Fachhochschule Konstanz
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences

Entwicklung eines Monitoringtool in einer J2EE Umgebung als Teil eines Community Portals

Herbert Hartmann

Walldorf, 25. Oktober 2002

Diplomarbeit

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

an der

Fachhochschule Konstanz

- Hochschule für Technik, Wirtschaft und Gestaltung -

im Fachbereich Informatik / Wirtschaftsinformatik

Thema:	Entwicklung eines Monitoringtool in einer J2EE Umgebung als Teil eines Community Portals
Diplomand:	Herbert Hartmann Maierhof 1 88364 Wolfegg
Firma:	SAP AG Neurottstrasse 15 69190 Walldorf
Betreuer:	Professor Dr. Ralf Leibscher, Fachhochschule Konstanz Dipl.-Inform. Heinz Würth, SAP AG, X Product Team ERM
Abgabetermin:	25.10.2002

Zusammenfassung

Thema:	**Entwicklung eines Monitoringtool in einer J2EE Umgebung als Teil eines Community Portals**
Diplomand:	Herbert Hartmann Maierhof 1 88364 Wolfegg
Firma:	SAP AG Neurottstrasse 15 69190 Walldorf
Betreuer:	Professor Dr. Ralf Leibscher, Fachhochschule Konstanz Dipl.-Inform. Heinz Würth, SAP AG, X Product Team ERM
Abgabetermin:	25.10.2002
Schlagworte:	J2EE, EJB, R/3 Monitoring, SAP J2EE-Engine, SAP Enterprise Portal, Zugriff von Java auf R/3, Portal-Service, Portal-Komponenten

Gegenstand dieser Arbeit ist ein Tool, das das Monitoring von Tabellen über mehrere SAP R/3-Systeme hinweg ermöglichen soll. Es wird in einer J2EE Umgebung implementiert und in ein Community Portal eingebettet. Das Tool soll dabei periodisch, innerhalb von vorgebbaren Zeitabständen automatisch starten.
Für diese Aufgabe gibt es ein R/3-Programm, das abgelöst werden soll und dessen Funktionsumfang Grundlage der Neuentwicklung ist. Es müssen drei Tabellen aus den R/3-Systemen gelesen werden: die Tabelle, welche die Loginzeiten der Benutzer je System und Mandant beinhaltet, die Tabellen mit den Systemmeldungen und die mit den Transporten.
Die Tabellen werden im neuen Tool über den RFC (Remote Funktion Call) Funktionbaustein RFC_READ_TABLE aus den R/3-Systemen gelesen. Diese Daten werden dann in einer externen Datenbank gespeichert. Um über ein Portal auf die Daten zugreifen zu können, wird eine entsprechende Schnittstelle entwickelt.
Die gewünschten Tabellen und Einschränkungen werden über eine Konfigurationsdatei definiert. Somit ist es möglich, nicht nur die drei genannten Tabellen aus den R/3-Systemen zu lesen. Es werden für alle Systeme und deren Mandanten auftretende Fehler dokumentiert, um dem Administrator des Monitors die Möglichkeit zu geben, einfach herauszufinden, warum bestimmte Systeme eventuell nicht ausgelesen werden konnten.. Eine Fehlermeldung aus dem Portal kann, im internen R/3-Fehlermeldungssystem, für das entsprechende System angelegt werden. Zudem wird bei jedem Durchlauf die Anzahl der gelesenen Datensätze je System und Tabelle für mögliche Auswertungen dokumentiert.
Abschliessend werden Erweiterungen beschrieben, welche in möglichen weiteren Versionen umgesetzt werden könnten. Dazu müssen allerdings noch eingehendere Vor- und Nachteils Betrachtungen angestellt werden.

Inhaltsverzeichnis

Vorwort

Mit Abschluss dieser Arbeit werde ich mein Studium der Wirtschaftsinformatik abschliessen. Dabei möchte ich diese Gelegenheit nutzen mich bei allen Personen zu bedanken, die mich bei der Erstellung dieser Diplomarbeit unterstützt haben.

Besonderen Dank gebührt meinem Betreuer an der Fachhochschule, Professor Dr. Ralf Leibscher. Er hat sich viel Zeit genommen, um mir während der Diplomarbeitszeit begleitend und beratend zur Seite zustehen. Zusätzlich hat er Besprechungstermine immer für mich vorteilhaft angesetzt und stand auch jederzeit für Fragen und Probleme zur Verfügung.

Desweiteren möchte ich mich generell bei allen meinen Kollegen der Abteilung X Product Team ERM (ehemals DWP) der SAP AG bedanken. Dabei möchte ich einige besonders hervorheben:

- Meinen Betreuer Heinz Würth, der mir die Chance und das Vertrauen gab, meine Diplomarbeit in seiner Abteilung zu erstellen.
- Cyrille Waguet, der mir sowohl bei technologischen Fragen als auch beim Finden von Lösungswegen jederzeit tatkräftig zur Seite stand und mich auch immer wieder zu neuen Ideen angeregt hat.
- Joachim Lindenberg, der mich beim Softwareentwicklungsprozess und bei der Erstellung dieser Arbeit mit konstruktiver Kritik jederzeit unterstützt hat.
- Annette Häußler, die durch ihre Empfehlung mir es ermöglicht hat meine Arbeit bei der SAP anzufertigen.

Desweiteren möchte ich mich bei meiner Familie bedanken, besonders bei meinem Bruder Josef. Er hat mich schon während des gesamten Studiums unterstützt und hat viel Zeit in das Korrekturlesen meiner Arbeit investiert.

Abschliessend sollen noch zwei meiner Kommilitonen und Freunde an dieser Stelle erwähnt werden, Timo Seeger und Makram Sakji. Sie haben mich über die ganze Zeit hinweg ermutigt und mir dabei geholfen, Dinge wieder richtig zu sehen, wenn ich vielleicht schon über das Ziel hinausgeschossen war.

Walldorf, den 25.10.2002

Herbert Hartmann

Abkürzungsverzeichnis

API	:	Application Programming Interface
BMP	:	Bean-managed persistence
CMP	:	Container-managed persistenc
CMR	:	Container-managed relationship
CSN	:	Customer Service New
CSR	:	Customer Service Reporting
CSS	:	Customer Service System
CUE	:	Customer Usage Evaluation
DBMS	:	Database Management System
DD	:	Deployment Descriptor
DNS	:	Domain Name Service
DTD	:	Document Type Definition
EAR	:	Enterprise Archive
EIS	:	Enterprise Information System
EJB	:	Enterprise Java Bean
EJB QL	:	Enterprise Java Bean Query Language
HTML	:	Hypertext Markup Language
HTTP	:	Hypertext Transfer Protocol
IDE	:	Integrated Development Environment
J2EE	:	Java 2 Environment, Enterprise Edition
J2SE	:	Java 2 Environment, Standard Edition
JAR	:	Java Archive
JCo	:	Java Connector
JCP	:	Java Community Process
JDK	:	Java Development Kit
JDT	:	Java Development Tools
JMS	:	Java Messaging Service
JNDI	:	Java Naming und Directory Interface
JSP	:	Java Server Pages
JSR	:	Java Specification Request
JVM	:	Java Virtual Machine
PAR	:	Portal Archive
PDE	:	Plugin development environment
RFC	:	Remote Function Call
RMI	:	Remote Method Invokation
SAP	:	Software Anwendungen Produkte
SDK	:	Software Development Kit
SNC	:	Secure Network Communication
Spec	:	Spezifikation
SQL	:	Sequence Query Language
SSL	:	Secure Socket Layer
TCP/IP	:	Transmission Control Protocol / Internet Protocol
WAR	:	Web Archive
WebAS	:	Web-Applikation-Server
XML	:	Extensible Markup Language
ZAR	:	Portal Service (Zervice) Archive

1 Aufgabenstellung

Innerhalb der SAP gibt es derzeit (Mitte 2002) weit über 1000 interne R/3-Systeme, die verwaltet werden müssen. Verschiedenste Programme bereiten hierfür die Informationen, wie z.B. Systemmeldungen, Loginzeiten der Benutzer oder Transportinformationen auf und visualisieren diese. Grundlage eines schnellen und informativen Zugriffs auf die Daten liefert das ABAP-Programm `ZSCANSYSTEMS` in dem R/3-System `CUE` (Customer Usage Evaluation), welches Daten aus fast allen R/3-Systemen liest und diese wiederum in verschiedenen Tabellen speichert. Auf diese Tabellen greifen andere Programme zu und bereiten die Daten zweckentsprechend auf. Das Programm `ZSCANSYSTEMS`, das die Systeme scannt, wird alle sechs Stunden automatisch ausgeführt. Unabhängig davon, ob eine Anfrage vorhanden ist, die die aktualisierten Daten benötigt. Das CUE-System wird in naher Zukunft nicht mehr zur Verfügung stehen. Da allerdings die Informationen, die durch das bisherige Programm gesammelt werden, weiterhin benötigt werden, wird nach einer neuen Möglichkeit gesucht, diese Daten anderweitig zu sammeln und bereitzustellen.

SAP setzt verschiedene Portale auf ihrem eigenen Enterprise Portal auf, um gewünschte Daten und Informationen jederzeit und von jedem Ort abrufbar und anzeigbar zu machen. Als Grundlage wird die J2EE-Technologie (Java 2 Platform, Enterprise Edition [@SUN02b]) eingesetzt. J2EE bietet durch seine Offenheit und Plattformunabhängigkeit Vorteile.

Diese Arbeit gibt zunächst einen Überblick über den technologischen Hintergrund. Mittels IST-Analyse wird der aktuelle Zustand aufgenommen. Aus dieser Analyse heraus werden die Mängel genauer betrachtet und eine detaillierte Soll-Beschreibung, eine sog. Spezifikation (Spec), erzeugt. Auf Basis dieser Spec wird ein Design erstellt, das im Laufe der Arbeit implementiert wird. Bei der Implementierung werden die allgemeinen SAP-Entwicklungsrichtlinien für die Entwicklung angewandt. Dazu gehören neben der Anforderungsanalyse und Implementierung auch Testläufe und eine Dokumentation. Dieses Projekt soll anschliessend in ein Community Portal eingebettet werden.

Ziel ist es, ein lauffähiges Programm zu entwickeln, das die alte Funktionalität des Programms `ZSCANSYSTEMS` ersetzt, dessen Schwachstellen behebt und die Daten in Portalen bereitstellt.

2 Technologischer Hintergrund

Dieses Kapitel gibt einen Überblick, welche Technologien bei SAP zur Bearbeitung des vorliegenden Projekts eingesetzt werden. Das besondere Augenmerk liegt hierbei auf den Tools zur Entwicklung mit der Programmiersprache Java.

2.1 Verwendung von Java bei SAP

Java ist eine Plattform-unabhängige Programmiersprache. Hierauf aufgebaut gibt es eine Klassenbibliothek, die diverse Schnittstellen zur Erweiterung bietet bzw. spezifiziert. Wegen der Offenheit[1] der Technologie und der grossen Community, die hinter der Technologie steht, hat sich SAP für einen weitreichenden Einsatz von Java entschieden. Zudem engagiert sich SAP bei der Definition und dem Einsatz von Standards. Dadurch wird die Erweiterung, Portierung und Eingliederung von anderen Programmen und Projekten erleichtert.
SAP ist Mitglied im Java Community Process (JCP). Dort arbeiten momentan (Mai 2002) über 400 Firmen an der Weiterentwicklung und Standardisierung der Java Technologie. Zudem kann SAP Java Specification Requests (JSR) erstellen und ist bereits an mehreren JSRs beteiligt.
Seit 2001 ist Java bei SAP offiziell die zweite Entwicklungssprache, wobei meist die J2EE-Umgebung eingesetzt wird. Java wird eingesetzt, da es damit einfacher ist, Daten und Informationen auf verschiedenen Endgeräten bereitzustellen, als mit der SAP eigenen Entwicklungsumgebung ABAP. Dabei wird Java sowohl zur Entwicklung von Kundenlösungen eingesetzt, als auch bei der Tool-Entwicklung.

2.2 Überblick J2EE

Die von Sun spezifizierte J2EE (Java 2 Platform, Enterprise Edition) Umgebung hat als Grundlage das J2SE (Java 2 Platform, Standard Edition). Daher kann es alle Vorteile, die die Java Plattform bereits mit sich bringt (u.a. plattformunabhängig, objekt-orientiert, vorgefertigte Klassenbibliothek, usw.) nutzen.
Die J2EE Spezifikation 1.3 definiert dazu verschiedene Services. Diese können von den Java-Entwicklern genutzt werden und sind für alle Enterprise-Komponenten vollständig zugängig. Jeder dieser Services verfügt über ein API. Die Services sind:

- HTTP, HTTPS,
- J2EE Connector Architecture, welches ein SPI (Service Provider Interface) darstellt, das es ermöglicht mit nicht in Java geschriebenen Systeme zu kommunizieren,
- JavaBeans Activation Framwork (JAF), bietet JavaBeans folgende Funktionalitäten:
  - Es bestimmt den Datentyp von Variablen,
  - kapselt den Zugang zu Daten und
  - stellt fest, welche Operationen of einen bestimmten Datentyp angewendet werden können.
- JavaIDL, ermöglicht es J2EE-Komponenten CORBA-Objekte über das IIOP-Protokoll aufzurufen,
- JavaMail, liefert eine API zur Erstellung von Mail- und Messaging Applikationen,
- Java API for XML Parsing (JAXP), ist ein API zum Parsen von XML-Dateien,
- Java Authentication and Authorization Service (JAAS), ist ein Service zur Benutzerauthentifizierung und Autorisierung,

[1] Offenheit bedeutet in diesem Zusammenhang die Plattform- und Firmenunabhängigkeit und die Möglichkeit bei der Weiterentwicklung und dem Erstellen von Standards mitwirken zu können.

- Java Transaction API (JTA), definiert lokale Java-Schnittstellen zwischen der Transaktionsverwaltung und Komponenten des verteilten Transaktionssystems,
- JDBC, bietet Zugang zu relationalen Datenbanken,
- JMS, ermöglicht asynchrone Kommunikation,
- JNDI, bietet eine Zugangsschnittstelle zu verschiedensten Namens- und Verzeichnisdiensten und
- RMI over IIOP, kombiniert Eigenschaften von RMI und CORBA.

J2EE verwendet ein vielstufiges Applikationsmodell. Die Applikationslogik ist in Funktionskomponenten aufgeteilt. Die verschiedenen Applikationskomponenten können auf verschiedenen Rechnern installiert sein, je nachdem zu welcher Stufe der J2EE-Umgebung sie gehören. Im allgemeinen folgen J2EE-Umgebungen dem Drei-Stufen-Modell (3 tier), die über drei Systeme verteilt sind. Der Client läuft auf der Client-Maschine, die Business- und Web-Stufe auf dem J2EE-Rechner und die Enterprise Information System (EIS)-Stufe auf einem EIS Rechner (z.B. Datenbankserver).

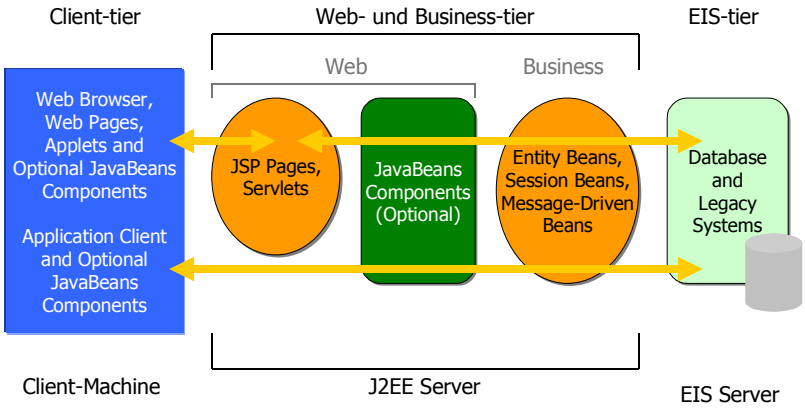

Abb. 2-1: J2EE Drei-Stufen-Modell nach [@SUN02b]

Zusätzlich bietet J2EE die Möglichkeit nach dem MVC (Model-View-Controller) Paradigm zu arbeiten. Hierbei würden die Entity Beans den Model Aspekt abdecken, die Session und Message-Driven Beans den Controller und JSPs und Servlets das View.

2.2.1 J2EE-Komponenten

Eine J2EE-Komponente ist eine eigenständige funktionale Software-Einheit. Integriert in eine J2EE-Applikation mit ihren Klassen und Dateien kommuniziert diese mit anderen Komponenten.
Die J2EE-Spezifikation definiert die folgenden Komponenten:

- Applikations-Clients und Applets sind Komponenten, die beim Client laufen.
- Java Servlet und JavaServer Pages (JSPs) sind Web-Komponenten, die auf einem Web-Server laufen.
- Enterprise JavaBeans (EJBs) sind Business-Komponenten, die auf einem Applikations-Server laufen.

Alle J2EE-Komponenten selbst sind in Java geschrieben und werden kompiliert wie jedes andere Programm auch. Es gibt mehrere Unterschiede zwischen J2EE-Komponenten und Standard Java Klassen. J2EE-Komponenten werden in eine J2EE-Applikation integriert und auf die Einhaltung der J2EE-Spezifikation überprüft. Zudem werden sie in einen J2EE-Server *deployed* (installiert), der sie ausführt und verwaltet (vgl. Abb. 2-2).

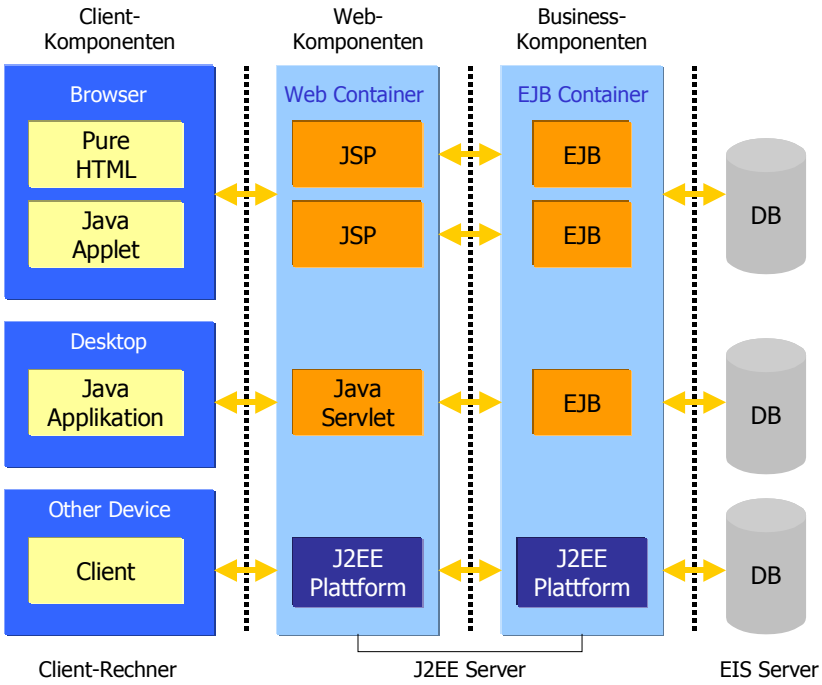

Abb. 2-2: J2EE Applikationsmodell nach [@SUN02b]

Innerhalb dieses Projektes soll die komplette Business-Logik mittels Enterprise JavaBeans implementiert werden. Dies ermöglicht es, Clients so „dünn“ wie möglich zu halten, um die Applikationen so einfach wie möglich verwalten zu können.

2.2.2 J2EE-Container

Für jede Komponente der J2EE-Architektur gibt es Container, die verschiedene Dienste anbieten. Dadurch muss man diese Dienste nicht selbst entwickeln und kann sich auf die Lösung des Business-Problems konzentrieren (separation of concern). Konfigurierbare Dienste, die dabei von den Containern bereitgestellt werden, sind u.a.

- Sicherheitsmechanismen,
- Transaktionsmanagement,
- Remoteverbindungen und
- Java Naming und Directory Interface (JNDI).

Desweiteren gibt es nicht-konfigurierbare Dienste, die von den Containern bereitgestellt werden. Dies sind z.B. Dienste,

- die den Lebenszyklus von EJBs und Servlets bestimmen,
- die das Pooling von Datenbankverbindungen verwalten,
- die die Datenpersistenz sicher stellen oder
- den Zugriff auf J2EE-APIs beschreiben.

Der Dienst, der die Datenpersistenz realisiert, ist immer vorhanden. Trotzdem ist es möglich, dass Entity Beans eine eigene Kontrolle über die Persistenz der Daten implementieren. Dies wird als *bean-managed persistence* bezeichnet, gegenüber der standardmässigen *container-managed persistence*.

Ein J2EE-Server enthält zwei verschiedene Arten von Containern. Einerseits den Web Container, der die Ausführung von JSPs und Servlets verwaltet. Andererseits den EJB Container, der alle Enterprise JavaBeans verwaltet.

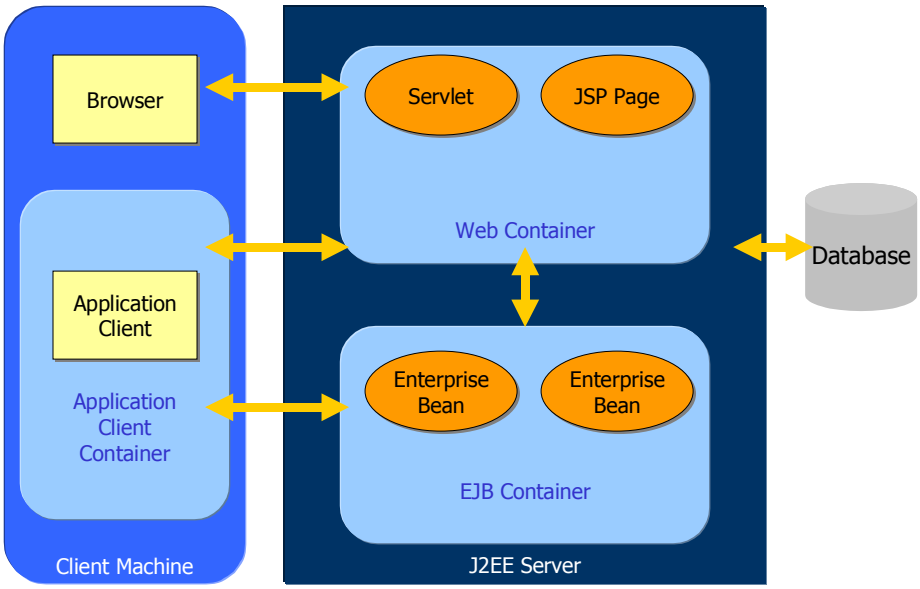

Abb. 2-3: J2EE-Container nach [@SUN02b]

2.2.3 Enterprise JavaBeans (EJBs)

EJBs sind in Java geschriebene server-seitige Komponenten, die Business-Konzepte darstellen. Sie kapseln die Businesslogik einer Applikation. EJBs vereinfachen aus mehreren Gründen die Entwicklung von grossen und verteilten Applikationen. Erstens da der EJB Container System-Level Dienste bereitstellt, zweitens da die Beans, und nicht der Client, die Businesslogik enthalten. Dadurch kann sich der Client-Entwickler auf die Darstellung konzentrieren, drittens sind EJBs portierbare Komponenten.
Beispiele für Geschäftsvorfälle, die durch EJBs abgebildet werden können, sind das Anlegen einer Hotelreservierung, das Anlegen eines neuen Kunden, die Buchung eines Fluges oder auch die Änderung einer Lieferantenanschrift. Allgemein sind Einsatzpunkte von EJBs skalierbare Applikationen oder transaktionsorientierte Module.

EJB2.0 beschreibt drei verschiedene Bean-Typen: *Session Beans*, *Entity Beans* und *Message-driven Beans* (vgl. [EJBS01]).
Session Beans haben u.a. folgende Eigenschafen:

- Sie können immer nur einen Client bedienen.
- Sie können Transaktionen verwalten.
- Sie repräsentieren nicht direkt Daten einer Datenbank, können aber auf diese zugreifen und sie ändern.
- Sie sind relativ kurzlebig.
- Sie werden gelöscht, wenn der EJB-Container abstürzt. Der Client muss ein neues Session Bean erzeugen bevor er weiter arbeiten kann.

Entity Beans haben u.a. Eigenschaften wie folgt:

- Sie stellen eine objekt-orientierte Sicht auf Daten in Datenbanken dar.
- Sie können Business-Objekte repräsentieren.
- Sie erlauben es von mehreren Clients genutzt zu werden.
- Sie sind relativ langlebig. Sie leben solange die Daten in der Datenbank sind.
- Sie überleben einen Absturz des EJB-Containers, da ihre Daten in einer Datenbank gespeichert sind. Sollte der Container abstürzen während eine Transaktion ausgeführt wird, werden die Daten auf den letzten richtigen Stand zurückgeführt (rollback).

Message-driven Beans haben u.a. folgende Eigenschaften:

- Sie werden nach Erhalt einer Nachricht (JMS Message) ausgeführt (JMS = Java Message Service).
- Sie werden asynchron ausgeführt.
- Sie können Transaktionen verwalten.
- Sie spiegeln nicht direkt Daten einer Datenbank wieder, können aber auf solche zugreifen und diese ändern.
- Sie sind relativ kurzlebig.
- Sie haben keinen Zustand.
- Bei einem Absturz des Containers werden sie gelöscht. Der Container muss ein neues message-driven Bean erstellen, um weitere Nachrichten zu bearbeiten.

Die drei Bean-Typen werden weiter unten genauer beschrieben. Dort wird auf ihre Eigenheiten eingegangen, wie z.B. *stateful* / *stateless* Session Beans und Entity Beans mit *container-managed* oder *bean-managed* Persistenz.

2.2.3.1 EJB-Contracts

Die EJB-Architektur beschreibt zwei Verträge (*contracts*). Einmal einen zwischen Client und EJB und zum anderen einen zwischen EJB und Container.

Client-view contract

Die Client-view Verträge bestehen zwischen Client und Container (vgl. [EJBS01]). Sie bieten eine einheitliches Entwicklungsmodell für Applikationen die EJBs als Komponenten verwenden. Dieses Modell erlaubt das wiederverwenden der Komponenten.
Ein Client eines EJBs kann ein anderes EJB sein, das im selbem oder einem anderen Container vorhanden ist. Es kann auch ein eigenständiges Java-Programm sein, beispielsweise eine Anwendung, ein Applet oder ein Servlet. Der Client-view eines EJBs kann darüber hinaus auch für Clients bereitgestellt werden, die nicht in Java entwickelt sind, wie z.B. CORBA Clients.
Der Client eines Session oder Entity Beans kann ein *remote client* oder ein *local client* sein.
Die EJB remote client Sicht ist verteilt nutzbar. Sowohl entfernte (remote) als auch lokale Programme können auf ein EJB über das selbe remote Objekt des EJBs zugreifen. Die Objekte, die die Schnittstellen für die remote Sicht implementieren sind RMI-Objekte. Sie können vom Client über RMI aufgerufen werden.
Die EJB local client Sicht ist nicht verteilt nutzbar. Es können nur lokale Komponenten auf ein EJB über dessen lokale Sicht zugreifen. Lokale Komponenten laufen in der selben Java Virtual Machine (JVM) wie das lokalgenutzte Enterprise JavaBean.
Obwohl es möglich ist ein lokales und ein remote Objekt für ein EJB bereitzustellen, wird normalerweise nur eines der beiden angeboten.
Für beide Sichten muss sowohl ein *Home-* als auch ein *Component-Interface* vom Entwickler bereitgestellt werden.
Das *Home-Interface* eines EJBs definiert Methoden für den Client zum Erstellen, Löschen und Finden von EJB-Objekten des selben Typs. Das Home-Interface muss, wie bereits erwähnt vom Bean-Entwickler bereitgestellt werden. Der Container erstellt dann beim Installieren des Beans eine Klasse, die dieses Interface implementiert. Für remote EJBs muss ein remote Home-Interface erstellt werden, für lokale ein local Home-Interface.

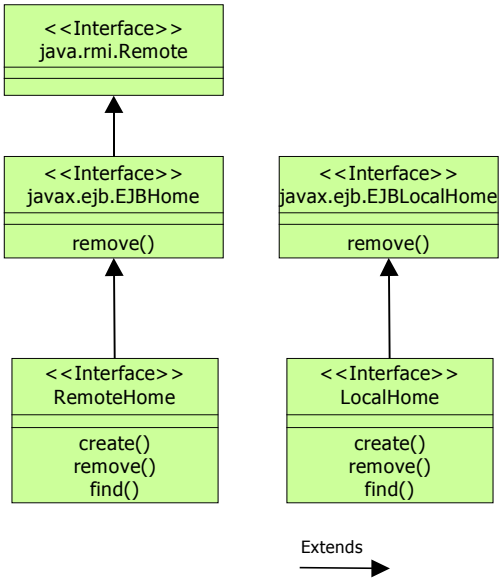

Abb. 2-4: EJB Home-Interfaces

Ein Client kann die Home-Interfaces über das Standart Java Naming and Directory Interface (JNDI[2]) API lokalisieren.

Auf ein EJB-Objekt kann über das *Component-Interface* des EJBs zugegriffen werden. Dieses Interface definiert die Business-Methoden die vom Client aufgerufen werden können. Der Container erzeugt aus dem vom Entwickler bereitgestellten Interface eine Klasse, die dieses implementiert. Auch das Component-Interface ist entweder remote oder local.

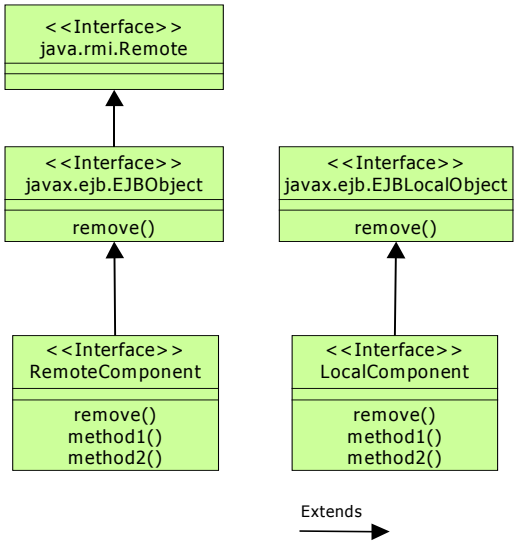

Abb. 2-5: EJB Component-Interfaces

Message-driven Beans haben weder ein Home- noch ein Component-Interface. Daher haben sie auch keine Client-Sicht wie hier beschrieben. Ein Client kann über JNDI ein JMS-Ziel ausfindig machen, an welches er die Nachrichten senden muss, die von einem message-driven Bean abgearbeitet werden sollen.

[2] Genaueres über den JNDI Naming Service wird in diesem Dokument nicht beschrieben, da es den Rahmen sprengen würde. Näheres ist unter [@SUN02b] zu finden.

Component contract
Der Component contract beschreibt das Zusammenarbeiten zwischen einem EJB und dem Container (vgl. [EJBS01]). Dabei erfolgt die Kommunikation zwischen Bean und Container über sogenannte Callback-Methoden. Diese Methoden müssem von der Bean-Klasse implementiert werden. Der Container ruft dann je nach Ereignis, das aufgetreten ist, eine oder mehrere diese Methoden auf. Die Callback-Methoden sind vom Bean-Typ abhängig. Abb. 2-6 zeigt die benötigten Methoden der einzelnen Bean-Arten. Dabei muss eine konkrete Bean-Klasse eine dieser drei Interfaces implementieren.

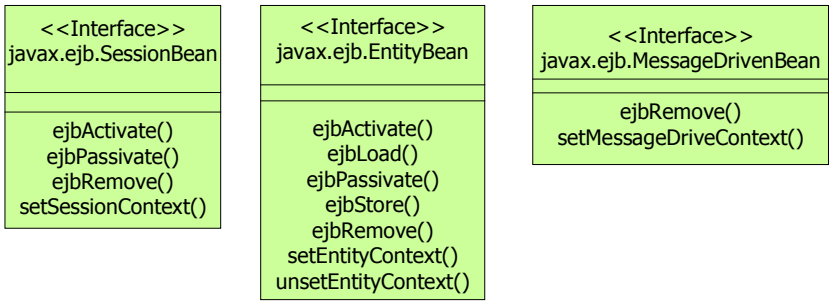

Abb. 2-6: EJB Klassen-Interfaces

Zu den weiteren Anforderungen dieses Vertrags gehört u.a., dass (vgl. [EJBS01])

- der Bean-Entwickler die Business-Methoden des Componenten-Interfaces in der Bean-Klasse implementiert und der Container die Clientaufrufe an diese weiterleitet,
- bei message-driven Beans die onMessage()-Methode in der Bean-Klasse implementiert sein muss und der Container diese aufrufen muss, wenn eine Nachricht für dieses Bean angekommen ist,
- in der Bean-Klasse für jede create()-Methode des Home-Interfaces eine ejbCreate() und für Entity Beans zusätzlich eine ejbPostCreate()-Methode mit der selben Signatur erstellt werden muss die der Container während des Erstellens einer Instanz der Bean aufrufen muss,
- bei bean-managed Entity Beans ejbFind()-Methoden implementiert werden müssen,
- der Container die Persistenz bei container-managed Entity Beans implementieren muss und
- der Container für die EJB-Instanzen Transaktionen, Sicherheitsmechanismen und Ausnahmen verwalten muss.

2.2.3.2 Session Beans

Session Beans dienen dazu Interaktionen von Entity und anderen Session Beans zu verwalten und auf Ressourcen zuzugreifen. Sie führen Aufgaben für den Client aus. Im Gegensatz zu Entity Beans sind Session Beans nicht persistente Business-Objekte. Sie spiegeln keine Daten aus Datenbanken wieder. Session Beans entsprechen dem Controller in einer Model-View-Controller-Architektur. Sie kapseln die Businesslogik in einer 3-tier Architektur. Alle Session Beans müssen das Interface `javax.ejb.SessionBean` implementieren.
Es gibt zwei Typen von Session Beans: *Stateless* (zustandslos) und *stateful* (zustandsbehaftet). Stateless Session Beans bestehen aus Businessmethoden, die sich wie Prozeduren verhalten. Sie arbeiten nur mit den Argumenten, die ihnen beim Aufruf übergeben werden. Stateless Beans werden stateless genannt, da sie kurzlebig sind; sie halten keinen *conversational-state* (Konversationsstatus) zwischen Methodenaufrufen aufrecht. Statetul Session Beans kapseln Businesslogik und Status spezifisch zum Client. Sie werden stateful genannt, da sie den Konversationsstatus zwischen Methoden und Aufrufen aufrechterhalten. Allerdings nur solange sie existieren.

Der conversational-state ist der Status zwischen Client und Bean. Der Client kann über Methoden des Beans verschiedene Daten an das Bean senden, welches die Daten aufrechterhält, bis es gelöscht wird. Der Konversationsstatus kann von mehreren Methoden im selben Stateful Session Bean gemeinsam genutzt werden.

Stateless Session Beans

Stateless Session Beans sind, wie alle Session Beans, keine dauerhaften Business-Objekte. Jeder Aufruf einer Businessmethode ist unabhängig von vorherigen Aufrufen. Da stateless Session Beans zustandslos sind, sind sie für den EJB-Container einfacher zu verwalten. Dadurch werden Anfragen schneller bearbeitet und weniger Ressourcen belegt. Der Performance-Vorteil wird gegen den Nachteil eingetauscht, dass Stateless Session Beans keine Daten von Aufruf zu Aufruf erhalten können.
Stateless Beans können eine Ansammlung von unabhängigen, aber verwandten Diensten bereitstellen. Sie könnten u.a. dazu genutzt werden, um auf Datenbanken zuzugreifen oder um komplexe Berechnungen auszuführen. Sie sind aber nicht auf diese Fälle beschränkt, da sie zum Ausführen jeglicher Art von Diensten genutzt werden könnnen. Stateless Session Beans können auch dazu verwendet werden direkt auf Datenbanken zuzugreifen, oder Interaktionen zwischen anderen Beans zu koordinieren.
Ein Beispiel für ein Stateless Session Bean, ist ein Taschenrechner. Für jede Funktion (z.B. addieren, subtrahieren, usw.) werden entsprechende Methoden entwickelt, denen man die zu berechnenden Werte übergibt.

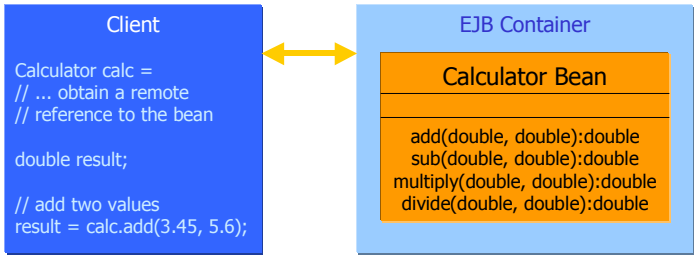

Abb. 2-7: Beispiel eines Stateless Session Bean

Das `Calculator` Bean in Abb. 2-7 definiert vier Business Methoden. Jeder dieser Methoden müssen zwei Werte übergeben werden. Das Bean speichert jedoch weder die übergebenen Werte noch das Ergebnis.

