Diplomarbeit

zur Erlangung des Grades eines
Diplom Wirtschaftsinformatikers (FH)
am Fachbereich Wirtschaft an der
Fachhochschule Nordostniedersachsen
in Lüneburg

Sommersemester 2003

Titel:

Design und prototypische Implementierung einer Komponente für das Java Enterprise Framework basierend auf der J2EE Connector Architecture

eingereicht von:	Maik Ahnfeldt Börnsener Str. 12 b 21039 Börnsen Tel.: 0170/9075217 mail@maik-ahnfeldt.de Matrikelnr.: 134468
spätester Abgabetermin:	06. Juli 2003
1. Gutachter:	Dipl.-Wirtschaftsinf. (FH) Thomas Slotos, MSc.
2. Gutachter:	Prof. Dr. rer. nat. Dieter Riebesehl
Betreuer (Praxis):	Dipl.-Informatiker Nail Araz

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

A2A	Application-to-Application
ABAP	Advanced Business Application Programming
ACID	Atomicity Consistency Isolation Durability
ALE	Application Link Enabling
API	Application Programming Interface
B2B	Business-to-Business
B2C	Business-to-Consumer
BAPI	Business Application Programming Interface
BOR	Business Object Repository
BPC	Business Process Connector
CCI	Common Client Interface
CORBA	Common Object Request Broker Architecture
CO	Controlling
DCOM	Distributed Component Object Model
DOT	Distributed Object Technology
DTD	Document Type Definition
EAI	Enterprise Application Integration
EIS	Enterprise Information System
ERP	Enterprise Resource Planning
FI	Finance
GUI	Graphical User Interface
HTML	HyperText Markup Language
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
JCA	J2EE Connector Architecture
Jco	Java Connector
JDBC	Java Database Connectivity
JDK	Java Development Kit

JEF	Java Enterprise Framework
JNDI	Java Native Directory Interface
JNI	Java Native Interface
JSP	Java Server Page
JVM	Java Virtual Machine
LHT	Lufthansa Technik AG
LSY	Lufthansa Systems Group GmbH
LSYBS	Lufthansa Systems Business Solutions GmbH
MM	Material Management
MOM	Message Oriented Middleware
MRO	Maintenance Repair and Overhaul
MVC	Model View Controller
ODBC	Open Database Connectivity
OMG	Object Management Group
PS	Project Systems
RAR	Resource Adapter Archive
RPC	Remote Procedure Call
SD	Sales and Distribution
SGML	Standard Generalized Markup Language
SM	Service Management
SQL	Structured Query Language
TPM	Transaction Processing Monitor
UI	User Interface
URL	Uniform Resource Locator
W3C	World Wide Web Konsortium
WSAD	WebSphere Studio Application Developer
XML	Extensible Markup Language

Kapitel 1

Einleitung

In den vergangenen Jahrzenten wurden Applikationen und Anwendungssysteme aufgrund der Spezialisierung der Hersteller auf bestimmte Aufgabengebiete mit spezifischen Technologien, die nicht zwingend kompatibel zueinander waren, implementiert. Durch den Einsatz von Internettechnologien konnten Gechäftsprozesse unabhängig von Zeit, Ort und der zugrundeliegenden Hardware-Plattform effizient abgewickelt werden. Dazu war es notwendig, die in den einzelnen Applikationen und Anwendungssystemen abgebildeten Teilprozesse systemübergreifend in einer heterogenen Systemlandschaft zu kombinieren. Es wurde versucht, die älteren Technologien zu erweitern, anzupassen oder auszutauschen, so dass diese mit neueren Technologien kommunizieren konnten. Dieser Ansatz war jedoch mit hohem Risiko und Kosten verbunden. Aus diesen Gründen ist die Forderung und der Bedarf entstanden, auf existierende Softwaresysteme zuzugreifen, ohne diese verändern bzw. manipulieren zu müssen.

1.1 Motivation

Die Lufthansa Systems Business Solutions GmbH (LSYBS) entwickelt Applikationen auf Basis der Java 2 Enterprise Edition (J2EE)[1]. Der grösste Kunde der LSYBS ist die Lufthansa Technik AG (LHT) im Geschäftsbereich Maintenance, Repair and Overhaul (MRO)[2]. Bei der LHT werden verschiedene Enterprise Information Systems (EISs)[3], welche Geschäftsprozesse und Daten kapseln, eingesetzt. Hersteller von EIS bieten für den Zugriff auf die Geschäftsprozesse und Daten i.A. eigene, nicht standardisierte Mechanismen an, die ebenfalls bei der LHT im Einsatz sind. Anwendungsentwickler müssen sich mit den unterschiedlichen Mechanismen auseinandersetzen, woraus ein hoher Aufwand resultiert. Ein weiteres Defizit ist, dass der Zugriffsmechanismus einiger EIS Hersteller keine Transaktionssteuerung und Sicherheits-Management unterstützt. Im Rahmen dieser Diplomarbeit soll untersucht werden, ob die eben genannten Defizite durch den Einsatz der J2EE Connector Architecture (JCA) behoben werden können.

1.2 Ziel

Das Java Enterprise Framework (JEF) ist eine Eigenentwicklung der LSYBS. Durch JEF wird die Implementierung von J2EE Applikationen standardisiert. Die Grundfunktionalität von JEF wird durch Komponenten erweitert. Das Hauptziel dieser Diplomarbeit ist die Entwicklung einer JEF Komponente, die auf Basis von JCA die Einbindung von bestehenden EIS in zukünftige JEF Anwendungen vereinfachen soll. Die von der LHT gestellten Forderungen an die Komponente werden auf Erfüllbarkeit untersucht. Ferner soll der Einsatz der Komponente anhand eines beispielhaften

[1] Java 2 Enterprise Edition siehe Glossar.

[2] Unter MRO werden Dienstleister zusammengefasst, die Flugzeuge reparieren und überholen.

[3] Enterprise Information System siehe Glossar

Zugriffs auf SAP R/3[4] verdeutlicht werden.

1.3 Vorgehensweise

Zunächst erfolgt im zweiten Kapitel eine allgemeine Diskussion über Enterprise Application Integration (EAI). Hier werden die fachlichen Grundlagen und Ansätze für die Integration von EIS erläutert.
Im dritten Kapitel wird eine Anforderungsanalyse erstellt. Die beteiligten Unternehmen sowie deren bisher verwendeten Mechanismen zur EAI werden kurz beschrieben. Die innerbetrieblichen- und Kundenanforderungen an die zu entwickelnde Komponente werden erläutert.
Das vierte Kapitel geht auf die technischen Grundlagen ein. Es erfolgt eine Erläuterung der verwendeten Architekturen und Technologien, die zur Realisierung der Komponente verwendet werden.
Der Entwurf der Komponente sowie eine Diskussion über die Umsetzbarkeit der Anforderungsanalyse erfolgt im fünften Kapitel.
Zum Abschluss erfolgt im sechsten Kapitel die technische Umsetzung des Entwurfs. Dieser wird durch einen beispielhaften Zugriff auf SAP R/3 erläutert.

1.4 Konvention

Der Entwurf und die Implementierung der JEF Komponente orientiert sich an den Verträgen der JCA Spezifikation 1.0. Die in der Spezifikation 1.5 zusätzlichen Verträge finden keine Berücksichtigung.

[4] SAP R/3 siehe Glossar.

Kapitel 2

Enterprise Application Integration

2.1 Übersicht

Seit Mitte der neunziger Jahre existiert der Begriff des Enterprise Application Integration (EAI). Der Begriff findet eine häufige Verwendung. Trotz dieser vielfachen Verwendung gibt es keine einheitliche Definition. Zusätzlich sind die Ziele sowie die Hindernisse bei der EAI oft nicht bekannt.

2.1.1 Definition

Zunächst eine von zahlreichen Definition von EAI:

> „Enterprise Application Integration is a set of technologies that allows the movement and exchange of information between different applications and business processes within and between organizations.“ [ebi03]

Für die Implementierung von Applikationen werden stets die neuesten Technologien eingesetzt. Diese sind nicht zwingend kompatibel zueinander. Daraus resultiert, dass

aktuelle Applikationen nicht mit bereits bestehenden Applikationen interagieren können. In der EAI werden daher Technologien bereitgestellt, die ein Austauschen von Informationen zwischen diesen heterogenen Applikationen ermöglicht. Wofür dieser Austausch von Informationen dient, wird jedoch nicht in der vorangegangenen Definition von [ebi03] angesprochen.

> „Es geht also darum, heterogene Anwendungen eines Unternehmens so zu integrieren, dass sie sich möglichst so verhalten, als wären sie von Anfang an dafür entworfen worden, die aktuellen Geschäftsprozesse eines Unternehmens zu unterstützen.“ [Kel02, Seite 5]

Diese beiden Zitate werden zu einer Definition von EAI zusammengefasst, welche in dieser Diplomarbeit gültig ist:

> Enterprise Application Integration ist eine Menge von Technologien, die eine Integration der in den heterogenen Applikationen abgebildeten Teilprozesse und Daten ermöglichen, damit diese Teilprozesse und Daten wie die aktuellen Geschäftsprozesse eines oder mehrerer Unternehmen wirken.

2.1.2 Ziele

Nachdem die Definition von EAI erfolgt ist, wird auf die Ziele von EAI eingegangen. In der gültigen Definition von EAI ist das Hauptziel bereits angegeben: Die Integration von Teilprozessen und Daten. Aus dem Streben, das Hauptziel zu erreichen, resultieren weitere Ziele (vgl. [RMB01, Seite 12]):

- Durch EAI wird festgestellt, welche Applikation welche Geschäftsprozesse und Daten abbilden. Dadurch kann eine Übersicht über die Systemkomplexität und

- eine Vermeidung redundanter Daten sowie Funktionalität erreicht werden.
- Im Zuge von EAI kann die Performance der beteiligten Applikationen überprüft und optimiert werden.

2.1.3 Hindernisse

Das Erreichen der eben genannten Ziele bei der EAI wird durch häufig auftretende Hindernisse erschwert (vgl. [RMB01, Seite 12]):

- Die Organisation und die Struktur von Unternehmen ist komplex. Somit ist die Funktionalität und die Architektur der verwendeten Applikationen unüberschaubar.
- Der technische Fortschritt bei EAI Technologien führt zu einem gewaltigen Bedarf an Fachpersonal, die über ein breites und aktuelles Wissen über diese Technologien verfügen.
- Durch die Vernetzung des Unternehmens nach innen und aussen müssen Sicherheitsaspekte eine höhere Berücksichtigung finden.

Zu Beginn von EAI ging es zunächst nur um die Integration von Applikationen innerhalb eines Unternehmens (Application-to-Application). Heute existieren weitere EAI Formen.

2.1.4 Formen

Es lassen sich drei Formen von EAI identifizieren (vgl. [Sai01, Seite 209]):

Application-to-Application (A2A) Bei der Application-to-Application Integration geht es um die Integration von Applikationen innerhalb eines Unternehmens. Deshalb wird A2A häufig auch als interne Integration bezeichnet.

Business-to-Business (B2B) Business-to-Business ist durch eine Integration von Applikationen über die Unternehmensgrenzen hinweg gekennzeichnet. Die Technologien des Internets haben für die Entstehung dieser Integrationsform einen massgeblichen Anteil. Als Beispiel dient hier der Zugriff von einem SAP R/3-System auf ein Enterprise Resource Planning (ERP)[1] System eines anderen Unternehmens, um eine Bestellung durchzuführen. Als Synonym für B2B wird vielfach externe Integration verwendet.

Business-to-Consumer (B2C) Die letzte Integrationsform ist Business-to-Consumer, die auch als Web-GUI-Integration bezeichnet wird. Dabei soll ein Endkunde auf die Geschäftsprozesse eines Unternehmens zugreifen. Es geht bei dieser Form also weniger um die Integration von Applikationen, sondern darum, einen Endkunden in den Geschäftsprozess zu integrieren. Dieses geschieht typischerweise über eine Web-basierte Applikation.

Soweit zu den Zielen, Hindernissen und Formen von EAI. Im nächsten Abschnitt erfolgt eine Beschreibung der Integrationslevel.

2.2 Integrationslevel

Jeder Integrationslevel ist durch bestimmte Merkmale charakterisiert. Diese Mermale sind auszugsweise (vgl. [RMB01, Seite 18]):

- Komplexität der Integration,

[1] Enterprise Resource Planning siehe Glossar.

- welche Integrationstechniken vorhanden sind und
- welches Wissen benötigt wird, um die Integration durchzuführen.

Die Merkmale finden sich teilweise in der Beschreibung der Integrationslevel wieder.

2.2.1 Präsentationslevel

Der einfachste Integrationslevel ist der Präsentationslevel. Das Konzept des Präsentationslevel ist, dass auf das User Interface (UI)[2] der zu integrierenden Applikation zugegriffen wird. Durch dieses Konzept wird es einem neu entwickelten User Interface ermöglicht, die benötigten Informationen aus dem alten UI abzubilden. Jede Interaktion auf dem neuen UI wird an das entsprechende UI weitergeleitet. Die Schnittstelle für die EAI ist also das User Interface einer bestehenden Applikation. Der Vorteil von Integration auf dem Präsentationslevel besteht darin, dass dieser Ansatz schon länger bekannt ist und eine hohe Anzahl von erprobten Tools für diese Integration existiert. Auf der anderen Seite kann nur auf die Geschäftsprozesse und Daten zugegriffen werden, die das alte UI anbietet. (vgl. [RMB01, Seite 22])

2.2.2 Datenlevel

Die Integration auf dem Datenlevel zielt direkt auf die Datenschicht einer Applikation. Das Konzept der Integration auf dem Datenlevel sieht vor, dass heterogene Applikationen sich die Geschäftsdaten teilen. Dazu existieren unterschiedliche Ansätze. Ein Ansatz ist, dass heterogene Applikationen auf ein und dieselbe Datenquelle zugreifen. Hierfür existieren bereits standardisierte und applikationsunabhänge Mechanismen. Die Daten werden dabei zur Laufzeit in applikationsabhängige Datenformate umgewandelt. Ein weiterer Ansatz ist, dass die Geschäftsdaten von einer

[2]Schnittstelle, die einem Anwender die Interaktion mit einer Applikation ermöglicht.

Datenquelle auf weitere Datenquellen transferiert werden. Für beide Ansätze existieren bewährte Werkzeuge. Die Integration auf dem Datenlevel ist flexibler als die Integration auf dem Präsentationslevel, da sämtliche Geschäftsdaten zur Verfügung stehen. Ein erheblicher Nachteil ist, dass mehrere Applikation von einem Datenmodell abhängig sind. Wird ein Datenmodell manipuliert, müssen die Zugriffe auf die Datenschicht in den Applikationen angepasst werden. (vgl. [RMB01, Seite 24])
Ein weiteres Hinderniss bei diesem Integrationslevel ist, dass bei einigen Applikationen keine Trennung zwischen Geschäftslogik und Datenschicht existiert (vgl. [Cum02, Seite 24]).

2.2.3 Funktionslevel

Ein hoher Anteil von IT Budgets wird für die Implementierung von Geschäftslogik verwendet. Es liegt daher nah, die Geschäftslogik so zu konzipieren, dass diese auch von weiteren Applikationen verwendet werden kann. Die Integration auf dem Funktionslevel erfolgt über den Zugriff auf dem Code einer Applikation. Der einfachste Fall liegt vor, wenn ein Application Programming Interface (API)[3] für die Geschäftslogik vorliegt. Der schlechteste Fall ist es, wenn zusätzlicher Code benötigt wird, um die Geschäftslogik wiederverwenden zu können. Dieser Ansatz ist zwar technisch komplex, aber flexibler. Es wird mehr Geschäftslogik angeboten als bei der Integration auf dem Präsentationslevel. Zusätzlich erfolgt die Manipulation der Daten in der Geschäftslogik. (vgl. [RMB01, Seite 27])

Auf die Techniken für die unterschiedlichen Integrationslevel wird in Abschnitt 2.3.3 eingegangen. Bevor aber detaillierter auf diese Techniken eingegangen wird, erfolgt eine Beschreibung der wesentlichen Bestandteile von EAI.

[3] Application Programming Interface siehe Glossar.

2.3 Bestandteile

Enterprise Application Integration basiert auf 3 Bestandteilen:

Kommunikationsmodel definiert die Art der Kommunikation (snychron/asynchron).

Integrationsmechanismen beschreibt die Mechanismen, mit denen eine Anfrage zum Empfänger[4] gesendet werden kann.

Middleware beschreibt die Technologien, welche die Integration zwischen heterogenen Applikationen ermöglicht.

2.3.1 Kommunikationsmodel

Das Kommunikationsmodell legt fest, wie der Informationsaustausch zwischen den zu integrierenden Applikationen erfolgt. Hierbei wird zwischen synchroner und asynchroner Kommunikation unterschieden.

Synchrone Kommunikation

Bei der synchronen Kommunikation wartet der Sender[5] auf eine Antwort vom Empfänger, bevor dieser mit seinem Prozess fortfährt. Verwendet wird dieses Kommunikationsmodell, wenn der weitere Verlauf des Prozesses abhängig von der Antwort des Empfängers ist. Es existiert eine starke Kopplung zwischen Sender und Empfänger. Webbasierte Applikationen benötigen typischerweise synchrone Kommunikation. Ein Anwender löst eine Aktion aus und wartet bis zur Antwort, bevor er

[4]Empfänger siehe Glossar.

[5]Sender siehe Glossar.

weitere Aktionen auslöst. Als konkretes Beispiel dient hier ein webbasierter Onlineshop, in dem der Anwender einen Artikel zu seinem Warenkorb hinzufügen möchte. Auf die Aktion „Artikel zum Warenkorb hinzufügen“ erwartet der Anwender eine Bestätigung, bevor dieser weitere Aktionen auslöst.

Es existieren drei Typen der synchronen Kommunikation: (vgl. [RMB01, Seite 42])

Request/Reply Request/Reply ist der Basistyp für die synchrone Kommunikation. Eine Applikation sendet eine Anfrage zu einer weiteren Applikation. Der Prozess wird erst fortgesetzt, wenn eine Antwort vom Empfänger gesendet wird. Die Antwort kann eine einfache Bestätigung für den erfolgreichen Empfang der Anforderung sein oder eine komplexe Ergebnismenge. Request/Reply wird dann eingesetzt, wenn die Antwort Informationen enthält, die den weiteren Prozessverlauf bestimmen. Request/Reply hat jedoch zwei entscheidende Nachteile: Das Laufzeitverhalten wird durch lange Antwortzeiten negativ beeinflusst. Des weiteren kann es sein, dass der Prozess des Senders nicht fortgesetzt werden kann, da der Empfänger Probleme bei der Erzeugung oder Übertragung des Ergebnis hat.

One-Way One-Way ist der einfachste Typ der synchronen Kommunikation. Der Sender sendet eine Anfrage. Der Prozess des Senders wird solange unterbrochen, bis dieser eine Bestätigung für den Empfang der Anfrage vom Empfänger erhält. Dieser Typ findet Verwendung, wenn eine Synchronisierung zwischen den Prozessen des Senders und Empfängers nötig ist. Das Laufzeitverhalten ist gegenüber Request/Reply verbessert. Der Empänger muss nur eine Bestätigung für den Empfang der Anfrage senden. Das Erzeugen und Senden dieser Bestätigung ist typischwerweise nicht so komplex wie das Erzeugen und Senden der eigentlichen Ergebnismenge.

Polling Bei diesen Typ der synchronen Kommunikation wird dem Sender gestattet,

den Prozess nach dem Senden einer Anfrage eingeschränkt fortzuführen. Der Sender wartet nach dem Abschicken der Anfrage nicht auf eine Antwort, sondern setzt seinen Prozess fort. Der Prozess wird periodisch unterbrochen, damit der Sender überprüfen kann, ob eine Antwort vorliegt. Liegt eine Antwort vor, entfällt das periodische Unterbrechen des Prozesses. Dieser Typ wird verwendet, wenn der Prozess des Senders nur teilweise von der Antwort abhängig ist. Vom Laufzeitverhalten gesehen ist dieser Typ gegenüber Request/Reply und One-Way optimaler. Dem gegenüber steht die aufwändige Implementierung des Polling.

Asynchrone Kommunikation

Bei der asynchronen Kommunikation braucht der Sender seinen Prozess nicht zu unterbrechen, um auf eine Antwort vom Empfänger zu warten. Der Prozess des Senders ist unabhängig von der Antwort des Empfängers. Die Kopplung zwischen Sender und Empfänger ist also nicht so stark wie bei der synchronen Kommunikation. Es existieren drei weit verbreitete Typen der asynchronen Kommunikation:

Point-to-Point Der Sender schickt eine Nachricht[6] an eine bestimmte Queue (Schlange)[7] und setzt dann seinen Prozess fort. Die Nachricht verweilt solange in der Queue, bis diese entweder ihre Gültigkeit verloren hat oder von einem Empfänger konsumiert wurde. Es existiert dabei nur ein Empfänger für diese Nachricht. Die Nachrichten in der Queue werden typischerweise persistiert, wodurch selbst nach einem Systemabsturz der Empfänger die Nachricht erhalten sollte. (vgl. [ABD01, Seite 970])

[6] Nachricht siehe Glossar.

[7] „Eine Folge heisst Schlange, wenn Elemente nur am Ende eingefügt und am Anfang entfernt werden dürfen...“ [Eng93, Seite 620]

Publish/Subscribe Der Sender schickt seine Nachricht nicht an eine Queue, sondern an ein „Topic“. Bei dem Topic handelt es sich um ein bestimmtes Themengebiet, bei der sich interessierte Empfänger registrieren können. Die Nachricht wird dann an alle Empfänger geschickt, die sich für diesen Topic registriert haben. (vgl. [ABD01, 971])

Broadcast Die Nachricht wird an alle potentiellen Empfänger geschickt. Der Empfänger entscheidet dabei, ob die Nachricht für ihn relevant ist. Der Nachteil bei diesem Typ ist, dass der Empfänger alle eingehenden Nachrichten auf Relevanz überprüfen muss. Ab einer bestimmten Anzahl von Nachrichten ist der Empfänger mehr damit beschäftigt, die eingehenden Nachrichten auf Relevanz zu überprüfen, als die für die jeweilige Nachricht definierte Geschäftslogik abzuarbeiten. (vgl. [RMB01, Seite 47])

2.3.2 Integrationsmechanismen

Im Abschnitt 2.3.1 Kommunikationsmodel wurde vorgestellt, wie der Informationsaustausch über die synchrone und asynchrone Kommunikation zwischen den zu integrierenden Applikationen erfolgt. Folgend stellt sich die Frage, welche Mechanismen überhaupt existieren, um eine Anfrage zu erzeugen und diese an einen Empfänger zu senden. Oder konkreter ausgedrückt: Welche Möglichkeiten gibt es, die Operationen[8] von Applikationen aufzurufen?

Laut [RMB01, Seite 48] existieren hierfür zwei Möglichkeiten:

Messaging Messaging sieht vor, dass der Sender eine Nachricht erzeugt und zu einem oder mehreren Empfängern schickt. Diese Nachricht enthält Informationen darüber, welche Operation ausgeführt werden soll und die dafür benötigten

[8] Operation siehe Glossar.

Daten. Da die Nachricht alle benötigten Informationen für das Ausführen einer Operation enthält, ist eine geringe Kopplung zwischen Sender und Empfänger gegeben. Demgegenüber steht, dass die Nachricht ein Format besitzen muss, welches sowohl der Sender als auch der Empfänger versteht. Der Erzeuger von diesen Nachrichten, typischerweise der Anwendungsentwickler, muss dieses Format kennen.

Schnittstellen Dagegen wird über wohldefinierte Schnittstellen dem Anwendungsentwickler ein Gerüst vorgegeben, wie er eine Operation aufrufen kann und wie er die dafür benötigten Daten übergibt. Dieser Ansatz ist aus Sicht des Anwendungsentwickler einfacher und weniger fehleranfällig als das Erzeugen einer Nachricht mit einem bestimmten Format. Der Nachteil von Schnittstellen ist, dass die Kopplung zwischen Sender und Empfänger stärker als bei Messaging ist.

