorgestellt und mit Hinblick auf den Einsatz für Web-basierte WfMS beurteilt.

4.1.1 Common Gateway Interface

Das *Common Gateway Interface (CGI)* ist die einfachste und zugleich älteste Möglichkeit, dynamisch Web-Seiten zu generieren [PW01]. CGI ist eine standardisierte Schnittstelle für die Kommunikation zwischen Web-Server und externen Programmen. Die Kommunikation läuft über Systemumgebungsvariablen oder über die Standardeingabe ab. Der Hauptvorteil von CGI ist eine breite Unterstützung durch viele Web-Server sowie die Unabhängigkeit von einer konkreten Programmiersprache.

Trotz dieser Freiheit sind in der Praxis die meisten Programme Skripte *(CGI-Scripts)*, welche für die *UNIX-Shell* oder in der Programmiersprache *Perl (Practical Extraction and Reporting Language)* [SSP99] geschrieben sind. Perl ist eine leistungsfähige objektorientierte Programmiersprache, welche zum Beispiel Module für den Datenbankzugriff, die Benutzung von Netzwerkdiensten und den speziellen Einsatz als *CGI-Script* bietet. Das *CGI-Modul* erleichtert dem Entwickler die Erstellung von HTML-Seiten und den Zugriff auf HTTP-Parameter [SSP99].

Die Programmiersprachenunabhängigkeit ermöglicht die Interoperabilität mit nahezu allen WfMS. CGI unterstützt keine *Middleware*-Technologien (wie etwa CORBA), sondern definiert als Schnittstelle lediglich die Kommunikation über Systemumgebungsvariablen und über die Standardeingabe. Dies macht insbesondere den Aufbau von mehrschichtigen und über mehrere Rechner verteilten (z.B. wegen Skalierbarkeit) Architekturen schwierig.

Eines der größten Probleme von CGI ist jedoch die fehlende Trennung von statischen und dynamischen Inhalten einer Web-Seite. Dies hat zur Folge, daß immer die komplette HTML-Seite von dem Programm (z.B. auch das statische Grundgerüst via `println("<html>");` über durch das Programm über die CGI-Schnittstelle ausgegeben werden muß, obwohl vielleicht nur ein kleiner Teil der Seite dynamische Veränderungen aufweist. Dies macht vor allem die Anwendungsentwicklung sehr aufwendig.

Ein weiteres Problem ist, daß jedes *CGI-Script* in einem eigenen Prozeß ausgeführt wird. Für jeden Aufruf eines Skripts muß vom Betriebssystem ein neuer Prozeß erzeugt werden. Da es sich hierbei um eine sehr „teure“ Operation handelt [Web98], führt dies zu einem großen *Overhead* und zu einer schlechten Performanz von *CGI-Scripts*. Diesen Nachteil versucht *FastCGI* [PW01] zu vermeiden, indem das Skript nach der Bearbeitung einer Anfrage nicht terminiert, sondern auf das Eintreffen neuer Anfragen wartet. Außerdem kommen schnelle Verfahren zur Parameterübergabe (bidirektionale TCP-Verbindungen oder *Pipes*) zum Einsatz.

Für Perl und den Apache Web-Server [Apa01a] gibt es mit *Embedded Perl* (z.B. Module *Apache::ePerl* und *Apache::embperl*) Lösungen, welche den Perl-Code direkt in das HTML-Dokument integrieren und im Web-Server ausführen [PW01]. Die Einbettung beseitigt zwar einige der obengenannten Probleme, führt aber insgesamt zu einer Abhängigkeit von der verwendeten Programmiersprache (Perl) und dem eingesetzten Web-Server (Apache).

4.1.2 PHP

PHP ist eine Skriptsprache, welche im Gegensatz zu Perl speziell für Web-Applikationen entwickelt wurde [PHP01]. PHP bietet neben Bibliotheken zur Anbindung an diverse Datenbanken und Schnittstellen zu verschiedenen Netzdiensten (z.B. SMTP, POP, IMAP oder LDAP) zusätzliche Funktionen [PW01], welche vor allem im Zusammenhang mit Web-Applikationen häufig benötigt werden. So sind zum Beispiel die dynamische Generierung von GIF-Grafiken und PDF-Dokumenten sowie die Unterstützung von XML-Applikationen oder einer Sitzungssemantik bereits in PHP enthalten.

Ein direkter Zugriff auf das WfMS bzw. dessen API ist in der Regel jedoch nicht möglich. Hierzu müssen erst spezielle Schnittstellen-Module entwickelt werden, welche die Interoperabilität ermöglichen. Anschließend gestaltet sich die Anwendungsentwicklung und -wartung in PHP durch die oben angesprochenen Funktionen sehr komfortabel.

Der PHP-Code wird (ähnlich wie bei *Embedded Perl*) direkt in das HTML-Dokument

integriert. Der PHP-Processor verarbeitet den Code und fügt die Ausgabe in das HTML-Dokument ein, bevor es zum Client geschickt wird.

Die einfache Anwendungsentwicklung in PHP soll nun an einem kleinen Beispiel-Skript, welches der Suche von E-Mail-Adressen in einer Datenbank dient, verdeutlicht werden:

```
<html>
...
<body>
<h1>E-Mail-Adressen-Suche</h1>
<?php
    ...
    $link = mysql_pconnect($host, $user, $pass);
    $query = "select id, email from $table where ".
        "nachname = '$nachname' and vorname = '$vorname'";
    $res = mysql_query($query, $link);
    $r = mysql_fetch_array($res, MYSQL_ASSOC);
    print "Benutzer " . $vorname . " " . $nachname;
    print " hat folgende Email-Adresse:<br>":
    print "<b>" . $r["email"] . "</b>";
?>
</body>
</html>
```

Das obige Skript baut eine Verbindung zu einer (MySQL-)Datenbank auf und fragt über ein *SQL-Statement* die E-Mail-Adresse des durch die Parameter `$vorname` und `$nachname` spezifizierten Benutzers ab. Anschließend wird das Ergebnis durch die `print`-Funktion ausgegeben.

PHP ist eine *Open Source* Entwicklung und läßt sich als Modul in den Apache Web-Server integrieren [PW01, Apa01a]. Eine Verfügbarkeit ist damit für viele Systemplattformen gewährleistet.

4.1.3 Server Side Includes

Server Side Includes (SSI) [Len01, NCS01, Apa, Wor01f] werden wie bei PHP in das HTML-Dokument integriert sowie vom Web-Server geparst und interpretiert. Anschließend wird die Ausgabe in die HTML-Seite eingefügt. Um das zeitaufwendige Parsen aller vom Web-Server angeforderten Dateien zu verhindern, werden Dateien mit SSI-Elementen normalerweise mit der Endung `".shtml"` gespeichert. Die SSI-Anweisungen sind in HTML-Kommentare eingebettet und haben folgendes Format [NCS01]:

```
<!--#command tag1="value1" tag2="value2" -->
```

Es gibt zahlreiche Befehle zum Einfügen von Inhalten aus Dateien (`#include`), zum Ausführen von externen Programmen (`#execute`) und zur Ausgabe von Parametern (`#echo`).

Allerdings gibt es keinen einheitlichen Standard, sondern jeder (Web-Server-)Hersteller implementiert unterschiedliche SSI-Funktionalitäten. So bieten einige Web-Server neben den oben angesprochenen Befehlen auch Kontrollkonstrukte wie IF-Abfragen oder Schleifen sowie Methoden zum Zugriff auf Datenbanken. Der fehlende Standard erschwert die Server-übergreifende Entwicklung von Web-Applikationen mit SSI. Für eine detaillierte Befehlsreferenz sei auf die jeweiligen Dokumentationen der Hersteller verwiesen [NCS01, Apa, Wor01f].

Der Aufbau einer Datei mit SSI-Elementen und die Leistungsfähigkeit von SSI sollen an folgendem kleinen Beispiel verdeutlicht werden:

```
<html>
...
<body>
<!--#include virtual="/header.html"-->

<!--#config timefmt="%A %B %d, %Y" -->
Heute ist der <!--#echo var="DATE_LOCAL" -->
und Sie betrachten gerade das Dokument <!--#echo var="DOCUMENT_NAME"--
>
mit einem <!--#echo var="HTTP_USER_AGENT"--> Browser.

Auf dem Web-Server laufen gerade folgende Prozesse:<br>
<!--#exec cmd="ps -a" -->

Sie sind der <!--#include virtual="/cgi-bin/counter.pl" -->. Besucher.
</body>
</html>
```

Durch die erste `#include`-Anweisung wird der Kopfteil des Dokuments aus der Datei `header.html` eingebunden. Danach werden einige Sytemvariablen des Servers durch die `#echo`-Anweisung in das Dokument eingefügt. Die `#exec`-Anweisung ruft über das Betriebssystem das Programm `ps` (Anzeige der im System laufenden Prozesse), dessen Ausgabe ebenfalls eingefügt wird. Schließlich wird noch ein Zähler (externes Perl-Skript) gerufen und die Anzahl der Seitenaufrufe in den Text integriert.

SSI bietet selbst nur sehr begrenzte Möglichkeiten einer sinnvollen Anwendungsentwicklung. Der Import von statischen Seitenelementen (z.B. Kopf- oder Fußteil) ist ein Beispiel dafür. Komplexe Anwendungsteile, wie die Interaktion mit dem WfMS, können nur in externen Programmen realisiert werden. Diese werden, wie bei CGI, als eigenständiger Prozeß ausgeführt, was zu den bereits in Abschnitt 4.1.1 besprochenen Nachteilen führt.

4.1.4 Web-Server-Module

Viele Web-Server bieten die Möglichkeit, selbst programmierte Module in den Server einzubinden und definierte HTTP-Anfragen von ihnen abwickeln zu lassen [Apa01a]. Die Kommunikation zwischen Web-Server und Modul erfolgt dabei über eine spezielle API des Web-Servers (z.B. Apache API, NSAPI oder ISAPI) [Apa01b], welche dem Modul zum Beispiel Informationen über HTTP-Parameter zur Verfügung stellen. In der Regel sind die API der Web-Server nur für eine begrenzte Anzahl von Programmiersprachen verfügbar. Dies führt vor allem dann zu Problemen, wenn es keine gemeinsame Programmiersprache für die Web-Server- und die WfMS-API gibt.

Durch die enge Integration mit dem Server stellen Module eine sehr performante Lösung der serverseitigen Programmierung dar. Allerdings müssen sie für die spezifische API eines Web-Servers entwickelt werden und sind in der Regel zwischen mehreren Web-Servern nur mit hohem Aufwand portierbar. Einen weiteren Nachteil stellt (wie bei CGI) die fehlende Trennung zwischen statischen und dynamischen Seiteninhalten dar.

4.2 Web Application Server

Die in Abschnitt 4.1 angesprochenen Technologien ermöglichen zwar die serverseitige Implementierung von Web-Applikationen, sie stellen jedoch in der Regel nur partielle Erweiterungen der HTTP-Funktionalität dar. *Web Application Server* [Mon01] bieten im Gegensatz dazu eine „Gesamtarchitektur“ für die Erstellung Web-basierten Applikationen an. Neben der Grundfunktionalität eines Web-Servers bieten sie zum Beispiel eine Unterstützung für:

- Transaktionen (z.B. *JTS*, *MTS*, *CTM* oder *CICS*)
- Datenbankzugriffe (z.B. *JDBC Connection Pools*)
- Namensdienste (z.B. *JNDI*, *X.500* oder *LDAP*)
- Authentifizierung (z.B. *JAAS*, *NIS*)
- Messaging (z.B. *JMS*)

Für *Web Application Server* haben sich zwei Architekturen etabliert. Dies sind zum einen die *Java 2 Enterprise Architecture* von Sun [Sun01b] und zum anderen der *Microsoft Internet Information Server* von Microsoft [Mic01c]. Sie werden in den beiden folgenden Abschnitten vorgestellt.

4.2.1 Java 2 Enterprise Edition

Unter dem Namen *Java 2 Enterprise Edition (J2EE)* wurde von Sun 1999 eine neue Java-Architektur eingeführt [Sun01b], welche spezielle Technologien zum Server-seitigen Einsatz von Java und zur Entwicklung von Web-Applikationen [Mon01] enthält. Die J2EE-Server werden *Application Server* genannt und implementieren die in der Spezifikation enthaltenen Dienste. Durch die gute Unterstützung Web-basierter Lösungen hat sich für J2EE-Server auch der Begriff *Application Server* etabliert. Die einzelnen Baustandteile von J2EE (vgl. Abbildung 4.1) werden in den folgenden Abschnitten vorgestellt, und abschließend ihr Zusammenspiel in der Gesamtarchitektur erläutert.

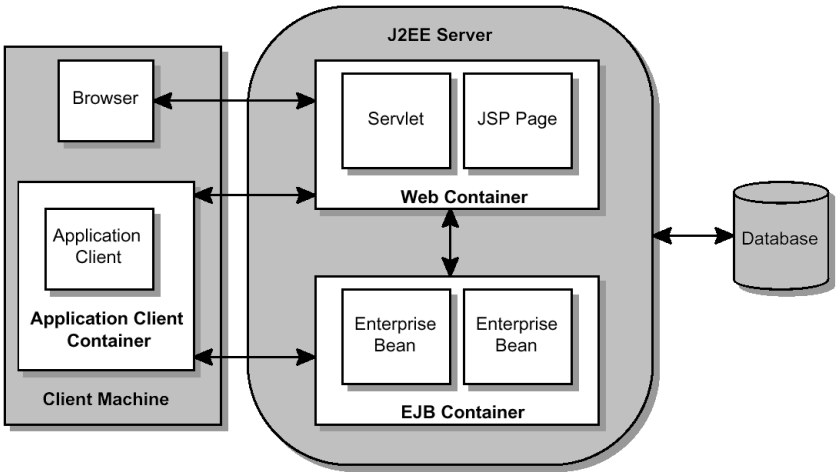

Abbildung 4.1: *Java 2 Enterprise Edition* Architektur [BGJ+01]

4.2.1.1 Enterprise JavaBeans

Zentrales Element moderner Softwareentwicklung sind *Software-Komponenten* [Bal01]. Komponenten sind abgeschlossene Software-Bausteine, welche einfach zu komplexen Anwendungen zusammengestellt werden können.

Mit *JavaBeans* wurde von Sun bereits ein (allgemeines) Komponentenmodell für Java geschaffen [Sun01f]. Allerdings besitzen *JavaBeans* keinen speziellen Fokus und damit auch keine besondere Unterstützung für bestimmte Einsatzgebiete. Sie können sowohl auf Client- als auch auf Server-Seite für verschiedenen Zwecke (z.B. als graphische GUI-Komponente) eingesetzt werden.

Im Gegensatz dazu stellen *Enterprise JavaBeans (EJB)* ein Server-seitiges Komponentenmodell dar, welches speziell für die Entwicklung und den Betrieb von komponentenbasierten Geschäftsanwendungen spezifiziert wurde [Sun01e, Mon01, GT00]. Besonderer Wert wurde bei EJB auf Eigenschaften wie Transaktionssicherheit, Portabilität,

Mehrbenutzerbetrieb, Skalierbarkeit und die einfache Anbindung von persistenten Datenspeichern (z.B. Datenbanken) gelegt. Folgende Typen von EJB werden unterschieden:

- *Entity Beans*
  Objekte werden durch *Entity Beans* modelliert und verwaltet (z.B. persistente Datenspeicherung). Ein Komponenten-basiertes WfMS könnte zum Beispiel eine *Entity Bean* `Workflow-Instance` enthalten.

- *Session Beans*
  Die Implementierung der *Business Logic* erfolgt in den *Session Beans*. Zum Instantiieren oder Beenden von Workflow-Instanzen könnte ein WfMS eine *Session Bean* `Process-Manager` anbieten.

Die EJB werden innerhalb des J2EE-Servers in einer Laufzeitumgebung (*EJB-Container*) ausgeführt. Der *EJB-Container* übernimmt die Verwaltung des Lebenszyklus einer EJB (z.B. Erzeugen, Aus-/Einlagern, usw.), die Datenspeicherung und ihre Steuerung (z.B. unter Beachtung von Transaktions- und Nebenläufigkeitseigenschaften).

In mehrschichtigen Architekturen (vgl. Abschnitt 3.3) eignen sich EJB sehr gut zur Implementierung der *Business Logic*. Die klaren Schnittstellendefinitionen einer EJB ermöglichen den leichten Austausch der Komponenten sowie die Inbetriebnahme (*deployment*) in einem *Application Server* eines anderen Herstellers. Zwar bestehen in der Praxis teilweise gewisse Inkompatibiliäten durch propietäre Erweiterungen [GT00], allerdings werden neue Versionen der J2EE-Spezifikation dieses Problem beheben.

4.2.1.2 Java Servlets

Neben dem *EJB-Container* stellen *Web Application Server* auch eine Laufzeitumgebung (*Web Container*) für spezielle Web-Komponenten bereit (vgl. Abbildung 4.1). *Java Servlets* eine Möglichkeit der Unterstützung von Web-basierten Clients dar. Sie basieren auf einem Server-seitigen Komponentenmodell, welches vom Hersteller des *Web Application Server* implementiert wird [Sun01a, HC01, Mon01]. *Servlets* bieten über eine komfortable Java-API unter anderem folgende Funktionen:

- Methoden zur dynamischen Erstellung von (Antwort-)Seiten
- Zugriff auf HTTP-Parameter und Verwaltung von *Cookies*
- automatische Verwaltung einer *Session* mittels verschiedener Verfahren zur *Session-ID* Übermittlung (vgl. Abschnitt 3.4.1)
- Weiterleitung *redirect* zu anderen Web-Seiten

4.2.1.3 Java Server Pages

Die Anwendungsentwicklung mit *Servlets* gestaltet sich recht aufwendig, da wie bei CGI (vgl. Abschnitt 4.1.1) immer die komplette Seite über entsprechende Ausgabefunktionen erstellt werden muß. Eine Trennung zwischen statischen und dynamischen Inhalten gibt es bei *Servlets* nicht. Diesen Nachteil beseitigen *Java Server Pages (JSP)*. JSP erlauben die Einbettung von Java in das HTML-Dokument [Sun01d]. Die JSP wird beim Aufruf (für den Anwender transparent) automatisch in ein *Servlet* übersetzt, kompiliert und anschließend wie ein normales *Servlet* ausgeführt. Durch die Kompilierung ergeben sich im Vergleich zu *Servlets* keine wesentlichen Performanzunterschiede. Praktische Beispiele für den Einsatz von JSP werden in Kapitel 6 und in Abschnitt 7.2.2 vorgestellt.

Die in J2EE enthaltenen Technologien stellen eine sehr gute Unterstützung Web-basierter Anwendungen bereit. Welche Möglichkeiten der IIS anbietet, wird im folgenden Abschnitt geklärt.

4.2.2 Microsoft Internet Information Server

Der *Microsoft Internet Information Server (IIS)* (vgl. Abbildung 4.2) ist die Basis für die Realisierung von Web-Applikationen auf der Architektur der Microsoft Betriebssysteme (*Windows NT* und *Windows 2000*). In den folgenden Abschnitten werden zuerst wichtige Grundlagen für die Anwendungsentwicklung unter Windows vorgestellt und dann die Fähigkeiten des IIS erläutert.

4.2.2.1 Component Object Model

Ein zentrales Element bei der Entwicklung von Anwendungen spielt das *Component Object Model (COM)* [Mic01c, GT00]. Die eigentliche Grundidee von COM war die Realisierung von *Verbunddokumenten* [App99, JBS99], welche durch *OLE (object linking and embedding)* aus Teildokumenten zusammengesetzt werden konnten (z.B. durch die Integration einer Excel-Tabelle in ein Word-Dokument). Da es sich bei COM um eine programmiersprachenunabhängige Spezifikation auf binärer Ebene handelt, wurde COM bald auch zur Entwicklung von Software-Komponenten eingesetzt [Bal01]. Heute basieren große Teile des Betriebssystems und die meisten Anwendungen auf COM. Der einfache Einsatz von COM unter *Visual Basic* [App99] sorgte für eine zusätzliche Verbreitung und die Entwicklung vieler „Standardkomponenten“. Der Zugriff auf relationale Datenbanken kann zum Beispiel realtiv einfach über *ActiveX Data Objects (ADO)* abgewickelt werden. Eine Erweiterung von COM, welche im Web großen Einsatz findet, sind *ActiveX*-Steuerelemente. Sie sind in der Lage, sich beim System selbst zu registrieren, und ermöglichen damit eine einfache Installation [Bal01]. Mit *DCOM (Distributed COM)* wurde COM um den Zugriff auf COM-Komponenten auf entfernten Computer ergänzt, und COM+ unterstützt die besonderen Aspekte von Server-seitigen Komponenten.