Stateful Session Beans

Stateful Session Beans gehören zu genau einem Client und verwalten einen conversational-state zwischen verschiedenen Methodenaufrufen. Anders als bei Stateless Session Beans, werden Stateful Bean-Instanzen nicht von mehreren Clients verwendet. Wenn ein Client ein Stateful Bean erstellt, stellt diese Bean-Instanz nur diesem Client seine Dienste bereit. Dies ermöglicht es den Konversationsstatus zu verwalten.
Um Ressourcen zu sparen, können Stateful Session Beans passiviert werden, wenn sie längere Zeit nicht gebraucht werden. Dies erfolgt dadurch, dass der conversational-state auf einen Sekundärspeicher geschrieben wird und die Instanz gelöscht wird. Die Referenz, die der Client auf dieses Bean hat, bleibt davon unberührt. Sie bleibt erhalten und nutzbar, während das Bean passiviert ist. Wenn ein Client eine Methode eines passivierten Beans aufruft, aktiviert der Container das Bean. Dabei legt er eine neue Instanz an und setzt den conversational-state auf den Status, der beim Passivieren auf den Sekundärspeicher geschrieben wurde.
Das Standardbeispiel eines Stateful Session Beans, ist ein Warenkorb in einem Webshop. Dabei merkt sich das Bean, über mehrere Aufrufe hinweg, die Artikel, die bereits in den Warenkorb gelegt wurden.

Um als Vergleich auf das Beispiel des Stateless Beans zurückzukommen, soll aus diesem nun ein Stateful Bean gemacht werden. Dabei muss sich das Bean nur die übergebenen Werte merken. Die Methoden arbeiten dann mit diesen Werten.

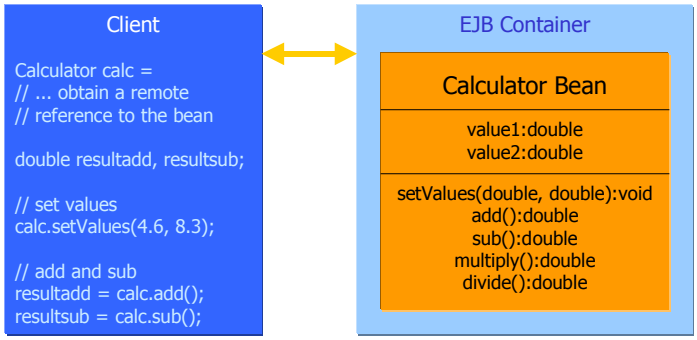

Abb. 2-8: Beispiel eines stateful Session Beans

Bevor der Client in Abb. 2-8 die Rechenfunktionen des Beans ausführen kann, muss er mit `setValues(double,double)` zwei Werte übergeben. Diese werden vom Bean in den Variablen `value1` und `value2` gespeichert. Danach kann der Client die Methoden zum Berechnen der Werte ausführen. Die Werte müssen nicht erneut übergeben werden wie beim stateless Bean oben. Das Bean arbeitet solange mit den übergebenen Werten bis, entweder die Instanz gelöscht oder die Werte geändert werden. Die Werte werden allerdings nicht, wie bei Entity Beans dauerhaft gespeichert, sondern nur solange wie die Instanz des Beans existiert.

2.2.3.3 Entity Beans

Entity Beans bieten eine objekt-orientierte Schnittstelle zu Daten, auf die über JDBC zugegriffen werden. Entity Beans bieten ein Komponentenmodell. Dieses erlaubt Bean-Entwicklern sich auf die Businesslogik des Beans zu konzentrieren, während der Container die Verwaltung der Persistenz, Transaktionen und Zugriffskontrolle übernimmt. Entity Beans implementieren die `javax.ejb.EntityBean` Klasse, welche Methoden definiert, die das Bean zum Interagieren mit dem Container benutzt (vgl. Abb. 2-6).

Es gibt zwei Arten von Entity Beans: *Container-Managed Persistence* (CMP) Beans und *Bean-Managed Persistence* (BMP) Beans. Bei CMP verwaltet der Container die Persistenz des Entity Beans. Der EJB-Container selbst implementiert Methoden, um die Entity Felder mit der Datenbank abzugleichen. Es wird kein Datenbankzugriff im Code der Bean-Klasse definiert. Bei BMP ist das Entity Bean für das Lesen und Schreiben des eigenen Status selbst zuständig. Dies geschieht meist mittels JDBC.

Container-Managed Persistence (CMP)

Container-Managed Persistence (CMP) Beans sind Entity Beans, bei denen der EJB-Container den Abgleich der Daten mit z.B. einer Datenbank übernimmt. Sie lassen sich vom Entwickler am einfachsten erstellen. Die Unterstützung durch den J2EE-Server ist jedoch eine schwierige Aufgabe. Denn der Entwickler muss sich nicht selbst um eine Datenzugriffslogik kümmern. Die Persistenzanforderungen erledigt der J2EE-Server, der die Synchronisation zwischen Bean-Status und Datenbank übernimmt.
Bei der Container-Managed Peristence muss ein J2EE-Server Tools bereitstellen, die es ermöglichen, die container-managed Felder des Beans den entsprechenden Spalten einer bestimmten Tabelle zuzuordnen. Wenn die Felder eines Beans einmal mit einer Datenbanktabelle verbunden sind und das Bean installiert ist, verwaltet der Container das Erstellen, Laden, Ändern und Löschen der Einträge der entsprechenden Tabelle.

Ein Teil der container-managed Felder wird als Primary Key des Beans definiert. Der Primary Key ist ein Index oder Zeiger, auf einen eindeutigen Eintrag in einer Datenbanktabelle. Über das Primary Key Feld werden die Daten eines Beans gefunden. Primärschlüssel, die aus einzelnen Feldern bestehen, werden durch ihre entsprechenden Objekthüllen (object wrappers) dargestellt: z.B. wird eine Variable vom Typ `int` in der Bean-Klasse, einem Bean Client als `java.lang.Integer` Typ erscheinen. Primary Keys, die aus mehreren Feldern bestehen, werden als eigene Klassen definiert.
Um Beziehungen zwischen Beans darstellen zu können wurde mit EJB2.0 die *Container-Managed Relationship* (CMR) eingeführt. Dabei werden *one-to-one*, *one-to-many* und *many-to-many* Beziehungen unterstützt. CMR ist nur zwischen Beans möglich, die das Local-Interface implementieren. Die Integrität der Beziehungen wird vom EJB-Container verwaltet. Sollte beispielsweise in einer one-to-one Beziehung eine Seite der Beziehung geändert werden, ersetzt der Container die alte Beziehung mit der neuen. Zudem ist es möglich eine *cascade delete* Option anzugeben, durch welche der Container beim Löschen eines Objekts alle von diesem Objekt abhängigen Objekte ebenfalls löscht.

Bean-Managed Persistence (BMP)
Das Bean-Managed Persistence Entity Bean, verwaltet das Abgleichen seines Status mit der Datenbank selbst und wird dabei vom Container unterstützt. Das Bean nutzt dazu eine Datenbank API, normalerweise JDBC, um seine Felder zu lesen und in die Datenbank zu schreiben. Der Container benachrichtigt das Bean durch aufruf der entsprechenden Callback-Methoden, wenn ein solcher Abgleich erfolgen soll und verwaltet die Transaktion des Beans automatisch. BMP ermöglicht dem Entwickler Persistenzoperationen auszuführen, die besonders komplex sind oder performant sein müssen. Oder es soll eine Datenquelle genutzt werden, die vom Container nicht unterstützt wird.

Ein Beispiel für ein Entity Bean ist ein Bean, mit dem man Kunden in einer Datenbank verwaltet. Dabei greift der Client auf das Bean zu, das die Kundendaten für den Client aus der Datenbank holt. Der Client greift dabei nur auf die im Bean bereitgestellten Methoden zu und nicht direkt auf die Datenbank. Das Bean definiert u.a. Methoden zum Erstellen neuer und zum Suchen, Ändern und Löschen vorhandener Kunden.

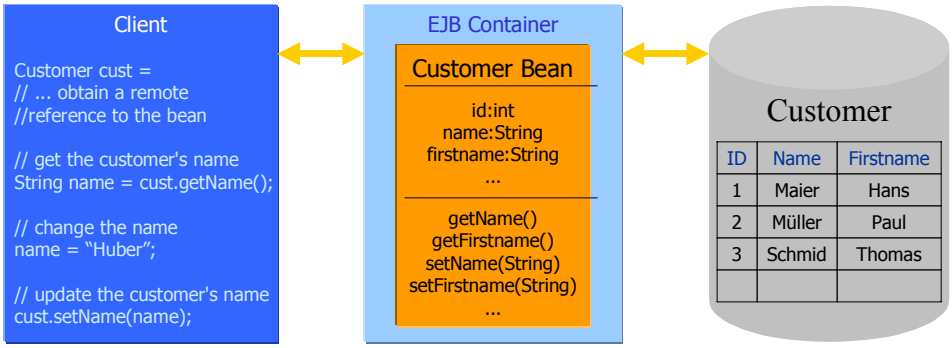

Abb. 2-9: Beispiel eines Entity Bean

Abb. 2-9 zeigt ein `Customer` Bean, welches verschiedene Business Methoden definiert, mit denen die Einträge in der Tabelle Customer geändert werden können. Das Bean hat natürlich noch weitere Methoden, darunter die vorgegebenen Callback-Methoden (siehe Abb. 2-6). Die in Abb. 2-9 beschriebenen Methoden sind spezifisch für das Bean in diesem Beispiel. Ein Client, egal welcher Art (JSP, Servlet, Session Bean, Java Client, usw.), kann auf das Bean zugreifen, nachdem über das Home-Interface eine Referenz auf das Component-Interface erstellt wurde. Nachdem die Referenz erstellt wurde, wird mittels dieser auf die Methoden des

Beans zugegriffen. Hier kann man sich die Namen der Kunden aus der Datenbanktabelle zurückgeben lassen oder diese ändern.

2.2.3.4 Message-Driven Beans

Message-Driven Beans (MDB) erlauben J2EE-Applikationen Nachrichten asynchron zu verarbeiten. Dies wird über einen JMS (Java Message Service) *message listener* ermöglicht, der ähnlich wie ein *event listener* arbeitet. Hierbei werden die Nachrichten in eine Warteschlange gestellt und vom Server nacheinander abgearbeitet. Die Nachrichten können von allen J2EE-Komponenten und anderen JMS Applikationen, aber auch von Systemen, die keine J2EE-Technologie verwenden, gesendet werden. In der EJB2.0 Spezifikation ist es bisher nur möglich, JMS Nachrichten zu versenden. In der näheren Zukunft soll es auch möglich sein andere Arten von Nachrichten verschicken zu können.
Message-Driven Beans haben, im Gegensatz zu den anderen beiden Bean-Typen, weder ein Home noch ein Componenten Interface. Sie bestehen nur aus der Bean-Klasse. Diese implementiert die Interfaces `javax.ejb.MessageDrivenBean` und `javax.jms.MessageListener`.
Da MDBs wie Stateless Session Beans keinen conversional-state haben sind alle Instanzen identisch. Wenn der EJB-Container eine Nachricht für ein Message-Driven Bean erhält, ruft er die `onMessage()`-Methode einer beliebigen freien Instanz dieses Beans auf. Diese Instanz führt daraufhin seine Funktionalität aus. Nachdem das Bearbeiten der Nachricht beendet ist, wird die Instanz vom Container wieder in einen Pool gelegt und steht dadurch der nächsten eintreffenden Nachricht zur Verfügung.

2.2.3.5 Lebenszyklen

Alle drei verschiedenen Bean-Typen haben auch unterschiedlichen Lebenszyklen. Zusätzlich gibt es auch noch einen Unterschied der Lebenszyklen zwischen stateless und stateful Session Beans. Die Lebenszyklen werden dabei vom EJB-Container verwaltet und über die Callback-Methoden beeinflusst.

Stateless Session Beans
Diese Beans haben zwei unterschiedliche Stati. Entweder sie existieren nicht, oder sie sind in einem Status, in dem sie jederzeit Client-Anfragen bearbeiten können. In diesen Pool werden sie gelegt, wenn der Container ein Instanz dieses Beans anlegt. Der Lebenszyklus ist dabei vollkommen unabhängig davon, ob ein Client die `create()` oder `remove()`-Methode des Home-Interfaces aufruft. Denn der Container selbst entscheidet, wann eine neue Instanz angelegt werden soll, oder eine vorhandene gelöscht werden soll.

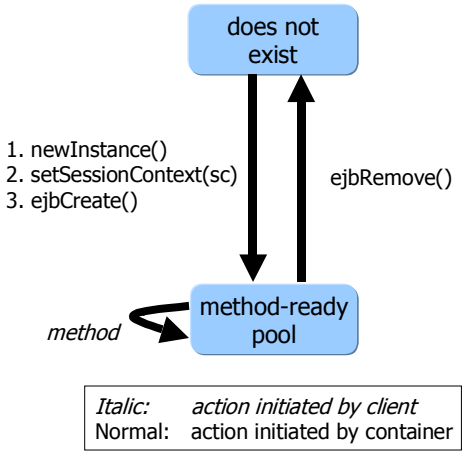

Abb. 2-10: Lebenszyklus von stateless Session Beans nach [EJBS01]

Stateful Session Beans

Dieser Beantyp hat drei Stati. Entweder es existiert nicht oder es ist zum Bearbeiten von Methoden bereit oder es ist passiviert. Da stateful Session Beans immer nur einen Client bedienen hat auch der Aufruf der create()-Methode des Clients direkt die Folge, dass eine Instanz des Beans angelegt wird. Diese Instanz ist dann in einem Status in dem der Client deren Methoden ausführen kann. Wenn diese Instanz eine länger Zeit nicht genutzt wurde kann der Container entscheiden es zu Passivieren. Sollte in diesem Status erneut eine Methode dieser Instanz aufgerufen werden, aktiviert der Container diese Instanz wieder damit der Client damit weiterarbeiten kann.

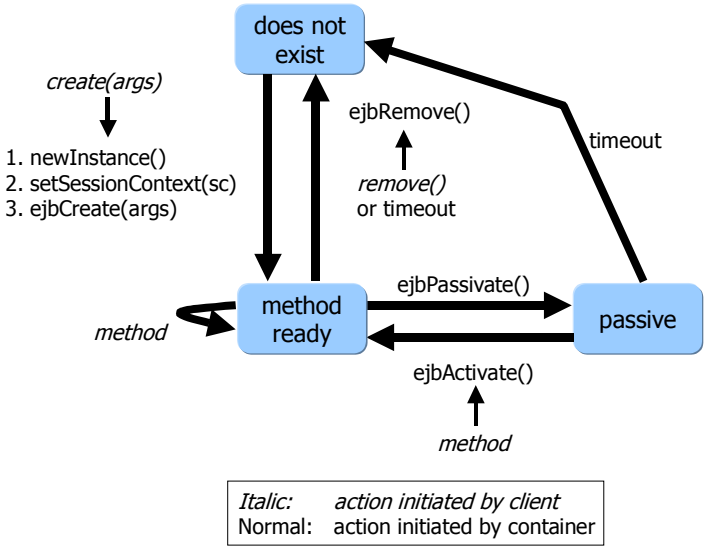

Abb. 2-11: Lebenszyklus von stateful Session Beans nach [EJBS01]

Message-driven Beans

Message-driven Beans haben den selben Lebenszyklus wie stateless Session Beans. Sie sind unabhängig vom Client. Nur der Container entscheidet, wann eine Instanz erstellt werden soll und wann diese gelöscht werden soll.

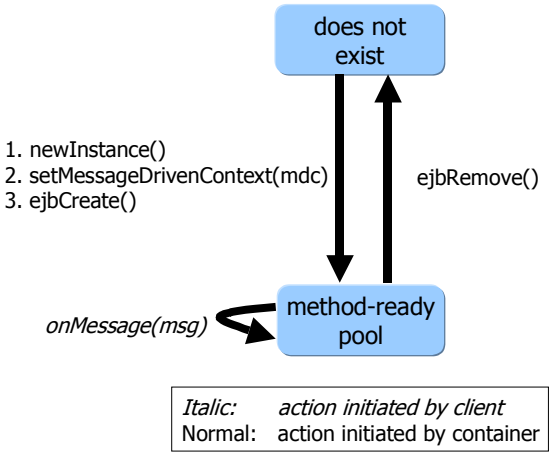

Abb. 2-12: Lebenszyklus von Message-driven Beans nach [EJBS01]

Entity Beans
Der Lebenszyklus von Entity Beans kennt drei Stati. Entweder es existiert nicht oder es ist in einem Pool oder es ist zum Interagieren mit verschieden Clients bereit. Dabei ist der Container dafür verantwortlich, dass für den Client eine Instanz zur Verfügung steht.

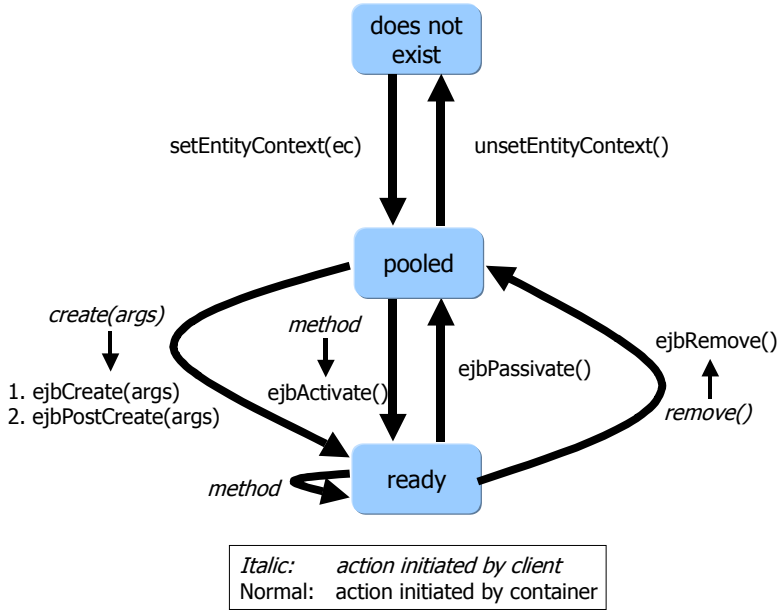

Abb. 2-13: Lebenszyklus von Entity Beans nach [EJBS01], [@SUN02a]

2.2.3.6 Deploying EJBs

Wie bereits erwähnt, verwaltet der Container u.a. Persistenz, Transaktionen und die Zugriffskontrolle automatisch für EJBs. Dies wird alles deskriptiv über den XML *Deployment Descriptor* (DD) gesteuert. Wenn ein Bean in einem Container installiert (deployed) wird, liest dieser den DD, um herauszufinden, wie Transaktionen, Persistenz und Zugriffskontrolle behandelt werden sollen.
Ein Deployment Descriptor hat ein vordefiniertes Format, welches alle EJB konformen Beans nützen müssen und alle EJB konformen Server lesen können müssen. Dieses Format ist in einem XML *Document Type Definition* (DTD) festgelegt. Der DD beschreibt u.a.:

- Den Typ des Beans (Session, Entity oder Message-driven),
- die Namen des Component- und Home-Interfaces und der Bean-Klasse,
- die Transaktionsattribute jeder Methode des Beans,
- die Sicherheitsrollen, die auf die einzelnen Methoden Zugriff haben und
- ob die Persistenz bei Entity Beans automatisch oder im Bean selbst verwaltet wird.

Prog. 2-1 zeigt ein Beispiel eines XML Deployment Descriptors.

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC "-
//Sun Microsystems, Inc.
//DTD Enterprise
JavaBeans 1.1//EN"
 "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>
```

```
        This bean represents a customer
      </description>
      <ejb-name>CustomerBean</ejb-name>
      <home>CustomerHome</home>
      <remote>Customer</remote>
      <ejb-class>CustomerBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>Integer</prim-key-class>
      <reentrant>False</reentrant>

      <cmp-field>
        <field-name>name</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>firstname</field-name>
      </cmp-field>
    </entity>
  </enterprise-beans>
  <assembly-descriptor/>
</ejb-jar>
```

Prog. 2-1: XML Deployment Descriptor [@JGUR02]

Der Aufbau eines DDs muss den Regeln des ejb-jar_1.1.dtd, welches unter http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd zu finden ist, entsprechen. Der oberste Tag `<ejb-jar>` enthält alle weiteren Tags. Der wichtigste dabei ist der `<enterprise-beans>` Tag. Innerhalb diesem wird das jeweilige Bean beschrieben. Dazu gehören alle wichtigen Daten, wie der Name des Beans (`<ejb-name>`), der Home (`<home>` und Component-Interfaces (`<remote>`) und der Bean-Klasse (`<ejb-class>`). Zudem werden hier die Bean-Typ spezifischen Daten wie Persistenzart (`<persistence-type>`), Primary-Klasse (`<prim-key-class>`), ob das Bean Reentrant (`<reentrant>`) sein soll und die Felder die container-managed sind (`<cmp-field>`) für Entity Beans angegeben. Bei Session Beans steht stattdessen etwas zum Session- und Transaktionstyp. Zudem müssen auch eventuell vorhandene Referenzen zu anderen Beans auch innerhalb dieses Tags beschrieben werden.
Innerhalb des `<assembly-descriptor>` Tags können u.a. Definitionen zu Sicherheitsrollen, Zugriffsrechten und Transaktionen stehen.

EJB-fähige Applikationsserver stellen zumeist Tools bereit, mit denen DDs erstellt werden können; dies vereinfacht den Prozess erheblich. Beim Deployment müssen die Component, Home und Bean Class-Dateien (kompilierte Quellcode-Dateien) und der Deployment Descriptor in eine JAR Datei gepackt werden. Der DD muss in der JAR Datei mit dem Namen `META-INF/ejb-jar.xml` gespeichert werden. Diese JAR Datei, ist Anbieter-unabhängig. Sie kann in jedem J2EE-Server installiert werden, der die J2EE-Spezifikationen unterstützt. Wenn ein Bean in einem J2EE-Server installiert wird, wird der XML DD aus der JAR Datei gelesen, um festzustellen, wie das Bean zur Laufzeit behandelt werden soll. Beim installieren werden die Attribute des DDs auf die Umgebung des Containers abgebildet. Der Sicherheitszugriff auf das Sicherheitssystem wird angegeben und das Bean wird dem Naming Service des Containers hinzugefügt. Sobald das Deployment abgeschlossen ist, steht das Bean für Client Anwendungen und für die Verwendung durch andere Beans zur Verfügung.

2.3 Entwicklungsumgebung

Für die Entwicklung von J2EE-Komponenten steht den Entwicklern bei SAP eine ganze Reihe an Tools zur Verfügung. Dabei gibt es Arbeitsgruppen innerhalb SAP, die Tools, die auf dem Markt zu finden sind, genauer analysieren. Diese Teams geben Empfehlungen für die Entwickler heraus. Dabei spielen u.a. die Faktoren Erweiterbarkeit, Anpassbarkeit und Kosten eine Rolle. Daher können Tools auch durch andere Tools ersetzt werden.
Im folgenden werden die Tools beschreiben, die bei der Realisierung des Projektes zum Einsatz kommen.

2.3.1 Perforce

Perforce wurde von der Firma Perforce Software Inc. entwickelt [@PERF02]. Perforce dient der gemeinsamen Nutzung von Dateien unter vielen Anwendern. Software-Konfigurations- und Verwaltungs-Werkzeuge mit Versionskontrolle, Freigabeverwaltung (release management), das Verfolgen von Fehlern, Simultan-Entwicklung, Lebenszyklusverwaltung und das Benachrichtigen bei erfolgten Änderungen sind die Hauptbestandteile des Tools.
Perforce baut auf einer Client/Server Architektur auf, wobei der Server eine UNIX- oder Windows-Plattform benötigt. Die Clients können dagegen auf vielen verschiedenen Plattformen laufen (u.a. UNIX, Linux, Windows, VMS, Mac, BeOS). Die Clients kommunizieren mit dem Server über TCP/IP. Zwei Applikationen erledigen dabei die Hauptaufgaben von Perforce, dies sind *p4d* und *p4*. *p4d* ist das Server-Programm und verwaltet das gemeinsam genutzte (shared) Datei-Repository. Es ermöglicht die Übersicht über die Anwender, Clients und andere Perforce Metadaten. Das *p4* läuft auf jedem Perforce-Client und sendet die Benutzeranfragen zur Bearbeitung zum *p4d* Serverprogramm.
Die Anwender können Dateien und Verzeichnisse auf ihren eigenen PCs erstellen und diese über Perforce Kommandos an das Datei-Repository, das als *depot* bezeichnet wird, übertragen. Sie können ebenso Dateien aus dem Repository auf den eigenen Rechner, als *client workspace* bezeichnet, übertragen und diese dann lesen, editieren und wieder auf den Server übertragen. Auf diesem wird dann eine neue Datei mit dieser Überarbeitung angelegt. Dies geschieht, damit die alten Dateien nicht überschrieben werden und verloren gehen, da sie immer noch zugriffsbereit sein sollen. Jedes Mal wenn Dateien zum *depot* gesendet werden, wird eine *changelist* erstellt. In dieser Liste ist zu jeder Datei beschrieben, wie sie behandelt werden soll (z.B. anlegen, ändern, löschen). Soll eine Datei gelöscht werden, geschieht dies nicht physisch, sondern sie wird nur als gelöscht markiert. Damit geht sie nicht verloren. Die *changelist* wird atomar behandelt, d.h. nur wenn alle Änderungen erfolgreich verlaufen, werden auch alle durchgeführt. Sollte eine Änderung fehlschlagen wird keine Änderung ausgeführt.
Es ist in Peforce auch möglich, dass mehrere Personen gleichzeitig an einer Datei arbeiten. Hierbei fragt Perforce dann nach, wie mit den verschiedenen Versionen verfahren werden soll: Z.B. ob alle Änderungen zusammengeführt werden oder ob nur eine davon gültig sein soll und die anderen verworfen werden sollen.
Perforce hat auch Schutzmechanismen, die nicht-autorisierte oder versehentliche Zugriffe auf ein Depot verhindern. Hierbei können Berechtigungen Benutzer- und IP-Adressen-basiert vergeben werden.

Die Benutzeroberfläche des Client-Programms ist in drei Bereiche geteilt. Auf der linken Seite werden alle Verzeichnisse und Dateien des Depots, in dem man sich gerade befindet, angezeigt. Im rechten Bereich kann man sich alle Dateien anzeigen lassen, die zur Änderung freigegeben sind, sowohl die eigenen als auch die aller anderen Benutzer. Im unteren Bereich des Fensters werden Statusmeldungen ausgegeben, die der Client vom Server erhält.

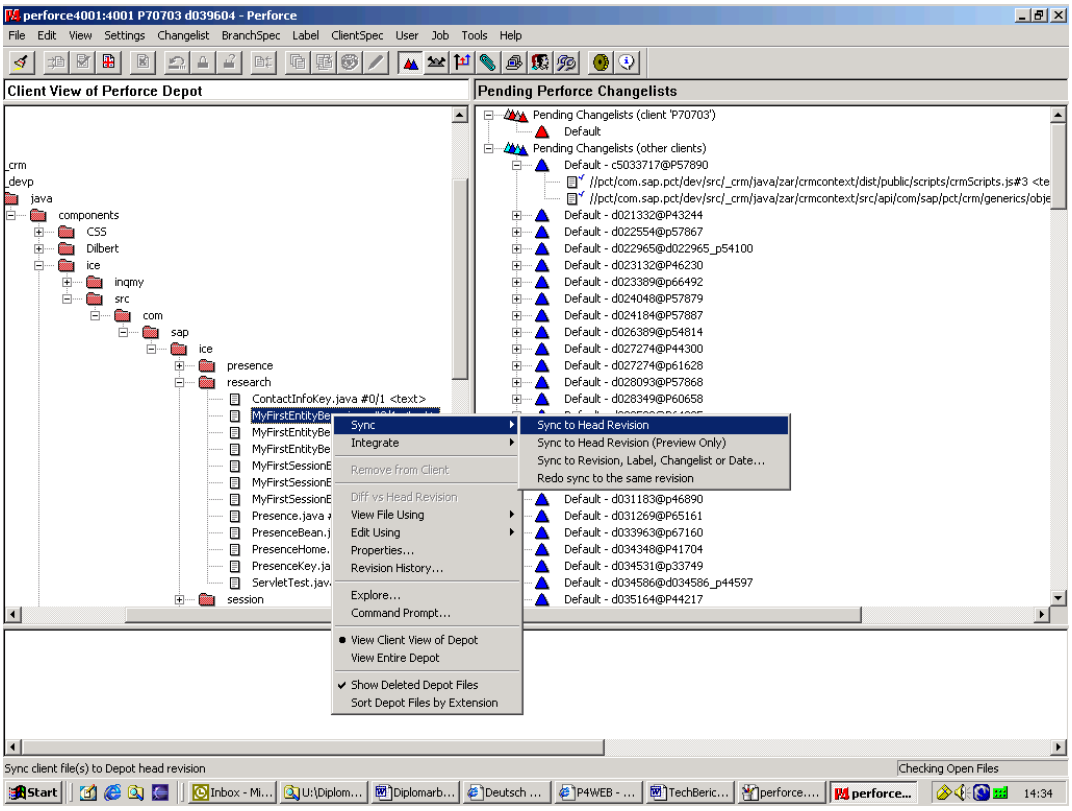

Abb. 2-14: Benutzeroberfläche Perforce

Mit der rechten Maustaste kann man eine Datei oder ein Verzeichnis auswählen und über ein Popup-Menü bestimmen wie mit dem ausgewählten Objekt verfahren werden soll: z.B. für einen Änderung sperren, oder als schreibgeschützte Datei auf den eigenen Rechner übertragen.
In der SAP-Entwicklung gibt es momentan 46 offizielle Perforce-Server auf denen 122 Depots eingerichtet sind. Diese decken den grossteil der nicht ABAP-Entwicklungsaktivitäten bei SAP ab [@SAP02f].

2.3.2 Ant

Ant ist ein Java-basiertes *build*-Tool. Es wurde unter dem Mantel der Apache Software Foundation, innerhalb des *The Jakarta Project* [@AANT02], entwickelt und wird dort auch weiterentwickelt. Ant hat Ähnlichkeit mit *Make*, wie man es von UNIX kennt und kann kompilierte Dateien mit einem Zeitstempel versehen. Dadurch kann es erkennen, ob Dateien geändert wurden und neu kompiliert werden müssen. Es kann somit den Kompilierungsvorgang verkürzen, da es inkrementell agiert. Der Sourcecode ist frei verfügbar und darf unter Beachtung der Lizenzbedingungen angepasst werden.
Ant wird über das Kommando `ant` in der Kommandobox aufgerufen. Das *target* Projekt (s.u.) wird als Aufrufparameter übergeben. Verschiedene weitere Argumente sind verfügbar. Ant sucht beim Aufruf standardmässig nach der Datei `build.xml`. Will der User eine andere Konfigurationsdatei angeben, ist dies über das Argument `-buildfile <file>` möglich. Wenn das Argument `-find` angegeben wird, sucht Ant alle übergeordneten Verzeichnisse ab, bis es entweder die erste `build.xml` Datei gefunden hat oder am Wurzelverzeichnis angekommen ist.
Die Konfigurationsdateien werden im XML-Format erstellt, wobei jede einen `<project>` Tag, das alle weiteren Tags einschliesst, enthält. Hierbei kann ein Name und das Basisverzeichnis angegeben werden und es muss ein default target angegeben werden. Dieses wird ausgeführt, wenn beim Aufruf dieser Datei vom Benutzer kein auszuführendes target

übergegeben wird. In den `<target>` Tags wird angegeben, welche Tasks ausgeführt werden sollen. Hier können alle Tasks, die Ant zur Verfügung stellt (z.B. java, javac, jar, mkdir, unzip, usw.) ausgeführt werden. Jeder Task hat dabei seine eigenen Parameter. Zudem können auch selbst neue Tasks entwickelt werden, die dann in Ant mit eingebunden werden können. Jedes Project kann verschiedene *Properties* haben, welche entweder in der Konfigurationsdatei gesetzt werden oder auch beim Aufruf von Ant angegeben werden können. Properties haben einen Namen und einen Wert. Auf diese Werte kann mittels `${<prop_name>}` zugegriffen werden.
Ein Beispiel für eine XML-Konfigurationsdatei ist Prog. 2-2.

```
<project name="MyProject" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="src" value="."/>
  <property name="build" value="build"/>
  <property name="dist"  value="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Prog. 2-2: Ant XML Konfigurationsdatei [@AANT02]

2.3.3 Eclipse

Für die Quellcode-Entwicklung wird das Tool Eclipse [@ECLI02] eingesetzt. Es ist ein Open Source Projekt, das von führenden Softwarefirmen (IBM, Borland, Merant, QNX Software Systems, Rational Software, RedHat, SuSE, TogetherSoft und WebGain) ins Leben gerufen wurde. Das Ziel ist es, eine robuste, mit allen Features ausgestattete, qualitativ hochwertige

Plattform für die Entwicklung von hoch integrierten Tools zu entwickeln. Es besteht aus drei Teilprojekten: Plattform, JDT (Java development tools) und PDE (Plugin development environment).
Die Eclipse Plattform ist eine offene und erweiterbare IDE (Integrated Development Environment) für unterschiedliche Programmiersprachen. Sie bietet Bausteine und eine Basis für die Entwicklung von integrierten Software-Entwicklungstools. Sie definiert mehrere Frameworks und allgemeine Dienste, die benötigt werden, um eine umfassende Tool-Integrationsplattform zu unterstützen. Diese Dienste und Frameworks stellen die allgemein gebräuchlichen Mittel dar, die von den meisten Entwicklern benötigt werden. Dies schliesst eine Standard-Benutzeroberfläche und ein Projektmodell zum Verwalten von Quellen ein. Desweiteren bietet es portierbare native *widgets* und Benutzeroberflächen-Bibliotheken, automatische Verwaltung von Compilern und Buildern, eine sprachenunabhängige Debug-Infrastruktur und eine Infrastruktur für verteiltes Multi-User Resourcen- und Versionsmanagement.
Das JDT Projekt liefert die Tool plug-ins, u.a. die Java-Entwicklungsumgebung. Es fügt sowohl eine Java-Projekt-Umgebung und Java-Perspektiven zur Eclipse Workbench hinzu, als auch einige Views, Editoren, Wizards, Builders und Quellcode-Verwaltungs-Tools. Zusätzlich zu der Java Perspektive gibt es inzwischen auch eine C/C++ Perspektive. Diese ermöglicht das Entwickeln von C/C++ Programmen.
Das PDE Projekt bietet eine Reihe von Views und Editoren, welche das Erzeugen von Plug-ins für Eclipse vereinfachen. Mittels PDE kann man u.a. eine Plug-in Beschreibungsdatei erstellen, die die Runtime und weitere Plug-ins spezifizieren und Schnittstellen definieren.
Alle Komponenten, die von den drei beschriebenen Projekten entwickelt und getestet wurden, werden im Eclipse SDK (Software Development Kit), das im Internet steht, zusammengefügt.

Die grössten Vorteile von Eclipse liegen darin, dass es ohne Probleme erweiterbar ist, was für die SAP ein ganz wichtiger Punkt ist. Zudem kommt hinzu, dass es eine grosse Community gibt, hinter der grosse Softwarefirmen stehen, die Eclipse ständig weiterentwickelt und neue Plug-ins erstellt.

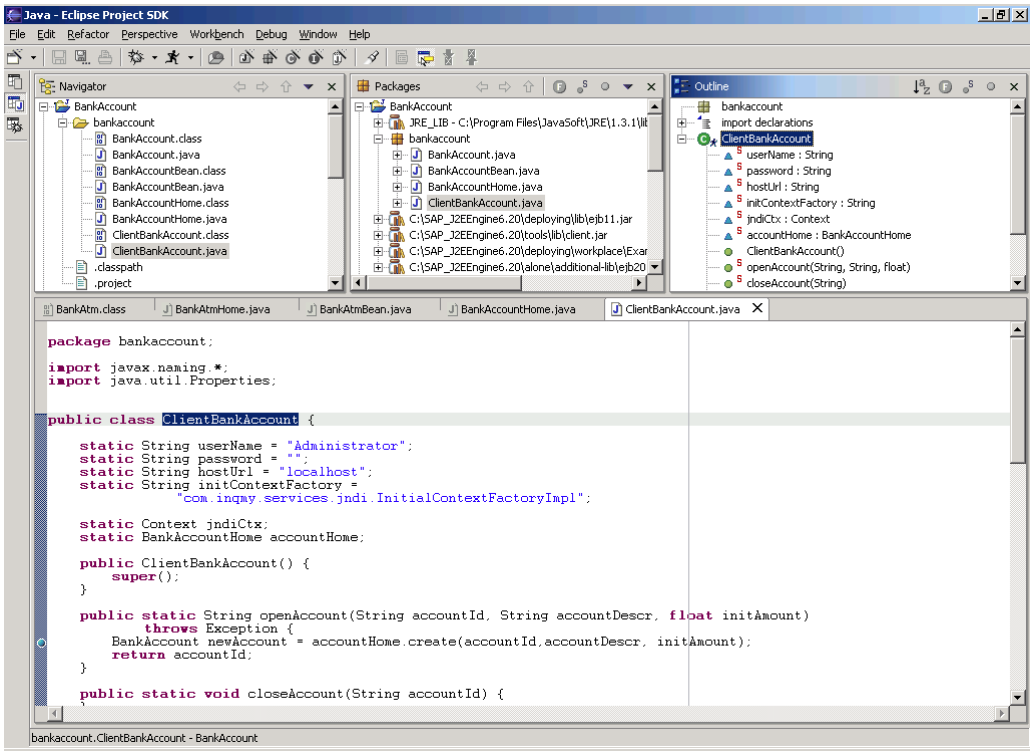

Abb. 2-15: Eclipse SDK

Seit Mai 2002 wird Eclipse offiziell bei der SAP als neues Entwicklungstool eingesetzt und wird JBuilder mehr und mehr ablösen. Grundlage für die Entscheidung war, dass Eclipse zur Tool-Integration wesentlich bessere Voraussetzungen bietet als JBuilder. Darüberhinaus ist als Open Source-Produkt Eclipse frei von Lizenzkosten. Dies ermöglicht SAP im Support-Fall uneingeschränkten Zugriff auf die Quellen, sowie eine Beteiligung an der Eclipse-Weiterentwicklung.