Die beiden Möglichkeiten für den Aufruf einer Operation werden in einem sogenannten „Connector“ zusammengefasst.

> „A Connector is logic that is programmed into an application whose sole purpose is to provide access to the presentation, data, or functionality of the application in a structured manner. The Connector hides the complexity of translating and communicating a message or an invocation on an interface for use by the application.“[RMB01, Seite 51]

Aus Sicht eines Anwendungsentwicklers ist ein Connector der „single point of access“ für das Erzeugen einer Anfrage an einem beliebigen Empfänger. Zusätzlich wird durch ein Connector die Komplexität der Middleware vor dem Anwendungsentwickler verborgen. Auf Middleware wird in dem folgenden Abschnitt 2.3.3 detaillierter eingegangen.

2.3.3 Middleware

Es gibt keine einheitliche Definition für den Begriff Middleware. Im Rahmen von EAI wird Middleware als Technologie gesehen, welche die Interaktion zwischen heterogenen Applikationen ermöglicht. In dieser Diplomarbeit gilt folgende Definition für den Begriff Middleware:

> Middleware ist der Oberbegriff für eine Menge von Technologien, welche die Interaktion zwischen heterogenen Applikationen über wohldefinierte Schnittstellen oder Nachrichten ermöglicht.

Es existieren 5 Basistypen für Middleware: (vgl. [RMB01, Seite 52]

- Remote procedure call
- Database access middleware
- Message oriented middleware
- Distributed object technology
- Transaction processing monitor

Jeder dieser Basistypen wurde für ein bestimmtes Problemgebiet entworfen, das sich im Zusammenhang mit der Integration von Applikationen ergab. Jeder Basistyp ist für ein oder mehrere Integrationslevel ausgelegt und unterstützt synchrone und/oder asynchrone Kommunikation. Im Folgenden wird detaillierter auf die einzelnen Basistypen eingegangen.

Remote procedure call

Remote procedure call (RPC) erlaubt den Aufruf von Prozeduren innerhalb eines Netzwerkes. Dabei benötigen der Sender und der Empfänger des Aufrufes keinerlei

Kenntnisse über die zugrundeliegende Netzwerktopologie. RPC ist dem Funktionslevel zuzuorden und unterstützt nur synchrone Kommunikation. Der Sender muss seinen Prozess also unterbrechen, bis der Empfänger eine Antwort zurückliefert. RPC wurde in den siebziger Jahren erstmalig eingesetzt und ist stark mit dem prozeduralen Paradigma assoziiert. Durch die zunehmende Verbreitung des objektorientierten Paradigmas wurde RPC zunehmend durch andere Middleware, wie *distributed object technology* oder *message oriented middleware*, ersetzt. RPC findet häufig nur noch in den Unternehmen Verwendung, die noch mit prozeduralen Sprachen wie C oder Turbo Pascal entwickeln.

Database access middleware

Database access middleware erlaubt den Zugriff von heterogenen Applikationen auf verteilte Datenbanken und Dateien. Der Präsentationslevel und Funktionslevel wird dabei umgangen. Die Hersteller von Datenbanken bieten propietäre Mechanismen für den Zugriff auf ihre Datenbank an. Für die Manipulation der Daten existierte zwar ein Standard, nämlich die Structured Query language (SQL), aber nicht für den Zugriff auf die Datenbanken. Durch Open Database Connectivity (ODBC) wird eine Menge von Schnittstellen definiert, die einen standardisierten Zugriff auf relationale Datenbanken bieten. ODBC ist seit den achtzigern Jahren ein weit verbreiteter Standard.

Message oriented middleware

Message oriented middleware (MOM) unterstützt die Integration von heterogenen Applikationen durch das Austauschen von Nachrichten. MOM konvertiert die Nachrichten in ein für den Sender und Empfänger verständliches Format. Zusätzlich unterstützt MOM das Erzeugen, Persistieren und Versenden von Nachrichten. Wie

in Abschnitt 2.3.2 erläutert, enthalten die Nachrichten die vom Empfänger auszuführende Operation und die dafür benötigten Daten. MOM ist besonders gut für das Auslösen von Aktionen in verteilten Applikationen und das konsistente Halten von verteilten Daten geeignet.

Distributed object technology

Distributed object technology (DOT) ist RPC ähnlich, nur das es auf das objektorientierte Paradigma ausgerichtet ist. Durch DOT besteht die Möglichkeit, auf die Schnittstellen von Objekten zuzugreifen, die in einem Netzwerk verteilt sind. Ürsprünglich war DOT nur für die synchrone Kommunikation ausgerichtet. Um eine grössere Flexibilität bei der Integration zu erreichen, wurde DOT um die asynchrone Kommunikation erweitert. Die bekanntesten Vertreter von DOT sind die Common Object Request Broker Architecture (CORBA) von der Object Management Group (OMG), Microsofts Distributed Component Object Model (DCOM) und Enterprise Java Beans (EJBs)[9] von Sun Microsystems.

Transaction processing monitors

Durch Transaction processing monitors (TPM) werden die Transaktion über mehrere Applikationen hinweg verwaltet und gesteuert.
Eine Transaktion definiert eine logische Verarbeitungseinheit, die aus mehreren Operationen besteht. Für diese logische Verarbeitungseinheit gelten die sogenannten ACID Eigenschaften (vgl. [SSN01, Seite 55]):

Atomicity (Atomarität) Die in einer Transaktion beabsichtigten Änderungen werden entweder vollständig oder überhaupt nicht durchgeführt.

[9]Enterprise Java Bean siehe Glossar.

Consistency (Konsistenz) Die Applikation wird von einem konsistenten Zustand in einem anderen konsistenten Zustand geführt.

Isolation (Isolation) Werden mehrere Transaktionen parallel ausgeführt, muss das Ergebnis demjenigen einer seriellen Ausführung identisch sein.

Durability (Dauerhaftigkeit) Die Wirkung einer abgeschlossenen Transaktion muss dauerhaft sein.

Ein TPM muss dafür sorgen, dass die ACID Eigenschaften für Transaktionen gewährleistet werden. Durch den zunehmenden Bedarf, Transaktionen über verteilte Applikationen hinweg steuern zu können, wuchs auch die Forderung nach einem Standard. Diese Forderung wurde durch die X/Open CAE Spezifikation, welche die XA Schnittstelle beschreibt, erfüllt. TPM Funktionalität findet sich häufig in MOM und DOT wieder und ist der komplexeste Middlewaretyp.

2.4 J2EE und EAI

In der J2EE Version 1.2 existierte nur ein Standard für die Integration von relationalen Datenbanken, die Java Database Connectivity (JDBC). Ansonsten bot J2EE eine hohe Anzahl von APIs, welche Mechanismen für die Integration von EISs anboten. Ein Standard für die Verwendung dieser Mechnismen existierte nicht. Ab der J2EE Version 1.3 ist die J2EE Connector Architecture (JCA) fester Bestandteil der Spezifikation. Laut Aussage von SUN wird JDBC weiterhin als Standard für die Integration von relationalen Datenbanken bestehen bleiben. Für die Integration von weiteren EIS in J2EE basierte Applikationen soll die JCA zukünftig Verwendung finden (vgl. [Jey02, Seite 11]).

Der Integrationslevel für JCA ist auf den Datenlevel und den Funktionslevel ausgerichtet. Weiterhin findet nur die A2A Integration in dieser Diplomarbeit Berücksichtigung. Die JCA wird detaillierter in Abschnitt 4.2 behandelt. Eine Begründung, warum die zu entwickelnde Komponente auf der JCA basieren wird, erfolgt in der Anforderungsanalyse.

Kapitel 3

Anforderungsanalyse

In diesem Kapitel soll zuerst auf die Unternehmen eingegangen werden, die für die Diplomarbeit relevant sind. Danach folgt eine Istanalyse über die Integration der vorhandenen EISs. Schliesslich werden die innerbetrieblichen- und Kundenanforderungen an die zu entwickelnde Komponente aufgelistet.

3.1 Unternehmensprofil

Es folgt eine kurze Beschreibung der Lufthansa Systems Group GmbH, der Lufthansa Systems Business Solutions GmbH und der Lufthansa Technik AG.

3.1.1 Lufthansa Systems Group GmbH

Die Lufthansa Systems Group (LSY) wurde am 10. Oktober 1994 mit dem Ziel gegründet, die IT Kompetenz des Lufthansa Konzerns in einem flexiblen Unternehmensverbund zu bündeln. Zur Zeit besteht die Gruppe aus 16 Unternehmen, welche weltweit ca. 4500 Mitarbeiter beschäftigen. 2001 wurde ein konsolidierter Jahres-

umsatz von 748,30 Millionen Euro (ohne Binnenumsatz) erwirtschaftet. Die LSY gliedert sich in drei Bereiche, wobei jedem Bereich eine Kernkompetenz zugeordnet ist. (vgl. [Luf02] und [Luf01])

Airline Der Bereich Airline ist auf Fluggesellschaften ausgerichtet. Typische Aufgaben für diesen Bereich sind beispielsweise die digitale Erfassung und Abrechnung von Flugtickets sowie die Einsatzplanung des Kabinenpersonals.

Infrastructure Die Kompetenz des Bereiches Infrastructure ist die Betreuung von Desktops und Servern, sowie der Aufbau und die Wartung von Netzwerken.

Aviation Zu dem Bereich Aviation zählen alle Unternehmen, die in der Luftfahrtbranche tätig sind, ausgenommen die Fluggesellschaften. Die Lufthansa Systems Business Solutions GmbH (LSYBS) gehört in diesen Bereich.

3.1.2 Lufthansa Systems Business Solutions GmbH

Die Lufthansa Systems Business Solutions GmbH (LSYBS) wurde am 26. September 2000 gegründet. Zur Zeit werden ungefähr 438 Mitarbeiter in Frankfurt und Hamburg beschäftigt. Der Jahresumsatz 2001 betrug 89,6 Millionen Euro. Der Schwerpunkt des Unternehmens liegt in der Geschäftsprozessberatung von Unternehmen der Aviation- sowie der Travel- und Transportation-Branche. Es werden geeignete Anwendungssysteme für diese Geschäftsprozesse entwickelt, implementiert und gewartet. Eine Business Unit der LSYBS ist *Maintenance & Engineering* (HAM AM). Diese ist auf das Tätigkeitsfeld von Technikdienstleistern in der Luftfahrtindustrie ausgerichtet. Der grösste Kunde dieser Business Unit ist die Lufthansa Technik AG. (vgl. [Luf02] und [Luf01])

3.1.3 Lufthansa Technik AG

Die Lufthansa Technik AG (LHT) ist ein Anbieter für technische Dienstleistungen in der Luftfahrtindustrie. Die Dienstleistungen reichen von der Instandhaltung, der Überholung über die komplette Neuausstattung von Flugzeugen bis hin zur technischen Beratung und Forschung. Die Lufthansa Technik AG wurde im Oktober 1994 gegründet. In 60 weltweiten Niederlassungen arbeiten insgesamt 11000 Mitarbeiter. 2001 wurde ein Jahresumsatz von 2 Milliarden Euro erwirtschaftet. (vgl.[Luf03a])

3.2 Istsituation

In der Luftfahrtindustrie muss eine grosse Menge an Informationen verarbeitet werden. Jedes sicherheitsrelevante Teil hat einen dokumentierten Lebenslauf, wobei ein Triebwerk alleine schon aus 38.000 einzelnen Bauteilen besteht. Jede Minute, die ein Flugzeug im Reparaturdock verbringt, verursacht der Airline Kosten:

> „Ein Jumbo-Jet, der mit einem Defekt am Boden steht, kostet seinem Betreiber fast 50.000 Mark Zinsen am Tag und eine halbe Million an Umsatzausfall.“ ([Luf03b])

Zeit ist also ein wichtiger Wettbewerbsfaktor. Durch die hohe Verfügbarkeit von Informationen und durch gut abgestimmte Prozesse lässt sich dieser Wettbewerbsfaktor positiv beeinflussen. Die Abbildung dieser Prozesse geschieht bei der LHT hauptsächlich durch das SAP R/3 System. Für die Archivierung von Dokumenten werden FileNet Komponenten eingesetzt.

3.2.1 SAP R/3

Das SAP R/3 Release 3.0 wurde im Mai 98 produktiv gesetzt und seit dem 01. Juni 2002 ist SAP R/3 4.6 C2 im Einsatz. Zur Zeit greifen ca. 4500 Dialoguser auf das SAP R/3-System zu. Es sind 4 Systeme (Schulungs-, Entwicklungs-, Spiegel- und Produktionssystem) mit insgesamt 11 Mandanten eingerichtet. Folgende Module werden eingesetzt (vgl. [Mal03]):

- Controlling (CO)
- Finance (FI)
- Material Management (MM)
- Project Systems (PS)
- Sales and Distribution (SD)
- Service Management (SM)

Für die Wartung und die Eigenentwicklung von SAP R/3 ist die LSYBS zuständig.

3.2.2 FileNet

Für die Archivierung von Dokumenten werden bei der LHT zwei Komponenten des Unternehmens FileNet Corporation eingesetzt: FileNet Panagon Image Services V 3.4.2 und FileNet Panagon DocWarehouse V 4.0. Folgende Dokumenttypen werden bei der LHT über diese Komponenten archiviert (vgl. [Sta03]):

- Eingangsrechnungen
- Ausgangsrechnungen

- Betriebsmittelzeichnungen (CAD Dokumente)
- Werkstatt Wareneingangsbelege
- Technische Dokumentation

3.2.3 J2EE Applikationen

Bei der LSYBS werden seit ca. 2 Jahren Applikationen auf Basis von J2EE entwickelt. Durch ein Kundenprojekt, welches über mehrere Standorte der LSYBS erfolgte, entstand die Idee und die Motivation zum Java Enterprise Framework (JEF). Es existierte kein Standard für die Implementierung von J2EE Applikationen. Die Entwicklerteams der jeweiligen Standorte hatten die J2EE Applikationskomponenten nicht einheitlich verwendet, wodurch sich Inkompabilitäten bei der Zusammenführung ergaben. Um dieses Problem in Zukunft zu vermeiden, wurde JEF von der LSYBS entwickelt. JEF, auf das in Kapitel 4.1 näher eingegangen wird, basiert auf J2EE und dient als Grundlage für zukünftige J2EE Applikationen bei der LSYBS. Die Funktionalität von JEF ist durch eine Komponentenarchitektur erweiterbar. Im Rahmen dieser Diplomarbeit soll eine Komponente entwickelt werden, welche die Integration von EIS in JEF Applikationen standardisiert. Die Bezeichnung für diese Komponente ist *JEF Connector*.

3.2.4 Integration

In diesem Abschnitt werden die bisherigen Integrationsansätze von SAP R/3 und FileNet vorgestellt.

SAP R/3 Die Entscheidung fällt zugunsten einer J2EE Applikation aus, wenn der Aufwand für die Erweiterung bzw. Modifikation des SAP R/3-Systems

den Nutzen übersteigt. Da die Kernprozesse in SAP R/3 abgebildet werden, bleibt der Zugriff auf das SAP R/3-System unvermeidbar. Die Integration von SAP R/3 in J2EE Applikationen erfolgte über den Middlewaretyp RPC. Die Schnittstellen für das Transaktion- und Sicherheits-Management wurden bei der LSYBS selbst entwickelt. Die Implementierung der RPC erfolgt in der prozeduralen Programmiersprache C. Die C Routinen wurden über das Java Native Interface (JNI)[1] oder C aufgerufen. Der Aufwand für diese Implementierung in C und Java ist sehr hoch. Dazu kommt noch die Vermischung zwischen dem objektorientierten Paradgima bei Java mit dem prozeduralen Paradigma in C. Zusätzlich kritisch bei dieser Lösung ist, dass nur noch ein Mitarbieter mit ausgeprägten C und SAP R/3 Kenntnissen bei der LSYBS vorhanden ist.

FileNet Der direkte Zugriff auf FileNet Komponenten wird zum grössten Teil über das SAP R/3-System durchgeführt (vgl. [Sta03]). Für die Integration von FileNet Komponenten in J2EE Applikationen existierte bisher kein Bedarf. In zukünftigen J2EE Applikationen könnte sich die Anforderung ergeben, direkt auf FileNet Komponenten zuzugreifen.

Die bisherigen Lösungen für die Integration von EISs in J2EE Applikationen sind für zukünftige Projekte nicht ausreichend oder noch gar nicht vorhanden. Die Entscheidung, dass die JCA zukünftig für die Integration von EISs in J2EE Applikationen verwendet wird, wurde bereits vor dem Erstellungszeitpunkt dieser Diplomarbeit von der Lufthansa Technik AG gefällt:

> „Wir gehen davon aus, dass die JCA sich als Standard für die Integration von EISs durchsetzen wird.“ (Zitat Carsten Ernst, Abteilungsleiter HAM TI LHT [Informationssysteme der Lufthansa Technik AG])

[1] Das Java Native Interface (JNI) ist eine API für den Aufruf von native Code.

3.3 Anforderungen

Die innerbetrieblichen- und Kundenanforderungen sind entscheidend für das Design der Komponente. Werden Anforderungen nicht vollständig erfasst, folgt mit hoher Wahrscheinlichkeit eine aufwändige Anpassung bzw. Modifikation der Komponente. Die Kundenanforderungen wurden in einem Interview mit den zuständigen Teamleiter der Abteilung HAM TI LHT (Informationssysteme der Lufthansa Technik AG) erörtert. Die innerbetrieblichen Anforderungen wurden mit dem Entwicklern von JEF analysiert.

3.3.1 Kundenanforderung

1. Die Hauptforderung der LHT an die zu entwickelnde Komponente ist die Integration von SAP R/3 auf der Ebene von Business Application Programming Interfaces (BAPIs).

2. Zusätzlich soll der Entwurf die Integration weiterer EISs berücksichtigen. Dabei hat der Kunde die Integration von SAP R/3 auf der Ebene von BAPIs eine höhere Priorität zugeordnet als die Integration von FileNet Komponenten.

3. Die Komponente soll auf dem Applikationsserver Weblogic 6.1 von BEA Systems lauffähig sein.

4. Es ist nur synchrone Kommunikation zwischen dem Applikationsserver und den einzubindenen EISs gefordert.

3.3.2 Innerbetriebliche Anforderung

1. Die zu entwickelnde Komponente muss der Definiton einer JEF Komponente entsprechen (JEF Components siehe Abschnitt 4.1.3).

2. Die Komponente soll dem Anwendungsentwickler die Möglichkeit bieten, eine Operation auf einem bstimmten EIS auszuführen, ohne dabei detaillierte Kenntnisse über JCA oder dem jeweiligen EIS besitzen zu müssen. Ausserdem sollte sich dieser möglichst wenig mit Aspekten wie Verbindungs, -Transaktions-, und Sicherheits-Management auseinandern setzen müssen.

Eine Diskussion über die Umsetzbarkeit der Anforderungen erfolgt im Kapitel 5.4. Zunächst sollen aber im folgenden Kapitel die technischen Grundlagen erläutert werden, welche für die zu entwickelnde Komponente notwendig sind.

Kapitel 4

Technische Grundlagen

In diesem Kapitel erfolgt eine Betrachtung der Technologien, die als Grundlage der JEF Connector Komponente dienen. Das ist zunächst das Java Enterprise Framework (JEF), von dem der Zugriff auf die JEF Connector Komponente erfolgen soll. Die JEF Connector Komponente basiert auf der J2EE Connector Architecture (JCA) und wird über die Extensible Markup Language (XML) konfiguriert. Da laut Kundenanforderung BAPIs über die JEF Connector Komponente aufgerufen werden sollen, wird detaillierter auf die objektorientierte Sicht des SAP R/3-Systems eingegangen.

4.1 Java Enterprise Framework

Dieser Abschnitt behandelt das Java Enterprise Framework (JEF) und ist wie folgt strukturiert:

- In Abschnitt 4.1.1 werden die grundlegenden Designprinzipien beschrieben.
- Im nachfolgenden Abschnitt 4.1.2 wird auf die Funktionalität der Basiskom-

ponenten eingegangen.

- Schliesslich wird im Abschnitt 4.1.3 das Komponentenmodell erläutert, mit dem die Funktionalität von JEF erweitert werden kann.

4.1.1 Architektur

JEF basiert auf den Designprinzipien der Multitier Architektur und dem Model-View-Controller Entwurfsmuster.

Multitier Architektur

Applikationen werden in logische und physikalische Schichten (Tiers) unterteilt. Jede dieser Schichten hat ein eigenes Aufgabengebiet, wobei eine Schicht auf die Funktionalität der anderen Schichten zugreifen kann. Durch die Aufteilung einer Applikation in Schichten soll eine bessere Wartbarkeit, Skalierbarkeit und Wiederverwendbarkeit erreicht werden. (vgl. [Anf01, Seite 32])
Die Abbildung 4.1 zeigt den Aufbau der J2EE Multitier Architektur.

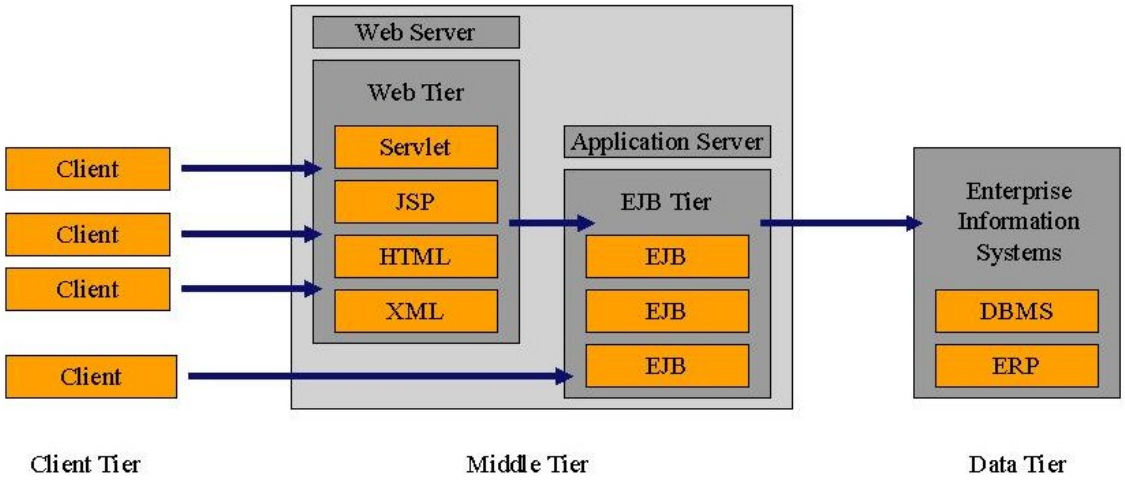

Abbildung 4.1: Multitier Architektur [Tho99, Seite 4]

Folgende Schichten werden in der Abbildung 4.1 identifiziert [Anf01, Seite 32 - 34]:

Client Tier Der Client dient zur Anzeige von Informationen sowie zur Entgegennahme von Benutzeraktion. J2EE unterstützt unterschiedliche Typen von Clienttechnologien, wie z.B. HTML, Applets, XML und auf Java basierende Standalone Clients.

Middle Tier Die Abbildung 4.1 stellt zwar nur eine 3 Tier Architektur dar, aber Sun unterteilt die Middle Tier in weitere Schichten. Servlets und Java Server Pages (JSPs) stellen die Dienste in der Web Tier dar, auf die der Client zugreifen kann. Die Geschäftslogik wird über EJBs abgebildet, die in der Application Tier enthalten sind.

Data Tier In der Data Tier (häufig auch Enterprise Information Systems Tier genannt) erfolgt der Zugriff auf die EISs.

Model-View-Controller Entwurfsmuster

Neben der Multitier Architektur gibt es einen weiteren wichtigen Designbestandteil von JEF, das Model-View-Controller (MVC) Entwurfsmuster.