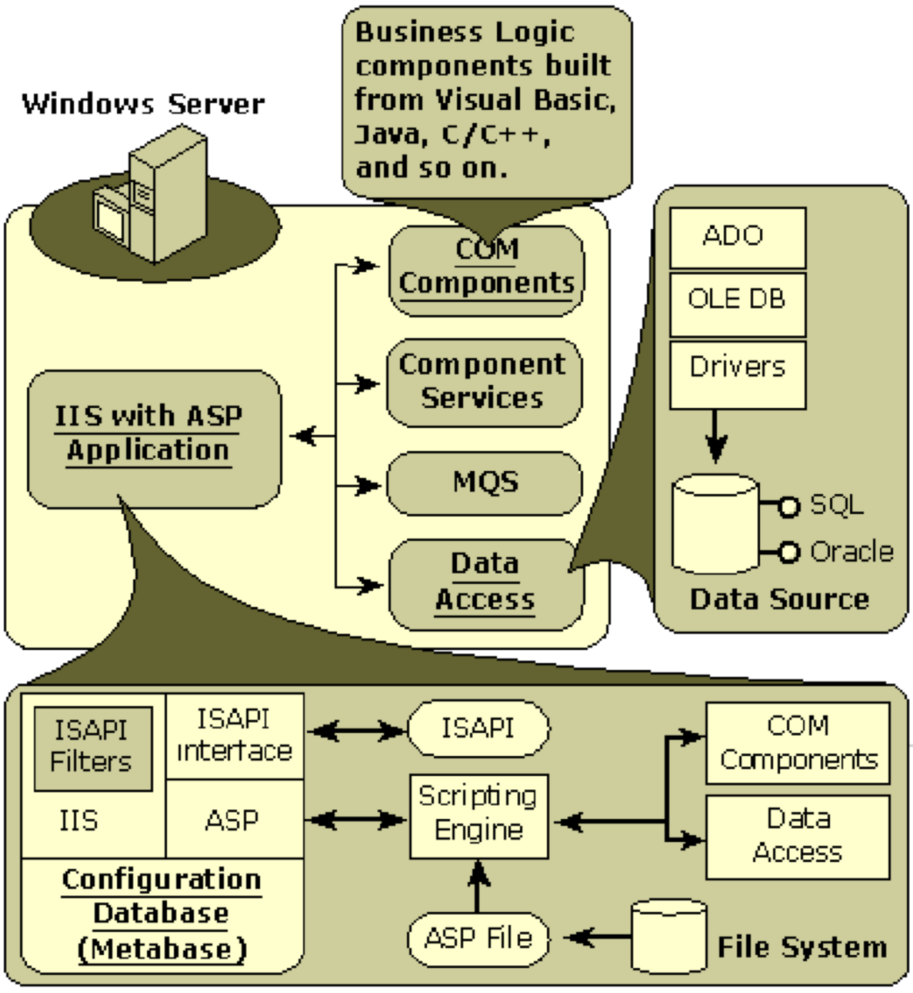

Abbildung 4.2: Architektur des Microsoft Internet Information Servers[Mic01c]

4.2.2.2 Active Server Pages

Für die Erstellung dynamischer Web-Inhalte mit dem IIS wurden die *Active Server Pages (ASP)* entwickelt. Sie werden Server-seitig ausgeführt und erlauben (ähnlich wie JSP) die Integration von Skript-Elementen in das HTML-Dokument.

In der Regel erfolgt die Programmierung der Skripte mit *Visual Basic Script (VBScript)* [App99]. ASP stellt Funktionen zur Generierung von (Antwort-)Seiten, zur Verarbeitung von Formularparametern oder zur Verwaltung einer *Session* bereit. Darübert hinaus ist mit *VBScript* der Zugriff auf COM möglich. Für die Implementierung einer komplexen *Business Logic* für einer Web-Applikation bietet sich die Erstellung einer eigenen COM-Komponente an.

Hier ein kleines Beispiel, welches den praktischen Einsatz von ADO beim Zugriff auf eine Datenbank über SQL und die dynamische Seitenerstellung mit ASP demonstriert:

```
<%@ LANGUAGE = VBScript %>
<HTML>
<HEAD> <TITLE>Simple ADO Query</TITLE> </HEAD>
<BODY BGCOLOR="White" topmargin="10" leftmargin="10">
  ...
  <%
    Dim oConn
    ...
    filePath = Server.MapPath("authors.mdb")
    Set oConn = Server.CreateObject("ADODB.Connection")
    oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" &
               "Data Source=" & filePath
    Set oRs = oConn.Execute("SELECT * From authors")
  %>
  <TABLE border = 1>
    <%
      Do while (Not oRs.eof) %>
      <tr>
        <% For Index=0 to (oRs.fields.count-1) %>
          <TD VAlign=top><% = oRs(Index)%></TD>
        <% Next %>
      </tr>
      <% oRs.MoveNext
         Loop
      %>
   </TABLE>
  ...
</BODY>
</HTML>
```

4.3 Zusammenfassung

Die in Abschnitt 4.1 vorgestellten Techniken zur serverseitigen Unterstützung von Web-Applikationen erscheinen unter Betrachtung der verschiedenen Kriterien unterschiedlich gut geeignet (vgl. Tabelle 4.1).

Hinsichtlich der Anbindung von Datenbanken bieten Perl und PHP zum Beispiel sehr gute Lösungen. Die Integration eines WfMS ist hingegen schwierig, weil deren API meist nur in C++ oder Java zur Verfügung steht, und damit ein direkter Zugriff mit Perl oder PHP nicht möglich ist. Eine WfMS-Anbindung durch CGI ist grundsätzlich fast immer möglich, allerdings bestehen hier die oben genannten Probleme (vgl. Abschnitt 4.1.1).

	CGI	PHP	SSI	Module	Servlet	JSP	ASP
Interoperabilität							
Zugriff auf WfMS	+	○	–	○	○	○	○
Middleware-Unterstützung	–	○	–	○	+	+	○
Dynamische Inhalte							
Performanz	–	+	○	+	+	+	○
Ausgabeformate	–	+	–	–	+	+	+
Cient-Unterstützung	–	○	–	–	+	+	○
Anwendungsentwicklung							
Entwicklungsaufwand	–	○	–	–	○	+	+
Wartungsaufwand	–	+	○	–	○	+	+
benötigtes (Fach-)Wissen	–	○	+	–	–	+	+
Architektur							
Aufbau	–	○	–	○	+	+	+
Skalierbarbarkeit	–	○	–	○	+	+	+
Zusatzdienste	○	+	–	–	+	+	+
Offenheit							
offener Standard	+	+	–	–	+	+	–
Plattformunabhägigkeit	+	○	–	–	+	+	–

Erläuterung: + = gut, ○ = neutral, – = schlecht

Tabelle 4.1: Bewertung von Server-Techniken

Server Side Includes sind nur für sehr einfache Aufgaben (z.B. Einfügen von Inhalten) zu gebrauchen. Komplexere Programmieraufgaben, wie die Implementierung einer Web-Schnittstelle, lassen sich mit ihnen nicht bewältigen. Web-Server-Module ermöglichen eine effektive Programmierung und bieten eine gute Integration in den Web-Server, was vor allem bei der Performanz deutlich wird. Hinsichtlich Portabilität bestehen jedoch starke Einschränkungen durch die jeweilige API des Server-Herstellers.

Die beste Unterstützung von Web-Applikationen wird durch die in Abschnitt 4.2 vorgestellten Architekturen J2EE und IIS erreicht. Sie bieten beide mit ihren Technologien *Java Servlets* und *Java Server Pages* bzw. *Active Server Pages* eine gute Basis für die Realisierung Web-basierter WfMS. Welche Architektur präferiert wird, hängt in erster Linie von der bereits vorhandene EDV-Infrastruktur des Unternehmens bzw. den Schnittstellen des WfMS ab.

Kapitel 5

Web-Clients

Einer der großen Vorteile von Web-basierten Applikationen ist die Verfügbarkeit für viele verschiedene Clients und Systemplattformen. Trotz einheitlicher Protokolle (z.B. HTTP) und standardisierter Beschreibungssprachen (z.B. HTML) bestehen zwischen den Web-Clients teilweise recht große Unterschiede. Diese sollen anhand folgender Kriterien untersucht werden:

- Darstellung
  - Text- bzw. Grafikdarstellung
  - Flexibilität der GUI
- Funktionalität
  - Implementierung Client-seitiger Logik
  - Kommunikationsmöglichkeiten (z.B. Verbindungsaufbau zum WfMS)
- Software-Ergonomie
  - Anpaßbarkeit der Bedienungsabläufe
  - Konfigurationsmöglichkeiten für ein individuelles *Look-And-Feel*
- Offenheit
  - Plattformunabhängigkeit
  - Kompatibilität zu Standards

Eine exakte Einteilung der verschiedenen Arten von Web-Clients ist schwierig, da es keine genauen Definitionen für Begriffe wie *Thin* oder *Thick Web-Client* gibt. In Tabelle 5.1 ist eine denkbare Untergliederung des Begriffs *Web-Client* dargestellt. Anhand dieser Untergliederung wird in den folgenden Abschnitten eine Betrachtung der verschiedenen Arten von Web-Clients und eine Bewertung gemäß den oben aufgeführten

Kriterien durchgeführt. Außerdem wird bei jedem Client-Typ ein praktisches Beispiel für einen möglichen Einsatzzweck gegeben.

Kategorie	**Web-Client**					
Unterkategorie	Ultra-Thin Web-Client	Thin Web-Client	Thick Web-Client			
mögl. Technologie	WML, VoiceXML	HTML	DHTML	HTML + ActiveX	HTML + Applet	HTML + Plug-In
mögl. Einsatzort	Mobiltelefon	PDA	PC	Windows-PC	PC	PC

Tabelle 5.1: Übersicht der verschiedenen Arten von Web-Clients

5.1 Thin Web-Client

Applikationen lassen sich in verschiedene Funktionsbestandteile zerlegen (vgl. Abschnitt 3.3). Ist im Client nur die *Presentation Logic* implementiert, spricht man von einem *Thin Web-Client*. Die *Business Logic* und die *Data Logic* werden komplett im Server ausgeführt. Deshalb sind die Systemanforderungen an den Client (z.B. bezüglich Rechenleistung) sehr niedrig. *Thin Web-Client* eignen sich daher auch für den Einsatz auf weniger leistungsstarke Endgeräte (z.B. einem PDA).

Thin Web-Clients sind in der Regel für die Darstellung von HTML konzipiert und bieten darüber hinaus wenig erweiterte Möglichkeiten (z.B. benutzerdefinierte GUI).

In Bezug auf ihre Funktionalität weisen *Thin Web-Clients* im Vergleich zu konventionellen Workflow-Clients (vgl. Abschnitt 3.5) große Einschränkungen auf. Selbst einfache Aufgaben, wie die Überprüfung eines (HTML-)Formularfeldes auf ein korrektes Datenformat, können nicht im Client durchgeführt werden. Um diese Funktionalität zu implementieren, müssen die Formulardaten zum Server geschickt werden, dieser überprüft sie und generiert ggf. eine Antwortseite mit entsprechenden Fehlermeldungen.

Thin Web-Clients bieten unter dem Gesichtspunkt der Software-Ergonomie ebenfalls nur beschränkte Fähigkeiten. Das Verhalten und das Aussehen sind durch den jeweiligen Client ziemlich starr vorgegeben. Die Reaktion auf Ereignisse oder Konfiguration für ein individuelles *Look-And-Feel*) sind nicht möglich. Trotzdem reicht die Funktionalität der *Thin Web-Clients* für viele Einsatzzwecke vollkommen aus.

Beispiel: Ein typisches Beispiel für den Einsatz eines *Thin Web-Clients* ist die Genehmigung eines Urlaubsantrags. Der Mitarbeiter startet einen Workflow und trägt in der ersten WF-Aktivität den gewünschten Zeitraum ein. Für die Erfassung der Daten (Name, Urlaubsbeginn und -ende) reichen einfache Textfelder aus, und die Übertragung ist

als String möglich. Nach Abschluß der ersten Aktivität bekommt der Vorgesetzte den Urlaubsantrag zur Genehmigung in seiner Arbeitsliste angezeigt. Auch hier lassen sich alle Informationen ohne eine komplexe GUI mit reinem HTML darstellen. Auch die Ausgabeparameter („Ja“ oder „Nein“) sind ebenfalls wieder gut als String übertragbar.

Die fehlende Möglichkeit der Ausführung von Programmen ist einerseits ein Nachteil, andererseits macht sie *Thin Web-Clients* auch zu einem sehr sicheren Client. Es können keine Programme unberechtigt auf sicherheitskritische Bereiche des (Client-)Rechners zugreifen und es bestehen auch keine potentiellen Fehlerquellen (z.B. „Sicherheitslöcher“ in der *Java Virtual Machine*).

5.2 Thick Web-Client

Für viele Einsatzzwecke reicht die Funktionalität der Thin Web-Clients nicht aus, oder die Ausführung auf Serverseite besitzt gravierende Nachteile (z.B. häufiger Verbindungsaufbau zum Server, hohes Datentransfervolumen, usw.). Thick Web-Clients stellen eine Erweiterung der Thin Web-Clients dar. Sie beinhalten neben der *Presentation Logic* auch die *Business Logic* (bzw. Teile davon). Die *Business Logic* kann durch zwei unterschiedliche Verfahren implementiert werden:

- durch Skriptsprachen (z.B. *Javascript* bei *Dynamic HTML*)
- durch Integration von compilierten Programmbausteinen (z.B. *Java Applets*, *ActiveX*-Steuerelemente oder *Plug-Ins*).

Die einzelnen Verfahren werden in den folgenden Abschnitten vorgestellt.

5.2.1 Dynamic HTML

Die Darstellungsmöglichkeiten von *Dynamic HTML (DHTML)* sind, wie beim *Thin Web-Client*, auf HTML beschränkt. Allerdings integriert DHTML in den bisher statische HTML-Dokument dynamische Eigenschaften [Goo98, Mic01b, MN98, HL01]. DHTML erlaubt die Programmierung durch Skriptsprachen (in der Regel *Javascript*) sowie die Reaktion auf Ergeignisse (z.B. beim Ändern eines Eingabefeldes). Zudem ermöglicht DHTML den Zugriff auf die Elemente eines HTML-Dokuments über das *Document Object Model (DOM)* [Wor01c, Goo98]. Das DOM bildet die Struktur eines HTML-Dokuments in Form einer Objekthierachie ab. Über das DOM können zum Beispiel Werte aus einem Formular gelesen werden oder es können Veränderungen am Dokument selbst vorgenommen werden (z.B. Änderung der Hintergrundfarbe).

Beispiel: Die Vorteile von DHTML sollen an folgendem Beispiel verdeutlicht werden:

```
<html>
<head>
<script language="JavaScript">
<!--
  function checkText(text) {
    if (text.value == "") {
      alert("Fehler in Text!");
      text.focus(); return false;
    } else return true;
  }
  function checkEmail(email) {
    validEmail = /^\w+\@\w+(\.\w+)+$/;
    result = validEmail.test(email.value);
    if (result == false) {
      alert("Fehler in E-Mailadresse!");
      email.focus(); return false;
    } else return true;
  }
  function checkInteger(integer) {
    validInteger = /^\d+$/;
    result = validInteger.test(integer.value);
    if (result == false) {
      alert("Fehler in Ganzzahl!");
      integer.focus(); return false;
    } else return true;
  }
  function checkForm() {
    if(checkText(document.Formular.User) &&
       checkText(document.Formular.Ort) &&
       checkEmail(document.Formular.Mail) &&
       checkInteger(document.Formular.Alter)) return true;
    return false;
  }
//-->
</script>
</head>
<body>
<form name="Formular" action="mailto:test@ralf-kiessling.de"
      method=post onSubmit="return checkForm()">
<pre>
Name:     <input type="text" name="User"  onChange="checkText(this)">
Wohnort:  <input type="text" name="Ort"   onChange="checkText(this)">
E-Mail:   <input type="text" name="Mail"  onChange="checkEmail(this)">
Alter:    <input type="text" name="Alter" onChange="checkInteger(this)">
</pre>
<input type=submit value="Absenden">
</form>
</body>
</html>
```

Bei Änderungen (`onChange`) in den Formulareingabefeldern und beim Absenden der Formulardaten (`onSubmit`) werden Funktionen zur Überprüfung der Eingabe gerufen (z.B. `checkEmail`). Beim Auftreten von Fehlern werden Hinweismeldungen ausgegeben und das Absenden der fehlerhaften oder unvollständigen Daten zum Server verhindert. Dadurch wird zum einen der Server entlastet, und der Benutzer bekommt zum anderen eine schnellere Rückmeldung bei Fehlern.

Fehlende und nicht eingehaltene Standards machen die Entwicklung von Browser-unabhängigem DHTML jedoch zu einem großen Problem [Sch00]. Der „Browser-Krieg“ zwischen verschiedenen Herstellern, welche versuchen ihren propietären Standard durchzusetzen, macht die Situation noch schwieriger [Per00].

5.2.2 Java Applets

Zwar läßt sich mit DHTML dynamisches Verhalten in ein HTML-Dokument integrieren, allerdings sind die Möglichkeiten der Programmierung mit *Javascript* beschränkt.

Java Applets sind kleine Java-Programme, welche durch ein spezielles Tag (`<applet>`) in ein HTML-Dokument integriert werden können [Sun01c, MN98]:

```
<html>
...
<applet code="TextBanner.class" width="200" height="100">
  <param name="text" value="Herzlich Willkommen">
  <h1> Ihr Browser unterstützt keine Applets :-( </h1>
</applet>
...
</html>
```

Bei der Darstellung der HTML-Seite lädt der Browser (*Code-On-Demand*) die Klasse des *Java Applets* (`TextBanner.class`) und führt sie in einer speziellen Laufzeitumgebung (*Java Virtual Machine, JVM*) aus. Das *Java Applet* besitzt eine eigene GUI, welche in die Darstellung der (restlichen) Seite integriert wird.

Das *Java Applet* wird in der JVM aus Sicherheitsgründen mit gewissen Einschränkungen in einer sog. *Sandbox* ausgeführt. So kann es zum Beispiel nicht auf das lokale System zugreifen und auch nur Verbindungen zu dem Web-Server aufbauen, von dem es geladen wurde.

Beispiel: Die oben genannten Sicherheitseinschränkungen bringen ein Problem mit sich, wenn das *Java Applet* eine Verbindung zum Workflow-Server aufbauen möchte, dieser aber auf einem anderen Rechner als der Web-Server läuft. Um dies zu ermöglichen, ist zuerst eine Anpassung der Einstellungen (*Java Policies*) nötig. Das *Java Applet* kann bei einem Web-Client zum Beispiel die komfortable Darstellung der Arbeitsliste übernehmen oder es dient als *Trigger* und löst bei Veränderung in der Arbeitsliste ein Update der HTML-Seite aus.

5.2.3 ActiveX-Steuerelemente

ActiveX-Steuerelemente (vgl. Abschnitt 4.2.2.1) werden ähnlich wie *Java Applets* in das HTML-Dokument eingebunden:

```
<html>
...
<object id="Calendar"
        classid="CLSID:8E27C92B-1264-101C-8A2F-040224009C02">
</object>
...
</html>
```

Sie besitzen eine eigene GUI und werden im HTML-Dokument integriert dargestellt. Im Gegensatz zu *Java Applets* werden sie nicht bei jedem Seitenaufruf neu geladen, sondern installieren sich (nach Rückfrage) beim ersten Aufruf dauerhaft auf dem Clientrechner. Sie unterliegen während der Ausführung keinen speziellen Sicherheitseinschränkungen.

Beispiel: Neben der Möglichkeit neue *ActiveX-Steuerelemente* zu installieren, hat die Anwendung auch Zugriff auf alle bereits auf dem Rechner vorhandenen. Zur komfortablen Anzeige eines Kalenders kann zum Beispiel das Windows Kalender-Steuerelement benutzt werden.

5.2.4 Plug-Ins

Eine weitere Möglichkeit, die Fähigkeiten des Web-Browsers zu erweitern, bilden *Plug-Ins* [Net01, PI01]. Hierbei handelt es sich um kleine Zusatzprogramme, welche die Darstellung bestimmter Web-Inhalte ermöglichen. Einige häufig eingesetzte *Plug-Ins* dienen:

- zur Anzeige von Dokumenten (z.B. für das *Portable Document Format (PDF)* [Ado01])
- zur Ausgabe von Animationen (z.B. *Shockwave Flash* [Mac01])
- dem Abspielen von Musik oder Videos (z.B. *RealPlayer* [Rea01] oder *QuickTime* [App01])

Die oben genannten *Plug-Ins* sind in der Regel schon im Lieferumfang des Browsers enthalten. Darüber hinaus existieren jedoch noch viele weitere *Plug-Ins* für spezielle Datenformate, welche über das Internet bezogen [PI01] und nachträglich installiert werden können.

