BEWERTUNGSASPEKTE DER KOMPONENTENORIENTIERTEN SOFTWAREENTWICKLUNG AM BEISPIEL VON JAVA-KOMPONENTEN

Arbeitsgruppe Softwaretechnik – Software Measurement Laboratory SMLab

Andreas Schmietendorf, Reiner Dumke, Evgeni Dimitrov, Steffen Nakonz

Otto-von-Guericke-Universität Magdeburg
Fakultät Informatik - Institut für verteilter Systeme
Postfach 4120, D-39016 Magdeburg

http://ivs.cs.uni-magdeburg.de/sw-eng

Abstrakt

Der Einsatz von Software-Komponenten gilt allgemein als Schlüssel für die Etablierung einer ingenieurmäßigen Vorgehensweise im Rahmen der Softwareentwicklung. Insbesondere neu zu implementierende Systeme machen umfangreichen Gebrauch von Software-Komponenten. Im Rahmen dieses Preprints wird auf grundlegende Aspekte der komponentenorientierten Softwareentwicklung eingegangen. Im einzelnen handelt es sich dabei um eine Abgrenzung des Komponentenbegriffs, die Prozesse einer komponentenbasierten Softwareentwicklung sowie eine Übersicht zu potentiellen Komponenten-Architekturen und domainspezifischen Komponentenansätzen. Vertiefend wird auf Komponenten-Architekturen im Umfeld der Java-Technologie eingegangen. Im weiteren wird auf die Themenstellung der Beschreibung und Bewertung von Komponenten eingegangen, wobei die Spezifikation qualitativer Eigenschaften im Vordergrund steht. Zur Verdeutlichung potentieller Bewertungsansätze werden praktisch durchgeführte Analysen von Enterprise JavaBean-Komponenten vorgestellt.

Vorwort

Die Wiederverwendung vorgefertigter Komponenten in der Softwareentwicklung wird im industriellen Umfeld als ein entscheidendes Erfolgskriterium zur Erhöhung der Produktivität und Qualität sowie Verringerung der Bereitstellungszeiten des endgültigen Produktes angesehen. Obwohl die Idee der Softwarekomponenten über dreißig Jahre alt ist, konnten erst in den letzten zehn Jahren wirklich praxistaugliche Ansätze eingeführt werden. Für Studenten der Informatik bzw. Wirtschaftinformatik ist die Kenntnis verfügbarer Komponentenmodelle von entscheidender Bedeutung für das spätere Berufsleben. Der vorliegende Preprint verfolgt gleich mehrere Zielstellungen: Zum einen sollen praktisch verwertbare Komponentenansätze vorgestellt, zum anderen potentielle Schwachstellen identifiziert und der Bedarf weiterführender Forschungsaufgaben aufgezeigt werden. Darüber hinaus erfolgt eine Einführung in die Grundbegriffe der komponentenorientierten Softwareentwicklung.

Der vorliegende Preprint verfolgt im einzelnen die folgenden Zielstellungen:

1. Vorstellung der Einflusskriterien auf die komponentenorientierte Softwareentwicklung, dabei werden produkt-, ressourcen- und prozessbezogene Themenstellungen aufgegriffen.

2. Einführung in die Technologie der Java-Komponenten, speziell der JavaBeans und Enterprise JavaBeans. Grundlegende Kenntnisse der Programmiersprache Java werden beim Studium dieses Abschnitts vorausgesetzt.

3. Identifikation zu lösender Problemstellungen im Kontext der Bewertung und Beschreibung von Softwarekomponenten.

4. Herausarbeiten von Grenzen des Komponenten-Paradigmas und Darstellung des möglichen Übergangs von Komponenten zu agentifizierten Komponenten.

Dementsprechend dient der Preprint sowohl einer Einführung in die Komponententechnologie, als auch der Darstellung potentieller Forschungsaufgaben.

Magdeburg, April 2002

Inhaltsverzeichnis

1 EINLEITUNG 9

2 KOMPONENTENORIENTIERTE SOFTWAREENTWICKLUNG 11

2.1 Aspekte der Wiederverwendung von Software 11

2.2 Eigenschaften von Komponenten 12
2.2.1 Verfügbare Komponenten-Definitionen 12
2.2.2 Anforderungen an Komponentenarchitekturen 12
2.2.3 Ansätze zur Klassifizierung des Komponentenbegriffs 14

2.3 Prozesse der komponentenorientierten Softwareentwicklung 14
2.3.1 Übersicht potentieller Prozesse 14
2.3.2 Domain-Engineering 15
2.3.3 Reuse-Management 16

2.4 Industrielle Ansätze zu Komponenten-Architekturen 17
2.4.1 Übersicht möglicher Komponenten-Modelle 17
2.4.2 Java-basierte Komponentenansätze 17
2.4.3 Microsoft ActiveX und COM+ 18
2.4.4 Standardisierung des Komponentenansatzes durch die OMG 19