2.3.4 SAP J2EE-Engine

Die SAP J2EE-Engine ist ein J2EE-Applikations- und Webserver, welcher alle in der J2EE-Spezifikation aufgelisteten Technologien unterstützt. Der Server wurde von der SAP eigenen Tochterfirma InQmy Technologies Ltd. entwickelt. Er trug bis zur Eingliederung dieser Firma in die SAP AG auch den Namen InQmy. Die Engine kommt auch beim Web Application Server (Web AS) zum Einsatz. Die folgenden Aussagen über die SAP J2EE-Engine sind aus [@SAP02a] entnommen.

Die SAP J2EE-Engine ist momentan in der Version 6.20 vorhanden. In dieser Version können nur die in EJB1.1 spezifizierten Beantypen verwendet werden. Sie kann sowohl als stand-alone als auch als Cluster Version installiert werden. Die stand-alone Version ist eine Entwicklungsversion der J2EE-Engine, welche weniger Resourcen nützt als die Cluster-Version, aber die gleiche Funktionalität bereitstellt. Sie verbindet einen Dispatcher und einen Server in einem Prozess. Die Kommunikation zwischen den beiden Komponenten ist für die lokale Kommunikation optimiert. Bei der Stand-alone Version laufen Server und Dispatcher in der gleichen Java Virtual Maschine (JVM).

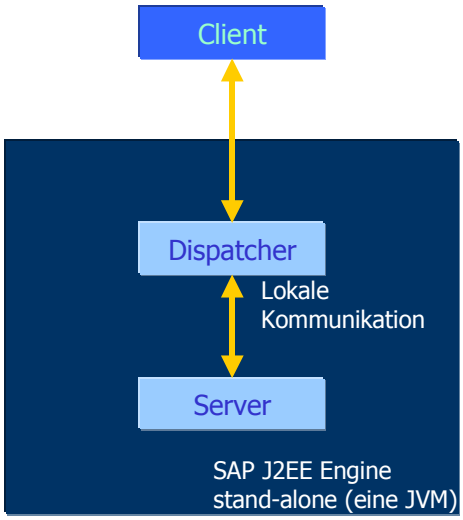

Abb. 2-16: Stand-alone Server nach [ZENZ02]

Der Dispatcher nimmt die Anfragen vom Client entgegen. Er leitet diese an den Server weiter, welcher die Applikationslogik ausführt. Das Ergebnis wird an den Dispatcher zurückliefert und dieser leitet es dann zum richtigen Client weiter.

Bei der Cluster Version hingegen können mehrere Dispatcher und Server auf den gleichen oder verschiedenen Rechnern eingesetzt werden. Hierbei leiten die Dispatcher die Client-Anfragen an den Server mit der geringsten Auslastung weiter. Jeder einzelne Server und Dispatcher läuft in einer eigenen Java Virtual Maschine.

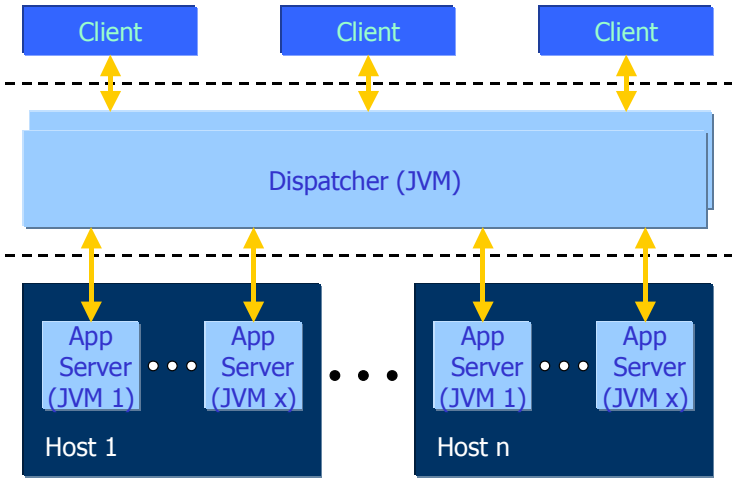

Abb. 2-17: Cluster Server nach [ZENZ02]

Dadurch, dass die SAP J2EE-Engine cluster-fähig ist, entstehen eine ganze Reihe von Vorteilen, die für Enterprisesysteme ausschlaggebend sind. Diese sind u.a.:

- Erhöhte Verfügbarkeit und Zuverlässigkeit: Der Ausfall einer Applikation oder eines Servers kann erkannt werden. Es wird auf einen anderen Server umgeschaltet, wodurch die Systemunterbrechungen minimiert werden.
- Verbesserte Performance: Mehrere Anfragen können nebenläufig verarbeitet werden, was kürzere Antwortzeiten ermöglicht.
- Höhere Skalierbarkeit: Es ist dynamische Lastverteilung (load balancing) und Skalierung über viele Server realisiert.
- Einfachere Verwaltung: Administratoren können einfach den Status aller Cluster überwachen und alle Cluster von einer zentralen Stelle aus verwalten.

2.3.4.1 Logische Entitäten

Die SAP J2EE-Engine besteht aus drei Arten logischer Entitäten: *managers*, *core services* und *services*. Sowohl *managers* als auch *core services* gehören zum Kernsystem.

Managers unterstützen die Verwaltung der Java Virtual Maschine über z.B. Thread Management und Speicher Management. Jeder Knoten (Server, Dispatcher) hat seine eigenen *managers*. Es ist möglich, neue *managers* hinzuzufügen oder die Schnittstellen von existierenden zu erweitern. Es gibt auch die Möglichkeit die Funktionalität des Kernsystem zu erweitern.

Die *Core services* enthalten die Basisfunktionalität der Engine und werden für das Cluster angeboten. Einige laufen nur auf dem Server oder nur im Dispatcher. *Core services* bieten für die Administration im Cluster und ihren Knoten Lastverteilung. Sie verwalten Sicherheits- und Kommunikationsfunktionen, das Deployment von neuen Services, das Deployment und die Aktualisierung von Komponenten, Event Logging usw.

Services erweitern die Funktionalität der SAP J2EE-Engine. Sie sind verteilte Systeme, welche nicht zum Kernsystem gehören. Sollte das Starten eines *service* fehlschlagen, so wird das Starten des Gesamtsystems nicht beeinflusst, es steht nur diese spezielle Funktionalität nicht zur Verfügung.

2.3.4.2 Logische Schichten

Die SAP J2EE-Engine realisiert eine Architektur aus stark vereinfacht drei Schichten: Applikations-, Verbindungs- und Transportschicht. Client Applikationen können ebenfalls in diese drei Schichten unterteilt werden.

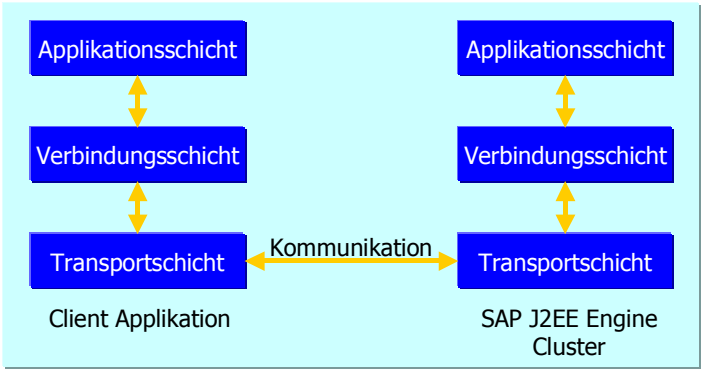

Abb. 2-18: Logische Schichten nach [@SAP02a]

Die Transportschicht ermöglicht den Datentransfer zwischen Client und Cluster. Die Daten existieren an der Transportschicht als eine Sequenz von Bytes. Diese können von Verschlüsselungsalgorithmen, Netzwerkprotokollen, Streams und Sockets bearbeitet werden.
Die Verbindungsschicht ist eine Zwischenschicht, welche Sessions und Remote Kommunikation unterstützt. HTTP wird auf dieser Schicht unterstützt.
Die Applikationsschicht implementiert die Logik der Benutzer-Applikation. Hier werden EJBs, JSPs, Servlets usw. gehalten.

2.3.4.3 Eigenschaften

Der Server unterstützt komponenten-basierte Enterprise Technologien. Er erlaubt das effiziente Entwickeln und Installieren von zuverlässigen, sicheren, skalierbaren und verwaltbaren Applikationen. Er ist komplett in Java geschrieben, was Übertragbarkeit und Multi-Plattform-Support erlaubt. Er ist zur Java 2 Platform, Enterprise Edition (J2EE) konform und implementiert Enterprise JavaBeans, Servlets, JSPs, JNDI, JMS, Java Mail wie es im Standard spezifiziert ist. Zudem unterstützt er eine Reihe von Industriestandard-Protokollen und bietet Erweiterungen, die für SAP eigene Bedürfnisse konzipiert sind. Die SAP J2EE-Engine 6.20 bietet folgende Eigentschaften: (vgl. [@SAP02a])

- Skalierbarkeit,
- Verfügbarkeit und Zuverlässigkeit, d.h. durchgehender Zugriff auf das Gesamtsystem 24 Stunden am Tag, 7 Tage die Woche,
- Transaktionssicherheit und Abhörsicherheit bei jeder Kommunikation,
- Caching, Multithreading und Multiprozessor-Fähigkeit, welches das Bearbeiten einer grossen Anzahl von gleichzeitigen Anfragen bietet, um eine optimale Performance während Spitzenbelastungen zu erreichen,
- Dynamische Lastverteilung, um kürzere Antwortzeiten und eine bessere Performance zu erreichen,
- Datenreplikation, um Ausfallsicherheit zu erreichen,
- Web-Seiten- und Komponenten-Verteilung, für grösstmögliche Skalierbarkeit und Zuverlässigkeit,
- Remote Verwaltung von vielen Cluster-Knoten, um die Konfiguration und Administration des Systems zu erleichern,

- eine Persistenz Engine, unterstützt Transaktionen, um Sicherheit und Datenintegrität zur Verfügung zu stellen,
- Event logging für System- und Applikationsereignisse, wie Ausfälle, Client errors, Informationsereignisse, usw.,
- stetiges Überwachen der Performance von Cluster Knoten und von SAP J2EE-Engine kritischen Modulen.

2.3.4.4 Funktionalität

Die Funktionalität der J2EE-Engine ist in verschiedene Dienste aufgeteilt. Es werden hier nur die Dienste aufgeführt, die die Schlüsselfunktionalitäten bieten.

Naming Service
Clients erhalten Zugriff auf Services der SAP J2EE-Engine durch einen JNDI-konformen Naming Service. Es ist ein verteiltes, skalierbares und zuverlässiges System, welches dem Client als eine Einheit erscheint. Der SAP J2EE-Engine Naming Service sieht die Verwaltung und Speicherung von serialisierbaren Objekten in einer Baumstruktur vor. Nicht serialisierbare Objekte, welche nur von derselben JVM zugänglich sind, sind ebenso vorgesehen. Der Naming Service berücksichtigt die Speicherung und Verwaltung von Cluster Remote Objekten. Diese sind nicht serialisierbar und können zu anderen Knoten mit Hilfe des P4 Protokolls übertragen werden. Das P4 Protokoll sieht die Generierung der client-side stubs eines jeden Remote Objekts vor. Diese stubs berücksichtigen die Remote Suche im Naming System.

P4 Service
Dies ist ein objekt-orientiertes Protokoll für die Remote Übertragung von Objekten. Es besteht aus zwei Untermodulen: der RMI/P4 Client Implementierung und dem P4 Service. Der Service ist verteilt im Cluster, und unterstützt den Transfer von Benutzer- und Transaktionsnummern. Er verwaltet die Übertragung von Remote Objekten auf der niedrigsten Transportschicht.

Enterprise JavaBeans Service
Die SAP J2EE-Engine 6.20 unterstützt nur zwei Beantypen: Session, beides stateful und stateless und Entity Beans. Alle Enterprise JavaBeans werden vom EJB Service verwaltet.
Session EJBs laufen lokal und unabhängig voneinander in der SAP J2EE-Engine. Sie sind nicht verteilt und kommunizieren nicht mit anderen Session EJBs auf anderen Knoten im Cluster. Die Session Beans werden vom EJB Session Container verwaltet. Dieser bietet eine skalierbare Laufzeitumgebung für gleichzeitiges Ausführen von mehreren Anfragen an ein Session Objekt.
Entity EJBs werden vom Entity EJB Container verwaltet. Clients können jedes Bean mit dem Primary Key der Datenbank, im SAP J2EE-Engine Entity Beans Container, lokalisieren. Für jedes Bean existiert ein Container auf jedem Server im Cluster. Der EJB Container stellt dynamisch das Abgleichen zwischen einem Cache von aktiven Instanzen und einem Pool von passiven Instanzen her. Die Instanzen werden entsprechend der Speichermanager-Lastverteilung passiviert oder abgelegt. Die Container verwalten den Persistenzstatus von *container-managed* Entity Beans. Sie berücksichtigen eine einzige, aktive Instanz je Primary Key. Der Container verwaltet hierbei parallele Zugriffe. Er prüft nach dem Ausführen einer Businessmethode, ob die Transaktion Daten geändert hat. Zudem erlaubt er keine zweite Transaktion, die dieselben Daten ändern möchte, um *dirty writing* zu verhindern. Der Entity EJB Container ermöglicht das Erstellen von mehreren Instanzen eines *bean-managed* EJBs. Jede Instanz entspricht einem bestimmten Primary Key und einer bestimmten Transaktion.

Servlets und JavaServer Pages Service
Die Servlets und JSPs werden in der SAP J2EE-Engine von dem *Servlets_jsp Service* verwaltet. Der Service arbeitet als Container von Webapplikationen und Servlets und stellt eine JSPEngine bereit. Er wird nicht direkt von den Clients benützt, sondern über den SAP J2EE-Engine HTTP Server. Dieser bietet zusammen mit dem *Servlets_jsp Service* Clients die Möglichkeit mit dynamischen Web-Applikationen zu arbeiten. In einem SAP J2EE-Engine Cluster ist es nicht notwendig, ein Servlet oder JSP auf mehreren Server Knoten zu installieren. Dies wird vom *Servlets_jsp Service* im Cluster automatisch gemacht, um die Bearbeitungszeit von Anfragen zu verkürzen und eine grössere Skalierbarkeit zu erreichen.

Monitor Service
Die SAP J2EE-Engine bietet ein integriertes Aufzeichnungs- und Überwachungssystem. Dieses wird vom *Monitor Service* benützt, um dynamisch Informationen über den Systemstatus bereitzustellen. Je nach dem, welcher Modus gewählt wird, können die Daten über die SAP J2EE-Engine, in Microsoft Excel oder ähnlichen Programmen angezeigt werden. Der *Monitor Service* bietet Informationen über das gesamte Cluster, einen einzelnen Knoten oder ein Modul.

2.3.4.5 Administrator Tool

Im Administrator Tool der SAP J2EE-Engine kann man alle Eigenschaften von allen *managers* und *services* des Servers oder Clusters anzeigen und ändern. Die beiden wichtigsten *services* beim Installieren von Applikationen sind die *services dbpool* und *deploy* im Server-Knoten.
Im *dbpool* können verschiedene Datenbankpools angelegt werden. Über diese können die Entity Beans auf die tatsächlichen Datenbanken und Tabellen zugreifen. Hier wird definiert, wo die Datenbank sich befindet und welcher Benutzername und Passwort für den Zugriff gültig sind. Zudem wird der Name des Pools definiert, der beim Deployment von Entity Beans angegeben werden muss, und der spezielle Datenbanktreiber der Datenbank. Beim Auswählen des Datenbanktreibers und dem Hinzufügen der entsprechenden Treiber Archiv-Dateien, ist es bis jetzt allerdings nur möglich immer eine Datei nach der anderen hinzuzufügen. Das Tool kann sich zudem den Pfad nicht merken, aus welchem die bisherigen Dateien hinzugefügt wurden. Dies ist ein bischen umständlich, besonders da das Navigieren im Dateisystem nicht immer ganz so funktioniert, wie man das von Windows Software kennt.

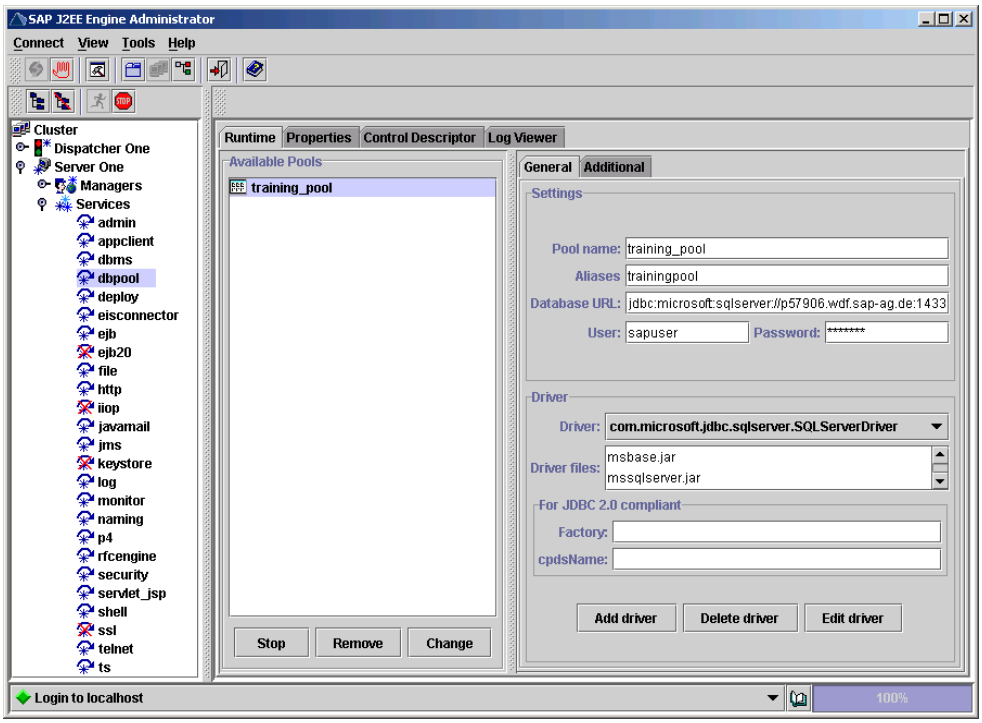

Abb. 2-19: Screenshot Admintool SAP J2EE-Engine, dbpool service

Der *deploy* Service zeigt alle Applikationen, Beans, Servlets und JSPs an, die in dem Server installiert sind.

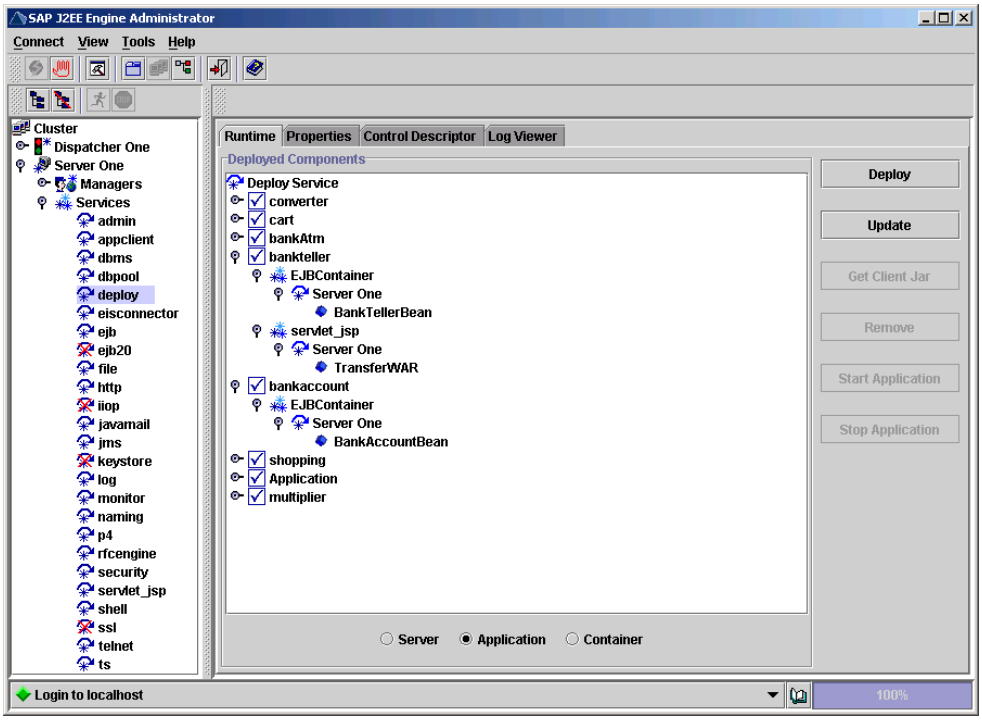

Abb. 2-20: Screenshot Admintool SAP J2EE-Engine, deploy service

2.3.4.6 Deployment-Tool

Das Deployment-Tool wurde entwickelt, um das Installieren von Enterprise Applikationen zu vereinfachen. Das Deployment-Tool erzeugt dabei die `.jar` und `.war` Dateien und aus diesen Dateien heraus auch eine `.ear` Datei. Dabei werden alle wichtigen Beschreibungs-Dateien, wie z.B. der *Deployment Descriptor*, erzeugt. Um einen Einblick über den Installations-Ablauf zu geben, wird im folgenden das Deployment einer Anwendung, die aus einem Session Bean, einem Entity Bean und einem Servlet besteht, beschreiben.

- Als erstes muss im Deployment-Tool ein neues Projekt mit dem Namen der Applikation angelegt werden.
- Mit `Datei➔ Options` wird der Classpath zu den `.class` Dateien gesetzt. Hierbei muss beachtet werden, dass man nicht in die Package-Verzeichnisstruktur hinein navigiert, sondern das Wurzelverzeichnis, unter dem alle Verzeichnisse und Klassen sind, angibt.
- Im Reiter `J2EEComponents` über das Menü `J2EEComponents➔ Add EJB Group` kann eine neue EJB Gruppe hinzugefügt werden, die als `.jar` Datei gespeichert wird.

Session Beans:

- Über `J2EEComponents➔ Add EJB` kann ein neues Bean hinzugefügt werden. Hier lässt sich nun der Name des Beans auswählen, über den man dann auch vom Client oder anderen Beans aus, das Bean finden kann. Dabei kann ausgewählt werden, ob man ein Entity oder ein Session Bean hinzufügen möchte. Wenn ein Session Bean hinzugefügt wird, muss das Component- und Home-Interface und die Bean-Klasse angegeben werden, wobei der Name des Packages, in dem das Bean sich befindet, mit angegeben werden muss. Dann muss angeben werden, welchen `Demarcation-` (Bean, Container) und `Management-Type` (stateful, stateless) das Bean hat.
- Um nun eine Referenz zu einem anderen Bean anzulegen, muss man auf `References` gehen, um dort den `Reference Name` des Beans anzugeben mit dem man kommunizieren will. Dies ist der Name über den man das Bean finden kann. Dann wird der Bean Typ (Session, Entity) des Referenz-Beans ausgewählt, der Name des Home- und Component-Interfaces, wieder mit Packagestruktur und der `Reference Link`, welcher der Name des Beans ist, mit dem es installiert wurde.

Entity Beans:

- Das Anlegen eines Entity Beans erfolgt gleichermassen, wie bei einem Session Bean. Es kann nur nicht der Demarcation- und Management-Type angegeben werden. Anstatt dessen muss die Primary Key Klasse angegeben werden. Für Java Standardtypen ist dies java.lang.Long oder java.lang.Integer usw. Wenn eine eigene Primary Key Klasse erzeugt wurde, muss diese angegeben werden, ebenfalls wieder mit dem Package-Namen. Als letztes muss noch das Primary Feld angegeben werden, wobei der Package-Name hier nicht benötigt wird. Auf diesem Fenster kann jetzt noch ausgewählt werden, ob das Bean `Reentrant` sein soll und welches `Persistenz-Management` genutzt werden soll (`bean-` oder `container-managed`).
- Sofern dieses Bean mit anderen Entity Beans kommunizieren soll, kann ebenfalls unter `References` eine Referenz zu einem anderen EJB angegeben werden. Genaues s.o.
- Im Reiter `Storage` kann man dann noch festlegen, welche der Membervariablen der Bean gespeichert werden sollen.

Servlets und JSPs:

- Um ein Servlet oder JSP hinzuzufügen, muss als erstes über `J2EEComponents`➔ `Add Web` ein Web Archive, `.war` hinzugefügt werden.
- Als nächstes kann man im Reiter `Files` ein oder mehrere Verzeichnisse hinzufügen, in dem sich die `.class` Dateien der Servlets oder JSPs befinden, die man hinzufügen möchte, allerdings muss auch wieder auf die Package-Struktur achtgegeben werden.
- Nach dem diese Verzeichnisse hinzugefügt wurden, kann man über `J2EEComponents`➔ `Add Servlet/Jsp`➔ `Add from files` sich alle Servlets und JSPs anzeigen lassen, die in den Verzeichnissen gefunden wurden. Hier kann man auswählen, welche man tatsächlich hinzufügen möchte.

Erstellen der Archive:

- Nachdem nun alle EJBs und Servlets/JSPs hinzugefügt wurden, werden über `J2EEComponents`➔ `Make All Archives` die `.jar` und `.war` Dateien erzeugt.

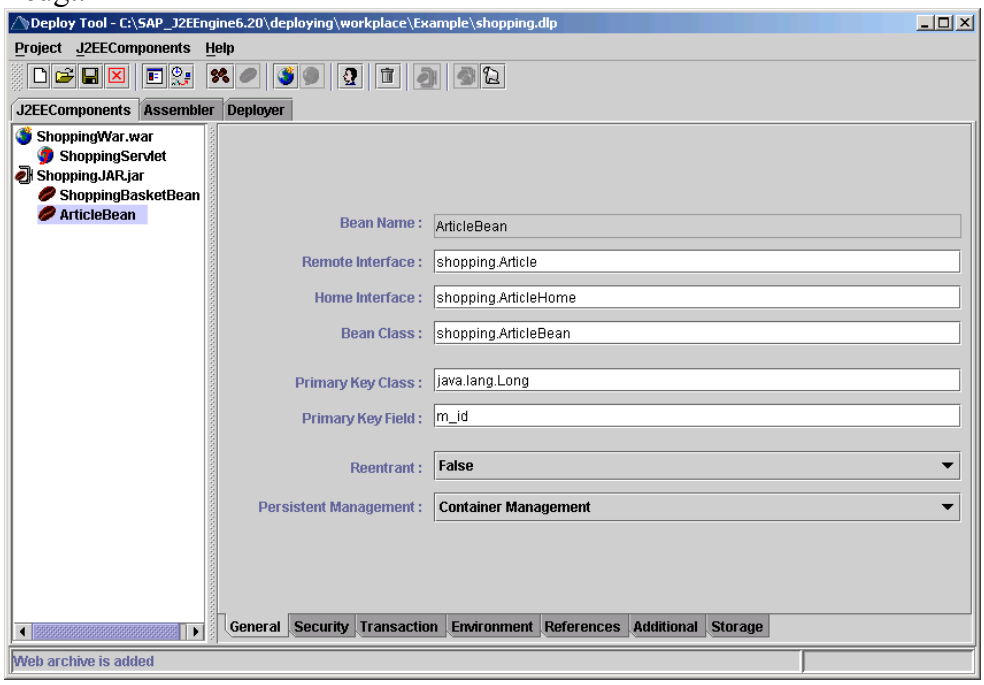

Abb. 2-21: Deploy Tool J2EE-Engine, J2EEComponents

- Nun wechselt man auf den Reiter `Assembler` in dem die `.ear` Datei erzeugt wird.
- Wenn dies erfolgreich geschehen ist, kann man zum Reiter `Deployer` wechseln. Hier muss man noch, bevor das komplette Projekt installiert werden kann, bei allen Entity Beans unter `Storage` den Poolnamen und die Tabelle angeben, auf die sich das Entity Bean bezieht.

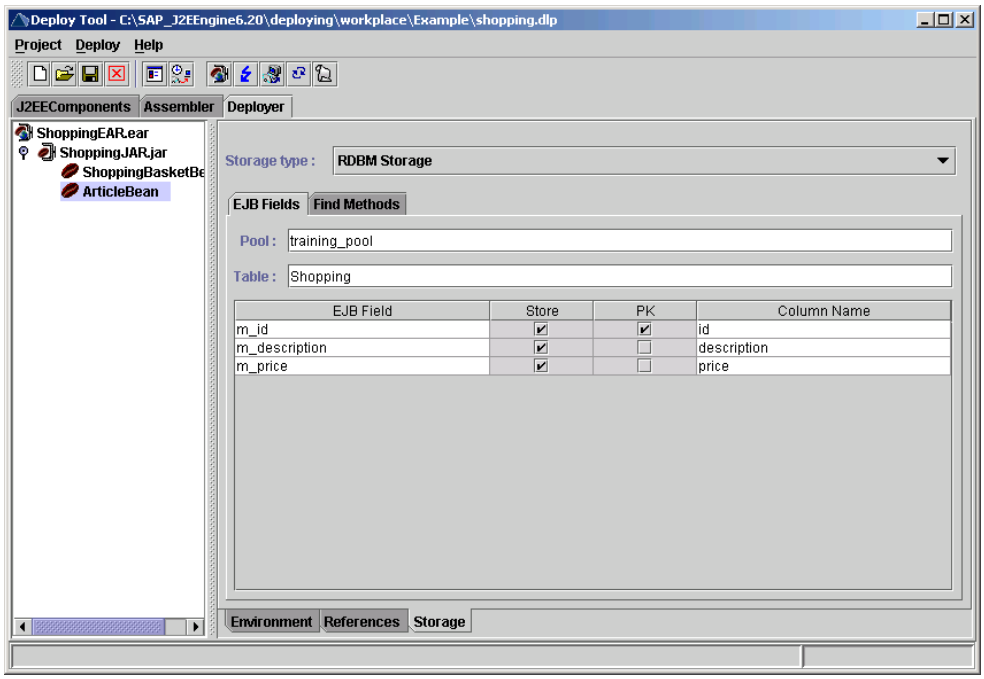

Abb. 2-22: Deploy Tool J2EE-Engine, Deployer

- Ist dies erfolgt, kann über `J2EEComponents`➔ `Deploy`➔ `Deployment`➔ `Deploy Ear` die Applikation installiert werden.

Wenn man ein Servlet in einem Browser aufrufen möchte, muss man dies über `URLs` von folgender Struktur machen, http://<servername>[:<port>]/<WebApp_name>/servlet/<servlet_name>.
Das Deploy Tool ist allerdings noch nicht ganz ausgereift. Manchmal werden geänderte `.class` Dateien, beim erneuten Erstellen der Archive und beim Deployment nicht aktualisiert und übernommen. Um trotzdem die geänderten Dateien einzubeziehen, muss das Projekt geschlossen und neu geöffnet werden. Das Deploy Tool hat auch eine Funktion, die alle Archive automatisch nacheinander erstellt und das Deployment in einem Zug durchführt.

Die komplette J2EE-Engine wird ständig weiterentwickelt. Es ist möglich jedes Problem (Bug) den zuständigen Mitarbeitern zu melden. Dadurch kann man davon ausgehen, dass vorhandene Probleme in neuen Versionen nicht mehr auftauchen.

2.3.5 SAP Java Connector (JCo)

SAP JCo ist eine Java-Programmierschnittstelle, über die es möglich ist, mit SAP R/3-Systemen ab dem Release 3.1H zu kommunizieren. Es kann von Java aus auf RFC-fähige (Remote Function Call) Funktionsbausteine zugegriffen werden und ABAP Programme können Java Programme aufrufen.
JCo bietet zwei Verbindungsmöglichkeiten: Zum einen kann man direkt eine Verbindung zu einem R/3-System erstellen und muss diese, wenn sie nicht mehr gebraucht wird, beenden. Zum anderen kann man über *Connection Pools* arbeiten, bei denen mehrere Verbindungen zu einem R/3-System bestehen. Es wird immer die nächste freie Verbindung genutzt. Diese wird wieder an den Pool zurückgeben, wenn sie nicht mehr gebraucht wird. Pool-Verbindungen sind z.B. sinnvoll, wenn über einen Webserver auf ein R/3-System oft zugegriffen wird. Dabei muss nicht bei jedem Zugriff eine Verbindung hergestellt werden. Beim Verbindungsaufbau müssen neben Benutzername, Passwort und Mandant auch entweder der Message-Server und –Port oder der Hostname und die Systemnummer des gewünschten R/3-

Systems angegeben werden. Zudem kann man noch die Sprache angeben, die benutzt werden soll.
Über JCo lässt sich jeder ABAP Datentyp in einen Java Datentyp umwandeln. Dadurch kann man alles, was aus der einen Welt kommt, auch in der anderen Welt sinnvoll verarbeiten. JCo bietet zudem Unterstützung zur Generierung der Import/Export Parameter und Tabellen. Es können auch die im Funktionsbaustein auftretenden Ausnahmen in Java abgefangen und ausgewertet werden. Das Navigieren über die Import/Export Tabellen wird von der Schnittstelle ebenso bereitgestellt, wie auch die Information über die Datenformate (String, int, double, usw.) der einzelnen Daten.

2.3.6 SAP Enterprise Portal 5.0

Heutzutage reicht es nicht mehr, nur auf eine Grosszahl von Informationen zugreifen zu können. Man muss diese Informationen in der richtigen Form bekommen, um mit ihnen produktiv arbeiten zu können.
Es werden Informationen von Geschäftsanwendungen, Datenbanken, e-mail, gemeinsam genutzen Dokumenten und dem Internet benötigt. Aus diesem Grund wird eine Lösung benötigt, die diese verschiedenen Informationen an einer Stelle zusammen bringt und bereitstellt. Solch eine Lösung ist das SAP Enterprise Portal (EP).
Das EP bietet folgende Möglichkeiten (vgl. [@SAP02d]):

- Verschiedene Typen von Daten (Anwendungen, Datenbanken, Dokementen und Internetinformationen) zu verwalten.
- Alle Informationen eines Unternehmens ohne Berücksichtigung der Herkunft zu vereinen.
- Inhalte Nutzern nach ihren Anforderungen und Interessen bereitzustellen.
- Die Zusammenarbeit und Personalisierung zu vereinfachen.

Das Enterprise Portal vereint Applikationen, Informationen und Dienste die in einem Unternehmen vorhanden sind in einem System. Es ist eine personalisierbares und interaktives System, welches Arbeitnehmen, Partnern, Lieferanten und Kunden einen Single Point of Access bietet. Auf das EP kann über verschiedene Geräte, von jedem Ort und zu jeder Zeit zugegriffen werden.
Die Portal Plattform umfasst u.a. folgende Teile (vgl. [@SAP02d]):

- Die iView Technologie, zum Erstellen und Administrieren von iViews. Es können entweder die von SAP bereitgestellten genutzt werden, oder eigene erstellt werden. iViews können in verschiedenen Sprachen programmiert werden, u.a. Java, JavaScript, ASP, XML, COM.
- Die Unification Technologie, die einen vereinten Zugriff auf Applikationen ermöglicht, um Drag&Relate Operationen bereitzustellen.
- Die Knowledge Management Plattform, das Busisness Package „Portal User“ und die Collaboration Business Packages. Zusätzlich den R/3 Unifier zum Erstellen von Drag&Relate Operationen zwischen SAP Transaktionen.
- Den Page Builder, der für das Rendering der HTML-Seiten zuständig ist, die der Nutzer sieht und mit denen er arbeitet.
- Das User Management, zum Administrieren von Nutern und zum Abbilden der Nutzer Ids in LDAP Verzeichnissen. Dies ermöglicht das Verwenden von Single Sign-On, auch wenn Nutzer in verschiedenen Systemen unterschiedliche Ids haben.
- Das User Role Management, zum Erstellen der Benutzerrollen, welche vorkonfiguierte Portalseiten für Nutzer bieten.

Zu den Schlüsseleigenschaften des Enterprise Portals gehören u.a. die in Tab. 2-1 beschriebenen.