Beispiel: Mit *Shockwave Flash* ist es möglich, kleine Animationen mit Benutzerinteraktion im Web-Client auszuführen [Mac01]. Darüber hinaus gibt es Techniken *Flash*-Inhalte dynamisch auf dem Server zu generieren. Auf diese Weise ist es zum Beispiel möglich, ein *Flash*-Element auf dem Server zu erstellen, um damit einen Prozeßgraph im Web-Client komfortabel visualisieren zu können.

5.3 Ultra-Thin Web-Client

Ultra-Thin Web-Clients stellen im Vergleich zu *Thin Web-Clients* eine noch stärke Funktionsreduzierung dar. Sie besitzen in der Regel eine stark eingeschränkte *Presentation Logic* und ermöglich daher oft nur eine teilweise Darstellung von Web-Inhalten (z.B. ohne Tabellen oder *Frames*). Zur Beschreibung von Inhalten für *Ultra-Thin Web-Clients* wurden spezielle Sprachen wie WML (vgl. 3.2.3) entwickelt.

Die Funktionalität ist ebenfalls sehr begrenzt. Die Kommunikation zwischen *Ultra-Thin Web-Client* und WfMS kann sehr unterschiedlich ausfallen. Dies hängt vor allem auch an den verschiedenen Protokollen (z.B. HTTP, WAP, ...), Übertragungskanälen (z.B. Internet, Funknetz, ...) und der Schwierigkeit nur über spezielle Knoten (*Gateways*) kommunizieren zu können. Daher ist die Interaktion mit dem Web-Server wesentlich schwieriger bzw. teilweise gar nicht möglich (z.B. keine Rücksendung von Daten wie bei Formularen).

Beispiel: Typische *Ultra-Thin Web-Clients* sind Mobiltelefone. Sie eignen sich nicht für eine „sinnvolle“ Bearbeitung von WF-Aktivitäten, aber ihre Darstellungsmöglichkeiten reichen aus, um sie zum Beispiel als einfacher *Monitoring Client* einsetzen zu können. Außerdem können Textnachrichten (*Short Message Service, SMS*) vom WfMS dazu verwendet werden, Warnungen beim Überschreiten von Deadlines zu versenden.

5.4 Zusammenfassung

Zwischen den vorgestellten Arten von Web-Clients bestehen große Unterschiede bezüglich Darstellungsvermögen und Funktionalität (vgl. Tablle 5.2). Dies ist jedoch nicht verwunderlich, wenn man bedenkt, daß sich das Spektrum der Systemplattformen von einem Mobiltelefon bis zu einem PC erstreckt. Deshalb ist ein direkter Vergleich zwischen einem *Ultra-Thin Web-Client* und einem *Thick Web-Client* auch nur wenig sinnvoll.

Kategorie	**Web-Client**					
Unterkategorie	**Ultra-Thin Web-Client**	**Thin Web-Client**	**Thick Web-Client**			
			DHTML	HTML + ActiveX	HTML + Applet	HTML + Plug-In
Darstellung	–	○	○	+	+	+
Funktionalität	–	○	+	+	+	○
Software-Ergonomie	–	○	+	+	+	○
Offenheit	○	+	○	–	+	○

Erläuterung: + = gut, ○ = neutral, – = schlecht

Tabelle 5.2: Bewertung der verschiedenen Arten von Web-Clients

Kapitel 6

Beispiel-Implementierung

Die Beispiel-Implementierung basiert auf dem *ADEPT* Workflow-Management-System (vgl. Abschnitt 7.1.4) und folgenden Web-Technologien:

- *Java Server Pages (JSP)* (vgl. Abschnitt 4.2.1.3)
- *Extensible Markup Language (XML)* (vgl. Abschnitt 3.2.2)
- *Extensible Style Language Transformation (XSLT)* (vgl. Abschnitt 3.2.2.2)

Neben der Realisierung eines Web-Clients (vgl. Abschnitt 6.3) war die Client-unabhängige Formularbeschreibung ein weiterer Aspekt der Beispiel-Implementierung. In Abschnitt 6.1 werden die in diesem Zusammenhang bestehenden Probleme und in Abschnitt 6.2 eine Lösungsvariante vorgestellt.

6.1 Motivation

In der Regel adressieren Web-basierte WfMS nicht nur einen ganz bestimmten Typ eines Web-Clients (z.B. den Web-Browser eines Herstellers in einer Version), sondern müssen beim unternehmensweiten bzw. -übergreifenden Einsatz mit einer Vielzahl unterschiedlicher Web-Clients zusammenarbeiten können.

Zum einen basieren die Web-Clients teilweise auf unterschiedlichen Seitenbeschreibungssprachen (z.B. HTML, WML, ...) und zum anderen gibt es sehr große Unterschiede in Bezug auf die unterstützten Technologien (*ActiveX-Steuerelemente* können z.B. nur im *Microsoft Internet Explorer* verwendet werden). Aber auch Standards, wie zum das *Document Object Model (DOM)* [Wor01c], werden von den Herstellern der Web-Clients unterschiedlich implementiert [KS01].

Um den Web-Client adäquat zu unterstützen und seinen vollen Funktionsumfang ausschöpfen zu können, müssen in der Regel für jeden Web-Client „maßgeschneiderte“ Inhalte erstellt werden.

Denkbare Formate bei der Unterstützung verschiedene Web-Clients sind:

- HTML
- HTML (mit *Java Applets* oder *ActiveX-Steuerelementen*)
- DHTML (für den *Netscape Navigator* oder den *Microsoft Internet Explorer*)
- WML
- XML

Für eine sehr kleine Anzahl von Web-Clients und eine überschaubare Menge an Web-Inhalten ist diese Anpassung vielleicht noch manuell durchführbar. Für den breiten Einsatz (z.B. im Zusammenhang mit WfMS) sind Verfahren zur automatischen Generierung unterschiedlicher Versionen nötig. Für die Erstellung verschiedener Dokumentformate mit JSP gibt es zwei unterschiedliche Ansätze [Sun00]:

- *Single Pipeline*
  Beim *Single Pipeline*-Ansatz (vgl. Abbildung 6.1) wird die (XML-)Ausgabe von einer einzigen JSP erzeugt, und dann durch XSL-Stylesheets für den jeweiligen Web-Client aufbereitet.

- *Multiple Pipeline*
  Der *Multiple Pipeline*-Ansatz (vgl. Abbildung 6.2) besitzt für jede Art von Web-Client eine eigene JSP, welche direkt die Ausgabe in HTML, WML, usw. generiert.

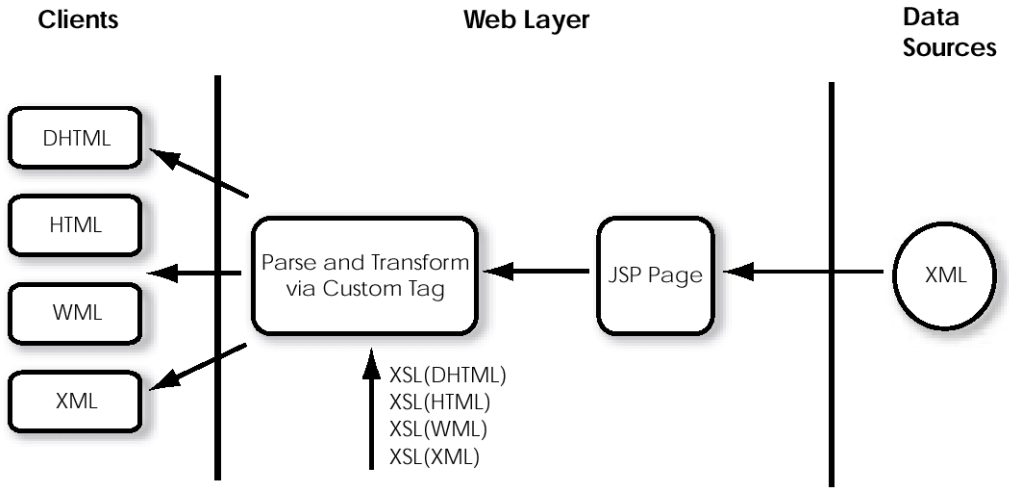

Abbildung 6.1: Single Pipeline Generating Multiple Markup Languages [Sun00]

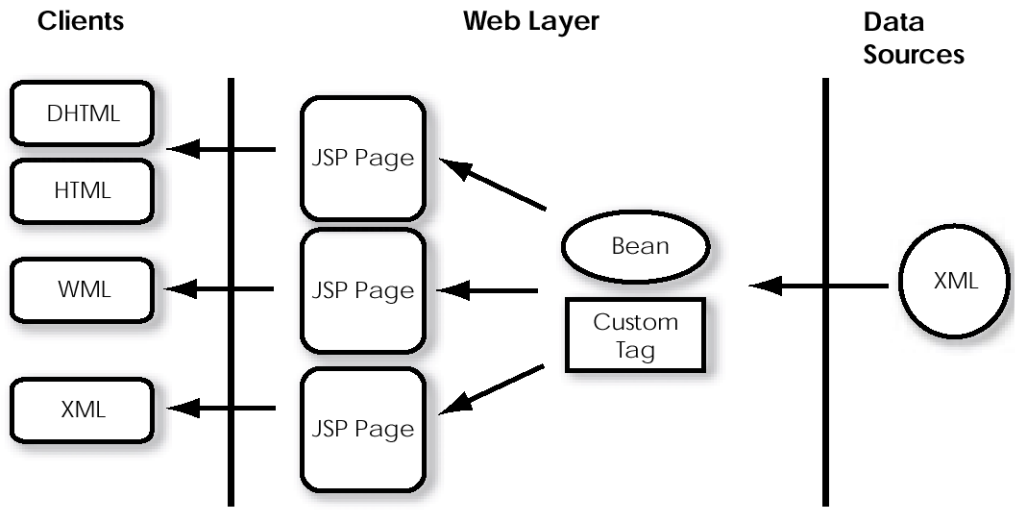

Abbildung 6.2: Multiple Pipelines Generating Multiple Markup Languages [Sun00]

6.2 Client-unabhängige Formularbeschreibung

Die Beschreibung der Formulare erfolgt Client-unabhängig in XML (vgl. Abschnitt 3.2.2). Zum einen unterstützt XML sehr gut den strukturierte Aufbau von Formularen, und zum anderen bieten JSP bereits eine Unterstützung bei der Verarbeitung von XML (z.B. durch XSLT).

Hier ein kleines Beispiel für eine Formularbeschreibung:

```
<page>
<form>
<table>
  <row>
    <cell>
      <label>Vorname</label>
    </cell>
    <cell>
      <workflow:includeParameter name="vorname" />
    </cell>
  </row>
  <row>
    <cell>
      <label>Nachname</label>
    </cell>
    <cell>
      <parameter>
        <name>Nachname</name>
        <value><workflow:getParameterValue name="nachname" /></value>
        <type>String</type>
      </parameter>
    </cell>
```

```
  </row>
  <row>
    <cell>
      <label>Geburtsdatum</label>
    </cell>
    <cell>
      <parameter>
        <name>Geburtsdatum</name>
        <value><workflow:getParameterValue name="geburtsdatum" /></value>
        <type>Date</type>
      </parameter>
    </cell>
  </row>
</table>
</form>
</page>
```

Der Aufbau des Formulars ist durch die einfache Namenswahl der Elemente (z.B. `<form>`, `<table>`, `<row>`, usw.) selbsterklärend. Eine Besonderheit stellt das Element `<paramter>` dar. Es enthält mit seinen Subelementen `<name>`, `<value>` und `<type>` alle Informationen, welche zur Darstellung bzw. zur Eingabe des Parameters benötigt werden. Mit diesen Informationen kann zum Beispiel ein Eingabefeld in einem HTML-Formular erzeugt werden. Der Wert der Parameter wird durch die `workflow`-*Taglib* eingebunden (vgl. Abschnitt 6.3).

Aus den XML-Formularbeschreibungen können über verschiedene XSL-Stylesheets die Dokumentformate für die unterschiedlichen Web-Clients generiert werden. Dabei wird auch der mit `<type>` angegebene Typ des Parameters berücksichtigt. Für den Typ `Date` gibt es zum Beispiel folgende drei Darstellungsmöglichkeiten:

- als einfaches HTML-Textfeld
  `<input type="text" />`

- als *ActiveX*-Kalendersteuerelement
  `<object id="Calendar" classid="CLSID:8E27C92B-1264-101C-8A2F-040224009C02">`

- als *Java Applet*
  `<applet codebase="." code="DateApplet.class" name="Calendar" ...>`

Abhängig von den Fähigkeiten des Web-Client wird eine der drei Alternativen gewählt. Beim *ActiveX*-Kalendersteuerelement und beim *Java Applet* wird zusätzlich noch *Javascript*-Code eingefügt, welcher die Werte aus dem Steuerelement bzw. Applet liest und in ein verstecktes Eingabefeld schreibt.

6.3 Architekur

Nachdem in Abschnitt 6.2 die Formularbeschreibungssprache erläutert wurde, soll hier die Architektur des Web-Client vorgestellt werden (vgl. Abbildung 6.3).

Jede Browser-Anfrage gelangt zuerst zu der zentralen JSP `index.jsp`, welche als *Dispatcher* eingesetzt ist. Hat sich der Benutzer noch nicht angemeldet, d.h. das `WorkflowServer`-Objekt existiert noch nicht, erfolgt eine Weiterleitung zur Seite `login.jsp`. Nach der Anmeldung wird die Art des Web-Client ausgelesen und der Anwender auf die individuelle Startseite für seinen Client weitergeleitet.

Bei der Ausführung von WF-Applikationen leitet der *Dispatcher* den Anwender zu einer JSP weiter, welche eine XML-Formularbeschreibung enthält. Die JSP wird ausgeführt und liest über die `workflow`-*Taglib*, welche über das `WorkflowServer`-Objekt mit *ADEPT* kommuniziert, die Eingabeparameter aus. Sie werden in die Formularbeschreibung eingefügt `<value>...</value>` und im Anschluß daran durch ein XSL-Stylesheet für den jeweiligen Web-Client formatiert.

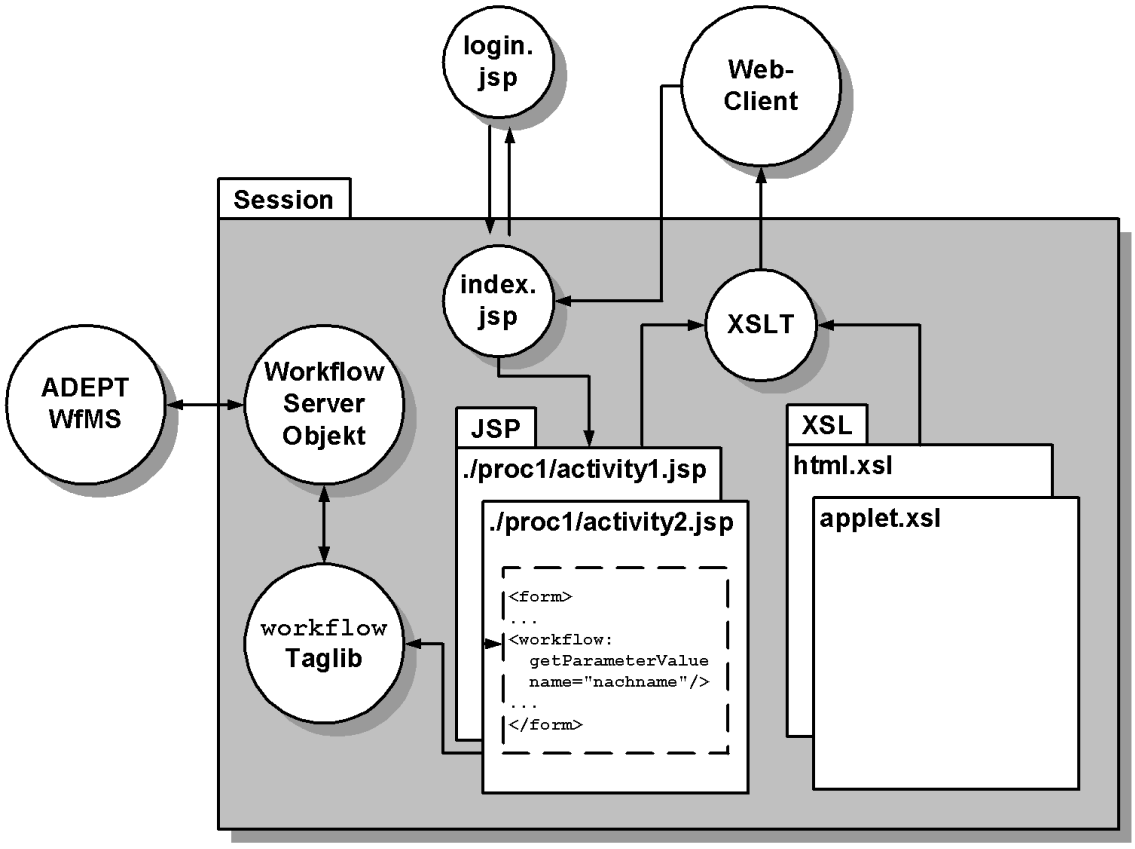

Abbildung 6.3: Architektur der Beispiel-Implementierung

Kapitel 7

Web-Anbindung in existierenden Workflow-Management-Systemen

Die Bedeutung des Web als Schnittstelle für WfMS wird immer größer. Neben einigen Forschungsprojekten bieten fast alle namhaften WfMS-Hersteller bereits Web-Clients an oder arbeiten an deren Entwicklung. Dieses Kapitel beschäftigt sich mit der Architektur von Web-Clients in existierenden WfMS.

7.1 Lösungen in prototypischen Systemen

Stellvertretend für die vielen Forschungsprojekte, welche sich mit der Thematik von Web-basiertem Workflow-Management beschäftigen, sollen hier vier Systeme, welche in Form von Prototypen implementiert wurden, exemplarisch vorgestellt werden: *Panta Rhei*, *WebFlow*, *WW-Flow* und *ADEPT*.

7.1.1 Panta Rhei

Der an der Universität Klagenfurt entwickelte WfMS-Prototyp *Panta Rhei* [EGL98] war eines der ersten Systeme mit einer Web-basierten Benutzerschnittstelle. Über diese werden alle Interaktionen mit dem WfMS durchgeführt. Der HTTP-Server bildet die Schnittstelle des WfMS zum Web-Client oder anderen Systemen (vgl. Abbildung 7.1). Er interpretiert (Client-)HTTP-Anfragen und ruft über die CGI-Schnittstelle die entsprechenden Prozeduren des WfMS. Die Datenbank speichert Workflow-relevante Daten (Prozeß- und Organisationsmodelle, Prozeßinstanzdaten, usw.) persistent.

Bei der Abfrage der Arbeitsliste eines Benutzers liest zum Beispiel eine Prozedur die Einträge direkt aus der Datenbank und stellt sie dynamisch als HTML-Seite dar (vgl. Abbildung 7.2 rechts). Andere Interaktionen, wie das Starten oder Beenden einer Aktivität, bewirken den Aufruf von Funktionen der Workflow Engine.