Zunächst eine Definition des Begriffs Entwurfsmuster:

> „Beschreibungen zusammenarbeitender Objekte und Klassen, die massgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen“. [GHJ96, Seite 4]

Das MVC Entwurfsmuster wurde ursprünglich zur Konstruktion von Benutzerschnittstellen verwendet. Sun verwendet dieses Entwurfsmuster, um Anforderungen an verteilte Applikation zu analysieren. Durch diese weitere Abstraktion entstand die Möglichkeit, eine Applikation in logische Komponenten zu unterteilen, so dass der Architekturentwurf für J2EE erleichtert wurde. (vgl. [Kas01, Seite 21-22])
Die drei Komponenten sind:

- Model
- View
- Controller

Im Folgenden werden diese drei logischen Komponenten detaillierter beschrieben (vgl. [Kas01, Seite 23]):

Model Anwendungsspezifische Daten werden im Model gehalten. Das Model bietet nach aussen Funktionen an, durch welche die Daten manipuliert werden können. Als Beispiel dient die Abbildung einer Datenbanktabelle. Das Model wäre eine oder mehrere Zeile/n dieser Datenbanktabelle. Durch die Funktionen

des Models könnten die Werte dieser Zeile/n manipuliert und ausgelesen werden. Nach der Manipulation würde das Model dem ihm zugeordneten View/s benachrichtigen.

View Ein View zeigt die Daten an, die im Model hinterlegt sind. Jeder View legt fest, wie die Werte präsentiert werden. Die Zeile einer Datenbanktabelle könnte entweder in Form einer Tabelle, als Histogramm oder als Kuchendiagramm dargestellt werden.

Controller Die Benutzeraktionen, die auf einem View ausgeführt wurden, werden an den Controller weitergeleitet. Dieser wandelt die Benutzeraktionen in Aufrufe an das Model um, wodurch die Daten des Models manipuliert werden. Nachdem der Aufruf an das Model erfolgt ist, entscheidet der Controller über den View, in dem die Daten angezeigt werden sollen.

Das Zusammenspiel zwischen den einzelnen Komponenten wird in der Abbildung 4.2 dargestellt.

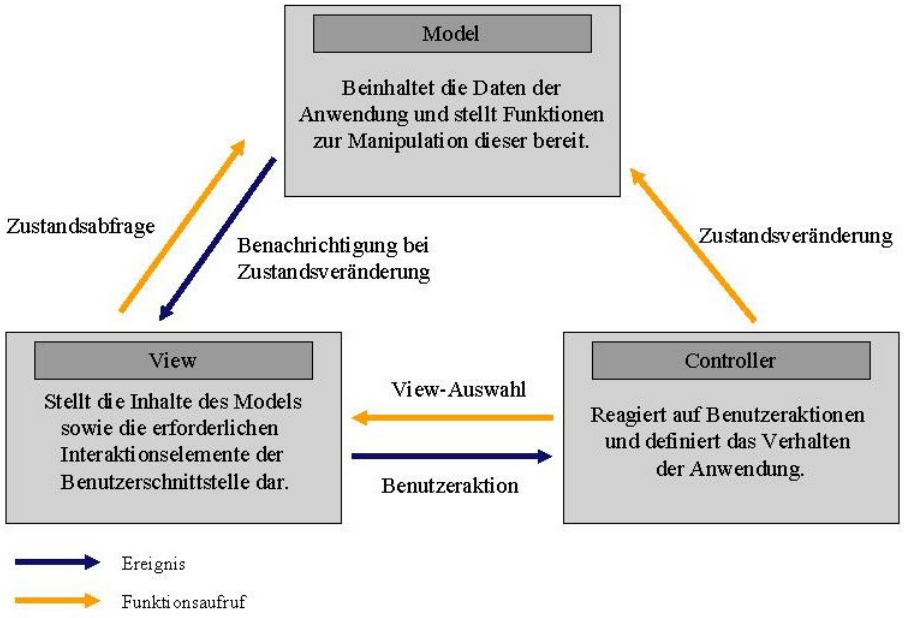

Abbildung 4.2: Model-View-Controller [Kas01, Seite 23]

4.1.2 Basiskomponenten

Die Ausführungen dieses Abschnittes basieren hauptsächlich auf [Anf01]. Die Abbildung 4.3 zeigt die Basiskomponenten von JEF. Zusätzlich verdeutlicht die Abbildung 4.3 den Einsatz der Multitier Architektur und des Model View Controller Entwurfsmusters.

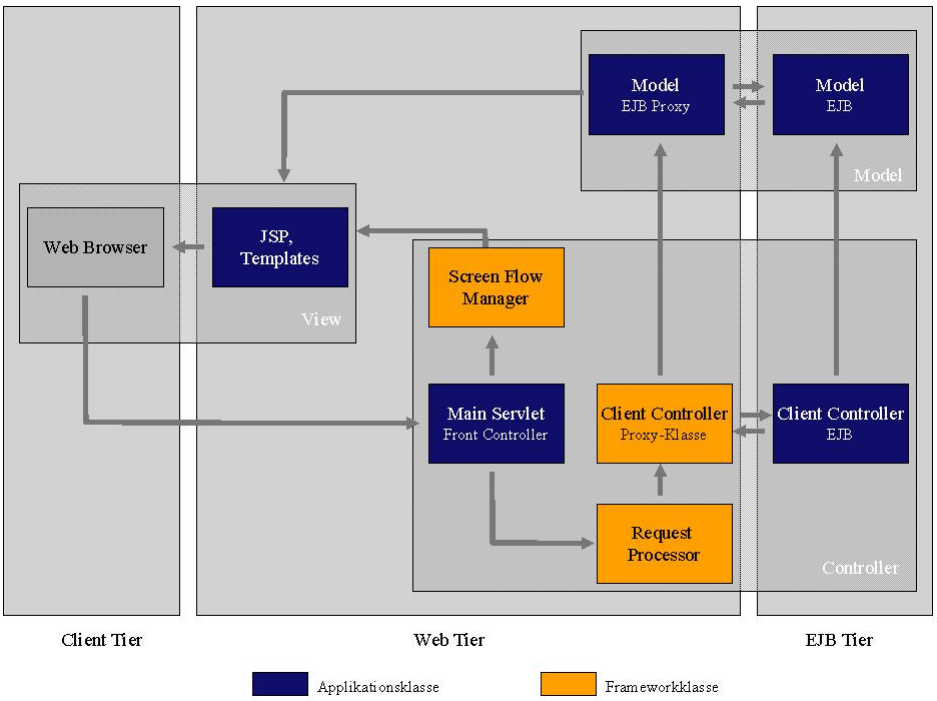

Abbildung 4.3: Übersicht JEF [JEF02, Seite 5]

Model

- **Model Enterprise Java Beans** Die Model Enterprise Java Beans (EJBs) bilden die Geschäftslogik ab. Dabei werden sowohl Entity Beans[1] als auch Session Beans[2] eingesetzt.

- **Model EJB-Proxy** Im Model Bereich einer JEF Applikation befinden sich neben den Enterprise Java Beans und deren Hilfsklassen auch Proxy-Klassen, sogenannte „EJB-Proxies“. Die Proxy-Klassen stellen ein Proxy-Objekt für die Clients einer JEF-Applikation dar und sind somit die Schnittstelle zwischen dem Model und den Views. Durch den Einsatz von EJB-Proxies wird eine

[1] Entity Bean siehe Glossar.
[2] Session Bean siehe Glossar.

Performanceverbesserung erreicht, da der Remote-Zugriff auf die EJBs entfällt.

View

- **Templates** Für die Gewährleistung eines einheitlichen Designs der Oberfläche bilden Templates[3] die zentralen Strukturen der HTML-Oberflächen von JEF Applikation. Dadurch wird es ermöglicht, einen festen Rahmen für die Oberfläche vorzugeben, wobei die Inhalte dennoch dynamisch erzeugt und zusammengesetzt werden.

Controller

- **Front Controller** Die Aufgabe des Front Controllers ist die zentrale Verwaltung der Views aus einem Objekt heraus. Durch den Front Controller wird die Wartung der JEF Applikation optimiert. Beispielsweise kann eine Seite der JEF Applikation umbenannt oder verschoben werden. Ohne den Einsatz einer zentralen Komponente ist hier eventuell ein hoher Wartungsaufwand nötig. Desweiteren werden im Front Controller Aktionen der Benutzer in Anwendungsereignisse umgewandelt und an den Request Processor weitergeleitet.

- **Request Processor** Die Geschäftslogik wird dem MVC Muster folgend durch das Model abgebildet und ausgeführt. Der Controller ist verantwortlich für die Initiierung und Steuerung dieser Geschäftslogik. Der Startpunkt für diese Kontrollaktivitäten ist in der JEF Architektur der Request Processor. Die Aufgabe des Request Processors ist im engeren Sinne die Interpretation der Benutzeraktion und die Ableitung der entsprechenden Geschäftslogik im Model der Applikation. Dazu werden Ereignisobjekte (Events) benutzt, die vom

[3]Ein Template ist eine Seite, bei denen Markups (z.B. HTML) als Platzhalter für die Generierung von dynamsichen Inhalten dienen. (vgl. [ABD01, Seite 240])

Client ausgelöst werden.

- **Sreen Flow Manager** Neben der Analyse und Interpretation der Benutzeraktionen ist die zweite wichtige Aufgabe der Controller Tier die Festlegung der anzuzeigenden Views. Dieses übernimmt der Screen Flow Manager. Aufgerufen wird der Screen Flow Manager vom Servlet, und zwar nachdem der Request Processor die Geschäftslogik durchgeführt hat. Dabei findet der Screen Flow Manager nur bei HTML-Clients Verwendung. Bei Clients auf Basis von Swing Komponenten liegt die Verantwortung für die Flusskontrolle bei den Swing-Clients selbst. JEF bietet hierfür momentan keinen Standard an.

- **Client Controller** Bei dem Client Controller handelt es sich um eine Stateful Session Bean. Diese Enterprise Java Bean ist der zentrale Zugriffspunkt auf Model-Komponenten. Die Client Controller EJB steuert den Lebenszyklus der Model-Komponenten und leitet Ereignisobjekte (Events) an die entsprechenden State Handler. Der State Handler ist eine Klasse, in der die einem Ereignisobjekt zugeordnete Geschäftslogik von einem Anwendungsentwickler implementiert wird. Jedem vom Client ausgelösten Ereinigsobjekt ist eine State Handler Klasse zugeordnet.

4.1.3 Komponenten-Architektur

Unter Verwendung der JEF Komponenten Architektur kann die Funktionalität von JEF dynamisch erweitert werden. In JEF selbst sind schon einige Standardkomponenten enthalten. Ebenso ist es möglich, den Framework Komponenten von Drittanbietern hinzuzufügen. Die Ausführungen dieses Abschnittes basieren auf [JEF02]. Die Komponenten-Architektur besteht aus zwei Sektionen: Komponenten und Extensions. Die Abbildung 4.4 zeigt eine schematische Übersicht der Komponenten Architektur.

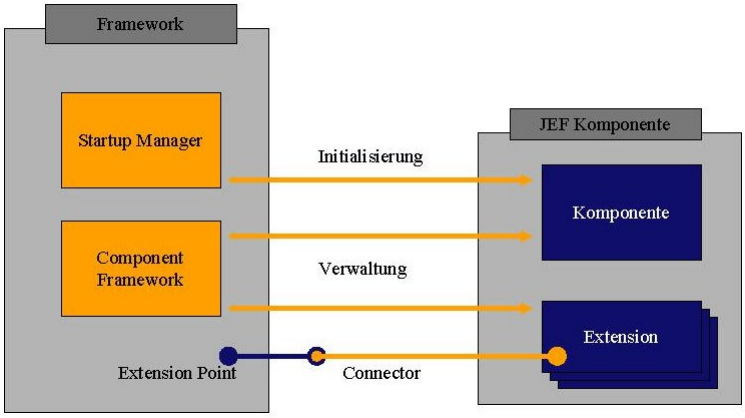

Abbildung 4.4: JEF Komponenten Architektur [JEF02, Seite 18])

Komponenten

Die JEF Komponente ist das Standardkonstrukt, um die Funktionalität von JEF zu erweitern. Eine JEF Komponente besteht dabei aus nur zwei Bestandteilen: Die Komponentenklassen und Konfigurationsdateien. Damit eine Klasse eine JEF Komponente wird, muss diese ein Interface implementieren, welches vom JEF Interface `com.lhsystems.j2ee.components.IComponent` erbt. In der Klasse erfolgt dann die Abbildung der Komponentenfunktionalität. Die Initialisierung der Komponente erfolgt über den Startup Manager. Zu jeder JEF Komponente gehört eine XML Konfigurationsdatei, in der die Attribute für die Konfiguration der Komponente hinterlegt sind. Neben dieser komponentenspezifischen XML-Datei gibt es noch eine anwendungsspezifische XML-Datei. In dieser Datei werden die Komponenten angegeben, die für eine JEF Applikation verwendet werden sollen. Der Anwendungsentwickler kann über eine Component Registry auf JEF Komponenten zugreifen. (vgl. [JEF02, Seite 19])

Extensions

Im Gegensatz zu einer Komponente kann der Anwendungsentwickler den Aufruf einer Extension nicht beeinflussen. Wann eine Extension ausgeführt wird, entscheidet das Framework selbst. Extensions bekommen dadurch die Möglichkeit, den internen Kontrollfluss des Frameworks zu nutzen und diesen teilweise zu beeinflussen. Eine Extension wird über sogenannte Extension Points in das Framework eingebunden. Ab diesen Extension Points wird der Kontrollfluss des Frameworks von der Extension beeinflusst. Extensions werden analog zu den Komponenten über XML-Dateien konfiguriert. Eine JEF Extension kann nur mit einer Komponente zusammen existieren. Die Implementierung einer Komponentenklasse ist immer zwingend notwendig, wenn eine JEF Extension entwickelt werden soll. Der Einsatz von Komponenten, die keine Extensions beinhalten, ist dagegen zulässig. (vgl. [JEF02, Seite 21])

Soweit die Ausführungen zu JEF. Es folgt eine Beschreibung der J2EE Connector Architecture (JCA), auf welcher die JEF Connector Komponente basieren wird.

4.2 J2EE Connector Architecture

Zunächst sollen unterschiedliche Definitionen dargestellt werden.

> 1. Definition:
> „The J2EE Connector Architecture (JCA) provides an easy approach for developers to seamlessy integrate Enterprise Information Systems (EIS) with J2EE platform components.“ [ABD01, Seite 1010)]

Die erste Definition sagt aus, das JCA zum Ziel hat, den Anwendungsentwickler die Integration von EISs in J2EE Applikationen zu erleichtern.

> 2. Definition:
> „An architecture for integration of J2EE servers with EISs. There are two parts to this architecture: an EIS vendor-provided resource adapter and an application server that allows this resource adapter to plug in. This architecture defines a set of contracts, such as transactions, security, connection management, that a resource adapter has to support to plug in to an application server.“ [Jey02, Seite 14]

Die zweite Definition zeigt, wodurch die Erleichterung für den Anwendungsentwickler erreicht wird. Die Erleichterung besteht darin, dass die JCA ein Vertrag definiert, über dem das Verbindungs-, Transaktions- und Sicherheits-Management zwischen dem Applikationsserver und dem EIS geregelt wird. Der Anwendungsentwickler braucht sich nur auf die Umsetzung der Geschäftsprozesse zu konzentrieren.

Die beiden Definitionen werden zusammengefasst, wodurch in dieser Diplomarbeit folgende Definition für JCA gilt:

> Eine Architektur, die dem Entwickler eine Integration zwischen einem J2EE Applikationsserver und EISs dadurch erleichtert, dass ein Vertrag zur Steuerung des Verbindungs-, Transaktions- und Sicherheits-Management definiert wird. Die Implementierung dieses Vertrages erfolgt in einem herstellerspezifischen Resource Adapter, auf den über standardisierte Schnittstellen zugegriffen werden kann.

Die Abbildung 4.5 gibt auf hoher Ebene einen Überblick über den Aufbau von JCA. In der Abbildung 4.5 ist der Resource Adapter ausserhalb des Applikationsservers dargestellt. Dieses liegt darin begründet, dass die Verantwortlichkeiten und Aufgaben zwischen dem Resource Adapter und dem Applikationsserver verdeutlicht werden sollen.

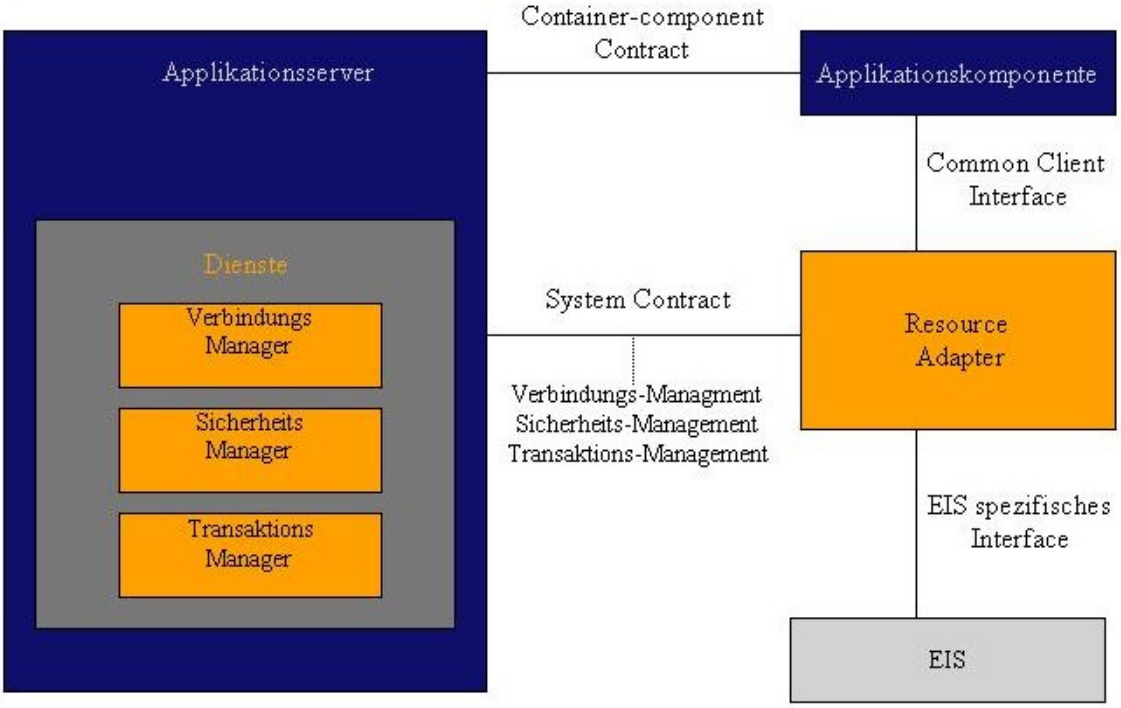

Abbildung 4.5: Überblick JCA (vgl. [Sun03a])

In der Abbildung 4.5 lassen sich drei Bestandteile der JCA identifizieren, auf die in den nachfolgenden Abschnitten detaillierter eingegangen wird:

- Resource Adapter
- System Contract
- Common Client Interface

4.2.1 Resource Adapter

Unter Verwendung des Resource Adapters können Applikationskomponenten[4] auf ein EIS zugreifen. Der Zugriff einer Applikationskomponente auf den Resource Adapter erfolgt über das Common Client Interface (CCI)[5]. Der Aufbau des CCI wird im Abschnitt 4.2.3 erläutert. Für jedes EIS wird ein Resource Adapter in den Adressraum des Applikationsserver eingebunden. Im Resource Adapter erfolgt die Implementierung des System Contract, der das Verbindungs-, Transaktions - und Sicherheits-Management zwischen dem Applikationsserver und dem jeweiligen EIS regelt. Auf den System Contract wird detaillierter im Abschnitt 4.2.2 eingegangen. In der JCA Spezifikation ist nicht definiert, wie der Zugriff innerhalb des Resource Adapters auf das jeweilige EIS erfolgt. Dieses könnte mit den in Abschnitt 2.3.3 vorgestellten Middlewaretypen realisiert werden. (vgl.[SSN01, Seite 31-33])

4.2.2 System Contract

Folgende Dienste sind im System Contract der JCA Spezifikation 1.0 definiert (vgl. [Jey02, Seite 10]):

- Verbindungs-Management
- Transaktions-Management
- Sicherheits-Management

In der JCA Spezifikation 1.5 sind folgende weitere Dienste hinzugefügt worden, die aber im Rahmen dieser Diplomarbeit keine Berücksichtigung finden (vgl. [Jey02, Seite 19 - 20]):

[4] Applikationskomponente siehe Glossar.

[5] Common Client Interface siehe Glossar.

Lifecycle-Management Ermöglicht einem Applikationsserver den Lebenszyklus eines Resource Adapters (Deployment[6], Start/Herunterfahren des Applikationsservers) zu steuern.

Work-Management Erlaubt einem Resource Adapter, sogenannte „Work“ Objekte an den Applikationsserver zu übergeben, die dieser dann ausführt (z.B. Aufruf von Applikationskomponenten).

Transaction-Inflow Einem Resource Adapter wird es ermöglicht, die Interaktionen mit einem Applikationsserver sowie Applikationskomponenten in einem externen Transaktionskontext einzubinden.

Message-Inflow Ermöglicht einem Resource Adapter die asynchrone Kommunikation über das Erzeugen und Versenden von Nachrichten. In der 1.0 Spezifikation wird nur die synchrone Kommunikation unterstützt (vgl. [SSN01, Seite 85]).

Verbindungs-Management

Eine Applikationskomponente ermittelt ein `javax.resource.cci.ConnectionFactory` Objekt aus den JNDI[7] Namensraum eines Applikationsservers. Über die `ConnectionFactory` kann die Applikationskomponente `Connection` Objekte erzeugen lassen. Die `Connection` Objekte stellen eine logische Verbindung zu einem EIS dar. Das Erzeugen der physikalischen Verbindung zum EIS ist die Aufgabe des Resource Adapters. Das Verwalten von physikalischen Verbindungen geschieht über das Connection Pooling, das bei einer Multitier Architektur in den Verantwortungsbereich des Applikationsserver fällt.

[6] Deployment siehe Glossar.

[7] Java Naming and Directory Interface siehe Glossar.

Connection Pooling Eine Applikation, die auf einem Server läuft, bietet Dienste für Clients an. Steigt die Anzahl der Clients, die die Dienste anfordern, so ist es wichtig, die Anfragen so effizient wie möglich zu bedienen, um steigende Wartezeiten zu vermeiden. Eine Möglichkeit, diese Forderung zu erfüllen, ist das Verwenden von bereits existierenden Objekten, die einen hohen Verwendungsgrad von Resourcen haben. Die Erzeugung eines Objektes in Java benötigt viele Resourcen. Zunächst muss Speicher allokiert werden und anschliessend erfolgt die Initialisierung des Objektes. Die Java Virtual Machine (JVM) muss das Objekt verwalten, damit im Rahmen des Garbage Collection Prozesses ein Objekt, welches nicht mehr referenziert wird, zerstört werden kann. Bei Objekten, die eine Netzwerkverbindung zu einem EIS kapseln, kommt noch die Kommunikation zwecks Verbindungsaufbau und Authentifikation[8] hinzu. Das Prinzip des Connection Pooling ist, eine Anzahl von Objekten vor der eigentlichen Anfrage des Clients zu erzeugen und diese in einem Pool zu verwalten. Durch die gemeinsame Verwendung von Objekten wird eine hohe Skalierbarkeit und Performance des Servers erreicht. (vgl. [ABD01, Seite 207])

Architektur Die Ausführungen für diesen Abschnitt basieren hauptsächlich auf (vgl. [SSN01, Seite 206- 212]). Das Verbindungs-Management wird anhand von Sequenzdiagrammen illustriert. Es gelten folgende Bedingungen für das Sequenzdiagramm 4.6:

1. Managed Environment[9].
2. Das Transaktions Management wird vollständig vernachlässigt.
3. Die `ConnectionFactory` wurde bereits aus dem JNDI Namensraum des Applikationsservers ermittelt.

[8] Authentifikation siehe Glossar.