Roles	Bietet die Möglichkeit Inhalte den verschiedenen Rollen in einem Unternehmen zuzuordnen. Das EP bietet bereits über 300 vordefinierte Industrie-spezifische und –übergreifende Rollenvorlagen (Templates). Diese können einfach angepasst werden.
Personalization	Ermöglicht es das EP individuellen Anforderungen anzupassen. Zum Beispiel können Nutzer einfach das Aussehen des Portals ändern, die gewünschte Sprache auswählen oder nicht benötigte Inhalte ausblenden.
iViews	Diese Portal Anwendungen bringen die Inhalte den Rollen entsprechend zu den Nutzern und helfen ihnen dabei ihre Geschäftsvorgänge zu bearbeiten.
Content Management	Bietet den gesamten Lebenszyklus der Content-Entwicklung. Dazu gehört u.a. Erstellen, Klassifizieren, Strukturieren und Anbieten von Inhalten.
Syndication	Hilft Nutzern beim Zugreifen auf Informationen, Anwendungen und Diensten, beim Erlangen von Informationen durch Analysen und Auswertungen und beim Anzeigen der Inhalte wo und wann sie benötigt werden.
Drag&Relate	Ermöglicht es Objekte auf dem Browser mittels Drag&Drop so zuverschieben, um die richtigen Informationen zu finden und spezielle Tätigkeiten auszuführen.
Security	Bietet Single Sign-On und Internet Sicherheitstechnologien, um die Authentifizierung und eine sichere Kommunikation zwischen Nutzern und den verschiedenen Ressourcen sicherzustellen.

Tab. 2-1: Schlüsseleigenschaften des SAP Enterprise Portals (vgl. [@SAP02d])

Das Enterprise Portal 5.0 bietet zwei Möglichkeiten um Funktionalitäten hinzuzufügen. Dies ist einerseits über Portal-Services und andererseits über Portal-Komponenten möglich.

2.3.6.1 Portal-Services

Ein Portal-Service bietet Funktionalitäten, die für Komponenten innerhalb der *iView Runtime for Java* global zugreifbar sind. Die Komponenten greifen auf die Services über APIs zu. Das Enterprise Portal 5.0 bringt bereits einige Services mit. Dabei gibt es zwei Gruppen in die die Services aufgeteilt werden können. Einerseits sind dies die Basis-Services, die Teil der iView Runtime for Java sind und andererseits Zusatz-Services die auf der iView Runtime aufsetzen.

Basis-Services:

Internationalization	Wenn Texte in den iViews verwendet werden, die dem Nutzer angezeigt werden, muss sichergestellt werden, dass diese Texte übersetzt werden können. Der Internationalization-Service stellt sicher, dass die Komponente die Sprachressourcendatei verwendet, die mit der Loginsprache des Nutzers übereinstimmt.
Access of Profiles	Ein *Profile* ist eine Eigenschaftsdatei, die Name-Wert Paare definiert, welche die iView Runtime nutzt zum Ausführen von iViews. Die Runtime bietet zusätzlich eine Stelle, um solche Name-Wert Paare zu speichern und ein API, über welches auf diese Daten zugegriffen werden kann. Die Profiles enthalten zwei Arten von Informationen. Einerseits Nutzer spezifische Informationen und andererseits iView Konfigurationsinformationen.
Access of Resources	Der Ausdruck *Resources* bezieht sich auf Dateien, die mit iViews geliefert werden. Ressourcen in dieser Hinsicht können von zwei verschiedene Arten sein, öffentlich oder privat. Öffentlich sind die, welche in den öffentlichen Teil des Web-Servers kopiert werden.

	Private Ressourcen sind dagegen im nicht-sichtbaren Bereich. Dies bedeutet, dass auf die öffentlichen Ressourcen über URLs zugegriffen werden kann während auf die privaten kein solcher Zugriff möglich ist. Um auf diese zugreifen zu können, bietet die iView Runtime eine spezielle API, die von den Komponenten dazu genutzt werden kann.

Tab. 2-2: Basis-Services des Enterprise Portals 5.0 (vgl. [@SAP02d])

Zusatz-Services:

Client Eventing	Das Client Framework ermöglicht es Portal-Komponenten in einer einfachen Art und Weise auf dem Client zu kommunizieren. Mehrere Features, wie z.B. *automatic relaxing of the JavaScript Origin Policy*, *Client Eventing* und *Client Data-Bag* vereinfachen die Integration und Kommunikation von verschiedenen Portal-Komponenten aus unterschiedlichen Servern.
Logger	Dieser Service wird genutzt, um Problemdiagnosen und detaillierte Informationen über auftretende Probleme bereitzustellen.
URL Generator	Der URL Generator Service bietet die Möglichkeit URLs • Zu Seiten, • zum SAP GUI for HTML, um SAP Transaktionen (über ITS) anzuzeigen, • zu Services die auf einem ITS definiert sind (MiniApps und IACs (Internet Application Components)) und • zu SAP Business Warehouse Reports generieren.
JCo Client Service	Dieser Service definiert alle benötigten Aktionen für Verbindungen zu SAP Systemen über eine RFC-Verbindung.
HTML Business for Java	HTML Business for Java liefert dem Entwickler eine effiziente Sammlung von *Controls* ähnlich wie JavaSwing. Diese Controls basieren auf Servlets und JSPs. Der Entwickler verwendet JavaBean-ähnliche Komponenten oder JSP tags. Render-Klassen übersetzen die Komponenten in HTML-Tags.
Portal Data Viewer	Mit diesem Service ist es möglich Tabellendaten darzustellen.

Tab. 2-3: Zusatz-Services des Enterprise Portals 5.0 (vgl. [@SAP02d])

2.3.6.2 Portal-Komponenten

Der Nutzer eines Enterprise Portal 5.0 interagiert immer mit Portal-Komponenten. Sie stellen die Funktionalität bereit, die der Nutzer benötigt. Dabei können verschiedene iViews aus den Portal-Komponenten erzeugt werden. Die Portal-Komponenten haben die Möglichkeit auf alle Portal-Services zuzugreifen.
Eine Portal-Komponente besteht aus einem oder mehreren Master iViews. Sie hat einen Applikationsteil (z.B. eine Java-Klasse und/oder JSPs) und kann Ressourcen (z.B. Bilder, Eigenschaftsdateien, usw.) nutzen. Eine Portal-Komponente benötigt eine Eigenschaftsdatei (property file), die der Portal-Runtime u.a. mitteilt,

- welche Klasse beim Aufruf der Komponente genutzt werden soll,
- welche Services die Komponente benötigt und
- in welchem Modus die Portal-Komponente laufen soll.

Alle Dateien die zu einer Portal-Komponente gehören werden in einer par-Datei zusammengefasst (PAR = Portal Archive). Die par-Datei wird dann im Enterprise Portal

deployed (installiert). Die par-Datei nutzt des zip-Format und kann somit über jedes Programm geöffnet werden, dass dieses lesen kann.

2.3.7 SAP WebDynpro

WebDynpro ist die neue User-Interface Technologie der SAP. Sie bietet eine Entwicklungs- und Laufzeitumgebung für Web-Applikationen. Sie erweitert die klassische Entwicklung von Web-Applikationen dahingehend, dass es leicht möglich ist, anpassungsfähige User-Interfaces zu entwickeln. WebDynpro verwendet ähnliche Designprinzipien wie die Dynpro-Technologie von SAP in R/3. Allerdings stellt es eine komplett neue Technologie dar, die ausschliesslich auf Web-Applikationen spezialisiert ist.
WebDynpro soll es ermöglichen, soviel Coding wie möglich zu vermeiden und eine Unabhängigkeit, sowohl vom Backend-System als auch von der Frontend-Technologie zu erreichen. Damit möglichst wenig eigenes Coding geschrieben werden muss, bietet WebDynpro eine beschreibendes Metamodell, um Benutzeroberflächen zu entwickeln. Über diese abstrakte Definition generiert WebDynpro lauffähige Web-Applikationen für unterschiedliche Laufzeitplattformen wie J2EE, ABAP oder .NET.
Zusätzlich verwendet WebDynpro JavaScript zum Rendern der Oberflächen direkt im Browser. Dies bedeutet, dass das HTML das der Benutzer sieht, auf dem Rechner des Benutzers aus den übertragenen Daten und Layout Informationen erzeugt wird. Dadurch wird der Frontend sehr dynamisch und es können die Seiten flicker-frei aktualisiert werden.
Für die Entwicklung von WebDynpro Applikationen gibt es eine graphische Werkzeugsammlung, die in einer IDE (Integrated Development Environment) integriert ist. Die Werkzeugsammlung hilft Entwicklern dabei, die WebDynpro Metadaten zu entwickeln und ermöglicht es direkt aus der IDE die Applikation zu erstellen, installieren und zu testen.

3 IST Zustand

Wie bereits in der Aufgabenstellung kurz beschrieben, werden von über 1000 R/3-Systemen bei SAP Informationen über Systemmeldungen, Loginzeiten und Transporte durch ein ABAP-Programm (`ZSCANSYSTEMS`) gesammelt. Diese Daten werden in drei verschiedenen Tabellen abgespeichert. Das ABAP-Programm läuft im *CUE* (Customer Usage Evaluation) System, welches in naher Zukunft nicht mehr zur Verfügung steht.

Beim Aufruf des Programms kann man verschiedene Parameter angeben: Die zu selektierenden Systeme, die Mandanten, verschiedene Administrationskürzel, den Benutzernamen und das Passwort für die Systeme, die gelesen werden sollen. Das Programm liest aus einer Tabelle des *CUE* Systems alle internen Systeme aus. Um die jeweilige IP-Adresse der Hosts zu bekommen, macht es einen DNS Lookup mittels der Kernel-Funktion `RFCCONTROL`. Dann ruft das Programm für jedes einzelne dieser Systeme den Funktionsbaustein `BDWP_UMON_SCAN_ONE_SYSTEM` mit den, beim Start des Hauptprogramms angegebenem, Benutzernamen und Passwort auf. Die Ausführung des Funktionsbausteins geschieht asynchron, d.h. es werden mehrere Systeme gleichzeitig gelesen. Durch diese Parallelisierung wird viel Zeit eingespart, da es vorkommen kann, dass das Login für ein System fehlschlägt. Dies hätte einen Abbruch mit entsprechender Systemfehlermeldung zur Folge. Das Weiterarbeiten des ABAP-Programms würde dadurch unnötig gebremst werden.

Der Funktionsbaustein `BDWP_UMON_SCAN_ONE_SYSTEM` führt für jedes System, das er scannen soll, verschiedene weitere Funktionsbausteine aus. Die Funktionsbausteine sperren und entsperren Schlüsselbereiche auf die zugegriffen wird, testen, ob eine RFC-Verbindung zu dem entsprechendem System aufgebaut werden kann, lesen die gewünschten Daten aus den benötigten Tabellen und schliessen die RFC-Verbindungen wieder.

Die gelesenen Daten befinden sich in den Tabellen `USR02` (Loginzeiten der Benutzer), `TEMSG` (Systemmeldungen) und `E070` und `E07T` (Transportinformationen). Sie werden von dem Funktionbaustein in den Tabellen `/dwp/usr02`, `/dwp/temsg` und `/dwp/e070` gespeichert. Beim Schreiben der Daten in die entsprechenden Tabellen werden die Daten mit gleichem Primärschlüssel überschrieben, alle anderen hinzugefügt und alte Daten gelöscht. Auf die Tabellen greifen verschiedene Programme zu und bereiten die Daten auf.

Das ABAP-Programm `ZSCANSYSTEMS` wird momentan alle sechs Stunden im Batch über das R/3-Jobsystem aufgerufen. Es liest alle verfügbaren Systeme auf die es Zugriff hat, unabhängig davon, ob die Daten aktualisiert werden müssen oder nicht. Wenn einmal ein System nicht verfügbar war, werden die Daten erst beim nächsten Scannen aktualisiert.

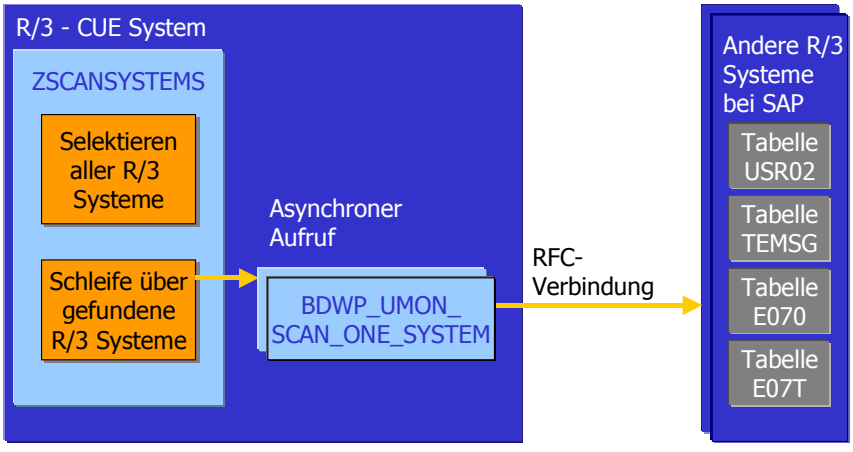

Abb. 3-1: Programmablauf des IST Zustands

Die Tabellen, in denen die Daten bisher gespeichert wurden, befinden sich ebenfalls im *CUE*-System. Auch diese müssen neu, in einem externen Datenbank-System, erstellt werden. Die Tabellen haben eine flache Struktur und keine Beziehungen zu anderen Tabellen.
Das Kürzel PK in den folgenden Tabellen in der Spalte Key steht für Primary Key (Primärschlüssel).

Die Datenbanktabelle `/dwp/usr02` hat (Stand: 21.05.02) 1.059.379 Einträge. Sie speichert die Loginzeiten der Benutzer. Ihre Struktur ist in Tab. 3-1 angegeben.

Feldname	Key	Datenelement	Typ	Länge	Kurzbeschreibung
SRCSYSTEM	PK	SRCSYSTEM	CHAR	10	System in dem sich das Original befindet
MANDT	PK	MANDT	CLNT	3	Mandant
BNAME	PK	XUBNAME	CHAR	12	Benutzername im Benutzerstamm
USTYP		XUUSTYP	CHAR	1	Benutzertyp (A=Dialog, C=CPIC, D=DBC, O=ODC)
TRDAT		XULDATE	DATS	8	Letztes Login-Datum
LTIME		XULTIME	TIMS	6	Letzte Login-Uhrzeit

Tab. 3-1: Struktur der R/3 - Tabelle /dwp/usr02

Die Systemmeldungen werden in der Datenbanktabelle `/dwp/temsg` gespeichert, welche zum obigen Zeitpunkt 2.791 Einträge und die Struktur in Tab. 3-2 hat.

Feldname	Key	Datenelement	Typ	Länge	Kurzbeschreibung
SRCSYSTEM	PK	SRCSYSTEM	CHAR	10	System, in dem sich das Original befindet
DBSERV	PK	SYDATHOST	CHAR	8	Rechnername
MAINDOMAIN	PK	SYDATDOM	CHAR	30	R/3 Systemdaten: Domäne des Servers
ID	PK	CHAR10	CHAR	10	Characterfeld der Länge 10
EMTEXT		EMTEXT	CHAR	60	Express-Message Text
DATCRE		EMEDATCRE	DATS	8	Anlege-Datum einer Express-Message
TIMCRE		EMETIMCRE	TIMS	6	Anlege-Uhrzeit einer Express-Message
DATDEL		EMEDATDEL	DATS	8	Verfalls-Datum einer Express-Message
TIMDEL		EMETIMDEL	TIMS	6	Verfalls-Uhrzeit einer Express-Message
AUTHOR		EMEAUTHOR	CHAR	12	Autor der Expressnachricht
APPLSERVER		MSNAME	CHAR	20	Server-Name
NOROW		CHAR3	CHAR	3	Feld der Länge 3 Bytes
CUROW		CHAR3	CHAR	3	Feld der Länge 3 Bytes
CLIENT		MANDT	CLNT	3	Mandant
DATREM		EMEDATDEL	DATS	8	Verfalls-Datum einer Express-Message
TIMREM		EMETIMDEL	TIMS	6	Verfalls-Uhrzeit einer Express-Message

Tab. 3-2: Struktur der R/3 - Tabelle /dwp/temsg

Die Datenbanktabelle `/dwp/e070`, welche die Transportinformationen speichert, hatte zum oben genannten Zeitpunkt 6.549.420 Einträge und die Struktur in Tab. 3-3.

Feldname	Key	Datenelement	Typ	Länge	Kurzbeschreibung
SRCSYSTEM	PK	SRCSYSTEM	CHAR	10	System, in dem sich das Original befindet
TRKORR	PK	TRKORR	CHAR	20	Auftrag/Aufgabe
TRFUNCTION		TRFUNCTION	CHAR	1	Funktion des Auftrags/Aufgabe
TRSTATUS		TRSTATUS	CHAR	1	Status des Auftrags/Aufgabe
TARSYSTEM		TR_TARGET	CHAR	10	Transportziel eines Auftrags
AS4USER		AS4USER	CHAR	12	Autor der letzten Änderung
AS4DATE		AS4DATE	DATS	8	Datum der letzten Änderung
AS4TIME		AS4TIME	TIMS	6	Uhrzeit der letzten Änderung
STRKORR		STRKORR	CHAR	20	Übergeordneter Auftrag
LANGU		DDLANGUAGE	LANG	1	Sprachenschlüssel
AS4TEXT		AS4TEXT	CHAR	60	Kurzbeschreibung von Repository-Objekten
CLIENT		TRCLIENT	CHAR	3	Quellmandant eines Auftrags

Tab. 3-3: Struktur der R/3 - Tabelle /dwp/e070

3.1 Mängel

Die bisherige Lösung, birgt einige Mängel, die es in einer neuen Version zu bereinigen gilt. Diese lassen sich in vier Bereiche einteilen. Erstens geht es um die Zugriffe, auf die verschiedenen Systeme und deren Erfolg oder Misserfolg und die Ursachen dafür. Ein weiterer Punkt ist die Inkonsistenz der Tabellen in denen die Daten nach dem Lesen gespeichert werden. Desweiteren gibt es keine Auswertungsmöglichkeiten darüber,

- wie viele Systeme gescannt wurden,
- wie viele Datensätze aus den jeweiligen Systemen zusammen getragen wurden und
- die Ausführungshäufigkeit und die Ausführungszeitpunkte.

Zusätzlich ist das Aufbauen von Verbindungen zwischen R/3 und R/3 nicht so geschickt wie zwischen Java und R/3. Auch die Einbettung in Portale lässt sich nicht ohne einem R/3-System ermöglichen.

3.1.1 Fehlerbehandlung beim Systemzugriff

In der aktuellen Lösung wird nicht dokumentiert, ob das zum Scannen aufgerufene System erreichbar war und der Lese-Vorgang erfolgreich abgeschlossen werden konnte. Somit kann die Problemursache nicht festgestellt werden. Es könnte z.B., nicht verfügbar gewesen sein, was kein grosses Problem darstellen würde, da im nächsten Lauf das System ausgelesen wird. Aber es könnten auch Benutzername oder Passwort für dieses System falsch gewesen sein. In diesem Falle, wiederholt sich der Fehler bei jedem Aufruf bis der jeweilige Systemadministrator den wiederholten Zugriffsversuch bemerkt und die für die Ausführung des Programms zuständige Personen kontaktiert. Hier könnte man sich eine Fehlermeldung vorstellen mit deren Hilfe der Systemadministrator feststellen kann, was er ändern muss.

3.1.2 Inkonsistenz der Tabellenstruktur

Wie in den Tab. 3-1, Tab. 3-2, Tab. 3-3 zu sehen ist, besteht ein Teil der Primärschlüssel aus dem Systemnamen in dem sich das Original befindet. Zudem sind in der Tabelle /dwp/temsg der DBSERV und die MAINDOMAIN, die auch Informationen über das Originalsystem bieten, ein Teil des Primärschlüssels. Der Unterschied zwischen den Tabellenstrukturen liegt darin, dass bei den anderen beiden Tabellen diese Information anfangs nicht benötigt wurden und sie aus diesen Gründen nicht berücksichtigt wurden. Allerdings sollte, um die Konsistenz der Tabellenstrukturen herzustellen, dies geändert werden und die Daten in alle Tabellen aufgenommen werden.

3.1.3 Datenauswertung

Es ist bisher keine Auswertung implementiert, über die ersichtlich wird, wie viele Systeme erfolgreich gescannt wurden oder welche Systeme Probleme lieferten. Wenn z.B. mehr als 50% der Systeme Probleme liefern würden, könnte dies darauf hinweisen, dass das Scan-Programm bei der Durchführung Fehler verursacht und somit überprüft und notfalls geändert werden sollte. Ebenfalls wird nicht dokumentiert, wie viele Datensätze aus den jeweiligen Systemen gelesen werden.

3.1.4 Ausführungshäufigkeit/-zeitpunkt

Wie bereits weiter oben erwähnt, wird die aktuelle Lösung über das R/3-Jobsystem alle sechs Stunden erneut ausgeführt. Dies geschieht ohne Bezug darauf, ob aktualisierte Daten benötigt werden oder nicht. Durch eine angepasste Ausführungshäufigkeit könnte die Netzlast und die Last der einzelnen Systeme verringert werden.

3.1.5 Verbindungen in Java besser als in R/3

Der Verbindungsaufbau bei RFC-Aufrufen ist innerhalb der R/3-Systeme auch nicht optimal. Für jedes System, zu dem eine Verbindung aufgebaut werden will, muss im rufenden R/3 eine Systembeschreibung mit den entsprechenden Verbindungsparametern vorhanden sein. Aus Java kann eine RFC-Verbindung weitaus einfacher erstellt werden. Es werden beim Aufruf nur die notwendigen Daten benötigt. Dabei genügen zum Auffinden des R/3-Systems schon der Message-Server und -Port.

3.1.6 Einbettung in Portale ohne R/3

Ein weiterer Nachteil des vorhandenen Systems ist es, dass die gescannten Daten nicht ohne R/3-System in ein Portal eingebettet werden können. Es muss immer das R/3-System vorhanden sein, welches die gescannten Daten spiechert. Dies kann durch die Nutzung der vorhanden Infrastruktuer (J2EE) geändert werden. Da die Daten aus R/3 gelesen Daten in einer eigen Datenbank zur Verfügung stehen und ohne eine Verbindung zu einem R/3-System direkt im Portal angezeigt werden können.

4 Sollbeschreibung (Spec)

In diesem Kapitel werden die Anforderungen für das zu entwickelnde System beschrieben. Dabei wird die Struktur des SAP Spec-Templates [@SAP02h] verwendet, das von der HORIZON-Gruppe (SAP-interne Qualitätssicherung) zur Verfügung gestellt wird.

4.1 Anforderungen (Überblick)

In diesem Abschnitt wird der Umfang der Anforderungen an das zu erstellende System umrissen. Dazu gehören neben der Funktionalität, die Zielgruppen, sowie die Programmschnittstellen.

4.1.1 Leistungsumfang

Dieses Projekt soll das Monitoring von Loginzeiten der Benutzer, von Transportinformationen und von Systemmeldungen ermöglichen. Das Monitoring soll dabei über alle vorhandenen R/3-Systeme möglich sein. Dabei sollen die zu entwickelnden Tools, in eine J2EE Umgebung eingebettet werden. Es gilt in erster Linie das bereits vorhanden ABAP-Programm `zscansystems` in Java neu zu entwickeln. Dieses Programm selektiert zuerst alle benötigten Systeme und scannt jedes einzeln nach den erforderlichen Daten. Die gescannten Daten werden in eigens dafür erstellten Tabellen abgespeichert. Um dieselbe Funktionalität in einer Java-Applikation bereit zu stellen, müssen über den CSS DNS-Alias (CSS = Customer Service System) alle benötigten Systeme und die Daten, um diese aufzufinden, selektiert werden. Der Alias verweist immer auf das momentan laufende Service System innerhalb von SAP, CSN (Customer Service New) oder CSR (Customer Service Reporting). Dann müssen die benötigten Tabellen in jedem R/3-System gelesen und in einem externen Datenbanksystem (MonitoringDB) abgelegt werden. Zu den bisher gesammelten Daten sollen noch weitere Informationen gespeichert werden. Diese sollen Fehler und Probleme beim Scannen der Systeme dokumentieren, darauf reagieren und Protokolle (Anzahl gescannter Daten, Systeme) des Scanvorgangs ermöglichen.
Wenn ein Zugriff auf ein System nicht möglich sein sollte, da die Benutzerinformationen falsch oder nicht vorhanden sind, soll eine geeignete IT/IBC Meldung (interne Fehlermeldung an die IT) auf Wunsch des Administrators erstellt werden können. Diese soll so erstellt werden, dass der zuständige Bearbeiter ohne Rückfragen an den zuständigen Mitarbeiter für dieses Programm die nötigen Änderungen durchführen kann. Sollte es vorkommen, dass eine grosse Zahl von Systemen (z.B. 50%) beim Lesen Fehler generieren, sollte dem Zuständigen des Programms eine Meldung ausgegeben werden, damit er das Problem analysieren kann und gegebenenfalls dieses Programm ändert.
Das zu erstellende Programm soll automatisch in gewissen Zeitabständen starten. Zudem soll es adaptiv sein und erkennen, wann und welche Systeme öfters gescannt oder weniger oft gescannt werden sollten. Es soll auch eine Möglichkeit geben, dass nicht nur Systeme, sondern auch Daten mit unterschiedlicher Frequenz gescannt werden. Auch soll es möglich sein, den Scanvorgang jederzeit von Hand zu starten.

Desweiteren muss eine Möglichkeit erstellt werden, über die auf die gesammelten und gespeicherten Daten zugegriffen werden kann. Dabei ist eine Einbettung in Portale einer der wichtigsten Aspekte.

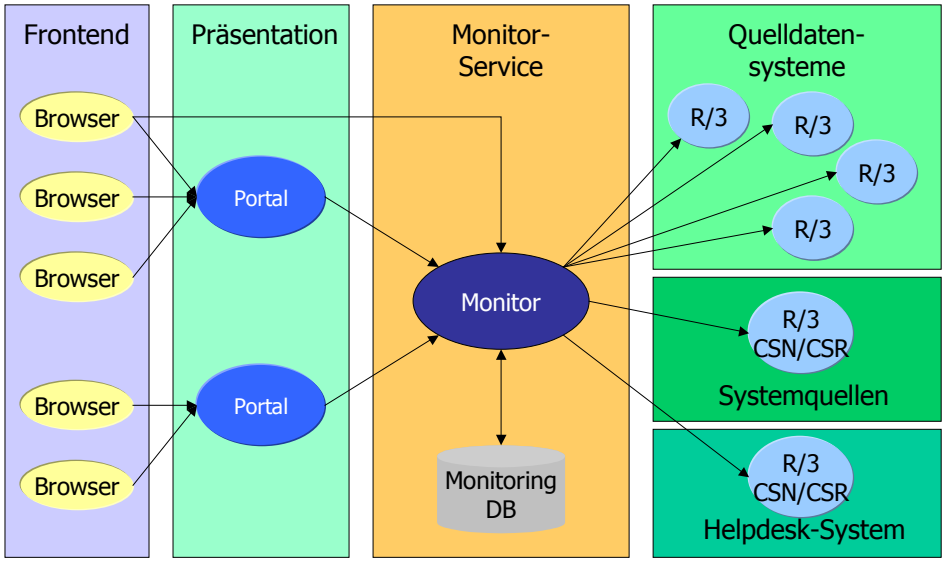

Abb. 4-1: Überblick über das Monitoring System

Wie in Abb. 4-1 zu erkennen ist, kann das Monitor-System in mehrere Teile aufgeteilt werden. Das Frontend, das die Schnittstelle zu den Nutzern des Systems bildet. Die Präsentation, auf deren Ebene die Portale laufen. Der Monitor-Service, der für die Selektion und Speicherung der Daten zuständig ist. Die Systemquellen, aus denen die verfügbaren Systeme gelesen werden, die gescannt werden können, die Quelldatensysteme, aus denen die Daten gelesen werden und einem Helpdesk-System, über welches Fehlermeldungen aufgeben werden können.

4.1.2 Zielgruppen, Rollen

Es gibt verschiedene Rollen für Personen, die mit diesem System arbeiten können. Dies sind der Monitor-Admin, der System-Admin und der Monitor-User.

4.1.2.1 Monitor-Admin

Der Monitor-Admin überwacht das System. Er plant die Ausführungszeitpunkte und kann auch den Scanvorgang selbst starten. Er gibt an, welche Systeme und Tabellen gescannt, und wo die Daten gespeichert werden sollen.

4.1.2.2 System-Admin

Ein System-Admin ist eine Person, die für ein oder mehrere R/3-Systeme verantwortlich ist. Er sollte sich alle Daten zu seinen Systemen anzeigen lassen können, die gesammelt werden (z.B. welche User waren am System angemeldet, Systemmeldungen des Systems, Transporte die noch nicht freigegeben sind). Zudem kann er Systemmeldungen direkt im Monitoring-System pflegen, für den Fall, dass ein R/3-System nicht verfügbar ist und er so den Nutzern mitteilen möchte, wie lange dies der Fall ist.

4.1.2.3 Monitor-User

Monitor-User sind User, die sich die gesammelten Daten anzeigen lassen können. Dies können ihre eigenen Transportaufträge in den verschiedenen Systemen sein oder die Systemmeldungen der Systeme mit denen sie arbeiten.

4.1.3 Programmschnittstellen

Hier sollen die Schnittstellen beschrieben werden mit denen das Monitor-System arbeitet. Dazu werden die Komponenten betrachtet mit welchen das Monitor-System kommuniziert. (vgl. Abb. 4-2)

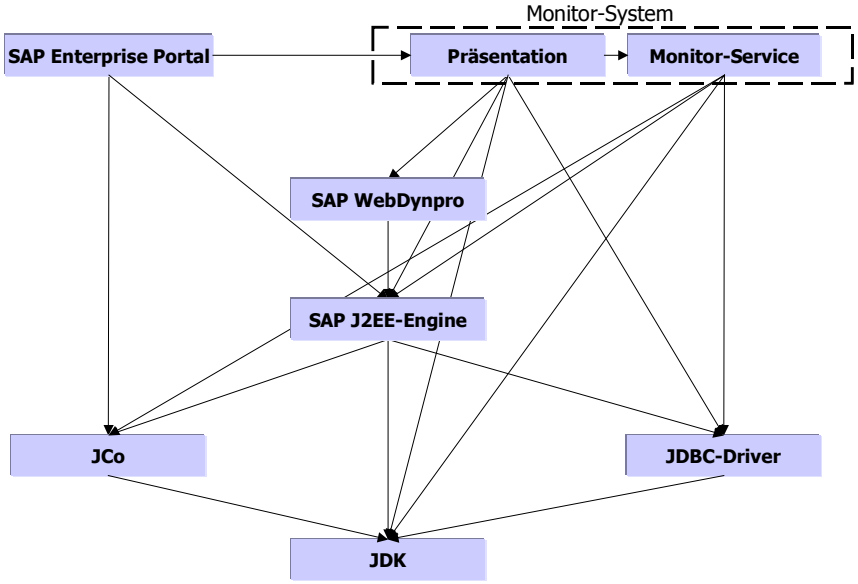

Abb. 4-2: Programmschnittstellen des Monitor-Service

Der Monitor-Service und die Präsentation verwenden Schnittstellen zur J2EE-Engine und zu den JDBC-Treibern. Zur J2EE-Engine, da das System in dieser laufen wird und zu JDBC-Treibern, da die gelesenen Daten in einer externen Datenbank gespeichert und aus dieser gelesen werden. Dabei bietet auch die J2EE-Engine eine Schnittstelle zu den JDBC-Treibern, da es auch die Möglichkeit gibt, dass die Engine über Entity Beans mit der Datenbank kommuniziert.

Der Monitor-Service verwendet zum Lesen der R/3-Systeme die von SAP entwickelte Schnittstelle JCo (Java Connector). Über diese Schnittstelle können in Java RFC-Verbindungen zu R/3-Systemen hergestellt und Daten transferiert werden.

Die Datenbereitstellung bietet eine Schnittstelle für Portale für den Zugriff auf die Daten. Das Einbinden der Daten in Portale soll über WebDynpro erfolgen, um diese neue Technologie und ihre Möglichkeiten kennen zu lernen.

4.2 Anforderungen im Detail

Nachfolgend wird das System aus Anwendersicht in Form von Use Cases beschrieben. Zur Verdeutlichung wurden alle Use Cases mit Hilfe von Diagrammen illustiert.

4.2.1 Liste der Use Cases

Tab. 4-1 zeigt die Liste der Use Cases. Sie ist nach Benutzerrolle und Priorität sortiert.

Use Case/Funktion	Priorität (1 – 4)	Benutzerrolle	Bemerkungen
Monitor-Service	1	Monitor-Admin	
(Re-)Konfigurieren des Service	1	Monitor-Admin	

Use Case/Funktion	Priorität (1 – 4)	Benutzerrolle	Bemerkungen
Administrieren des Service	1	Monitor-Admin	
Daten zum Aktualisieren markieren	2	Monitor-Admin	
Anlegen einer IT/IBC-Meldung	3	Monitor-Admin	Als angenehmes Feature gedacht, daher nicht allzu hoch gewichtet.
Systemmeldungen der genutzten R/3-Systeme anzeigen	2	Monitor-User	
Eigene Transporte anzeigen	2	Monitor-User	
Systemmeldungen pflegen	2	System-Admin	
User eines R/3-Systems anzeigen	3	System-Admin	Da vom System-Admin auch direkt in den R/3-Systemen möglich, fällt die Gewichtung eher niedrig aus.
Transporte eines R/3-Systems anzeigen	3	System-Admin	Da vom System-Admin auch direkt in den R/3-Systemen möglich, fällt die Gewichtung eher niedrig aus.

Tab. 4-1: Liste der Use Cases

4.2.2 Beschreibung der einzelnen Use Cases

Nachfolgend wird jeder aufgeführte Use Case kurz beschrieben.

4.2.2.1 Monitor-Service

Der Monitor-Service liest alle Sysetme mit ihren Daten aus CSN/CSR. Zudem werden alle R/3-Systeme anfangs in einer vorgegebenen Häufigkeit gescannt. Der Service erkennt während fortdauernder Laufzeit, von welchen Systemen Daten häufiger und weniger häufig genutzt werden und passt die Scan-Intervalle der Systeme daran an.

4.2.2.2 (Re-)Konfigurieren des Service

Der Monitor-Admin muss den Service anfangs konfigurieren, damit dieser die nötigen Daten bekommt, um seine Aufgaben auszuführen. Nach dem der Service läuft, kann er immer wieder eingreifen um wichtige Daten zu ändern.
Hier kann eingestellt werden,

- wie auf CSS zugegriffen werden kann,
- welche Systeme selektiert werden sollen,
- welche Tabellen gelesen werden sollen,

- wo die Daten abgespeichert werden sollen (Tabelle, Felder),
- wie Username und Passwort für die Systeme sind,
- wann der Service die Selektion starten soll und
- nach welcher Zeit die Selektion wiederholt werden soll.

4.2.2.3 Administrieren des Service

Der Monitor-Admin kann den kompletten Service starten und stoppen. Zudem kann er die Protokolle die zu den Scan-Läufen angelegt werden sich anzeigen lassen und auswerten. Falls er daraus entscheiden sollte, dass er für bestimmte Systeme den Scan-Vorgang von Hand starten möchte ist dies ebenfalls möglich.

4.2.2.4 Daten zum Aktualisieren markieren

Der Monitor-Admin kann dem Service angeben, welche Daten beim nächsten Scan-Vorgang auf jeden Fall neu aus den R/3-Systemen gelesen werden sollen. Dadurch erhält er die Möglichkeit, in die Scan-Frequenz von bestimmten Systemen und Daten einzugreifen.

4.2.2.5 Anlegen einer IT/IBC-Meldung

Der Monitor-Admin bekommt die Möglichkeit sich alle Systeme mit Zugriffsproblemen und ihren Ursachen anzeigen zu lassen. Dabei kann er dann direkt Systeme auswählen für die IT/IBC-Meldungen in CSN/CSR angelegt werden sollen. Dabei wird entweder eine vom System vorkonfigurierte Standardmeldung aufgegeben oder der Monitor-Admin gibt selbst einen Meldungstext an. Dabei kann er auch eine Komponente angeben.

4.2.2.6 Systemmeldungen der genutzten R/3-Systeme anzeigen

Falls ein Monitor-User sich für bestimmte R/3-Systeme die entsprechenden Systemmeldungen anzeigen lassen kann, ist dies über den Monitor möglich. Dabei besteht die Möglichkeit sich alle Meldungen, von allen Systemen, an denen der Nutzer sich in einem bestimmten Zeitraum angemeldet hat anzeigen zu lassen. Zudem ist es möglich, dass der Nutzer die Meldungen für bestimmte, von ihm gewünschte Systeme, angezeigt bekommt.

4.2.2.7 Eigene Transporte anzeigen

Der Monitor-User kann sich auch alle eigenen Transporte über alle Systeme hinweg anzeigen lassen, um zu sehen, welche/wie viele Transporte von ihm bearbeitet werden/wurden. Dabei ist es möglich sich auch nur die noch nicht freigegebenen anzeigen zu lassen, um zu sehen wo noch Arbeit geleistet werden muss, oder die Transporte freigegeben werden müssen.
Der Monitor-User kann beim Selektieren folgendes auswählen:

- Systeme, die angezeigt werden sollen,
- die Transportnummer,
- den Status der Transporte, die er sehen möchte,
- das Datum der letzten Änderung und
- die Mitwirkenden des Auftrags.

4.2.2.8 Systemmeldungen pflegen

Der System-Admin kann für R/3-Systeme im Monitor-System Systemmeldungen anlegen. Dies kann sinnvoll erscheinen, für den Fall, dass ein System momentan nicht verfügbar ist, und er den Nutzern des Systems über diesen Weg den Grund und die voraussichtliche Ausfallzeit mitteilen möchte. Zudem muss er auch die Möglichkeit bekommen, Meldungen zu löschen.
Beim Pflegen von Systemmeldungen kann der System-Admin das System auswählen, zu welchem er eine neue Meldung hinzufügen möchte und die benötigten Daten eingeben.

