Fraunhofer Institut
Experimentelles
Software Engineering

Anforderungserfassung zum Projekt UNIVERSYS

Entwicklung eines universell einsetzbaren verteilten Systems zur Betriebsführung

Autoren:
Michalis Anastasopoulos
Thomas Forster

„UNIVERSYS“
(BMBF Förderkennzeichen:
01 ISV 32 A - D)

IESE-Report Nr. 171.06/D
Version 1.0
May 2004

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft.
Das Institut transferiert innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach (geschäftsführend)
Prof. Dr. Peter Liggesmeyer
Fraunhofer-Platz 1
67663 Kaiserslautern

Schlagworte: UNIVERSYS

Copyright © Fraunhofer IESE 2006

Inhaltsverzeichnis

Copyright © Fraunhofer IESE 2006

Copyright © Fraunhofer IESE 2006

1 Kurzfassung des Arbeitspakets

Im ersten Arbeitsschritt müssen die grundlegenden Anforderungen und Möglichkeiten deklariert werden, unter denen die Umsetzung des Entwicklungszieles überhaupt gewährleistet werden kann. D.h., welche Erfordernisse ergeben sich aus Anwenderperspektive, und wie kann das System diesen gerecht werden – insbesondere, wenn man verteilte Betriebsumgebungen betrachtet. *UNIVERSYS* ist ein Kriterienkatalog zu Grunde zu legen, der gemeinsam mit den Evaluationspartnern RWE Umwelt Südwest und Stadtwerke Lemgo erarbeitet werden kann. In dieser frühen Projektphase sollen in diesem Zusammenhang vor allem vorhandene Lösungsansätze für die Replikation verteilter Daten sowie deren Konsistenzerhaltung beleuchtet und bewertet werden.

Für die standortunabhängige bzw. mobile Nutzung der Daten ist eine grundlegende Analyse der angestrebten Kommunikationsmodelle hinsichtlich ihrer Einsetzbarkeit, Performance und Stabilität erforderlich. In diesem Zuge müssen die Kriterien für die Datenaustauschformate definiert werden, um eine Datenreplikation und -synchronisation auch über schmalbandbreitige Kommunikationsverbindungen gewährleisten zu können. Darüber hinaus soll in der ersten Projektphase ein grundlegendes Konzept entwickelt werden, wie das Gesamtsystem auch webbasiert umgesetzt werden kann. Neben einer durchzuführenden Bedarfsanalyse hinsichtlich der bevorzugten Hardwareumgebungen stehen die grundsätzlichen Fragen nach dem Zugriffsschutz und der Datensicherheit im Vordergrund.

Copyright © Fraunhofer IESE 2006

2 Kriterienkatalog

Nachfolgend werden Kriterien aufgelistet, die nach Anleitung des Fraunhofer IESE von den Projektpartnern Greengate AG und GEF-RIS AG erfasst wurden. Das Augenmerk liegt auf Eigenschaften der Einsetzbarkeit, Leistungsfähigkeit und Stabilität, die ein Produkt bzw. eine Architektur gemäß dem UNIVERSYS-Entwicklungsziel erfüllen muss. Um die Kriterien übersichtlich zu strukturieren, wurde jeder Anforderungsbereich in kleinere Unterbereiche aufgeteilt.

2.1 Einsetzbarkeit

Anwendungskontext (Eigenschaften bzw. Einschränkungen der zu entwickelnden Anwendung)	1. Clients sollen auch auf mobilen Endgeräten eingesetzt werden können (= keine ständige Netzwerkanbindung) 2. Das System muss in einer Citrix / MS-Terminal-Server Umgebung lauffähig sein 3. Das System muss den Datenabgleich zwischen mehreren Standorten ohne ständige Netzverbindung unterstützen 4. Das Modell muss die unterschiedlichen Attribute und Datensätze der beteiligten Systeme aufeinander mappen können. 5. Die Anwendungen werden für unterschiedliche Systemumgebungen (.Net, JAVA, Win32, PocketPC, Palm) entwickelt. 6. Die Anwendungen sind mehrschichtig (2-Tier oder n-Tier) aufgebaut.
Existierende Infrastruktur (Rolle der existierenden Infrastruktur wie z.B. Entwicklungsumgebung)	1. COM / .NET - Anbindung muss möglich sein 2. Erfahrungen mit C# / VisualStudio.NET sind vorhanden (= bevorzugte Sprache und Entwicklungsumgebung) 3. UML - Tool mit C# / VisualStudio .NET - Integration ist vorhanden (ObjectiF 5.0) 4. Das Modell muss die heterogene Systemumgebung (Windows, Linux, PocketPC, Palm) unterstützen. 5. Das Modell muss mit den eingesetzten Entwicklungsumgebungen (VS.NET, Delphi 5, JAVA) implementierbar sein. 6. Das Modell muss unterschiedliche Datenbankserver und dateibasierte Datenhaltung unterstützen (Oracle, Interbase, MS SQLServer, MySQL, XML)
Kosten (Rolle der Anschaffungskosten)	1. Nach Möglichkeit sollten keine zusätzlichen Lizenzkosten für Software von Drittanbietern entstehen 2. Für Oracle- / Interbase Datenbanken fallen keine zusätzlichen Kosten an,

Copyright © Fraunhofer IESE 2006

	da diese von den beteiligten Einzelsystemen benötigt werden 3. Für den Einsatz des Kommunikationsmodells dürfen keine Runtimekosten für Kommunikationskomponenten (z.B. CORBA) anfallen. 4. Müssen Entwicklungskomponenten lizenziert werden, dann sollten diese mit Quelltext vorliegen. Der Einsatz von Open Source Komponenten ist zu bevorzugen.
Sonstiges (Allgemeine Anforderungen bzgl. Einsetzbarkeit)	1. Offene, anpassbare Schnittstellen -> einfache Erweiterbarkeit des Gesamtsystems durch neue Clients / andere Einzelsysteme (ev. mit bereits vorhandenen Schnittstellen) 2. Der Einsatz des Kommunikationsmodells soll auch für Windows Terminalclients mit nicht eindeutiger IP-Adresse möglich sein. Diese Anbindung muss eventuell über (COM-) Bridges erfolgen, z.B. beim Einsatz IP-basierter Webservices.

2.2 Leistungsfähigkeit

Latenz (Zeit, die ein System braucht, um auf eine Anfrage wie z.B. Klicken eines Knopfs oder Verschicken einer Nachricht zu reagieren)	1. Reaktionszeit auf Funktionsaufrufe zwischen den Einzelsystemen <= 1s (ausgenommen kompletter Grafikaufbau im GIS - dieser kann, abhängig vom gewählten Bildausschnitt und der Netzwerkperformance , erheblich länger dauern)
Durchsatz (Die Anzahl der Operationen, die ein System in einem Zeitintervall ausführen kann)	1. Datenabgleich (10000 Objekte) <= 10 min 2. Da für den Datenabgleich und die Replikation auch schmallbandbreitige Datenverbindungen (ISDN, RS232) verwendet werden, müssen die Datensätze eventuell komprimiert werden um den erforderlichen Datendurchsatz zu erreichen (insbesondere bei XML-/SOAP-Datenübertragung).
Sonstiges (Allgemeine Anforderungen bzgl. Leistungsfähigkeit)	1. Die Performance muss im Multiuserbetrieb erhalten bleiben (o.g. Werte auch bei mind. 20 zeitgleichen Zugriffen) 2. Bei RPC-Aufrufen zur Interaktion zwischen den Benutzeroberflächen der Programme (GIS-Objekt selektieren etc) spielt die Performance eine untergeordnete Rolle.

2.3 Systemstabilität

Verfügbarkeit (Dienstbereitschaft des Systems)	1. Der Datenabgleich muss an Arbeitstagen zwischen 6 und 20 Uhr möglich sein (= garantierte 24h / 7t - Verfügbarkeit ist nicht notwendig) 2. Ein Systemausfall bedingt keinen Datenverlust, da die Einzelsysteme ihre Daten halten.
Ausfallsicherheit (Die Fähigkeit des Systems einen Ausfall zu verkraften)	1. Die beteiligten Systeme müssen unabhängig voneinander weiterarbeiten können (ein Ausfall bleibt - sofern nicht Koppelungsfunktionalität genutzt wird - auf das betroffene System beschränkt)

Copyright © Fraunhofer IESE 2006

	2. Bei Abbruch der Datenübertragung muss diese verlustfrei wiederholt werden können 3. Die Replikations- und Abgleichdaten werden von den beteiligten Systemen gespeichert, die Einzelsysteme arbeiten weitestgehend unabhängig voneinander.
Vertraulichkeit (Inwieweit werden wichtige Daten vor unberechtigtem Lesezugriff oder unberechtigter Veröffentlichung geschützt?)	1. Zugriffsschutz durch Benutzerauthorisierung 2. Direkter Zugriff auf die Koppelungskomponente wird nur für Administratoren ermöglicht 3. Eventuell wird ein Abgleich der Benutzerverwaltungen der Einzelsysteme benötigt
Integrität (Inwieweit werden wichtige Daten vor unberechtigtem Schreibzugriff geschützt?)	1. Rechtevergabe auf Benutzerebene 2. Integritätskontrolle der Daten erfolgt durch die Einzelsysteme unabhängig voneinander. Integritätsregeln werden projektspezifisch abgeglichen
Wartbarkeit (Inwieweit kann das System auch während der Wartung seine Dienste anbieten?)	1. Während der Wartung eines Einzelsystems können die anderen Systeme ungestört weiterarbeiten. 2. Während der Wartung der Koppelungskomponente steht die Koppelungsfunktionalität nicht zur Verfügung.
Sonstiges (Allgemeine Anforderungen bzgl. Systemstabilität)	1. Das Kommunikationsmodell muss zukunftssicher sein, es sollte auf einem system- und herstellerunabhängigen Standard basieren.

2.4 Zusammenfassung

Aus den oberen Tabellen lässt sich schließen, dass die Einsetzbarkeit einer Kommunikations- und Replikationsarchitektur in erster Linie von der unterstützten Heterogenität der Clients, Datenbankserver und Betriebssysteme abhängt. Diese Heterogenität bezieht sich sowohl auf die Lauffähigkeit der zu entwickelnden Produkte als auch auf die Umsetzbarkeit der Architektur.

Bei der Kommunikation der Einzelsysteme mittels Funktionsaufrufe sind gute Reaktionszeiten eine wichtige Anforderung. Darüber hinaus muss der Datenabgleich in einem Replikationsszenario auch bei schmalbandbreitigen Verbindungen schnell erfolgen. Dabei spielt die Skalierbarkeit im Sinne von Mehrbenutzerbetrieb und steigenden Datenmengen eine wichtige Rolle. Demnach muss die Systemperformanz auch bei hoher Belastung akzeptabel sein.

Datenverluste sind weder bei einer unterbrochenen Datenübertragung noch bei einem Systemausfall zulässig. Dies soll durch einen modularen Systemaufbau erzielt werden, der es erlaubt, Ausfälle an konkreten Modulen und unabhängig vom restlichen System zu isolieren. Zudem ermöglicht diese Modularisierung die isolierte Wartung sowie die getrennte Integritätskontrolle einzelner Systeme.

Copyright © Fraunhofer IESE 2006

Die Sicherheit der Datenzugriffe ist von großer Bedeutung. Benutzerautorisierung und -authentifizierung müssen dabei unterstützt werden und die entsprechende Benutzerverwaltung muss auch verteilt, also in replizierten Systemen, möglich sein.

Abschließend kann man folgern, dass die angestrebte Replikationslösung nicht nur Daten verteilt verwalten und abgleichen muss, sondern auch Datenschemata sowie Integritätsregeln.

Was die Anschaffungskosten betrifft, sind Entwicklungsumgebungen mit einem wohldefinierten Lieferumfang vorzuziehen, die keine Zusatzkosten in sich bergen. Daneben ist das Vorhandensein des Quellcodes entweder in lizenzierter oder in Open-Source-Form ein wichtiger Punkt.

Copyright © Fraunhofer IESE 2006

3 Kommunikationsmodelle

Eine große Herausforderung bei der Entwicklung komplexer Enterprise Anwendungen besteht darin, existierende Komponenten innerhalb einer solchen Anwendung zu integrieren. Bei diesen Komponenten handelt es sich oftmals um (Software) Subsysteme verschiedener Hersteller, die auf unterschiedlichen Betriebssystemplattformen laufen und bei deren Implementierung verschiedenartige Technologien eingesetzt wurden. Solche heterogene Systeme, die innerhalb einer Applikation integriert werden müssen sind zum Beispiel verschiedenartige Clients, Datenbanken, Webserver und Application Server. Eine wesentliche Aufgabe bei der Integration dieser Systeme ist die Bereitstellung einer Kommunikationsinfrastruktur für diese verschiedenartigen Subsysteme. Denkbare Kommunikationspartner sind zum Beispiel Webclients und Webserver, Desktopclient (Fatclient) und Datenbank, Webserver und Application Server oder auch unterschiedliche Enterprise Applikationen. Die Art und Weise, wie diese Systeme miteinander kommunizieren wird in einem Kommunikationsmodell beschrieben. Zu einem Kommunikationsmodell gehört die Beschreibung der Systemarchitektur, wobei sowohl spezifische Architekturentwürfe als auch Architekturstile berücksichtigt werden. Weiterhin beinhaltet das Kommunikationsmodell die eingesetzten Techniken und Semantiken zur Realisierung der Kommunikation, das der Kommunikation zugrunde liegende Protokoll sowie das Austauschformat für die Daten. Im Folgenden werden die verschiedenen Teile des Kommunikationsmodells herausgegriffen und eingehender betrachtet.

An dieser Stelle ist anzumerken, dass dieses Kapitel zum Teil aus Ergebnissen des Verbundprojektes Application2Web 47[1] zusammengestellt wurde.

3.1 Architekturstile

Ein Architekturstil bzw. -Muster klassifiziert eine Menge von konkreten, untereinander ähnlichen Architekturen mit Hilfe der für sie typischen Eigenschaften. Architekturstile können auch als Pendant zu Design-Patterns auf Architekturebene angesehen werden.

Bekannte Architekturstile sind zum Beispiel das Layering, bekannt vom OSI – Modell und dem Aufbau von Betriebssystemen, der Client–Server Architekturstil, Objektorientierung und Pipes & Filters. Architekturstile sind miteinander kombinierbar, so findet man bei Webanwendungen eine Kombination aus einer mehrschichtigen Architektur mit einem Client–Server–Stil.

Copyright © Fraunhofer IESE 2006

3.1.1 Client-Server Architekturstil

Beim Client-Server Architekturstil nehmen mehrere Clients die Dienste eines Servers in Anspruch. In dieser Betrachtung werden Clients entsprechend ihrem Einsatzgebiet in Desktop Clients, Web Clients und Mobile Clients unterteilt. Die Komplexität der Clients ist ausschlaggebend für die Dienste, die ein Server zur Verfügung stellen muss. Ein Fat Client implementiert umfangreichere Funktionalitäten als ein Thin Client und muss deshalb nicht im gleichen Maß auf Serverdienste zurückgreifen wie ein Thin Client. So ist ein Web Client typischerweise ein Thin Client, denn er ist auf einen Browser angewiesen und besteht im wesentlichen nur aus einer Eingabemaske und eventuell einigen Skripten, die Benutzereingaben auf der Clientseite validieren. Teilweise ermöglicht der Browser das Caching von Daten und das Abspeichern von Session Informationen. Das Update der Clients erfolgt üblicherweise nur auf Serverseite. Komplexere Funktionalitäten wie Persistenzmechanismen sind grundsätzlich auf einem Server hinterlegt. Die Server Funktionalitäten sind üblicherweise über eine Schnittstelle zugänglich. Ein Fat Client ermöglicht es, komplexere Aufgaben auf der Client Seite zu bewältigen. Üblicherweise ist der Fat Client Browserunabhängig und ist besser den persönlichen Bedürfnissen des Benutzers entsprechend konfigurierbar. Diese Vorteile müssen durch einen größeren Verwaltungsaufwand erkauft werden, denn Fat Clients können nicht auf eine ähnlich einfache Weise wie Thin Clients (Web Clients) an einer zentralen Stelle administriert werden.