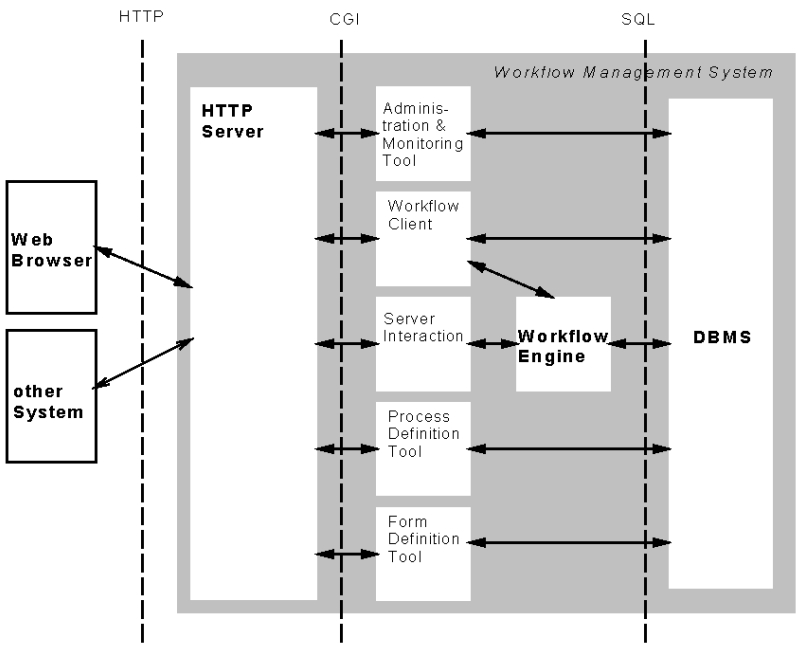

Abbildung 7.1: Panta Rhei Architektur [EGL98]

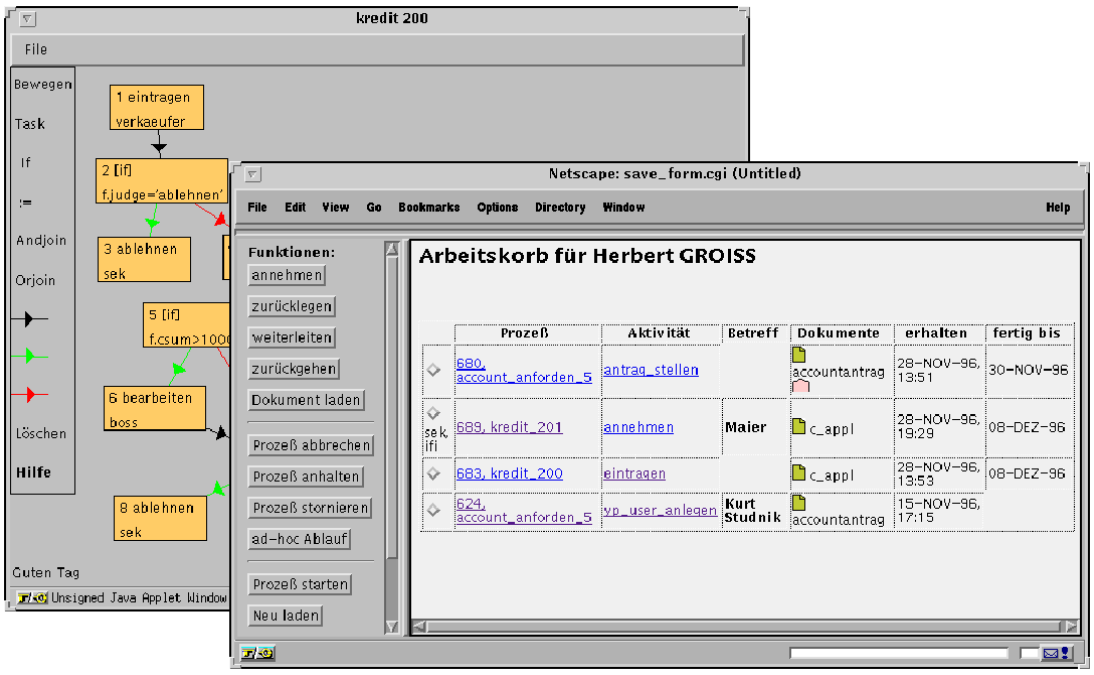

Abbildung 7.2: Arbeitsliste (links) und *Process Designer* (rechts) von Panta Rhei [EGL98]

Sie interpretiert die Prozeßmodelle und beinhaltet Transaktions- und Zeitkomponenten. Die Bearbeitung von Aktivitäten erfolgt Client-seitig durch HTML-Formulare. Eingabeüberprüfungen und einfache Berechnungen können dabei mittels JavaScript implemeniert und direkt im Browser ausgeführt werden (vgl. Abschnitt 5.2.1). Da „reines“ HTML jedoch nur begrenzte Möglichkeiten zur Darstellung bietet, sind komplexe Werkzeuge (z.B. *Process Definition Tool*) als Java-Applet realisiert. Sie werden im Web-Browser ausgeführt (vgl. Abbildung 7.2 links) und kommunizieren über JDBC direkt mit der Datenbank. Die strikte Trennung der Komponenten durch Standardschnittstellen, wie HTTP, CGI und SQL, erlaubt die leichte Austauschbarkeit der Komponenten und Portierbarkeit auf verschiedene Server und Clients.

7.1.2 WebFlow

Der WfMS-Prototyp *WebFlow* entstand 1997 im Rahmen einer Diplomarbeit an der Universität Ulm [Ill97, WIS98, WI]. Ziel war die Entwicklung eines Agenten-basierten WfMS mit Java-Applets, welche die Verwirklichung von Ideen wie *Code-On-Demand*, Plattformunabhängigkeit (durch die Verwendung von Java) sowie eigenverantwortlichen und dezentralen Workflows, welche als Applet im Browser ablaufen, erlauben. Bei *Code-On-Demand* wird das Programm, welches ausgeführt werden soll nicht lokal installiert, sondern bei Bedarf (dyamisch) geladen. Die Java-Applet Technolgie unterstützt diesen Ansatz sehr gut, weil der Browser die zur Ausführung der Workflow-Applikation benötigten Klassen vom Web-Server lädt.

Für die Realisierung wurde die in Abbildung 7.3 dargestellte Architektur gewählt. Sie besteht im wesentlichen aus einem Java-/Applet-fähigen Web-Browser, einer JDBC-Datenbank und einem Web-Server. Zwischen Web-Client und Web-Server ist die *WebFlow Engine* geschaltet. Sie prüft alle vom Client eingehenden HTTP-Requests und leitet diese entweder an den Web-Server weiter oder bearbeitet sie selbst.

Die Benutzerauthentifizierung, das dynamische Erstellen von Arbeitslisten, die Übertragung der auszuführenden Workflow-Applikation zum Klienten (incl. Workflowdaten, zugehörigem (HTML-)Dokumenten und Applikation (Java-Applet)) sowie die Überwachung des Datenflusses und zeitlicher Einschränkungen (z.B. Deadlines) gehören zu den Aufgaben der *WebFlow Engine*. Die Integration der *Workflow Engine* in die Gesamtarchitektur gleicht der Java-Servlet-Technologie (vgl. Abschnitt 4.2.1.2), welche zum damaligen Zeitpunkt allerdings noch nicht verfügbar war.

Die Workflow Applikationen werden als Java-Applet in den Browser geladen und darin ausgeführt. Sie greifen dabei direkt auf die Datenbank zu. Administrative Aufgaben, wie das Anlegen von Benutzern, sind ebenfalls als Workflow-Applikation realisiert. Sie arbeiten dann direkt auf den Tabellen, welche für die Benutzerverwaltung reserviert sind.

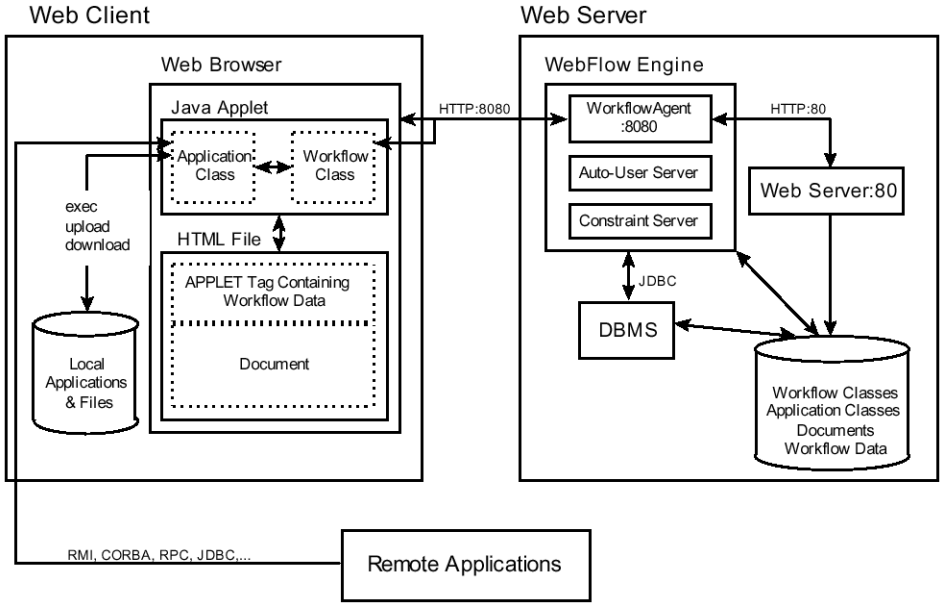

Abbildung 7.3: WebFlow Architektur [Ill97]

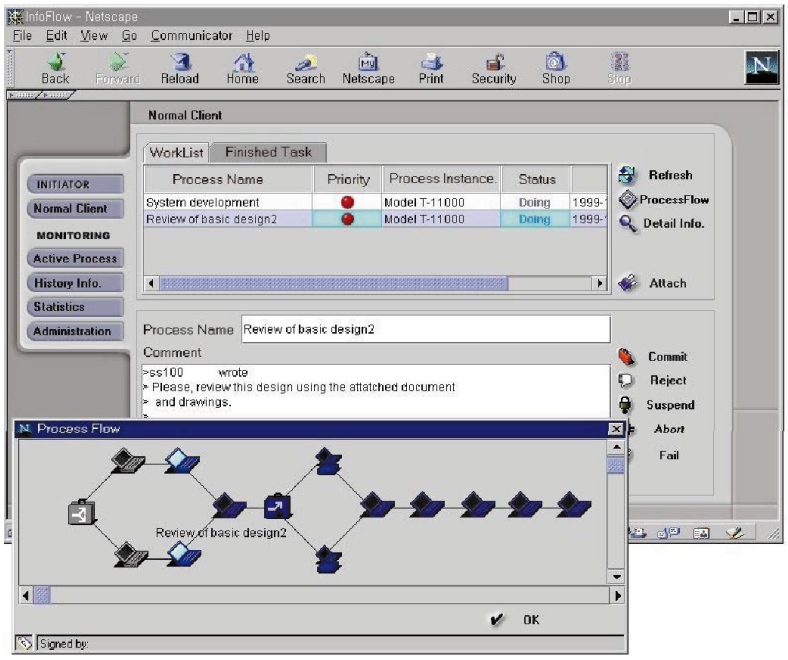

Abbildung 7.4: WW-Flow Normal Client [KKK+00]

7.1.3 WW-Flow

Der an der *Seoul National University* entwickelte WfMS-Prototyp *WW-Flow* [KKK+00] erlaubt die Abbildung geschachtelter Workflows, die verteilt von mehreren *Workflow Engines* ausgeführt werden können. Zur Laufzeit unterstützt *WW-Flow* ein flexibles Workflow-Management. Abbildung 7.5 zeigt die Architektur einer *WW-Flow*-Konfiguration mit zwei *Workflow Engines* und mehreren Clients.

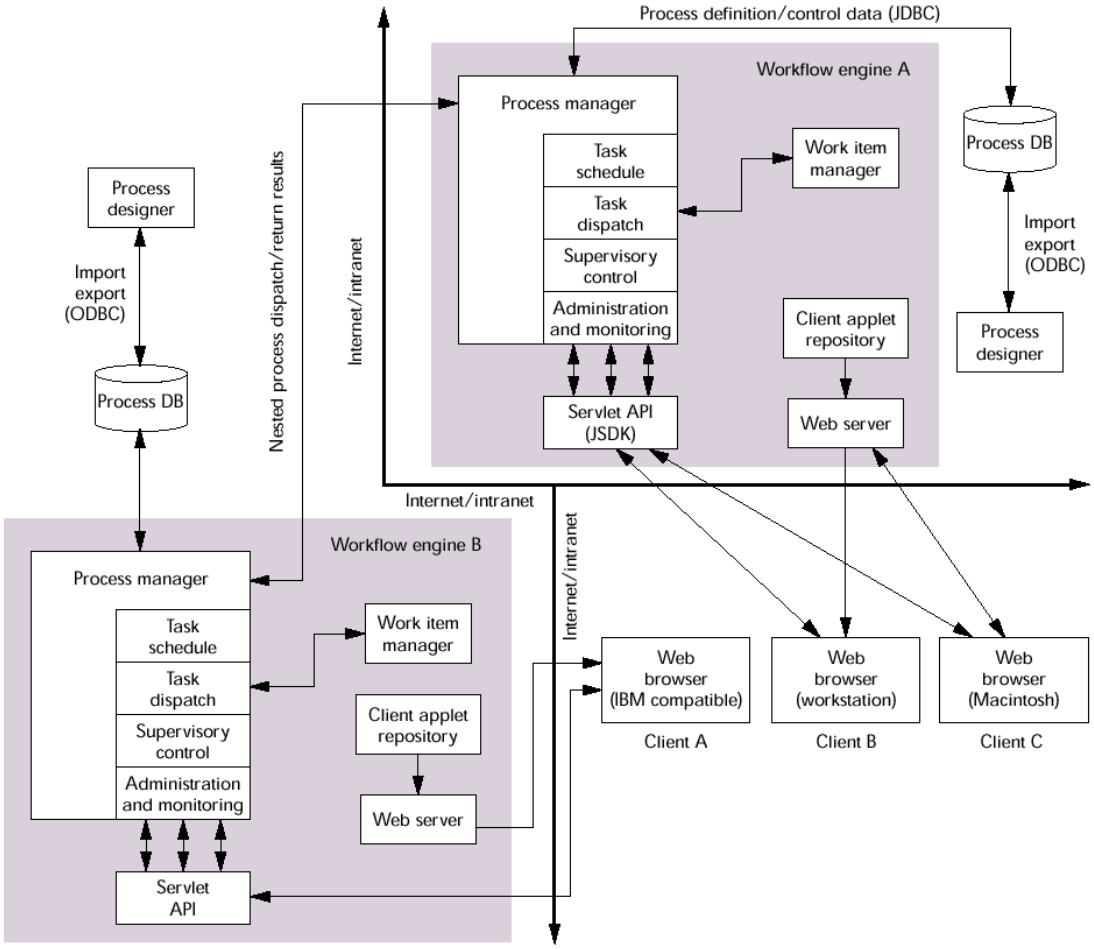

Abbildung 7.5: WW-Flow Gesamtarchitektur [KKK+00]

Die Clients sind bis auf den *Process Designer* alle Web-basiert. Der *Process Designer* dient der Erstellung von Prozeßmodellen und greift dabei direkt über ODBC auf die Datenbank zu. Die Web-Clients bestehen aus *Java Applets* (vgl. Abbildung 7.4), welche über einen Web-Server geladen werden. Die weitere Kommunikation findet direkt zwischen dem *Java Servlet* und der *Workflow Engine* bzw. dem *Process Manager* statt.

7.1.4 ADEPT

Das Ziel des Projekts *ADEPT (Application Development Based on Encapsulated Premodeled Process Templates)* an der Universität Ulm ist die Erarbeitung der technologischen Grundlagen und die Entwicklung eines WfMS der nächsten Generation [RBD00, RD98, uID01]. Einige zukunftsweisende Themen sind bereits im WfMS-Prototyp enthalten:

- $ADEPT_{base}$ als ausdrucksstarkes WF-Metamodell
- $ADEPT_{flex}$ zur Unterstützung dynamischer WF-Änderungen
- $ADEPT_{time}$ zur Berücksichtigung temporaler Aspekte
- $ADEPT_{distribution}$ zur besseren Skalierbarkeit durch eine verteilte WF-Steuerung

Im Rahmen eines Praktikums wurde für das *ADEPT*-WfMS eine Web-Schnittstelle entwickelt [KMN00] (vgl. Abbildung 7.6). Die Darstellung der Arbeitsliste, der WF-Applikationen sowie der Funktionen zur Steuerung von Prozessen und zur Durchführung dynamischer Änderungen erfolgt in drei verschiedenen *Frames* (vgl. Abschnitt 3.2.1.4). Dadurch ist es möglich, Aktualisierungen (z.B. Arbeitslisten-Updates) in einem *Frame* durchzuführen ohne die Darstellung in den anderen *Frames* zu beeinflussen.

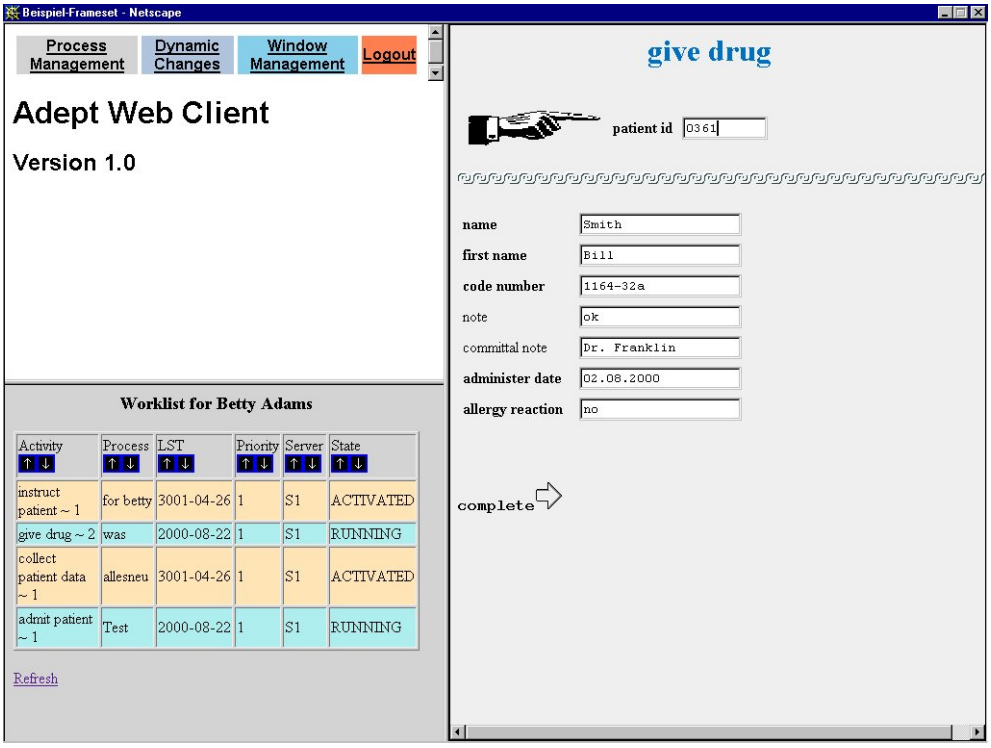

Abbildung 7.6: Screenshot des *ADEPT* Web-Client [KMN00]

Der Web-Client basiert auf der *Java Servlet* Technologie (vgl. Abschnitt 4.2.1.2). Ein zentrales *Servlet* (*Dispatcher*) nimmt die Anfrage entgegen und leitet sie an die Klassen `Workflow`, `Worklist` oder `Application` weiter, welche die Erstellung der (Antwort-)Seite übernehmen. Zu diesem Zweck werden HTML-Vorlagen geparst und interpretiert. Sind in den Vorlagen HTML-Formularen enthalten, wird unter Beachtung von Namenskonventionen ein *Mapping* zwischen den im Formular enthaltenen Feldern und den Parametern einer einer Workflow-Applikation durchgeführt. Die Werte von bereits belegten Parmetern werden bei der Interpretation dynamisch in das HTML-Formular eingefügt.

Spezieller Tags werden für die Darstellung der Arbeitsliste (z.B. <ADEPT_ACTIVITYNAME>) oder zur Ausgabe von Statusmeldungen (z.B. <ADEPT_RESULT>) eingesetzt.

Die Verbindung zum WfMS wird durch ein `WFServer`-Objekt der *ADEPT*-Java-API gehalten, welches nach der Anmeldung im Kontext der *Session* gespeichert wird und im weiteren Verlauf der Sitzung den Aufruf von WfMS-Funktionen über *Remote Method Invocation (RMI)* ermöglicht.