[9] Managed Environment siehe Glossar

4. Aus Übersichtlichkeitsgründen werden die Parameter der Methoden vernachlässigt.

5. Es existieren entsprechende physikalische Verbindungen im Connection Pool.

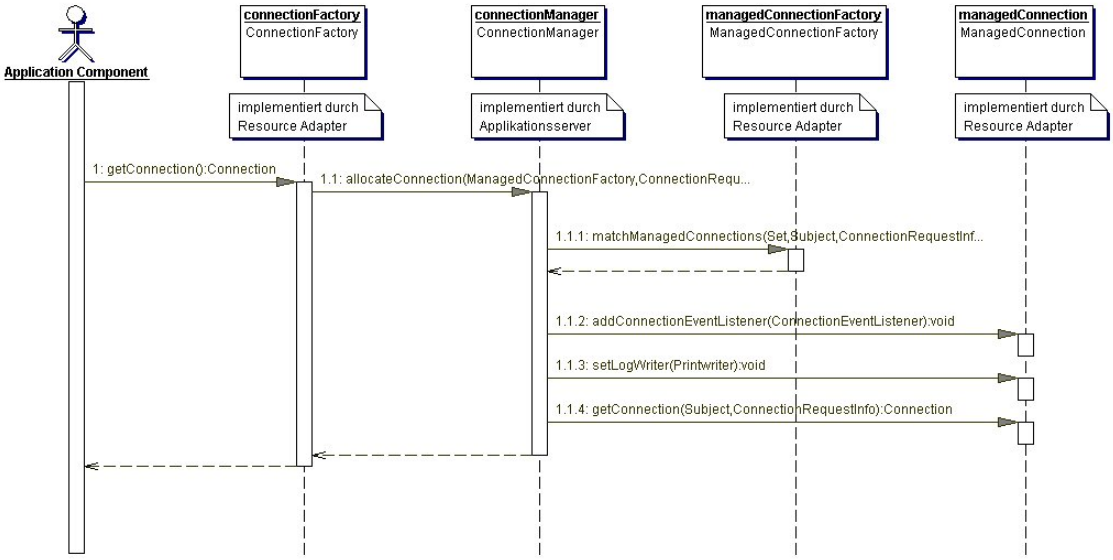

Abbildung 4.6: Verbindungs-Management mit entsprechenden physikalischen Verbindungen im Connection Pool (vgl. [SSN01, Seite 211])

Die Numerierung bezieht sich auf den Botschaftenfluss des Sequenzdiagramm 4.6.

1 Eine Applikationskomponente sendet eine `getConnection()` Botschaft[10] an das `javax.resource.cci.ConnectionFactory` Interface, welches vom Resource Adapter implementiert wird. Die `getConnection()` Methode liefert ein `Connection` Objekt an den Aufrufer zurück, welches eine logische Verbindung zum EIS auf Applikationsebene repräsentiert.

[10]Botschaft siehe Glossar.

1.1 Die `ConnectionFactory` reicht die Anfrage über die `allocateConnection()` Methode an den `ConnectionManager` weiter. Über das Interface `ConnectionManager` wird in einer Managed Environment dem Applikationsserver ermöglicht, dem Resource Adapter verschiedene Dienste anzubieten. Diese beinhalten Transaktions- und Sicherheits-Management, Error Logging/Tracing sowie Connection Pooling. Wie der Hersteller des Applikationsserver diese Dienste implementiert, wird in der JCA nicht spezifiziert.

1.1.1 Der Applikationsserver identifiziert aus dem Connection Pool die physikalischen Verbindungen, die der Anfrage entsprechen könnten. Die Kriterien, nach dem die physikalischen Verbindungen selektiert werden, sind abhängig von der Implementierung des Applikationsservers. Der Resource Adapter implementiert das Interface `ManagedConnectionFactory`. Die selektierte Menge an pyhsikalischen Verbindungen wird über die `matchManagedConnection()` Methode an den Resource Adapter übergeben. Dieser identifiziert nach eingenen Kriterien aus der übergebenen Menge eine passende physikalische Verbindung. Der Rückgabewert der `matchManagedConnection()` Methode ist ein `javax.resource.spi.ManagedConnection` Objekt, welches die physikalische Verbindung zum EIS repräsentiert.

1.1.2 Der Applikationsserver registriert sich als `ConnectionEventListener` bei einem `ManagedConnection` Objekt. Durch die Implementierung des `javax.resource.spi.ConnectionEventListener` Interface kann der Applikationsserver auf Ereignisse des `ManagedConnection` Objekt reagieren, um das Connection Pooling zu steuern, Verbindungen freizugeben oder auf Fehlersituationen zu reagieren.

1.1.3 Optional kann der Applikationsserver eine `setLogWriter()` Botschaft an das `ManagedConnection` Objekt schicken, um diesem Error Logging/Tracing zu er-

möglichen.

1.1.4 Über die `getConnection()` Methode wird eine 1:1 Beziehung zwischen einer logischen Verbindung und einer physikalischen Verbindung hergestellt.

Für das Sequenzdiagramm 4.7 gelten bis auf die Bedingung Nr. 5 die gleichen Bedingungen wie beim Sequenzdiagramm 4.6. Für die Bedingung Nr. 5 gilt, dass **keine** entsprechenden physikalischen Verbindungen im Connection Pool existieren.

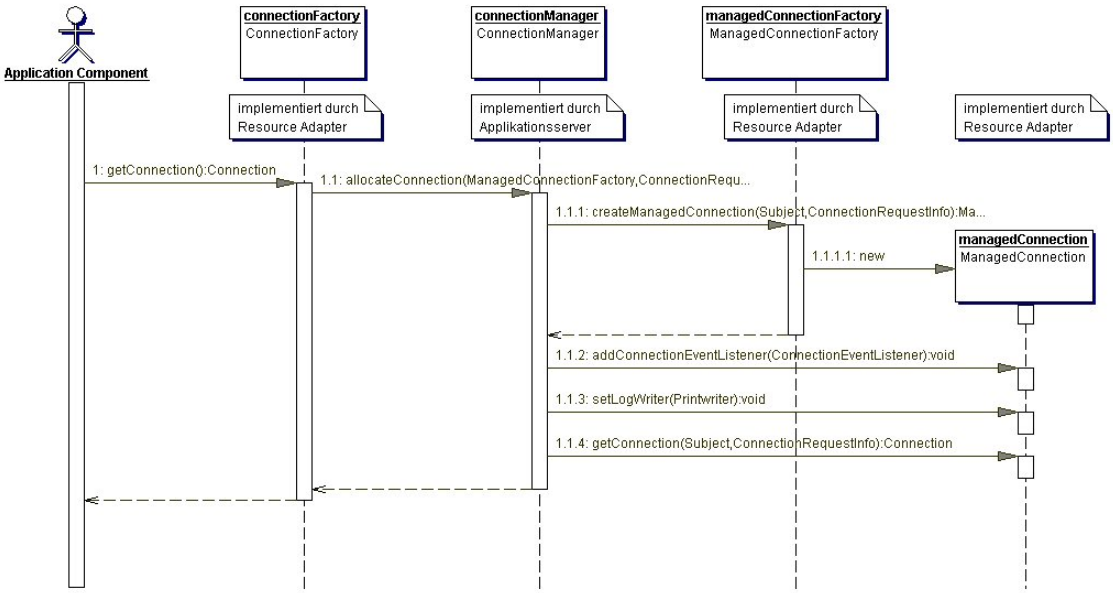

Abbildung 4.7: Verbindungs-Management ohne entsprechende physikalische Verbindungen (vgl. [SSN01, Seite 208])

Bis auf Sequenznummer 1.1.1 ist der Botschaftenfluss derselbe wie beim Sequenzdiagramm 4.6

1.1.1 Findet der `ConnectionManager` kein entsprechendes `ManagedConnection` Ob-

jekt im Connection Pool, wird über die `ManagedConnectionFactory` ein neues `ManagedConnection` Objekt erzeugt und an den Aufrufer zurückgegeben.

Transaktions-Management

Durch das Transaktions-Management wird die Steuerung von Transaktionen zwischen einem Applikationsserver und einem oder mehreren EISs geregelt. Es existieren drei Stufen, die die Transaktionsunterstützung des Resource Adapter bzw. des zugrundeliegenden Resource Manager[11] klassifizieren (vgl. [SSN01, Seite 63]):

- NoTransaction
- XATransaction
- LocalTransaction

Ein Resource Adapter muss eine von diesen drei Stufen bereitstellen. Ein JCA konformer Applikationsserver muss alle drei Transaktionsstufen unterstützen. Die Transaktionsstufe NoTransaction bedeutet, dass der Resource Adapter keine der beiden anderen Stufen unterstützt. Auf die restlichen Transaktionsstufen wird folgend näher eingegangen.

XATransaction Die Verantwortung für die Steuerung von Transaktionen über mehrere Resource Manager hinweg liegt bei dem Transaktions Manager des Applikationsservers. Die J2EE Connector Architecture bezeichnet eine derartige Transaktionssteuerung als XA oder auch globale Transaktion. In dieser Arbeit wird der Begriff globale Transaktion verwendet.

[11] Resource Manager siehe Glossar.

Java Transaction API Das Java Transaction API besteht aus standardisierten Schnittstellen, die einem Transaktions Manager die Steuerung von Parteien, die an verteilten Transaktionen teilnehmen, ermöglicht. Eine Partei ist i.d.R. ein Resource Manager. Damit ein Resource Manager an einer JTA Transaktion teilnehmen kann, muss dieser das `javax.transaction.xa.XAResource` Interface implementieren. Das `XAResource` Interface ist ein Java Mapping des standardisierten XA Interface, das wiederum auf der X/Open CAE Spezifikation basiert.(vgl.[Sun03b]).

Das 2 Phasen Commit Protokol Das `javax.transaction.xa.XAResource` bietet Methoden zur Unterstützung des 2 Phasen Commit Protokolls. Zwei Phasen werden benötigt, damit eine Transaktion über mehrere Resource Manager hinweg abgeschlossen werden kann.

Die erste Phase besteht darin, dass der Transaktions Manager bei jedem an der Transaktion teilnehmenden Resource Manager die `prepare()` Methode aufruft. Durch diese Methode wird jeder beteiligte Resource Manager aufgefordert, ein Commit für die Transaktion vorzubereiten. Der Resource Manager liefert eine Antwort zurück, ob er der Aufforderung auf ein Commit nachkommen kann. Die zweite Phase besteht darin, dass der Transaktions Manager auf die Antwort der Resource Managers entsprechend reagiert. Hat nur ein Resource Manager seine Transaktion nicht erfolgreich durchgeführt, so ruft der Transaktions Manager bei allen Resource Managern die `rollback()` Methode auf. Hat jeder Resource Manager die Transaktion erfolgreich durchgeführt, wird `commit()` aufgerufen. (vgl. [SSN01, Seite 235])

Das Sequenzdiagramm 4.8 verdeutlicht das 2 Phasen Commit Protokoll. Folgende Bedingung gilt für das Sequenzdiagramm:

- Beteiligte Resource Manager haben ihre Transaktionen erfolgreich ausgeführt.

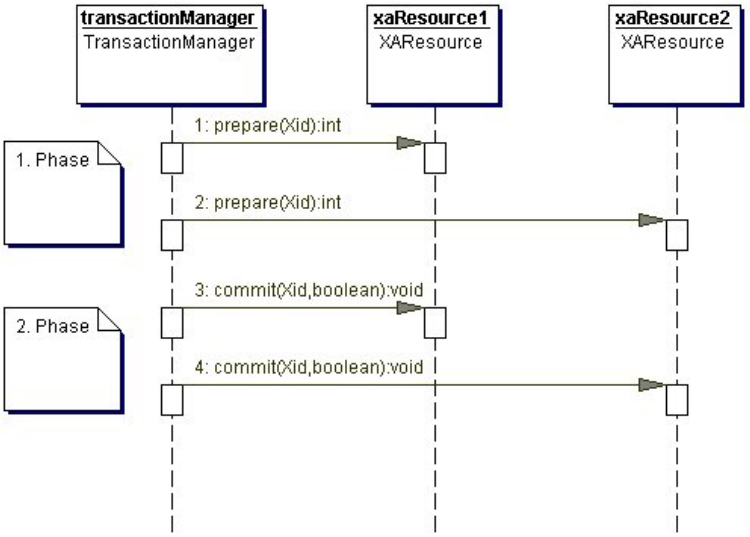

Abbildung 4.8: 2 Phasen Commit Protokoll (vgl. [SSN01, Seite 236])

Transaktions Scenario In den beiden vorangegangenen Abschnitten wurde auf die Grundlagen von globalen Transaktionen eingegangen: Der Java Transaction API und dem 2 Phasen Commit Protokoll. Anhand eines Sequenzdiagrammes soll illustriert werden, wie in der JCA über die Java Transaction API ein Resource Manager in den transaktionellen Kontext eines Applikationsservers eingebunden wird. Das Sequenzdiagramm ist in der Abbildung 4.9 dargestellt und ist eine Erweiterung zum Verbindungs-Management (siehe 4.2.2).

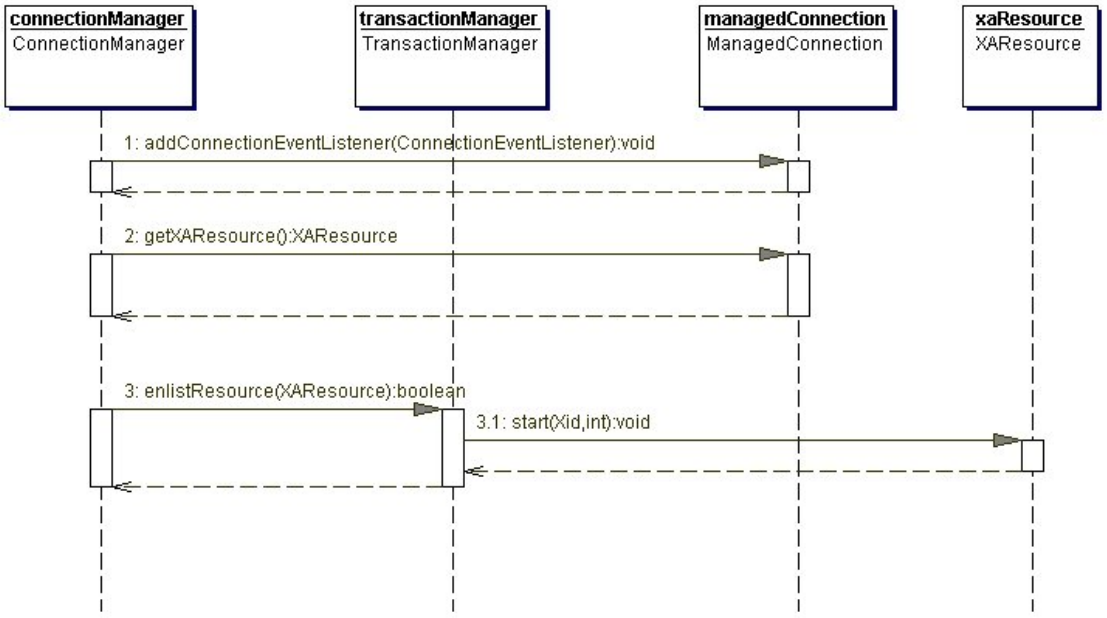

Abbildung 4.9: Beziehung zwischen Resource Manager und Transaktions Manager (vgl. [SSN01, Seite 240])

Die Numerierung bezieht sich auf den Botschaftenfluss des Sequenzdiagramm 4.9

1 Der `ConnectionManager` registriert sich als `ConnectionEventListener` bei einem `ManagedConnection` Objekt.

2 Zwischen einem `ManagedConnection` und einem `XAResource` Objekt besteht eine 1:1 Beziehung. Über die `getXAResource()` Methode wird das `XAResource` Objekt, welcher der `ManagedConnection` zugeordnet ist, zurückgegeben.

3 Der `ConnectionManager` verwendet die `enlistResource()` Methode des Transaktions Manager, um das `XAResource` Objekt in den aktuellen Transaktionskontext einzubinden.

3.1 Der Transaktions Manager ruft die `start(Xid,boolean)` Methode des `XAResource`

Objekts auf. Über ein Objekt, welches das Interface `Xid` implementiert, wird der Transaktionskontext eindeutig identifiziert. Über den Parametertyp `Xid` wird dem `XAResource` Objekt mitgeteilt, welchem Transaktionskontext es zugeordnet wurde. Alle Operationen, die auf der `ManagedConnection` ausgeführt werden, sind somit über das `XAResource` Objekt einem eindeutigen Transaktionskontext zugeordnet.

LocalTransaction Eine Transaktion, die nur einen Resource Manager verwaltet, wird in der JCA als lokale Transaktion bezeichnet. Die Transaktionssteuerung übernimmt eine Applikationskomponente, die auf dem Applikationsserver deployed ist. Es wird bei lokalen Transaktionen also kein Transaktions Manager benötigt. Es besteht eine 1:1 Beziehung zwischer einer ManagedConnection und einem Objekt, welche das Interface `LocalTransaction` implementiert. Über das Interface `javax.resource.spi.LocalTransaction` erfolgt die Steuerung von lokalen Transaktionen (vgl. [SSN01, Seite 230]).

Demarcation-Management Bei den bisherigen Ausführungen ging es hauptsächlich um die Steuerung von Transaktionen. Die Festlegung von Transaktionsgrenzen liegt ausserhalb der JCA Spezifikation. Zur Abrundung des Themas Transaktionen soll dieses trotzdem kurz angesprochen werden. Die J2EE Plattform lässt zwei Möglichkeiten für die Festsetzung von Transaktionsgrenzen zu (vgl. [DYK01, Seite 331]):

Bean-managed Transaction Demarcation Bei Bean-managed Transaction Demarcation werden die Transaktionsgrenzen von einer Applikationskomponente festgelegt. Dieses passiert bei globalen Transaktionen über das `javax.transaction.UserTransaction` Interface. Alle Aufrufe zwischen `begin()` und `commit()` gehören zum Transaktionskontext. Im Falle von lokalen Transaktionen können

die Transaktionsgrenzen auch über das Interface `javax.resource.spi.LocalTransaction` festgelegt werden.

Container-managed Transaction Demarcation Die Transaktionsgrenzen werden in Abhängigkeit von den deklarativen Transaktionsattributen im Deployment Deskriptor der Bean durch den Container[12] gesetzt. Durch diese Transaktionsattribute wird dem Container mitgeteilt, ob für die jeweiligen Methodenaufrufe in einer EJB ein neuer Transaktionskontext erzeugt oder der Transaktionskontext des Clients verwendet werden soll.

Die J2EE Architektur lässt nur flache Transaktionen zu. Geschachtelte Transaktionen werden momentan nicht von der J2EE Architektur unterstützt (vgl. [DYK01, Seite 332]).

Sicherheits-Managment

Ein Unternehmen muss sich vor der Zerstörung oder Verfälschung von geschäftsrelevanten Daten schützen. In der JCA wird durch das Sicherheits-Management der gesicherte Zugriff eines Applikationsservers auf ein EIS geregelt.

Die sicherheitsrelevanten Informationen zur Authentifikation am EIS können entweder von einer Applikationskomponente (Component-Managed Sign-on) oder vom Applikationsserver (Container-Managed Sign-on) geliefert werden. (vgl. [Jey02, Seite 109 und 119])

Component-Managed Sign-on Die Anmeldeinformationen werden von einer Applikationskomponente der `getConnection(javax.resource.cci.ConnectionSpec)` Methode der `javax.resource.cci.ConnectionFactory` übergeben. Das Objekt, in

[12] Container siehe Glossar.

das die Anmeldeinformationen hinterlegt sind, implementiert das `javax.resource.cci.ConnectionSpec` Interface. Wie auf die Anmeldeinformationen zugegriffen wird, ist Resource Adapter spezifisch. Der Vorteil von Component-Managed Sign-on ist, dass die Anmeldeinformation zur Laufzeit der Appikation übergeben werden können. Nachteilig ist, wenn die Anmeldeinformationen statisch im Code hinterlegt sind, diese sich ändern und der Code neu übersetzt werden muss.

Container-Managed Sign-on Der Nachteil der Übersetzung existiert beim Container-Managed Sign-on nicht. Die Anmeldeinformation sind in einem Deployment Deskriptor des Applikationsservers hinterlegt. Diese werden vom Applikationsserver zusammengestellt und in einem `javax.security.auth.Subject` Objekt gekapselt. Die `Subject` Klasse ist in der Java Authentication and Authorization Service (JAAS) Spezifikation definiert. JCA empfiehlt, dass der Applikationsserver JAAS unterstützt. Das `Subject` Objekt wird der `createManagedConnection(Subject sub,ConnectionRequestInfo crInfo)` Methode der `ManagedConnectionFactory` übergeben. In dieser Methode werden die Anmeldeinformationen ausgelesen und eine physikalische Verbindung zum EIS hergestellt.

Es muss ferner sichergestellt werden, dass eine Instanz (Applikationskomponente oder Container), die sich durch die Verwendung entsprechender Anmeldeinformationen am EIS authentifizeren möchte, tatsächlich auch über die Identität verfügt, die mit den benutzten Anmeldeinformation assoziiert ist. In der JCA Spezifikation werden hierfür zwei Mechanismen empfohlen:

BasicPassword Authentifikation erfolgt über eine Benutzername/Passwort-Kombination.

Kerbv5 Die Authentifikation erfolgt über das Kerberos Version 5 Verfahren[13].

JCA konforme Applikationsserver sollten diese weit verbreiteten Authentifikationsmechanismen unterstützen. Die JCA Spezifikation fordert nicht, dass bestimmte Authentifikationsmechnismen vorhanden sein müssen. (vgl [SSN01, Seite 76])

4.2.3 Common Client Interface

Die JCA bietet durch das Common Client Interface (CCI) Applikationskomponenten und EIS Frameworks eine standardisierte API für den Zugriff auf heterogene EISs an. Im Gegensatz zu dem System Contract, der eingehalten werden muss, steht es dem Hersteller eines Resource Adpaters frei, seine eigene Client API zu erstellen. Das CCI dient weiterhin dazu, bestehende APIs, wie z.B. JDBC, zu erweitern und nicht zu ersetzen. (vgl. [SSN01, Seite 111])

[13]Kerberos siehe Glossar.

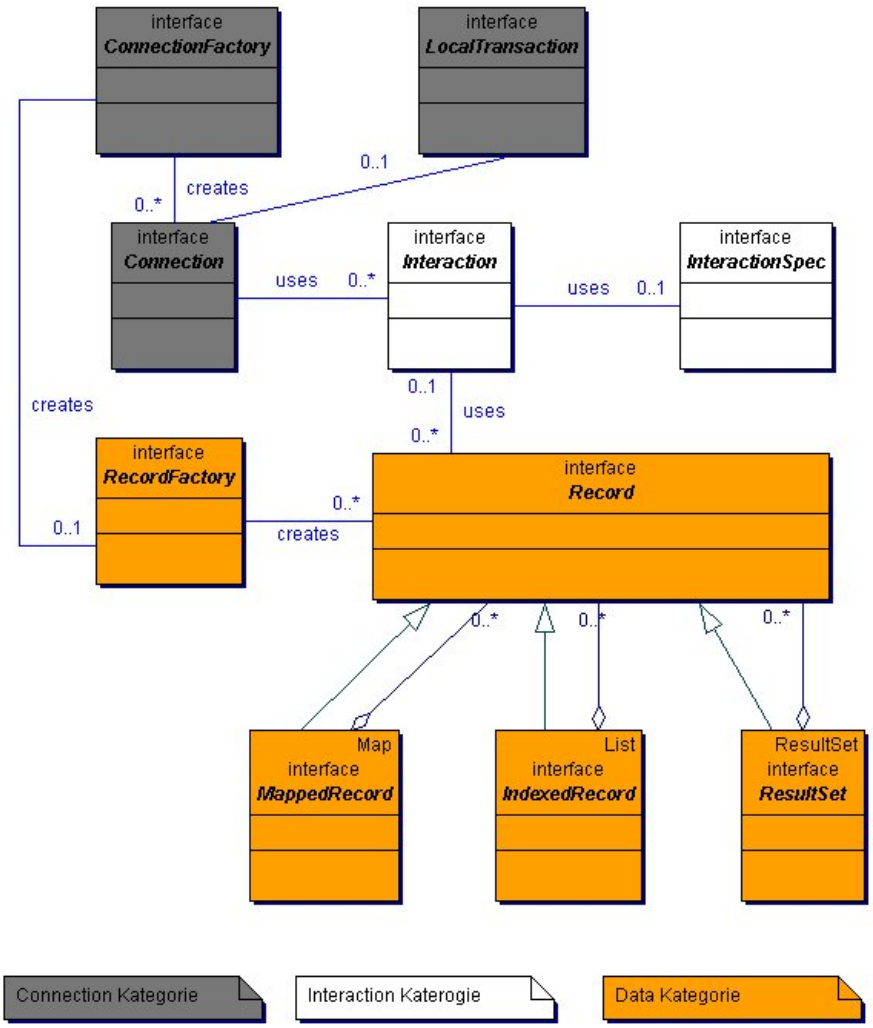

Abbildung 4.10: Package javax.resource.cci.* (vgl.[SSN01, Seite 112])

Das CCI besteht aus einer Menge von Schnittstellen und Klassen. Diese ermöglichen einer Applikationskomponente typische EIS Operationen, wie z.B. das Herstellen von Verbindungen, das Ausführen von Operationen und der Zugriff auf Daten. Sämtliche Schnittstellen und Klassen des CCI befinden sich im Package `javax.resource.cci.*` und lassen sich in 4 Kategorien unterteilen (vgl. [SSN01, Seite 118-126]):

Connection Repräsentiert Verbindungen zum EIS auf Applikationsebene.