2.5 Beispiele domainspezifischer Komponentenansätze 19
2.5.1 Komponentenbasierte Entwicklung von IN-Diensten 19
2.5.2 Komponentenbasierte softwaregesteuerte Messschaltungen 20
2.5.3 Nominale Bewertung der Ansätze 21

3 JAVABEAN-KOMPONENTEN 22

3.1 JavaBean Spezifikation 22
3.1.1 JavaBeans im Überblick 22
3.1.2 Grundprinzipien der JavaBean-Technologie 23
3.1.3 Funktion der BeanInfo-Klasse 24
3.1.4 Verteilung von JavaBeans 25

3.2 Beispiel der Entwicklung einer JavaBean 25
3.2.1 Spezifikation der Komponente 25
3.2.2 Implementierung funktionaler Eigenschaften 26
3.2.3 Aufbau und Funktion der BeanInfo-Klasse 28

3.3 Entwicklungen/Montage mit JavaBeans 30
3.3.1 Konzept einer bean-basierten Entwicklung 30
3.3.2 Referenzimplementierung BDK 30
3.3.3 Entwicklung unter dem Java-Studio 31

3.4 Zusammenfassende Betrachtungen 34

4 ENTERPRISE JAVABEANS 35

4.1 J2EE und EJB Spezifikation – ein Überblick 35
4.1.1 Bestandteile der J2EE-Spezifikation 35
4.1.2 Zielstellungen der EJB-Architektur 36
4.1.3 Das EJB-Komponentenmodell 37
4.1.4 J2EE-konforme Programmierschnittstellen 38
4.1.5 Verteilte Softwareentwicklung durch das EJB-Rollenmodell 39

4.2 Typen von Enterprise JavaBeans **41**
4.2.1 Entwicklung der EJB-Spezifikation 41
4.2.2 Aufgaben des Home und Remote-Interfaces 43
4.2.3 Zustandslose und zustandsbehaftete Session Beans 44
4.2.4 Persistenzbehaftete Entity Beans 44
4.2.5 Nachrichten gesteuerte Beans (Message Driven Beans) 45
4.2.6 Aufgaben des Deployment-Deskriptors 46

4.3 Beispiel der Implementierung eines EJB's **46**
4.3.1 J2EE Referenz-Umgebung 46
4.3.2 Schritte zur Implementierung und Ausführung eines EJB's 48
4.3.3 Implementierung der Server-Anwendung 49
4.3.4 Verteilung (Deployment) der Server-Anwendung 51
4.3.5 Implementierung der Client-Anwendung 60

4.4 Transaktionssicherung in EJB-Umgebungen **62**

4.5 EJB-basierte Application Server **63**

5 BESCHREIBUNG UND BEWERTUNG VON KOMPONENTEN **64**

5.1 Spezifikation von Komponenten **64**

5.2 Qualitätsbewertung von Komponenten **65**
5.2.1 Schritte zur Qualitätsspezifikation 66
5.2.2 Qualitätsklassen als Ordnungsrahmen (Schritt 1) 67
5.2.3 Komponentenspezifische Qualitätsmodelle (Schritt 2) 67
5.2.4 Determinieren der festgelegten Qualitätskriterien (Schritt 3) 68
5.2.5 Spezifikation der Qualitätseigenschaften (Schritt 4) 69

5.3 Ausgewählte Beispiele UML-basierter Spezifikationen **69**
5.3.1 Möglichkeiten einer UML-basierten Spezifikation 69
5.3.2 Beispiele UML-basierten Spezifikation 70
5.3.3 Messwertbezogene Spezifikationsansätze 71
5.3.4 Abschließende Anmerkungen 72

5.4 Empirische Analysen verfügbarer Bean-Komponenten **73**
5.4.1 Hintergründe zur Metriken-Anwendung 73
5.4.2 Berücksichtigung der Spezifikationebenen 73
5.4.3 Metriken-basierte Analyse elementarer EJB 74
5.4.4 EJB-basierte Fachkomponenten durch Design-Pattern 76
5.4.5 Potentielle Design-Empfehlungen 77

5.5 Performance Analysen im Umfeld der EJB-Technologie **78**
5.5.1 Zielstellung einer Performanceanalyse 78
5.5.2 Allgemeine Vorgehensweise zur Performanceanalyse 80
5.5.3 Verfügbare Arbeiten 81
5.5.4 EJB-Komponenten-Performance 82

6 ZUSAMMENFASSUNG **85**

7 QUELLENVERZEICHNIS **86**

8 ABKÜRZUNGEN **89**

ANLAGE: BEWERTUNGSKRITERIEN FÜR APPLICATION SERVER **90**

Abbildungsverzeichnis

Abbildung 2.1: Prozesse der komponentenorientierten Softwareentwicklung........................15

Abbildung 3.1: JavaBean Framework ([Orfali 1998] S. 659) 22

Abbildung 3.2: BDK - Bean Development Kit...30

Abbildung 3.3: Design der Anwendung unter dem Java-Studio 31

Abbildung 3.4: Customizer-Dialog zum setzen und lesen des vereinbarten Properties 32