Die Effizienz des Client-Server Architekturstils ist in der zentralen Anordnung der Services begründet. Damit lassen sich die Dienste leicht verwalten und auffinden, denn der Server ist bekannt. Dies impliziert jedoch, dass mehrere Clients den Server nutzen und eventuell auch gleichzeitig darauf zugreifen bzw. der Server parallele Zugriffe unterstützen muss. Damit muss das System Skalierbar sein, um Leistungsengpässe zu vermeiden und eventuell muss die Synchronisation paralleler Zugriffe vom Server unterstützt werden. Sollen Sicherheitsaspekte berücksichtigt werden, ist gegebenenfalls ein Autorisierungsmechanismus notwendig.

3.1.2 Peer to Peer Architekturstil

In einigen Anwendungsszenarien ist es vorteilhaft, wenn die Kommunikationspartner innerhalb der Architektur gleichgestellt sind und sowohl Dienste anbieten als auch die Dienste von anderen in Anspruch nehmen. Eine Peer to Peer Architektur unterstützt diesen Ansatz und erlaubt es einem beliebigen Endgerät oder einer Anwendung in einem Netzwerk mit einem anderen Endgerät zu kommunizieren. Dadurch wird eine symmetrische und dezentralisierte Kommunikation ermöglicht. In einer Peer to Peer Architektur registrieren sich die Kommunikationspartner (oder Peers) mit ihren Diensten im System. Durch die Registrierung können diese Dienste von anderen Peers aufgefunden und genutzt werden. Peer to Peer Netzwerke sind besonders ausfalltolerant, da Services re-

Copyright © Fraunhofer IESE 2006

dundant zur Verfügung stehen, dagegen stellt das Auffinden von Diensten und die Kommunikation zwischen den Peers im Vergleich zu einer Client Server Architektur eine besondere Herausforderung dar. Typischerweise kommen hier Multicast Techniken und Pipes zum Einsatz. Bekannt geworden sind Peer to Peer Netzwerke durch diverse Musiktauschbörsen, außerdem finden sie ihren Einsatz beispielsweise im Bereich der Kommunikation mobiler Endgeräte (Pervasive Computing), Web Services und File Sharing. Bekannte Peer to Peer Standards sind JXTA, UPnP und Jini.

3.1.3 Layering

Das Layering als weiterer Architekturstil findet seinen Einsatz häufig in Architekturen verteilter Systeme. Dabei werden logischen Systemkomponenten, die auch über verschieden physikalische Systeme verteilt sein können, in Schichten gruppiert. Die Kommunikation zwischen den Schichten erfolgt über spezifizierte Schnittstellen. Im Gegensatz zu einem monolithischen System wird es dadurch leichter, einzelne Komponenten herauszulösen um sie auszutauschen, ohne dass Subsysteme in anderen Schichten verändert werden müssen. Eine Schichtung impliziert eine Modularisierung der Anwendung und vereinfacht damit die Wiederverwendung. Außerdem kann eine Schichtung dazu verwendet werden, den direkten Zugriff auf Services von unteren Schichten zu verhindern. (Vor allem im Webbereich existieren typische Schichtenarchitekturen.)

Zweischichtige Architekturen ermöglichen den Zugriff von einem Client direkt auf ein Backend System, zum Beispiel eine relationale Datenbank. Hier muss der Client neben der Visualisierung der Daten auch die Logik zur Datenbankabfrage realisieren. Diese Art des Zugriffs ist zwar performant, allerdings haben Änderungen in der Datenbank Auswirkungen auf die Client Implementierung. Synchronisationsmechanismen, Autorisierungsmechanismen, Session Handling und weitergehende transaktionale Funktionalitäten, die über die Fähigkeiten der Datenbank hinausgehen müssen ebenfalls in der Client Schicht umgesetzt werden oder fallen ganz weg.

Dreischichtige Architekturen bestehen in der Regel aus einem Thin Client, einem Webserver und/oder Applicationserver, und einem Backendsystem. Die mittlere Schicht kapselt die Kommunikation mit dem Backendsystem und bietet üblicherweise die schon genannten Services zu Transaktion, Autorisierung und Session Handling. N-schichtige Systeme bauen auf dem dreischichtigen Modell auf und bieten Services wie Clustering zur Perfomancesteigerung und realisieren die damit verbundenenTechniken zur Konsistenthaltung der Daten und zum Lastausgleich. N-Schichtige Architekturen werden für Enterprise Anwendungen eingesetzt. Diese Anwendungen realisieren umfangreiche fachliche Logik, müssen einen hohen Durchsatz an Transaktionen bewältigen und die Möglichkeit bieten eine große Anzahl von Benutzern zu verwalten. Die Anforderungen für diese Anwendungen sind stabil und bekannt, es haben sich typische Architekturmodelle in diesem Bereich durchgesetzt.

Copyright © Fraunhofer IESE 2006

Abbildung 1 zeigt die mehrschichtige Architektur einer Enteprise Applikation, wie sie im Rahmen dieser Anforderungen häufig eingesetzt wird.

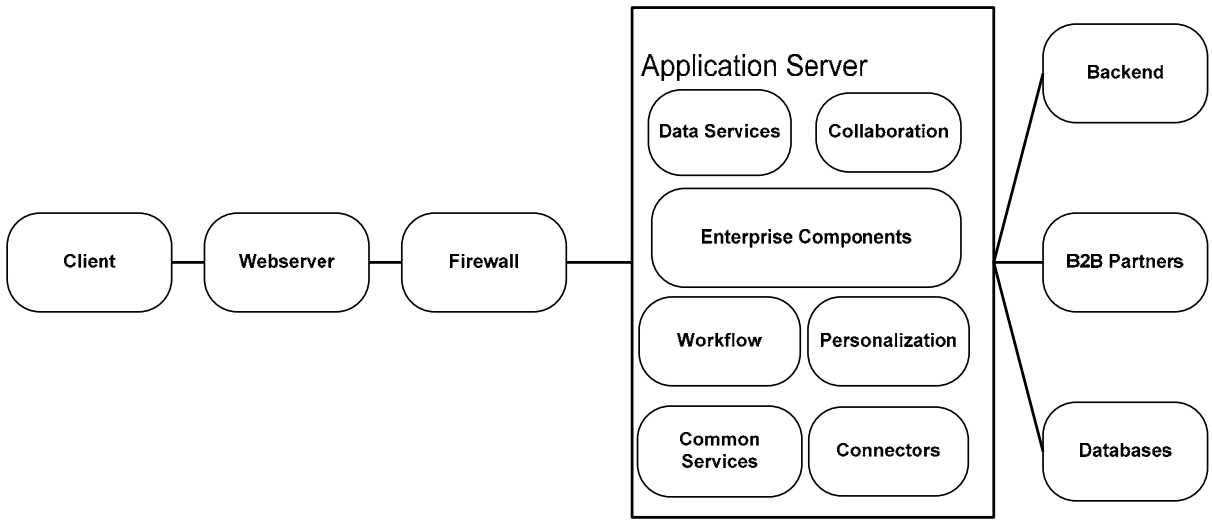


Abbildung 1: Enterprise Architektur

Die verwendeten Systeme haben sich bewährt und die eingesetzten Technologien sind erprobt. Eine zentrale Komponente dieser Architektur ist der Application Server. Der Application Server bietet den darüber liegenden Schichten eine Reihe von Services an, die den Entwicklungsaufwand erheblich reduzieren können. Üblicherweise folgen die Implementierungen der verschiedenen Application Server einheitlichen Standards. Diese Standards spezifizieren neben Sicherheitsmechanismen, Transaktionsmanagement und Clustering auch Naming Services zum Auffinden von Diensten in einer Peer to Peer Umgebung.

Im Bereich der Enterprise Anwendungen existieren speziell im Middlewarebereich eine Vielzahl von Lösungen, viele davon sind Open Source.

Microsoft:

- IIS (Internet information Server)
- .NET
- DCOM/COM

Java:

- J2EE Application Server: IBM Websphere, BEA, JBoss (Open Source)
- LAMP (Linux, Apache, MySQL, PHP) Open Source

Python:

- Zope Application Server

Copyright © Fraunhofer IESE 2006

Spezielle Middlewarelösungen:

- CORBA-basiert: Orbix, Voyager
- SAP
- Crossworlds
- Messaging Systeme: IBM MQSeries, BEA Tuxedo

3.1.4 XML-Pipeline

Abbildung 2 zeigt eine dokumentenorientierte XML-Pipeline, wie sie auch beim Client Server und Peer to Peer Architekturstil eingesetzt wird.

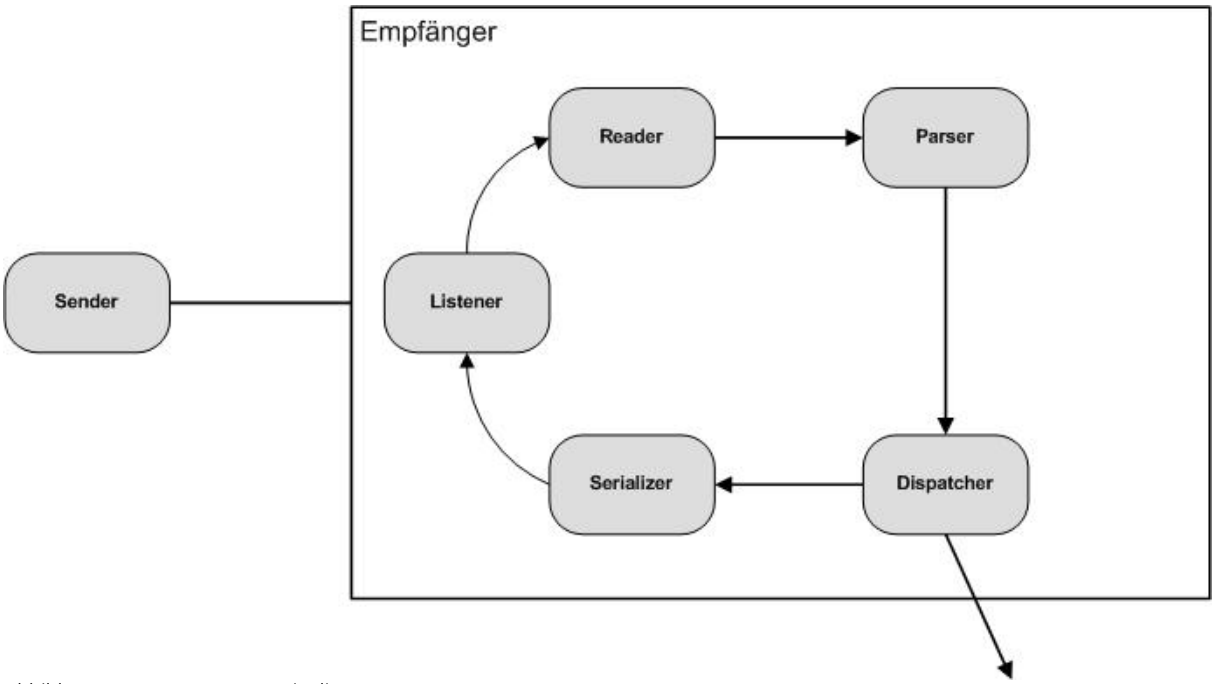


Abbildung 2: XML-Pipeline

Ein textbasierter Request (XML-Format) wird geparst, interpretiert und die entsprechenden Anfragen werden an andere Teile des Systems verschickt (dispatch). Das Ergebnis wird serialisiert und als Response an den Sender zurückgeschickt. Eine Service-orientierte Architektur (SOA) verwendet das Pipeline-Modell als grundlegenden Mechanismus zur Bearbeitung von Serviceanfragen. Da eine SOA im Wesentlichen auch einem Peer to Peer Architekturstil folgt sind weitere Dienste und Protokolle notwendig, um die angebotenen Services auffinden und nutzen zu können. Die Web Services Description Language (WSDL) dient zur Beschreibung der angebotenen Dienste und ermöglicht deren teilweise automatisierte Nutzung mit Hilfe eines Webservices Frameworks wie zum Beispiel AXIS. Universal Description, Discovery, and Integration (UDDI) liefert die

Copyright © Fraunhofer IESE 2006

Spezifikation für einen Webservices Verzeichnisdienst der es Anwendungen ermöglicht, ihre Dienste zu registrieren bzw. Dienste anderer aufzufinden und dynamisch zu nutzen.

Innerhalb der SOAs sind verschieden Szenarien bei einer Kommunikation zwischen den Endpunkten denkbar. Im einfachsten Fall handelt es sich lediglich um einen dokumentenbasierten Nachrichtenaustausch zwischen Anwendungen, komplexere Szenarien beinhalten den Aufruf von Remote Procedures. Nachrichtigen können bestätigt und verschlüsselt werden. Das Beilegen von Binärdaten ist ebenfalls möglich. Weiterhin werden Routing und Quality of Service Angaben unterstützt.

Eine Pipeline Architektur findet sich auch im Webpublishing Framework Cocoon von Apache wieder. Cocoon wird als Teil einer Client Server Architektur eingesetzt und dient nicht zur Kommunikation zwischen verschiedenen Anwendungen oder Anwendungsteilen wie sie bei den Service Orientierten Architekturen beschrieben wurde. Vielmehr geht es hier darum dynamisch XML-Daten in ein Format zu transformieren das auch Layoutangaben umfasst. Der Client ist hier ein beispielsweise ein Web Browser oder ein Mobile Client der zur Visualisierung von Daten für einen Endbenutzer dient. Bei einer Anfrage handelt es sich um einen einfachen HTTP-Get oder -Post Request. Dieser wird ausgewertet, die erforderlichen Daten werden aus einer Datenbank ermittelt und, falls noch nicht geschehen, in ein XML-Format gebracht. In Abhängigkeit vom Typ des anfragenden Clients wird daraus beispielsweise eine HTML- WML- oder PDF-Repräsentation erzeugt. Dieses Dokument wird dann als Response zum entsprechenden Browser zurückgeschickt.

3.2 Techniken

Neben den eingesetzten Architekturstilen ist die Beschreibung der eingesetzten Techniken zur Handhabung und Umsetzung bestimmter kommunikationsspezifischer Anforderungen innerhalb einer Enterprise Anwendung ebenfalls Teil des Kommunikationsmodells.

3.2.1 Patterns

Hat sich für die Umsetzung einer Anforderung eine bestimmte Technik bewährt und kann sie beim Design von Anwendungen in einem ähnlichen Kontext eingesetzt werden, so spricht man von einem Entwurfsmuster oder Design Pattern. Eine Vielzahl von Entwurfsmustern wurde in [3] gesammelt. Visitor, Singleton, Factory und State sind bekannte Patterns aus diesem Katalog und sind zunächst auf die Domäne der Objektorientierten Programmierung beschränkt. Muster existieren aber auch in den anderen Bereichen, so haben sich beispielsweise im Bereich der Enterprise Architekturen wiederkehrende Lösungen für typische e-

Copyright © Fraunhofer IESE 2006

tabliert. Sun bietet in seinen Java Blueprints einen Pattern Katalog für J2EE basierte Lösungen (java.sun.com/blueprints/patterns). Diese Muster beschreiben hauptsächlich Techniken, die Probleme der Skalierbarkeit, Performance, Ressourcenschonung und Verfügbarkeit behandeln. Application Server setzen solche Muster (siehe folgende Tabellen) ebenfalls um und bieten darauf aufbauend konfigurierbare Dienste an, die es beispielsweise ermöglichen, eine Anwendung zu skalieren.