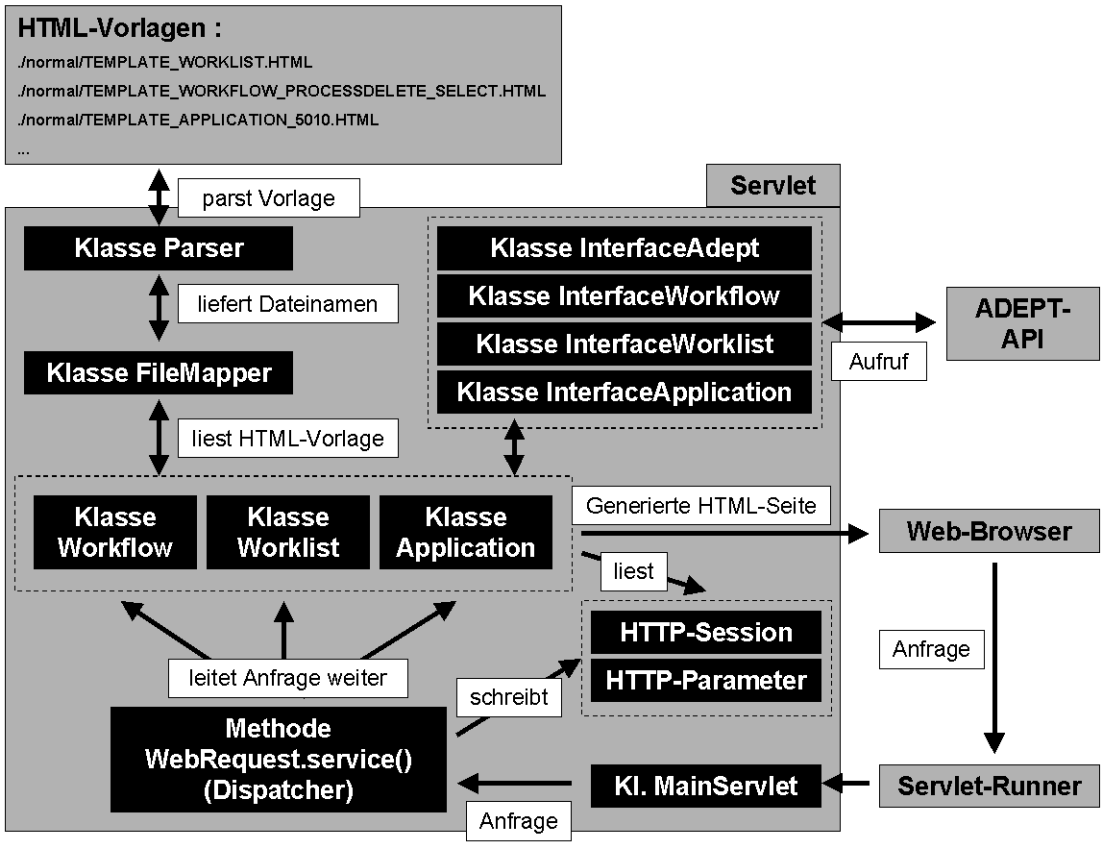

Abbildung 7.7: Architektur der *ADEPT* Web-Schnittstelle [KMN00]

7.2 Lösungen in kommerziellen Systemen

Die Bedeutung des Web als wichtige Benutzerschnitte und Plattform für Applikationen wurde auch von den bedeutenden Herstellern von WfMS erkannt. In den folgenden Abschnitten werden die Web-Clients von *Staffware*, *IBM MQSeries Workflow*, *Ultimus* und *pFlows* vorgestellt.

7.2.1 Staffware

Staffware ist ein prozeßorientiertes, sehr gut skalierbares und für verschiedene Systemplattformen (Windows NT, UNIX) verfügbares WfMS [Sta01, Sta99, Sta00b]. Es basiert auf leistungsfähigen Middlewarelösungen (wie CORBA) und nutzt moderne Komponentenarchitekturen (COM, EJB) zur Bereitstelllung seiner Funktionalität für den Entwickler.

Der Staffware Web-Client ermöglicht den Zugriff auf das Staffware-WfMS aus dem Inter- und Intranet. Er basiert auf dem *Microsoft Internet Information Server (IIS)* und der *Active Server Pages (ASP)* Technologie (siehe Abschnitt 4.2.2). Abbildung 7.8 zeigt die Integration des Web-Clients in die Architektur des Staffware-Systems.

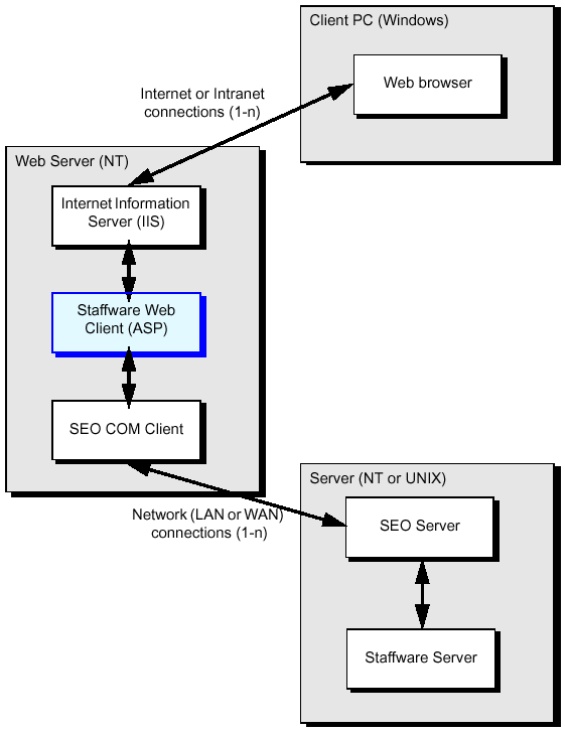

Abbildung 7.8: Architektur des Staffware Web-Client [Sta00a]

Der Web-Client benutzt die *Staffware Enterprise Objects (SEO)* zur Komminukation mit dem System. SEO sind Komponenten, welche die Funktionalität des WfMS für unterschiedliche Plattformen (Microsoft COM, Java EJB, CORBA) zur Verfügung stellen [Sta01]. Der Web-Client ruft aus den Visual Basic Skripten, welche in die ASP-Seiten eingebettet sind, die *SEO COM Client Objekte* und kommuniziert über sie mit dem *SEO*- bzw. dem *Staffware-Server* (siehe [Mar01]).

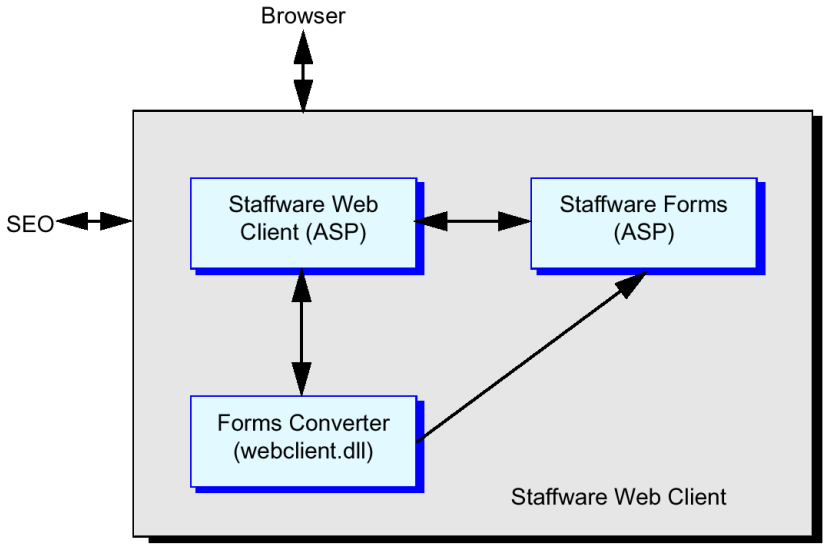

Abbildung 7.9: Interner Aufbau des Staffware Web-Client [Sta00a]

Der Staffware Web-Client besteht aus den in Abbildung 7.9 dargestellten Hauptkomponenten. Erhält der Staffware Web-Client (ASP) eine Browseranfrage, so sucht er im Cache (*Staffware Forms (ASP)*) nach der angeforderten ASP-Seite. Kann er die Seite nicht finden, ruft er den *Form Converter* auf, um die fehlende Seite generieren zu lassen. Der *Form Converter* hat die Aufgabe, Staffware Forms, welche für den normalen Standard-Client entwickelt wurden, in ASP-Seiten zu konvertieren. Die erzeugte ASP-Seite hat dann folgende vereinfachte Form:

```
...
<!-- #include file="..\..\standard.asp" -->
<%
...
’ build SEO node object
    if Init = true then
’ find our entry
    for i = 0 to session("LoggedInNodesCount")
        if session("NodeKey"&i) = NodeKey then
            set oNode = LoginNode(session("CompName"&i),
                                  session("NodeName"&i),
                                  session("Username"&i),
                                  session("Password"&i),
                                  session("TCPPort"&i),
                                  session("IPAddr"&i))
```

```
            i = session("LoggedInNodesCount")
        end if
    next
...
%>

<HTML>
<HEAD>
...
<script type="text/javascript" language="JavaScript">
...
</script>
</head>
<body onLoad="init();" >
...
<form id="StaffwareGuiForm"name="StaffwareGuiForm" onSubmit="return false">
<%
    if IsNewCase then
    'this is a case start, so show input field for case description
%>
    ...
    <input type="text" onChange="fields['CaseDescription'].updateField(this);"
      id="CaseDescriptionInput" name="CaseDescriptionInput" size="24"
      maxlength="24">
    ...
<%
    end if
%>
...
Name: <input type="text" id="NAME_1" name="NAME_1" size="20" maxlength="20"
class="REQ" onChange="updateSWField (this, 'NAME');">
...
</form>
<form id="StaffwareForm" name="StaffwareForm" acti-
on="S1Close.asp?<%=request.querystring%>"
      method="POST" onSubmit="return fillOutFields(this);" >
    <input type="hidden" id="CaseDescription" name="CaseDescription">
    <input type="hidden" id="DATUM" name="DATUM">
    ...
    <input type=SUBMIT value="<%= text(SWID_KEEP)%>" id="submitKeep"
      name="submitKeep" onClick="setKeep()">
    <input type=SUBMIT value="<%= text (SWID_RELEASE)%>" id="submitRelease"
      name="submitRelease" onClick="setRelease()">
</form>
...
</BODY>
</HTML>
```

Zuerst wird in der ASP durch `LoginNode` eine Verbindung zum *SEO-Server* hergestellt. Dann folgt die Erstellung des HTML-Quellcodes. Beim Laden der HTML-Seite im Client wird durch `<body onLoad="init();">` eine in Javascript geschriebene Funktion gerufen, welche zur Initalisierung der Felder im Formular dient. Staffware wendet einen Trick an, um die Formulareingaben prüfen zu können. Es definiert zwei Formulare, wobei nur eines für den Anwender im Browser sichtbar ist und von ihm bearbeitet werden kann. Ein anderes wird versteckt angelegt (`<input type="hidden" ...>`) und bleibt für den Anwender transparent. Werden im sichtbaren Formular Änderungen durchgeführt, werden diese durch die Funktion `updateSWField` geprüft (z.B. auf ein korrektes Datenformat) und erst dann in das versteckte Formular geschrieben, dessen Daten beim `Submit` verschickt werden.

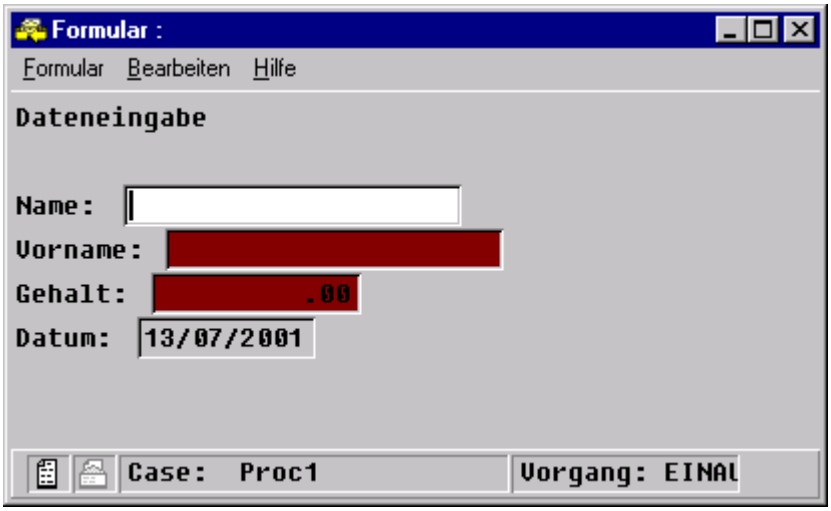

Abbildung 7.10: Screenshot eines Formulars im Staffware Standard-Client

Die Oberfläche des Staffware Web-Client ist recht ansprechend gestaltet und bietet Funktionen zur Ansicht der Arbeitsliste (*Work Queues*, siehe Abbildung 7.12), zur Übersicht über laufende Prozeßinstanzen (*Audit Trail*) und zum Starten von Prozessen (*Case Start*) (siehe Abbildung 7.10 und 7.11). Es wurde außerdem viel Wert auf die Benutzerführung gelegt. Wenn ein Formular zur Bearbeitung einer WF-Aktivität gestartet ist und der Anwender außerhalb des Formulars auf einen Link klickt, der den Inhalt des Formulars im Frame überschreiben würde, wird dies durch Javascript abgefangen. Der Benutzer kann dann selbst entscheiden, ob er die Bearbeitung unterbrechen und zu einem späteren Zeitpunkt fortsetzen möchte. Das Workitem wird in diesem Fall gesperrt („gelockt“) und mit einem entsprechenden Symbol in der Arbeitsliste dargestellt.

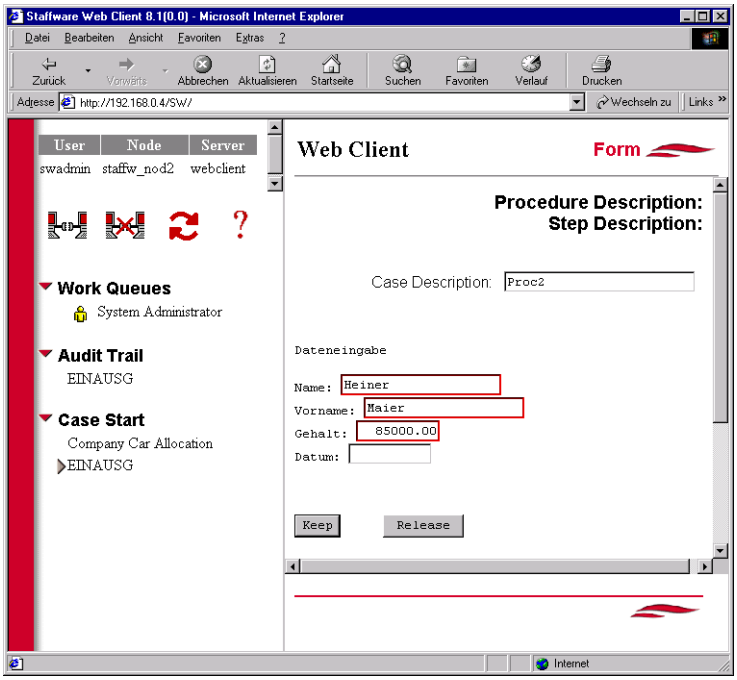

Abbildung 7.11: Screenshot eines Formulars im Staffware Web-Client

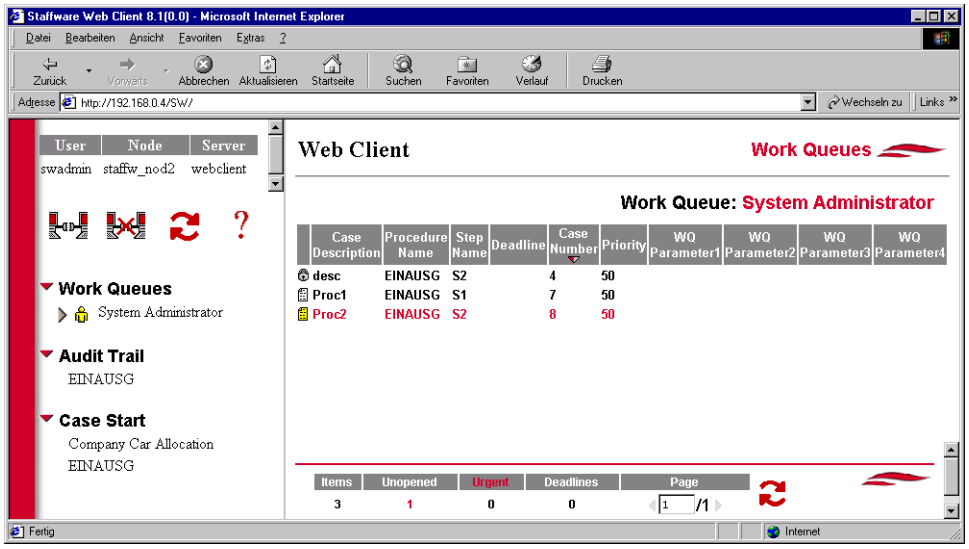

Abbildung 7.12: Screenshot der Arbeitsliste im Staffware Web-Client

7.2.2 IBM MQSeries Workflow

IBM MQSeries Workflow ist der Nachfolger des sehr populären WfMS *Flowmark* [IBM01b, LR00, Mar01]. *IBM MQSeries Workflow* ist ein prozeßorientiertes WfMS und basiert auf der (nachrichtenorientierten) Middleware *MQSeries* von IBM. Die Server für *IBM MQSeries Workflow* sind für ein breites Spektrum von Systemplattformen und -leistungsklassen verfügbar (von WindowsNT bis OS/390). Zur Modellierung der Workflows werden Aktivitätennetze [DR00, LR00] verwendet, welche die Trennung zwischen Kontroll- und Datenfluß erlauben.

Der Web-Client von *IBM MQSeries Workflow* basiert auf der *Java Servlet Technologie* von SUN (siehe Abschnitt 4.2.1.2). Die HTML-Seiten werden dynamisch durch die in Abbildung 7.13 dargestellten Komponenten generiert. Der Zugriff auf das WfMS erfolgt über eine Java-API und *JNI (Java Native Interface)*.

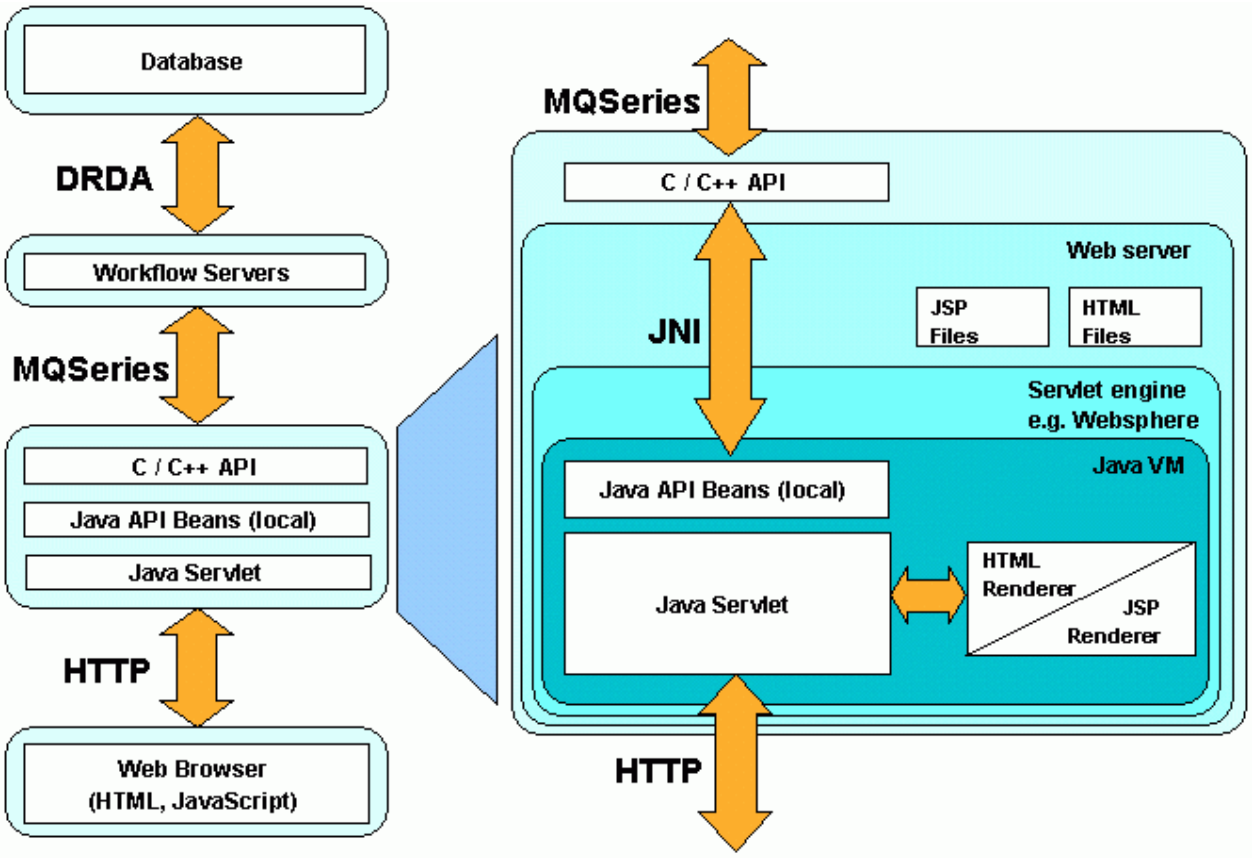

Abbildung 7.13: Architektur des IBM MQSeries Workflow Web Client [IBM01c]

JNI erlaubt den Zugriff aus der *Java Virtual Machine (JVM)* der *Servlet Engine* auf die *C++ Programmierschnittstelle (API)* von *IBM MQSeries Workflow*. Dies hat den Vorteil, daß bei der Installation von Workflow-Server und Web-Server auf derselben Maschine keine langsamen Netzverbindungen zwischen den beiden Server aufgebaut werden müssen, sondern die Kommunikation durch einen direkten Funktionsaufruf erfolgen kann. Die *C++ API* kommuniziert dann über *MQSeries* als Middleware mit dem Workflow-Server. Für den Fall, daß Workflow-Server und Web-Server auf unterschiedlichen Rechnern installiert sind oder daß es für die Plattform des Web-Servers keine Unterstützung für *MQSeries* gibt, kann die Java API auch für eine Kommunikation mit dem Workflow-Server über CORBA konfiguriert werden.