4.2.2.9 User eines R/3-Systems anzeigen

Der System-Admin kann sich für R/3-Systeme die Nutzer anzeigen lassen, die in einem gewissen Zeitraum oder sich irgendwann mal an diesen Systemen angemeldet haben. Dadurch bekommt er auf einen Blick alle User des/der ausgewählten Systems/Systeme. Falls er plant das System zu einem bestimmten Zeitpunkt zu warten oder abzustellen, kann er dadurch z.B., dass er die Nutzer des Systems kennt, sich mit ihnen auch direkt in Kontakt setzen und dies ihnen mitteilen.
Der System-Admin kann beim Selektieren folgendes auswählen:

- das System, das angezeigt werden soll,
- das letzte Login-Datum und –Uhrzeit und
- den Mandanten.

4.2.2.10 Transporte eines R/3-Systems anzeigen

Der System-Admin kann sich alle Transporte zu einem R/3-System anzeigen lassen. Dies ermöglicht es ihm z.B., zu sehen welche Personen noch offene Transporte in diesem System haben. Für den Fall eines herannahenden festgelegten Freigabezeitpunkts, kann er dadurch die entsprechenden Personen sehen, die noch Transporte freigeben müssen und diese direkt ansprechen.
Der System-Admin kann beim Selektieren folgendes auswählen:

- Systeme, die angezeigt werden sollen,
- die Transportnummer,
- den Status der Transporte, die er sehen möchte,
- das Datum der letzten Änderung und
- die Mitwirkenden des Auftrags.

4.3 Informationsentwicklung

Es muss eine Dokumentation für den Monitor-Admin erstellt werden, in der seine Aufgaben und Möglichkeiten beschrieben sind. Auch der System-Admin muss eine Dokumentation erhalten, die seine Möglichkeiten beschreiben.
Für die Monitor-User ist eine Dokumentation nicht unbedingt erforderlich, da ihre Möglichkeiten nur auf das Anzeigen von Daten beschränkt ist und dies intuitiv nutzbar sein sollte.

4.4 Rahmenbedingungen

In diesem Abschnitt wird auf die Rahmenbedingungen des Systems eingegangen. Dazu gehören das Datenvolumen, die Performance, der Datenschutz, die Datensicherheit und die Portabilität.

4.4.1 Datenvolumen

Das Datenvolumen der bisherigen Monitor-Tabellen im CUE-System liegt zum 21.05.2002

- für die Loginzeiten der Benutzer bei 1.059.379,
- für die Systemmeldungen bei 2.791 und
- für Transportinformationen bei 6.549.420.

Für die Datensätze, die die Loginzeiten der Benutzer speichern, sind im Schnitt 3.000 Datensätze je Arbeitstag zu erwarten. Bei den Systemmeldungen handelt es sich um weniger Datensätze, etwa 30 pro Tag. Die zu erwartende Anzahl an Datensätzen für Transportaufträge ist im Schnitt 40.000 je Arbeitstag. Es werden allerdings nicht alle Daten, die aus den R/3-Systemen gelesen werden jeweils hinzugefügt, sondern die mit gleichem Primärschlüssel werden ersetzt. Dies bedeutet z.B. für die Loginzeiten der Benutzer, dass immer nur das letzte Login eines Benutzer in einem System in der Monitoring Datenbank gespeichert wird.

Dasselbe gilt für Transportaufträge. Jeder Auftrag wird nur einmal in der Monitoring Datenbank vorhanden sein, kann aber öfters aus R/3 selektiert werden, falls der Auftrag z.B. geändert wurde.

4.4.2 Performance

Die Performance soll für alle Bereiche, die in Abb. 4-1 beschrieben sind, getrennt betrachtet werden.

4.4.2.1 Systemquellen

Beim Selektieren aller verfügbaren Systeme mit ihren Daten (Hostname, Instanz, usw.), die beim späteren Scannen benötigt werden, muss die Selektion so gewählt sein, dass nur die benötigten Daten selektiert werden.

4.4.2.2 Quelldatensysteme

Beim Scannen der eigentlichen Daten müssen geeignete Selektionsoptionen angegeben werden, um die Datenbank, die die Selektion durchführt, nicht zu überlasten. Speziell sollten die Selektionsoptionen auf vorhandene Indizes der Tabellen gerichtet sein.
Um das Netzwerk nicht zu überlasten, sollten nicht zu viele Systeme parallel gescannt werden.

4.4.2.3 Monitoring-Service

Beim Monitoring-Service muss darauf geachtet werden, dass der Datenaustausch mit der Datenbank so gestaltet wird, dass sowohl die Last der Datenbank als auch die des Netzwerks möglichst klein gehalten wird. Davon ist besonders das Abgleichen der gesammelten mit den bereits vorhandenen Daten betroffen. Hier muss ein geeigneter Weg gefunden werden.
Beim Selektieren der gespeicherten Daten, zum Anzeigen, müssen sowohl die Selektionskriterien als auch die Datenbank (eventuell durch Indizes) passend gestaltet werden.

4.4.2.4 Präsentation

Die Kommunikation zwischen Präsentation und Frontend muss ebenso innerhalb einer für den Nutzer angenehmen Zeit ablaufen, wie die zwischen Präsentation und Monitoring-System auch.

4.4.2.5 Frontend

Das Darstellen der Daten im Browser muss zeitlich geeignet erfolgen. Sollte allerdings der Nutzer sich Daten in einem grossen Umfang anzeigen lassen, muss er auch damit rechnen, dass das Anzeigen mehr Zeit in Anspruch nimmt. Besonders, weil WebDynpro viel mit JavaScript arbeitet und das HTML-Rendering auf dem Client ausführt.

4.4.3 Datenschutz

Loginzeiten der User sind sensible Daten. Doch wird jeweils immer nur das letzte Login eines Nutzers gespeichert. Zudem wird bei SAP mehr auf Kollaboration als auf Überwachen gesetzt. Deshalb müssen keine umfangreiche Datenschutzmassnahmen vorgenommen werden, um diese Informationen zu schützen. Zudem könnten berechtigte Personen sich die Daten auch direkt in den Systemen anzeigen lassen.
Es ist möglich, die Daten beim RFC-Zugriff über SNC (Secure Network Communication) und beim Anzeigen über SSL (Secure Socket Layer) zu schützen. Diese beiden Techniken kommen teilweise aus einem anderen Grund zum Einsatz. SNC, da auf manche R/3-Systeme aus Sicherheitsgründen nur mit SNC zugegriffen werden kann. SSL, da SAP-Portale auch mit SSL installiert werden können. Trotzdem müssen die Daten nicht mit SSL an den User gesendet werden.

4.4.4 Datensicherheit

Da die RFC-Aufrufe über TCP/IP von statten gehen, kann der Verlust oder die Verfälschung der Daten während der Übertragung nahezu ausgeschlossen werden. Es müsste schon willentlich geschehen, dass die Daten verfälscht werden. Diese Gefahr besteht allerdings nicht, da das System nur innerhalb des SAP-Netzwerks genutzt wird.
Der Verlust der gelesenen Daten spielt keine allzu grosse Rolle, da die Daten in den R/3-Quellsystemen weiterhin vorhanden sind und nach jedem Lauf des Monitors komplett neu übertragen werden könnten. Sollte der Monitor allerdings weitere Daten zu den Lesevorgängen speichern, sollte für diese eine Backup-Lösung gefunden werden.
Die Änderung der Strukturen der SAP-Felder stellt, falls diese geändert wurden, ein Problem dar, da diese dann falsch interpretiert werden könnten. In solch einem Fall muss der Service dieses Problem erkennen und den Monitor-Admin benachrichtigen, damit er die nötigen Massnahmen ergreifen kann.

4.4.5 Portabilität

Die Portabilität der eingesetzten Systeme wird an Hand der Bereiche aus Abb. 4-1 von rechts nach links betrachtet. Allerdings wird auf die Bereiche Systemquellen, Quelldatensysteme und Helpdesk-Systeme nicht näher eingegangen, da diese in dem vorliegenden Fall ausschliesslich aus R/3-Systemen bestehen. Die Bereiche Monitor-Service und Präsentation werden gemeinsam betrachtet, da diese auf demselben Rechner laufen sollen. Allerdings ist zu beachten, dass aus Portabilitätsüberlegungen keine Aussagen über die Performanz gegeben sind.

4.4.5.1 Monitor-Service und Präsentation

Der Monitor-Service soll vollständig in Java entwickelt werden. Dadurch kann eine gewisse Plattformunabhängigkeit erreicht werden, die nur durch die notwendigen Komponenten (Portal, J2EE-Server, Datenbank usw.) eingeschränkt wird.

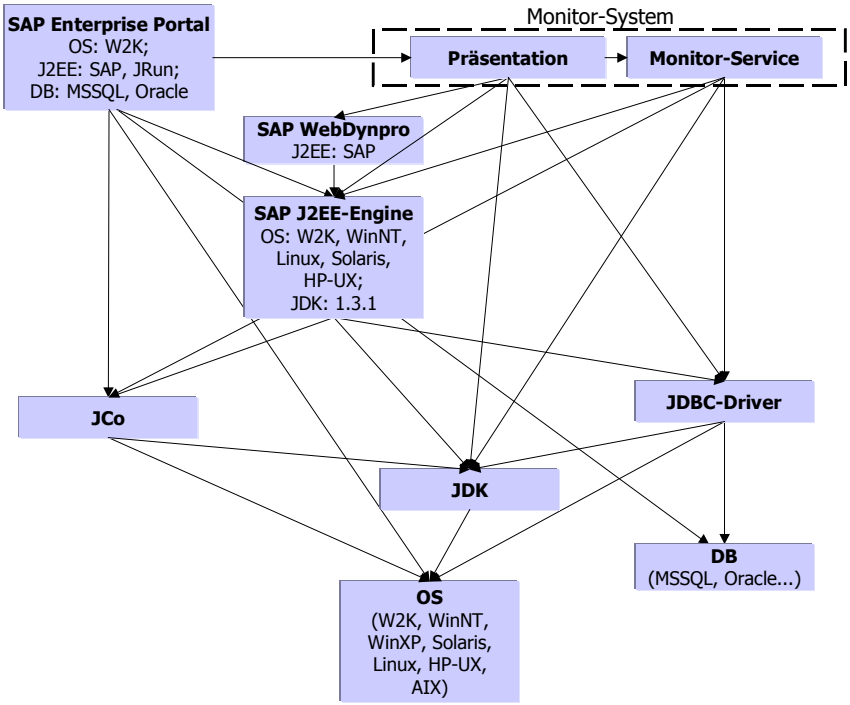

Abb. 4-3: Verbindung der Komponenten und ihre Einschränkungen (vgl. Abb. 4-2)

Aus Abb. 4-3 lässt sich erkennen welche Komponenten für das Monitor-System benötigt werden und mit welchen anderen Komponenten sie in Verbindung stehen. Dabei wurde bei

den Komponenten, die wesentliche Einschränkungen haben, diese mitangegeben. Hieraus lässt sich erkennen, dass als Betriebssystem ausschliesslich Win2000 zum Einsatz kommen kann, da nur dieses vom Enterprise Portal 5.0 unterstützt wird. Als Datenbank ist nur die Auswahl zwischen MS SQL-Server und Oracle möglich, da nur diese beiden vom Portal unterstützt werden. Die Auswahl des J2EE-Servers wird durch das Verwenden des WebDynpros auf die SAP J2EE-Engine eingeschränkt. Diese benötigt das JDK in der Version 1.3.1.
Aus diesen Überlegungen lässt sich nun erkennen, dass es nur bei der Auswahl der Datenbank eine Alternative gibt. Allerdings wird hier die Portabilität nicht mehr weiter diskutiert, da der MS SQL-Server gegebenüber Oracle den Vorteil bietet, dass er bereits in der Abteilung vorhanden ist.
Wenn das Monitor-System nicht auf dem gleichen Rechner laufen soll wie das Portal, ändert sich die Situation. Dadurch würde sich die Unix-Welt (Linux, HP-UX, Solaris) als Host öffnen. Das gleiche gilt, wenn das Enterprise Portal 6.0, welches nicht mehr von Win2000 abhängig sein wird, zur Verfügung steht. Diese Möglichkeiten werden aber nicht weiterverfolgt, da auch bei unterschiedlichen Rechnern diese mit Win2000 laufen würden. Dies aus dem Grund, da diese Maschinen und das Wissen darüber bereits vorhanden sind. Zudem wird die Version 6.0 des Enterprise Portals nicht vor Ende dieses Projektes verfügbar sein.

4.4.5.2 Frontend

Für das Frontend muss erneut das Enterprise Portal 5.0 und WebDynpro betrachtet werden. Beim Portal geht es dabei darum, für welche Browser eine Darstellung garantiert wird. SAP WebDynpro taucht hier ebenfalls wieder auf, da WebDynpro das HTML-Rendering mittels JavaScript im Browser vornimmt.

Komponente / System	Browser					
	IE 5.0	IE 5.01 SP2	IE 5.5 SP2	IE 6.0	Netscape 4.7	Netscape 6.2+
Enterprise Portal 5.0		X	X	X	X	
WebDynpro	X	X	X	X		X

Abb. 4-4: Portabilitätsmatrix Frontend ([@SAP02c], [@SAP02d])

Aus dieser Matrix lässt sich erkennen, dass als Browser die Internet Explorer 5.01 SP2, 5.5 SP2 und 6.0 zum Einsatz kommen können. Für das Enterprise Portal müssen dazu alle Browser auf Windows-Systemen laufen. Zu WebDynpro liessen sich keine Angaben finden, ob die Browser auch auf Macintosh- oder Linux-Systemen laufen können.

4.4.6 Installation, Upgrade

Die Installation des Monitors muss sehr einfach von statten gehen, es muss allerdings keine Routine erstellt werden, die das Installieren automatisiert. Auch sollten Upgrades einfach möglich sein.

4.5 Testanforderungen

Folgende Punkte müssen getestet werden:

- Werden die richtigen Systeme aus CSN/CSR gelesen und auch die richtigen Daten dazu?
- Selektiert der Service die richtigen Daten aus den Quellsystemen?
- Sind die Selektionsoptionen zum Scannen der Systeme richtig gewählt?
- Ist der Datenabgleich zwischen Monitor-Service und MonitoringDB geeignet, um den Datentransfer möglichst klein zu halten?
- Funktioniert die Adaptivität des Service richtig?
- Sind die Selektionskriterien zum Anzeigen der Daten richtig gewählt?
- Bekommen die User die richtigen Daten angezeigt?
- Werden auftretende Fehler richtig behandelt und dargestellt?

4.6 Systemvoraussetzungen

Voraussetzung für den Monitor-Service sind:

(1) Windows 2000 SP2,
(2) MS SQL Server 2000
(3) jdk 1.3.1,
(4) SAP J2EE-Engine 6.20,
(5) SAP Enterprise Portal 5.0 (beinhaltet SAP JavaConnector 2.0),
(6) SAP WebDynpro und
(7) JDBC-Treiber für den MS SQL Server 2000.

4.7 Einschränkungen

Falls WebDynpro nicht rechtzeitig in einer zum Entwickeln geeigneten Version vorliegen sollte, wird die Präsentation provisorisch mittels Portal-Komponenten die JSPs mit Standard-HTML einbinden entwickelt. Dabei würde sich dann die Möglichkeit ergeben, dass die Daten auch auf Netscape 4.7 (vgl. Abb. 4-4) angezeigt werden können. Auf diese Möglichkeit wird allerdings verzichtet, da bei SAP der Internet Explorer Standart Browser ist.
Nachteil dieser Lösung ist, dass es nicht dem Standart entspricht. Zudem ist auf diese Weise kein Branding möglich.

5 Design

Das Design für dieses Projekt wird, wie die Spec, nach den SAP Vorgaben erstellt. Dazu wird ebenfalls ein Template [@SAP02h] bereitgestellt, dessen Struktur in dieser Arbeit übernommen wird.

5.1 Grobdesign

Das Grobdesign beschreibt, wie die in der Spezifikation beschriebenen Anforderungen umgesetzt werden.

5.1.1 Architektur und Verhalten

Das Monitor-System besteht aus zwei Hauptmodulen, dem Monitor-Service und der Präsentation. Diese verwenden drei weitere Module, den XML-Controller, den DB-Controller und den R/3-Controller. Der Monitor-Service besteht dabei aus dem Portal-Service und dem MonitorAgent. Die Präsentation besteht aus den Web-Applikationen (WebApps) und der MonitorAPI. Abb. 5-1 zeigt alle Module mit den jeweiligen Verbindungen untereinander.

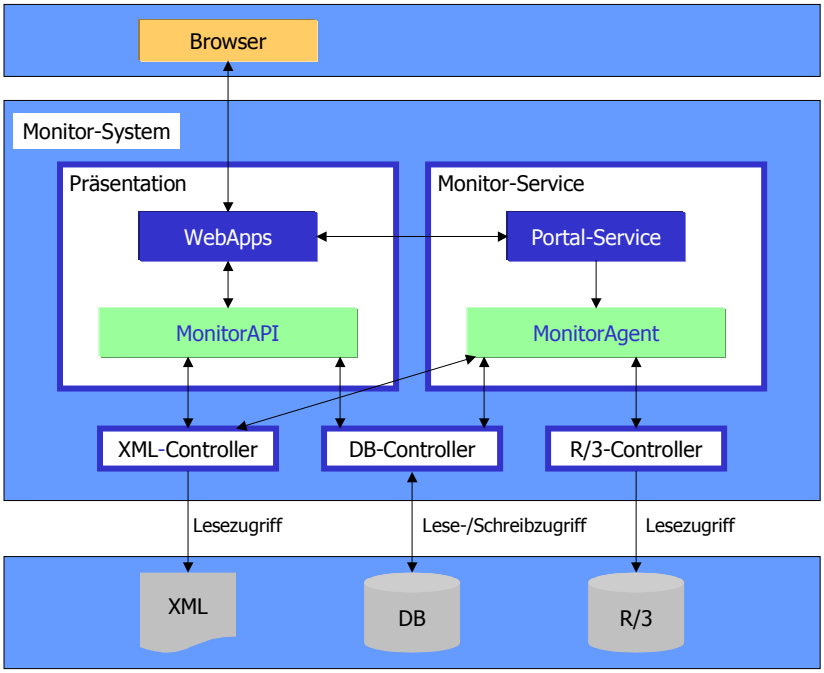

Abb. 5-1: Architektur des Monitor-Systems

Der **XML-Controller** stellt die Schnittstelle zu XML-Konfigurationsdateien dar. Er wird sowohl vom MonitorAgent als auch von der MonitorAPI aufgerufen, um die ihm angegebene XML-Datei zu lesen, zu verarbeiten und die Daten bereitzustellen.

Der **DB-Controller** ist die Schnittstelle zur MonitoringDB. Auch er wird vom MonitorAgent und von der MonitorAPI aufgerufen. Er liest und schreibt Daten aus den erforderlichen Tabellen der MonitoringDB.

Der **R/3-Controller** bietet eine Schnittstelle zu R/3-Systemen. Über ihn können Daten/Tabellen aus R/3 gelesen werden. Dazu wird SAP JCo verwendet. Über JCo wird der RFC-Funktiosbaustein `RFC_READ_TABLE` aufgerufen. Er greift auf die Systeme und Tabellen zu, für die ihn der MonitorAgent aufruft.

Die **Präsentation** des Systems besteht aus verschiedenen Web-Applikationen (WebApps) und der MonitorAPI. Die WebApps werden vom Nutzer über den Browser aufgerufen und leiten die Aufrufe an die MonitorAPI weiter. Diese liest aus einer XML-Datei Konfigurationen und kann über den DB-Controller die gesammelten Daten selektieren.

Der **Monitor-Service** ist das Herzstück des Monitor-Systems. Er liest die Daten aus den R/3-Systemen und speichert diese in einer eigenen Datenbank (MonitoringDB). Der Service kann dabei in zwei Bereiche aufgeteilt werden. Ein Bereich ist für das Lesen der Systemeinformationen über CSS aus CSN/CSR zuständig. Dabei werden die Systeme in der MonitoringDB gespeichert. Dieser Schritt wird durchgeführt damit die Systeme nicht bei jedem Scan-Vorgang über die Tabellen erneut aus R/3 gelesen werden müssen. Dadurch wird der Datenstrom zwischen R/3 und dem Monitor-Service und somit auch der Zeitaufwand des Service beim Scannen vermindert.
Der zweite Bereich des Monitor-Service scannt die verfügbaren Systeme. Dabei werden aus der MonitoringDB alle Systeme ausgelesen, die gescannt werden sollen und aus diesen die gewünschten Daten gelesen. Dabei ist es möglich, mehrere Systeme gleichzeitig zu scannen.
Der Monitor-Service wird dabei vom Portal-Service beim Starten des Portals in welchem er installiert ist, automatisch gestartet. Zudem können verschiedene WebApps über den Portal-Service das Lesen der Systeminformationen und das Scannen der Systeme mit unterschiedlichen Parametern starten. Den Hauptteil des Monitor-Services übernimmt der MonitorAgent, welcher das Lesen und Scannen verwaltet. Dabei nutzt er die drei Controller des Monitor-Systems.

5.1.1.1 Adaptivität des Monitor-Systems

Um das Monitor-System adaptiver als bisher zu gestalten wird der Secondchance-Algorithmus implementiert. Dieser Algorithmus kommt aus der Verwaltung des Virtuellen-Speichers bei Betriebssystemen. Er ist eine einfache Optimierung die zu einem FIFO (First In First Out) Algorithmus hinzugefüngt werden kann.
Mark Burgess beschreibt den Algorithmus folgendermassen (vgl. [@BURG99]):

> *Eine einfach Optimierung die zum FIFO Algorithmus hinzugefügt werden kann ist die folgende. Angenommen wir halten ein Refernz-Bit für jede Seite in der Seitentabelle. Jedesmal, wenn das Speicherverwaltungsmanagement eine Seite nutzt wird dieses Bit auf 1 gesetzt. Wenn eine neue Seite in der Tabelle aufgenommen werden muss, geht der Algorithmus, der die Seiten austauscht über alle Seiten und schaut sich das Bit an. Wenn es auf 1 gesetzt ist, setzt er es auf 0 und überspringt diese Seite. Die Idee dabei ist, dass bei Seiten die häufig genutzt werden das Bit oft gesetzt wird und diese dadurch nicht ausgelagert werden. Dieser Mechanismus erzeugt natürlich einen Overhead. Im Extremfall, wenn alle Seiten häufig genutzt werden muss der Seiten-Algorithmus über alle Seiten laufen und deren Bit auf 0 setzen, bevor er die Originalseite wieder findet. Selbst dann kann es sein, dass er keine Seite findet, die er ersetzen kann, falls das Bit erneut gesetzt wurde, während er sich die anderen Seiten angesehen hat. In solch einem Fall scheitert das Auslagerungssystem.*

Für dieses Projekt wird der Secondchance-Algorithmus folgendermassen angewendet. Für jedes System und jede Tabelle pro System die gescannt werden soll, wird ein zusätzliches Flag in der Datenbank gespeichert. Jedesmal wenn eine Tabelle eines Systems gelesen wurde wird das Flag dieser Tabelle auf 0 gesetzt. Sollte dann von diesem System und dieser Tabelle die Daten genutzt werden, wird dieses Flag auf 1 gesetzt. Das selbe gilt auch direkt für die Systeme. Sollte ein System gescannt worden sein, wird das Flag auf 0 gesetzt, und wenn

Daten dieses Systems aus der MonitoringDB genutzt wurden, wird das Flag wieder auf 1 gesetzt. Bei jedem neuen Scanvorgang des Monitor-Service selektiert er alle Systeme deren Flag auf 1 ist und scannt diese. Dabei werden allerdings auch immer nur die Tabellen des Systems gescannt, deren Flag ebenfalls auf 1 gesetzt ist.

5.1.1.2 R/3 Datenmodell

Die zugrundeliegenden Daten, die aus den verschiedenen R/3 Systemen gesammelt werden haben die in Abb. 5-2 aufgezeigten Verknüpfungen allerdings nicht über Fremdschlüssel-Beziehungen. Aus diesem Modell lassen sich auch bereits mögliche Auswertungen ableiten. Zum Beispiel könnte eine Auswertung erzeugt werden, in der man sich je System anzeigen lässt, welche Transporte noch von bestimmten Usern offen stehen.

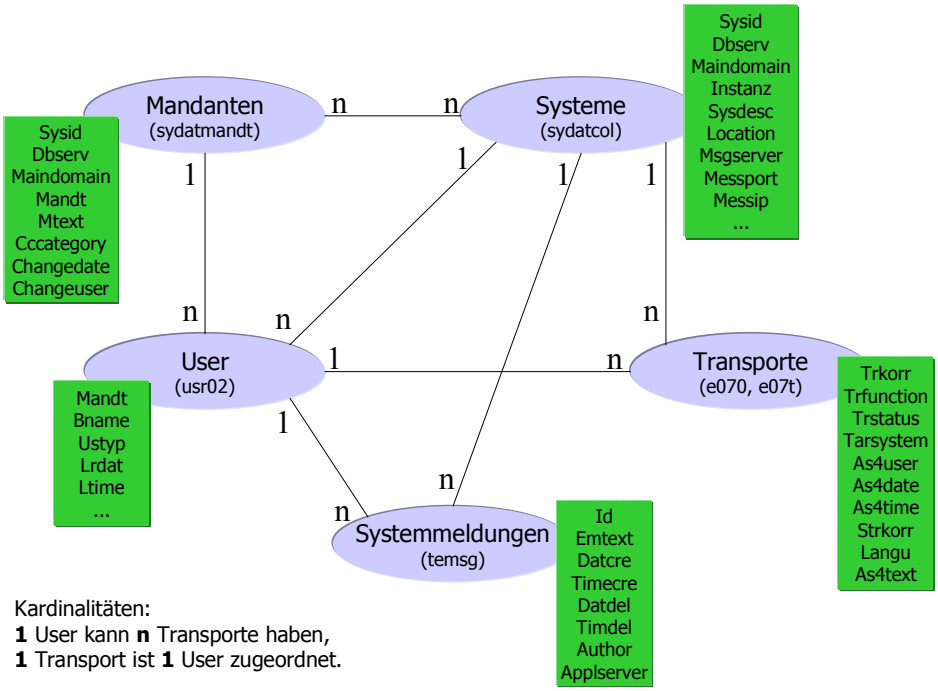

Abb. 5-2: R/3 Datenmodell der beteiligten Daten

5.1.2 Programmschnittstellen

Nachdem die Architektur beschrieben wurden, werden nun auch die Schnittstellen des Programms beschrieben.

5.1.2.1 XML-Beschreibung

Die XML-Beschreibung gibt der Anwendung an, wie auf das CSS-System zugegriffen werden kann. Zudem müssen die Login-Daten für das Scannen der Systeme festgelegt werden. Desweiteren muss der Lookup Name des Datenbankpools und aller verwendeten EJBs angegeben werden, mit denen die Anwendung arbeitet. Es müssen auch die Tabellen und Felder, die gelesen werden sollen, angegeben werden. Dabei muss festgelegt werden, welche R/3-Felder in welchen DB-Feldern gespeichert werden müssen, und von welchem Typ die DB-Felder sind. Es wird auch die Anzahl Threads angegeben, die genutzt werden sollen zum gleichzeitigen Scannen von mehreren Systemen. Zudem werden in dieser Datei alle Selektionen die bereitgestellt werden sollen festgelegt.

5.1.2.2 RFC_READ_TABLE

Zum Lesen der Daten aus den R/3-Systemen wird der Funktionsbaustein RFC_READ_TABLE verwendet. Man kann dabei angeben, welche Tabelle und auch welche Felder gelesen werden sollen. Zudem ist es möglich Selektionskriterien anzugeben. Der Funktionsbaustein liefert sowohl die Daten als einen String in einer Tabelle als auch Angaben zu jedem Feld (Startposition im Datenstring, Länge, Typ, Beschreibung) in einer weiteren Tabelle. Über diese Informationen kann dann jeder Eintrag der Datentabelle in den entsprechenden Feldern gespeichert werden.

5.1.3 Benutzungsschnittstellen

Um die Benutzungsschnittstellen darzustellen, werden die Abläufe aufgezeigt, die innerhalb des Systems geschehen.

5.1.3.1 Monitor-Service

Für den Monitor gibt es zwei Abläufe. Dies ist darauf zurückzuführen, dass es zwei Schritte gibt, die der Monitor-Service ausführt. Das Lesen der Systeme aus CSN/CSR und das Scannen der gewünschten Tabellen für jedes System.
Abb. 5-3 zeigt die Schritte, die im Monitor ablaufen, wenn die Systeme neu aus CSN/CSR gelesen werden.

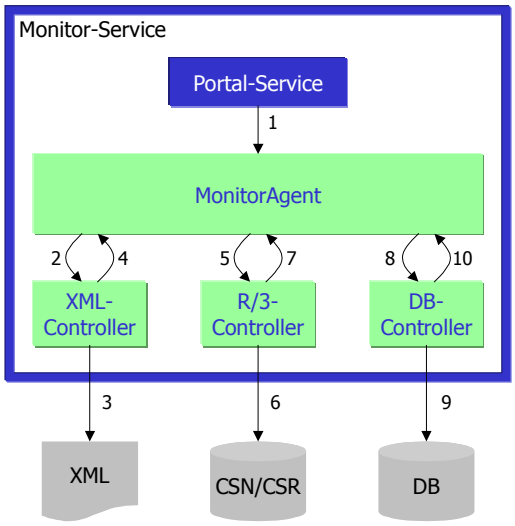

Abb. 5-3: Abläufe für das Lesen der Systeme

1. Über den Portal-Service wird die Methode zum Lesen der Systeme des MonitorAgents gestartet. Dabei können Selektionskriterien angegeben werden.
2. Diese ruft den XML-Controller mit der URL zur XML-Konfigurationsdatei auf.
3. Die XML-Datei wird gelesen.
4. Die gelesenen Informationen werden an den MonitorAgent zurückgeliefert. Konnte die XML-Datei nicht gefunden werden oder hatte sie Fehler wird dies gemeldet.
5. Der R/3-Controller wird aufgerufen um die den Kriterien entsprechenden Systeme aus CSN/CSR zu lesen.
6. Das CSN oder CSR System wird gelesen, entsprechend auf welches CSS verweist.
7. Die gelesenen Daten werden an den MonitorAgent zurückgeliefert.
8. Der MonitorAgent übergibt dem DB-Controller die gelesenen Daten, der diese in den entsprechenden Tabellen speichert.
9. Die Daten werden in die externe Datenbank geschrieben.
10. Der DB-Controller meldet dem MonitorAgent, ob das Speichern der Daten erfolgreich war.

Abb. 5-4 zeigt die Schritte, die im Monitor ablaufen, wenn die Tabellen von jedem einzelnen System gelesen werden.

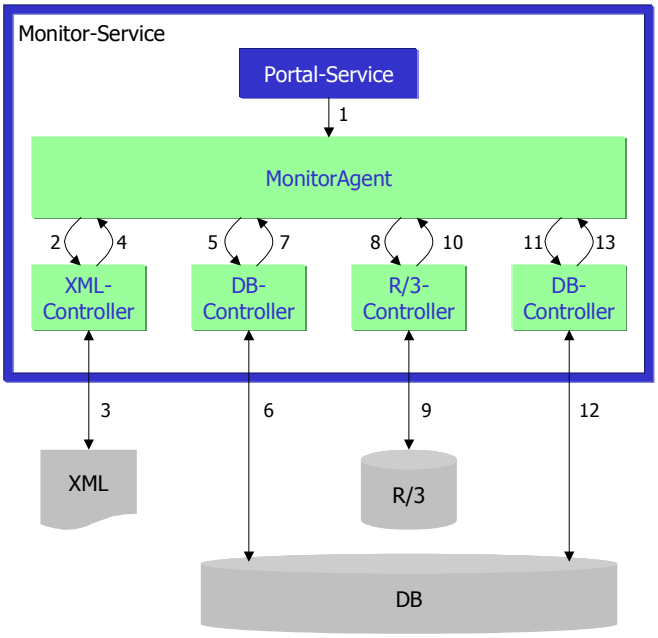

Abb. 5-4: Abläufe beim Scannen der Tabellen

1. Über den Portal-Service wird die Methode zum Scannen der Systeme des MonitorAgents gestartet. Dabei können Selektionskriterien angegeben werden.
2. Der XML-Controller wird aufgerufen mit der URL zur XML-Konfigurationsdatei.
3. Die XML-Datei wird gelesen.
4. Die gelesenen Informationen werden an den MonitorAgent zurückgeliefert. Konnte die XML-Datei nicht gefunden werden oder hatte sie Fehler wird dies gemeldet.
5. Der DB-Controller wird aufgerufen, um die Systeme die gescannt werden sollen aus der Datenbank zu lesen.
6. Die Systeme werden aus der Datenbank gelesen.
7. Die gelesenen Systeme werden vom DB-Controller an den MonitorAgent übergeben.
8. Der MonitorAgent ruft für jedes einzelne System und jede einzelne Tabelle den R/3-Controller auf, der diese in dem angegebenen R/3-System liest.
9. Die Daten werden aus dem R/3-System gelesen.
10. Die gelesenen Daten werden an den MonitorAgent zurückgeliefert.
11. Der MonitorAgent übergibt dem DB-Controller die gelesenen Daten, der diese in den entsprechenden Tabellen speichert.
12. Die Daten werden in die externe Datenbank geschrieben.
13. Der DB-Controller teilt dem MonitorAgent mit, ob das Speichern der Daten erfolgreich war.

5.1.3.2 Präsentation

Für die Präsentation der Daten gibt es ein Ablaufmodell. Dieses beschreibt die Schritte, die innerhalb des Systems durchlaufen werden, wenn die Daten angezeigt werden sollen.

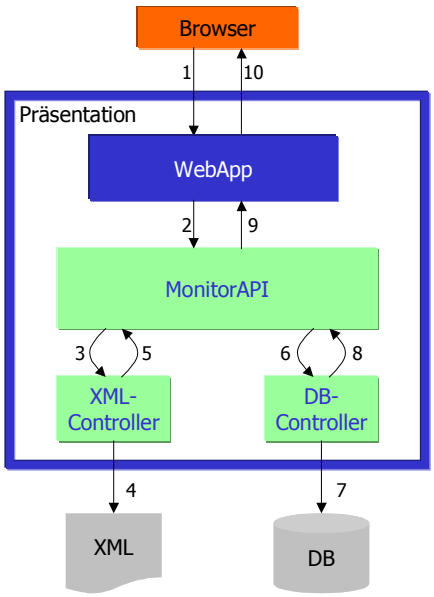

Abb. 5-5: Ablauf der Präsentation

1. Im Browser wird eine Web-Applikation (WebApp) aufgerufen, die einen Teil der gelesenen Daten aufbereiten und anzeigen soll. Der User kann bestimmte Einschränkungen angeben, die bei der Selektion berücksichtigt werden.
2. Die WebApp ruft die MonitorAPI auf und teilt ihr mit, welche Daten selektiert werden sollen.
3. Der XML-Controller wird zum Lesen der XML-Konfigurationsdatei aufgerufen. Diese enhält Informationen über die Datenbank (Lookup Name) und die Selektionen.
4. Die angegebene XML-Datei wird gelesen.
5. Es werden die gelesenen Daten an die MonitorAPI zurückgegeben. Falls ein Fehler beim Lesen auftrat oder die Datei nicht vorhanden war wird dies der API mitgeteilt.
6. Die MonitorAPI ruft den DB-Controller auf und teilt ihm mit, welche Daten gelesen werden müssen.
7. Die Daten werden aus der Datenbank gelesen.
8. Der DB-Controller gibt die Daten an die MonitorAPI zurück.
9. Die MonitorAPI sendet die gewünschten Daten zur aufrufenden WebApp.
10. Die WebApp bereitet die Daten auf und sendet diese dann an den Browser.

Die im Browser angezeigten Daten werden in Listenform aufbereitet.

5.2 Feindesign

In diesem Kapitel, werden die wichtigsten Punkte einzelnen beschrieben. Dabei geht es um das gleichzeitige Scannen von Systemen, dem Verbindungsaufbau zu den R/3-Systemen, dem Datenabgleich und Transaktionskonzept genauso, wie um die Datenbank und die einzelnen Module des Systems.

5.2.1 Gleichzeitiges Scannen von Systemen

Um das gleichzeitige Scannen von mehreren Systemen zu ermöglichen muss eine geeignete Möglichkeit gefunden werden. Dies könnte einerseits durch Message-driven Beans und andererseits über Threads ermöglicht werden.
Mittels Message-driven Beans könnte für jedes System eine JMS Nachricht an den J2EE-Server gesendet werden, welcher diese dann nacheinander an die Instanzen des Beans weiterleitet. Diese JMS Nachricht würde dabei die Übergabeparameter beinhalten. Nachdem für jedes System eine Nachricht erzeugt und gesendet wurde, würde dieser Teil des Systems wieder beendet sein. Das Scannen der Systeme läuft dann völlig unabhängig davon. Die Anzahl der gleichzeitigen Scans hängt dabei davon ab, wie viele Instanzen des Beans vom Container erzeugt werden können. Dies kann beim Installieren des Beans angegeben werden. Zum Ändern der Anzahl müsste direkt im J2EE-Server dieser Wert geändert werden.
Wenn das gleichzeitige Scannen mittels Threads implementiert wird, kann die Anzahl bei jedem Lauf neu angegeben werden. Dies kann sowohl über eine Konfigurationsdatei als auch direkt beim Start mit angegeben werden. Da allerdings die Threads nicht asynchron gestartet werden können, muss das Programm, welches diese pro System startet, solange laufen, bis für alle Systeme Threads erzeugt wurden.
Die Message-driven Beans wären durch ihr asynchrones Verhalten für dieses Projekt optimal. Allerdings muss auf die Verwendung von Threads zurückgegriffen werden, da es mit der SAP J2EE-Engine 6.20 nicht möglich ist Message-driven Beans zu entwickeln (vgl. Kapitel 2.3.4).