Caching/Staging
Problem: Eine Klasse von Objekten •ist aufwändig zu erzeugen •verbraucht nach dem Erzeugen relativ wenig Ressourcen •wird evtl. von mehreren Clients genutzt •wird vor allem lesend genutzt
Lösung: Richte einen Zwischenspeicher ein, in den bereits erzeugte Objekte abgelegt werden können. Richte einen Mechanismus ein, der obsolet gewordene oder selten genutzte Objekte wieder entfernt.
Beispiele: Trennung von Redaktionsserver und Live-Server in CMS.

Tabelle 1: Caching Pattern

So kann die Skalierbarkeit und Performanz einer Anwendung durch Caching und Pooling Mechanismen verbessert werden da der Aufwand für die teure Objekterzeugung verringert wird. Tabelle 1 und Tabelle 2 beschreiben ein Lösungsmuster für Probleme die im Zusammenhang mit der Objekterzeugung bestehen.

Pooling
Problem: Eine grosse Anzahl von Clients benötigt jeweils ein Objekt einer bestimmten Klasse. Dieses Objekt ist aufwändig zu erzeugen und/oder verbraucht nach dem Erzeugen viel Ressourcen
Lösung: Richte einen Pool von Objekten ein, den alle Verbindungen nutzen. Erzeuge ggf. einen Mechanismus, der den verbindungsbezogenen Zustand eines Objekts schnell extrahieren und wiederherstellen kann.
Beispiele: Datenbankverbindungen Entity Beans

Tabelle 2: Pooling Pattern

Copyright © Fraunhofer IESE 2006

Ein weiteres Muster dass die Ressourcenschonung unterstützt ist die verbindungslose Schnittstelle. Tabelle 3 zeigt das Muster, wonach Anwendungsteile, die verbindungsbezogen und zustandsbehaftet sind sowie Anwendungsteile, die an keine Session und an keinen Zustand gebunden sind, in unterschiedlichen Komponenten realisiert werden sollten.

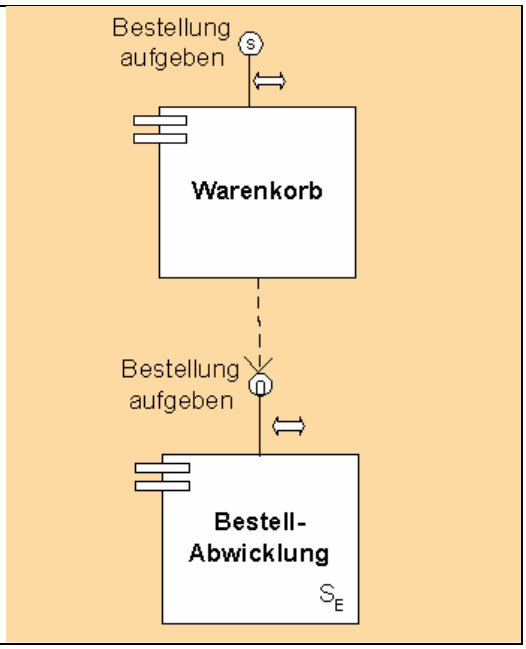
Verbindungslose Schnittstelle	
Problem: Eine Komponente soll viele Clients bedienen können, mit Ressourcen sparsam umgehen und gut replizierbar sein.	
Lösung: Entwerfe die Schnittstelle der Komponente nach Möglichkeit so, dass sie keinen verbindungsbezogenen Zustand (Conversational State) halten muss. Erzeuge, wenn nötig, eine separate Komponente, die nur den verbindungsbezogenen Zustand	
Beispiele: Bestellungen – Warenkorb-Logik	

Tabelle 3: Verbindungslose Schnittstelle

Die obige Notation wurde im Rahmen des Application2Web-Projektes eingeführt [1]. Die Schnittstelle der Warenkorbkomponente ist zustandsbehaftet und wird deswegen mit einem S notiert. Auf der anderen Seite ist die Komponente zur Bestellabwicklung zustandlos und deswegen wird ihre Schnittstelle mit einer Null notiert. Das Symbol S_E in der unteren rechten Ecke der Bestellabwicklungskomponente stellt die Bezugskategorie der Daten dar, die in diesem Fall unternehmensbezogen sind (S_E=Enterprise State). Darüber hinaus symbolisiert der Pfeil ⇔, dass die Operation „Bestellung aufgeben“ jeweils bidirektional und synchron abläuft (siehe nächsten Absatz).

3.2.2 Kommunikationsarten

Die Kommunikation zwischen Teilen der Anwendung kann asynchron oder synchron erfolgen. Bei einer synchronen Koppelung schickt der Client eine Anfrage an den Server und blockiert die weitere Programmausführung, bis der Server eine Antwort geliefert hat. Bei einer asynchronen Kopplung wird der Programmfluss nicht blockiert, denn es wird nicht gewartet, bis der Rückgabewert einer Operation vorliegt. Für die beiden Techniken existieren leicht unterschied-

Copyright © Fraunhofer IESE 2006

liche Varianten. Bei einer "synchronous one-way" Kommunikation sendet der Client lediglich eine Nachricht an den Server. Die „synchronous two-way" Kommunikation bezeichnet eine Kommunikationsart, bei der ein Client eine Response auf seinen Request erwartet. Dabei blockiert der Programmfluss des Clients bis die Serverantwort eintrifft. Werden die Requests in nur einem Thread generiert blockiert die gesamte Client Applikation. Die "deferred synchronous" Kommunikation erlaubt es einem Client, das Ergebnis der Operation zu einem späteren Zeitpunkt abzufragen, wobei der Thread bei der Ergebnisabfrage auch hier blockiert, falls das Ergebnis zu diesem Zeitpunkt noch nicht vorliegt.

Bei einer asynchronen Koppelung kann der Client in jedem Fall mit der Programmausführung fortfahren. Üblicherweise sind Client und Server über einen Callback Mechanismus verbunden und der Server meldet sich beim Client zurück, wenn das Ergebnis vorliegt. Ein entscheidender Vorteil der asynchronen Koppelung ist die Ressourcenschonung, weil nicht für jede Anfrage ein Thread gestartet werden muss wenn verhindert werden soll, dass die Anwendung blockiert. Tabelle 4 veranschaulicht die Arten, Einsatzmöglichkeiten und Vorteile von verschiedenen Arten der asynchronen Kopplung.

Copyright © Fraunhofer IESE 2006

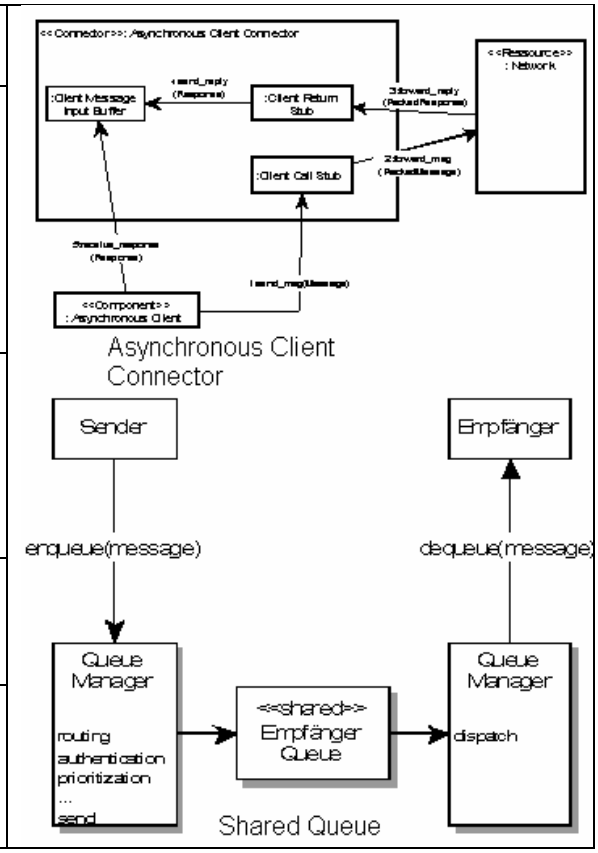
Asynchrone Koppelung	
Problem: Eine Anwendung greift auf verteilte Subsysteme zu. Es sollen geringe Antwortzeiten, hoher Durchsatz und trotz der gelegentlichen Wartung einzelner Subsysteme eine hohe Zuverlässigkeit erreicht werden.	
Lösung: Verwende eine Message-orientierte Middleware (MOM) und kopple die Subsysteme damit asynchron.	
Beispiele: Asynchronous Client Connector, Shared Queue	
Risiken: Komplizierte Fehlerbehandlung	

Tabelle 4: Asynchrone Koppelung

3.2.3 Aufrufsemantiken

In Abhängigkeit von der geforderten Zuverlässigkeit der Client-Server Kommunikation (Quality of Service) existieren unterschiedliche Aufruf Semantiken. Bei der „maybe"-Semantik handelt sich um eine unzuverlässige Art der Kommunikation. Es wir lediglich garantiert, dass die Operation bzw. Anfrage maximal einmal ausgeführt wird. Dabei erfolgt keine Rückmeldung über Erfolg oder Misserfolg der Operation. Bei einer „at-least once"-Semantik ist die Obergrenze für die Anzahl der Anfrageausführungen prinzipiell nicht festgelegt. Diese Semantik ist für idempotente Operation wie lesende Dateizugriffe gut geeignet.

Eine „at-most-once"-Semantik garantiert die Atomizität einer Operation, das heißt sie wird genau null oder einmal ausgeführt. Ein Misserfolg wird dem Aufrufer mitgeteilt, und er kann entsprechend darauf reagieren. Außerdem ist es gewährleistet, dass die Operation nicht nur teilweise ausgeführt wurde. Für Anwendungen die hohe Zuverlässigkeitsanforderungen besitzen wird die „at-most once"-Semantik zu einer „exactly-once"-Semantik erweitert. Ein verteiltes Transaktionskonzept und die Berücksichtigung des Wiederanlaufs ausgefallener

Copyright © Fraunhofer IESE 2006

Komponenten garantieren in jedem Fall, dass eine Operation genau einmal ausgeführt wird.

3.3 Datenaustauschformate und Protokolle

Formate geben an, auf welche Art und Weise Daten, die zwischen den Kommunikationspartnern ausgetauscht werden, strukturiert sind. Formate können textbasiert oder binär codiert sein. Beispiele für textbasierte Formate sind XML, HTML, cHTML oder WML. Binärformate codieren teilweise textbasierte Formate, z.B. Base64 und UNICode oder werden für andere Dateitypen verwand, beispielsweise JPG, Zip-Dateien, NDR und CDR.

Die Regeln, wie die Daten zwischen Kommunikationspartnern ausgetauscht werden, ist in einem Protokoll spezifiziert. Protokolle sind häufig an ein Format gebunden, so arbeitet das http-Protokoll mit dem RFC822 Format und die der CORBA Technologie zugrunde liegenden Protokolle verwenden zum Datenaustausch das CDR-Format. Beispiele für Protokolle sind neben den bereits genannten SMTP, HTTPS, OBEX, WAP, i-Mode, IIOP und DCOM. Protokolle können verbindungslos oder verbindungsorientiert sein. Ein verbindungsloses Protokoll ist zum Beispiel UDP (User datagramm Protokoll). TCP (Transport Control Protocol) und das darauf basierende http-Protokoll sind dagegen verbindungsorientiert. Das bedeutet, dass vor dem Versenden der Daten eine (virtuelle) Verbindung zum Zielsystem aufgebaut wird, und somit die Erreichbarkeit sichergestellt wird während UDP ohne Verbindungsaufbau arbeitet, sondern die Daten einfach versendet. http ist außerdem ein Beispiel für ein zustandsloses Protokoll, da die einzelnen Übertragungen und Befehle nicht voneinander Abhängen. Allerdings lassen sich über zustandslosen Protokollen zustandsbehaftete Protokolle implementieren. Im einfachsten Fall werden im Fall von http bei der Kommunikation vom Server Session-Ids in die URL codiert oder Cookies verschickt um den Client identifizieren zu können.

XML setzt sich als Datenaustauschformat mehr und mehr durch. Dafür lassen sich mehrere Gründe anführen. Daten lassen sich mit Hilfe eines XML-Formats leicht strukturieren. Da es textbasiert ist sind diese Daten auch für den Menschen gut lesbar. XML ist flexibel, so lassen sich durch den Schema und DTD Mechanismus ohne großen Aufwand eigene Sprachen auf Basis von XML definieren. Die Validierung der Dokumente gegen diese Grammatiken erfolgt durch existierende Parser. Generell sind Tools zur Bearbeitung und Validierung von XML-Dateien weit verbreitet und vielfach kostenlos. Weiterhin ist XML standardisiert und plattformunabhängig. Es eignet sich somit zur Kommunikation in einer heterogenen Systemumgebung. Durch das Format wird eine klare Trennung der Zuständigkeiten ermöglicht, denn XML repräsentiert nur Daten und beinhaltet keinerlei Informationen zur Darstellung. Die Darstellung der Daten kann also völlig unabhängig von der Bereitstellung erfolgen. Allerdings hat das textbasierte Datenformat auch Nachteile. Es ist weniger ressourcenschonend als

Copyright © Fraunhofer IESE 2006

ein Binärformat und auch nicht für Binärdaten geeignet. Die bereits erwähnten Mechanismen zur Validierung bzw. zur Definition einer Grammatik sind im Fall der DTDs ungenügend, Schema-Definitionen sind dagegen unhandlich und kompliziert.

3.3.1 Lösungen für schmalbandige Kommunikation

Für den Datenaustausch mit und zwischen mobilen Endgeräten, bei denen die Kommunikation hauptsächlich über die Luftschnittstelle erfolgt, existieren Lösungen, welche die speziellen Eigenschaften dieser Geräte berücksichtigen. Als Datenaustauschformat wird die Wireless Markup Language (WML) eingesetzt. Mit WML können Hypertext-Informationen für Geräte mit kleinen Displays definiert werden, also für Mobil-Telefone, PDAs und dergleichen. WML erfüllt eine ähnliche Funktion wie HTML, das sich besser für Geräte mit größeren Displays eignet, also für Notebooks, Desktop-PCs, Workstations, Fernsehapparate und dergleichen. WAP (Wireless Application Protocol) ist das zugrundeliegende Protokoll, mit dem die WML-Daten von den mobilen Endgeräten abgefragt und zu diesen übermittelt werden. Die WAP-basierte Kommunikation findet zwischen dem mobilen Endgerät und einem WAP-Gateway statt. Der WAP-Gateway selbst dient als Proxy zwischen dem Endgerät und einem Webserver und kommuniziert mit dem Server über das http-Protokoll.

Ein weiteres Format, das speziell auf die Bedürfnisse von Endgeräten mit kleinen Displays abgestimmt ist Compact HTML (cHTML). Das zugrundeliegende Protokoll zum Austausch dieses Formats ist iMODE.

Auf Ebene der Typischerweise kommunizieren mobile Engeräte über Protokolle wie GPRS, Infrarot, Bluetooth, IEEE802.11 (Wireless LAN) und USB.

In der der Domäne der mobilen Internet Technologie definiert Openwave eine einheitliche Plattform Architektur.