Als Basis des Web-Client von *IBM MQSeries Workflow* dient ein Web-Server mit *Java Servlet* Unterstützung (z.B. Jakarta-Tomcat oder IBM Websphere). Er bildet die Schnittstelle zum WfMS, indem er von ihm erhaltenen Informationen durch eine `Viewer`-Klasse generisch in HTML-Seiten konvertiert und über HTTP den Clients zugänglich macht. Die `Viewer`-Klasse kann an eigene Anforderungen angepaßt oder es können *Java Server Pages (JSP)* zur Erzeugung der HTML-Seiten verwendet werden (vgl. Abschnitt 4.2.1.3). Die JSP erlauben dem Anwender, seine eigenen Designvorstellungen besser umzusetzen, da sie eine individuelle Gestaltung der Seite und die Programmierung in Java zulassen. Der Aufbau einer solchen JSP soll am Beispiel einer mit dem Web-Client ausgelieferten Applikation verdeutlicht werden [IBM01c]:

```
...
<jsp:useBean id="context" scope="request"
             type="com.ibm.workflow.servlet.client.RequestContext"/>
...
<%
  WorkItem           workItem    = context.getWorkItem();\\
  ReadOnlyContainer  input       = context.getContainer();
  long               customerID  = 0;
  long               loanID      = 0;
...
  try {
    customerID = input.getLong( "CustomerID" );
    loanID     = input.getLong( "LoanID"     );
  } catch( FmcException xcpt ) { out.println( xcpt.toString( ) ); }
...
<html>
<head> <meta http-equiv="Content-Type" content="text/html; ... </head>
<body>
...
        <%=context.openForm("checkInWorkItem", workItem.persistentOid())%>
        <table border="0" cellspacing="1" width="100%">
          <tr>
            <td><b>Customer ID</b></td>
            <td><%=customerID%></td>
            <td><b>Loan ID</b></td>
            <td><%=loanID%></td>
          </tr>
          <tr>
            <td><b>Customer Name</b></td>
            <td><%=getField( customerResultSet, "FIRSTNAME" )%>
```

```
                <%=getField( customerResultSet, "LASTNAME" )%> </td>
          ...
            <td><font color="#FF0000"><b>Approve</b></font></td>
            <td><input type="radio" name="Approved" value="1">Yes</td>
            <td><input type="radio" name="Approved" value="0">No</td>
          </tr>
        </table>
        <p>
        <input type="submit" name="submitData" value="Complete Work Item">
        <input type="button" name="cancel" value="Cancel"
          onClick="javascript:location.replace('
          <%=context.getCommand("cancelWorkItem",
             workItem.persistentOid())%>')">
        <input type="button" value="Display Credit History"
          name="DisplayCreditHistory"
          onClick="javascript:window.open('<%=context.getURL("{_HTMLDir_}/...
        </p>
      </form>
...
</body>
</html>
```

Die obige JSP implementiert die Aktivität „Bewilligung“ („GetApproval“) in einem Kreditbearbeitungsprozeß. Hat ein Anwender diese Aktivität in seiner Worklist ausgewählt, wird die JSP durch den Web-Server bearbeitet und die Ausgabe an den Klienten geschickt.

Die *JavaBean* `RequestContext` ermöglicht der JSP den Zugriff auf das Worktitem und dessen Inputcontainer:

```
WorkItem workItem = context.getWorkItem();
ReadOnlyContainer input =context.getContainer();
```

Auf diese Weise wird zum Beispiel das Feld `CustomerID` ausgelesen:

```
customerID = input.getLong( "CustomerID" );
```

und später in den HTML-Code integriert (<%=`customerID`%>). Weitere Felder wie `FIRSTNAME`, `LASTNAME` oder `AMOUNT` werden anhand der Schlüssel `CustomerID` und `LoanID` über *SQL-Anfragen* direkt aus der Datenbank ausgelesen. Für den Rückgabewert `Approved` werden Radiobuttons in einem Formular angelegt, welche entsprechend der Entscheidung über den Kreditantrag `0` oder `1` in dem HTTP-Parameter `Approved` beim Drücken des `Submit`-Buttons zurückliefern. Dieser Parameter kann dann vom Servlet in den Ausgabecontainer geschrieben und das Workitem beendet werden.

7.2.3 Ultimus

Ultimus nutzt die Technologien des *Internet Information Server (IIS)* und des *Microsoft Transaction Server (MTS)* zur Bereitstellung seiner Workflow-Funktionalität über das Web (vgl. Abschnitt 4.2.2) [Ult01a]. Der Workflow Server basiert auf einer *COM/DCOM-Architektur* (vgl. Abbildung 7.14) und kommuniziert mit dem IIS über die Dienste des MTS.

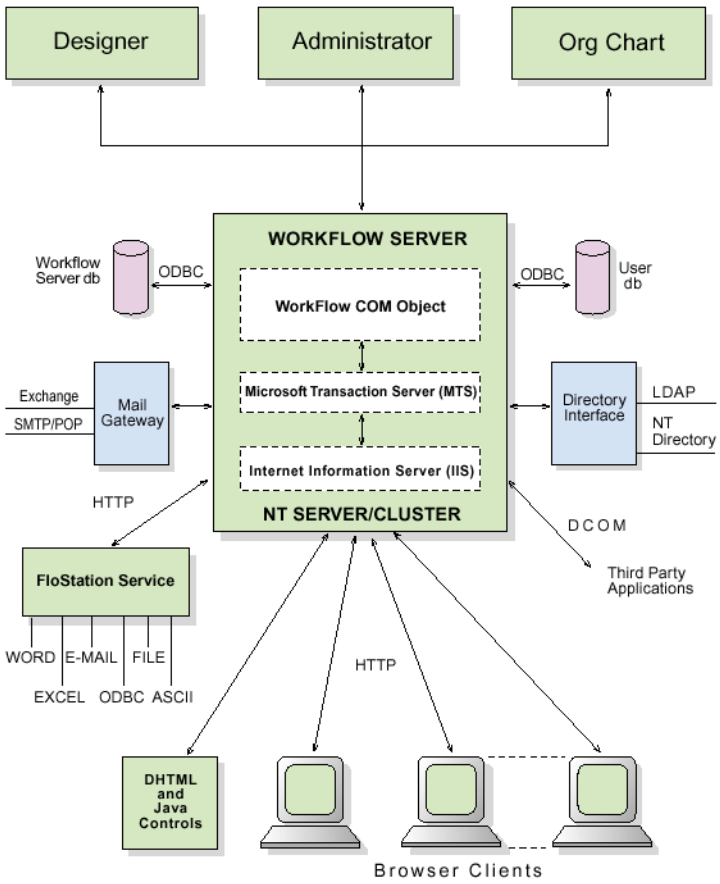

Abbildung 7.14: Architektur des *Ultimus* WfMS [Ult01b]

Bei den *Ultimus Browser Clients* werden für den *Microsoft Internet Explorer Dynamic HTML* und *ActiveX*-Steuerobjekte sowie für den *Netscape Communicator* und *Javascript* und *Java Applets* eingesetzt.

7.2.4 pFlows

Die *pFlows Work Process Engine* wurde nach dem WfMC-Standard für die *Enactment Services* implementiert (vgl. Kapitel 2). Das System basiert auf einer Architektur vgl. Abbildung 7.15) mit einem *Web Application Server* nach der *Java 2 Enterprise Edition* [Sun01b] und der *Enterprise Java Beans* Spezifikation [Sun01e].

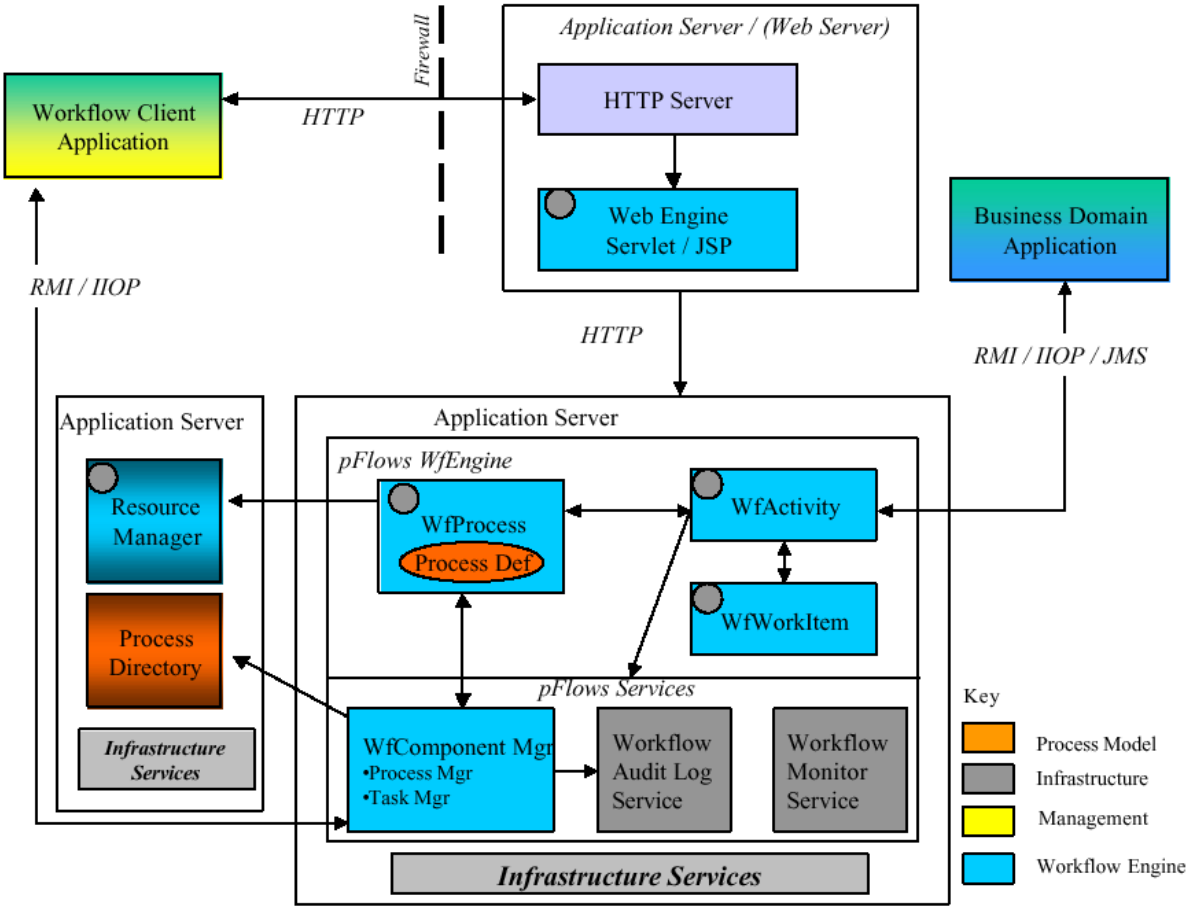

Abbildung 7.15: Architektur der pFlows Work Process Engine [Men01]

Die *Workflow Engine* ist durch mehrere verteilte EJB-Komponenten realisiert. Die Kommunikation zwischen Workflow-Client und der *Workflow Engine* läuft entweder über einen HTTP-Server mit Servlet- und JSP-Unterstützung oder direkt über eine J2EE- und CORBA-kompatible Middlewareplattform. Informationen darüber, welche Technologien auf Clientseite eingesetzt werden (z.B. *DHTML*, *Java Applets*, usw.), waren leider nicht vorhanden.

7.3 Vergleich und Bewertung

Die vorgestellten Systeme verfolgen teilweise ganz unterschiedliche Ansätze, welche auch stark durch den zeitliche Kontext ihrer Entstehung geprägt sind. Tabelle 7.1 faßt die Vor- und Nachteile der Systeme noch einmal kurz zusammen.

	Prototypisch implementierte WfMS			
	Panta Rhei	**WebFlow**	**WW-Flow**	**ADEPT**
Technologie (Server)	CGI	eigene	Servlet	Servlet
Serverunabhägigkeit	+	+	+	+
Technologie (Client)	HTML, Applet	HTML, Applet	Applet	HTML
Clientunabhägigkeit	+	+	+	+
Funktionalität (Client)	–	○	+	○

	Kommerzielle WfMS			
	Staffware	**MQSeries**	**Ultimus**	**pFlows**
Technologie (Server)	ASP	Servlet/JSP	ASP	Servlet/JSP
Serverunabhägigkeit	–	+	–	+
Technologie (Client)	DHTML	HTML	DHTML, ActiveX, Applet	
Clientunabhägigkeit	–	+	+	
Funktionalität (Client)	+	–	–	

Erläuterung: + = gut, ○ = neutral, – = schlecht

Tabelle 7.1: Vergleich und Bewertung der Web-basierten WfMS

Panta Rhei benutzt eine der ältesten Techniken (CGI) zur Erstellung dynamischer Inhalte und *WebFlow* schaltet einen eigenen Server zur Integration des WfMS vor. Dadurch erreichen beide eine sehr große Unabhängigkeit vom verwendeten Web-Server. Auch die Darstellung im Client als HTML-Seite bzw. Java-Applet ist sehr ähnlich und besitzt eine breite Unterstützung durch viele Web-Browser. Sicherheitskritisch sind die direkte, unverschlüsselte Kommunikation mit der Datenbank über JDBC und der freie Zugriff auf alle Tabellen. Außerdem erfordern die Java-Applets die Gewährung bestimmter Zugriffrechte (*Java Policies*), was den Administrationsaufwand auf Klientenseite erhöht.

WW-Flow arbeitet ebenfalls mit Java-Applets im Browser. Allerdings ist aus der vorhandenen Dokumentation nicht ersichtlich, wie die Interaktion mit dem Servlet im Detail aussieht und ob daraus vielleicht ebenfalls sicherheitskritische Situationen entstehen können.

Auf Serverseite werden beim *ADEPT*-Web-Client *Java Servlets* eingesetzt. Die Präsentation im Client erfolgt in „reinem“ HTML, sodaß es keine speziellen Browserabhängigkeiten gibt, und eine Darstellung auch auf *Thin Web-Clients* (vgl. 5.1) möglich ist.

Staffware und *Ultimus* basieren beide auf dem *Microsoft Internet Information Server*. Als Web-Browser wird bei Staffware nur der *Microsoft Internet Explorer* unterstützt. Diese sehr einseitige Ausrichtung auf eine Systemplattform ist für den unternehmensweiten Einsatz negativ zu sehen. Ultimus bietet wenigstens noch die Unterstützung für andere Web-Browser (z.B. Netscape Communicator) an.

IBM MQSeries Workflow setzt auf der Java Servlet und Java Server Pages Technologie auf, welche mittlerweile eine breite Unterstützung durch viele Web-Server auf vielen Systemplattformen gefunden hat.

pFlows setzt auf Serverseite sehr moderne Komponententechnologien ein, welche nicht nur im Hinblick auf die Anbindung von Web-Clients (z.B. Konformität zu *J2EE*), sondern auch für die Integration von WF-Funktionlität in Anwendungen interessant sind. Software-Komponenten bieten in Bezug auf Wiederverwendbar- und Erweiterbarkeit momentan das größte Potential.

Die Funktionalität der Clients läßt vor allem bei den kommerziellen Systemen zu wünschen übrig. Sie unterstützen in der Regel lediglich die Arbelistenverwaltung und die Ausführung von Aktivitäten via Web-Browser. Doch selbst hier sind wichtige Funktionen (vgl. Abschnitt 2.4), wie ein automatisches Update der Arbeitsliste nicht implemeniert. Die Administration des WfMS kann nur bei *Panta Rhei*, dem ältesten der vorgestellten Systeme, komplett Web-basiert durchgeführt werden. Zwar gilt es immer, Sicherheitsaspekte bei der Client-seitigen Verarbeitung und der Übertragung durch das Netz zu bedenken, allerdings gibt es Möglichkeiten (z.B. Authorisierung bei J2EE oder SSL), diese zu verbessern. Die Benutzerführung im Client ist bei *Staffware* am besten realisiert. Staffware prüft die Formulareingaben über JavaScript auf ein korrektes Eingabeformat und warnt den Anwender beim Verlassen eines Frames mit einer gestarteter Aktivität.

Vor allem hinsichtlich der Benutzerfreundlichkeit und der Funktionalität haben die Web-Clients gegenüber den „Standard-Workflow-Clients“ noch Nachholbedarf. Der Einsatz von ActiveX, Java-Applets und DHTML werden zu einer Verbesserung der Situation führen. Dennoch bleiben die Möglichkeiten der Ausführung von Programmcode im Web-Browser begrenzt, so daß Web-Clients nur für bestimmte Einsatzgebiete geeignet sind (siehe Kapitel 8).

Kapitel 8

Diskussion

In den vorangehenden Kapiteln wurden Aspekte Web-basierter WfMS und Technologien zu ihrer Umsetzung vorgestellt. In diesem Kapitel soll die Thematik konventioneller und Web-basierter Workflow-Clients nochmals im Gesamtzusammenhang behandelt werden. Dazu werden in Abschnitt 8.1 die Funktionalitäten von konventionellen und von Web-Clients verglichen sowie in Abschnitt 8.2 die möglichen Einsatzzwecke und in Abschnitt 8.3 die Potentiale Web-basierter WfMS diskutiert.

8.1 Funktionalitätsvergleich

WfMS besitzen in der Regel unterschiedliche Schnittstellen, welche die vom System bereitgestellte Funktionalität den verschiedenen Anwendergruppen zur Verfügung stellen. Welche Funktionalität von konventionellen Workflow-Clients und welche von Web-Clients bereitgestellt werden kann, wird in den folgenden Abschnitten geklärt.

Web-Clients sind in ihrer Funktionalität konventionellen Workflow-Clients üblicherweise unterlegen. Die standardisierten Formate zur Seitenbeschreibung (z.B. HTML) sowie zu deren Übertragung (z.B. HTTP) lassen den Entwicklern nur einen sehr begrenzten Freiraum. Individuelle Client-Programmierung läßt sich nur durch erweiterte Technologien wie *Javascript*, *Java Applets* oder *ActiveX-Steuerobjekte* realisieren (vgl. Kapitel 5). Diese Erweiterungen führen allerdings auch zu Problemen, da sie nicht von allen Clients unterstützt werden. *Dynamic HTML* ist ein Beispiel für diese Kompatibilitätsprobleme zwischen Web-Browsern unterschiedlicher Hersteller.

Konventionelle Workflow-Clients, welche als eigenständige Anwendung ausgeführt werden, können dagegen das gesamte Spektrum der vom WfMS unterstützten Programmierschnittstellen (z.B. API in C/C++, Java oder als Komponentenmodell), GUI-Komponenten (z.B. Java Swing, Motif), sowie der Kommunikations- und Middlewaretechnologien (z.B. RPC, CORBA oder TP-Monitore) einsetzen.

Bezogen auf WfMS lassen sich folgende drei Benutzerschnittstellen unterscheiden (vgl. Kapitel 2.4):

8.1.1 Anwenderschnittstelle

Die Anwenderschnittstelle eines WfMS umfaßt folgende Hauptfunktionen (vgl. Abschnitt 2.4.1.2):

- *Arbeitslisten-Darstellung*
  Die wichtigste Aufgabe der Anwenderschnittstelle ist die Darstellung der Arbeitsliste. In einem konventionellen WfMS-Client geschieht dies in der Regel in Form einer Tabelle, welche den Namen der Aktivitäten und zusätzliche Informationen (wie WF-Instanz oder Priorität) enthält. Diese Darstellungsform ist auch im Web-Client sehr gut möglich, da HTML bereits Formatierungselemente für Tabellen (vgl. Abschnitt 3.2.1) enthält.