Interaction Ermöglicht einer Applikationskomponente mit dem EIS zu interagieren.

Data Dient zur Verarbeitung von Daten.

Metadata Enthält Metadaten über die Implementierung des Resource Adapter.

Im folgenden wird detaillierter auf die wichtigsten Schnittstellen der vier Kategorien eingegangen:

Connection

- **ConnectionFactory** Auf Applikationsebene wird über das `ConnectionFactory` Interface eine Verbindung zum EIS hergestellt. Die logische Verbindung wird durch ein `Connection` Objekt repräsentiert. Dieses Objekt kann über die `public Connection getConnection()` Methoden des `ConnectionFactory` Interface erzeugt werden. Die `ConnectionFactory` wird über einen JNDI `lookup()` aus dem Namensraum des Applikationsservers gelöst.

- **ConnectionSpec** Über das `ConnectionSpec` Interface werden von einer Applikationskomponente sicherheitsrelevante Informationen zur Authentifikation am EIS geliefert. Der Resource Adapter, der das `ConnectionSpec` Interface implementiert, ist für das Auslesen dieser Informationen verantwortlich.

- **Connection** Durch das `Connection` Interface wird eine Verbindung auf Applikationsebene repräsentiert. Über die `public Interaction createInteraction()` Methode kann eine Applikationskomponente mit dem EIS interagieren. Es sind mehrere Interaktionen über ein `Connection` Objekt möglich. Sollen die

Grenzen einer lokalen Transaktion über eine Applikationskomponente festgelegt werden (bean-managed transaction demarcation), so geschieht dieses über das `LocalTransaction` Interface. Ein `LocalTransaction` Objekt kann über die `getLocalTransaction()` Methode eines `Connection` Objekts erzeugt werden.

- **LocalTransaction** Das `LocalTransaction` Interface ermöglicht einer Applikationskomponente oder einem Container die Steuerung einer lokalen Transaktion. Eine `LocalTransaction` ist 1:1 mit einem `Connection` Objekt assoziiert.

Interaction

- **Interaction** Im `Interaction` Interface werden Methoden bereitgestellt, die es einer Applikationskomponente ermöglichen, mit dem EIS zu interagieren. Die `execute()` Methoden sind die wichtigsten im `Interaction` Interface. Es existieren zwei unterschiedliche Signaturen der `execute` Methoden. Die erste Methode enthält drei Parameter: ein `InteractionSpec` Objekt, ein Eingaberecord und ein Ausgaberecord. Dieses `InteractionSpec` Objekt spezifiziert, welche Operation des EIS ausgeführt werden soll. Der Eingaberecord enthält Eingabeparameter für die aufzurufende Operation, im Ausgaberecord wird die Ergebnismenge des Operationsaufrufs hinterlegt. Bei der zweiten Methode ist der Ausgaberecord nicht als Parameter, sondern als Rückgabewert definiert. Ein `Interaction` Objekt kann über die `createInteraction()` Methode des `Connection` Interface erzeugt werden. Dieses `InteractionSpec` Objekt ist mit dem `Connection` Objekt assoziiert. Über die `getConnection()` Methode kann ermittelt werden, mit welcher `Connection` das `Interaction` Objekt assoziiert ist. Wenn eine Applikationskomponente die Interaktionen mit einem EIS beendet hat, so sollte die `close()` Methode aufgerufen werden. Über diese Methode kann der Resource Adapter die mit den Operationsaufrufen allokierten

Resourcen des EIS wieder freigeben.

- **InteractionSpec** Über das Interface `InteractionSpec` wird definiert, welche Operation im EIS aufgerufen werden soll. Das Standard `InteractionSpec` Interface enthält keine Methoden, sondern nur Attibute, welche den Interactionmodus definieren. Die Interaction kann entweder als synchron sendend, synchron sendend und empfangend oder nur als synchron empfangend durchgeführt werden. Bei der Implementierung eines Resource Adapters würde das `InteractionSpec` Interface um Methoden erweitert werden, z.B. `setFunctionName(String functionName)`, um die aufzurufende Operation definieren zu können.

Data

- **Record** Das `Record` Interface ist die Java Repräsentation für strukturierte Daten, die zwischen einer Applikationskomponente und einem EIS ausgetauscht werden. `Record` Objekte werden als Parameter in der `execute()` Methode des `Interaction` Interface übergeben. Weitere `Record` Interfaces des CCI, die alle von `Record` erben, sind `MappedRecord`, `IndexedRecord` und `ResultSet`.

- **IndexedRecord** Interface] Ein `IndexedRecord` ist eine sortierte Liste, auf deren Elemente über ein Index zugegriffen wird. Das Interface erbt von dem Interfaces `java.util.List` und `Record`.

- **MappedRecord** Das `MappedRecord` ist ein `Record`, auf dessen Elemente über ein Key-Value-Mapping zugegriffen wird. Das Interface erbt von `Record` und `java.util.Map`.

- **ResultSet** Das `ResultSet` Interface erbt von `Record` und dem JDBC Interface `java.sql.ResultSet`. Das `ResultSet` stellt eine JDBC Tabelle dar.

- **RecordFactory** Über die `RecordFactory` können Record Objekte erzeugt werden.

Metadata

- **ConnectionMetaData** bietet Metadaten über eine existierende Verbindung zum EIS. Über ein `Connection` Objekt kann ein `ConnectionMetaData` Objekt erzeugt werden, über das Informationen zur `Connection` abgefragt werden können.
- **ResourceAdapterMetaData** enthält Metadaten über die Implementierung des Resource Adapters. Die Erzeugung eines `ResourceAdapterMetaDate` Objekts erfolgt über die `ConnectionFactory`.

4.2.4 Das M*N Problem

Um von einem Applikationsserver auf unterschiedliche EIS zuzugreifen, ist JCA als Standard nicht zwingend. Es kann daher angenommen werden, dass der Hersteller A einen Applikationsserver entwickelt. Für den Hersteller A ist es viel Entwicklungsaufwand, die Integration jedes einzelnen EIS Typs zu berücksichtigen. Auf der anderen Seite gibt es den Hersteller B, der ein EIS anbietet. Pro Applikationsserver muss der Hersteller B einen Connector entwickeln, damit auf sein EIS zugegriffen werden kann. Die Lösungen für die EIS Integration sind von Hersteller zu Hersteller unterschiedlich. Dieser Sachverhalt lässt sich in einer Formel darstellen. Es existieren m Hersteller von Applikationsservern und n Hersteller von EISs. Der Aufwand, alle Applikationsserver mit allen EISs zu integrieren, beträgt ohne JCA m*n. Durch JCA wird dieser Aufwand erheblich verringert. Der Hersteller B muss den System Contract nur einmalig erfüllen. Hersteller A braucht nur einen Resource Adapter,

der den System Contract der JCA implementiert. Dieser eine Resource Adapter kann in unterschiedliche Applikationsserver integriert werden. Umgekehrt kann ein Applikationsserver unterschiedliche Resource Adapter einbinden. Der Gesamtaufwand für die Integration von EISs reduziert sich also auf m + n. (vgl. [Jey02, Seite 16])

In dem Abschnitt 4.2 sollte ein Grundverständnis über die JCA vermittelt werden. Da laut Kundenanforderungen SAP R/3 auf BAPI Ebene über die JEF Connector Komponente integriert werden soll (Kundenanforderungen siehe 3.3.1), wird im nächsten Abschnitt näher auf die objektorientierte Sicht des SAP R/3-Systems eingegangen.

4.3 SAP R/3

BAPIs sind Bestandteile des SAP R/3 Business Frameworks, auf das folgend detaillierter eingegangen wird. Die Ausführungen dieses Abschnittes basieren auf „Allgemeine Einführung in die BAPIs“ aus [SAP03]. Zusätzlich wird auf den Java Connector (JCo) eingegangen.

4.3.1 Business Framework

Durch das SAP R/3 Business Framework wird eine komponentenbasierte Architektur angeboten, die eine Interaktion und Integration zwischen Softwarekomponenten von SAP R/3 und anderen Herstellern ermöglicht. Es sieht eine Strukturierung der R/3-Funktionalität in Business Komponenten vor. Dadurch wird es Kunden und Partnern ermöglicht, eigene Komponenten an das R/3-System anzubinden. Die Nutzung von objektorientierter Technologie und Objektmodellen trägt wesentlich zur Komplexitätsreduzierung des Gesamtsystems bei. Die Grundlage für die Kom-

ponentenentwicklung bei SAP R/3 ist die Architektur des Business Frameworks. Die wesentlichen Bestandteile sind:

- Business Komponenten
- Business Objekttypen
- Business Application Programming Interfaces (BAPIs)
- Application Link Enabling (ALE)

Business Komponenten

Business Komponenten sind wartbare Komponenten, die sich aus mehreren Business Objekttypen zusammensetzen. Als Beispiel dient die Business Komponente *Human Resources* (Personalwesen), die die Business Objekttypen *Employee* (Mitarbeiter) und *Applicant* (Bewerber) enthält. Geschäftsprozesse können in einer Business Komponente implementiert werden oder sich über mehrere Business Komponenten erstrecken.

Business Objekttypen

Der Begriff des Business Objekttyps ist analog zu dem Begriff der Klasse aus dem objektorientierten Paradigma. Betriebswirtschaftliche Abläufe werden durch Business Objekttypen abgebildet. Die Geschäftsprozesse und die dazugehörigen Daten werden gekapselt. Dadurch bleibt die Implementierung der Geschäftsprozesse und die innere Struktur des Business Objekttyps verborgen. Die Kapselung erfolgt über 4 Schichten:

Zugriffsschicht Die äusserste Schicht ist die Zugriffsschicht. In dieser Schicht sind

die Kommunikationsdienste definiert, über die der Zugriff auf einen Business Objekttyp erfolgt.

Schnittstellenschicht In der Schnittstellenschicht sind die Schnittstellen definiert, über die auf einen Business Objekttyp zugegriffen werden kann.

Integritätsschicht In der Integritätsschicht wird durch die Definition von Geschäftsregeln und die Beschränkung von Werten die Konsistenz eines Business Objekttyps gewährleistet.

Implementierungsschicht Dieses ist die innerste Schicht, in der die Implementierung der abzubildenden Geschäftsprozesse stattfindet.

Business Objekttypen werden durch folgende Eigenschaften definiert:

Grunddaten Die Grunddaten sind Metadaten über den Objekttyp. Dieses sind exemplarisch eine Bezeichnung, eine Kurzbeschreibung, Erstellungsdatum usw..

Schlüsselfelder Ein Objekt eines Business Objekttyps wird über Schlüsselfelder ermittelt. Erfolgt der Aufruf einer Objektmethode, so werden die Werte für die Schlüsselfelder als Importparameter der Methode übergeben.

Methoden Über Methoden wird das Verhalten eines Objekttyps und seinem Objekten definiert.

Attribute Über Attribute wird der Zustand eines Objekttyps und seiner Objekte definiert.

Ereignisse Über ein Ereignis signalisiert ein Objekt seiner Umgebung, dass eine Zustandsänderung eingetreten ist.

Business Application Programming Interfaces

Business Application Programming Interfaces (BAPIs) sind standardisierte und stabile Methoden, über die ein Zugriff auf Business Objekttypen ermöglicht wird. Durch die BAPIs und die Business Objekttypen ist eine objektorientierte Sicht auf die betriebswirtschaftliche Funktionalität von SAP R/3 gegeben. BAPIs sind seit dem SAP R/3 Release 3.1 verfügbar und die Anzahl von BAPIs nimmt stetig zu. Das Business Object Repository (BOR) ist das zentrale Werkzeug, über das Business Objekttyen und BAPIs verwaltet werden. BAPIs können synchron und asynchron aufgerufen werden. Der synchrone Aufruf erfolgt über RPC fähige Funktionsbausteine. Die Zuordnung eines Funktionsbausteins zu einem BAPI erfolgt über das BOR. Der asynchrone Zugriff erfolgt über Application Link Enabling (ALE) (ALE siehe 4.3.1).

Es folgt eine Auflistung von BAPI Merkmalen, die für das Grundverständnis von BAPIs relevant sind.

Namenskonvention Ein BAPI wird über die Angabe des Namens vom Business Objekttyps, gefolgt vom Namen des BAPIs, vollständig identifiziert. Der Name eines BAPIs sollte dabei in englischer Sprache sein. Zusätzlich sollte der BAPI Name so gewählt werden, dass er seiner Funktion entspricht. Analog zur Programmiersprache Java sind die Namensteile durch einen Punkt getrennt. Als Beispiel dient hier der Standard BAPI getDetail() zu dem Business Objekttyp PurchaseOrder: `PurchaseOrder.getDetail()`.

standardisierte BAPIs und Parameter Es existieren in SAP R/3 einige standardisierte BAPIs. Über diese BAPIs wird eine häufig verwendete Gundfunktionalität der Business Objekttypen erreicht. Ein Beispiel für eine standar-

disierte BAPI ist die getDetail() BAPI, welche Detailinformationen zu einen Objekt an den Aufrufer zurückliefert.

Datenbankkonsistenz Alle BAPIs, welche Daten manipulieren, hinterlassen immer einen konsistenten Datenbankzustand. Datenbankänderungen erfolgen entweder vollständig oder überhaupt nicht.

Keine Dialogorientierung An die aufrufende Applikation werden von dem BAPIs
kein Bildschirmdialoge zurückgegeben.

Berechtigung Für Applikationen, die BAPIs ausführen, muss der Aufrufer eine bestimmte Berechtigung haben.

Soweit zu der Definition und den Merkmalen von BAPIs. Folgende Punkte zeigen die Vorteile des Einsatzes von BAPIs:

Betriebswirtschaftlicher Standard Business-Objekttypen und deren BAPIs sind der Standard für den betriebswirtschaftlichen Inhalt des SAP R/3-Systems. Durch sie wird die Integration von SAP R/3 und anderer Softwarekomponenten auf betriebswirtschaftlicher Ebene ermöglicht.

Konformität mit Standards BAPIs wurden in Zusammenarbeit von Kunden und Partnern der SAP sowie durch führende Namensorganisationen entwickelt. Das Ziel dieser Zusammenarbeit ist die Entwicklung eines Kommunikationsstandards zwischen betriebswirtschaftlichen Systemen.

Stabilität und Abwärtskompatibilität Nach der Einführung und Freigabe eines BAPI durch SAP bleiben seine Schnittstellendefinition und Parameter langfristig stabil. Dadurch wird sichergestellt, dass Applikationen von Manipulationen am SAP R/3-System nicht beeinflusst werden.

Objektorientierung BAPIs bieten Zugriff auf die Prozesse und Daten der Business Objekttypen. Die Implementierung der Prozesse sind in den Business Objekttypen gekapselt. Dieses entspricht den Prinzipien des objektorientierten Paradigma.

Offenheit Der Zugriff auf BAPIs kann von Entwicklungsplattformen erfolgen, die RPC unterstützen.

Application Link Enabling

Application Link Enabling (ALE) ist Middleware, über die Geschäftsprozesse zwischen R/3 Systemen sowie zwischen R/3 und Fremdsystemen integriet werden. Die zu einem ALE-Verbund gehörenden Anwendungssysteme sind lose gekoppelt. Der Austausch der Daten erfolgt asynchron. Dabei wird sichergestellt, dass die Daten auch im Empfängersystem ankommen, wenn dieses zum Sendezeitpunkt nicht erreichbar ist. Nur der lesende Zugriff auf Daten erfolgt über synchrone Verbindungen.

4.3.2 Java Connector

Sämtliche Ausführungen in diesem Abschnitt basieren auf [Sch02]. Der Java Connector ist eine von SAP entwickelte Middleware, um von Java Applikationen auf SAP R/3 zugreifen zu können. Jedoch ist der Java Connector (Jco) **nicht** JCA konform. Es wird der in der JCA Spezifikation beschriebene System Contract nicht erfüllt. Der Java Connector basiert auf einer RPC-API. Hinter dieser RPC-API steckt eine C Bibliothek. Der Aufruf dieser RPC-API über den Java Connector bleibt dem Aufrufer von JCo verborgen. Der JCo unterstützt sowohl inbound (Java ruft ABAP[14]) als auch outbound (ABAP ruft Java) communications. Über JCo können RPC fähige

[14] Advanced Business Application Programming (ABAP) ist die Programmiersprache des SAP R/3 Systems.

Funktionsbausteine und BAPIs aufgerufen werden. Es werden zwei Verbindungsarten unterstützt:

direkte Verbindung Der Aufrufer baut eine Verbindung auf und schliesst diese bei Bedarf.

Connection Pool Der Aufrufer wird aus einem Connection Pool bedient.

SAP R/3 unterstützt nicht das 2 Phasen Protokoll, somit können über den JCo nur lokale Transaktionen durchgeführt werden (vgl.[SSN01, Seite 266]).

Soweit zu der objektorientierten Sicht des SAP R/3-Systems. Im folgenden Abschnitt wird auf die Extensible Markup Language (XML) eingegangen, über die die JEF Connector Komponente konfiguriert wird.

4.4 Extensible Markup Language

Die Extensible Markup Language (XML) ist ein Standard, über den strukturierte Dokumente beschrieben und erstellt werden können. XML wurde von einer Arbeitsgruppe entwickelt, die 1996 unter der Schirmherrschaft des World Wide Web Consortium (W3C) gegründet wurde. Das W3C definiert die Extensible Markup Language wie folgt:

> „Die Extensible Markup Language, abgekürzt XML, beschreibt eine Klasse von Datenobjekten, genannt XML-Dokumente, und beschreibt teilweise das Verhalten von Computer-Programmen, die solche Dokumente verarbeiten. XML ist ein Anwendungsprofil (application profile) oder eine eingeschränkte Form von SGML, der Standard Generalized Markup Language [ISO 8879]. “ [TBPSM03])

4.4.1 Aufbau eines Dokuments

Ein XML Dokument besteht aus zwei Teilen: Dem Kopf, welcher Informationen für XML Parser und XML Applikationen enthält und dem eigentlichen Inhalt des Dokuments. Der Inhalt des Dokumentes ist baumartig strukturiert. Die Wurzel dieses Baumes wird als „Root Element“ bezeichnet. Das Root Element ist das oberste Element in einem XML Dokument. Ein Element besteht aus einem öffnenden und einem schliessendem Tag. Der Inhalt zwischen dem öffnenden und schliessendem Tag kann aus einer Zeichenkette, weiteren Elementen oder einer Mischung aus beidem bestehen. Durch die Verschachtelung von Elementen erhält ein XML Dokument seine baumartige Struktur. Enthält ein Element keinen Inhalt, wird es auch als leeres Element ausgewiesen. Elemente können Attribute besitzen. Ein Attribut besteht aus einem Namen und einem Wert. (vgl. [McL01, Seite 13]).
Ein XML Dokument ist wohlgeformt, wenn es folgende Eigenschaften besitzt (vgl. [TBPSM03]):

- Ein Dokument hat ein oder mehrere Elemente.
- Elemente bestehen aus einem öffnenden und einem schliessenden Tag.
- Die Elemente des Dokuments sind korrekt geschachtelt.
- Es existiert genau ein Root Element, welches alle weiteren Elemente umschliesst.

4.4.2 Schema

XML Schema ist der Nachfolger der Document Type Definition (DTD). Über ein XML Schema werden Grammatiken definiert, welche die Struktur eines XML Do-

kumentes beschreiben. Etwas konkreter ausgedrückt wird folgendes über ein XML Schema festgelegt (vgl. [McL01, Seite 21]):

- Welche Elemente vorkommen dürfen,
- aus welchen Elementen ein Element besteht sowie
- die Datentypen der Elemente und Attribute.

XML Schema hat gegenüber der DTD folgende Vorteile (vgl. [Cos03][Folie 6-7]):

- Die Syntax entspricht der von XML.
- Es wird eine hohe Anzahl von Datentypen angeboten.
- Die Wertebereiche von Datentypen lassen sich begrenzen.

4.4.3 Java und XML

Es existieren zwei weit verbreitete APIs, um über Java ein XML Dokument manipulieren zu können (vgl. [McL01, Seite 38-48] und [McL01, Seite 103]):

SAX Simple API for XML (SAX) ist eine ereignisgesteuerte API. Das XML Dokument wird sequentiell gelesen und abhängig vom Element werden bestimmte Methoden aufgerufen. Über diese Methoden findet dann die Manipulation des XML Dokuments statt.

DOM Beim Document Object Model (DOM) wird das gesamte XML Dokument in den Arbeitsspeicher geladen und in einer Baumstruktur abgelegt. Diese Baumstruktur kann dann über die DOM-API ausgelesen und manipuliert werden.

Häufig werden diese Schnittstellen mit Parsern verwechselt. Parser dienen dazu, ein XML Dokument einzulesen, auf Wohlgeformtheit zu überprüfen, eventuell zu validieren und die Schnittstellen von SAX und/oder DOM zu implementieren. Validieren bedeutet hierbei, die Struktur eines XML Dokumentes gegen die Grammatik einer DTD oder eines XML Schemas zu überprüfen. Mittlerweise existiert eine hohe Anzahl von Parsern, die beide APIs unterstützen und gegen eine DTD oder ein XML Schema validieren können, z.B. Xerces-J von Apache.

Soweit zu den technischen Grundlagen. Es wurde auf JEF eingegangen, das zugrundeliegende Framework für die JEF Connector Komponente. Die JEF Connector Komponente basiert auf der JCA und wird im Rahmen der JEF Komponenten Architektur über XML konfiguriert. Über die JEF Connector Komponente soll eine Integration von SAP R/3 auf BAPI Ebene erfolgen. Im nächsten Kapitel erfolgt eine Erläuterung des Entwurfes, in dem die in diesem Kapitel erläuterten technischen Grundlagen Verwendung finden.

Kapitel 5

Entwurf

In diesem Kapitel wird der Entwurf der JEF Connector Komponente erläutert. Dabei wird schwerpunktmässig auf die Integration und dem Aufruf von BAPIs eingegangen. Schliesslich wird diskutiert, ob durch den Entwurf alle Forderungen der Anforderungsanalyse erfüllt werden konnten.