Copyright © Fraunhofer IESE 2006

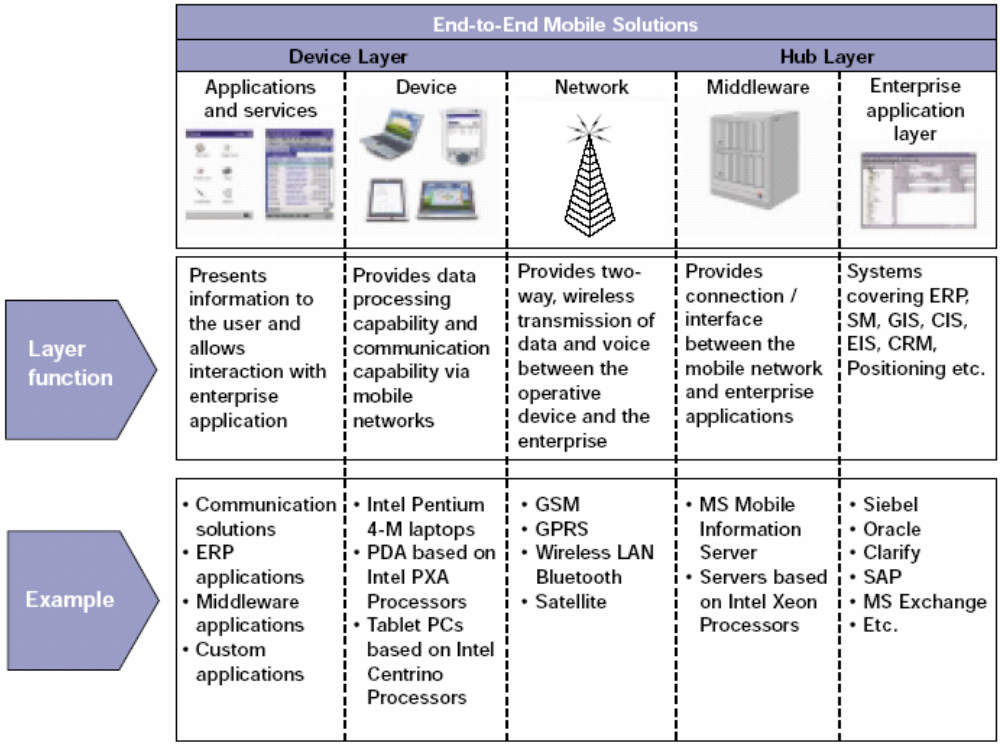


Abbildung 3: Allgemeine Architektur einer Mobile Client Lösung (Quelle: [4])

Abbildung 3 zeigt beispielhaft die Hardware und Software Infrastruktur, die benötigt wird um mobile Endgeräte mit einer Enterprise Anwendung im Backend zu verbinden.

Copyright © Fraunhofer IESE 2006

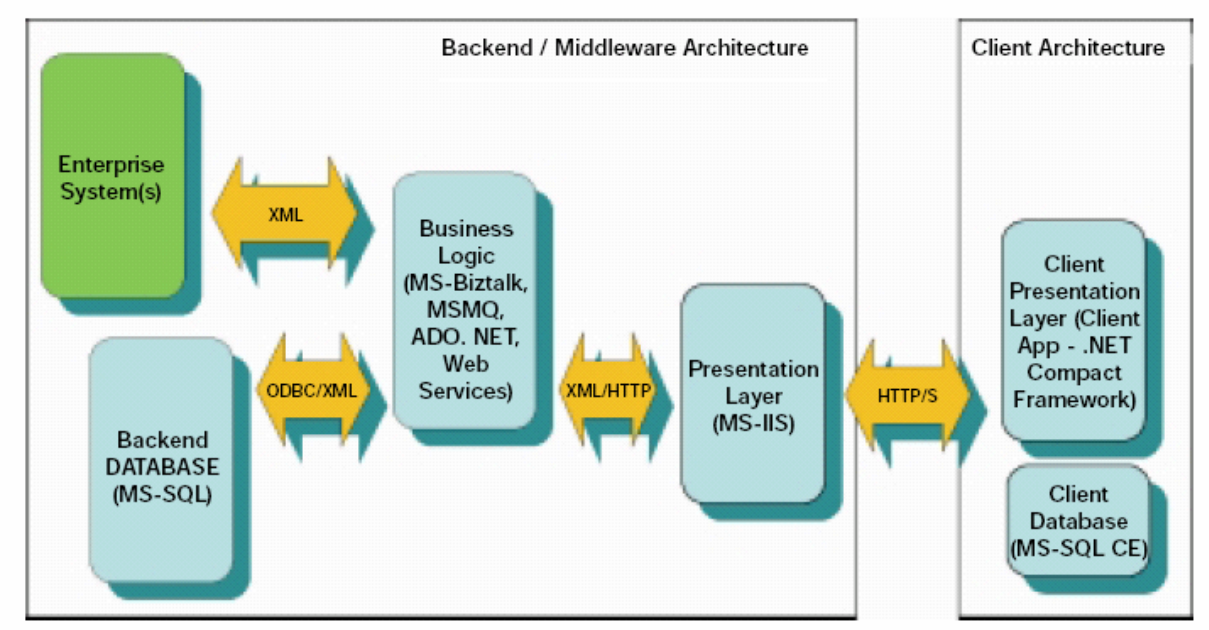


Abbildung 4: Microsoft basierte mobile Client Lösung (Quelle: [4])

Diese Architektur kann für konkrete Software Technologien in den unterschiedlichen Layers weiter spezialisiert werden. Abbildung 4 zeigt die eine entsprechende Software Architektur, wie sie mit Microsoftprodukten realisiert werden kann.

In Abbildung 5 ist die vergleichbare Java-basierte Lösung zu sehen.

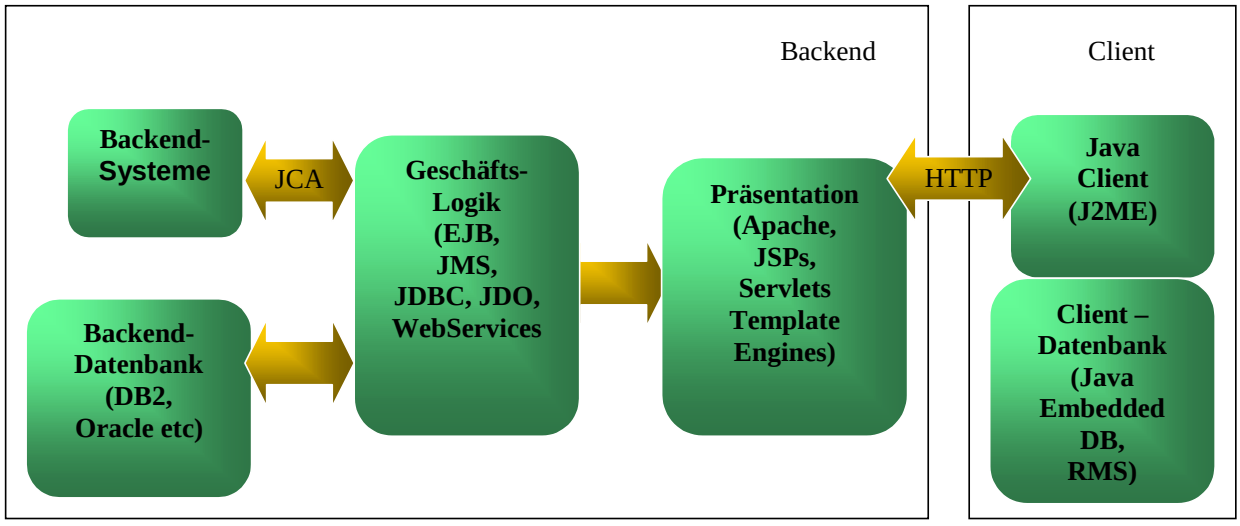


Abbildung 5: Java basierte mobile Client Lösung

Copyright © Fraunhofer IESE 2006

Im Backendbereich existieren innerhalb der beiden Architekturen keine wesentlichen Unterschiede zu einer Architektur, in der keine mobilen Clients bedient werden. Sollte es sich um eine Webbasierte Darstellung von Daten handeln, müssen bei deren Aufbereitung eventuell Einschränkungen bei den Möglichkeiten zur Darstellung berücksichtigt werden.

Die Client Architektur bietet bei beiden Technologien vergleichbare Lösungen. Unter Microsoft erfolgt die Programmierung eines Clients mit dem .NET Compact Framework. Java greift auf eine Untermenge des Standardsprachumfangs zurück, die Java 2 Microedition. Zusätzlich ist es erforderlich, dass auf dem mobilen Gerät eine virtuelle Maschine zur Interpretation des Java Byte Codes installiert ist.

Um Daten auf der Client Seite speichern zu können, bietet Microsoft eine MS-SQL Datenbank für das CE Betriebssystem. Es existieren noch weiter Datenbanken für unterschiedliche Client Betriebssysteme, zum Beispiel von Sybase, Oracle, Poet und IBM. Die Nutzung dieser Datenbanken ist auch aus einer Java Anwendung heraus möglich, die auf einem mobilen Client läuft. Zusätzlich bietet Java in der J2ME ein einfaches Record-basiertes Speichersystem.

3.4 Integration von Legacy Systemen

Die Integration von Microsoft Technologien beziehungsweise von Legacy-Anwendungen, die diese Technologien einsetzen, wird durch die .NET Plattform unterstützt. Im Application-To-Application (A2A) Bereich existieren mehrere Lösungen So können ActiveX-Steuerelemente direkt in die .NET Umgebung importiert werden. COM-Applikationen können entweder durch Early-Binding über den Import von Type Libraries integriert werden oder durch Late-Binding über das .NET Reflection-Framework. Sollen Windows Anwendungen, die Microsoft Foundation Classes verwenden integriert werden, kann über Platform Invoke (P/Invoke) auf DLLs zugegriffen werden. Für CORBA-basierte Lösungen existieren spezielle Brückenlösungen zur Integration, zum Beispiel IIOP.Net. Im Business-To-Business (B2B) Bereich ist die Integration externer, oftmals heterogener Enterprise Anwendungen nötig. Microsoft bietet hier als Integrationsplattform den Biztalk-Server. Andere Integrationslösungen, die nicht Microsoft spezifisch sind, umfassen Webservices und Workflow Management Systeme.

Java bietet im Bereich A2A ebenfalls umfangreiche Integrationslösungen für Legacy-Systeme. COM-Anwendungen und ActiveX-Controls können über spezielle Brückenlösungen integriert werden. Die Kommunikation mit nativen, Betriebssystemspezifischen Anwendungen kann über Sockets und Java Native Interfaces (JNI) erfolgen.

Im Bereich B2B bietet auch Java diverse Integrationslösungen an. Der J2EE Standard spezifiziert eine Unterstützung zur Kommunikation mit Legacy Syste-

Copyright © Fraunhofer IESE 2006

men mit Hilfe der J2EE Connector Architecture (JCA), die auch den Einsatz von Webservices vorsieht. Application Server wie JBoss bieten bereits eine Implementierung dieses Standards an. Eine Integration kann zusätzlich auch über Workflow Management Tools geschehen.

	COM	CORBA	J2EE	SOAP	.NET
COM		Bridge	Java Plugin SOAP Bridge	.NET SOAP Toolkit	Regasm Tool
CORBA	Bridge		RMI/IIOP JAVA IDL	CORBA2SOAP Brige	Bridge
J2EE	MS Java SOAP Bridge	RMI/IIOP JAVA IDL Bridge SOAP		JAX RPC	SOAP
SOAP	.NET SOAP Toolkit	SOAP2CORBA Bridge	JAX RPC		Standard
.NET	Tlblmp Tool	Bridge	SOAP	Standard	

Tabelle 5: Integration verschiedener Technologien

Tabelle 5 gibt einen Überblick darüber, wie die genannten Technologien integriert werden können.

3.4.1 Enterprise Service Bus

Der Enterprise Service Bus (ESB) gilt als eine vielversprechende Architektur zur unternehmensweiten Systemintegration. Die Architektur basiert auf einem lose gekoppelten und nachrichtenbasierten Kommunikationsmodell, so wie es aus älteren Messaging-Systemen wie BEA Tuxedo oder IBM MQSeries bekannt geworden ist. Allerdings wird dieses Modell um die Unterstützung von Web Services sowie Transformations- und Routing-Services ergänzt, wobei XML als gemeinsames Datenaustauschformat zum Einsatz kommt. Die folgende Tabelle bietet einen Überblick über die Dienste eines Enterprise Service Bus und die damit verbundenen Standards.

Copyright © Fraunhofer IESE 2006

Service	Standards
Transport	HTTP
Nachrichten	SOAP
Service-Beschreibung bzw. -vertrag	WSDL
Verzeichnisdienst	UDDI
Zuverlässigkeit	HTTPR, WS-Reliability, ebXML Messaging Service
Benutzeroberfläche	WSUI, WSIA, WSRP
Verschlüsselung	XML Encryption, WS Security
Signierung	XML digital Signature, WS Security
Einmalige Anmeldung (Single Sign On)	SAML
Zugriffskontrolle	XACML
Transaktionen	BTP, XAML, WS-Transaction, WS-Coordination
Verwaltung öffentlicher bzw. privater Schlüssel	XKMS (XKISS, XKRSS)
Prozessdarstellung	BPEL, BPML, XLANG, WSFL, WSCI, WS Choreography
Administration	OMI
XML-Dialekte für Geschäftsnachrichten	fpML, CML, etc
XML-Dialekte für Geschäftsprozesse	RosettaNet, etc

Tabelle 6 ESB-Dienste

Die ESB-Initiative versucht allgemeingültige Schwächen existierender Integrationslösungen zu beseitigen, wie beispielsweise die Anpassungsschwierigkeit aufgrund proprietärer Lösungen. Die Hauptidee unter ESB ist die Umwandlung einer Speichen- (hierzu siehe 4.2) zu einer Busarchitektur. Es wird also eine Sammelleitung zur Kommunikation zwischen heterogenen Umgebungen aufgebaut. Das Ziel ist, an diese Sammelleitung viele unterschiedliche Quell- und Zielsysteme samt Transformationskomponenten anschließen zu können. Eine Übersicht zum Thema ESB findet sich unter [8]. Als führender Softwareanbieter, der die ESB-Konzepte umsetzt, gilt die Firma Sonic Software. Außerdem steht auch das Open Source Toolkit openadaptor zur Verfügung (mehr hierzu in [9]).

3.4.2 Entscheidungsmodell

Abschließend soll zusammenfassend ein allgemeines Entscheidungsmodell geliefert werden, das benutzt werden kann um ein Kommunikationsmodell für die konkrete Anwendung zu erstellen:

Copyright © Fraunhofer IESE 2006

- **Wahl des Architekturstils**: Soll eine Client–Server oder eine Peer to Peer Architektur zum Einsatz kommen?
- **Definition der Schichten**: Welche Dienste sollen angeboten werden und welche Client Typen werden unterstützt? Auf welche Art und Weise soll die Integration von bestehenden Anwendungen erfolgen? Soll die Anwendung webfähig sein, wie wichtig ist die Skalierbarkeit?
- **Kommunikation:** Soll die verteilte Kommunikation zwischen den Anwendungsteilen synchron oder asynchron ablaufen.? Welche Quality of Service Anforderungen werden gestellt? Erfolgt die Kommunikation über eine Luftschnittstelle und welche Technologien eignen sich um die entsprechenden Anforderungen mit den begrenzten Ressourcen zu realisieren?
- **Formate und Protokolle:** Soll ein Textbasiertes oder ein binäres Datenaustauschformat eingesetzt werden? Welches Protokoll ist dafür am besten geeignet, welche Technologie unterstützt dieses Protokoll?