- *Arbeitslisten-Sortierung*
  Zu einer komfortablen Darstellung von Arbeitslisten gehört die Möglichkeit, diese nach unterschiedlichen Kriterien (z.B. alphabetisch, nach Priorität, nach Workflow-Instanzen, usw.) zu sortieren. Damit der Benutzer die von ihm präferierte Sortierreihenfolge nicht bei jedem Abruf der Arbeitliste erneut einstellen muß, ist es sinnvoll, diese im Kontext der Sitzung des Benutzers (*Session*) oder sogar in einem (Benutzer-)Profil in der Datenbank abzulegen. Bei den weiteren Abfragen der Arbeitsliste kann diese dann gemäß den (Benutzer-)Kriterien sortiert dargestellt werden.

- *Arbeitslisten-Selektion*
  Vielfach ist es wünschenswert, dem Benutzer nur bestimmte Ausschnitte seiner Gesamtarbeitsliste (z.B. nur die Einträge einer bestimmten WF-Instanz) anzuzeigen. Hier ist es wie bei der Sortierung wichtig, die Selektionskriterien im Benutzerkontext abzuspeichern.

- *Arbeitslisten-Update und aktive Benachrichtigungen*
  Damit der Anwender immer einen aktuellen Überblick über die anstehenden Tätigkeiten hat, ist es wichtig, seine Arbeitsliste bei Änderungen „zeitnah“ zu aktualisieren (*Update*). Konventionelle Workflow-Clients besitzen eine direkte Verbindung zum WfMS und können deshalb aktiv über Veränderungen informiert werden [Wol97]. Da der Web-Client nach dem Abruf der Arbeitsliste aber keine Verbindung mehr zum Web-Server bzw. WfMS hat, kann eine aktive Benachrichtigung nicht direkt erfolgen. Der Anwender muß die Arbeitsliste entweder manuell (z.B. durch die *Reload*-Funktion des Browsers) aktualisieren oder der Entwickler kann in der HTML-Seite Zeitintervalle (durch *Meta-Flags* [MN98]) für eine automatische Aktualisierung einstellen. Letzteres hat allerdings den Nachteil, daß die Arbeitsliste auch neu geladen wird, wenn es keine Änderungen gegeben hat. Diese „unnötigen“ Aktualisierungen erhöhen die Last des Web- sowie des WfMS-Servers, was sich vor allem bei vielen Clients negativ auf die Performanz des Systems auswirkt.

  Die Integration eines Java Applets, welches eine Verbindung zum WfMS besitzt und bei Benachrichtigungen ein Update der Seite anstößt, löst dieses Problem. Allerdings bestehen im Zusammenhang mit Applets Sicherheitsbeschränkungen (vgl. 5.2.2), welche zum

Beispiel keinen Verbindungsaufbau zu einem beliebigen Server (außer dem Web-Server) zulassen. Nur durch spezielle Sicherheitseinstellungen kann dies umgangen werden.

Aktive Benachrichtigungen werden, außer bei Veränderungen der Arbeitsliste, auch für Meldungen beim Überschreiten einer *Deadline* oder für sonstige Fehlermeldungen (z.B. Abbruch eines Prozesses) benötigt. In diesem Fall kann die entsprechende Meldung vom Applet zum Beispiel in einem eigenen neuen Fenster (*Pop-Up*) dargestellt werden.

- *Verwaltung von Workflow-Instanzen und -Aktivitäten*
  Die Verwaltung von Workflow-Instanzen ist sowohl im konventionellen WfMS-Client als auch im Web-Client gut möglich. HTML-Formulare ermöglichen durch Listenfelder eine einfache Auswahl von zu startenden bzw. zu beenden WF-Instanzen.

  Für die Verwaltung von Workflow-Aktivitäten können im wesentlichen dieselben Verfahren wie bei Workflow-Instanzen verwendet werden.

- *Einbezug externer Anwendungen*
  Die Integration externer Anwendungen gestaltet sich allgemein schwierig. Da konventionelle Clients vollen Zugriff auf das Betriebssystem haben, können sie die externen Anwendungen direkt (bzw. unter Einbezug von Wrappern) mit den nötigen Parametern (z.B. Verbindungsinformation zum WfMS oder den Workflow-Daten) starten. Baut die externe Applikation keine eigene Verbindung zum WfMS auf, über welche sie die Parameterdaten übertragen kann, so muß diese Aufgabe ebenfalls vom Workflow-Client übernommen werden.

  Konventionelle Clients können unter Windows über *Object Linking and Embedding (OLE)* bequem Daten mit anderen Anwendungen austauschen. Für Web-Clients existiert eine solche normierte Schnittstelle allerdings nicht. Insbesondere erschweren die Sicherheitseinstellungen der Browser oder der *Java Virtual Machine* den Zugriff auf das Betriebssystem und damit auch auf andere Applikationen. Theoretisch ist es zwar denkbar, durch spezielle Sicherheitseinstellungen Daten zu speichern und Anwendugen zu starten, allerdings erfordert dies einen erhöhten Konfigurationsaufwand auf Client-Seite. Für ActiveX-Steuerelemente existieren geringere Sicherheitseinschränkungen, weshalb sich diese besser für die Integration von Applikationen im Windows-Umfeld eignen.

  Die angesprochenen Verfahren ermöglichen zwar die (begrenzte) Integration externer Anwendungen, allerdings gehen wichtige Vorteile von Web-Applikationen, wie die Unabhänigkeit vom Client oder der geringe Konfigurationsaufwand, verloren.

8.1.2 Entwicklerschnittstelle

Die Entwicklerschnittstelle eines WfMS (*Buildtime*) ermöglicht u.a. die Erstellung und Verwaltung (z.B. Import / Export) von Prozeßvorlagen. Daneben spielen Modelloptimierungen und die Prüfung der Modelle auf formale Korrektheit (z.B. Verklemmungsfreiheit, Erreichbarkeit bestimmter Zustände) ebenfalls eine wichtige Rolle.

- *Darstellung und Editieren von WF-Graphen*
  Die Modellierung von Workflows erfolgt durch graphische Unterstützung, weshalb die

Darstellung eines WF-Graphen eine zwingende Grundfunktionalität ist. Konventionelle *Buildtime-Clients* bieten außerdem die Möglichkeit, diese Graphen auf einfache Weise editieren oder erweitern zu können (z.B. durch *Drag-And-Drop*). Die Darstellung eines WF-Graphen ist im Web-Client prinzipiell ebenfalls möglich. Dynamisch erzeugte Grafiken oder *Flash*-Elemente (vgl. 5.2.4) sind Beispiele dafür.

Ein sinnvolles Editieren von Graphen ist mit diesen Technologien allerdings nicht möglich, da die Logik auf Client-Seite nicht ausreichend ist und deshalb häufige Server-Anfragen nötig wären. Beim Einfügen oder Entfernen von Aktivitäten müßte der Server zum Beispiel jedesmal eine neue Grafik dynamisch erzeugen. Neben dem hohen Datentransfervolumen erzeugt dies insbesonders auf Server-Seite eine große Last. Ein effektives Editieren von Graphen ist nur durch *Java Applets* bzw. *ActiveX*-Steuerelemente möglich. Sie sind in der Lage, die Eingaben direkt zu visualisieren und können außerdem die komplexen Daten verwalten, welche bei der Erstellung eines WF-Graphen entstehen.

- *Optimierungen und Konsistenzprüfungen*
  Automatische Optimierungen und Konsistenzprüfungen durch das WfMS lassen sich sehr gut serverseitig durchführen, solange der Interaktiongrad des Benutzers relativ niedrig ist. Die Zustimmung des Benutzers zu einer optimierten Version eines WF-Modells oder die Ausgabe von Fehlern bei der Konsistenzprüfung lassen sich auch durch Web-Clients gut realisieren.

- *Import / Export von Prozeßvorlagen*
  Die Die Entwicklerschnittstelle eines WfMS bietet in der Regel auch Möglichkeiten zum Im- bzw. Export von Prozeßvorlagen. Diese Vorlagen sind meistens im lokalen Dateisystem gespeichert. Deshalb ist hierzu ein Zugriff des Clients nötig. HTML bietet durch einem sog. *HTTP-File-Upload* [MN98] die Möglichkeit Dateiinhalte ähnlich wie normale HTML-Formulardaten via HTTP zum Web-Server zu versenden.

8.1.3 Administratorschnittstelle

Die Administratorschnittstelle dient der Verwaltung und Überwachung des WfMS (*Monitoring Client*).

- *Anzeige von Informationen und Statistiken*
  Zur Überwachung des WfMS zählen zum Beispiel die Anzeige von WF-Instanzen mit „überfälligen“ Aktivitäten oder die Ausgabe von Statistiken über die Auslastung von WfMS-Servern. Diese Informationen können entweder in textueller Form (z.B. als Listen) oder graphisch durch dynamisch erzeugte Grafiken (z.B. Prozeßgraph einer WF-Instanz) dargestellt werden.

- *Verwaltung von Benutzern oder der Organisationsstruktur*
  Einfache Aufgaben, wie das Anlegen von Benutzern, können durch HTML-Formulare erledigt werden. Für komplexe oder graphisch unterstützte Tätigkeiten, etwa Änderungen des Organisationsmodells, bieten sich *Java Applets* bzw. *ActiveX*-Steuerelemente, welche in Verbindung zum WfMS stehen, an.

8.1.4 Zusammenfassung

Tabelle 8.1 faßt zusammen, welche (Grund-)Funktionalität konventionell realisierte WfMS-Clients und Web-Clients typischerweise anbieten.

Die in der Tabelle enthaltenen Funktionen stellen nur einen Teilausschnitt dar im praktischen Einsatz benötigten Client-Funktionalität dar. Allerdings sind viele der aufgezeigten Probleme sowie deren Lösungen auch auf andere Bereiche übertragbar. Für Web-Clients wird zusätzlich angegeben, welche Web-Technologien zur Unterstützung einzelner Funktionen eingesetzt werden können.

Funktionalität	konv. Client	Web-Client	mögliche Web-Technologie
Anwenderschnittstelle			
Arbeitslisten-Darstellung	+	+	z.B. als HTML-Tabelle
Arbeitslisten-Sortierung	+	+	Sortier- bzw. Selektionskriterien werden in einer *Session* gespeichert und bei der dynam. Erstellung der Arbeitsliste berücksichtigt
Arbeitslisten-Selektion	+	+	siehe Arbeitslisten-Sortierung
Arbeitslisten-Update	+	○	z.B. durch *Reload* in bestimmten Zeitintervallen oder mit Hilfe eines Java Applets, welches eine Verbindung zum WfMS hat und einen *Reload* der Seite automatisch auslöst
Aktive Benachrichtigungen	+	○	z.B. durch Java Applet, welches eine Verbindung zum WfMS hat und Meldungen am Bildschirm ausgeben kann
Verwaltung v. WF-Instanzen	+	+	z.B. durch HTML-Formulare
Verwaltung v. WF-Aktivitäten	+	+	z.B. durch HTML-Formulare
Externe Anwendungen	○	-	z.B. durch Java Applets mit „gelockerten“ Sicherheitseinstellungen oder durch ActiveX-Steuerelemente
Entwicklerschnittstelle			
Darstellung v. WF-Graphen	+	○	z.B. durch dynamisch erzeugte Grafik
Editieren v. WF-Graphen	+	-	z.B. durch Java Applet oder ActiveX-Steuerelement
Import / Export von Prozeßvorlagen	+	○	z.B. durch HTTP-Upload
Administratorschnittstelle			
Anzeige von Statistiken	+	+	z.B. in Textform durch HTML oder graphisch (z.B. als Diagramm) durch dynamisch erzeugte Grafik
Verwaltung v. Benutzern	+	+	z.B. durch HTML-Formulare
Verwaltung d. Org.struktur	+	○	z.B. mit graphischer Unterstützung durch Java Applet oder ActiveX-Steuerelement

Erläuterung: + = gute Unterstützung, ○ = bedingte Unterstützung, – = schlechte Unterstützung

Tabelle 8.1: Vergleich der Funktionalität von konv. und Web-basierten Workflow-Clients

8.2 Einsatzzwecke

Wie bereits im vorangehenden Abschnitt diskutiert, unterscheiden sich Web-Client und konventioneller Workflow-Client vor allem hinsichtlich ihrer Funktionalität. Generell stellt sich die Frage, ob bei allen Einsatzzwecken eines WfMS Client-seitig die volle Funktionalität (z.B. Arbeitslisten-Update, aktive Benachrichtigungen, usw.) benötigt wird oder ob sich für bestimmte Anwendungen Web-Clients vielleicht sogar die bessere Alternative darstellen. Hierbei sind folgende Kriterien von Bedeutung:

- *Installationsaufwand*
  Web-basierte WF-Anwendungen haben den großen Vorteil, daß sich der Installations- und Konfigurationsaufwand auf Client-Seite aufgrund der Verwendung von Web-Clients (z.B. (Standard-)Web-Browser) auf ein Minimum reduziert. Daher können auch solche Anwender das WfMS nutzen, die keinen Workflow-Client installiert haben bzw. installieren wollen.

- *Client-Update*
  Die Einführung einer neuen Client-Version bereitet bei der Installation auf vielen Arbeitsplätzen große Probleme. Die Verwaltung der WF-Clients und WF-Applikationen auf einem zentralen Server macht diese Umstellung bei Web-basierten WfMS sehr einfach.

- *Formularorientierung*
  Ein wesentlicher Punkt bei Web-basierten WF-Anwendungen ist, daß sich die Workflows formularorientiert darstellen lassen müssen (vgl. Abschnitt 2.3.1). Nur so kann eine Unterstützung durch gängige Standards wie HTML (vgl. Abschnitt 3.2.1.3) oder vergleichbare Ansätze (z.B. WML) erreicht werden.

- *Datensstruktur*
  Web-Clients können in der Regel nur einfache Datenstrukturen (z.B. Strings) verarbeiten. Komplexe Objekte (z.B. Listen) können nicht durch die standardisierten Kommunikationsmechanismen (z.B. als HTTP-Paramter) übertragen werden. Dadurch wird das Format der Ein-/Ausgabeparameter stark eingeschränkt. Mit *XForms* [Wor01j] etwa wird durch die *W3C* an Möglichkeiten für Formulare mit komplexeren Datentypen (z.B. Datums- oder Währungstypen) und Formatüberprüfungen gearbeitet, allerdings werden diese Techniken von den meisten Web-Browsern bisher nicht unterstützt und kommen daher in der Praxis kaum zum Einsatz.

Anhand der oben genannten Kriterien lassen sich im wesentlichen folgende Einsatzzwecke für Web-Clients (vgl. Abschnitt 8.2.1) und für konventionelle Workflow-Clients (vgl. Abschnitt 8.2.2) ausmachen:

8.2.1 Web-Client

Handelt es sich um Geschäftsprozesse, welche vielen Anwendern über eine standardisierte und weit verbreitete Schnittstelle ohne eine spezielle Installation zugänglich gemacht werden sollen (wie im B2B- oder B2C-Bereich), sind Web-Clients die richtige Wahl. Ein typisches Szenario für einen sehr gut über das Web realisierbaren Workflow ist die Buchung einer Reise in einem „virtuellen“ Reisebüro im Web. Der Kunde wählt ein Angebot aus und drückt dann einen Knopf, welcher einen Workflow zur Buchung der Reise startet. Anschließend kann der Kunde in mehreren (als Formulare implementierten) Aktivitäten, seine Anschrift, die gewünschte Zahlungsweise und eine abschließende Bestätigung der Reise durchführen. Die jeweils zu erfassenden Daten sind als Strings darstellbar und lassen sich daher auch gut über HTML-Formulare übertragen.

Web-basierte WfMS eignen sich für eine für den Client transparente Ablaufsteuerung (durch das WfMS) und den Datenfluß zwischen mehreren HTML-Seiten. Hier zeigt sich auch der Vorteil gegenüber „starr“ programmierten Web-Anwendungen, welche keine schnelle Anpassung an neue Gegebenheiten (z.B. zusätzliche Erfassung einer E-Mail-Adresse) erlauben.

8.2.2 Konventioneller Workflow-Client

Die Stärken eines konventionellen Workflow-Clients liegen in der direkten Kommunikation mit dem WfMS und mit fremden Applikationen. Auf diese Weise ist es zum Beispiel möglich, direkte (aktive) Benachrichtigungen vom WfMS, z.B. bei Änderungen in der Arbeitsliste oder bei Überschreiten von *Deadlines*, zu erhalten. Außerdem können auf diese Weise auch komplexe Daten(strukturen), wie Textobjekte ausgetauscht und ggf. von externen Applikationen (z.B. einer Textverarbeitung) bearbeitet werden.

Für spezielle Anwendungsbereiche (z.B. in Kliniken) wird es sicher sinnvoll sein, den Zusatzaufwand für Installation und Konfiguration in Kauf zu nehmen, weil die hinzugewonne Funktionalität deutlich überwiegt und im allgemeinen auch benötigt wird.

8.3 Potential

Die beiden Client-Arten haben unterschiedliche Anwendungsgebiete und werden daher auch in den nächsten Jahren beide ihre Existenzberechtigung haben. Ob konventionelle Workflow-Clients einmal vollständig von Web-Clients abgelöst werden, wird in großem Maße von den Fähigkeiten der Clients in Bezug auf „Programmierbarkeit“ und die Integration externer Anwendungen abhängen. Durch die große Popularität des Internet und Web-basierter Anwendungen besitzen Web-Clients allerdings ein sehr großes Potential, welches durch den Einsatz von WfMS in unterschiedlichen Bereichen von Unternehmen, im E-Commerce und speziell im B2C-Bereich noch vergrößert wird.

Kapitel 9

Zusammenfassung und Ausblick

Im Rahmen der Arbeit erfolgte eine Betrachtung aktueller Technologien zur Web-Anbindung von WfMS. Nach einer Analyse der speziellen Anforderungen von WfMS (z.B. Arbeitslisten-Update) wurden die verschiedenen Web-Technologien auf ihre Eignung untersucht. Eine „Ideallösung“ gibt es nicht, sondern es muß in der Regel immer ein Kompromiß zwischen bereitgestellter WF-Funktionalität und der Einschränkung auf bestimmte Web-Clients gefunden werden. *Thin Web-Clients* sind zwar für viele Systemplattformen verfügbar, bieten aber deutlich weniger Funktionalität als *Thick Web-Clients*. Diese basieren häufig auf nicht standardisierten (z.B. DHTML) oder proprietären Technologien (z.B. *ActiveX*) und sind daher wenig portabel.

Auf Serverseite bilden *Web Application Server* eine gute Basis für die Anbindung von WfMS. Sie bieten neben Verfahren zur Generierung dynamischer Web-Inhalte vor allem eine Infrastruktur, welche für den unternehmensweiten Einsatz von Web-basierten WfMS und die Unterstützung vieler Anwender wichtig ist. Dabei spielen Aspekte wie Skalierbarkeit, Transaktionssicherheit oder Mechanismen zur Authentifizierung eine große Rolle.

Der Einsatz von Web-Clients ist nicht in allen Bereichen sinnvoll. Speziell die *Buildtime*-Komponenten des WfMS, welche eine komplexe GUI erfordern und strukturierte Daten zur Beschreibung der Prozeßmodelle mit dem System austauschen, sind für die Realisierung durch einen Web-Client weniger geeignet.

Web-basierte WF-Anwendungen bieten sich vor allem für den Einsatz in heterogenen EDV-Landschaften an. Der Anwender muß keinen speziellen Workflow-Client installieren, sondern nutzt einen (Standard-)Web-Client (der auf den meisten Systemen schon vorhanden ist) für den Zugang zum WfMS. So ist es möglich, externen Anwendern (z.B. Geschäftspartner, Lieferanten, Kunden, usw.) WF-Anwendungen zu eröffnen. Im Bereich *Business-To-Business (B2B)* können Geschäftspartner über einen Web-Schnittstelle des WfMS zum Beispiel einfach und ortsunabhängig den Status von Lieferungen abfragen. Beim *E-Commerce* und im Bereich *Business-To-Customer (B2C)* ermöglichen Web-basierte WfMS die für den Kunden transparente Abwicklung von Geschäftsprozessen (z.B. bei einer Bestellung).

In Zukunft werden die meisten Hersteller von WfMS neben einem konventionellen Workflow-Client auch eine Web-Schnittstelle anbieten, welche einen universellen Zugang zum WfMS für verschiedene Web-Clients und Einsatzgebiete ermöglicht. Welche Technologien sich dabei etablieren werden, ist aufgrund der dynamischen Entwicklungen im Web schwierig vorherzusagen.