5.2.2 Verbindungsaufbau zu R/3

Der Aufbau einer Verbindung zu R/3 ist mit den vorhanden Daten zu den Systemen über drei Wege möglich. Erstens durch Angabe des Message-Servers und –Ports, zweitens über Message-Server und Systemname (SYSID) und drittens über den Datenbankservernamen (DBSERV) und die Instanznummer.
Die erste Variante dabei ist die mit den grössten Erfolgsausichten und wird daher als Standarteinstellung verwendet. Sollte es vorkommen, dass der Message-Port für ein System nicht angegeben ist, kann man über den Systemnamen zusätzlich zum Message-Server eine Verbindung herstellen. Sollte auch dies keinen Erfolg bringen, kann versucht werden über den Datenbankservernamen (DBSERV) und die Instanznummer eine Verbindung aufzubauen. Dies wird allerdings nur erfolgreich sein, für den Fall dass auf dem Datenbankserver gleichzeitig ein Gateway-Host installiert ist, der die Verbindung ermöglicht. Zusätzlich zu diesen Daten müssen noch ein korrektes Login und der entsprechende Mandant zu welchem eine Verbindung aufgebaut werden soll angegeben werden
Da für die vorhanden Systeme mit unterschiedlichen Logins eine Verbindung aufgebaut werden kann, werden alle dem Monitor bekannten Logins getestet, bis eines erfolgreich war. Dieses wird dann zu dem entsprechenden Mandanten gespeichert und beim nächsten Mal direkt verwendet.
Für den Fall, dass Fehler beim Verbindungsversuch auftreten müssen diese abgefangen und entsprechend behandelt werden. Vom Monitor-System werden dabei die folgenden Fehler abgefangen:

(1) No RFC authorization,
(2) User is not in validity date,
(3) User is locked,
(4) User not authorized,
(5) Name or password incorrect,
(6) Not authorized to logon,
(7) RFC logon timeout,
(8) Connect to SAP gateway failed und
(9) Connect to message server failed.

Die Reihenfolge, in der die Fehler aufgeführt sind, hat dabei eine wichtige Bedeutung. Es müssen alle möglichen Logins für ein System getestet werden. Nachdem alle getestet wurden und es konnte keine Verbindung aufgebaut werden, wird dies in der Datenbank zu den Daten des entsprechenden Systems gespeichert. Dabei wird der Fehler mit der kleinsten Nummer in der obigen Reihenfolge mit den dazugehörigen Login-Daten gespeichert. Sollte beispielsweise für User1 der Fehler 3 (User is locked) und danach für User2 der Fehler 6 (Not authorized to logon) auftreten wird in der Datenbank User1 und der Fehler 3 gespeichert. Wenn die Fehler 7 (RFC logon timeout) und 8 (Connect to SAP gateway failed) auftreten, werden keine weiteren Verbindungsaufbauversuche durchgeführt. Dies deshalb, da das System momentan nicht ansprechbar ist. Wenn Fehler 9 (Connect to message server failed) auftritt, werden die Verbindungsdaten geändert. Es werden Message-Server und –Port aus den Verbindungseigenschaften gelöscht und der Gateway-Host mit Instanznummer angegeben, sofern die Daten vorhanden sind. Danach werden wieder alle Logins durchgetestet. Sollten die Daten zum Gateway-Host und Instanznummer nicht vorhanden sein, wird der Versuch eine Verbindung herzustellen abgebrochen, da das System momentan nicht erreicht werden kann. Auch diese Fehler werden in der Datenbank gespeichert. Sollte keiner der genannten Fehler, sondern ein anderer auftreten, wird dieser in der DB gespeichert. Dies aber nur, wenn bei den weiteren Logins auch keiner von den angegebenen auftritt.

5.2.3 Datenabgleich

Der Abgleich der gelesenen Daten mit den bereits in der MonitoringDB vorhanden erfolgt für die Systemmeldungen anders als für die Loginzeiten und die Transporte. Bei den Systemmeldungen müssen jedes Mal alle vorhanden zuerst gelöscht werden, bevor die neuen hinzugefügt werden. Dies muss erfolgen damit in der MonitoringDB keine Meldungen vorhanden sind, die in R/3 bereits gelöscht wurden.
Bei den Loginzeiten und den Transporten hingegen müssen nur die tatsächlichen Änderungen in der MonitoringDB fortgeschrieben werden. Dafür muss ein Verfahren implementiert werden, welches dieses sinnvoll durchführt. Hierbei sind zwei Verfahren denkbar. Erstens es werden je System die gelesenen Daten mit denen in der Datenbank verglichen und nur die Änderungen an die Datenbank gesendet. Zweitens es werden alle gelesenen Daten an die Datenbank gesendet wobei für vorhandene Schlüssel eine Aktualiseierung erfolgt und für neue diese hinzugefügt werden.
Beim ersten Verfahren würden bei jedem Abgleich der Daten, alle bereits zu diesem System vorhandenen Daten aus der Datenbank gelesen. Daraufhin würden diese Daten im Hauptspeicher mit den neugelesenen verglichen und nur die Änderungen (Akutalisierungen und Neueinträge) an die Datenbank gesendet. Dies würde die Anzahl der Datentransfers zwischen Monitor und Datenbank gering halten. Allerdings wäre dieses Verfahren je nach Anzahl der bereits vorhanden Daten sehr Speicher intensiv.
Beim Durchführen aller Änderungen auf der Datenbank müsste zuerst für jeden Datensätz eine Aktualisierung versucht werden. Sollte dies nicht erfolgreich sein, da der Datensatz noch nicht vorhanden ist muss über einen zweiten Schritt dieser hinzugefügt werden. Dies würde im *worst case* bedeuten, dass 2n Statements an die Datenbank gesendet werden müssten (n=Anzahl der Datensätze). Dies würde bei grösseren Datenmenge eine hohe Netzbelastung bedeuten. Um dieses etwas zu verbessern, bietet JDBC die Möglichkeit über sogennante Batch-Aufrufe mehrere Statements auf einmal an die Datenbank zu senden.
Da für die Transporte und die Loginzeiten in der Datenbank eine sehr grosse Anzahl Datensätze vorhanden sind (vgl. Kapitel 4.4.1) würde für die erste Variante viel Hauptspeicher benötigt werden. Zudem werden aus R/3 nur tatsächliche Änderungen gelesen. Dies macht ein Vergleichen der neuen Daten mit den in der MonitoringDB vorhandenen völlig unnötig. Aus diesem Grund wird die zweite Variante für dieses Projekt verwendet.

Da diese, wie bereits erwähnt, mittels eines Batch-Aufrufs mehrere Statements auf einmal an die Datenbank senden kann, stellt sich die Frage, wie gross die Zahl der Statements gewählt werden soll. Dabei soll eine möglichst performante und von der Verbindungshäufigkeit annehmbare Lösung implementiert werden. Als Entscheidungshilfe wurden verschiedene Geschwindigkeitstests durchgeführt. Die Anforderung bestand darin 3237 Datensätze an die Datenbank zu senden und auszuführen. Dies wurde mit verschiedenen Batch-Grössen durchgeführt und jeweils die Gesamtdauer aufgezeichnet. Dabei ergab sich das in Tab. 5-1 aufgezeigte Ergebnis.

Batchgrösse [Anzahl Anweisungen]	1. Durchlauf [s]	2. Durchlauf [s]
250	34,302	33,820
500	31,777	32,258
1000	33,170	31,728
1500	32,298	31,337
2000	31,297	31,197
2500	31,117	32,909
3000	30,556	31,457
3500	31,307	32,288

Tab. 5-1: Performancetest mit Batchaufrufen zur Datenbank

Die beschriebenen Daten sind wie folgt zu interpretieren. Bei einer Batchgrösse von 250 Anweisungen für die Datenbank, wurden in einem ersten Durchlauf 34,302 Sekunden und in einem zweiten Durchlauf 33,820 Sekunden für die Abarbeitung aller 3237 Datensätze benötigt. Die hier ermittelten Daten müssen natürlich mit Vorsicht betrachtet werden. Einerseits sind die Zahlen nur Momentaufnahmen und können von Datenbanksystem und Rechner variieren. Zudem wird dabei keine Aussage darüber getroffen, wie stark die Grösse der Batchaufrufe die Verbindung zwischen J2EE-Server und Datenbank belastet.
Da sich die Zeiten für die Durchführung dieses Einzelfalles ziemlich gleichen wird die Möglichkeit impementiert die Grösse des Batches jederzeit zu ändern. Dadruch kann der Administrator festlegen, welche Batchgrösse gewählt werden soll.

5.2.4 Transaktionskonzept

Zur Sicherstellung der Datenkonsistenz muss ein Transaktionskonzept in Betracht gezogen werden. Dabei kann es zwei Situationen geben, die eine Dateninkonsistenz verursachen können. Zum einen kann dies passieren, wenn beim Aktualiseren der Datenbank ein Fehler auftritt. Dieser würde den Abbruch des Änderungsvorgangs bedeuten und es müsste ein *Rollback* durchgeführt werden. Für diesen Fall kann das Transaktionskonzept des EJB-Containers verwendet werden. Dabei kann man Beans oder Methoden von Beans bestimmte Transaktionstypen zuweisen. Dadurch muss sich der Entwickler nicht selbst um die Implementierung einer Transaktion kümmern. Der passende Typ hierfür ist die Option `Required`. Dadurch stellt der Container sicher, dass eine Transaktion gestartet wird.
Ebenfalls zu einer Dateninkonsitenz kommen kann es, wenn Daten aus einem System nacheinander aus R/3 gelesen werden, aber das Aktualisieren der MonitoringDB nicht in der gleichen Reihenfolge erfolgt. Diese Gefahr besteht jederzeit durch das Verwenden von Threads. Deshalb muss sichergestellt werden, dass ein System erst dann wieder gescannt werden kann, wenn der vorherige Scan komplett abgeschlossen ist. Um die Kontrolle darüber zu bekommen, gibt es wieder zwei Möglichkeiten. Zum einen im Programm selbst und zum anderen in der Datenbank. Da das Monitor-System allerdings in einer J2EE-Umgebung laufen soll ist nur die Lösung über die Datenbank sinnvoll. Der Grund dafür ist, dass der Monitor auf verschiedenen Servern mit derselben MonitoringDB gleichzeitig laufen könnte und somit nicht die gleichen Speicherressourcen nützen würden. Der Monitor auf Server1 hat keine

Kenntnis darüber was der Monitor auf Server2 durchführt. Deshalb müssen beide auf ein und dieselbe Informationsquelle zugreifen. Dafür bietet sich die gemeinsame MonitoringDB geradezu an. In ihr muss deshalb vermerkt werden, welches System gerade gescannt wird. Bevor ein System gescannt werden kann, muss in der MonitoringDB geprüft werden, ob bereits ein Scan für dieses System stattfindet. Sollte dies der Fall sein, wird dieses System im aktuellen Durchlauf ausgelassen. Dies wird über ein Flag realisiert, das auf 0 gesetzt wenn kein Scan für dieses System läuft und auf 1 wenn einer durchgeführt wird.

5.2.5 Datenbank

Die Datenbank besteht aus zwei unterschiedlichen Teilen. Auf der einen Seite sind Tabellen, die die Daten aus den R/3-Systemen wiederspiegeln (Datentabellen) und auf der anderen Seite Tabellen, die Daten über den Scanvorgang enthalten (Systemtabellen). Diese können u.a. für Protokollierungen verwendet werden.
Da dieses Projekt in erster Linie ein bereits bestehendes ABAP-Programm ersetzen soll, sind die zu lesenden Daten bekannt. Somit ergibt sich ein Teil der Struktur der zu erstellenden Tabellen aus den bisherigen. Zusätzlich werden Optimierungen der bisherigen Strukturen vorgenommen.
Um eine Übersicht zu gewinnen, wie erfolgreich und umfangreich der Scan-Vorgang ist, werden Daten dazu neu in der Datenbank abgespeichert. Zudem werden Fehler, die mit den R/3-Systemen und deren Tabellen auftreten dokumentiert. Diese Informationen stehen dann dem Monitor-Admin zur Verfügung, der die Daten sowohl anzeigen als auch löschen kann. Auch werden zusätzliche Parameter für die Adaptivität des Systems und das Transaktionskonzept benötigt.

5.2.5.1 Mapping ABAP-Datentypen zu MS SQL-Datentypen

Da die Datentypen in ABAP unterschiedlich zu denen des MS SQL Servers 2000 sind, muss hier ein Mapping vorgenommen werden. In Tab. 5-2 sind alle ABAP Dictionary Datentypen dargestellt und die dazugehörigen MS SQL Server Datentypen.

ABAP Dictonary Typ	ABAP Typ	Java Typ	JDBC Typ	MS SQL Server Typ
ACCP	N(6)	String	VARCHAR	varchar
CHAR n	C(n)	String	VARCHAR	varchar
CLNT	C(3)	String	VARCHAR	varchar
CUKY	C(5)	String	VARCHAR	varchar
CURR n,m	P((n+1)/2) DECIMAL m	BigDecimal	DECIMAL	decimal
DEC n,m	P((n+1)/2) DECIMAL m	BigDecimal	DECIMAL	decimal
DATS	D(8)	Date	DATE	datetime
FLTP	F(8)	double	FLOAT	float
INT1	X(1)	int	TINYINT	tinyint
INT2	X(2)	int	SMALLINT	smallint
INT4	X(4)	int	INTEGER	int
LANG	C(1)	String	VARCHAR	varchar
NUMC n	N(n)	String	NUMERIC	numeric
PREC	X(2)	Byte[2]	VARBINARY	varbinary
QUAN n,m	P((n+1)/2) DECIMAL m	BigDecimal	DECIMAL	decimal
RAW n	X(n)	byte[n]	VARBINARY	varbinary

TIMS	T(6)	Date	DATE	datetime
UNIT	C(n)	String	VARCHAR	varchar
VARC n	C(n)	String	VARCHAR	varchar
LRAW	X(n)	byte[n]	VARBINARY	varbinary
LCHR	C(n)	String	VARCHAR	varchar

Tab. 5-2: Mapping der Datentypen aus R/3 auf MS SQL Server (vgl. [@SAP02i], [SCHU02], [@SUN02c], [@MISO02])

Anhand dieses Wissens können die Tabellen, die aus R/3 übernommen werden entsprechend in MS SQL Server Tabellen abgebildet werden. Da in R/3 keine Fremdschlüssel-Beziehungen vorhanden sind, wird in den zu erstellenden Tabellen ebenso auf diese verzichtet.
Nachfolgend werden alle Tabellen die erstellt werden kurz beschrieben und ihre Struktur aufgezeigt. Dabei wird für die Felder die aus R/3 übernommen werden sowohl der Originaldatentyp aus R/3 angegeben, als auch der MS SQL Server Datentyp.

5.2.5.2 Systemdaten (dwp_allSystems)

Diese Tabelle wird zum bestehenden System hinzugefügt. In den bisherigen Tabellen des Monitors in R/3 wurden die Daten über das System, aus dem sie kommen (SYSID, DBSERV, MAINDOMAIN) teilweise direkt, teilweise gar nicht in den Datentabellen aufgenommen. Um diese Daten an einer zentralen Stelle zu speichern wird diese Tabelle mit zusätzlichen Informationen (z.B. MSGSERVER, MESSPORT) neu eingeführt. Diese Informationen werden für einen erfolgreichen Zugriff auf das jeweilige System benötigt. Diese Daten werden aus dem Grund in der MonioringDB gespeichert, da während eines Scan-Vorgangs nicht zuerst alle Systeme aus R/3 neu gelesen werden sollen, sondern aus der MonitoringDB. Dieses hat den Hintergrund, da sich die Daten der Systeme nicht so oft ändern, dass eine Selektion bei jedem neuen Scan-Vorgang nötig ist. Somit kann der Datenstrom zwischen R/3 und Monitor-System verkleinert werden. Zudem kann somit die Secondchance-Methode einfach über ein Flag implementiert werden. Zum Sicherstellen, dass immer nur ein Scan für ein System läuft wird ebenfalls ein Flag hinzugefügt.
In Tab. 5-3 sind alle Felder mit ihren wichtigsten Informationen angegeben, die zur Tabelle dwp_allSystems gehören.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Typ	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	30	R/3 Systemdaten: Domäne des Servers
INSTANZ		varchar	2	CHAR	2	R/3 Systemdaten: Instanznummer
MSGSERVER		varchar	30	CHAR	30	Messageserver
MESSPORT		varchar	12	CHAR	12	Port des Messageservers
ADMINUSER		varchar	10	CHAR	10	R/3 Systemmonitor: R/3 System Administration
THEMK1		varchar	20	CHAR	20	Servicesystem: Themenkreis
THEMK2		varchar	20	CHAR	20	Servicesystem: Themenkreis
THEMK3		varchar	20	CHAR	20	Servicesystem: Themenkreis

SCFLAG		smallint				Markierung für Secondchance
SCAN		smallint				Markierung, ob das System gerade gescannt wird

Tab. 5-3: Tabelle dwp_allSystems

5.2.5.3 Mandanten je System (dwp_allSystemsMandt)

Da nicht alle Daten die gelesen werden Mandanten-übergreifend sind müssen für diese alle Mandanten gelesen werden. Deshalb wird diese Tabelle erzeugt, welche die möglichen Mandanten für jedes System enthält. Da jeder Mandant einem System zugeordnet werden muss, werden die Schlüsseldaten eines Systems mit übernommen. Zusätzlich zu den Mandanten werden noch Daten gespeichert, die dokumentieren, ob Fehler beim Verbindungsaufbau aufgetreten sind und mit welchem Benutzernamen und Passwort bereits eine Verbindung hergestellt werden konnte. Dies ermöglicht es dem Monitor-System sich direkt mit dem richtigen Login anzumelden. Ohne vorher alle dem Monitor bekannten Logins nacheinander abzuarbeiten und dabei eventuell Fehler zu erzeugen. Tab. 5-4 zeigt die Struktur der Tabelle dwp_allSystemsMandt.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Typ	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	8	R/3 Systemdaten: Domäne des Servers
MANDT	PK	varchar	3	CLNT	3	Mandant
ERRORTYPE		smallint		-	-	Fehlertyp
ERRORTEXT		varchar	512	-	-	Fehlermeldungstext
R3USER		varchar	30	-	-	Username mit dem eine Verbindung hergestellt werden kann
R3PWD		varchar	30	-	-	Passwort zu dem Usernamen
LASTUPDATE		datetime		-	-	Letztesmal aus CSS gelesen
AFDAT		datetime		-	-	Letzter Ausführungstag
AFTIM		datetime		-	-	Letzte Ausführungszeit

Tab. 5-4: Tabelle dwp_allSystemsMandt

5.2.5.4 Loginzeiten der Benutzer (dwp_usr02)

In dieser Tabelle werden wie bisher in R/3 die Loginzeiten der Benutzer gespeichert. Dabei wird wieder das dazugehörige System mit angegeben.
Tab. 5-5 beschreibt die Struktur der Tabelle dwp_usr02.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Typ	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	30	R/3 Systemdaten: Domäne des Servers
MANDT	PK	varchar	3	CLNT	3	Mandant
BNAME	PK	varchar	12	CHAR	12	Benutzername im Benutzerstamm
USTYP		varchar	1	CHAR	1	Benutzertyp (A=Dialog, C=CPIC, D=DBC, O=ODC)
TRDAT		datetime		DATS	8	Letztes Login-Datum
LTIME		datetime		TIMS	6	Letzte Login-Uhrzeit

Tab. 5-5: Tabelle dwp_usr02

5.2.5.5 Systemmeldungen (dwp_temsg)

Die Tabelle dwp_temsg hatte bereits in R/3 die Systemdaten dabei. Deshalb kann diese komplett so übernommen werden. Es müssen nur die Datentypen entsprechend abgebildet werden.
In Tab. 5-6 ist die Struktur der Tabelle dwp_temsg beschrieben.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Type	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	30	R/3 Systemdaten: Domäne des Servers
ID	PK	varchar	10	CHAR	10	Characterfeld der Länge 10
EMTEXT		varchar	60	CHAR	60	Express-Message Text
DATCRE		datetime		DATS	8	Anlege-Datum einer Express-Message
TIMCRE		datetime		TIMS	6	Anlege-Uhrzeit einer Express-Message
DATDEL		datetime		DATS	8	Verfalls-Datum einer Express-Message
TIMDEL		datetime		TIMS	6	Verfalls-Uhrzeit einer Express-Message
AUTHOR		varchar	12	CHAR	12	Autor der Expressnachricht
APPLSERVER		varchar	20	CHAR	20	Server-Name
NOROW		varchar	3	CHAR	3	Feld der Länge 3 Bytes
CUROW		varchar	3	CHAR	3	Feld der Länge 3 Bytes
CLIENT		varchar	3	CLNT	3	Mandant

Tab. 5-6: Tabelle dwp_temsg

5.2.5.6 Transportaufträge (dwp_e070)

Auch bei der Tabelle dwp_e070 für die Transportaufträge werden nun alle Systemdaten mitaufgenommen. Zudem wird das Feld, welches den Sprachschlüssel enthält (LANGU) als Primary Key definiert, da sonst die unterschiedlichen Sprachen verloren gehen würden.
Tab. 5-7 beschreibt die Struktur der Tabelle dwp_e070.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Typ	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	30	R/3 Systemdaten: Domäne des Servers
TRKORR	PK	varchar	20	CHAR	20	Auftrag/Aufgabe
LANGU	PK	varchar	1	LANG	1	Sprachenschlüssel
TRFUNCTION		varchar	1	CHAR	1	Funktion des Auftrags/Aufgabe
TRSTATUS		varchar	1	CHAR	1	Status des Auftrags/Aufgabe
TARSYSTEM		varchar	10	CHAR	10	Transportziel eines Auftrags
AS4USER		varchar	12	CHAR	12	Autor der letzten Änderung
AS4DATE		datetime		DATS	8	Datum der letzten Änderung
AS4TIME		datetime		TIMS	6	Uhrzeit der letzten Änderung
STRKORR		varchar	20	CHAR	20	Übergeordneter Auftrag
AS4TEXT		varchar	30	CHAR	30	Kurzbeschreibung von Repository-Objekten
CLIENT		varchar	3	CLNT	3	Quellmandant eines Auftrags

Tab. 5-7: Tabelle dwp_e070

5.2.5.7 Anzahl gelesener Datensätze (dwp_datavolumes)

Diese Tabelle wird hinzugefügt, um das Datenvolumen, das aus R/3 gelesen und in der MonitoringDB gespeichert wird zu dokumentieren. Dadurch ergibt sich die Möglichkeit für zukünftige Änderungen oder Erweiterungen genaueres über das zu erwartende Datenvolumen einfach zu erfahren. Das Datenvolumen wird pro System und Tabelle gespeichert. Allerdings wird die Anzahl der Daten pro Tag kummuliert. Dies erfolgt, damit nicht unzählige neue Einträge hinzugefügt werden müssen.
Tab. 5-8 beschreibt die Tabellenstruktur der Tabelle dwp_datavolums.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Typ	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	30	R/3 Systemdaten: Domäne des Servers
MANDT	PK	varchar	3	CLNT	3	Mandant
AFDAT	PK	datetime		-	-	Ausführungsdatum

TABLENAME	PK	varchar	30	-	-	Name der gelesenen Tabelle
RECORDS		int		-	-	Anzahl der gelesenen Datensätze

Tab. 5-8: Tabelle dwp_datavolums

5.2.5.8 Secondchance auf Tabellenebene (dwp_tablesecondchance)

Um die Secondchance-Methode auf Tabellenebene einfach implementieren zu können, wird diese Tabelle hinzugefügt. Sie hält im maximal Fall zu jedem System alle Tabellen, die gescannt werden sollen. Dabei wird durch ein Flag (SCFLAG) gespeichert, ob diese Tabelle in den aktuellen Scan-Vorgang miteinbezogen werden soll oder nicht.
In Tab. 5-9 ist die Struktur der Tabelle dwp_tablesecondchance beschrieben.

Feldname	Key	SQL-Typ	SQL-Länge	ABAP-Typ	ABAP-Länge	Kurzbeschreibung
SYSID	PK	varchar	8	CHAR	8	System in dem sich das Original befindet
DBSERV	PK	varchar	8	CHAR	8	Rechnername
MAINDOMAIN	PK	varchar	30	CHAR	30	R/3 Systemdaten: Domäne des Servers
TABLENAME	PK	varchar	30			Name der Tabelle
SCFLAG		smallint				Markierung für Secondchance

Tab. 5-9: Tabelle dwp_tablesecondchance

5.2.6 Module

Das Gesamtsystem besteht wie unter 5.1.1 bereits kurz beschrieben aus mehreren Modulen. Diese werden nachfolgend alle getrennt betrachtet und beschrieben.

5.2.6.1 Monitor-Service

Der Monitor-Service besteht aus einem Portal-Service und dem MonitorAgent.

Portal-Service (Package: `com.sap.pct.devp.monitor.portal.service`)
Der Portal-Service wird benötigt, um den Monitor-Service direkt beim Start des Portals ebenfalls zu starten. Dabei ist der Portal-Service für das automatisch wiederholende Lesen und Scannen der Systeme zuständig. Um dies realisieren zu können, muss sowohl für das Lesen der Systeme als auch für das Scannen dieser ein Timer-Objekt erstellt werden. Diesem kann der erste Startzeitpunkt und der Wiederholungsrhythmus angegeben werden. Zusätzlich benötigt das Timer-Objekt einen Task der gestartet werden soll. Dies muss eine Instanz einer Klasse sein, die von der Klasse `java.util.TimerTask` abgeleitet ist. Die abgeleitete Klasse implementiert in der `run()`-Methode die Abläufe die ausgeführt werden sollen. Dies sind einmal das Lesen der Systeme über CSS und zum Zweiten das Scannen der einzelnen Systeme.
Desweiteren bietet der Portal-Service eine Schnittstelle für Portal-Komponenten zum Starten und Stoppen der einzelnen Tasks. Dabei können bei einem neuerlichen Start Auswahlparameter übergeben werden. Zudem ist es möglich zu bestimmen, ob ein neuerlicher Start sich wiederholen oder nur einamlig durchgführt werden soll.

MonitorAgent (Package: `com.sap.pct.devp.monitor.agent`)
Der MonitorAgent leitet den Ablauf des Monitor-Service. Er ist für das Lesen und Scannen der Systeme zuständig. Der MonitorAgent liest aus einer XML-Konfigurationsdatei alle für einen erfolgreichen Durchlauf notwendigen Daten. Er bietet zwei Schnittstellen an, eine zum Lesen der Systeme aus CSS und eine zum Scannen der in der MonitoringDB vorhandenen Systeme. Dabei besteht die Möglichkeit bei beiden Schnittstellen Einschränkungen mit anzugeben. Diese Einschränkungen beziehen sich auf die Systeme die gelesen oder gescannt werden sollen. Der Einstiegspunkt des MonitorAgents wird als stateless Session Bean (`MonitorAgentBean`) implementiert. Als Bean deshalb, weil der Monitor in einer J2EE-Umgebung implementiert werden soll. Dabei wäre anstatt einem Session Bean auch ein Message-driven Bean denkbar. Da allerdings die Vorrausetzungen für ein Message-driven Bean nicht gegeben sind (vgl. Kapitel 2.3.4) wird auf die entsprechenenden Vor- und Nachteile nicht näher eingegangen. Dadurch dass der MonitorAgent gegenüber seinem Client, in diesem Fall dem Portal-Service, keinen Status verwalten muss wird ein stateless Session Bean gewählt.
Beim Scannen der Systeme muss der MonitorAgent das gleichzeitige Scannen von Systemen ermöglichen. Da dies über Threads erfolgt, muss er eine Klasse (`ScanOneSystemThread`) implementieren die von der Klasse `java.lang.Thread` abgeleitet ist. Vom `MonitorAgentBean` wird für jedes System eine Instanz dieser Klasse erzeugt. Die Thread-Klasse ruft ihrerseits das Bean auf, welches ein System scannt. Dieses Bean (`ScanOneSystemBean`) ist ebenfalls ein stateless Session Bean, da auch hier kein Status zwischen Client (Thread-Klasse) und Bean gehalten werden muss. Ein Grund warum die Funktionalität zum Scannen eines Systems als Bean realisiert wird ist, da dabei nicht für jedes System eine neue Instanz erzeugt wird. Der Container erzeugt maximal so viele Instanzen, wie es gleichzeitige Threads gibt. Nachdem eine Instanz wieder frei wird, kann sie dem nächsten Thread ihre Funktionalität bereitstellen.
Auch das Lesen der Systeme über CSS wird nicht direkt im MonitorAgentBean implementiert. Es wird dafür eine eigene Klasse (`ReadCSS`) erzeugt, die diese Funktionalität kapselt. Hierbei wird eine normale Java-Klasse einem Session Bean vorgezogen, da kein Vorteil durch das Verwenden eines Session Beans ersichtlich ist. Das Lesen der Systeme ist jedes mal nur ein einziger Aufruf aus dem `MonitorAgentBean` heraus und würde die zusätzlichen Kosten (u.a. Remoteaufruf) für ein Session Bean nicht rechtfertigen.

5.2.6.2 Präsentation

Auch die Präsentation des Monitor-Systems besteht aus zwei Bereichen. Zum einen den WebApps, die als Portal-Komponenten implementiert werden und der MonitorAPI.

Portal-Komponenten (Package: `com.sap.pct.devp.monitor.portal.comp`)
Die Portal-Komponenten werden in zwei Klassen implementiert. Zum einen der `MonitorAdmin` und zum anderen der Klasse `DataPresentation`.
Der Nutzer kann über den Browser das Starten und Stoppen des Monitor-Service verwalten. Dabei kann sowohl das Lesen der Systeme über CSS, als auch das Scannen der einzelnen Systeme gestartet und gestoppt werden. Dies erfolgt über die Klasse `MonitorAdmin`, welche die gewünschten Aktionen an den Portal-Service weiterleitet.
Die Klasse `DataPresentation` bietet alle Funktionen die zum Anzeigen und Ändern der Daten über den Browser benötigt werden. Dafür greift sie auf das MonitorAPI zu. Zudem kann über sie eine interne Fehlermeldung (IT/IBC) zu einem System angelegt werden. Dabei müssen über eine Konfigurationsdatei die dazu benötigten Daten angegeben werden.

MonitorAPI (Package: `com.sap.pct.devp.monitor.api`)
Die MonitorAPI bildet die Schnittstelle zwischen den Web-Applikationen und der Datenbank.

Sie stellt alle Datenselektionen bereit, die möglich sind. Zudem bietet es die Möglichkeit Daten zu ändern. Welche Selektionen und Änderungen an den aus R/3 gelesenen Daten möglich sind, wird dabei in einer XML-Konfigurationsdatei festgelegt.
Die MonitorAPI besteht aus einem stateless Session Bean (`MonitorAPIBean`). Als Session Bean deshalb, da einerseits es der zentrale Einstiegspunkt ist, um an die gesammelten Daten zu gelangen. Zum zweiten, damit die Daten auch einem breiten Spektrum von unterschiedlichen Clients über Systemgrenzen hinweg bereitgestellt werden können. Stateless, weil zwischen Client und Bean keine Daten gehalten werden müssen. Der Client ruft eine Methode auf, das Bean bearbeitet die Anfrage und liefert ein Ergebnis zurück. Danach kann der nächste Client bedient werden.

5.2.6.3 XML-Controller

Der XML-Controller (Package: `com.sap.pct.devp.monitor.xml`) stellt die Schnittstelle zu einer benötigten XML-Datei dar. Er ist für das Lesen und richtige Verarbeiten der Datei zuständig. Für den Fall, dass es Probleme mit der Datei gibt (nicht vorhanden oder Fehler beim Lesen) erzeugt der XML-Controller (`ParseMonitorXML`) eine Fehlermeldung. Die interpretierten Daten werden vom XML-Controller über geeignete Methoden zur Verfügung gestellt.
Der XML-Controller wird als normale Java-Klasse implementiert, da keine Gründe vorhanden sind, die eine Implementierung als Bean rechtfertigen würden.

5.2.6.4 R/3-Controller

Der R/3-Controller (Package: `com.sap.pct.devp.monitor.r3`) bildet die Schnittstelle zu den R/3-Systemen. Er ist für das korrekte Lesen der Tabellen und Daten über den Funktionsbaustein RFC_READ_TABLE in R/3 zuständig. Der Zugriff erfolgt über die JCo Schnittstelle. Das Erstellen des JCO.Clients und das Verwalten der Verbindung erfolgt dabei aus Gründen der einfacheren Wiederverwendbarkeit bereits im MonitorAgent. Dies ist darauf zurückzuführen, da im MonitorAgent an zwei Stellen eine Verbindung hergestellt werden muss und dabei auf zwei unterschiedliche Arten. Aber an beiden Stellen wird dann dieselbe Klasse zum Lesen der Daten aus R/3 verwendet.
Da ein JCO.Client-Objekt nicht serialisierbar ist, kann der R/3-Controller auch nicht als Bean implementiert werden. Er benötigt das JCO.Client-Objekt und das Übergeben dieses wäre sonst nicht möglich. Deshalb ist der R/3-Controller als normale Java-Klasse implementiert. Dabei gibt es zwei Klassen, `RfcReadTable` und `ReadTable`. `RfcReadTable` implementiert und kapselt den Funktionsbaustein RFC_READ_TABLE. `ReadTable` wird vom MonitorAgent aufgerufen und bekommt die Daten übergeben, die für den Funktionsbaustein benötigt werden und setzt diese. Dabei legt sie eine Instanz der Klasse `RfcReadTable` an. Zusätzlich wandelt sie die aus R/3 gelesenen Daten in eine geeignete Struktur um und gibt diese an das rufende Programm zurück.

5.2.6.5 DB-Controller

Der DB-Controller (Package: `com.sap.pct.devp.monitor.dao`) bildet die Schnittstelle zur externen Datenbank. Er vereint alle Klassen und Methoden, die benötigt werden um die Daten in die entsprechenden Tabellen zu schreiben und sie zu lesen.
Hier stellt sich jetzt ganz konkret die Frage wie der Datenzugriff implementiert werden soll. Es bieten sich fünf Möglichkeiten an, die es zu diskutieren gibt:

- CMP Entity Beans,

- BMP Entity Beans,
- Session Beans mit JDBC,
- Message-driven Beans mit JDBC und
- normale Java-Klassen mit JDBC.

Bei CMP Entity Beans müsste für jede Tabelle ein Entity Bean erstellt werden. Hierbei würde der Container die komplette SQL-Logik implementieren und es müssten nur die gewünschten Finder-Methoden angegeben werden. Der Container würde für jede einzelne Zeile einer Tabelle eine eigene Instanz anlegen, wenn diese gebraucht wird. Da es sich hier um Massendaten handelt, würde dies einen grossen overload erzeugen, da die Tabellen nur aktualisiert werden und danach die Instanzen vorerst nicht mehr gebraucht würden. Der overload besteht darin, dass für jede einzelne Instanz vom Container die Methoden ejbCreate, ejbLoad, ejbStore, ejbActivate, ejbPassivate, ejbRemove usw. ausgeführt würden. Ein weiterer Punkt ist, dass bei einer Tabellenänderung das dazugehörige Entity Bean geändert werden müsste. Wollte man eine weitere Tabelle aus R/3 lesen und in der Datenbank mit abspeichern, müsste man zusätzlich zu der Tabelle auch noch ein CMP Entity Bean erzeugen.
BMP Entity Beans haben natürlich ähnliche Probleme wie CMP Beans bezüglich der Änderung von Tabellen und der Methoden die der Container automatisch aufruft. Allerdings kann man den Zugriff auf die Datenbank mittels JDBC-Code den eigenen Bedürfnissen anpassen und die Kommunikation mit der Datenbank somit flexibler gestalten. Aber bei dieser Form würde das Einbinden einer neuen Tabelle noch mehr Aufwand bedeuten, als bei CMP Beans, da dann für jede Tabelle wieder die komplette Datenbankzugriffslogik programmiert werden müsste.
Bei der Verwendung von JDBC in einem Session Bean gibt es keinen ähnlichen overload wie bei den Entity Beans, da die Daten direkt zur Datenbank gesendet werden und nicht vom Container verwaltet werden müssen. Zudem kann man hier direkt angeben, welche Felder gelesen oder geschrieben werden sollen und somit den Datenstrom zwischen Datenbank und System kleiner halten. Mittels JDBC ist eine Implementierung vorstellbar, bei der die Tabellen und Felder mit denen gearbeitet werden soll, über eine XML-Datei angegeben werden. Somit könnte das Programm sehr flexibel mit Änderungen der Datenbank oder mit neuen Tabellen arbeiten. Dies würde eine grosse Vereinfachung gegebenüber Entity Beans bedeuten, da praktisch nichts programmiert und installiert werden muss, sondern nur in der XML-Datei die entsprechenden Einträge vorgenommen werden müssen. JDBC hat natürlich gegenüber CMP Entity Beans den Nachteil, dass die Unabhängigkeit zur Datenbank nicht mehr so einfach und gut möglich ist. Trotz Verwendung von ANSII-SQL besteht die Gefahr, dass beim Wechsel auf ein anderes DB-System Änderungen vorgenommen werden müssen.
Message-driven Beans mit JDBC haben die gleichen Vor- und Nachteile wie Session Beans mit JDBC. Der einzige Unterschied ist, dass Message-driven Beans asynchron arbeiten.
Auch normale Java-Klassen die JDBC verwenden haben die bei Session Beans angesprochenen Vor- und Nachteile. Zudem haben sie noch den Vorteil, das sie nicht über einen Performanz-teuren Remote-Aufruf arbeiten.
Die Entscheidung fällt für dieses Projekt auf JDBC in Session Beans. Gegenüber den Entity Beans, da Session Beans sowohl bei der Kommunikation mit der DB als auch bei der Belastung des Containers eindeutig besser abschneiden. Zudem ist die Änder- und Erweiterbarkeit des Systems von der Datenbank-Seite einfacher möglich. Die etwas eingeschränkte Unabhängigkeit von der Datenbank wird in Kauf genommen. Dies, da dieses System vorerst auf nur einem Datenbank-System, dem MS SQL Server 2000 laufen muss.
Gegenüber den Message-driven Beans ist einmal anzuführen, dass deren Vorraussetzungen nicht gegeben sind (vgl. Kapitel 2.3.4). Desweiteren ist eine asynchrones Arbeiten nicht erwünscht, da der MonitorAgent über den Erfolg und Misserfolg informiert werden und auch Daten vom DB-Controller erhalten muss.