Copyright © Fraunhofer IESE 2006

4 Replikationssysteme

Unter dem Begriff Replikation versteht man die Erzeugung und Wartung mehrerer Kopien eines Datenbestands oder eines Dienstes. Diese Kopien heißen auch Replikate und können auf verschiedene, voneinander entfernte Systeme verteilt werden. Die Gründe für eine Datenreplikation können sowohl funktionale als auch nicht-funktionale Anforderungen sein:

- Funktional: Mitarbeiter im Außendienst benötigen eine lokale Arbeitskopie der Daten auf ihrem mobilen Endgerät
- Nicht-Funktional: Die Replikation soll die Systemverfügbarkeit und –leistungsfähigkeit verbessern

Partitionierung und Replikation stehen in enger Beziehung. Die Partitionierung beschränkt sich auf die Teilung eines Datenbestands. Beispielsweise wird eine Datenbanktabelle, die aus 100 Datensätzen besteht, in zwei Tabellen mit jeweils 50 Datensätzen geteilt. Die entstehenden Teile, die womöglich ungleich sind, werden dann auf verschiedene Systeme verteilt. Bei der Replikation handelt es sich hingegen um die Verteilung von gleichen Daten. Die reine Partitionierung trägt so zum Lastausgleich bei und die reine Replikation zur Systemverfügbarkeit.

In einem Replikationssystem ist auch die Datensychronisation von grosser Bedeutung. Dabei handelt es sich um den Datenabgleich zwischen den Replikaten.

4.1 Replikationsphasen

Bevor eine Datenreplikation stattfinden kann, müssen die Replikationsdaten festgelegt werden. Typischerweise wird dabei ein Teil des gesamten Datenbestandes ausgewählt. Das heißt, die Partitionierung stellt oft den ersten Schritt einer Replikation dar. Diese Partitionierung ist keine einfache Aufgabe und erfordert die Abwägung verschiedener Faktoren. Eine der wichtigsten Fragestellungen ist hierbei die Handhabung der Datenbeziehungen. Wenn also ein Datenobjekt repliziert werden soll, muss entschieden werden, ob das Objekt samt aller Referenzen sowie aller untergeordneten Objekte zu replizieren ist. Dafür gibt es diverse Strategien, die unter anderem in [7] erläutert werden.

Die Phasen einer initialen Datenreplikation sehen also folgendermaßen aus.

Copyright © Fraunhofer IESE 2006

Phase	Erklärung
Partition	Auswahl der Daten aus dem Quellsystem, die auf das Zielsystem zu kopieren sind.
Transform	Transformation der Daten und Datentypen – Anpassung an das Zielsystem
Transport	Übertragung der Daten auf das Zielsystem
Apply	Speicherung der Daten auf dem Zielsystem

Tabelle 7 Phasen einer initialen Replikation

Auf der anderen Seite ist während der Synchronisation keine Partitionierung mehr notwendig. Stattdessen müssen die Datenmodifikationen festgestellt werden, die für den Datenabgleich von Bedeutung sind. In vielen Fällen werden aus Performanzgründen nur die modifizierten Daten und nicht der gesamte Datenbestand abgeglichen. Die Phasen einer Synchronisation sind in der folgenden Tabelle aufgelistet.

Phase	Erklärung
Capture	Datenmodifikationen herausfinden
Transform	Transformation der Daten und Datentypen – Anpassung an das Zielsystem
Transport	Übertragung der Daten auf das Zielsystem
Apply	Speicherung der Daten auf dem Zielsystem

Tabelle 8 Phasen einer Synchronisation

4.2 Replikationsarchitekturen

Replikationsarchitekturen können nach den folgenden zwei Hauptkategorien eingeordnet werden:

- Peer to Peer
- Master/Slave

Die Peer To Peer Replikation setzt praktisch den entsprechenden Architekturstil um, der in Absatz 3.1.2 beschrieben wurde. Die Kommunikation ist in diesem Fall bidirektional. Mit anderen Worten dürfen Änderungen an allen Replikaten vorgenommen werden, die gleichberechtigt und eigenständig sind.

Die Master/Slave Architektur unterscheidet hingegen zwischen Masters und Slaves. Schreibzugriffe sind nur auf Master-Systemen erlaubt. Die Slave-Replikate sind oft klein und beim Datenabgleich kann der Einfachheit halber der gesamte Datenbestand synchronisiert werden.

Copyright © Fraunhofer IESE 2006

Zu dem Master/Slave Ansatz gibt es verschiedene Varianten:

- Single-Master: Es gibt nur ein Master-System
- Hub-and-Spoke (zu deutsch Speichennetz): Das ist ein Verteilungsansatz, bei dem es ein zentrales Master-System gibt - den Hub (zu deutsch Nabe) - und zahlreiche Slave-Systeme - die Spokes (zu deutsch Speichen). Die Speichenkoppelung verläuft so nur über die Nabe. Verschiedene Naben können aufgebaut werden, wenn die Kommunikation zwischen Speichennetzen gewünscht ist.
- Multi-Master: Diese Variante kommt dem Peer-To-Peer Ansatz näher. Es existieren mehrere Master-Systeme und Datenmodifikationen müssen zwischen allen Masters abgeglichen werden.

4.3 Synchronisationsansätze

Für den Datenabgleich kommen verschiedene Vorgehensweisen zum Einsatz. Generell unterscheidet man zwischen Push- und Pull-Ansätzen. Im ersten Fall muss das Replikat, das Änderungen vorgenommen hat, dafür sorgen, dass die modifizierten Daten allen anderen Replikaten bekannt gemacht werden. Beim Pull-Ansatz muss das Replikat prüfen, ob modifizierte Daten vorliegen, und dementsprechend die Daten beschaffen.

Schnappschüsse erweisen sich als der bekannteste Pull-Ansatz. Dabei wird ein Teil, also ein Schnappschuss, von einem großen Datenbestand kopiert. Wenn der Schnappschuss auf den neuesten Stand gebracht werden soll, wird er erneut gemacht.

Die Push-Replikation kann synchron oder asynchron erfolgen. Dies entspricht den Kommunikationsarten, die im Absatz 3.2.2 diskutiert wurden. Bei dem synchronen Datenabgleich muss das Replikat warten, bis alle anderen Replikate den Datenabgleich bestätigt haben. In diesem Fall spricht man oft von einer transaktionalen Replikation gemäß dem 2-Phase-Commit Protokoll. Bei der asynchronen Variante werden die Daten in Warteschlangen geleitet, sodass das Replikat ununterbrochen seine Ausführung fortsetzen kann.

4.4 Replikationsmuster

Microsoft hat wiederkehrende Probleme im Bereich der Replikation identifiziert und entsprechende Musterlösungen beschrieben. Diese stehen auf der Website des Microsoft Developer Network (msdn.microsoft.com) zur Verfügung.

Copyright © Fraunhofer IESE 2006

4.5 Replikation in bekannten Datenbanksystemen

Betrachtet man große kommerzielle sowie freie Datenbanksysteme, so kann man feststellen, dass unterschiedliche Vorgehensweisen zum Einsatz kommen. Ein guter Vergleich zwischen Sybase, IBM DB2 und Oracle findet sich in [5].

In diesem Beitrag stellt sich heraus, dass für die Feststellung der Datenmodifikationen oft die Protokolldatei des Datenbanksystems verwendet wird. Dort protokolliert die Datenbank alle Datenzugriffe inklusive der Schreibzugriffe.

Alternativ können Trigger zum Einsatz kommen. Es handelt sich um benutzerdefinierte Prozeduren, die automatisch bei Erfüllung einer bestimmten Bedingung gestartet werden [6]. Eine geeignete Bedingung wäre in diesem Fall die Änderung eines Datensatzes.

4.6 Anwendungsbereiche der Replikation

Die Replikation findet hauptsächlich im Bereich der Datenbanken und der verteilten Systemen Anwendung. Darüber hinaus werden Daten auch in Versionierungs- oder einfachen Dateisystemen repliziert. Windows unterstützt beispielsweise Offline-Dateien, die lokal gespeichert und mit den Online-Dateien abgeglichen werden, wenn eine Netzwerkverbindung steht.

Weitere Anwendungsbereiche der Replikation sind:

- Verzeichnisdienste (z.B. LDAP oder DNS)
- Web oder Application Server (z.B. Replikation von Session-Daten in einem Cluster, Replikation von EJBs)
- PDAs (Replikation von Terminkalender, Addressbuch usw.)

Auch die Implementierung von Replikationsanwendungen wird zurzeit von gängigen Programmierschnittstellen unterstützt. So kann man bei .Net so genannte Datasets verwenden, die Datensätzen aus einer Datenbank lokal kapseln. Ebenso gibt es bei Java Resultsets und Rowsets. Bei der anstehenden Java-Laufzeitumgebung J2SE 1.5.0 steht sogar eine Reihe spezieller Klassen zur Verfügung, die zur Konfliktauflösung während des Datenabgleichs zum Einsatz kommen können, wie das folgende Sequenzdiagramm zeigt.

Copyright © Fraunhofer IESE 2006

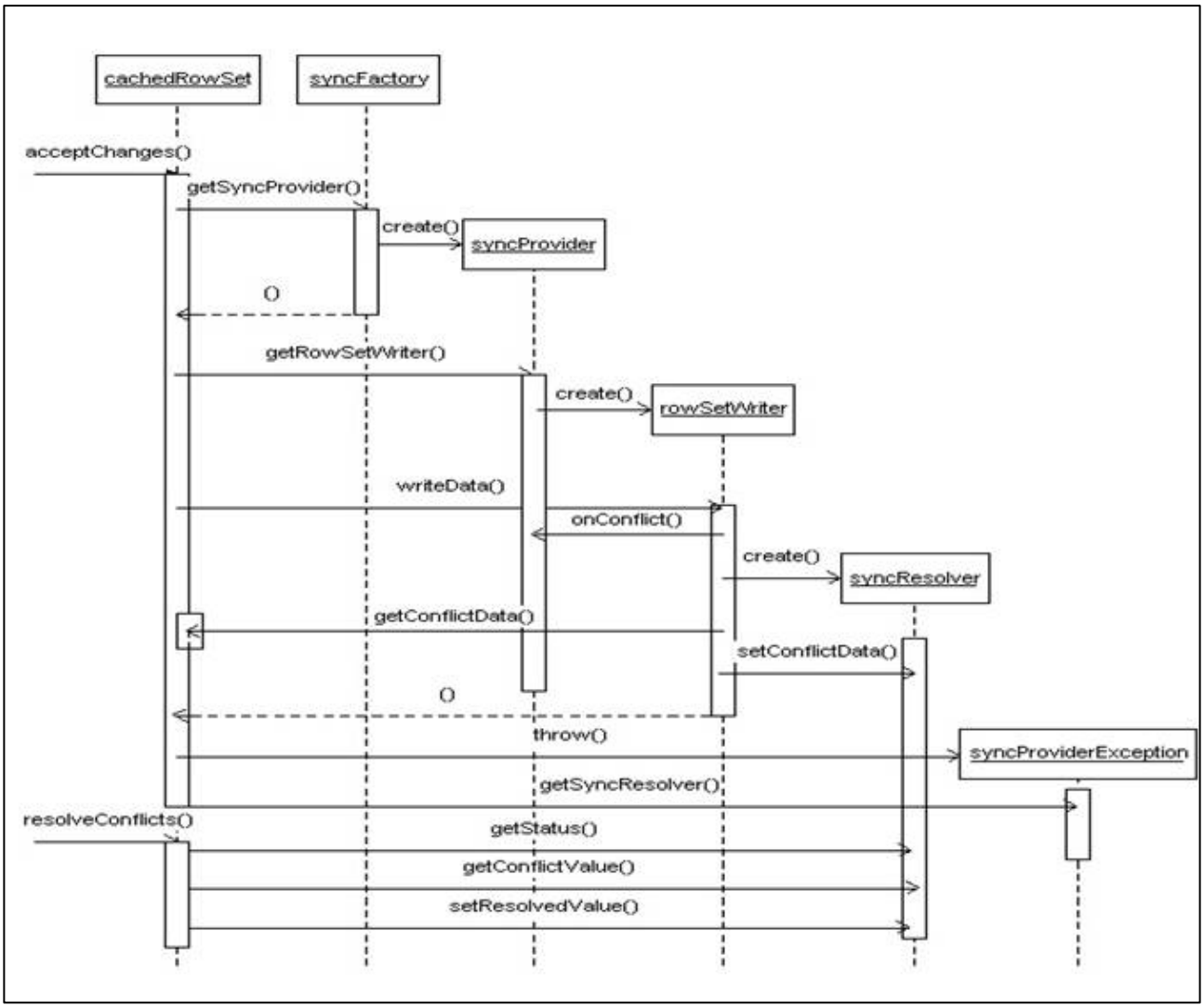


Abbildung 6 Datenabgleich mit J2SE 1.5.0

4.7 Konflikte

4.7.1 Konflikttypen

Beim Datenabgleich stellt die Konfliktauflösung einen entscheidenden Punkt dar. Ein Konflikt kommt typischerweise vor, wenn der gleiche Datensatz an verschiedenen Replikaten verändert wird. Man spricht von einem Konflikt, weil die unterschiedlichen Datenmodifikationen aufeinander stoßen. In einer Konfliktsituation muss entschieden werden, welches Replikat als Gewinner hervorgeht bzw. wie alle Änderungen zu kombinieren sind.

Copyright © Fraunhofer IESE 2006

Die verschiedenen Arten von Konflikten werden in der folgenden Tabelle aufgelistet.

Konflikt	Erklärung
Einmaligkeit	Gleiche Daten werden verteilt angelegt
Update	Gleiche Daten werden verteilt verändert
Löschen	Gleiche Daten werden verteilt gelöscht
Primärschlüssel	Gleiche Primärschlüssel werden vergeben
Integrität	Modifikationen an einer Stelle verletzen Integrität an einer anderen Stelle
Validierung	Modifikationen an einer Stelle verletzen Regeln an einer anderen Stelle

4.7.2 Konfliktauflösung

Konflikte können verschiedenartig aufgelöst werden. Die erste Möglichkeit bezeichnet man als proaktiv. Mit anderen Worten dürfen Konflikte gar nicht passieren. Verteilte Transaktionen fallen in diese Kategorie. Konflikte werden dabei grundsätzlich vermieden, indem jedes beteiligte System eine Datenmodifikation bestätigen muss.

Single-Master-Architekturen vermeiden ebenso Konflikte, da Modifikationen nur auf dem Master erlaubt sind. So koordiniert der Master gleichzeitige Schreibzugriffe und schließt Konflikte aus.

Primärschlüsselkonflikte können durch eindeutige ID-Verteilung vermieden werden. Hierzu kommen GUID (Globally Unique Identifiers) oder UUID (Universally Unique Identifiers) typischerweise zum Einsatz.

Oft werden Konflikte gemeldet, die aber eigentlich keine Konflikte sind. Wenn zum Beispiel vom Datensatz Angestellte an einer Stelle die Anschrift und an einer anderen Stelle die Telefonnummer geändert werden, ist dies kein Konflikt. Beim Datenabgleich können beide Änderungen akzeptiert werden. Jedoch kommt es bei Datenbanksystemen oft vor, dass die Suche nach Konflikten grobkörnig, auf kompletten Datensätzen läuft. In solchen Fällen erscheint das obige Beispiel als Konflikt. Feinkörnige Kontrolle, die auf der Spaltenebene arbeitet, würde dieses Problem lösen.