5.1 Übersicht

Eine innerbetriebliche Anforderung an die JEF Connector Komponente ist, dass diese dem Anwendungsentwickler Schnittstellen zur Verfügung stellt, die einen „einfachen“ Operationsaufruf an ein EIS ermöglicht. Unter „einfach“ wird in diesem Kontext verstanden, dass der Anwendungsentwickler wenig Kenntnisse von JCA und dem zu integrierenden EIS besitzen muss. Für den Aufruf von EIS Operationen bietet das Common Client Interface komfortable Schnittstellen an. Die JCA Spezifikation sieht vor, dass der Hersteller des Resource Adapters die Schnittstellen des CCI so erweitert, dass ein Aufruf einer Operation zum jeweiligen EIS ermöglicht wird.

Ein Ansatz für die JEF Connector Komponente wäre, dass CCI des jeweiligen Resource Adapters für den Aufruf von Operationen anzubieten. Dieser Ansatz hat jedoch zwei entscheidende Nachteile:

1. Es würde eine starke Kopplung zu den Schnittstellen des jeweiligen Resource Adapters existieren, welches das CCI spezifisch zum EIS erweitert. Würde in einer bereits existierenden JEF Applikation ein anderer Resource Adapter zum Einsatz kommen, müssten alle Aufrufe über die Schittstellen des alten Resource Adapters an dem neuen angepasst werden.

2. Der Anwendungsentwickler müsste sich mit dem CCI des jeweiligen Resource Adapters auseinandersetzen.

Nach diesem Ansatz ist die Forderung nach einer „einfachen“ Schnittstelle also nicht erfüllbar. Stattdessen werden die aufzurufenden Operationen und die Resource Adapter abhängigen Informationen in einer XML-Datei hinterlegt. Bei der Initialisierung der JEF Connector Komponente durch den Startup Manager (Startup Manager siehe 4.1.3) werden in Abhängigkeit von dieser XML-Datei Objekte erzeugt, welche die auszuführenden Operationen und die Resource Adapter abhängigen Informationen kapseln. Der Anwendungsentwickler braucht nur noch die auszuführende Operation aus der XML-Datei anzugeben und die Eingabeparameter zu spezifizieren. Dieses geschieht über eine Ereignisklasse, die unabhängig vom CCI des Resource Adapter ist. Den eigentlichen Aufruf über das CCI übernimmt dann die JEF Connector Komponente. Durch diesen Ansatz werden die beiden Nachteile aus dem vorangegangenen Ansatz aufgehoben.

1. Würde in einer bestehenden JEF Applikation ein anderer Resource Adapter eingesetzt, müssten keine Schnittstellen angepasst werden, da der Aufruf einer

Operation über eine Ereignisklasse stattfindet, die unabhängig vom Resource Adapter ist.

2. Der Anwendungsentwickler spezifiziert nur die auszuführende Operation und die Eingabeparameter. Die JEF Connector Komponente verbirgt die Komplexität von JCA und dem zu integrierenden EIS.

Das Transaktions- und Sicherheits-Management wird über den Deployment Deskriptor[1] des Resource Adapters konfiguriert.

5.2 JEF Komponente

Nach der JEF Spezifikation [JEF02] haben JEF Komponenten ein Interface zu implementieren, welches vom Interface `com.lhsystems.j2ee.components.IComponent` erbt. Die Komponentenklasse für die JEF Connector Komponente ist die Klasse `com.lhsystems.j2ee.components.jca.EISRepositoryImplementation`. Diese implementiert das Interface `EISRepository`, welches vom JEF Interface `IComponent` erbt. Die Abhängigkeiten der Klassen und Interfaces der JEF Connector Komponente werden durch das Klassendiagramm 5.1 verdeutlicht:

[1] Deployment Deskriptor siehe Glossar.

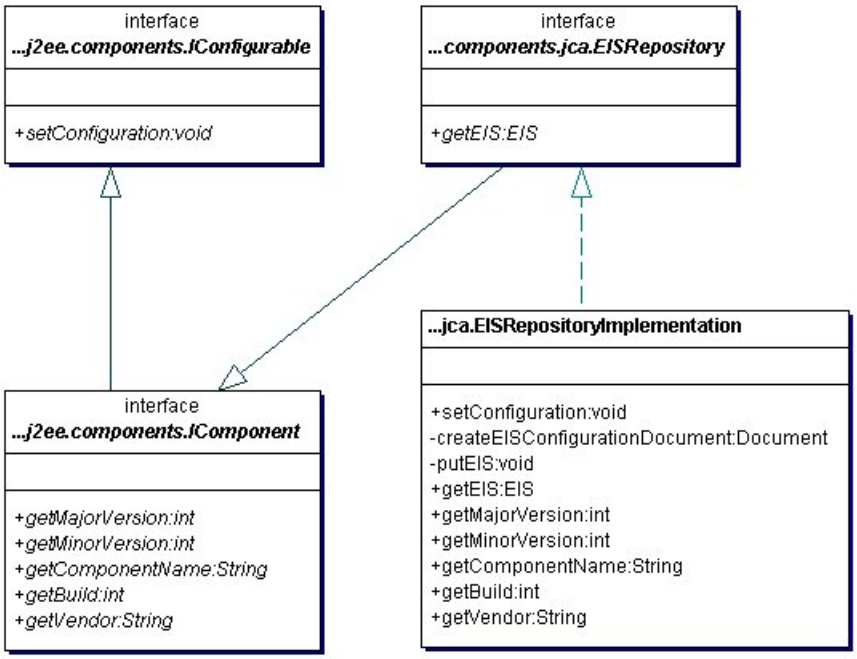

Abbildung 5.1: JEF Komponente

In dem JEF Interface `IComponent` sind getter Methoden vereinbart, über die Meta Informationen der Komponente abgefragt werden können. Diese Meta Informationen sind exemplarisch Versionsnummer, Hersteller und Name der Komponente.
Das Interface `IConfigurable` deklariert die `void setConfiguration(IConfiguration configuration)` Methode. Diese Methode wird vom Startup Manager aufgerufen, wodurch die Konfiguration der Komponente erfolgt (siehe JEF Komponenten 4.1.3). Die Parameter für die Konfiguration können über das Interface `IConfiguration configuration` abgefragt werden. Die Werte der Parameter werden in einer XML-Datei festgelegt. Die Bezeichnung für diese XML Datei ist frei wählbar. In den weiteren Ausführungen wird diese Datei aus Verständnisgründen als „Connector.xml“ bezeichnet.

5.2.1 Connector.xml

In der Connector.xml wird festgelegt, auf welche EISs in einer JEF Applikation zugegriffen werden soll. Dieses geschieht über Tupel. Der Schlüssel des Tupels ist der JNDI Name der `ConnectionFactory` für das jeweilige EIS. Der zugehörige Wert ist eine URL, die auf eine weitere XML-Datei verweist. Diese Datei wird in den weiteren Ausführungen als „EISConfigurator.xml“ bezeichnet.

5.2.2 EISConfigurator.xml

Über diese EISConfigurator.xml werden die Operationen definiert, die in dem jeweiligen EIS ausgeführt werden sollen. Da die Erweiterung des CCI herstellerabhängig ist, ist auch die Methodik, wie eine aufzurufende Operation definiert wird und welche Informationen dafür benötigt werden, EIS spezifisch. Daraus ergibt sich, dass die Struktur der EISConfigurator.xml und die enthaltenen Informationen abhängig vom erweiterten CCI des jeweiligen Resource Adapter ist. Die Struktur und die benötigten Informationen werden daher in einem XML Schema definiert. Das XML Schema dient neben der Validierung als Metainformation für den Anwendungsentwickler, wie die jeweilige EISConfigurator.xml Datei für gültige Operationsdefinitionen strukturiert sein muss.

5.2.3 Initialisierung

Zunächst ein vollständiges Klassendiagramm des Package `com.lhsystems.j2ee.components.jca.*`.

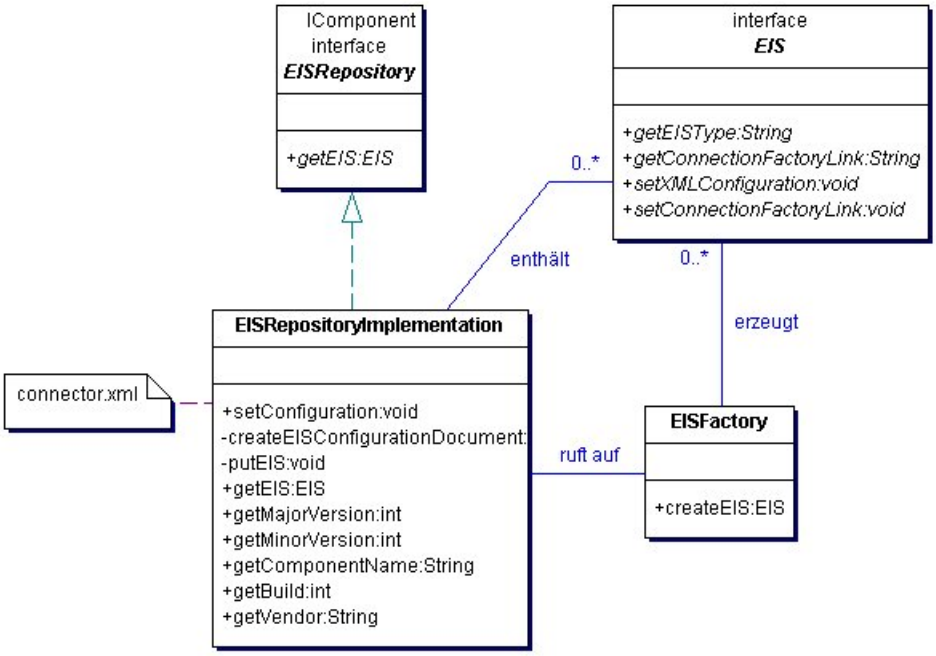

Abbildung 5.2: Package com.lhsystems.j2ee.components.jca.*

In diesem Package sind die Klassen enthalten, die zur adapterunabhängigen Initialisierung der JEF Connector Komponente benötigt werden. Der Botschaftenfluss bei der Initialisierung der JEF Connector Komponente wird anhand des Sequenzdiagramm 5.3 illustriert.

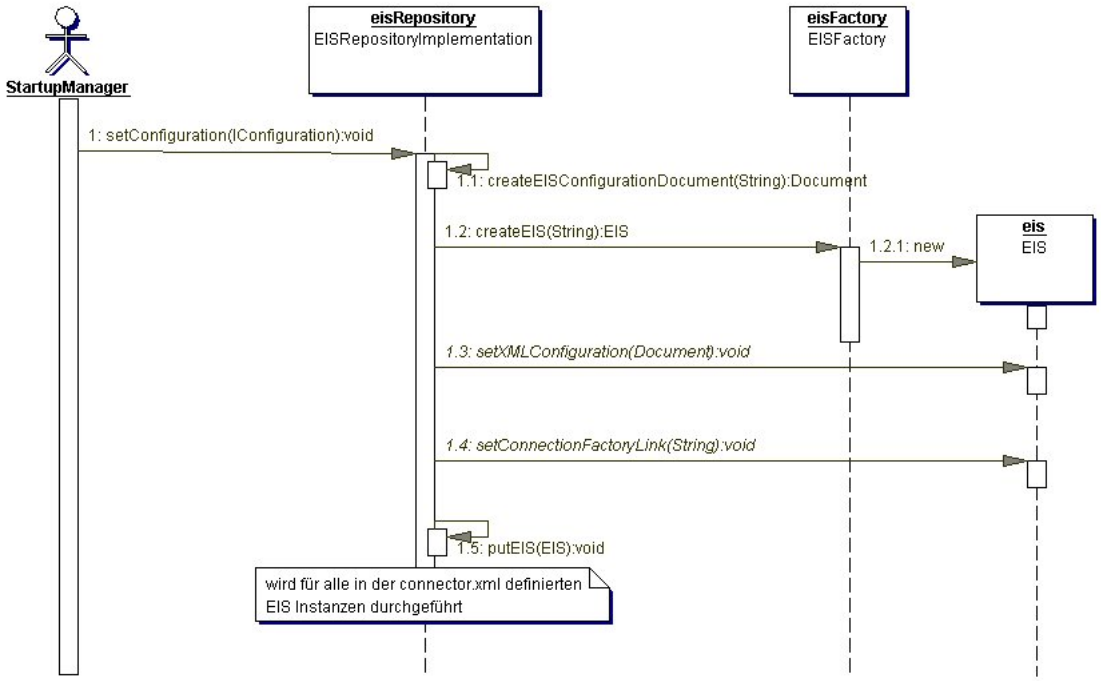

Abbildung 5.3: Initialisierung JEF Connector Komponente

1 Der Startup Manager erzeugt Objekte von den Komponenten, die in der JEF Applikation verwendet werden sollen. In diesem Fall also auch von der Klasse `EISRepositoryImplementation`. Nach der erfolgreichen Erzeugung ruft der Startup Manager die `setConfiguration(configuration:IConfiguration)` auf.

1.1 Es wird über alle Tupel in der Connector.xml iteriert. Der Schlüssel des Tupels ist, wie bereits erläutert, der JNDI Name der `ConnectionFactory`. Der dazugehörige Wert ist eine URL. Anhand dieser URL wird die EISConfigurator.xml ermittelt. Die URL wird der `createEISConfigurationDocument(eisConfigurationURL:String):Document` als String übergeben. In dieser Methode wird die XML-Datei geparst und validiert. Über die SAX-Schnittstellen (SAX siehe 4.4.3) wird ein `org.jdom.Document` Objekt erzeugt und zurückgegeben.

1.2 In der EISConfigurator.xml ist angegeben, um welchen EIS Typ es sich handelt (SAP R/3 oder FileNet usw). Abhängig vom EIS Typ erzeugt die `EISFactory` Objekte, welche das Interface `EIS` implementiert.

1.3 Über die `setXMLConfiguration(configuration:Document):void` Methode des `EIS` Interface wird das in 1.1 erzeugte `Document` Objekt übergeben. Anhand dieses `Document` Objekt werden die Objekte erzeugt, welche die aufzurufenden Operationen und die Resource Adapter spezifischen Informationen kapseln.

1.4 Über die `setConnectionFactoryLink(link:String):void` Methode wird dem `EIS` Objekt der JNDI Name der `ConnectionFactory` übergeben. Beim Aufruf einer Operation wird über diesen JNDI Namen ein `ConnectionFactory` Objekt aus dem JNDI Kontext des Applikationsservers ermittelt. Durch die `ConnectionFactory` wird ein `Connection` Objekt erzeugt, welche auf Applikationsebene eine logische Verbindung zum jeweiligen EIS repräsentiert. Über dieses `Connection` Objekt erfolgt schliesslich der Operationsaufruf über das CCI des Resource Adapters.

1.5 Das erzeugte `EIS` Objekt wird im `EISRepository` abgelegt.

Soweit also zur adapterunabhängigen Initialisierung der JEF Connector Komponente über den Startup Manager. Es können beliebig viele JCA konforme Resource Adapter in eine JEF Applikation über die Connector.xml integriert werden.

5.3 Integration von BAPIs

Für die Integration von BAPIs wird der JCA konforme Resource Adapter der Firma Insevo Inc. eingesetzt (Insevo Resource Adapter siehe Kapitel 6.1). Hier zunächst

das XML Schema, über das die Resource Adapter spezifische Struktur der EISConfigurator.xml definiert ist.