Alle bisherigen Begründungen würden auch für normale Java-Klassen gelten. Doch haben Session Beans auch ihnen gegenüber noch einen entscheidenden Vorteil zu bieten. Es handelt sich dabei um das vom EJB-Container bereitgestellte Transaktionsmanagement. Dieses wird, wie unter 5.2.4 beschrieben, benötigt. Da dafür in Session Beans keine einzige Zeile Code geschrieben werden muss erhalten sie den Vorzug gegenüber normalen Java-Klassen.
Dadurch wirft sich auch gleich die nächste Frage auf. Soll das Bean stateful oder stateless sein? Da zwischen den Aufrufen der einzelnen Methoden kein Status gehalten werden muss, fällt die Entscheidung auf ein stateless Session Bean.

Dieses Bean (`DataAccessBean`) hat neben dem Lesen und Schreiben der Daten noch eine weitere wichtige Aufgabe. Es muss die Daten die aus R/3 kommen in das richtige Format umwandeln. Dabei besonders betroffen sind die Datums- und Uhrzeitangaben. Alle Datumsfelder im R/3 haben die Form "`yyyymmdd`". Die Felder für die Uhrzeit entsprechen der Form "`hhmmss`" (vgl. [SCHU02]). Da der SQL-Server allerdings das Datum in der Form "`yyyy-mm-dd`" und die Uhrzeit in der Form "`hh:mm:ss`" erwartet, muss hier eine entsprechende Konvertierung stattfinden.

5.2.6.6 Überblick über den Monitor-Service

Abb. 5-6 soll einen Überblick über die wichtigsten Klassen und Beans und deren Aufrufstruktur geben, die vom Monitor-System verwendet werden. Dabei sind diese in die beschriebenen Module eingeteilt.

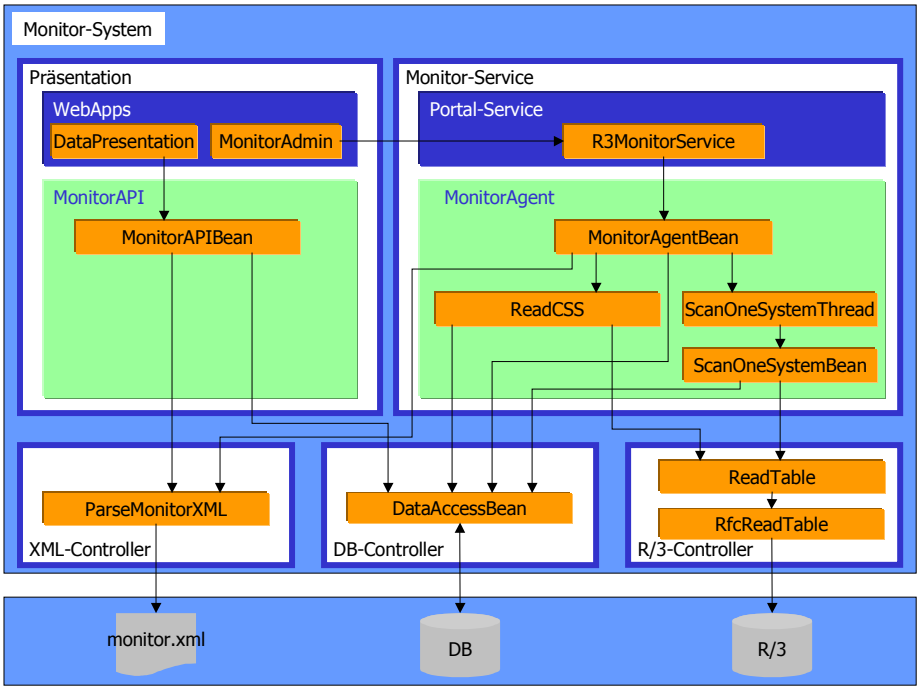

Abb. 5-6: Klassen und EJBs des Monitor-Systems

5.2.7 EnterpriseJavaBeans

Das Monitor-System besteht wie beschreiben aus mehreren EJBs. Diese werden nachfolgend etwas genauer beschrieben.

5.2.7.1 MonitorAgentBean

Das `MonitorAgentBean` bietet den Einstieg in den Monitor-Service. Es bietet, neben den Standart EJB-Methoden zwei weitere Methoden, eine zum Lesen der Systeme über CSS und

eine zum Starten des Scanvorgangs über die Tabellen der Systeme. Diese Methoden werden vom Portal-Service ausgeführt.
In der `ejbCreate()`-Methode wird als erstes über die Klasse `ParseMonitorXML` die XML-Datei eingelesen, die die benötigten Konfigurationsdaten enthält. Zudem werden die Referenzen zu den Home-Interfaces der in diesem Bean verwendeten Beans erzeugt und in Klassenvariablen gespeichert.
Die `selectSystemsToScan()`-Methode startet das Lesen der Systeme über CSS. Der Methode werden verschiedene Parameter übergeben. Bei diesen Parametern handelt es sich um Selektionskriterien für die Auswahl der Systeme die gelesen werden sollen. Darin wird eine Instanz der Klasse `ReadCSS` erzeugt. Daraufhin wird die Methode `selectSystems()` der Instanz aufgerufen, die das Lesen der Systeme durchführt. Dieser werden die Selektionskriterien und die Verbindungsoptionen übergeben.
Die zweite Methode des Beans, `startAgent()`, wird aufgerufen, um das Scannen der einzelnen Systeme zu starten. Dieser Methode werden Parameter übergeben über die die Auswahl der Systeme, die gescannt werden sollen, eingeschränkt werden kann. Es werden dann über die Methode `getSystemList()` des `DataAccessBean` die gewünschten Systeme aus der MonitoringDB gelesen. Nun wird für jedes dieser Systeme ein Thread-Objekt der Klasse `ScanOneSystemThread` erzeugt und sofort gestartet. Dabei werden dem Objekt die Daten übergeben, die für das Lesen des Systems benötigt werden. Allerdings wird nur eine bestimmte Anzahl von Threads gleichzeitig erzeugt und gestartet, um das System nicht zu überlasten. Nach Erreichen dieser Anzahl wartet die Methode bis einer der laufenden Threads beendet ist, bevor wieder ein neuer erzeugt und gestartet wird.

5.2.7.2 ScanOneSystemBean

Das `ScanOneSystemBean` ist für das Scannen aller gewünschten Tabellen und Mandanten eines einzigen Systems und das Speichern der gelesenen Daten in der MonitoringDB zuständig. Dies erfolgt in der Methode `scanSystem()`. Hier wird als erstes in der Datenbank nachgesehen, ob für dieses System bereits ein Scanvorgang läuft. Wenn nicht wird der entsprechende Flag gesetzt. Danach werden die nötigen Selektionskriterien für die einzelnen Tabellen ermittelt und angegeben. Dann versucht das Bean über ein `JCO.Client`-Objekt eine Verbindung zu dem gewünschten System herzustellen. Sollte dies nicht erreicht werden können, werden die Fehler die beim Verbindungsaufbau auftreten in der MonitoringDB gespeichert und die Methode beendet. Wenn ein Verbindungsaufbau zustande gekommen ist, werden alle Tabellen nacheinander über die Klasse `ReadTable` gelesen und über das `DataAccessBean` in der MonitoringDB gespeichert. Wenn der Scanvorgang komplett abgeschlossen ist, wird in der Datenbank der zu beginn gestetzte Flag wieder zurückgesetzt und die Methode beendet.

5.2.7.3 DataAccessBean

Das `DataAccessBean` ist die Schnittstelle zwischen Monitor-System und MonitoringDB. Jeder Zugriff auf die Datenbank läuft über dieses Bean. Dabei enthält dieses Bean Methoden, mit denen die Daten, die an die Datenbank gesendet werden sollen, in die richtige Form umgewandelt werden können (z.B. Zeichenketten innerhalb von ’’). Zudem bietet das Bean die Möglichkeit alle Systemtabellen, die noch nicht in der Datenbank vorhanden sind, automatisch anzulegen. Auch die Tabellen die aus R/3 gelesen werden, können in der Datenbank angelegt werden, wenn sie noch nicht vorhanden sind. Dadurch ist es nicht nötig, die Tabellen in der Datenbank selbst anzulegen, da dies alles von diesem Bean und somit dem Monitor ausgeführt werden kann. Wenn eine weitere Tabelle gescannt werden soll wird auch diese dann angelegt. Sie muss nur in der XML-Datei richtig angegeben werden.

5.2.7.4 MonitorAPIBean

Das `MonitorAPIBean` stellt die Schnittstelle zwischen den Web-Applikationen und den Daten der MonitoringDB dar. In der `ejbCreate()` Methode wird als erstes über die Klasse `ParseMonitorXML` die XML-Konfigurationsdatei eingelesen. Dann wird eine Referenz zum `DataAccessBean` Home-Interface erstellt. Die Web-Applikation ruft die Methode `getData()` des Beans auf. Diese Methode erhält von der Applikation eventuelle Einschränkungsparameter und eine Angabe darüber, welche Auswertung durchgeführt werden soll. Die Einschränkungsparameter werden verarbeitet und es wird ein select-String erzeugt der über das `DataAccessBean` an die Datenbank gesendet wird. Die Daten die das `DataAccessBean` zurückliefert werden entsprechend verarbeitet und der rufenden Web-Applikation übergeben.
Zum Aktualisieren und hinzufügen von Daten gibt es die Methoden `updateData()` und `insertData()`. Ihnen müssen die Änderungsdaten und die Bezeichnung der gewünschten Änderung übergeben werden. Dabei sind nur die Änderungen, die in der XML-Datei definiert sind, möglich.
Damit es möglich ist, manuell Tabellen und Systeme für einen Scan zu markieren gibt es noch die Methoden `setSystemSC()` und `setTableSC()`. Ihnen müssen die Daten übergeben werden, die notwendig sind um das entsprechende System oder die Tabelle zu finden und zu markieren. Wenn die Markierung eines Systems geändert wird, werden auch automatisch alle Tabellen für dieses System geändert.

5.2.8 Klassen

Nachfolgend werden noch die wichtigsten Klassen beschrieben die zum Monitor-System gehören.

5.2.8.1 R3MonitorService

Der Portal-Service, über den der Monitor-Service gestartet werden kann ist in der Klasse `R3MonitorService` implementiert. Er liest in einem ersten Schritt eine Konfigurationsdatei ein, in welcher die Daten zum Auffinden des `MonitorAgentBean` sowie die Startverzögerungs- und Wiederholungszeiten angegeben sind. Als nächstes wird über einen Lookup eine Referenz zum Home-Interface des Beans erzeugt. Daraufhin wird ein Objekt vom Typ `Timer` erstellt. Diesem Timer-Objekt wird beim Einplanen ein Objekt der Klasse `MonitorSelectTask`, die Startverzögerungszeit und die Zeit nach der der Task erneut ausgeführt werden soll übergeben. Die Klasse `MonitorSelectTask` ist dabei von `java.util.TimerTask` abgeleitet. Zum Scannen der Systeme wird ein neues Timer-Objekt erstellt. Auch dieses Objekt wird eingepant, mit einem Objekt der Klasse `MonitorScanTask` und den in der Konfigurationsdatei angegebenen Zeiten. Die beiden genannten Klassen sind innerhalb der Portal-Service-Klasse implementiert.
Zusätzlich zum automatischen Starten der Tasks, ist es über den Portal-Service auch möglich einmalige Tasks mit eventuell gewünschten Wertebereichen zu starten. Auch können die sich wiederholenden Tasks gestoppt und neue wiederholende Tasks (z.B. mit anderen Einschränkungen) gestartet werden.

5.2.8.2 ReadCSS

Die Klasse `ReadCSS` ist für das Lesen und Speichern der Systeme und ihren Daten aus CSN/CSR zuständig. Da das Speichern der gelesenen Daten über das `DataAccessBean` läuft, wird im Konstruktor eine Referenz zum Home-Interface des Beans erzeugt.
Der komplette Ablauf des Lesens und Speicherns wird in der Methode `selectSystems()` durchgeführt. Dieser Methode werden dabei alle wichtigen Daten (u.a. Einschränkungen,

Verbindungsoptionen) übergeben. Als erstes werden die übergebenen Einschränkungen zu den entsprechenden Feldern hinzugefügt. Daraufhin wird eine Verbindung (`JCO.Client`) zu CSN/CSR aufgebaut. Danach wird ein Objekt der Klasse `ReadTable` erzeugt und die Methode `readTableOwn()` aufgerufen. Dieser werden neben dem verbundenen `JCO.Client`, die Tabelle und die Felder die gelesen werden sollen übergeben. Diese Methode liefert nach erfolgreichem Ausführen eine Liste der gefundenen Systeme mit ihren Daten zurück. Diese Liste wird der Methode `storeSystems()` des `DataAccessBean` übergeben, welche die Daten in der DB speichert.

5.2.8.3 ScanOneSystemThread

Um ein gleichzeitiges Scannen der Systeme zu erreichen müssen wie bereits erwähnt Threads erzeugt werden. Beim Erzeugen dieser Threads muss ein Klasse angegeben werden, die von der Klasse `Thread` abgeleitet ist. Diese Klasse ist für das Monitor-System die Klasse `ScanOneSystemThread`. Sie muss neben einem Konstruktor noch die Methode `run()` implementieren, die ausgeführt wird, wenn die `start()`-Methode des erzeugten Thread-Objekts aufgerufen wird.
Über den Konstruktor werden der Klasse die zum Scannen eines Systems benötigten Daten übergeben und in private Klassenvariablen geschrieben. Zudem wird im Konstruktor ein Lookup gemacht, um eine Referenz auf das Home-Interface des Beans `ScanOneSystemBean` zu erzeugen, welches ein System scannt.
Die `run()`-Methode erzeugt eine Remote-Referenz über die im Konstruktor erzeugte Home-Referenz und ruft die Methode `scanSystem()` der Bean auf. Dabei werden alle Parameter die zum Lesen eines Systems benötigt werden übergeben.

5.2.9 Konfigurationsmöglichkeiten

Es gibt drei Stellen an denen das Monitor-System konfiguriert werden kann. Dies sind eine XML-Datei und zwei Properties-Dateien jeweils eine für den Portal-Service und eine für die Portal-Komponente. Die Portal-Komponente ist für alle WebApps zuständig. Die beiden Properties-Dateien enthalten dabei die Informationen, um den MonitorAgent und die MonitorAPI aufzufinden. Zusätzlich sind in der Portal-Service Property-Datei die Zeiten angegeben, nach deren Ablauf das Lesen der Systeme und das Scannen der Tabellen erfolgen soll. Dies kann jeweils getrennt angegeben werden. Bei der Portal-Komponente sind zusätzlich die Daten zum Anlegen einer IT/IBC-Meldung angegeben.
In der XML-Datei sind u.a. Angaben über den Zugriff auf CSS, die Tabellen die gelesen werden sollen und deren Selektionskriterien und den möglichen Selektionen die auf die gelesenen Daten möglich sind enthalten. Im Anhang ist sowohl die DTD als auch die XML-Datei enthalten.

6 Implementierung

Dieses Kapitel beschreibt nur Auszüge der Implementierung. Die Auswahl wurde danach getroffen, was für das Verständnis oder Weiterentwicklung durch andere Entwickler wichtig ist. Es wird daher beschrieben,

- wie die Tabellen mit ihren Feldern, die gelesen und gespeichert werden, innerhalb des Codings dargestellt werden,
- wie der Funktionsbaustein `RFC_READ_TABLE` mittels SAP JCo angesprochen wird,
- wie die Daten in die Datenbank geschrieben werden und
- wie das gleichzeitige Starten und Verwalten von mehreren Systemscans erfolgt.

6.1 Tabellen und Felder die gelesen werden sollen

Die Tabellen und Felder werden, wie im Design beschrieben, über eine XML-Datei dem Monitor-Service bereitgestellt. Dabei ist es möglich für jede Tabelle und jedes Feld verschiedene Eigenschaften anzugeben. Die Eigenschaften werden alle in `Properties`-Objekten gespeichert. Die Bezeichner der einzelnen Eigenschaften sind als Konstanten definiert. Beim Zugriff auf die einzelnen Eigenschaften wird immer die entsprechende Konstante verwendet. Um alle Felder einer Tabelle zusammenzufassen, werden die `Properties`-Objekte in `ArrayList`-Objekten gespeichert. Für jede Tabelle gibt es ein solches Objekt. Da die Anzahl der Tabellen, die gescannt werden können, variabel gehalten werden sollte, werden die `ArrayList`-Objekte in einem `HashMap`-Objekt gespeichert. Dabei wird der Tabellenname als Schlüssel hinterlegt. Dadurch ist sichergestellt, dass die Felder einer bestimmten Tabelle für diese wiedergefunden werden können. Die Felder werden alle in einer Liste gespeichert, damit sie an einer Stelle sind und man über sie navigieren kann.

Da die Tabellen ebenfalls Eigenschaften haben, werden auch diese in `Properties`-Objekten gespeichert. Auch hier sind wieder Konstanten definiert, über die der Zugriff erfolgt. Alle diese Objekte, die Tabellen darstellen, werden in ein `ArrayList`-Objekt gespeichert. Dadurch hat man alle Tabellen in einem Objekt über das navigiert werden kann. Zur Verdeutlichung s. Abb. 6-1.

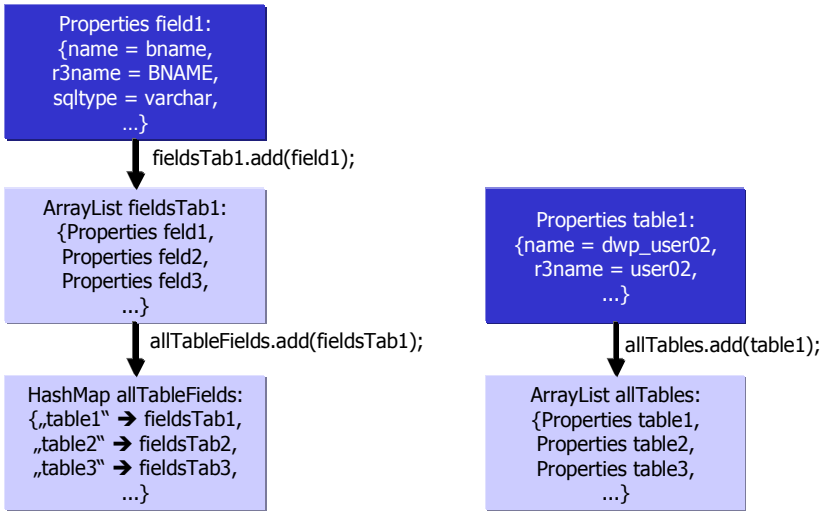

Abb. 6-1: Veranschaulichung der Implementierung der Tabellen und ihrer Felder

6.2 Lesen der Daten über SAP JCo und RFC_READ_TABLE

Um Tabellen aus den R/3-Systemen zu lesen, wird der Funktionsbaustein `RFC_READ_TABLE` verwendet. Um diesen in Java ausführen zu können, wurde die Klasse `RfcReadTable` entwickelt, welche die SAP JCo Schnittstelle verwendet. In der Klasse muss über die SAP JCo Schnittstelle ein `JCO.Repository`-Objekt erzeugt werden. Diesem wird dabei ein Bezeichner und der verbundene `JCO.Client` übergeben. Über dieses erzeugte Objekt, kann ein `IFunktionTemplate`-Objekt erhalten werden, wobei der Name des gewünschten Funktionsbausteins dabei übergeben werden muss. Über die Methode `getFunction()` des Template-Objekts bekommt man das `JCO.Function`-Objekt, das den Funktionsbaustein in Java darstellt. Über dieses Objekt kann auf die Import-, Exportparameter und die Tabellen, die der Funktionsbaustein verwendet zugegriffen werden. Der Programmausschnitt in Prog. 2-1 zeigt das Erstellen des `JCO.Function`-Objekts in Java. Die Variable `con` ist dabei das `JCO.Client`-Objekt, das eine offene Verbindung zu einem R/3-System hält. Die Variablen `m_Connection` (`JCO.Client`) und `m_Function` (`JCO.Function`) sind Klassenvariablen.

```
public void createFunction(JCO.Client con) throws JCO.Exception {
  try {
      m_Connection = con;
      String name = "RFC_READ_TABLE";
      JCO.Repository repository = new JCO.Repository("RfcConnection",
m_Connection);
      IFunctionTemplate ft =
      repository.getFunctionTemplate(name.toUpperCase());
      m_Function = ft.getFunction();
  } catch (JCO.Exception ex) {
      throw new JCO.Exception(ex.getGroup(),ex.getKey(),ex.getMessage());
  }
}
```

Prog. 6-1: Erstellen des `JCO.Function`-Objekts

Bevor der Funktionsbaustein ausgeführt werden soll, müssen ihm verschiedene Parameter übergeben werden. Dabei handelt es sich um den Namen der Tabelle, die gelesen werden soll, den Feldern die aus dieser Tabelle gelesen werden sollen und den Selektionskriterien die bei der Auswahl angewendet werden soll. Werden keine Felder explizit angegeben, werden alle Felder der Tabelle gelesen.
Der Name der Tabelle, die gelesen werden soll, muss dem Funktionsbaustein mittels des Importparamters `QUERY_TABLE` übergeben werden. Prog. 6-2 zeigt dabei die Methode die dafür entwickelt wurde.

```
public void setQueryTable(String queryTable) {
      m_QueryTable = queryTable;
      m_Function.getImportParameterList().setValue(m_QueryTable,
"QUERY_TABLE");
}
```

Prog. 6-2: Festlegen der Tabelle die gelesen werden soll

Da es in vielen Fällen nicht notwendig ist, alle Felder der gewünschten Tabelle zu lesen, besteht die Möglichkeit dem Funktionsbaustein mitzuteilen, welche Felder gelesen werden sollen. Dies erfolgt über die Tabelle `FIELDS`. Neben dem Namen des gewünschten Feldes (`FIELDNAME`) hat diese Tabelle noch die Parameter `OFFSET`, `LENGTH`, `TYPE` und `FIELDTEXT`. Diese Tabelle und ihre Felder werden beim Ausführen des Funktionsbausteins alle gefüllt. Mit diesen Informationen kann dann über die Daten, die gelesen wurden,

navigiert werden. Denn die gelesenen Daten stehen zeilenweise in einer Zeichenkette die 512 Zeichen aufnehmen kann. Über die `OFFSET`, `LENGTH` und `TYPE` Informationen jedes einzelnen Feldes, kann die gelieferte Zeichenkette in die einzelnen Felder aufgetrennt werden. Wenn nur bestimmte Felder der Tabelle gelesen werden sollen, kann also die Tabelle `FIELDS` mit den entsprechenden Feldnamen gefüllt werden. Dabei muss man nur die Namen der Felder angeben, die weiteren Daten sind nicht notwendig, da man diese eventuell auch von vorneherein nicht kennt. Der Codeausschnitt Prog. 6-3 zeigt, wie die Felder die gelesen werden sollen an die Funktion übergeben werden.

```
public void setFields(Collection fields) {
      m_Fields = m_Function.getTableParameterList().getTable("FIELDS");

      if (fields.size() > 0) {
            m_Fields.appendRows(fields.size());
            Iterator itFields = fields.iterator();
            m_Fields.setValue(itFields.next().toString(), "FIELDNAME");
            while (itFields.hasNext()) {
                  m_Fields.nextRow();
                  m_Fields.setValue(itFields.next().toString(),
"FIELDNAME");
            }
      }
}
```

Prog. 6-3: Setzen der Felder die gelesen werden sollen

Da es beim Lesen der Tabelle Einschränkungen geben kann, kann man auch diese angeben. Dies ist im Funktionsbaustein wieder über eine Tabelle (`OPTIONS`) möglich, deren einzigstes Feld `TEXT` ist und eine Zeichenkette der Länge 72 aufnehmen kann. Prog. 6-4 zeigt, wie die Selektionskriterien gesetzt werden, wobei die einzelnen Strings der Variable `options` nicht länger als 72 Zeichen sein dürfen.

```
public void setOptions(Collection options) {
      if (options.size() > 0) {
            m_Options =
m_Function.getTableParameterList().getTable("OPTIONS");
            m_Options.appendRows(options.size());

            Iterator itOptions = options.iterator();
            m_Options.setValue(itOptions.next().toString(), "TEXT");
            while (itOptions.hasNext()) {
                  m_Options.nextRow();
                  m_Options.setValue(itOptions.next().toString(), "TEXT");
            }
      }
}
```

Prog. 6-4: Setzen der Selektionskriterien, die bei der Selektion angewendet werden sollen

Nachdem die wichtigsten Daten der Funktion angegeben wurden, kann diese ausgeführt werden. Dies erfolgt über das `JCO.Client`-Objekt `m_Connection`, das eine offene Verbindung hält. Diesem wird beim Aufruf der Methode `execute()` einfach das `JCO.Function`-Objekt übergeben, das die angegebenen Daten enthält. Um in weiteren Methoden über die gelesenen Daten navigieren zu können, werden die Referenzen auf die Tabellen `FIELDS` und `DATA` des Funktionsbausteins in Klassenvariablen vom Typ `JCO.Table` gespeichert. Prog. 6-5 zeigt den beschriebenen Sachverhalt.

```
public void execute() throws JCO.Exception, JCO.AbapException {
      try {
            m_Connection.execute(m_Function);

            m_Fields =
m_Function.getTableParameterList().getTable("FIELDS");

            m_Data = m_Function.getTableParameterList().getTable("DATA");

      } catch (JCO.AbapException abapex) {
            throw new
JCO.AbapException(abapex.getKey(),abapex.getMessage());
      } catch (JCO.Exception ex) {
            throw new
JCO.Exception(ex.getGroup(),ex.getKey(),ex.getMessage());
      }
}
```

Prog. 6-5: Ausführen des Funktionsbausteins und Erhalten der Daten

Zum Bereitstellen der gelesenen Daten wurden zwei Methoden entwickelt. Eine, die immer eine Zeile zurückgibt, und eine die dem Aufrufer mitteilt, ob noch eine weitere Zeile vorhanden ist. Der Codeausschnitt Prog. 6-6 enthält diese beiden Methoden.

```
public String nextDataRow() {
      if (m_Data == null) {
            return "got no data";
      }
      m_Data.setRow(m_DataRow);
      m_DataRow++;
      return m_Data.getString("WA");
}

public boolean hasNextDataRow() {
      if (m_Data.getNumRows() > m_DataRow) {
            return true;
      } else {
            m_DataRow = 0;
            // um wieder auf die erste Zeile zu zeigen
            m_Data.setRow(m_DataRow);
            return false;
      }
}
```

Prog. 6-6: Bereitstellen der gelesenen Daten in einzelnen Zeilen

Um an die Feldnamen, deren Startposition und Länge zu kommen wurden drei weitere Methoden entwickelt. Zwei ähnliche wie bei Prog. 6-6 beschrieben und eine weitere, um an die Daten der verschiedenen Felder der Tabelle `FIELDS` zu gelangen. Dabei muss der Name des entsprechenden Feldes angegeben werden, dessen Daten man bekommen möchte. Der Codeausschnitt Prog. 6-7 zeigt alle drei Methoden.

```
public String nextFieldRow() {
      if (m_Fields == null) {
            return "got no fields";
      }
      m_Fields.setRow(m_FieldRow);
      m_FieldRow++;
      m_FieldCol = 0;
      String completeRow =
```

```
            m_Fields.getString("FIELDNAME")
                  + m_DelimiterString
                  + m_Fields.getString("OFFSET")
                  + m_DelimiterString
                  + m_Fields.getString("LENGTH")
                  + m_DelimiterString
                  + m_Fields.getString("TYPE")
                  + m_DelimiterString
                  + m_Fields.getString("FIELDTEXT");
      return completeRow;
}

public boolean hasNextFieldRow() {
      if (m_Fields.getNumRows() > m_FieldRow) {
            return true;
      } else {
            m_FieldRow = 0;
            // um wieder auf die erste Zeile zu zeigen
            m_Fields.setRow(m_FieldRow);
            return false;
      }
}

public String getFieldCol(String colName) {
      return m_Fields.getString(colName);
}
```

Prog. 6-7: Bereitstellen der Daten der gelesenen Felder

Die oben beschriebenen Methoden befinden sich alle in der Klasse `RfcReadTable`. Diese wird von der Klasse `ReadTable` genutzt, um die gewünschten Tabellen zu lesen. Bevor die `execute()`-Methode der Klasse `RfcReadTable` ausgeführt wird, werden die Selektionskriterien, die beim Aufruf angewendet werden sollen, zusammengebaut und übergeben. Ebenfalls wird der Name der Tabelle, und die Felder, die gelesen werden sollen, übergeben. Nachdem der Funktionsbaustein ausgeführt wurde und die Daten gelesen hat, werden diese in eine Liste vom Typ `ArrayList` gespeichert. Dabei wird für jede Zeile ein `Properties`-Objekt erstellt, das die Wertepaare (Feldname, Feldwert) für jedes Feld enthält, das gelesen wurde. Um die richtigen Werte der einzelnen Felder zu bekommen, muss mit den Daten, die in den `OFFSET` und `LENGTH` Feldern der Tabelle `FIELDS` stehen gearbeitet werden. Wie in Prog. 6-8 zu erkennen ist, stehen am Schluss dieser Funktion die gelesenen Daten, alle getrennt nach ihren Feldern, in der `ArrayList tableData` zur Verfügung.

```
RfcReadTable myScan = new RfcReadTable(jcoClient);

//...
// Selektionskriterien werden zusammengebaut und übergeben
// Tabellenname wird übergeben
// Felder die gelesen werden sollen,  werden übergeben
//...

myScan.execute();

Collection tableData = new ArrayList();
while (myScan.hasNextDataRow()) {
      Properties oneRow = new Properties();
      wa = myScan.nextDataRow();
      while (myScan.hasNextFieldRow()) {
            myScan.nextFieldRow();
            fname = myScan.getFieldCol("FIELDNAME");
```

```
            int offset = new
Integer(myScan.getFieldCol("OFFSET")).intValue();
            int length = new
Integer(myScan.getFieldCol("LENGTH")).intValue();
            int end = offset + length;
            if (offset <= wa.length() && end <= wa.length()) {
                  value = wa.substring(offset,end);
            }
            else if (offset <= wa.length() && end > wa.length()) {
                  value = wa.substring(offset);
            }
            else { value = ""; }
            oneRow.setProperty(fname,value);
      }
      tableData.add(oneRow);
      oneRow = null;
}
```

Prog. 6-8: Ablegen der gelesenen Daten in einer Liste, nach Feldern getrennt

Die erzeugte Liste mit den Daten wird dann an die aufrufende Funktion zurückgegeben und kann die Daten entsprechend weiterverarbeiten.

6.3 Speichern der Daten in der Datenbank

Das Speichern der Daten in der Datenbank erfolgt, wie bereits im Desgin beschrieben, innerhalb des `DataAccessBeans`. Dabei wird der Methode, die die Daten speichern soll, die Liste mit den gelesenen Daten übergeben. Dazu wird noch eine Liste mit allen Feldern und ihren Eigenschaften, die zu der zu speichernden Tabelle gehören übergeben.
Nach dem sichergestellt wurde, dass die Tabelle in der Datenbank vorhanden ist, in welche die Daten geschrieben werden sollen, läuft eine Schleife über alle aus R/3 gelesenen Zeilen. Danach wird für jede Zeile sowohl ein `update`-, als auch ein `insert`-Statement vorbereitet. Beim `update`-Statement wird bereits eine Bedingung fest angegeben. Es handelt sich dabei um den Primarykey des Systems, zu welchem die gelesene Tabelle gehört. Nun wird eine weitere Schleife erzeugt. Diese läuft über alle Felder, die zu der Tabelle gehören. Damit der Wert in der richtigen Form an die Datenbank übergeben wird, wird die Funktion `getRightDatatype()` aufgerufen. Diese ermittelt aus den übergebenen Daten (Werte der aus R/3 gelesenen Zeile und Eigenschaften des aktuelle Felds) in welcher Form der Wert in den Statements, die an die Datenbank gesendet werden, angegeben werden muss. Dabei werden u.a. alle Werte, die als Zeichenketten übergeben werden in Hochkomatas (’’) gesetzt, es werden Datums- und Uhrzeitangaben in die richtige Form umgewandelt usw. Die angepassten Werte werden dann dem `insert`- und `update`-Statement hinzugefügt. Dabei wird beim `update`-Statement geprüft, ob es sich beim aktuellen Feld um ein Schlüsselfeld handelt. Ist dies der Fall, wird der Feldname und der Wert im Bedingungsteil des Statements aufgenommen, ansonsten im Änderungsteil. Nachdem die Schleife über die Felder beendet ist und somit die komplette Zeile abgearbeitet ist, werden die `insert`- und `update`-Statements richtig zusammen gefügt. Das `insert`-Statement wird dann einer Liste mit allen anderen `insert`-Statements hinzugefügt, das `update`-Statement zum `Batch` des `m_stmt Statement`-Objekts. Der `Batch` hat den Vorteil, dass mehrere Statements auf einmal an die Datenbank gesendet werden können und nicht immer nur jeder einzeln. Als letztes wird die aktuelle Zeile aus der Liste gelöscht, um Speicher freizugeben. Der Codeausschnitt Prog. 6-9 zeigt, wie das beschriebene Verfahren in Java umgesetzt wurde.

```
Iterator itData = data.iterator();
while(itData.hasNext()) {
      propData = (Properties) itData.next();
      insertValue = "";
      Iterator itR3Fields = fields.iterator();
      updateValue = "update " + tableName + " set ";
      condition = " where " + MonitorSystemTables.MONITOR_SYSTEM_TAB_SYSNR
            + " = " + sysNr + " ";
      dbfields = "";
      while(itR3Fields.hasNext()) {
            Properties propField = (Properties) itR3Fields.next();
            dbfields = dbfields  + ", " +
propField.getProperty(IMonitorFieldProp.MONITOR_FIELD_PROP_DBNAME);
            value = this.getRightDatatype(propData,propField);
            insertValue = insertValue + "," + value;
            if
(propField.getProperty(IMonitorFieldProp.MONITOR_FIELD_PROP_KEY) != null) {
                  if (condition.indexOf("=") != -1) {
                        condition = condition + " and ";
                  }
            condition = condition
                        + propField.getProperty(
                        IMonitorFieldProp.MONITOR_FIELD_PROP_DBNAME)
                        + " = " + value;
            } else {
                  updateValue = updateValue
                        + propField.getProperty(
                        IMonitorFieldProp.MONITOR_FIELD_PROP_DBNAME
                        ) + " = " + value + ", ";
            }
      }
      insertValue = insertValue + ")";
      insertValue = "insert into " + tableName
      + " (" + MonitorSystemTables.MONITOR_SYSTEM_TAB_SYSNR
            + dbfields + ") values (" + sysNr + insertValue;
      updateValue = updateValue.substring(0,updateValue.lastIndexOf(",")) +
condition;
      collInsertBatch.add(insertValue);
      m_stmt.addBatch(updateValue);
      itData.remove();
}
```

Prog. 6-9: Erstellen der SQL-Statements für die gelesenen Daten

Nachdem alle Statements erstellt und teils im `Batch`, teils in einer Liste gespeichert wurden, können nun die Statements an die Datenbank gesendet werden. Dabei werden als erstes alle `update`-Statements zur Datenbank gesendet. Dabei wird ein `int Array` zurückgegeben, in dem für jedes einzelne Statement der Erfolg gespeichert wird. Wenn ein `update`-Statement erfolgreich durchgeführt werden konnte, der Eintrag, der aktualisiert werden sollte war vorhanden, wird dafür eine `1` gesetzt. War das Statement fehlerfrei, aber ohne Erfolg, wird eine `0` dafür gesetzt. Als nächstes wird dieses `Array` Zeile für Zeile durchgegangen und geprüft, ob die einzelnen Statements erfolgreich waren. Für jede Zeile, die durchgegangen wird, wird auch die Liste mit den `insert`-Statements durchgegangen. Da sowohl der `Batch` mit den `update`-Statements als auch die Liste mit den `insert`-Statements in der selben Reihenfolge sind, ist es dadurch möglich das entsprechende `insert`-Statement zu dem `update`-Statement zu erhalten. Wenn nun ein `update`-Statement nicht erfolgreich war, wird das entsprechende `insert`-Statement, in einen neuen `Batch` gespeichert und aus der Liste gelöscht um wiederum Speicher freizugeben. Nachdem alle Einträge des `Arrays`

durchgegangen wurden, wird, wenn nötig, der `Batch` mit den `insert`-Statements an die Datenbank gesendet und die bisher nicht vorhanden Einträge werden hinzugefügt. Prog. 6-10 stellt den Java-Code dar, der das Beschriebene implementiert.

```
int [] updateCounts = m_stmt.executeBatch();
m stmt.clearBatch();
Iterator itInsertBatch = collInsertBatch.iterator();
boolean boolDoInsert = false;
for (int a=0; a < updateCounts.length; a++) {
// wenn update eine 0 zurückgibt, ist der eintrag noch nicht vorhanden
// muss deshalb eingefügt werden.
      if (updateCounts[a] == 0) {
            String insert = (String) itInsertBatch.next();
            m_stmt.addBatch(insert);
            itInsertBatch.remove();
            boolDoInsert = true;
      } else {
            itInsertBatch.next();
            itInsertBatch.remove();
      }
}
if (boolDoInsert == true) {
      m_stmt.executeBatch();
}
```

Prog. 6-10: Absetzen der SQL-Statements an die Datenbank

Da das SQL `update`-Statement Einträge, die nicht vorhanden sind, nicht einfach hinzufügt, muss geprüft werden, ob ein `update`-Statement erfolgreich war oder nicht. Für jedes nicht erfolgreiche muss dann ein `insert`-Statement abgesetzt werden, um den entsprechenden Eintrag hinzuzufügen.
Diese Methode findet nur für die Loginzeiten und die Transporte Anwendung. Bei den Systemmeldungen werden alle zu dem aktuellen System gehörenden Daten zuerst aus der Datenbank gelöscht und dann die neuen komplett eingefügt. Dabei werden die `insert`-Statements in der selben Weise erzeugt. Welches der beiden Methoden für eine aus R/3 gelesene Tabelle angewendet werden soll, muss in der XML-Datei angegeben werden.