Alternativ zu der proaktiven Konfliktvermeidung setzen Datenbanksysteme Regeln und Prioritäten ein, um automatisch Konflikte zu lösen. So kann eine Regel beispielsweise besagen, dass die letzte Änderung alle anderen Änderungen überschreibt, oder es kann der Durchschnitt zweier Werte, die im Konflikt stehen, berechnet werden.

Copyright © Fraunhofer IESE 2006

Datenbanksysteme bieten darüber hinaus Unterstützung für die semiautomatische Konfliktlösung an. Ein Assistent führt dabei den Benutzer und schlägt mögliche Lösungen vor. Im Falle der Microsoft Access und SQL-Server Datenbanken sieht dieser Assistent wie folgt aus.

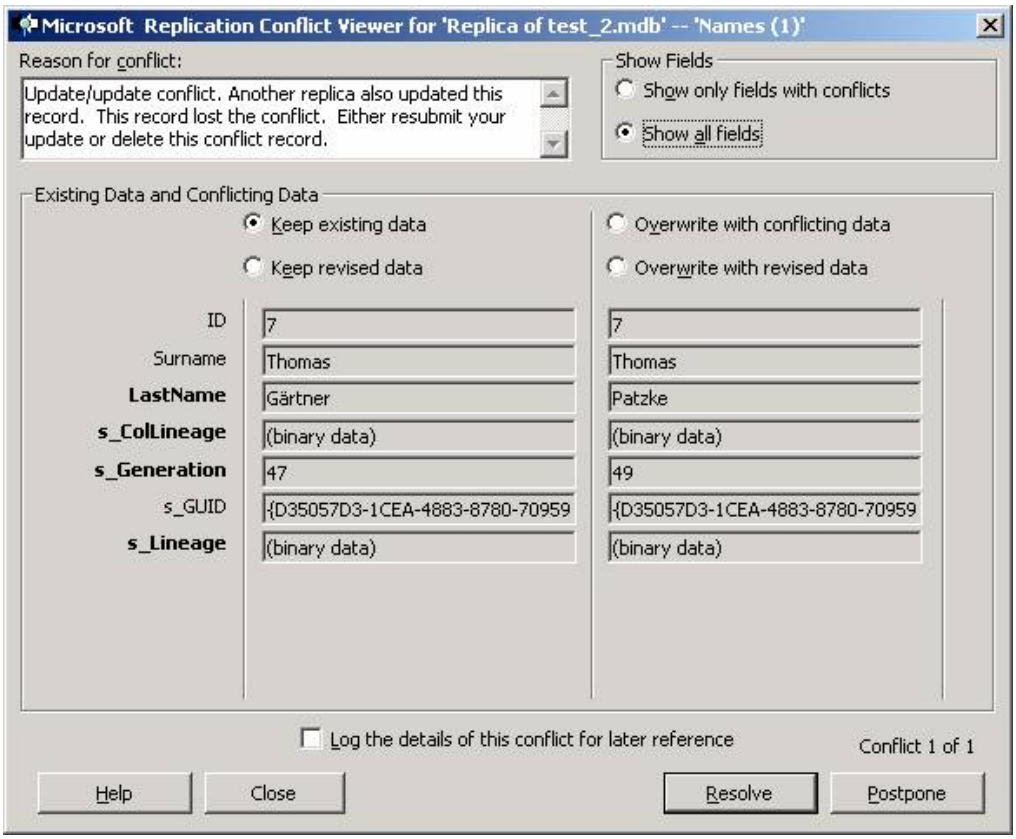


Abbildung 7 Microsoft Conflict Resolver

4.8 Datenmobilität

In modernen Anwendungsszenarien ist die Datenmobilität von großer Bedeutung. Dabei werden Unternehmensdaten zur Unterstützung des Außendienstes auf mobilen Endgeräten gespeichert. In dieser Hinsicht haben Anbieter bekannter Datenbanksysteme leichtgewichtige Varianten für Endgeräte mit eingeschränkten Ressourcen entwickelt:

Anbieter	Produkt	Plattform
Microsoft	SQL CE	Windows CE
IBM	DB2 Everyplace	Embedded Linux, PalmOS, Symbian OS, Windows CE
Oracle	Oracle Lite	EPOC, PalmOS und Windows CE

Tabelle 9 Mobile Datenbanken

Copyright © Fraunhofer IESE 2006

Im Java-Bereich existieren zudem so genannte eingebettete Datenbanken. Die können als Jar-Dateien in die aktuelle Anwendung eingebunden werden und bieten die Standardfunktionen von relationalen Datenbanken an. Außerdem bieten sich für die mobile Datenspeicherung das Record Management Store von Java Micro Edition (siehe auch Absatz 3.3.1) sowie die Java Card Schnittstelle.

4.9 Der SyncML-Standard

Der SyncML-Standard (www.syncml.org) ist eine Initiative der Open Mobile Alliance (www.openmobilealliance.org). Das Ziel ist die Standardisierung des mobilen Datenabgleichs. Dabei wurde großer Wert auf die mögliche Heterogenität der beteiligten Systeme sowie auf die Besonderheiten des Mobile Computings gelegt. Der Standard umfasst gegenwärtig die folgenden Spezifikationen:

- Architekturbeschreibung
- Repräsentationsprotokoll
- Synchronisationsprotokoll
- Bindung an bekannte Transportprotokolle
- Programmierschnittstellen
- Referenzimplementierung

Einer der wichtigsten Beiträge ist die Spezifikation der Operationen, Prozessschritte und Kommunikationsmodelle die einem Datenabgleich zugrunde liegen. Ein XML-Dialekt wurde hierzu spezifiziert, der die Repräsentation der Synchronisationsnachrichten und –daten ermöglicht.

4.10 Entscheidungsmodell

Abschließend wird auch hier ein Entscheidungsmodell aufgestellt, das die wichtigsten Fragestellungen einer Replikationsarchitektur zusammenfasst.

- Die Datensynchronisation kann entweder mittels spezieller Synchronisationssysteme (siehe beispielsweise www.intellisync.com) oder mittels Lösungen, die in Datenbanken eingebaut sind erfolgen. Im ersten Fall liegt der Mehrwert an der möglichen Heterogenität der beteiligten Systeme. Im anderen Fall sind Replikationslösungen bereits im Lieferumfang einer Datenbank enthalten, was Stabilität aber auch Einsparungen bedeutet.

- Architektur: Ist allgemeiner Schreibzugriff notwendig, so muss eine Peer-To-Peer oder Multi-Master Architektur aufgebaut werden. Sonst ist eine Single-Master Lösung ausreichend.

Copyright © Fraunhofer IESE 2006

- Push- oder Pull-Ansatz: Die Entscheidung hängt in erster Linie von der Änderungshäufigkeit der Daten ab. Daten die sich häufig ändern benötigen meistens einen Push-Ansatz.

- Capture: In der Capture-Phase werden die Datenmodifikationen ermittelt. Das läuft meistens über die Protokolldatei oder über Trigger ab. Trigger eignen sich besser für asynchrone Replikation und dynamische Daten.

- Transport: In der Transport-Phase werden die Replikationsdaten übertragen. Dabei stellt sich die Frage, ob der gesamte Datenbestand oder nur die Änderungen zu übertragen sind. Letzteres erhöht womöglich den Entwicklungsaufwand, ist aber für schmalbandbreitige Kommunikation geeignet.

- Konflikte: Die Strategie der Konfliktauflösung ist ebenso ein wichtiger Punkt. Viele Faktoren müssen hier abgewogen werden. Proaktive Auflösung mit Hilfe von Transaktionen kann zum Beispiel der Leistungsfähigkeit schaden. Automatische Auflösung erfordert die Definition von allgemeingültigen Regeln.

4.11 Schemareplikation

Das typische Replikationsszenario umfasst lediglich die Replikation von Daten unter der Voraussetzung, dass das Datenmodell oder –schema sich nicht ändert. Die Transformationphase ist zwar für die Anpassung an das Zielsystem verantwortlich, meistens handelt es aber lediglich um die Umwandlung der elementaren Datentypen. Microsoft unterscheidet hierzu zwischen Replikation und Extract Transform Load. Der letzte Ansatz kommt in Frage, wenn die Datenmodelle wesentlich voneinander abweichen. Extract Transform Load ist der typische Ansatz für den Aufbau von OLAP-Datenbeständen. Dabei werden Daten gesammelt und in mehrdimensionale Strukturen überführt, die eine weitgehende Datenanalyse ermöglichen.

Schemareplikation kann prinzipiell auf zwei Wegen erzielt werden:

1.) Daten sind selbstbeschreibend: Das replizierte Datenobjekt enthält Schemainformationen, die somit auch repliziert werden.

2.) Schemainformationen sind auch Datenobjekte: Das Datenbanksystem verwaltet das Schema als eine Gruppe von Datenobjekten, die somit gespeichert und repliziert werden können.

Copyright © Fraunhofer IESE 2006

5 Produktübergreifende Workflow-Modellierung

5.1 Workflow-Modellierung in UNIVERSYS

Die im Rahmen von UNIVERSYS angestrebte Kopplung der drei Informationssysteme soll es ermöglichen, den Arbeitsablauf des Anwenders produkt- bzw. modulübergreifend darzustellen. Das heißt, es wird erstmalig angestrebt, Sachdaten direkt mit ihren Zustandsgrößen und geografischen Informationen zu unterlegen, so dass der Nutzer, je nach Aufgabenstellung, sofort einen umfassenden Überblick über seine Betriebsumgebung erhält.

Auch wird ihm so die Möglichkeit gegeben, den Workflow in verschiedenen Bezugskategorien darzustellen und somit Ansatzpunkte für eine effizientere Betriebsführung zu erkennen.

5.2 Einführung in die Workflow-Modellierung

Unter dem Begriff Workflow hat man früher den Fluss von Dokumenten innerhalb einer Organisation verstanden. Dieser Dokumentenfluss hat verschiedenen Zwecken gedient wie zum Beispiel Verfassung, Genehmigung oder Korrektur eines Dokumentes. Heutzutage versteht man darunter die Gesamtheit der Beziehungen zwischen Arbeitsschritten in einem Geschäftsprozess. Darüber hinaus hat das Workflow-Management an Bedeutung gewonnen, dessen Ziel „die Automatisierung von Geschäftsprozessen oder Vorgängen in denen Dokumente, Informationen oder Arbeitsaufträge unter Berücksichtigung von Regeln oder definierten Verfahren von einem Teilnehmer zum nächsten gereicht werden“ [1].

Die technische Umsetzung des Workflow-Managements wird durch spezielle Werkzeugumgebungen – sogenannte Workflow-Managementsysteme – unterstützt. Der Fokus liegt dabei auf der Modellierung, Ausführung und Überwachung von Geschäftsprozessen.

Die Automatisierung von Geschäftsprozessen bedeutet sehr oft die Integration und Kommunikation verschiedener Applikationen, möglicherweise über Betriebsgrenzen hinweg. Dies kommt zustande, wenn ein Arbeitsablauf aus Schritten besteht, die jeweils von unterschiedlichen Applikationen und Rollen unterstützt werden. Ein derartiger Ablauf wird oft manuell abgewickelt, wie das folgende Beispiel zeigt:

1.) Der Projektleiter verfasst eine Aufgabenstellung mit einem entsprechenden Editor.

Copyright © Fraunhofer IESE 2006

2.) Er überprüft die Verfügbarkeit der Mitarbeiter mit einem Projektmanagement-Werkzeug.

3.) Er teilt die Aufgabe einem Mitarbeiter zu und verschickt ihm die Aufgabenstellung.

4.) Der Mitarbeiter liest die Aufgabenstellung und zieht die für ihn relevanten Informationen heraus.

5.) Er überprüft die Aufgabendatenbank, ob eine ähnliche Aufgabe bereits bewältigt wurde. Wenn nicht, legt er einen neuen Eintrag an, sonst ergänzt er einen bereits existierenden Eintrag.

6.) Er führt die Aufgabe aus.

Mit Hilfe eines Workflow-Managementsystems können derartige Arbeitsschritte weitestgehend automatisiert werden. In diesem Fall übernimmt das System die Übertragung relevanter Informationen sowie den Aufruf der beteiligten Applikationen

5.3 Technische Umsetzung

Um die Automatisierung der Ablausteuerung zu ermöglichen, müssen Geschäftsprozesse ausführlich beschrieben werden. Mit anderen Worten müssen grob bzw. umgangsprachlich beschriebene Arbeitsabläufe in Workflows mit hohem Detailgrad überführt werden. Das setzt beispielsweise voraus:

- Festlegung der beteiligten Applikationen und deren Aufrufparameter
- Typisierung der zu übertragenden Daten
- Formalisierung umgangssprachlicher Bedingungen

Viele Anbieter von Midlleware-Produkten, versuchen Workflow-Managementsysteme in den Lieferumfang ihrer Produkte zu integrieren. Als Beispiele können das Produkt WebLogic Integration von BEA, der Biztalk Server von Microsoft sowie die Open-Source-Produkte JaWE und Shark von Enhydra genannt werden.

Neben den Produktanbietern existieren gegenwärtig Organisationen, deren Ziel ist, die Verwaltung von Workflows mit Standards zu unterstützen. Die zwei bekanntesten Organisationen sind zur Zeit die Workflow Management Coalition (www.wfmc.org) und die Business Process Management Initiative (www.bpmi.org). Produktanbieter sind daher bemüht, Werkzeuge auf den Markt zu bringen, die mit diesen Standards konform sind.

Copyright © Fraunhofer IESE 2006

5.3.1 Umsetzung mit JaWE und Shark von Enhydra

Die Enhydra-Lösung basiert auf WfMC-Standards und besteht aus einem Workflow-Modellierungswerkzeug (JaWE) sowie aus einer Engine (Shark). Die Engine ist für die Ausführung und Überwachung der Workflows verantwortlich.

Nachfolgend sind die Schritte aufgelistet, die für die Modellierung und Ausführung eines Workflows mit JaWE und Shark notwendig sind:

1.) Definition des Workflow-Pakets: Das Workflow-Packet entspricht dem jeweiligen Entwicklungsvorhaben. Es enthält die zu modellierenden Arbeitsabläufe sowie die beteiligten Applikationen und Rollen. Die Anlage eines Workflow-Pakets bedeutet also das Setzen folgender Eigenschaften:

   - Datentypen: Globale Datentypen, die innerhalb von Prozessen genutzt werden
   - Beteiligte: Rollen, Menschen, Organisationseinheiten, Ressourcen oder Systeme, die an einem Prozess teilnehmen.
   - Applikationen: Werkzeuge, die Prozessschritte unterstützen. Hier müssen die Aufrufparameter formal definiert werden.

2.) Workflow-Modellierung: Jeder Workflow ist hier graphisch zu modellieren. Der Benutzer kann mit JaWE die diversen Bestandteile eines Workflows modellieren und ihre Eigenschaften bearbeiten. Die Bestandteile eines Workflows sind:

   - Aktivität: Das ist der Grundbaustein eines Prozesses und entspricht einem einzelnen Prozessschritt. Einer Aktivität können Werkzeuge hinterlegt werden. Somit kann beschrieben werden, wie eine Aktivität tatsächlich implementiert wird.
   - Aktivitätsübergang: Beschreibt den Übergang von einem Prozessschritt zu einem anderen. Ein Übergang beinhaltet oft Bedingungen, die erfüllt werden müssen, damit der Übergang tatsächlich stattfindet.
   - Unterprozess: Das ist ein untergeordneter Prozess, der außerhalb des aktuellen Workflow, möglicherweise in einer eigenen Laufzeitumgebung, ausgeführt wird.
   - Beteiligter: Beteiligten werden ganze Workflow-Bereiche zugeordnet. So ist ein Beteiligter für alle Aktivitäten, Übergänge und Unterprozesse in einem Bereich verantwortlich.