Listing 5.1: EISConfigurator Schema (siehe Anhang:/jefon/jefconweb/web-inf/xml/-sapr3.xsd

```
<xsd:element name="EIS">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="BAPI"  maxOccurs="unbounded">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="OBJECTTYPE" type="xsd:string"/>
        <xsd:element name="OBJECTNAME" type="xsd:string"/>
        <xsd:element name="METHOD" maxOccurs="unbounded">
          <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="FUNCTIONNAME"
                         type="xsd:string"/>
             <xsd:element name="FUNCTIONCALLTYPE"
                         type="xsd:unsignedByte"/>
             <xsd:element name="METHODNAME"
                         type="xsd:string"/>
          </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="EISTYPE" use="required" >
    <xsd:simpleType>
    <xsd:restriction base="xsd:string">
       <xsd:enumeration value="SAPR3"/>
       <xsd:enumeration value="FILENET"/>
    </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="EISCLASS" use="required"
                 type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

Folgend das Klassendiagramm, welches ein objektorientiertes Abbild des XML Schema ist.

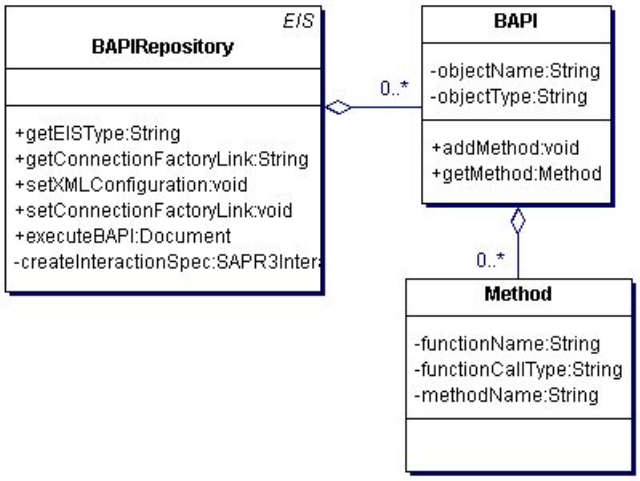

Abbildung 5.4: Package com.lhsystems.j2ee.components.jca.eis.sap.*

Das Erzeugen der Objekte findet in der `setXMLConfiguration(configuration:Document):void` Methode eines Objektes statt, welches das Interface `EIS` implementiert. In den Objekten dieser Klassen werden die Informationen gekapselt, welche für den BAPI Aufruf über das CCI des Resource Adapters von Insevo benötigt werden.

5.3.1 BAPI Aufruf

Abbildung 5.5 auf der nächsten Seite zeigt ein Klassendiagramm, in dem die Klassen aufgezeigt werden, welche für den Aufruf eines BAPIs benötigt werden.

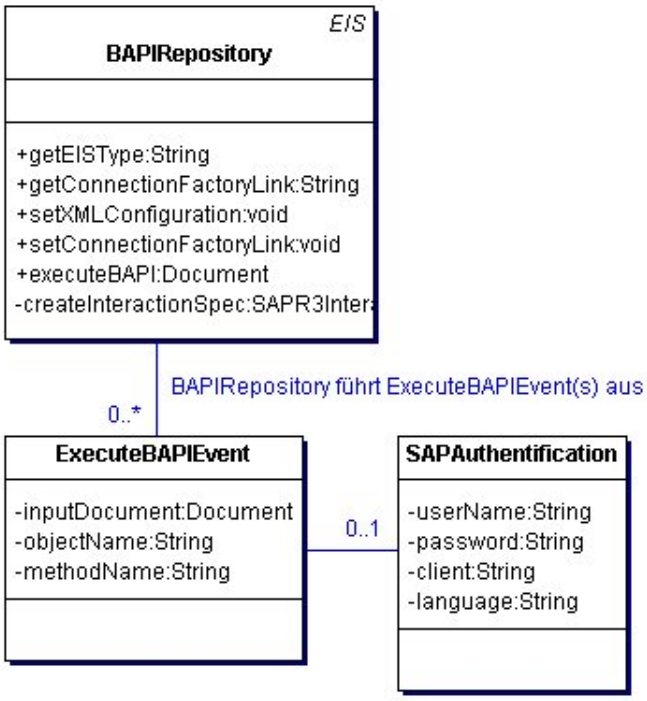

Abbildung 5.5: Klassendiagramm ExecuteBAPIEvent

Laut Abschnitt 4.3.1 wird ein BAPI über die Angabe des Namens vom Business Objekttyp gefolgt vom Namen des BAPIs eindeutig identifiziert. Welcher BAPI vom welchen Business Objekttyp aufgerufen werden soll, wird in einem Objekt der Klasse `ExecuteBAPEvent` festgelegt. Soll eine Component-Managed Sign-on erfolgen, geschieht dieses über ein Objekt der Klasse `SAPAuthentification`. Der BAPI Aufruf erfolgt in der `public Document executeBAPI (ExecuteBAPIEvent event)` der Klasse `BAPIRepository`. Der Ablauf in dieser Methode soll anhand des folgenden Sequenzdiagramms illustriert werden:

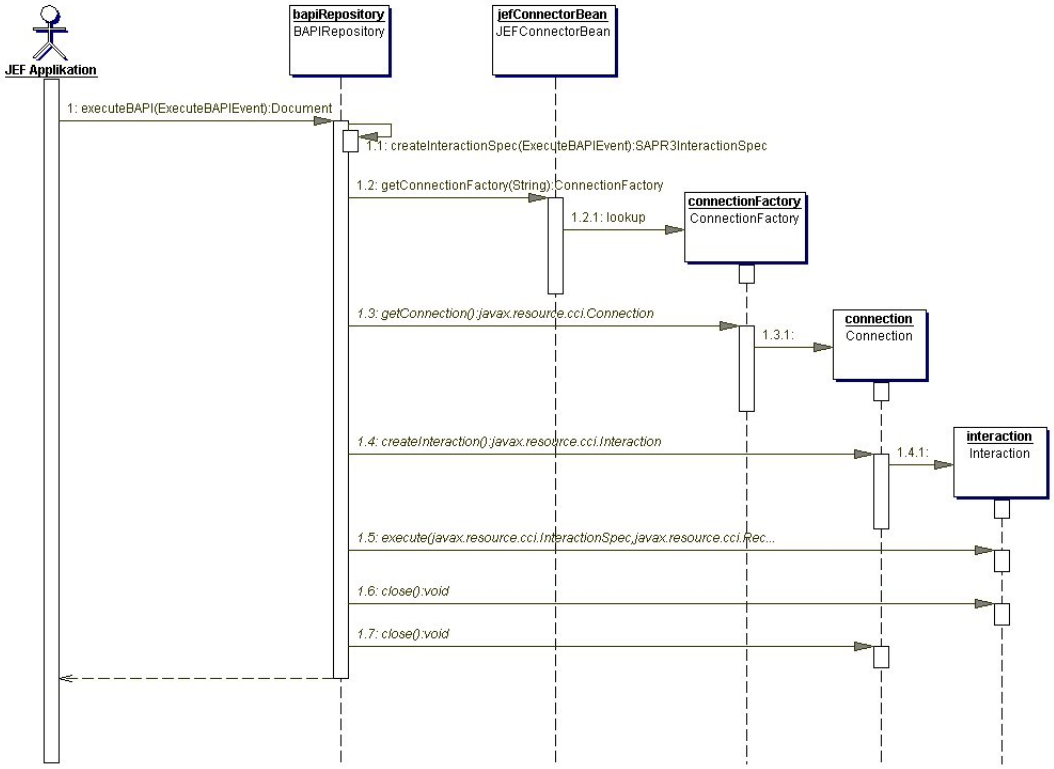

Abbildung 5.6: Sequenzdiagramm executeBAPI()

1.1 In der `ExecuteBAPIEvent` ist der auszuführende BAPI und der Business Objekttyp angegeben. Anhand dieser Angaben werden aus dem `EISRepository` die Objekte ermittelt, welche die Resource Adapter spezifischen Informationen kapseln. Diese Informationen werden einem Objekt der Klasse `SAPR3InteractionSpec` übergeben. Die Klasse `SAPR3InteractionSpec` ist eine Erweiterung der CCI Schnittstelle `InteractionSpec` (`InteractionSpec` siehe 4.2.3). Dieses `SAPR3InteractionSpec` Objekt enthält alle Resource Adapter spezifischen Attribute und Methoden, die zur Ausführung eines BAPI über den Resource Adapter von Insevo nötig sind.

1.2 In der Stateless Session Bean `JEFConnectorBean` sind zentral die Referenzen auf

die `ConnectionFactory` Objekte konfiguriert. Der Parameter für diese Methode ist der JNDI Name der `ConnectionFactory`.

1.3 Über die ermittelte `ConnectionFactory` wird ein `Connection` Objekt erzeugt, welches eine logische Verbindung zum EIS repräsentiert.

1.4 Erzeugung eines `Interaction` Objektes, über das EIS Operationen ausgeführt werden können.

1.5 Über die `execute()` Methode des `Interaction` Objektes wird der in dem `SAPR3-InteractionSpec` spezifizierte BAPI ausgeführt.

1.6 Schliessen der `Interaction`.

1.7 Schliessen der `Connection` und Rückgabe der Ergebnismenge an den Aufrufer.

5.3.2 Parameterübergabe

Die Eingabeparameter und der Rückgabewert für ein BAPI sind in einem Objekt der Klasse `org.jdom.Document` gekapselt. Eine andere Schnittstelle für die Parameterübergabe/rückgabe wird von dem Resource Adapter von Insevo Inc. nicht geboten. Dieses `Document` Objekt muss in der jetzigen Version des Entwurfes manuell erzeugt bzw. ausgelesen werden. Die Struktur des `Documents` hängt dabei von der Parameterliste und von den Typen der Parameter ab. Die Elementnamen vom `Document` Objekt entsprechen den Variablenbezeichnungen des Funktionsbausteins, der hinter dem Methodennamen des jeweiligen BAPIs liegt. Der Anwendungsentwickler muss also die Parameterliste der Methode und die Namen der Variablen kennen, um ein gültiges `Document` zu erstellen bzw. auszulesen. Der Anwendungsentwickler müsste Kenntnisse besitzen, wie ein BAPI aufgebaut ist und wie er an die Variablenbezeichnungen des Funktionsbausteins gelangt.

Dieses widerspricht der innerbetrieblichen Anforderung nach einem „einfachen“ Aufruf. Um diese Problematik zu vermeiden, existieren zwei Möglichkeiten:

1. Es wird ein anderer Resource Adapter für den Aufruf von BAPI Methoden eingesetzt, der alternative Schnittstellen für die Parameterübergabe/rückgabe anbietet. Dieses könnten, der JCA Spezifikation entsprechend, komfortablere Strukturen für die Parameterübergabe/rückgabe sein, wie z.B. `MappedRecord` siehe 4.2.3). Hier wäre der SAP R/3 Resource Adapter von In-Q-My zu nennen, der das CCI bis auf den Datentyp `IndexedRecord` unterstützt (vgl. [In-01, Seite 9]). Falls allerdings der Resource Adapter von Insevo Inc. produktiv bei der LSYBS eingesetzt wird, sollte bei der nächsten Version der JEF Connector Komponente der folgende zweite Ansatz erfolgen:

2. Die Struktur der Parameterliste für ein BAPI wird über ein XML Schema beschrieben. Dieses Schema wird manuell erstellt und in der EISConfigurator.xml für den jeweiligen BAPI referenziert. Die Werte für die Parameter werden in einem `MappedRecord` hinterlegt. Der Schlüssel für den `MappedRecord` ist der Name des Elements, der im XML Schema festgelegt ist. Die JEF Connector Komponente erstellt anhand des XML Schemas und des `MappedRecord` ein XML Dokument. Der Anwendungentwickler kann die Werte für die Ein/Ausgabe Parameter über den Schlüssel des `MappedRecord` festlegen, der in dem XML Schema angegeben ist. Es werden keine Kenntnisse über die Struktur der Parameterliste und Variablenbezeichnungen der Parameterliste benötigt.

Zur Zeit wird bei der LSYBS noch der Einsatz weiterer Resource Adapter für den Aufruf von BAPI Methoden untersucht. Die Entscheidung, welcher Ansatz weiter verfolgt wird, steht also noch aus. Zur Zeit muss das `Document` Objekt manuell erzeugt/ausgelesen werden.

5.4 Anforderungserfüllung

In diesen Abschnitt wird untersucht, ob durch den Entwurf die innerbetrieblichen- und Kundenanforderungen erfüllt werden konnten. Die hervorgehobene Text ist eine Zusammenfassung der ursprünglichen Anforderung.

5.4.1 Kundenanforderung

1. **Integration von SAP R/3 auf BAPI Ebene:** Die Hauptanforderung nach der Integration von SAP R/3 auf BAPI Ebene wurde in Abschnitt 5.3.1 behandelt. Die Integration von SAP R/3 auf BAPI Ebene wurde durch auf den Einsatz des JCA konformen Resource Adapter der Firma Insevo Inc. ermöglicht.

2. **Integration weiterer EIS:** Die Integration von weiteren EISs wurde in Abschnitt 5.2.1 behandelt. Die JEF Connector Komponente kann um weitere Resource Adapter ergänzt werden. Welche Resource Adapter für eine JEF Anwendung eingesetzt werden, wird über die Connector.xml definiert.

3. **Lauffähigkeit auf dem BEA Weblogic 6.1:** Die Erfüllbarkeit dieser Anforderung hängt hauptsächlich davon ab, ob der BEA Weblogic 6.1 die JCA unterstützt. Da nach Aussage von BEA eine Unterstützung der JCA beim Weblogic 6.1 gegeben ist, gilt diese Anforderung als erfüllt (vgl. [Bea03b]). Auf die technischen Schwierigkeiten mit dem BEA Weblogic 6.1 wird in Abschnitt 6.2 eingegangen.

4. **Synchrone Kommunikation:** Nach der JCA Spezifikation 1.0 wird nur synchrone Kommunikation unterstützt (vgl. [SSN01, Seite 85]. Da die JEF Connector Komponente auf der JCA Spezifikation 1.0 basiert, ist die Anforderung damit erfüllt.

5.4.2 Innerbetriebliche Anforderung

1. **JEF Connector Komponente muss der Definition einer JEF Komponente entsprechen:** Durch die Implementierung des Interfaces `IComponent` und die Konfiguration über XML Dateien ist diese Anforderung erfüllt (siehe Abschnitt 5.2).

2. **Einfacher Operationsaufruf** Die Anforderung nach einem „einfachen“ Operationsaufruf ist nur teilweise erfüllt. Der eigentliche Aufruf ist durch eine Resource Adapter unabhängige Ereignisklasse vereinfacht. Für die Parameterübergabe/rückgabe wird zum jetzigen Entwurfszeitpunkt Kenntnisse von SAP R/3 benötigt. Und zwar für das Erstellen/Auslesen des `Document` Objektes, welches die Parameterwerte kapselt.

Bis auf eine Anforderung konnten alle Anforderungen durch den Entwurf vollständig erfüllt werden.

5.5 Zusammenfassung

Die aufzurufenden Operationen und die Resource Adapter spezifischen Informationen werden in der EISConfigurator Datei hinterlegt. Die Struktur dieser Datei ist abhängig von den verwendeten Resource Adaptern. Es werden bei der Initialisierung der JEF Connector Komponente Objekte erzeugt, welche die aufzurufenden Operationen und die hierfür notwendigen Informationen kapseln. Der Aufruf einer Operation erfolgt über eine Eventklasse, welche vom eingesetzten Resoure Adapter unabhängig ist. Wird eine Operation aufgerufen, werden die in den Objekten hinterlegten Informationen verwendet, um einen Aufruf über die Resource Adapter spezifischen Schnittstellen abzusetzen. Der Einsatz der JEF Connector Komponente

erfolgt in einer Beispielapplikation im Kapitel 6.

Kapitel 6

Umsetzung

In diesem Kapitel wird die technische Umsetzung einer auf JEF basierenden Applikation beschrieben, die auf die JEF Connector Komponente zugreift. Die Bezeichnung dieser Applikation ist Jefcon. Über Jefcon soll es einem Benutzer ermöglicht werden, sich Detailinformationen zu einer Materialbestellung aus dem SAP R/3-System anzeigen zu lassen. Zunächst wird der verwendete Resource Adapter für den SAP R/3 Zugriff beschrieben. Danach erfolgt die Konfiguration des Applikationsservers. Schliesslich folgt eine Beschreibung der JEF Applikation Jefcon. Sämtliche Sourcen für diese Applikation befinden sich unter Anhang:/Jefcon.

6.1 Resource Adapter

Die JEF Connector Komponente basiert auf JCA konformen Resource Adaptern. Pro EIS muss ein JCA konformer Resource Adapter vorhanden sein. Für Jefcon wird der SAP R/3 Resource Adapter aus den Business Process Connector (BPC) Version 1.5 der Firma Insevo Inc. verwendet. Es wurden mehrere Unternehmen kontaktiert, ob diese einen Resource Adapter für eine Diplomarbeit frei zur Verfügung stellen

würden. Nur die Firma Insevo Inc. erklärte sich hierzu bereit.

6.1.1 Insevo Business Process Connector

Der Insevo Business Process Connector (BPC) unterstützt die Integration zwischen einem Applikationsserver und mehreren EISs. Der BPC basiert auf JCA konformen Resource Adaptern. Die Resource Adapter werden um Funktionalitäten erweitert, die in der JCA Version 1.0 nicht vorhanden sind (z.B. asynchrone Kommunikation zwischen Applikationsserver und EIS). Über eine Weboberfläche können sogenannte Services definiert werden. Ein Service repräsentiert eine EIS Operation, die von einer Applikationskomponente aufgerufen wird. Zusätzlich können neben den Services Events definiert werden, die zur asynchronen Kommunikation dienen.(vgl. [Ins02, Seite 2])
Abbildung 6.1 auf der nächsten Seite zeigt die Weboberfläche, über die Services und Events definiert werden können:

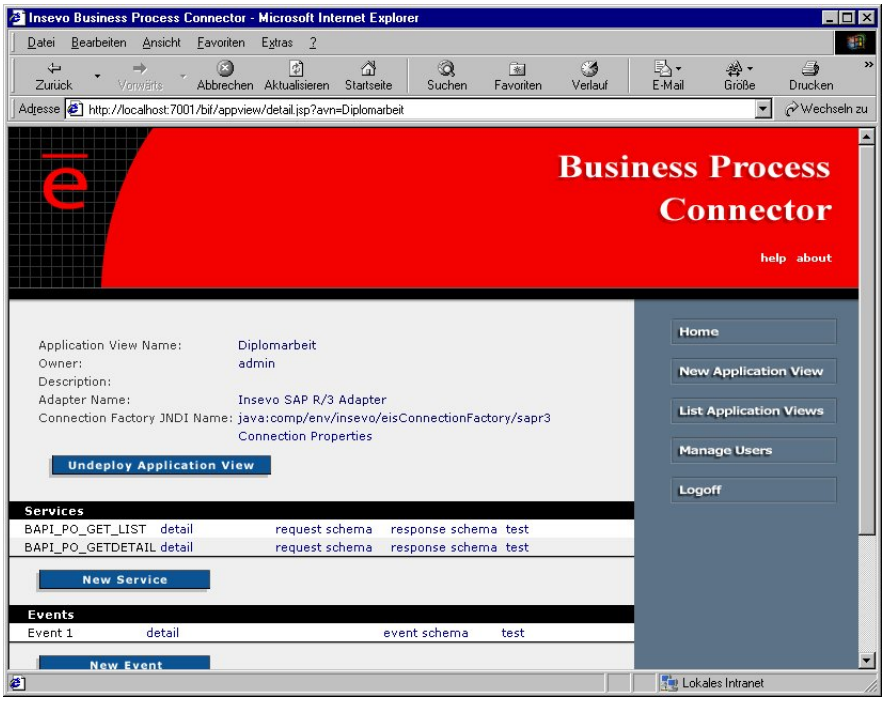

Abbildung 6.1: Business Process Connector

Über den BPC können neben SAP R/3 auch weitere EIS integriert werden. In der BPC Version 1.5 sind dieses Siebel, JD Edwards und PeopleSoft (vgl. [Ins02, Seite 5]). Für Jefcon wird nur der Resource Adapter für das SAP R/3-System, der in dem BPC enthalten ist, benötigt. Der Resource Adapter basiert auf dem JCo (JCo siehe 4.3.2). Nach der erfolgreichen Installierung des BPC wurden zwei Dateien generiert.

sapr3.jar Die sapr3.jar Datei enthält alle Klassen, die der Resource Adapter für die Implementierung des CCI und des System Contract benötigt.

sapr3.rar die sapr3.rar enthält einen Deployment Descriptor, über welchen der System Contract (siehe 4.2.2) zwischen dem Applikationsserver und dem Resource Adapter konfiguriert wird. Abhängig vom eingesetzten Applikationsserver kann die sapr3.rar weitere Deployment Descriptoren enthalten. Diese

Deployment Descriptors enthalten Parameter, die der Applikationsserver für das erfolgreiche Deployment eines Resource Adapters benötigt.

Es wurde keine Leistungsbeschreibung des Resource Adapters von Insevo Inc. zur Verfügung gestellt. Einige Leistungen in Bezug auf die JCA Spezifikation sind durch das SAP R/3 System selbst beschränkt. Die weiteren Leistungen wurden durch Testen des Resource Adapters oder durch die Standardwerte des mitgelieferten Deployment Deskriptors ermittelt.

- Keine Unterstützung von globalen und lokalen Transaktionen.
- CCI wird unterstützt (bis auf Mapped und Indexed Record).
- Authentifizierung nur durch Benutzername/Passwort-Kombination möglich.
- Durch die Beschränkung des SAP R/3 Systems keine Reauthentifizierung möglich (vgl.[SSN01, Seite 265]).

Im nächsten Abschnitt wird detaillierter auf den Deployment Descriptor für die Jefcon Applikation eingegangen. Der Deployment Descriptor ist unter Anhang:/-connectorModule/meta-inf/ra.xml zu finden.

6.1.2 Deployment Descriptor

Über die ra.xml erfolgt die Konfiguration des System Contract. Es wird nur auf die Elemente eingegangen, die für die Konfiguration des Verbindungs-, Transaktions- und Sicherheits-Management von massgeblicher Bedeutung sind.

Verbindungs-Management Es werden die Resource Adapter spezifischen Klassen angegeben, die die Verbindungen zum jeweiligen EIS herstellen.

Listing 6.1: Verbindungs-Management

```
<managedconnectionfactory-class>
   com.insevo.ra.sapr3.spi.SAPR3ManagedConnectionFactory
</managedconnectionfactory-class>
 <connectionfactory-interface>
   javax.resource.cci.ConnectionFactory
 </connectionfactory-interface>
 <connectionfactory-impl-class>
   com.insevo.ra.base.cci.ConnectionFactoryImpl
 </connectionfactory-impl-class>
  <connection-interface>
   javax.resource.cci.Connection
 </connection-interface>
 <connection-impl-class>
   com.insevo.ra.base.cci.ConnectionImpl
 </connection-impl-class>
```

Für eine Verbindung zum EIS benötigt der Resource Adapter Netzwerkinformationen, die typischerweise als Schüssel-Wert Kombination angegeben sind.

Listing 6.2: Verbindungs-Management Netzwerkinformation

```
<config-property>
   <config-property-name>
      url
   </config-property-name>
   <config-property-type>
      java.lang.String
   </config-property-type>
   <config-property-value>
      sap.frankfurt.dlh.de
   </config-property-value>
</config-property>
```

Der JNDI Name der `ConnectionFactory`, über die eine Applikationskomponente Verbindungen zum jeweiligen EIS herstellen kann.

Listing 6.3: Verbindungs-Management JNDI Name

```
<config-property>
   <config-property-name>
      JndiName
   </config-property-name>
   <config-property-type>
      java.lang.String</config-property-type>
   <config-property-value>
      insevo/eisConnectionFactory/sapr3
   </config-property-value>
</config-property>
```

Transaktions-Management Der Resource Adapter unterstützt keine lokale und keine globale Transaktion.

Listing 6.4: Transaktions-Management

```
<transaction-support>NoTransaction</transaction-support>
```

Sicherheits-Management Als Authentifikationsmechanismus wird eine Benutzername/Passwort Kombination verwendet. Im Falle eines Container-Managed Sign-on sind die Anmeldeinformationen in einem `javax.resource.spi.security.PasswordCredential` Objekt hinterlegt.

Listing 6.5: Sicherheits-Management

```
    <authentication-mechanism>
      <authentication-mechanism-type>
         BasicPassword
      </authentication-mechanism-type>
      <credential-interface>
         javax.resource.security.PasswordCredential
      </credential-interface>
    </authentication-mechanism>
```

Soweit zum verwendeten Resource Adapter und dem entsprechenden Deployment Deskriptor. Als nächstes wird das Deployment der Applikationskomponenten beschrieben, die für die JEF Connector Komponente notwendig sind.

6.2 Deployment

Es müssen zwei Applikationskomponenten auf dem Applikationsserver deployed werden, damit eine korrekte Funktionsweise der JEF Connector Komponente gegeben ist.

- JEFConnectorEJB
- Resource Adapter

Auf die Konfiguration der beiden Module und die technischen Schwierigkeiten beim Deployment auf den Bea Weblogic 6.1 wird in den nächsten beiden Abschnitten eingegangen.

6.2.1 JEF Connector EJB

Das JEFConnectorEJB Modul enthält eine Stateless Session Bean mit der Bezeichnung `JEFConnectorBean`. In dieser EJB werden die Referenzen zu den `ConnectionFactory` Objekten verwaltet. Über diese Referenzen wird angegeben, ob ein Container Managed Sign-on oder ein Component Managed Sign-on stattfinden soll. Ebenfalls wird angegeben, ob eine Verbindung „shareable“ ist oder nicht. Diese Angaben erfolgen in dem Deployment Descriptor der `JEFConnectorBean`. Für Jefcon wird folgende Referenz in der `JEFConnectorBean` angelegt:

Listing 6.6: JEF Connector EJB

```
<config-property>
<resource-ref>
   <res-ref-name>jefcon/sap/lhtr04</res-ref-name>
   <res-type>javax.resource.ccl.ConnectionFactory</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Der BEA Weblogic 6.1 wird über eine Webapplikation konfiguriert. Zur Laufzeit des Applikationsservers können über diese Webapplikation Applikationskomponenten deployed werden.

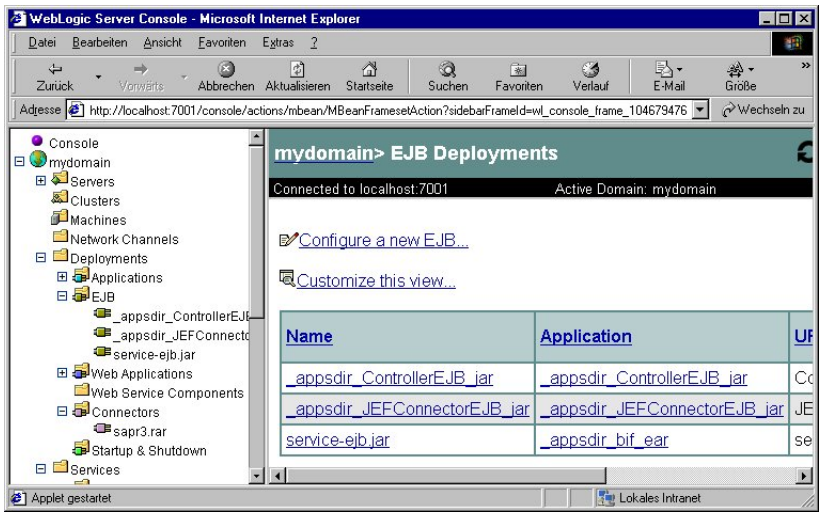

Abbildung 6.2: Weblogic Server Console

Beim Deployment des EJB Moduls traten folgende Probleme auf:

- Trotz der Bestätigung eines erfolgreichen Deployment war die `JEFConnector-Bean` nicht im JNDI Tree vorhanden.
- Wurde der Deployment Desrikptor zur Laufzeit über die Webapplikation be-

arbeitet, so erfolgte beim Speichern der Änderung ohne erkennbaren Grund ein Undeployment des EJB Moduls.

- Beim Hochfahren des Weblogics waren einige Properties des Deployment Deskriptors nicht mehr vorhanden.

Diese Probleme konnten nur durch das mehrmalige Hochfahren des Applikationsservers und/oder Redeployment des Moduls behoben werden. Bei einer Anfrage bei BEA Systems Inc. zu diesen Problemen wurde auf die Weblogic Version 7.0 verwiesen. Daraufhin wurde das Deployment auf den BEA Weblogic 7.0 ausgeführt. Das Deployment verlief wesentlich stabiler. Der Lufthansa Technik AG wurde daraufhin empfohlen, beim Einsatz der JEF Connector Komponente den BEA Weblogic 7.0 einzusetzen.

6.2.2 Resource Adapter

Der Insevo Resource Adapter für SAP R/3 besteht, wie in 6.1.1 bereits erläutert, aus zwei Archivdateien. Der sapr3.jar und der sapr3.rar. Laut JCA Spezifikation müssen alle Schnittstellen und Klassen, die der Resource Adapter benötigt, in eine oder mehrere JAR Dateien gepackt werden. Diese JAR Datei/en und der Deployment Descriptor für den Resource Adapter werden in einer RAR Datei archiviert (vgl. [Jey02, Seite 223]).
Es hat sich jedoch herausgestellt, dass bei diesem Weg der BEA Weblogic 6.1 Schwierigkeiten beim dynamischen Laden von Klassen hat. Diese Schwierigkeiten wurden behoben, in dem die sapr3.jar aus der sapr3.rar Datei ausgelagert und stattdessen in den classpath des Applikationsservers eingebunden wurde. Ansonsten ergaben sich beim Deployment des Resource Adapters über die Webapplikation die gleichen Probleme wie beim Deployment des Moduls JEFConnectorEJB.
Neben dem Deployment Deskriptor für den Resource Adapter enthält die sapr3.rar

noch eine weitere Datei, die weblogic-ra.xml. Diese Datei ist spezifisch für den Weblogic 6.1 und wird für das Deployment eines Resource Adapters benötigt. Hier einige zu konfigurierende Punkte aus der weblogic-ra.xml, die unter Anhang:/connector-Module/meta-inf/weblogic-ra.xml zu finden ist. (vgl.[Bea03a]).

- Parameter für den Connection Pool.
- JNDI Name, um eine 1:1 Assoziation zwischen einem Connection Pool und einer Connection Factory herzustellen.
- Logging Mechanismen.
- Sicherheitsattribute.

Nach dem erfolgreichen Deployment und der Konfiguration der beiden Module kann schliesslich von einer JEF Applikation auf die JEF Connector Komponente zugegriffen werden. Dieses wird in der Beispielapplikation Jefcon verdeutlicht.

6.3 Jefcon

Der Aufbau dieses Abschnittes orientiert sich an der Multitier Architektur, angefangen beim Client bis zu der EIS Tier.

6.3.1 Client Tier

Als Client wird für die Jefcon Applikation ein Applet verwendet. Das Applet besteht hierbei aus nur zwei Masken. Die erste Maske dient zur Eingabe einer Belegnummer, zu der die Detailinformationen ermittelt werden sollen.

Abbildung 6.3: Dialog Purchase Request

In der zweiten Maske wird die Ergebnismenge dargestellt. Es werden Informationen aus dem Bestellkopf angezeigt und die einzelnen Bestellpositionen werden aufgelistet.

Abbildung 6.4: Dialog Purchase Detail

Die Belegnummer wird in einem Ereignisobjekt der Klasse `com.lhsystems.jefcon.events.ShowPurchaseDetailEvent` gekapselt.

Listing 6.7: ShowPurchaseDetailEvent (siehe Anhang:/jefcon/source/)

```
public class ShowPurchaseDetailEvent
             extends ApplicationEventSupport {

 private String purchaseNumber;

 public ShowPurchaseDetailEvent() {
 }

 public String getHandlerClassName() {
  return
  "com.lhsystems.jefcon.handlers.ShowPurchaseDetailHandler";
 }

 protected void addUpdatedModels(Collection collection) {
```

```
    collection.add(JNDI.LHTK04);
  }

  public String getPurchaseNumber() {
    return purchaseNumber;
  }

  public void setPurchaseNumber(String purchaseNumber) {
    this.purchaseNumber = purchaseNumber;
  }
}
```

Dieses Event wird innerhalb der Web Tier vom AppletMainServlet entgegengenommen und an den Request Processor weitergeleitet. Anhand der Methode `getHandlerClassName()` wird vom Request Processor die entsprechende Handler Klasse geladen und die für dieses Ereignisobjekt (Event) definierte Geschäftslogik ausgeführt. Über die `addUpdatedModels(Collection collection)` wird JEF mitgeteilt, welcher Proxy nach der Ausführung der Geschäftslogik zu aktualisieren ist. In der Web Tier muss lediglich das AppletMainServlet konfiguriert werden. JSPs werden nicht benötigt, da als Client ein Applet verwendet wird. Wegen des geringen Implementierungsaufwand in der Web Tier wird diese übersprungen und es folgt die EJB Tier.