6.4 Simultanes Scannen von mehreren Systemen

Damit mehrere Systeme gleichzeitig gescannt werden können, müssen *Threads* implementiert werden. Allerdings sollen nicht alle Systeme, sondern nur eine bestimmte Anzahl von Systemen gleichzeitig gelesen werden. Dazu wurde die `int`-Variable `maxNumberThreads` implementiert, die die Anzahl der gewünschten Threads enthält. Als erstes wird eine Schleife über alle Systeme, die gelesen werden sollen, erstellt. Innerhalb dieser gibt es eine weitere `while`-Schleife, die sich solange wiederholt, bis sie durch ein `break`-Statement abgebrochen wird. Dies erfolgt, sobald für das aktuell zu scannende System ein Thread gestartet werden konnte. In einer `for`-Schleife, innerhalb dieser zweiten `while`, werden die Threads erzeugt und gestartet. Nachdem die Anzahl der möglichen Threads erreicht wurde, wird solange gewartet, bis einer der Threads beendet wird, und an dessen Stelle ein neuer erzeugt werden kann. Das Warten wird mittels der zweite `while`-Schleife ermöglicht. Damit auch andere Threads vom System CPU-Zeit bekommen, wird die Variable `waitCounter` hochgezählt. Nachdem sie einen bestimmten Wert erreicht hat, wird bei allen Threads die `yield()`-Methode aufgerufen. Diese ermöglicht anderen Threads, mit derselben Priorität, zu starten. Prog. 6-11 zeigt die Implementierung, die das Starten und Verwalten der Threads übernimmt.

```
while(itSys.hasNext()) {
      Properties sysProp = (Properties) itSys.next();
      int waitCounter=0;
      outer:
      while(true) {
            waitCounter++;
            for(int a=0;a<maxNumberThreads;a++) {
                  if (myThreadArray[a] == null) {
                        myThreadArray[a] = new ScanOneSystemThread(sysProp,
                              tables,allTablesFields,logins,startDate);
                        myThreadArray[a].start();
                        currentNumberThreads++;
                        break outer;
                  } else if (!myThreadArray[a].isAlive()) {
                        myThreadArray[a] = new ScanOneSystemThread(sysProp,
                              tables,allTablesFields,logins,startDate);
                        myThreadArray[a].start();
                        break outer;
                  } else if (waitCounter == 3000000) {
                        myThreadArray[a].yield();
                  }
            }
            if (waitCounter >= 3000000) {
                  waitCounter=0;
            }
      }
}
```

Prog. 6-11 Erstellen und Verwalten der Threads

7 Test

Bei der Entwicklung von Software-Systemen müssen fortlaufend Tests durchgeführt werden. Diese sollen die Qualität des Systems sichern und dabei helfen, Fehler in der Programmierung so früh wie möglich zu erkennen.
In diesem Projekt wurden mit Beginn der Entwicklung auch Tests durchgeführt. Dabei wurde jede neuentwickelte Funktion und Klasse erst einzeln gestestet, um deren Richtigkeit sicherzustellen. Nach erfolgreichem Einzeltest wurde das Zusammenwirken von einzelnen Teilen des Gesamtsystems getestet. Da die Entwicklung mit einem kleinen Teilbereich begonnen und Stück für Stück weitere Funktionalitäten hinzugefügt wurden, war es auch notwendig Klassen und Funktionen immer wieder zu testen. Dabei wurden die Tests in einem ersten Schritt lokal und ohne J2EE-Server durchgeführt. Dies ging so von statten, dass Instanzen von den benötigten Klassen oder Beans erstellt wurden und direkt die gewünschten Methoden aufgerufen und getestet wurden. In einem späteren Testzyklus wurde dann auch ein lokal installierter J2EE-Server miteinbezogen, um die entwickelten EJBs mit ihren Interfaces in ihrer benötigen Laufzeitumgebung zu testen.
Da das System hauptsächlich auf zwei Systeme zugreift, zum einen R/3 und zum anderen eine Datenbank, wurde auf die Zugriffsmechanismen ein besonderes Augenmerk gelegt. Beim Zugriff auf R/3 mittels SAP JCo musste in erster Linie darauf geachtet werden, dass das Erstellen einer Verbindung mit den richtigen Daten geschieht. Nachdem dieses sichergestellt war, musste auf die richtige Verwendung des Funktionsbausteins `RFC_READ_TABLE` geachtet werden, um die benötigten Daten aus R/3 zulesen. Die bereits im Vorgängersystem verwendeten Selektionskriterien mussten so übergeben werden, dass keine Fehler, die sog. Kurzdumps im R/3 System auslösen, auftreten.
Nachdem die einzelnen Teile des Systems erfolgreich getestet wurden, wurde das komplette System mit den Daten, die es im Produktivlauf bewältigen muss, getestet. Hierbei wurde das Augenmerk darauf gelegt, ob die richtigen Daten aus den R/3-Systemen gelesen werden und ob diese auch richtig abgelegt werden.
Als letzter Schritt mussten die Portal-Komponenten, die die gelesenen Daten aufbereiten und den Nutzern zur Verfügung stellen, getestet werden. Dabei war es wichtig, dass die gewünschten Daten mittels geeigneter Selektionen aus der Datenbank gelesen werden. Für die Fälle, in denen es möglich ist, über das Portal Daten zu manipulieren, muss der Nutzer über Erfolg und Misserfolg informiert werden. Beim Misserfolg muss dies über eine Fehlermeldung, mit welcher er die Probleme erkennen und lösen kann, erfolgen.

8 Produktivstart

Für einen erfolgreichen Produktivstart müssen die in Kapitel 4.6 beschriebenen Systemvoraussetzungen gegeben sein.
Wenn dies sichergestellt ist, muss als nächstes über das Administrationstool der J2EE-Engine ein Datenbankpool angelegt werden, der auf die im SQL Server vorhandene Datenbank verweist. Der Datenbankpool muss mit einem gültigen Benutzernamen und Passwort für die Datenbank versehen werden.
Ausgeliefert werden insgesamt fünf Dateien. Neben der XML-Datei und der dazugehörigen DTD-Datei wird noch das eigentliche Monitor-System als EAR-Datei, der Portal-Service als ZAR und die Portal-Komponenten als PAR-Datei ausgeliefert.
Die XML und DTD-Datei muss dann an einer Stelle gespeichert werden, auf welche die SAP J2EE-Engine Zugriff hat.
Als nächster Schritt muss die `r3monitor.ear` Datei in der J2EE-Engine installiert werden. Dabei müssen noch drei weitere Schritte vorgenommen werden:

(1) Beim `DataAccessBean` muss der angelegte Datenbankpool über den ResourceLink angegeben werden.
(2) Bei den Beans MonitorAgentBean und MonitorAPIBean muss der Pfad zur mitgelieferten XML-Datei über die bereits angelegten Umgebungsvariablen angepasst werden.
(3) Für die Applikation müssen Bibliotheks-Referenzen zu den Bibliotheken `sapjco.jar`, `logging.jar` und `inqmyxml.jar` angegeben werden.

Diese Einstellungen können alle über das Deploy-Tool der J2EE-Engine vorgenommen werden. Alternativ können die in (1) und (2) beschriebenen Einstellungen in den jeweiligen `ejb-jar.xml` Beschreibungen angegeben werden. Die in (3) beschriebenen Referenzen können auch in der Datei `<sapj2ee_home>\alone\managers\reference.txt` nach dem Schema "reference <applications_name> library:<library_name>" vorgenommen werden. In diesem Fall würde das wie folgt aussehen:

```
reference R3MonitorProduktiv library:jco
reference R3MonitorProduktiv library:logging
reference R3MonitorProduktiv library:inqmyxml
```

Nachdem die Applikation erfolgreich installiert werden konnte, wird als nächstes das Portal betrachtet. In einem ersten Schritt wird der Portal-Service installiert. Dies erfolgt durch einfaches kopieren der Service-Datei com.sap.pct.devp.monitor_service.zar in das Verzeichnis "`<portal_root>\WEB-INF\deployment`". Daraufhin muss das Portal neu gestartet werden, damit der Service installiert und gestartet wird.
Nächster Schritt ist es nun, die Portal-Komponenten zu installieren. Hierbei muss die Datei `com.sap.pct.devp.monitor_portal.par` im Portal über die entsprechende Seite hochgeladen werden. Daraufhin stehen alle Komponenten zur Verfügung. Aus diesen können dann iViews erzeugt werden, welche Seiten und Rollen zugeordnet werden können.
Sollte beim Aufruf des Service über die Komponenten, welche das Scannen und Lesen der Systeme aus R/3 ermöglichen, eine `ClassNotFoundException` auftreten, muss noch ein weiterer Schritt durchgeführt werden. Es muss die Datei monitor_serviceapi.jar, welche in der ZAR-Datei des Services enthalten ist, in das "`<portal_root>\WEB-INF\plugins\portal\lib`" Verzeichnis kopiert und das Portal neu gestartet werden. Dieser Schritt sollte im Normalfall nicht notwendig sein, hat sich allerdings bei der vorhandenen Umgebung als nötig erwiesen.

Nun sollten sowohl das Monitor-System als auch alle Portal-Komponenten ohne Probleme funktionieren. Allerdings gibt es noch einen Schritt, der eventuell notwendig ist. Dabei geht es darum, anfangs geeignete Einschränkungen beim Scannen der Systeme in der XML-Datei zu definieren. Dies deshalb, da es zumindest bei der SAP Systeme gibt, die grosse Datenmengen (z.B. >50000 Einträge) liefern können. Dies kann je nach Konfiguration der SAP J2EE-Engine ein Problem darstellen. Beispielsweise schafft der Produktivserver auf dem der Monitor installiert wurde (`devportal`) es nur maximal 66000 Einträge auf einmal zu verarbeiten. Dieses Wissen sollte auch dazu führen, anfangs das Scannen nur mit einem Thread zu starten und erst, wenn nur noch die neuen Daten gelesen werden, die Anzahl der Threads zu erhöhen.

9 Ausblick

Nachdem die erste Version produktiv läuft, lassen sich auch schon einige Dinge absehen, die Erweiterungs- und Änderungspotential haben. Diese lassen sich dabei in technologische und funktionale Bereiche aufteilen. Die technologischen sind dabei teilweise auf die fortschreitende Entwicklung bei SAP zurückzuführen.

9.1 Technologische Erweiterungen

Ein Hauptpunkt bei der technologischen Erweiterung ist die Entwicklung der Benutzeroberflächen mittels WebDynpro. Dies, da WebDynpro der Standard für die UI-Entwicklung bei SAP in Zukunft sein wird.
Ein weiterer Punkt für eine neue Version wäre die Portierung des Systems auf die J2EE-Engine 6.30 und mit ihr auf EJB2.0. Dies würde besonders durch die Verwendung der Local-Interfaces die Performance steigern, da der Overload durch die RMI-Aufrufe abgebaut werden könnte. Bis auf das MonitorAgentBean und das MonitorAPIBean könnten alle weiteren Beans lokal angesprochen werden. EJB2.0 würde es auch ermöglichen, die Threads durch Message-driven Beans zu ersetzen, die abgearbeitet werden, sobald Resourcen frei sind. Welches der beiden Methoden dabei die bessere ist, müsste durch genauere Untersuchungen und Tests ermittelt werden.
Auch könnte mit der J2EE-Engine 6.30 das von SAP entwickelte OpenSQL verwendet werden. Dadurch würde die Datenbankabhängigkeit vom Monitor-Systems zu OpenSQL verlagert werden. Wie sich herausgestellt hat, ist OpenSQL auch für die Version 6.20 verfügbar, allerdings nur für Testzwecke und in der Cluster-Version [@SAP02g]. Da OpenSQL bereits während der Entwicklung dieses Projektes zur Verfügung stand, dies allerdings übersehen wurde, hätten bereits Schnittstellen dafür vorgesehen werden können. Doch wurde immerhin versucht, bei der Verwendung von SQL-Statements nur die in SQL92 definierten zu verwenden. Bei der Portierung auf OpenSQL muss nun geprüft werden, wo die Unterschiede liegen und wie diese zu beheben sind.

9.2 Funktionale Erweiterungen

Ein wichtiger Punkt, um eine weitere Unabhängigkeit des Systems von den Daten, die gelesen werden sollen, zu erreichen, ist die vollständige Loslösung der Selektionskriterien vom Code. Im aktuellen System werden, um nur die neuesten Daten aus den R/3-Systemen zu lesen, ein Teil der Selektionskriterien anhand der vorhandenen Daten in der Datenbank erstellt. Dabei wird der aktuellste Datensatz für das jeweilige System/Mandant aus der Datenbank ermittelt und dessen Datum und Uhrzeit, welches aus R/3 gelesen wird, als Kriterium angegeben. Hier bietet sich die Möglichkeit dies über Konfigurationsdateien anzugeben und zur Laufzeit zu verändern, an.
Für den Fall, dass das Monitor-System fehlerhaft sein sollte, oder in bestimmten Systemen Fehler erzeugen würde, wäre es sinnvoll dem System-Admin eine Möglichkeit zu bieten selbst über das Portal eine interne Fehlermeldung aufzugeben, die an den Monitor-Admin geht. Auch wäre so eine Erweiterung für die Nutzer des Systems sinnvoll, da sie ja direkt sehen, ob die Daten die der Monitor bereitstellt korrekt sind, oder ob beim Anzeigen Fehler auftreten.
Um auch R/3-Systeme scannen zu können, auf die ausschliesslich über SNC (Secure Network Communication) zugegriffen werden kann, sollte das System erweitert werden. Dazu wäre es dann sinnvoll, dem MonitorAdmin die Möglichkeit zu geben den benötigten SNC-Namen sowie alle weiteren Verbindungsdaten über das Portal zu pflegen.

Desweiteren wäre es für die User des Systems nützlich zu sehen, wie aktuell die angezeigen Daten sind. Für den Fall, dass der Nutzer Daten eines Systems anzeigen will, das bisher noch nicht erfolgreich gelesen werden konnte, sollte ihm dies ersichtlich werden. Dabei ist sowohl der Zeitpunkt des letzten fehlerhaften Scans als auch die Ursache dafür interessant. Dadurch kann er dann Verbindung mit dem Monitor-Admin aufnehmen, der das Problem eventuell lösen kann.
Ein Punkt der sich diskutieren lässt, ist, dass die Systeme die gescannt werden sollen nicht aus einem R/3-System gelesen werden, sondern über die Systembeschreibungen des Portals ermittelt werden. Zusätzlich dazu könnte das Helpdesk-System, in welchem die internen Fehlermeldungen gespeichert werden, ebenso aus den Systembeschreibungen des Portals gelesen werden. Dabei stellt sich zusätzlich die Frage, ob das Ziel einer Fehlermeldung ein R/3-System ist oder z. B. ein CRM-System.
Bei den in diesem Projekt gelesenen Daten, genauer gesagt für die Loginzeiten der Benutzer, wäre es vielleicht sinnvoll, nicht nur den technischen Loginnamen anzuzeigen. Beim Scannen der Daten könnte beispielsweise ein Usermapping stattfinden, um Vor- und Nachnamen zu erhalten.

10 Resumée

Die Aufgabenstellung dieser Diplomarbeit war es, ein Tool in einer J2EE-Umgebung als Teil eines Community Portals zu entwickeln, welches es erlaubt unterschiedliche Tabellen aus mehreren SAP R/3 Systemen zu lesen. Da in erster Linie ein bereits vorhandenes Programm ersetzt werden sollte, musste dessen Funktionalität in vollem Umfang übernommen und erweitert werden. Das vorhandene Programm war dabei auf drei Tabellen spezialisiert. Bei der Entwicklung des neuen Monitors spielte darüber hinaus immer die Idee mit, ihn so unabhängig zu gestalten, dass er jedwede Tabelle aus R/3 lesen kann. Dies wurde insoweit erreicht, dass es nicht nur möglich ist, verschiedene Tabellen über Konfigurationsdateien zu spezifizieren, sondern zusätzlich noch einfache Selektionskriterien angegeben werden können.

Persönlich war die Diplomarbeit sehr lehrreich für mich. Durch das Durchlaufen eines kompletten Entwicklungszyklus', von der Anforderungsanalyse bis zum produktiven Einsatz, konnte ich alle Bereiche kennenlernen. Besonders möchte ich dabei die Spezifikationsphase hervorheben, zu der ich mir weitaus mehr Gedanken machen musste, als ich anfangs erwartet habe. Dabei spielten sogar rechtliche Themen eine Rolle. Daher ist es notwendig, alle für das zu entwickelnde System relevanten Sachverhalte zu diskutieren, auch wenn nicht alle aus den dann genannten Gründe berücksichtigt werden konnten. Zusätzlich zur Softwaretechnik habe ich die Möglichkeit bekommen, mir mit J2EE eine der neuesten Technologien anzueignen und erfolgreich anzuwenden.

Ein weiterer Punkt, der hervorzuheben ist, ist das Arbeiten in einer sehr grossen Softwarefirma kennen zu lernen, was viele Möglichkeiten und Erleichterungen, aber auch Probleme mit sich bringen kann. Zu den Erleichterungen zählt in jedem Falle, dass bereits viele Dinge entwickelt sind und genützt werden können, z.B. SAP JCo um auf R/3-Systeme zuzugreifen. Auch die Möglichkeit Personen zu finden und anzusprechen, die sich bereits in bestimmten Themengebieten auskennen und bei Problemen helfen, zählt zu den Vorteilen. Als Problem, welches bei einer grossen Firma auftreten kann, sehe ich, dass nicht mehr alles überschaubar ist, und dadurch ohne geeignete Hilfsmittel teilweise die Informationen nicht gefunden werden können.

Abschliessend kann ich sagen, dass mir das Arbeiten an diesem Projekt sehr viel Spass gemacht hat. Dazu beigetragen hat in einem grossen Maße das sehr gute Arbeitsklima in der Abteilung X Product Team ERM (ehemals DWP). Dadurch war es möglich bei Problemen jederzeit Hilfe zu bekommen.

11 Anhang A: XML-Konfigurationsdateien

Anhang A enthält sowohl die DTD, als auch die für dieses Projekt erstellte XML-Datei.

11.1 XML-DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT monitor (cssconnection, defaultvalues, systemscanranges?, login+,
table+, selection*, insert*, update*, delete*)>

<!ELEMENT cssconnection EMPTY>
<!ATTLIST cssconnection
      client CDATA #REQUIRED
      user CDATA #REQUIRED
      pwd CDATA #REQUIRED
      mshost CDATA #REQUIRED
      r3name CDATA #REQUIRED
>

<!ELEMENT defaultvalues (dbstorebatchsize, scansystemthreads)>
<!ELEMENT dbstorebatchsize (#PCDATA)>
<!ELEMENT scansystemthreads (#PCDATA)>

<!ELEMENT systemscanranges (systemscanrange+)>
<!ELEMENT systemscanrange EMPTY>
<!ATTLIST systemscanrange
      from CDATA #REQUIRED
      to CDATA #REQUIRED
>

<!ELEMENT login EMPTY>
<!ATTLIST login
      user CDATA #REQUIRED
      pwd CDATA #REQUIRED
      mandt CDATA ""
>

<!ELEMENT table (field+)>
<!ATTLIST table
      tbname CDATA #REQUIRED
      tbr3name CDATA #REQUIRED
      rowcount CDATA ""
      getallfields CDATA ""
      oncepersystem CDATA ""
      replaceall CDATA ""
>

<!ELEMENT field (replace?, fillbyfield?, fdselectoption?)>
<!ATTLIST field
      fdname CDATA #REQUIRED
      fdr3name CDATA #REQUIRED
      sqltype CDATA #REQUIRED
      sqllength CDATA ""
      key CDATA ""
      index CDATA ""
>

<!ELEMENT replace EMPTY>
<!ATTLIST replace
```

```
      repstart CDATA ""
      replength CDATA ""
      repvalue CDATA ""
      ifdatafieldnull CDATA ""
>

<!ELEMENT fillbyfield EMPTY>
<!ATTLIST fillbyfield
      ifvalue CDATA ""
      getfield CDATA ""
      getfieldstart CDATA ""
      getfieldlength CDATA ""
>

<!ELEMENT fdselectoption EMPTY>
<!ATTLIST fdselectoption
      optionoperator CDATA #REQUIRED
      value1 CDATA #REQUIRED
      value2 CDATA ""
>

<!ELEMENT selection (select-list, option-list, order-list)>
<!ATTLIST selection
      name CDATA #REQUIRED
      table CDATA #REQUIRED
      type CDATA "systable"
>
<!ELEMENT select-list (presentfield*)>
<!ELEMENT option-list (presentfield+)>
<!ELEMENT order-list (presentfield*)>

<!ELEMENT presentfield (#PCDATA)>
<!ATTLIST presentfield
      sqltype CDATA "varchar"
      foreignkey CDATA ""
>

<!ELEMENT insert (field-list)>
<!ATTLIST insert
      name CDATA #REQUIRED
      table CDATA #REQUIRED
      foreignkeytable CDATA ""
>
<!ELEMENT field-list (presentfield*)>

<!ELEMENT update (key-list, update-list)>
<!ATTLIST update
      name CDATA #REQUIRED
      table CDATA #REQUIRED
>
<!ELEMENT key-list (presentfield+)>
<!ELEMENT update-list (presentfield+)>


<!ELEMENT delete (key-list)>
<!ATTLIST delete
      name CDATA #REQUIRED
      table CDATA #REQUIRED
>
```

Prog. 11-1: monitor.dtd

11.2 XML-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE monitor SYSTEM "monitor.dtd">
<monitor>

      <cssconnection      client="001"      user="C5020590"      pwd="DWPDWP"
mshost="servprod" r3name="3929"/>

      <defaultvalues>
            <dbstorebatchsize>1500</dbstorebatchsize>
            <scansystemthreads>5</scansystemthreads>
      </defaultvalues>

      <systemscanranges>
            <systemscanrange from="000" to="LZZ"/>
            <systemscanrange from="M00" to="ZZZ"/>
      </systemscanranges>

      <login user="DEV_WP" pwd="OLIVE"/>
      <login user="ANZEIGER" pwd="DISPLAY"/>
      <login user="SUPPORT" pwd="HILFE"/>
      <login user="EARLYWATCH" pwd="SUPPORT" mandt="066"/>

      <table tbname="dwp_usr02" tbr3name="usr02">
            <field    fdname="MANDT"    fdr3name="MANDT"    sqltype="varchar"
sqllength="3" key="primaryKey"/>
            <field    fdname="BNAME"    fdr3name="BNAME"    sqltype="varchar"
sqllength="12" key="primaryKey"/>
            <field    fdname="USTYP"    fdr3name="USTYP"    sqltype="varchar"
sqllength="1"/>
            <field    fdname="TRDAT"    fdr3name="TRDAT"    sqltype="datetime"
index="true"/>
            <field    fdname="LTIME"    fdr3name="LTIME"    sqltype="datetime"
index="true"/>
      </table>
      <table    tbname="dwp_temsg"    tbr3name="temsg"    getallfields="true"
oncepersystem="true" replaceall="true">
            <field      fdname="ID"      fdr3name="ID"      sqltype="varchar"
sqllength="10" key="primaryKey"/>
            <field    fdname="EMTEXT"    fdr3name="EMTEXT"    sqltype="varchar"
sqllength="60">
                  <replace    repstart="58"    replength="2"    repvalue="     "
ifdatafieldnull="NOROW"/>
            </field>
            <field    fdname="DATCRE"    fdr3name="DATCRE"    sqltype="datetime"
index="true"/>
            <field    fdname="TIMCRE"    fdr3name="TIMCRE"    sqltype="datetime"
index="true"/>
            <field    fdname="DATDEL"    fdr3name="DATDEL"    sqltype="datetime"
index="true"/>
            <field    fdname="TIMDEL"    fdr3name="TIMDEL"    sqltype="datetime"
index="true"/>
            <field    fdname="AUTHOR"    fdr3name="AUTHOR"    sqltype="varchar"
sqllength="12" index="true"/>
            <field          fdname="APPLSERVER"          fdr3name="APPLSERVER"
sqltype="varchar" sqllength="20"/>
            <field    fdname="NOROW"    fdr3name="NOROW"    sqltype="varchar"
sqllength="3">
                  <fillbyfield          ifvalue=""          getfield="EMTEXT"
getfieldstart="59" getfieldlength="1"/>
            </field>
```

```
            <field    fdname="CUROW"    fdr3name="CUROW"    sqltype="varchar"
sqllength="3">
                  <fillbyfield          ifvalue=""          getfield="EMTEXT"
getfieldstart="58" getfieldlength="1"/>
            </field>
            <field   fdname="CLIENT"   fdr3name="CLIENT"   sqltype="varchar"
sqllength="3"/>
      </table>
      <table    tbname="dwp_e070"    tbr3name="e070v"    getallfields="true"
oncepersystem="true">
            <field   fdname="TRKORR"   fdr3name="TRKORR"   sqltype="varchar"
sqllength="20" key="primaryKey"/>
            <field         fdname="TRFUNCTION"         fdr3name="TRFUNCTION"
sqltype="varchar" sqllength="1"/>
            <field  fdname="TRSTATUS"  fdr3name="TRSTATUS"  sqltype="varchar"
sqllength="1" index="true"/>
            <field          fdname="TARSYSTEM"          fdr3name="TARSYSTEM"
sqltype="varchar" sqllength="10"/>
            <field  fdname="AS4USER"  fdr3name="AS4USER"  sqltype="varchar"
sqllength="12" index="true"/>
            <field  fdname="AS4DATE"  fdr3name="AS4DATE"  sqltype="datetime"
index="true">
                  <fdselectoption   optionoperator="BT"    value1="20020101"
value2="20020131"/>
            </field>
            <field  fdname="AS4TIME"  fdr3name="AS4TIME"  sqltype="datetime"
index="true"/>
            <field  fdname="STRKORR"  fdr3name="STRKORR"  sqltype="varchar"
sqllength="20" index="true"/>
            <field    fdname="LANGU"    fdr3name="LANGU"    sqltype="varchar"
sqllength="1" key="primaryKey"/>
            <field  fdname="AS4TEXT"  fdr3name="AS4TEXT"  sqltype="varchar"
sqllength="60"/>
            <field   fdname="CLIENT"   fdr3name="CLIENT"   sqltype="varchar"
sqllength="3"/>
      </table>

      <selection name="user" table="dwp_usr02" type="usertable">
            <select-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
                  <presentfield>BNAME</presentfield>
                  <presentfield>USTYP</presentfield>
                  <presentfield sqltype="datetime">TRDAT</presentfield>
                  <presentfield sqltype="datetime">LTIME</presentfield>
            </select-list>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
                  <presentfield>BNAME</presentfield>
                  <presentfield sqltype="datetime">TRDAT</presentfield>
                  <presentfield sqltype="datetime">LTIME</presentfield>
            </option-list>
            <order-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
```

```
                  <presentfield>BNAME</presentfield>
            </order-list>
      </selection>
      <selection name="sysmeld" table="dwp_temsg" type="usertable">
            <select-list/>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>ID</presentfield>
                  <presentfield>AUTHOR</presentfield>
                  <presentfield sqltype="datetime">DATCRE</presentfield>
                  <presentfield sqltype="datetime">TIMCRE</presentfield>
                  <presentfield sqltype="datetime">DATDEL</presentfield>
                  <presentfield sqltype="datetime">TIMDEL</presentfield>
            </option-list>
            <order-list/>
      </selection>
      <selection name="transports" table="dwp_e070" type="usertable">
            <select-list/>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>TRKORR</presentfield>
                  <presentfield>TRSTATUS</presentfield>
                  <presentfield sqltype="datetime">AS4DATE</presentfield>
                  <presentfield>STRKORR</presentfield>
                  <presentfield>AS4USER</presentfield>
                  <presentfield>ID</presentfield>
            </option-list>
            <order-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>TRKORR</presentfield>
            </order-list>
      </selection>
      <selection name="getsystems" table="dwp_allSystems">
            <select-list/>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
            </option-list>
            <order-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
            </order-list>
      </selection>
      <selection name="sysmandt" table="dwp_allSystemsMandt">
            <select-list/>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
            </option-list>
            <order-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
```

```
            </order-list>
      </selection>
      <selection name="datavolumes" table="dwp_datavolumes">
            <select-list/>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
                  <presentfield sqltype="datetime">AFDAT</presentfield>
                  <presentfield>TABLENAME</presentfield>
            </option-list>
            <order-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
                  <presentfield sqltype="datetime">AFDAT</presentfield>
                  <presentfield>TABLENAME</presentfield>
            </order-list>
      </selection>
      <selection name="tablesecondchance" table="dwp_tablesecondchance">
            <select-list/>
            <option-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>TABLENAME</presentfield>
                  <presentfield sqltype="smallint">SCFLAG</presentfield>
            </option-list>
            <order-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>TABLENAME</presentfield>
            </order-list>
      </selection>
      <insert             name="insertsysmsg"             table="dwp_temsg"
foreignkeytable="dwp_allSystems">
            <field-list>
                  <presentfield foreignkey="true">SYSID</presentfield>
                  <presentfield foreignkey="true">DBSERV</presentfield>
                  <presentfield foreignkey="true">MAINDOMAIN</presentfield>
                  <presentfield>ID</presentfield>
                  <presentfield>EMTEXT</presentfield>
                  <presentfield sqltype="datetime">DATCRE</presentfield>
                  <presentfield sqltype="datetime">TIMCRE</presentfield>
                  <presentfield sqltype="datetime">DATDEL</presentfield>
                  <presentfield sqltype="datetime">TIMDEL</presentfield>
                  <presentfield>AUTHOR</presentfield>
                  <presentfield>APPLSERV</presentfield>
                  <presentfield>NOROW</presentfield>
                  <presentfield>CUROW</presentfield>
                  <presentfield>CLIENT</presentfield>
                  <presentfield sqltype="datetime">DATREM</presentfield>
                  <presentfield sqltype="datetime">TIMREM</presentfield>
            </field-list>
      </insert>
      <update name="marksystemforscan" table="dwp_allSystems">
            <key-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
```

```
                  <presentfield>MAINDOMAIN</presentfield>
            </key-list>
            <update-list>
                  <presentfield>SCFLAG</presentfield>
            </update-list>
      </update>
      <update name="marktableforscan" table="dwp_tablesecondchance">
            <key-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>TABLENAME</presentfield>
            </key-list>
            <update-list>
                  <presentfield sqltype="integer">SCFLAG</presentfield>
            </update-list>
      </update>
      <update name="mandtforscan" table="dwp_allSystemsMandt">
            <key-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>MANDT</presentfield>
            </key-list>
            <update-list>
                  <presentfield sqltype="integer">ERRORTYPE</presentfield>
                  <presentfield>ERRORTEXT</presentfield>
                  <presentfield>R3USER</presentfield>
                  <presentfield>R3PWD</presentfield>
            </update-list>
      </update>
      <delete name="deletemsg" table="dwp_temsg">
            <key-list>
                  <presentfield>SYSID</presentfield>
                  <presentfield>DBSERV</presentfield>
                  <presentfield>MAINDOMAIN</presentfield>
                  <presentfield>ID</presentfield>
            </key-list>
      </delete>

</monitor>
```

Prog. 11-2: monitor.xml

Quellenverzeichnis

[@AANT02]	Ant Online Dokumentation, http://jakarta.apache.org/ant, 2002
[@BURG99]	Burgess, Mark: A short introduction to operating systems, http://home.lanet.lv/~sd70058/aboutos/os.html, 12.01.1999,
[@ECLI02]	Eclipse Online-Dokumentation, http://www.eclipse.org, Mai 2002
[EJBS01]	DeMichiel, Linda G.; Yalcinalp, L. Ümit; Krishnan, Sanjeev: Enterprise JavaBeans Spezification Version 2.0, 14. August 2001
[ILCH02]	Ilchev, Marco: Cross Platform Report SAP J2EE 6.20 b6, 18.03.2002
[@JGUR02]	JGuru: Enterprise JavaBeans™ Technology Fundamentals, http://developer.java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html, 03.06.2002
[@MISO02]	Microsoft: Microsoft SQL Server 2000 Driver for JDBC, http://msdn.microsoft.com/downloads/default.asp?URL=/downloads/sample.asp?url=/MSDN-FILES/027/001/779/msdncompositedoc.xml
[@OMG02]	OMG UML, http://www.uml.org
[@ORAC02]	Oracle: Oracle JDBC Frequently Asked Questions, http://otn.oracle.com/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm
[@PERF02]	Perforce Online-Dokumentation, http://www.perforce.com, Mai 2002
[@SAP02a]	SAP J2EE-Engine Dokumentation, Technical Overview, 2002
[@SAP02b]	SAP, Intranet: Java Connector https://sapneth9.wdf.sap.corp/connectors
[@SAP02c]	SAP, Intranet: WebDynpro https://sapneth10.wdf.sap.corp/webdynpro
[@SAP02d]	SAP, Intranet: Enterprise Portal, https://sapneth9.wdf.sap.corp/ep
[@SAP02e]	SAP Java Connectivity Builder Dokumentation, 19. März 2002
[@SAP02f]	SAP Perforce Dokumentation, http://p4web.wdf.sap-ag.de:1080/, Mai 2002
[@SAP02g]	SAP OpenSQL/SQLJ - SAPJ2EE-Engine Integration, http://dbi.wdf.sap-ag.de:1080/DBI/projects/java/sqlj/inqmy.html, August 2002
[@SAP02h]	SAP, HORIZON Development Processes, https://sapneth7.wdf.sap.corp/horizon, Mai 2002
[@SAP02i]	SAP R/3-Bibliothek, BC – ABAP Dictionary, Mai 2002
[SCHU02]	Schuessler, Thomas G.; www.ARAsoft.de: Developing Applications with the "SAP Java Connector" (JCo), 2002
[@SUN02a]	Sun: J2EE Tutorial, http://java.sun.com/j2ee/tutorial, Mai 2002
[@SUN02b]	Sun: Java Technology, http://java.sun.com, 2002
[@SUN02c]	Sun: Getting Started with the JDBC API, http://java.sun.com/j2se/1.4/docs/guide/jdbc, 2002
[ZENZ02]	Zenz, I.: SAP J2EE-Engine an Architectural Overview, 14.03.2002

Darstellungsverzeichnis

Programmverzeichnis

Tabellenverzeichnis

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Herbert Hartmann, geboren am 28.01.1979 in Ravensburg, ehrenwörtlich,

(1) dass ich meine Diplomarbeit mit dem Titel:
„Entwicklung eines Monitoringtool in einer J2EE Umgebung als Teil eines Community Portals“
bei der SAP AG, Walldorf unter Anleitung von Heinz Würth, SAP AG, Abteilung X Product Team ERM (ehemals DWP) und Professor Dr. Ralf Leibscher, Fachhochschule Konstanz selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Walldorf, den 25.10.2002 Herbert Hartmann