3.) Abbildung von Applikationen: Für die Ausführung eines Workflows mit Shark ist jetzt die Abbildung der mit JaWE vordefinierten Applikationen auf lauffähige Programme notwendig. Zum Beispiel wird jetzt die Applikation „Auftrag von Datenbank löschen“, die der Aktivität „Auftrag löschen“ hin-

Copyright © Fraunhofer IESE 2006

terlegt ist, auf eine Java-Klasse abgebildet. Eine derartige Java-Klasse muss die Methode execute() implementieren, die beim Auftritt der Aktivität aufgerufen wird.

4.) Abbildung von Beteiligten: Beteiligte Rollen, Personen usw., die mit JaWE modelliert wurden, können optional auf konkrete Shark-Benutzer abgebildet werden. So können unberechtigte Zugriffe auf Aktivitäten vermieden werden.

5.) Ausführung des Workflows: Die Workflow-Ausführung wird von Shark gesteuert. Das heißt, dass jede Aktivität einem Shark-Benutzer zugeteilt wird. Der Benutzer erhält so eine Arbeitsliste mit Aktivitäten, die er ausführen muss.

5.4 Workflow-Management und Applikationsintegration

Workflow-Management und Integration bereits bestehender Applikationen stehen heutzutage in enger Beziehung. Der Grund ist, dass die Logik für die Kopplung und Koordination existierender Systeme in ein Workflow-Managementsystem verlagert werden kann. Deswegen lohnt sich der Einsatz von Workflow-Systemen nur, wenn diese Logik kompliziert ist und vom Rest des Systems zu trennen ist.

Ein weiterer Grund für diese enge Beziehung ist, dass moderne Workflow-Managementsysteme die bekanntesten Integrationstechnologien standardmäßig anbieten. Dazu gehören Datentransformationen zwischen heterogenen Systemen sowie Nachrichtenaustausch (Messaging).

5.5 Workflow-Standards

Im Bereich des Workflow- bzw. Geschäftsprozess-Managements existieren gegenwärtig zahlreiche Standardisierungsversuche, die sich oft überlappen und damit leicht zu Unklarheiten führen. So gibt es Spezifikationen unter anderem für:

1.) Die konkreten Inhalte, die zwischen Geschäftspartner ausgetauscht werden. RosettaNet (www.rosettanet.org) spezifiziert zum Beispiel, welche Informationen die Nachricht über eine Warenanlieferung enthalten soll. Andere Standards in diesem Bereich sind: ebXML, CPFR, SCOR.

2.) Die Interaktion zwischen Prozessen, die auf getrennten Systemen laufen. So beschreibt beispielsweise der WF-XML Standard, wie die Liste aller laufenden Prozesse auf einer Workflow-Engine abgefragt werden kann. Die WSCI- und BPQL-Spezifikationen beschäftigen sich ebenso mit diesem Bereich.

Copyright © Fraunhofer IESE 2006

3.) Die Notation von Geschäftsprozessen. Die Business Process Modeling Notation definiert graphische Elemente für Prozessbestandteile wie Ereignisse, Aktivitäten, Bedingungen, Nachrichten usw. Als alternative Notation bieten sich auch Aktivitäts- oder Zustandsdiagramme gemäß der UML-Spezifikation.

An dieser Stelle ist zu erwähnen, dass Workflows und Geschäftsprozesse oft miteinander verwechselt werden. Aus dem Grund gibt es Bemühungen von den jeweiligen Standardisierungsgremien, also WfMC und BPMI, einen gemeinsamen Nenner zu finden. Ein allgemeingültiger Unterschied ist, dass Workflows sich mit Prozessen einer einzigen Organisation befassen, während Geschäftsprozesse sich über Organisationsgrenzen hinweg erstrecken und aus dem Grund zu den entsprechenden Technologien in Beziehung stehen, wie beispielsweise Web-Services oder Messaging.

Sowohl WfMC als auch BPMI haben Referenzmodelle für die Prozessmodellierung entwickelt. Dort werden die jeweiligen Konzepte (Prozess, Aktivität, Rolle usw.) und ihre Beziehungen spezifiziert. Basierend auf diesen Modellen gibt es bereits zwei XML-Dialekte:

1.) Die XML Process Definition Language (XPDL) der WfMC

2.) Die Business Process Modeling Language (BPML) der BPMI.

5.6 Umsetzung für UNIVERSYS

Die Umsetzung im Rahmen von UNIVERSYS erfordert zuerst Klarheit über die Ziele der produktübergreifenden Ablaufsteuerung. Die prinzipielle Frage ist, ob die übergreifende Workflow-Modellierung als Mittel zur Applikationsintegration dient oder andersherum. Wenn also die Workflow-Modellierung zur Integration führen soll, werden zuerst die produktübergreifenden Geschäftsprozesse modelliert und dann mit Workflow-Mitteln umgesetzt. Dies setzt möglicherweise die Beschreibung der Prozesse für die einzelnen Systeme voraus. Die verschiedenen Bezugskategorien des Workflows können hier als Workflow-Variabilität angesehen werden.

Der Aufwand dieser Lösung ist möglicherweise hoch, wenn die existierenden Systeme an die Anforderungen einer Workflow-Architektur angepasst werden müssen. Andererseits ist eine derartige Lösung möglicherweise zukunftssicher, da sie den Geschäftszielen entspricht, moderne Integrationstechnologien einsetzt und die Systeme auf allen Integrationsebenen (Daten, Anwendungslogik, Benutzungsschnittstellen) zusammenführt.

Der Integrationsaufwand kann gering bleiben, wenn die Kopplung zunächst ohne Workflow-Konzepte umgesetzt wird. Die Projektpartner haben bereits Er-

Copyright © Fraunhofer IESE 2006

fahrungen mit der Kopplung der Einzelnsysteme gemacht und können sie an dieser Stelle wieder verwenden. Nach der Systemintegration kann der resultierende Workflow beschrieben werden. Dieser kann an dieser Stelle zur Bewertung der Kopplung benutzt werden.

Copyright © Fraunhofer IESE 2006

6 Webbasierte Umsetzung

Bei einer webfähigen Anwendung realisieren die Clients in der Regel wenig Fachlogik. Im Wesentlichen übernehmen sie lediglich die dynamische Darstellung von Daten und ermöglichen die Manipulation dieser Daten durch den Benutzer. Komplexe Routinen werden auf dem Server ausgeführt. Auf Grund dieser Beschränkungen werden diese Clients auch als Thin Clients bezeichnet. Thin Clients werden in den meisten Fällen browserbasiert realisiert, sie können auf unterschiedlichen Endgeräten laufen. Abhängig von Displaygröße und weiteren Einflussgrößen existieren unterschiedliche Protokolle und Formate, die den unterschiedlichen Eigenschaften der Endgeräte Rechnung tragen. Diese Aspekte wurden bereits in Kapitel 3 behandelt. Neben den Thin Clients, die im Browser laufen existieren Lösungen, die ein umfangreicheres Userinterface und andere Services bieten, die lokal ausgeführt werden. Solche Clients bezeichnet man auch als Rich Clients. Fat Clients schließen das Spektrum ab, sie greifen nur noch auf einen Server zu, wenn sie neue Daten benötigen, der größte Teil der Berechnungen läuft auf dem Endgerät.

6.1 Architektur einer Webanwendung

Präsentationsschicht

Typische mehrschichtige Anwendung wurden bereits in Kapitel 3.1 vorgestellt. Wird die Webfähigkeit einer Anwendung mit Hilfe einer browserbasierten Lösung umgesetzt wird die Darstellungs- oder Präsentationsschicht nochmals in eine Server und Clientschicht unterteilt. Die Clientschicht umfasst die Endgeräte mit den unterschiedlichen Browserinstallationen. Als Format werden im Desktopbereich textbasierte Auszeichnungssprachen wie HTML und im Bereich der mobilen Endgeräte WML eingesetzt. Die Aufgabe des Browsers ist es, die übermittelten Markup Texte zu parsen und die darin enthaltene Information spezifikationsgemäß zu formatieren und darzustellen. Auf der Serverseite geht es darum die Daten entsprechend dem anfragenden Client in das richtige Format (also beispielsweise WML oder HTML) zu bringen und zu senden (sog. Multichanneling).

Durch die Benutzerinteraktion wird in einem Informationssystem die Datenbasis verändert. Diese Änderungen werden anschließend wieder dargestellt, daher sind statische Dokumente ungeeignet. Vielmehr sind auf der Serverseite Mechanismen erforderlich, die Markup Seiten dynamisch erzeugen können und zwar in Abhängigkeit von der Art der Anfrage und der zugrunde liegenden, sich verändernden Datenbasis. [10]

Copyright © Fraunhofer IESE 2006

Alternativ können auch Clients mit einer browserunabhängigen Benutzeroberfläche zum Einsatz kommen. Diese Anwendungen wurden bereits als Rich Clients oder Fat Clients eingeführt. Hier kann die Webserverfunktionalität entfallen, das heißt, der Client kann direkt auf den Application Server zugreifen. Allerdings sind auch Lösungen denkbar, bei der ein solcher Rich-Client mit dem Webserver kommuniziert um die Datenübertragung über ein reines http-Protokoll zu tunneln bzw. über das SOAP Protokoll mit einem entsprechenden Server der dann mit dem Backend kommuniziert. [48]

Entsprechende Technologien werden im weiteren Verlauf dieses Abschnittes noch vorgestellt und diskutiert.

6.2 Browserbasierte Lösungen

Browser Client

Im Desktopbereich existieren eine Reihe von Browserlösungen. Neben dem Internet Explorer von Microsoft, der hauptsächlich auf Windows Betriebssystemen und dem Mac OS zum Einsatz kommt existieren beispielsweise Lösungen von Mozilla, Netscape und Opera, die auch für Linux, Unix und das Mac OS angeboten werden. Im Bereich der Mobile Clients existieren ebenfalls eine Reihe von so genannten Micro-Browsern, die typischerweise mit der Software für die Embedded Devices mitgeliefert werden. Hier existieren beispielsweise Produkte von Openwave, Nokia, Symbian, Microsoft und ACCESS.

Web Server

Ein Webserver hat in einer Architektur welche Webbrowser einsetzt nicht nur die Aufgabe, statische Webseiten zu Hosten. Da sich die Datenbasis ständig ändert muss auch der Inhalt der Seiten angepasst werden, so dass sie immer wieder pro Anfrage neu erzeugt werden müssen. Viele Webserver stellen deshalb auch eine Laufzeitumgebung für Komponenten zur Verfügung, die diesen HTML-Output dynamisch erzeugen. Im Java Bereich wird eine solche Laufzeitumgebung als Servlet Engine bezeichnet. Es existieren unterschiedliche Technologien solche Output-Komponenten zu implementieren. Die entsprechende Java-Technologie arbeitet mit Java Server Pages (JSP) und Servlets, das Microsoft Gegenstück sind die so genannten Active Server Pages (ASP). Andere bekannte Lösungen sind PHP Skripte, die vom Webserver ausgeführt werden.

Im Java Bereich existieren auf Servlet- und JSP-Technologien aufbauende skalierbare Lösungen. Ein Beispiel hierfür ist das Model View Controller Framework Struts. Da JSP Seiten eine Mischung aus HTML-Templates und reinem Java (oder auch anderen Programmiersprachen) sind, entsteht unübersichtlicher und schwer wartbarer Code. Das liegt unter vor allem daran, dass Aspekte, die nur die Darstellung betreffen mit Controller Funktionalitäten und Strukturen und

Copyright © Fraunhofer IESE 2006

Funktionalitäten, die man in einer MVC Architektur als Modell bezeichnet in einer Komponente –der JSP-Seite- vermischt werden.

Das Open Source Framework Struts von Apache bietet mit seinem Action Konzept eine Lösung, bei der diese Teile getrennt und in eine Model View Controller Architektur überführt werden. Die JSP Seite dient dabei nur zur Visualisierung, Benutzerkommandos, wie zum Beispiel Button Clicks, werden auf vom JSP Code getrennte Controllerklassen (Actions) gemapped. Das Framework bringt die Methoden in der einem Benutzerkommando zugeordneten Controllerklasse zur Ausführung. In den Controllerklassen erfolgt beispielsweise auch der Zugriff auf das Datenmodell. Ein Ausschnitt des gesamten Datenmodells einer Anwendung liegt innerhalb des Struts-Frameworks vor und zwar in Form von so genannten Action Forms. Diese Action Forms kapseln Daten, die für die Visualisierung und Bearbeitung relevant sind und welche in einer zugeordneten JSP angezeigt werden sollen.

Eine weitere Java basierte Lösung ist das Webpublishing Framework Cocoon. Cocoon wurde bereits in Abschnitt 3.1.4 kurz vorgestellt und adressiert einen weteren Aspekt der serverseitigen Programmierung. Cocoon propagiert eine Trennung von Inhalt und Darstellung durch den Einsatz von XML. Dabei werden Daten in Form von XML Dokumenten verwaltet und auf Anfrage mit Hilfe von XSL-Stylesheets in ein Markup Format gebracht das von den unterschiedlichen Webbrowsern angezeigt werden kann. Dadurch werden auch Daten von der Visualisierung getrennt. Cocoon selbst ist prinzipiell ein Servlet, das als Controller dient und eingehende Requests unterbricht den Kontext der Anfrage analysisert und an die entsprechende Transformationslogik weiterleitet. Es bietet ähnlich wie die JSP Technologie die Möglichkeit dynamisch Output für unterschiedliche Browser zu erzeugen. Dabei muss allerdings zunächst der Inhalt generiert werden, das heißt es muss ein XML-Dokument erzeugt werden, das die gewünschten Daten enthält, anschließend wird der Transformationsprozess angestoßen. Durch diesen Ansatz ist es möglich Zuständigkeiten aufzuteilen. Dabei sind unterschiedliche Rollen zu besetzen, nämlich zum einen Webdesigner, welche die Stylesheets umsetzen und zum anderen Programmierer, die den Content bereitstellen.[15]

6.3 Rich-Client basierte Lösungen

Im Desktopbereich besteht alternativ zum Browser die Möglichkeit, einen Rich Client mit einer umfangreicheren grafischen Benutzeroberfläche einzusetzen der lokal installiert ist. Der Vorteil einer Rich Client Lösung gegenüber einem Webbrowser besteht darin, dass keine Formatierungs- und Layoutinformationen die verfügbare Bandbreite verschmälern. Durch den Wegfall der Übertragung von Formatierungsdaten und deren anschließende Darstellung durch den Browser ist das Userinterface reaktiver. Außerdem lassen sich solche Clients komfortabler und Benutzerfreundlicher ausstatten, da sie meistens mit Hilfe der