6.3.2 EJB Tier

In der EJB Tier wird im Kontext der ClientController Bean die `perform(ApplicationEvent applicationEvent)` Methode in der entsprechenden Handler Klasse durch den `StateHandler` ausgeführt. Durch diese Methode wird die dem Ereignisobjekt(Event) zugeordnete Geschäftslogik ausgeführt. In der Jefcon Applikation wird die Geschäftslogik durch den Aufruf einer BAPI über die JEF Connector Komponente abgbildet. Hierzu ein Ausschnitt aus der `perform(ApplicationEvent applicationEvent)` Methode der `com.lhsystems.jefcon.handlers.ShowPurchaseDetailHandler` Klasse.

Listing 6.8: ShowPurchaseDetailHandler (siehe Anhang:/jefcon/source/)

```
//Festlegen der aufzurufenden Methode des BAPIs.
ExecuteBAPIEvent executeBAPIEvent =
                    new ExecuteBAPIEvent("PurchaseOrder",
                                         "GetDetail");

//Erstellen des InputDocuments
 Document inputDoc = createInputDocument(event);
 executeBAPIEvent.setInputDocument(inputDoc);

//Ausführung
try {
 resultDoc = bapiRepository.executeBAPI(executeBAPIEvent);
 }
catch(ResourceException ex) {
 ex.printStackTrace();
 throw new EventException(ex.
                          getLinkedException().
                          getMessage());
}

 //Erzeugung des Models
 Bestellung bestellung = createBestellungsModel(resultDoc);
}
```

Die notwendige Konfiguration der JEF Connector Komponente erfolgt in der EIS Tier.

6.3.3 EIS Tier

In der Connector.xml werden die EISs festgelegt, auf die in einer JEF Applikation zugegriffen werden soll. Für Jefcon wird auf SAP R/3 zugegriffen.

Listing 6.9: Connector.xml (siehe Anhang:/jefcon/jefconweb/web-inf/xml/sap/lhtr04.xml)

```
<component>
  <class-name
  name="...components.jca.EISRepositoryImplementation"/>
  <configuration type="properties">
    <property
       key="java:comp/env/jefcon/sap/lhtr04"
       value="file:...jefconweb/web-inf/xml/sap/lhtr04.xml"/>
  /configuration>
  <extensions>
  </extensions>
</component>
```

Der Schlüssel ist der JNDI Name der `ConnectionFactory`. Der JNDI Name wurde im Deployment Deskriptor festgelegt. Der Wert ist eine URL, die auf eine XML-Datei verweist. In dieser XML-Datei sind die auszuführenden EIS Operationen und die Resource Adapter spezifischen Operationen hinterlegt. Für Jefcon wird nur ein BAPI-Aufruf benötigt, welcher in der lhtr04.xml definiert ist.

Listing 6.10: lhtr04.xml (siehe Anhang:/jefcon/jefconweb/web-inf/xml/components/Connector.xml)

```
<EIS xmlns=".../components/jca/eis/sap"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="...components/jca/eis/sap sapr3.xsd"
   EISTYPE="SAPR3"
   EISCLASS="...jca.eis.sap.BAPIRepository">
   <BAPI>
    <OBJECTTYPE>BUS2012</OBJECTTYPE>
      <OBJECTNAME>PurchaseOrder</OBJECTNAME>
          <METHOD>
            <FUNCTIONNAME>BAPI_PO_GETDETAIL</FUNCTIONNAME>
            <FUNCTIONCALLTYPE>0</FUNCTIONCALLTYPE>
            <METHODNAME>GetDetail</METHODNAME>
        </METHOD>
   </BAPI>
</EIS>
```

Die notwendigen Informationen für die lhtr04.xml liefert der BAPI Explorer des SAP R/3 Systems.

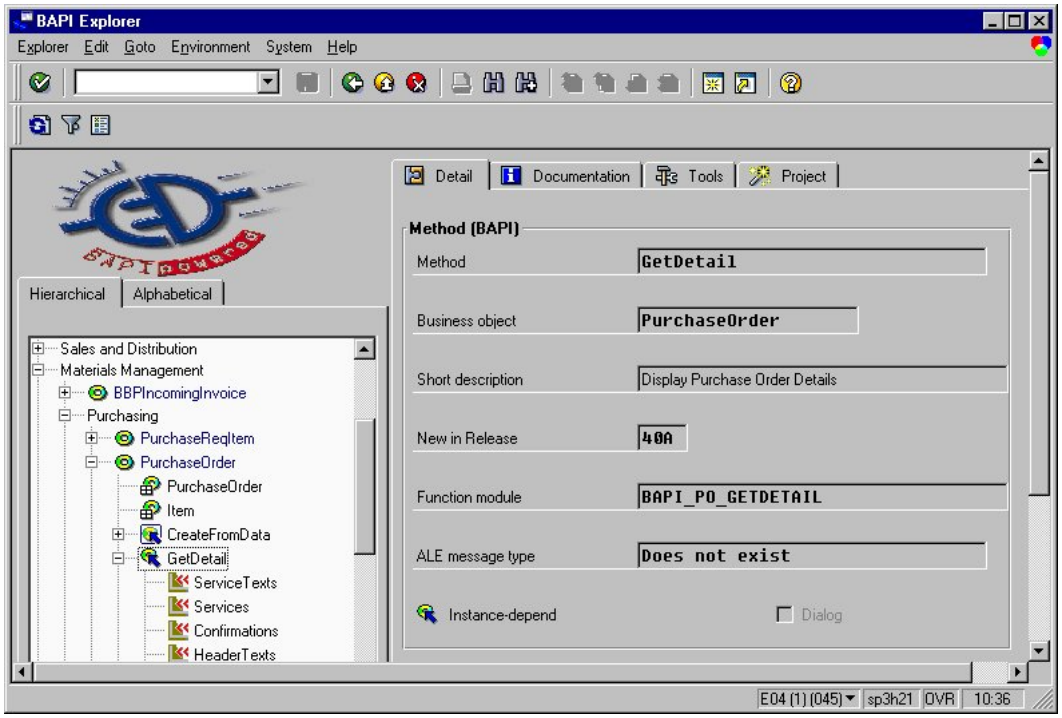

Abbildung 6.5: BAPI Explorer

6.4 Verwendete Entwicklungs- und Laufzeitumgebungen

Als Entwicklungsplattform wurde ein Rechner mit dem Betriebssystem Windows NT verwendet. Bei der LSYBS sind zur Zeit zwei Entwicklungsumgebungen im Einsatz, die für die Entwicklung der JEF Connector Komponente und von Jefcon verwendet wurden:

- JBuilder 7.0.155.107 Enterprise Edition von Borland.
- WebSphere Studio Application Developer 5.0.0 (WSAD 5) von IBM.

Beide Entwicklungsumgebungen verwenden das JDK 1.3.1.03 von Sun Microsystems Inc. Zur Versionskontrolle wurde ein CVS Server eingerichtet, der über WinCVS Version 1.2 angesprochen wurde. Die JEF Connector Komponente wurde auf dem Applikationsserver BEA Weblogic 6.1, BEA Weblogic 7.0 und IBM Websphere 5.0 deployed und getestet. Über die JEF Connector Komponente erfolgte der Zugriff auf SAP R/3 4.6c.

Kapitel 7

Schlussbetrachtung

In diesem Kapitel werden die bisherigen Ausführungen zusammengefasst und es erfolgt ein Ausblick.

7.1 Zusammenfassung

Es existieren in Unternehmen häufig mehrere heterogene Applikationen mit einem definierten Aufgabengebiet. Es entstand der Bedarf, die in den heterogenen Applikationen abgebildeten Teilprozesse und Daten in Geschäftsprozessen zusammenzufassen. Die Technologien, mit welchen die Applikationen implementiert sind, waren typischerweise nicht darauf ausgelegt, miteinander zu kommunizieren. Die Integration von EIS über den Einsatz von Middleware ist eine komplexe Aufgabe. Um die Integration von EISs in J2EE Applikationen zu erleichtern, wurde von Sun Microsystems die J2EE Connector Architecture (JCA) entwickelt. Die JCA soll als Grundlage für eine Komponente des Java Enterprise Framework dienen. Das Java Enterprise Framework dient bei der Lufthansa Systems Business Solutions GmbH als Implementierungsstandard für J2EE Applikationen. Die Komponente soll dabei als

Standard für die Integration von EISs aus dem Java Enterprise Framework heraus dienen. Die innerbetrieblichen- und Kundenanforderungen wurden auf Erfüllbarkeit untersucht und es wurde anhand einer Beispielanwendung die Integration von SAP R/3 auf BAPI-Ebene erläutert.

7.2 Ausblick

EAI wird auch weiterhin ein komplexes Gebiet sein. Es wird immer Applikationen geben, die mit aktuellen Technologien implementiert werden, welche jedoch nicht mit älteren Technologien interagieren können. Sun hat durch die JCA eine sinnvolle Architektur entwickelt, um EIS in J2EE Anwendungen zu integrieren. Durch die in der JCA Spezifikation 1.5 zusätzlich definierten Verträge wird diese Architektur um weitere Funktionalität, wie z.B. asynchrone Kommunikation, erweitert. Durch diese zusätzliche Funktionalität werden auch die Integrationsmöglichkeiten von EISs in JEF Applikationen zunehmen. Dieses wird auch auf die JEF Connector Komponente Einfluss haben.

Anhang A

CD

Das Hauptverzeichnis der dieser Diplomarbeit beiliegenden CD besteht aus 6 Verzeichnissen mit folgenden Inhalt:

Abbildungen Enthält alle Abbildungen, die in der Diplomarbeit verwendet wurden.

ConnectorModule Enthält die Deployment Descriptoren für den Resource Adapter.

Diplomarbeit Enthält die Diplomarbeit in digitaler Form als PDF-Datei.

Jef Enthält die Sourcen der JEF Connector Komponente.

Jefcon Enthält die Sourcen der Jefcon Applikation.

Quellen Enthält einige verwendete Quellen.

Literaturverzeichnis

[ABD01] Allamaraju, Subrahmanyam, Cedric Buest und John Davies: *Professional Java Server Programming, J2EE 1.3 Edition*. Wrox Press, 2001.

[Anf01] Anft, Stephan: *Konzept zum Einsatz der Java 2 Enterprise Edition bei der Lufthansa Systems Business Solutions*. Technischer Bericht, Diplomarbeit FH NORDAKADEMIE Elmshorn, 28. August 2001.

[Bal01] Balzert, Helmut: *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag GmbH, 2 Auflage, 2001.

[Bea03a] Bea Systems, Inc.: *Configuring the weblogic-ra.xml File*. http://e-docs.bea.com/wls/docs61/jconnector/config.html\#1234356, Zugegriffen am 26. Februar 2003.

[Bea03b] Bea Systems, Inc.: *Overview of the WebLogic J2EE Connector Architecture*. http://e-docs.bea.com/wls/docs61/jconnector/overview.html\#1208997, Zugegriffen am 25. Februar 2003.

[Bes95] Beste, Frank: *Kerberos*. Technischer Bericht, Universität Paderborn, Fachbereich Informatik, 31. Mai 1995. ANHANG:/[Bes95]/Kerberos.ps.

[Cos03] Costello, Roger L.: *XML Schema*. http://users.info.unicaen.fr/g̃irault/SAVED/NAPI/XML/xml-schemas1.ppt, Zugegriffen am 16. April 2003. Letze Änderung: 11. Dezember 2001.

[Cum02] Cummins, Fred A.: *Enterprise Integration: An Architecture for Enterprise Application and Systems Integration*. John Wiley & Sons, Inc., 2002.

[DYK01] DeMichiel, Linda, L. Ümit Yalçinalp und Sanjeev Krishnan: *Enterprise JavaBeans Specification, Version 2.0*. Sun Microsystems, Inc., 22 August 2001. ANHANG:/quellen/[DYK01]/ejb-2_0-fr2-spec.pdf.

[ebi03] ebizQ: *Glossar*. http://messageq.ebizq.net/shared/glossary.html, Zugegriffen am 26. April 2003.

[Eng93] Engesser, Hermann: *Duden Informatik*. Bibliographisches Institut & F.A. Brockhaus AG, 2 Auflage, 1993.

[GHJ96] Gamma, Erich, Richard Helm und Ralph Johnson: *Entwurfsmuster, Elemente wiederverwendbarer objektorientierter Software*. Addison Wesley Longman, 1996.

[In-01] In-Q-My Technologies GmbH: *The SAP Resource Adapter*, 15. Juni 2001. ANHANG:/[In-01]/SAPResourceAdapter.pdf.

[Ins02] Insevo, Inc.: *Insevo Business Process Connector User Guide*, July 2002. ANHANG:/quellen/[Ins02]/BIF_WLS_UG_15.pdf.

[JEF02] JEF Team: *Java Enterprise Framework, Programmers Guide 1.1*. Lufthansa Systems Business Solutions GmbH AM/I1, 2002.

[Jey02] Jeyaraman, Ram: *J2EE ConnectorArchitecture Specification, Java 2 Enterprise Edition, Version 1.5*. Sun Microsystems, Inc., 31 Oktober 2002. ANHANG:/quellen/[Jey02]/j2ee_connector-1_5-fr-spec.pdf.

[Kas01] Kassem, Nicholas: *Designing enterprise applications with the Java 2 platform, Enterprise Edition*. Addison Wesley, 2001.

[Kel02] Keller, Wolfgang: *Enterprise Application Integration, Erfahrungen aus der Praxis*. dpunkt-Verlag, 2002.

[Luf01] Lufthansa System Group GmbH: *Die neue Unternehmensgruppe*, 9. August 2001. ANHANG:/[Luf01]/Präsentation_LSY_09_08-deutsch.ppt.

[Luf02] Lufthansa System Group GmbH: *Geschäftsbericht*, 9. April 2002. ANHANG:/[Luf02]/gb2001_de.pdf.

[Luf03a] Lufthansa Technik AG: *Unsere Mission*. http://www.lufthansa-technik.de/e/company/mission/mission.html, Zugegriffen am 12. April 2003.

[Luf03b] Lufthansa Technik AG: *Wir über uns.* http://lww.lht.ham.dlh.de/lht/allg-inf/about.htm, Zugegriffen am 29. März 2003. Letze Änderung: 24 Januar 2001.

[Mal03] Malina, Mathias: *Alles Wissenwerte über SAP R/3.* http://lww.lht.ham.dlh.de/sap-web, Zugegriffen am 19. März 2003. Letze Änderung: 25 Februar 2003.

[McL01] McLaughin, Brett: *Java & XML.* JO'Reilly & Asssociates, Inc., 2 Auflage, August 2001.

[RMB01] Ruh, William A., Francis X. Maginnis und William J. Brown: *Enterprise Application Integration, A Wiley Tech Brief.* John Wiley & Sons, Inc., 2001.

[Sai01] Sailer, Mario: *Anforderungen, Entwicklung und Trends im Bereich Enterprise Application Integration (EAI).* SerCon GmbH, 6. April 2001. ANHANG:/[Sai01]/text_sailer_sercon.pdf.

[SAP03] SAP AG: *SAP Bibliothek: Business Framework Architecture.* http://help.sap.com, Zugegriffen am 3. April 2003.

[Sch02] Schuessler, Thomas G.: *Developing Applications with the „SAP Java Connector" (JCo).* ARAsoft GmbH, 2002. ANHANG:/[Sch02]/JCo Tutorial.pdf.

[SSN01] Sharma, Rahul, Beth Stearns und Tony Ng: *J2EE Connector Architecture and Enterprise Application Integration.* Addison Wesley, Dezember 2001.

[Sta03] Stachowiak, Lisa: *FileNET das zentrale Archivsystem der LHT.* http://lww.lht.ham.dlh.de/sap-web/systemmanagement/systeme/filenet.htm, Zugegriffen am 2. April 2003. Letze Änderung: 31 Oktober 2002.

[Sun03a] Sun Microsystems, Inc.: *J2EE Connector Architecture.* http://java.sun.com/j2ee/connector, Zugegriffen am 28. März 2003.

[Sun03b] Sun Microsystems, Inc.: *Package javax.transaction.xa.* http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/transaction/xa-/package-summary.html, Zugegriffen am 28. April 2003.

[TBPSM03] Tim Bray, Tim, Jean Paoli und C. M. Sperberg-McQueen: *Extensible Markup Language (XML) 1.0.* http://edition-w3c.de/TR/2000/REC-xml-20001006/#sec-intro, Zugegriffen am 3. April 2003. Letze Änderung: 6. Oktober 2000.

[Tho99] Thomas, Anne: *Java 2 Plattform, Enterprise Edition.* Sun Microsystems, Inc., 31 Juni 1999. ANHANG:/quellen/[Tho99]/j2eeAnneThomas.pdf.

Glossar

Application Programming Interface (API) Schnittstelle, über die ein standardisierter Aufruf von *Operationen* erfolgt.

Applikationskomponente Eine Softwareeinheit auf Applikationslevel, die von einem *Container* verwaltet und über einen *Deployment Deskriptor* konfiguriert wird. Die *J2EE* Plattform definiert 4 Typen von Applikationskomponenten: *Enterprise Java Beans*, Web Komponenten, Applets und Applikationsclients. (vgl. [SSN01, Seite 366])

Authentifikation Ein Prozess, bei dem sichergestellt wird, ob eine Instanz auch wirklich die Identität besitzt, mit der diese sich gegenüber einer anderen Instanz ausgibt. (vgl [SSN01, 365])

Botschaft „In der deutschen Literatur wird anstelle des Begriffs Botschaft häufig *Nachricht* verwendet. Der Begriff *Nachricht* ist jedoch in den Bereichen Datenübertragung und Betriebsysteme mit einer anderen Bedeutung belegt als der Begriff Botschaft im objektorientierten Paradigma.“ [Bal01, Seite 207]

Common Client Interface (CCI) Eine Menge von standardisierten Klassen und Schnittstellen, die EIS typische Operationen ermöglichen.

Container Eine standardisierte Laufzeitumgebung, die Dienste für die *J2EE* Applikationen anbietet. Bei den Diensten handelt es sich z.B. um Deploymentmechanismen, Transaktionsverwaltung und Sicherheitsmanagement. (vgl. [SSN01, Seite 366])

Deployment Der Prozess, bei dem eine *J2EE* Applikation in dem Adressraum des Applikationsservers eingebunden und installiert wird.

Deployment Deskriptor Eine XML Datei, die deklerative Attribute für das *Deployment* einer *J2EE* Applikation enthält. (vgl. [SSN01, Seite 367])

Empfänger Ein Empfänger ist eine Softwarekomponente, die eine Anfrage von einer anderen Softwarekomponente erhält. (vgl. [RMB01, Seite 41])

Enterprise Information System (EIS) Ein EIS verwaltet die Informationen eines Unternehmens und bietet Dienste für den Zugriff und die Manipulation dieser Informationen an. (vgl. [Jey02, Seite 14])

Enterprise Java Bean (EJB) Mit dem Begriff verbindet SUN Microsystems eine komponentenbasierte Architektur für die Entwicklung von objekt-orientierten und verteilten Applikationen. Eine EJB ist dabei eine Server-Komponente, über die die Geschäftsprozesse abgebildet und die von einem Container verwaltet wird. (vgl. [DYK01, Seite 41]).

Enterprise Resource Planning (ERP) Applikationen, welche ein breites Spektrum von den Geschäftsprozessen eines Unternehmens abbildet, z.B. SAP R/3. (vgl. [SSN01, Seite 368])

Entity Bean Eine Entity Bean ist eine *EJB*, die eine objektorientierte Sicht auf eine Entität bietet, die in einer zugrundeliegenden Datenbank oder einem *ERP* System abgelegt ist. Die Persistenz dieser Entität kann entweder von der Bean oder vom Container verwaltet werden. Eine Entity Bean wird über einen Primary Key eindeutig identifiziert. (vgl [DYK01, Seite 243])

Extensible Markup Language (XML) Eine standardisierte Sprache zur Beschreibung und Erstellung von strukturierten Dokumenten.

Java 2 Enterprise Edition (J2EE) Eine Menge aus Java 2 Technologien und Spezifikationen, die für die Entwicklung von serverseitigen Applikationen ausgerichtet ist.

Java Enterprise Framework (JEF) Eine Framework, über das die Implementierung von *J2EE* Applikationen standardisiert wird. Die Funktionalität von JEF kann über eine Komponenten Architektur erweitert werden.

Java Naming and Directory Interface (JNDI) Eine *API* für Verzeichnis- und Namensdienste.

Kerberos „Kerberos ist ein System, welches eine Zugriffskontrolle bei unsicheren Netzwerken ermöglicht. Es basiert auf einem key-distributen-Modell von Needham und Schroeder [NS78]. Es gestattet die Authentifizierung von Teilnehmern (Server bzw. Services und Benutzern).“ [Bes95]

Managed Umgebung Eine Managed Umgebung definiert eine operationale Umgebung für eine auf *J2EE* Technologien basierende Multitier Applikation, die auf *EISs* zugreift. (vgl. [Jey02, Seite 15])

Message siehe Nachricht.

Nachricht Eine Nachricht enthält strukturierte Informationen über eine auszuführende Aktion und die dafür benötigten Daten. (vgl. [RMB01, Seite 49])

Non-Managed Umgebung Eine Non-Managed Umgebung definiert eine operationale Umgebung für eine 2-Tier Applikation. (vgl. [Jey02, Seite 15])

Operation Im Rahmen dieser Diplomarbeit wird der Begriff Operation als ein Oberbegriff für Funktion, Prozedur und Methode verstanden.

Resource Adapter Über einen Resource Adapter werden über herstellerspezifische Schnittstellen physikalische Verbindungen zum jeweiligen *EIS* aufgebaut. Eine *Applikationskomponente* greift über standardisierte Schnittstellen auf den Resource Adapter zu. Durch einen *System Contract* wird das Verbindungs,- Transaktions- und Sicherheitsmanagement zwischen dem Applikationsserver und dem *EIS* geregelt. (vgl. [Jey02, Seite 18-19])

Resource Manager Ein Resource Manager ist für die Verwaltung von *EIS* Resourcen, auf die konkurrierend zugegriffen wird, zuständig. Ein Resource Manager kann an Transaktionen teilnehmen, die extern von einem Transaktions Manager verwaltet und gesteuert werden. (vgl. [Jey02, Seite 14])

SAP R/3 Ein international eingesetztes Standardprogramm für betriebswirtschaftliche Anwendungen in Unternehmen. Die Software bildet mit seinen betriebswirtschaftlichen Standardanwendungen Geschäftsprozesse im Unternehmen ab.

Sender Bei einem Sender handelt es sich um eine Softwarekomponente, die eine Anfrage an eine andere Softwarekomponente sendet. (vgl. [RMB01, Seite 41])

Session Bean Eine Session Bean ist eine *EJB*, die die Geschäftslogik einer Applikation abbildet. Es wird unterschieden zwischen einer stateless oder stateful Session Bean. Eine stateful Session Bean ist genau einem Client zugeordnet und der Container sorgt dafür, dass der Zustand der Bean für die Lebensdauer einer Sitzung erhalten bleibt. Bei einer stateless Session Bean ist das nicht der Fall. (vgl. [DYK01, Seite 69])

System Contract Eine Menge von Verträgen zwischen einem *Resource Adapter* und einem Applikationsserver, die das Verbindungs,- Transaktions- und Sicherheitsmanagement zwischen dem Applikationsserver und einem EIS regeln. (vgl. [Jey02, Seite 18])

Index

Erklärung zur Diplomarbeit

Name:	Ahnfeldt
Vorname:	Maik
Matrikelnr.:	134468

An den Prüfungsausschuss des Fachbereich Wirtschaft
der Fachhochschule Nordostniedersachsen
Volgershall 1
21339 Lüneburg

Erklärung zur Diplomarbeit

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Lüneburg, den _____

_____
(Maik Ahnfeldt)