Glossar

ASP	Active Server Page
API	Application Programming Interface
B2B	Business-To-Business
B2C	Business-To-Customer
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style-Sheets
(D)COM	(Distributed) Component Object Model
CTM	CORBA Transaction Monitor
DBMS	DataBase Management System
DTD	Document Type Definition
EIS	Enterprise Information System
ERP	Enterprise Resource Planing
EJB	Enterprise JavaBeans
GPO	Geschäftsprozeß-Optimierung
GPR	Geschäftsprozeß-Reegineering
GUI	Graphical User Interface
(X)HTML	(Extensible) HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
JAAS	Java Authentication and Authorization Service
JDBC	Java DataBase Connectivity
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Light-weight Directory Access Protocol

MTS	Microsoft Transaction Server
NIS	Network Information System
OLE	Object Linking and Embedding
OMG	Object Management Group
ORB	Object Request Broker
PDA	Personal Digital Assistant
RMI	Remote Method Invocation
SCM	Supply Chain Management
SOAP	Simple Object Access Protocol
SWAP	Simple Workflow Access Protocol
SSL	Secure Socket Layer
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WML	Wireless Markup Language
XML	Extensible Style Language
XSL	Extensible Style Language

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AAH98] N. Adam, V. Atluri, und W. Huang. Modeling and Analysis of Workflows Using Petri Nets. *Journal of Intelligent Inf. Systems*, 10(2):131–158, März / April 1998.

[Ado01] Adobe. Acrobat Reader 5. http://www.adobe.com/products/acrobat/, 2001.

[AL98] R. Agrawal und F. Leymann. Mining Process Models from Workflow Logs. In *Proc. 6th Internat. Conf. on Extending Database Technol. (EDBT'98)*, S. 469–483, Valencia, März 1998.

[Apa] Apache. Tutorial: Introduction to server side includes.

[Apa01a] Apache. Homepage. http://www.apache.org, 2001.

[Apa01b] Apache HTTP Server Version 2.0. Apache API notes. http://httpd.apache.org/docs-2.0/misc/API.html, 2001.

[App99] D. Appleman. *COM/ActiveX-Komponenten*. Markt & Technik Verlag, 1999.

[App01] Apple. QuickTime 5. http://www.apple.com/quicktime/, 2001.

[Bal01] H. Balzert. *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 2001.

[Bau01] T. Bauer. *Effiziente Realisierung unternehmensweiter Workflow-Management-Systeme*. Dissertation, Universität Ulm, Februar 2001.

[Ben01] A. Bentele. Workflow-Interoperabilitätsstandards für das Internet. Hauptseminar „Basistechnologien zur Realisierung von E-Business-Anwendungen", Abteilung DBIS, Universität Ulm, Januar 2001.

[BGJ+01] S. Bodoff, D. Green, E. Jendrock, M. Pawlan, und B. Stearns. The J2EE Tutorial. Sun Microsystems, 2001.

[BK99] G. A. Bolcer und G. Kaiser. SWAP: Leveraging the Web To Manage Workflow. *IEEE INTERNET COMPUTING*, S. 85–88, Januar / Februar 1999.

[Blo92] J. Bloomer. *Power Programming with RPC*. O'Reilly & Associates, 1992.

[BM98] H. Behme und S. Mintert. *XML in der Praxis*. Addison-Wesley, 1998.

[CCPP98] F. Casati, S. Ceri, B. Pernici, und G. Pozzi. Workflow evolution. *Data & Knowledge Engineering*, 24(3):211–238, Januar 1998.

[Dad96] P. Dadam. *Verteilte Datenbanken und Client/Server-Systeme*. Springer Verlag, 1996.

[DR00] P. Dadam und M. Reichert. Workflow-Management-Systeme. Skript zur gleichnamigen Vorlesung, Universität Ulm, Abteilung Datenbanken und Informationssysteme, Wintersemester 1999/2000.

[EGL98] J. Eder, H. Groiss, und W. Liebhart. The Workflow Management System Panta Rhei. In A. Dogac, L. Kalinichenko, T. Öszu, und A.Sheth, (Edt.), *Workflow Management Systems and Interoperability*. Springer-Verlag, 1998.

[EN93] C. A. Ellis und G. J. Nutt. Modeling and Enactment of Workflow Systems. In *Proc. 14th Conf. on Application and Theory of Petri Nets*, S. 1–16, Chicago, Juni 1993.

[Fis00] L. Fischer. *Workflow Handbook 2001*. Workflow Management Coalition. Future Federal Strategies Inc., 2000.

[Gar99a] Gartner Group. Enterprise Work Management, November 1999.

[Gar99b] Gartner Group. Why E-Business Craves Workflow Technology. T-09-4929, Dezember 1999.

[Gol90] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[Goo98] D. Goodman. *Dynamic HTML – The Definitive Reference*. O'Reilly, Juli 1998.

[Gre88] I. Greif. *Computer-Supported Cooperative Work: A Book of Readings*. Morgan Kaufmann Publ., 1988.

[Gri97] M. Grimm. ADEPT-TIME: Temporale Aspekte in flexiblen Workflow-Managemnt-Systemen. Diplomarbeit, Universität Ulm, 1997.

[GS00] F. Grabmeyer und M. Schimmer. Datenbezogene Standards und Normen im Internet. Seminar Electronic Business, Universität Regensburg, Institut für Wirtschaftsinformatik, Dezember 2000.

[GSSZ99] J. Gulbins, M. Seyfried, und H. Strack-Zimmermann. *Dokumenten-Management – Vom Imaging zum Business-Dokument*. Springer, 1999.

[GT00] V. Gruhn und A. Thiel. *Komponentenmodelle – DCOM, JavaBeans, Enterprise JavaBeans, CORBA*. Addison-Wesley, 2000.

[HC01] J. Hunter und W. Crawford. *Java Servlet Programming*. O'Reilly, 2nd edition, April 2001.

[HL01] G. Himmelein und S. Lennartz. Grundlagen zum Web-Building. *c't Magain für Computertechnik*, (3):130–135, März 2001.

[HM99] C. Hastedt-Marckwardt. Workflow-Management-Systeme – Grundlagen, Standards und Trends. *Informatik Spektrum*, 22(2):99–109, April 1999.

[HPS+00] J. G. Hayes, E. Peyrovian, S. Sarin, M. Schmidt, K. D. Swenson, und R. Weber. Workflow Interoperability Standards for the Internet. *IEEE Internet Computing*, S. 37–45, Mai / Juni 2000.

[Här97] T. Härder. Themenheft Workflow-Management. *Informatik Forsch. Entw.*, 12(2), 1997.

[HS95] M. Hammer und S. A. Stanton. *The Reengineering Revolution - The Handbook*. Harper Collins Publ., 1995.

[IAB96] IABG Industrieanlagengesellschaft mbH Ottobrunn. ProMInanD, 1996.

[IBM01a] IBM. Lotus Notes. http://www.lotus.com/home.nsf/welcome/notes, 2001.

[IBM01b] IBM. MQSeries Workflow Homepage. http://www.ibm.com/software/ts/mqseries/workflow/, 2001.

[IBM01c] IBM. MQSeries Workflow Web Client V2.2, 2001.

[Ill97] T. Illmann. Ein Agenten-basiertes Workflow-Management-System für das World Wide Web. Master's thesis, Universität Ulm, Fakultät für Informatik, Juli 1997.

[Int77] International Standards Organization (ISO). 7498: ISO OSI Basic Reference Model, 1977.

[Int01] IntraWare AG. Kurzbeschreibung BONAPART. http://www.intraware.de, 2001.

[Jab95] S. Jablonski. *Workflow-Management-Systeme - Modellierung und Architektur*. Thomson Publishing, 1995.

[JBS99] S. Jablonski, M. Böhm, und W. Schulze. *Workflow-Management - Entwicklungen von Anwendungen und Systemen*. dpunkt.verlag, 1999.

[JH98] G. Joeris und O. Herzog. Managing Evolving Workflow Specifications. In *Proc. 3rd IFCIS Conf. on Cooperative Information Systems (CoopIS'98)*, New York, August 1998.

[KBB98] P. Kammer, G. A. Bolcer, und M. Bergman. Requirements for Supporting Dynamic and Adaptive Workflows on the WWW, 1998. CSCW'98 Workshop on Adaptive Workflow Sytems.

[Kee01] J. L. Keedy. Objekt- und komponentenorientiertes Programmieren. Skript zur gleichnamigen Vorlesung, Universität Ulm, Abteilung Rechnerstrukturen, Sommersemester 2001.

[KKK+00] Y. Kim, S. Kang, D. Kim, J. Bae, und K. Ju. WW-FLOW: Web-Based Workflow Management with Runtime Encapsulation. *IEEE Internet Computing*, 4(3):55–64, Mai / Juni 2000.

[KMN00] R. Kießling, U. Martschat, und D. Neubert. Dokumentation zum Praktikum: Realisierung einer Web-basierten Benutzerschnittstelle für das Workflow-Management-System ADEPT. Universität Ulm, 2000.

[KS01] A. Kossel und J. Schmidt. Webstreit - Wer hat den schönsten Browser im Land? *c't Magain für Computertechnik*, (19):192–196, 2001.

[Len01] S. Lennartz. Dynamische Seiten - Server Side Includes richtig einsetzen. *c't Magain für Computertechnik*, (20):224–229, 2001.

[LR00] F. Leymann und D. Roller. *Production Workflow*. Prenctice Hall, 2000.

[Mac01] Macromedia. Flash 5. http://www.macromedia.com/software/flash/, 2001.

[Mar01] U. Martschat. Vergleich und Bewertung von Production Workflow-Management-Systemen. Diplomarbeit, Universität Ulm, August 2001.

[Men01] Mentisys. Learn more about pflows workflow engine. http://www.mentisys.com, 2001.

[Mic01a] Microsoft. Homepage. http://www.microsoft.com, 2001.

[Mic01b] Microsoft. Working with DHTML and the DHTML Object Model. http://msdn.microsoft.com/library/en-us/odeopg/html/deovrworkingwithdhtmldhtmlobjectmodel.asp, 2001.

[Mic01c] Microsoft Developer Network. Homepage. http://msdn.microsoft.com, 2001.

[MN98] S. Münz und Nefzger. *HTML 4.0 Handbuch*. Franzis-Verlag, 1998.

[Mon01] R. Monson-Haefel. *Enterprise JavaBeans*. O'Reilly, 2001.

[NCS01] NCSA. HTTP Server Side Includes (SSI). http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html, 2001.

[Net01] Netscape. Browser Plug-Ins. http://home.netscape.com/plugins/, 2001.

[Obe96] A. Oberweis. *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Teubner Verlag, 1996.

[Obj00] Object Management Group. *Workflow Management Facility Specification, V1.2*, April 2000.

[Obj01] Object Management Group. Homepage. http://www.omg.org, 2001.

[Per00] J. Perry. Cross-browser JavaScript . http://www.webdevelopersjournal.com/articles/cross_browser/javascript.html, Juni 2000.

[PHP01] PHP. Homepage. http://www.php.net, 2001.

[PI01] Plug-Ins.de. Die Plug-Ins Datenbank für Audio und Video Systeme. http://www.plug-ins.de/, 2001.

[Ple96] Plexus FloWare Software. Product Summary. BancTec Inc., 1996.

[PW01] C. Plessl und E. Wilde. Server-Side-Techniken im Web - ein Überblick. *iX - Magazin für professionelle Informationstechnik*, S. 88f., März 2001.

[RBD00] M. Reichert, Th. Bauer, und P. Dadam. ADEPT - Realisierung flexibler und zuverlässiger unternehmensweiter Workflow-Anwendungen. In *Proc. Knowledge Engineering, Management and Training (KnowTech'2000)*, Leipzig, September 2000.

[RD98] M. Reichert und P. Dadam. ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems*, 10(2):93–129, März / April 1998.

[RD00] M. Reichert und P. Dadam. Geschäftsprozeßmodellierung und Workflow-Management – Konzepte, Systeme und deren Anwendung. *Industrie Management (Themenheft: Modellierung und Simulation)*, 16(3), Juni 2000.

[Rea01] RealNetworks. RealPlayer 8 Plus. http://www.real.com/player, 2001.

[Rei00] M. Reichert. Workflow-Management-Systeme - Systemkategorien und Abgrenzung. Folien zur Vorlesung Workflow-Management-Systeme, Universität Ulm, Abteilung Datenbanken und Informationssysteme, Wintersemester 1999/2000.

[SBMW98] W. Schulze, C. Bussler, und K. Meyer-Wegener. Standardising on Workflow-Management - The OMG Workflow Management Facility. *ACM SIGGROUP Bulletin*, Volume 19(3), April 1998.

[Sch98a] A.-W. Scheer. ARIS - Modellierungsmethoden, Metamodelle, Anwendungen, 1998.

[Sch98b] W. Schulze. Fitting the Workflow Management Facility into the Object Management Architecture. In D. Patel, J. Sutherland, und J. Miller, (Edt.), *Business Object Design and Implementation II, Proc. of the Third OOPSLA Workshop on Business Object Design and Implementation*, S. 109–117, Atlanta/Georgia, 1998. Springer-Verlag.

[Sch98c] W. Schulze. Implementation and Application of Object-Oriented Workflow Management Systems. In *Addendum to the OOPSLA'98 Proceedings*, Vancouver/Canada, 1998. ACM. First International Workshop on OOWFMS, OOPSLA'98.

[Sch00] W. Schwarz. Browser-unabhängiges DHTML. *iX - Magazin für professionelle Informationstechnik*, S. 40f., Oktober 2000.

[Som92] I. Sommerville. *Software Engineering*. Addison-Wesley, 1992.

[SSP99] E. Siever, S. Spainhour, und N. Patwardhan. *PERL in a Nutshell*. O'Reilly, Januar 1999.

[Sta99] Staffware. Staffware 2000. White Paper, 1999.

[Sta00a] Staffware. Staffware Web Client Technical Guide for Windows NT and Windows 2000, Issue 2, Dezember 2000.

[Sta00b] Staffware. Staffware Web Client User's Guide for Windows NT and Windows 2000, Issue 2, Dezember 2000.

[Sta01] Staffware. Homepage. http://www.staffware.com, 2001.

[Ste96] H. Stempfle. Der Einsatz Offener Systeme in der Praxis - Technologien, Standards, Probleme. Diplomarbeit, Fachhochschule Augsburg, 1996.

[Stu99] S. Stumpp. *Projektmanagement mit MS Project*. Markt & Technik Verlag, 1999.

[Sun00] Sun Microsystems. Developing XML Solutions with Java Server Pages Technology. http://java.sun.com/products/jsp, 2000.

[Sun01a] Sun Microsystems. Enterprise JavaBeans. http://java.sun.com/products/ejb/, 2001.

[Sun01b] Sun Microsystems. Java 2 Platform Enterprise Edition (J2EE). http://java.sun.com/j2ee/, 2001.

[Sun01c] Sun Microsystems. Java Applets. http://java.sun.com/applets/, 2001.

[Sun01d] Sun Microsystems. Java Server Pages. http://java.sun.com/products/jsp/, 2001.

[Sun01e] Sun Microsystems. Java Servlets. http://java.sun.com/products/servlet/, 2001.

[Sun01f] Sun Microsystems. JavaBeans. http://java.sun.com/products/javabeans/, 2001.

[SWA01] SWAP Working Group. Homepage. http://www.ics.uci.edu/ ietfswap/, 2001.

[Tan98] A. S. Tanenbaum. *Computernetzwerke*. Prentice Hall, 1998.

[uID01] Abteilung Datenbanken und Informationssysteme (DBIS). ADEPT - Next Generation Workflow Management System. http://www.informatik.uni-ulm.de/dbis/f&l/forschung/workflow/ftext-adept_e.html, 2001.

[Ult01a] Ultimus. Homepage. http://www.ultimus.com, 2001.

[Ult01b] Ultimus. Ultimus Workflow Suite 4 Product Guide, 2001.

[vdA98] W. van der Aalst. The Application of Petri-Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[Ver00] C. Versteegen. Voll im Trend - Marktperspektiven von Dokumentenmanagement. *IT-Management Dokumentenmanagement Spezial*, S. 14f., 2000.

[Voi00] VoiceXML Forum. VoiceXML 1.0 Specification. http://www.voicexml.org/specs/VoiceXML-100.pdf, März 2000.

[WAP01a] WAP Forum. Homepage. http://www.wapforum.org, 2001.

[WAP01b] WAP Forum and World Wide Web Consortium. WAP Forum - W3C Cooperation White Paper. http://www.w3.org/TR/NOTE-WAP, 2001.

[Web98] M. Weber. *Verteilte Systeme*. Spektrum Akademischer Verlag, 1998.

[WI] M. Weber und T. Illmann. Using Java fot the Coordination of Workflows in the World Wide Web. In *Tagungsband der Fachtagung: Interaktion im Web - innovative Kommunikationsformen*.

[WIS98] M. Weber, T. Illmann, und A. Schmidt. WebFlow: Decentralized Workflow Management in the World Wide Web. In *Proceedings of the 16th IASTED International Conference on Applied Informatics, AI'98*, Februar 1998.

[Wol97] A. Wolpert. Ein ereignisorientiertes Programmiermodell auf der Basis von Workflow-Management-Technologie zur Realisierung integrierter medizinischer Arbeitsplätze. Diplomarbeit, Universität Ulm, 1997.

[Wor95] Workflow Management Coalition. *The Workflow Reference Model (Dokument Nr. TC00-1003)*, Januar 1995.

[Wor98a] Workflow Management Coalition. *A Common Object Model - Discussion Paper (Dokument Nr. TC-1022)*, Januar 1998.

[Wor98b] Workflow Management Coalition. *Workflow and Internet: Catalysts for Radical Change – A WfMC White Paper*, Juni 1998.

[Wor98c] Workflow Management Coalition. *Workflow Audit Data Specification (Dokument Nr. TC-1015)*, September 1998.

[Wor98d] Workflow Management Coalition. *Workflow Client API Specifications (WAPI) (Dokument Nr. TC-1002)*, Juli 1998.

[Wor98e] Workflow Management Coalition. *Workflow Management Application Programming Interface (Interface 2 & 3) Specification (Dokument Nr. TC-1009)*, Juli 1998.

[Wor99a] Workflow Management Coalition. *Process Definition Meta-Model & WPDL (Dokument Nr. TC-1016-P)*, Oktober 1999.

[Wor99b] Workflow Management Coalition. *Terminology and Glossary (Dokument Nr. TC-1011)*, Februar 1999.

[Wor99c] Workflow Management Coalition. *White Paper - Events*, April 1999.

[Wor00a] Workflow Management Coalition. *Workflow Interoperability - Wf-XML Binding (Dokument Nr. TC-1023)*, Mai 2000.

[Wor00b] World Wide Web Consortium. XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0, Januar 2000.

[Wor01a] Workflow Management Coalition. Homepage. http://www.wfmc.org, 2001.

[Wor01b] World Wide Web Consortium. Cascading Style Sheets (CSS) . http://www.w3.org/Style/CSS/, 2001.

[Wor01c] World Wide Web Consortium. Document Object Model (DOM). http://www.w3.org/DOM/, 2001.

[Wor01d] World Wide Web Consortium. Extensible Markup Language (XML) Homepage. http://www.w3.org/XML/, 2001.

[Wor01e] World Wide Web Consortium. HyperText Markup Language Homepage. http://www.w3.org/MarkUp/, 2001.

[Wor01f] World Wide Web Consortium. Jigsaw Server Side Include Commands. http://www.w3.org/Jigsaw/Doc/User/SSI.html, 2001.

[Wor01g] World Wide Web Consortium. Naming and Adressing Homepage. http://www.w3.org/Addressing/, 2001.

[Wor01h] World Wide Web Consortium. Simple Object Access Protocol Homepage. http://www.w3.org/TR/SOAP, 2001.

[Wor01i] World Wide Web Consortium. The Extensible Stylesheet Language (XSL). http://www.w3.org/Style/XSL/, 2001.

[Wor01j] World Wide Web Consortium. XForms Homepage. http://www.w3.org/TR/xforms/, 2001.

[Wor01k] World Wide Web Consortium. XML Schema Homepage. http://www.w3.org/XML/Schema, 2001.

[Wor01l] World Wide Web Consortium (W3C). Hypertext Transfer Protocol Homepage. http://www.w3.org/Protocols/, 2001.

[Wul97] V. Wulf. *Konfliktmanagement bei Groupware*. Vieweg-Verlag, 1997.

[Zah00] R. Zahavi. *Enterprise Application Integration with CORBA - Component and Web-Based Solutions*. OMG Press, 2000.

Index

Erklärung

Name: Ralf Kießling Matrikel-Nr.: 363325

Erklärung

Ich erkläre, daß ich die Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ....
Ralf Kießling