Copyright © Fraunhofer IESE 2006

umfangreichen GUI-Bibliotheken einer Programmiersprache erstellt werden und sich besser in den Desktop integrieren. Sie unterliegen sie nicht den Restriktionen einer Markup Sprache wie HTML mit ihren eingeschränkten Möglichkeiten zur Darstellung von Steuerelementen und Layouts. Der große Nachteil einer Rich Client Lösung besteht in der dezentralen Administration solcher Clients. Sie müssen bei einer Neuinstallation und Implementierungsänderungen auf jedes Endgerät aufgespielt werden. Der Appletmechanismus von Java kann ebenfalls zur Umsetzung von Rich Clients verwendet werden und bietet entscheidende Vorteile gegenüber der vorher genannten Lösung. Applets werden durch den Webbrowser von einem zentral administrierten Server geladen. Vorraussetzung dafür dass der Applet Client auf dem Endgerät ausgeführt werden kann ist, dass der Browser mit einer Java virtuellen Maschine (JVM) ausgestattet ist. Das Applet läuft dann innerhalb des Browsers. Der Vorteil dieser Lösung ist, dass bei der Implementierung des Applets mit einer Programmiersprache und den entsprechenden Bibliotheken gearbeitet werden kann. Der Nachteil liegt darin, dass grundsätzlich trotzdem mit einem Browser gearbeitet werden muss. Technologisch verwand aber von der Art der Benutzererfahrung unterschiedlich ist die Java Webstart Technologie. Java Webstart ist eine auf der JNLP-Spezifikation (Java Network Launching Protocol) von Sun basierende Technologie zur Verteilung und Versionierung von vollwertigen Java Anwendungen über ein Netzwerk. Zur Bereitstellung der Anwendung ist lediglich ein HTTP-Server notwendig, zur Ausführung wird auf der Clientmaschine eine entsprechende Java Laufzeitumgebung benötigt, die der Anwendung spezifizierte Services wie Datei- und Netzwerkzugriffe bietet. Der wesentliche Unterschied zu einem Applet besteht darin, dass nur der Download des Clients durch einen Browser angestoßen wird. Ansonsten läuft diese Anwendung völlig Browserunabhängig in einer eigenen und sicheren Umgebung. Die Anwendungsbibliotheken werden von Webstart lokal und für den Benutzer transparent gecached. Eine solche Anwendung kann dann immer wieder wie eine normale Desktopanwendung über ein Icon gestartet werden. Webstart bietet außerdem die Möglichkeit, inkrementelle Updates durchzuführen, so dass nur die Differenz zwischen zwei Anwendungsversionen und die Beschreibung der Differenz über das Netzwerk geschickt werden müssen. Die Webstart Laufzeitumgebung patched bei einem Update die entsprechenden Bibliotheken auf dem Client automatisch. Sowohl Applet als auch Webstart Client können entweder direkt z.B. über RMI, über einen Webserver mittels http oder beispielsweise auch über SOAP mit dem Backend System kommunizieren. Mit ClickOnce ist Microsoft im Rahmen seines .NET Frameworks dabei, eine vergleichbare Technologie einzuführen. In der Microsoft Terminologie werden Rich Clients die Webservices, Online-Offline-Szenarien, Anpassung an Spezifika des Endgeräts sowie automatische Installationen und Updates unterstützen auch unter dem Begriff Smart Clients zusammengefasst. [11] [12] [17]

Copyright © Fraunhofer IESE 2006

Aus Gründen der geringeren Bandbreite der Luftschnittstelle ist, je nach Anwendungsszenario, der Einsatz von Rich Clients dem Einsatz von (Micro-) Browsern auf dem mobilen Endgerät vorzuziehen.

Im Folgenden sollen kurz die beiden etablierten Alternativen für die mobile Client Programmierung vorgestellt und verglichen werden. Das .NET Compact Framework von Microsoft und die J2ME bieten die Möglichkeit, Rich Clients zu entwickeln. Die eingesetzte Programmiersprachen sind beim Compact Framework C# oder VisualBasic.NET für J2ME ist es Java. Beide Frameworks arbeiten mit managed Code und setzen eine virtuelle Maschine ein, für beide Plattformen existieren unter anderem Bibliotheken zur Erstellung von GUI Komponenten, zum lokalen Datenmanagement, zur Netzwerkprogrammierung und zur Benutzung von Webservices. Der Unterschied zwischen beiden Plattformen besteht vor allem darin, dass J2ME auf Standards beruht, die unter Einbeziehung der Industrie im so genannten Java Community Prozess entstanden sind bzw. entstehen, während das Compact Framework auf proprietären Lösungen von Microsoft beruht. Das bedeutet, dass .NET Anwendungen nur auf MS Pocket PCs lauffähig sind, während J2ME Anwendungen bereits von sehr vielen Herstellern durch die Bereitstellung eine VM auf den unterschiedlichsten Geräten und unabhängig vom Betriebssystem bereitgestellt werden. Allerdings haben Entwickler im .NET Compact Framework Bereich den Vorteil, dass sie bei Implementierungen mit einem Standardumfang von Ressourcen auf den Pocket PCs rechnen können. Die Entscheidung welche Plattform hier in Frage kommt kann nicht ohne weiteres beantwortet werden. Ein Entscheidungskriterium sind sicherlich die Endgeräte. In einer homogenen Client Umgebung, in welcher nur Microsoft Pocket PCs zum Einsatz kommen, kann auch das .NET Compact Framework problemlos verwendet werden. Sollten allerdings unterschiedliche Endgeräte mit unterschiedlichen Leistungsmerkmalen und verschiedenen Betriebssystemen unterstützt werden, so ist J2ME auf Grund seiner verbreiteten Standards vorzuziehen.

Beiden Technologien gemeinsam ist die Möglichkeit auch Webservices nutzen zu können. [13][14]

6.4 Webservices mit SOAP

In diesem Abschnitt soll eine konkrete Lösung für eine Webservices Infrastruktur beschrieben werden. SOAP wird als der kommende Standard für die Plattform und Anwendungsübergreifende Kommunikation angesehen. Im Java und C++ Bereich existiert mit AXIS von Apache als Open Source Lösung eine so genannte SOAP-Engine. Eine SOAP Architektur umfasst diverse Komponenten auf Client und Serverseite, die durch dieses AXIS Frameworks bereitstellt werden oder auch generiert werden können.

Copyright © Fraunhofer IESE 2006

AXIS lässt sich als Service in einem Servletfähigen Webserver wie Tomcat von Apache integrieren. In diesem Szenario wird als Transportprotokoll http benutzt, es ist allerdings auch möglich JMS und Mail einzusetzen.

Die einzelnen Services des Frameworks wie Beispielsweise Administration oder Transportprotokolle werden in Modulen verwaltet, die nach Bedarf konfiguriert werden können.

Die Infrastrukturen auf Client- und Serversite sind sich sehr ähnlich. Der Mechanismus, der umgesetzt wird entspricht im wesentlichen der XML-Pipeline wie sie bereits in Kapitel 3.1.4 vorgestellt wurde. AXIS setzt zusammen mit der XML-Pipeline zusätzlich Handlerketten ein. Neben dem verpacken und entpacken der XML-Nachrichten in der Pipeline existieren unterschiedliche Handler für Logging, Sicherheit und andere Services, die variabel konfiguriert werden können. Ein besonderer Handler, der so genannte Provider, ruft auf Serverseite den eigentlichen Dienst im Backendsystem auf. Dieser Handler entspricht in Abbildung 2 dem Dispatcher. Es existieren Dispatcher für den Zugriff auf Remote Objekte über COM, Corba und RMI aber auch auf lokale Services wie zum Beispiel einfache Java-Klassen. Die ausgehende Nachricht läuft dann wieder über die Handlerkette zurück. Auf Clientseite wird die Nachricht entgegengenommen und über eine weitere Handlerkette an den initial aufrufenden Prozess zugestellt. Clients können sowohl Thin Clients aber auch Rich bzw. Smart Clients sein.

Die AXIS-Engine unterstützt außerdem das Handling der Web Service Description Language (WSDL). Mit Hilfe des AXIS-Frameworks kann man sich für eigene Services WSDL Beschreibungen generieren lassen, die von anderen genutzt werden können bzw. es lassen sich Stubs oder Proxies für Remote Services, die man in seiner Anwendung nutzen möchte und für die eine WSDL Beschreibung vorliegt, erstellen.

Zusammenfassend ist zu sagen, dass es sich bei AXIS um eine mächtige Open Source Lösung handelt, die auch im Mobile Client Bereich eingesetzt werden kann und deren Handlermechanismus den Zugriff auf Backendsysteme unterschiedlichster Plattformen unterstützt. Sie ist außerdem nicht auf die Java Plattform beschränkt. [15][17]

Copyright © Fraunhofer IESE 2006

7 Abbildung des Kriterienkatalogs auf Ansätze

Anforderung	Ansatz/Technologie	Erläuterung/Beispiele
Client-Heterogenität	Trennung der Präsentation von der Logik	Durch die Trennung der Präsentations- von der Anwendungslogik können Geschäftsanwendungen verschiedenen Benutzeroberflächen zur Verfügung gestellt werden. Das Model-View-Controller Muster und seine Varianten (z.B. Struts) tragen diesem Ansatz Rechnung.
	XML/HTTP	XML ist plattformunabhängig und kann somit als gemeinsames Datenaustauschformat zwischen heterogenen Anwendungen zum Einsatz kommen. Web Services setzen auf XML und HTTP auf.
	Citrix Metaframe, Windows Terminal Server, Java Applets, Java Web Start, Microsoft ActiveX controls	Mit diesen Technologien können diverse Client-Umgebungen auf existierende Anwendungen zugreifen.
Datenbank-Heterogenität	Zentrale Zugriffsschicht	Der Aufbau einer zentralen Zugriffsschicht abstrahiert von den Besonderheiten diverser Datenbankschnittstellen. Letztere werden an diese Zugriffsschicht mittels geeigneter Konnektoren angeschlossen. Beispiele von diesem Ansatz stellen die ODBC, JDBC sowie JCA Standards dar.
Plattformübergreifende Anwendungsentwicklungs	Plattformunabhängige Programmiersprachen / Virtuelle Maschinen / Multi-Plattform Toolkits	Java, Python und Smalltalk zählen zu den bekanntesten plattformunabhängigen Programmiersprachen. Zu Java gibt es bereits Entwicklungsumgebungen für diverse Betriebssysteme. GTK auf der anderen Seite ist eine plattformunübergreifende Bibliothek zur Implementierung graphischer Benutzeroberflächen.
	Komponentenmodelle	Komponentenmodelle wie EJB, CORBA oder auch .NET können in verschiedenen Programmiersprachen und womöglich Umgebungen umgesetzt werden.
Reaktionszeiten	Arbeitsflussanalyse	Durch die Analyse des Arbeitsflusses einer Funktion können Engpässe gefunden werden, die zu schlechten Reaktionszeiten führen. Darüber hinaus ist die Variabilität in den Arbeitsflüssen zu minimieren, die zu unterschiedlichen Reaktionszeiten führt.

Copyright © Fraunhofer IESE 2006

Performanz schmalband-breitiger Verbindungen	Komprimierung	Datenkompression minimiert das Volumen der zu übertragenden Daten
	Zwischenspeicherung	Die Zwischenspeicherung von Daten und Datenbankverbindungen spart Datenübetragungen.
	Batch-Zugriff	Ein gebündelter Datenzugriff ist oftmals weniger aufwendig als viele Einzelzugriffe.
Skalierbarkeit hinsichtlich Mehrbenutzerbetrieb	Parallelität	Parallel laufende Prozesse, Prozessoren oder Systeme tragen zur Skalierbarkeit bei. Dies setzt womöglich ein modulares System voraus, sodass Module parallel laufenden Prozessen zugeteilt werden können.
Datenskalierbarkeit	Replikation	Datenreplikation erhöht die Datenkapazität und trägt zum Lastausgleich bei.
	Optimistische Sperrstrategien	Bei den optimistischen Sperrstrategien werden Dateninkonsistenzen der Leistungsfähigkeit halber vorübergehend toleriert.
Systemverfügbarkeit	Parallelität	Parallelität trägt auch zur Verfügbarkeit bei, da bei Ausfall eines Prozesses die anderen parallellufenden Prozesse fortsetzen können.
	Sicherung	Regelmäßige Datensicherung (Sicherungspunkte) erlaubt die (partielle) Wiederherstellung von Daten im Falle eines Ausfalls.
Sicherheit	Authentifizierug, Autorisierung	Die Authentifizierung sorgt dafür, dass nur Befugten der Zugriff erlaubt wird. Die Autorisierung auf der anderen Seite legt fest, welche Aktionen eine Befugter vornehmen darf.
	Verschlüsselung	Zur Sicherung einer Datenübertragung werden die Daten verschlüsselt und sind somit für Unbefugte unlesbar.
Schemareplikation	Metadaten	Schema-Informationen werden in Metadaten gespeichert, die auch abgeglichen werden können (siehe auch 4.11)

Copyright © Fraunhofer IESE 2006

Referenzen

[1] Homepage des App2Web-Projektes, http://app2web.fzi.de

[2] Homepage der Workflow Management Coalition (WfMC), http://www.wfmc.org/

[3] E. Gamma et al., Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[4] Mobile Workplace Solutions from Cap Gemini Ernst & Young, Solution Blueprint Manufacturing, Copyright ©2003 CGEY. Intel

[5] Doug Stacey, Replication: DB2, Oracle or Sybase, SIGMOD Record. Volume 24, No 4, December 1995

[6] A. Kemper, A. Eickler, Datenbanksysteme, Eine Einführung, Oldenburg Verlag, 1999

[7] Werner Dreyer, A Meta Model and an Infrastructure for the Non-Transparent Replication of Object Databases, Dissertation der Wirtschaftswissenschaftlichen Fakultät der Universität Zürich, Juli 1999

[8] Enterprise Service Bus (ESB): Lasting concept or latest buzzword?, TechMetrix Research 2003, www.techmetrix.com

[9] Reinhold Biedermann, "Klempner vom Dienst, Integrationslösungen auf Basis des Open Source EAI-Toolkits openadaptor", XML Magaziin 2.04

[10] Jürgen Dunkel, Andreas Holischke, „Softwarearchitektur für die Praxis", Springer Verlag 2003

[11] Duncan Mackenzie, "Introducing Client Application Deployment with 'ClickOnce' ", Oktober 2003, http://msdn.microsoft.com/netframework/?pull=/library/en-us/dnwinforms/html/clickonce.asp

[12] Wolfgang Miedl , „Die Renaissance des Rich Client", http://www.computerwoche.de/index.cfm?pageid=255&artid=55380&type=detail&category=160

Copyright © Fraunhofer IESE 2006

[13] Michael Juntao Yuan, "Let the mobile games begin, Part 1 A comparison of the philosophies, approaches, and features of J2ME and the upcoming .Net CF" , Februar 2003, http://www.javaworld.com/javaworld/jw-02-2003/jw-0221-wireless-p3.html,

[14] Mike Padilla, "Create rich client apps with the DOM", IBM Developer Works 2004, http://www-106.ibm.com/developerworks/web/library/wa-richcli.html

[15] The Apache Software Foundation, 2004, www.apache.org

[16] Java Webstart Technology, 2004, http://java.sun.com/products/javawebstart/

[17] Thomas Bayer, "Java Web Services", Java Magazin, 08.2003

Copyright © Fraunhofer IESE 2006

Dokumenten Information

Titel:	Anforderungserfassung zum Projekt UNIVERSYS Entwicklung eines universell einsetzbaren verteilten Systems zur Betriebsführung
Datum:	Mai 2004
Report:	IESE-171.06/D
Status:	Final
Klassifikation:	Öffentlich

Copyright 2006, Fraunhofer IESE.
Alle Rechte vorbehalten. Diese Veröffentlichung darf für kommerzielle Zwecke ohne vorherige schriftliche Erlaubnis des Herausgebers in keiner Weise, auch nicht auszugsweise, insbesondere elektronisch oder mechanisch, als Fotokopie oder als Aufnahme oder sonstwie vervielfältigt, gespeichert oder übertragen werden. Eine schriftliche Genehmigung ist nicht erforderlich für die Vervielfältigung oder Verteilung der Veröffentlichung von bzw. an Personen zu privaten Zwecken.