Abbildung 3.5: Konfigurationsdialog 33

Abbildung 3.6: Java-Bean basierte Applikation unter dem Java-Studio 34

Abbildung 4.1: Möglichkeiten EJB-basierter Architekturen 35

Abbildung 4.2: Komponenten-Typen in der EJB-Spezifikation Version 2.0..........................42

Abbildung 4.3: Zugriff auf die Bean-Instanzen...43

Abbildung 4.4: Übersicht zur Vorgehensweise der Entwicklung/Verteilung 48

Abbildung 4.5: Oberfläche des Deployment-Tools 51

Abbildung 4.6: Eingangsdialog des EJB-Wizard 52

Abbildung 4.7: Erzeugen eines JAR-Files innerhalb der Applikation (example1) 53

Abbildung 4.8: Konfiguration des EJB...53

Abbildung 4.9: Festlegen der Transaktionseigenschaften 54

Abbildung 4.10: Festlegen von Umgebungsvariablen...54

Abbildung 4.11: Konfiguration durch die EJB referenzierter EJB‘s.......................................55

Abbildung 4.12: Festlegung genutzter Ressourcen wie z.B. eine Datenbank 55

Abbildung 4.13: Festlegung von JMS-Referenzen..56

Abbildung 4.14: Festlegung von Security-Einstellungen 56

Abbildung 4.15: Ausgabe des erzeugten Deployment-Descriptors..57

Abbildung 4.16: Deployment der Anwendung..58

Abbildung 4.17: Festlegung der JNDI-Namen der Applikation und benötigter Referenzen ...58

Abbildung 4.18: Bereit zur Installation der Applikation 59

Abbildung 4.19: Dialog während und nach Beendigung der EJB-Installation........................59

Abbildung 4.20: Interaktionsschritte beim Zugriff auf eine EJB-Komponente........................61
Abbildung 5.1: Schritte zur Qualitätsspezifikation [Schmietendorf/Dumke 2001]..................66
Abbildung 5.2: Spezifikation im Rahmen von Sequenzdiagrammen......................................70
Abbildung 5.3: erweiterte UML-Verteilungsdiagramme 71
Abbildung 5.4: Messansätze im Rahmen einer Komponentenarchitektur...............................72
Abbildung 5.5: Umfangsmetriken von EntityBean-Komponenten (kurz EB)..........................75
Abbildung 5.6: Zusammenhang zwischen Bean-Klasse und Home-Interface 75
Abbildung 5.7: Umfangsanalyse von Session Beans (kurz SB) 76
Abbildung 5.8: Einsatz des Session Facade Muster 76
Abbildung 5.9: Session-Facade und Summe der Entity Beans..77
Abbildung 5.10: Elemente der Performance-Beschreibung [Schmietendorf/Scholz 2000] 80
Abbildung 5.11: Oberfläche zur Konfiguration des EJB-Benchmark 82
Abbildung 5.12: EJB-Initialisierungszeiten...83
Abbildung 5.13: Schreiben über ein Entity Bean (BMP) 83
Abbildung 5.14: Schreiben über ein Entity Bean (CMP) 83
Abbildung 5.15: Schreiben über ein Session Bean (SFS)...84
Abbildung 5.16: Schreiben über ein Session Bean (SLS) 84

Tabellenverzeichnis

Tabelle 1: Verleich der EJB-Komponentenytpen 42
Tabelle 2: Übersicht der Session Facade Fachkomponenten (SB und EB) 77
Tabelle 3: Multifaktorenanalyse auf der Basis technischer Kriterien......................................90
Tabelle 4: Multifaktorenanalyse auf der Basis wirtschaftliche Kriterien 92

1 Einleitung

Die Idee, Softwaresysteme auf der Basis bereits implementierter Softwarekomponenten zu entwickeln, wurde bereits im Jahr 1969 durch [McIlroy 1968] formuliert. Als Beispiele solcher frühzeitigen Komponentenansätze seien das Funktionsprinzip der Sprache C oder auch das UNIT-Konzept der Sprache PASCAL genannt. In Anlehnung an etablierte Ingenieur-Disziplinen, wie z.B. der Nachrichtentechnik, soll auch die Softwareentwicklung durch den Einsatz von standardisierten Halbfabrikaten ein ingenieurmäßiges Vorgehens adaptieren. Grundlage eines derartigen Vorgehens ist ein arbeitsteiliger Prozess zwischen „Zulieferer" (Fertigung von Komponenten) und „Endmontage" (Zusammensetzen der Komponenten) sowie Standards hinsichtlich der funktionalen und qualitativen Eigenschaften der eingesetzten Zwischenprodukte, um die kundenseitigen Anforderungen an das endgültige Produkt erfüllen zu können. Im Umfeld der Nachrichtentechnik werden neue Systeme grundsätzlich auf der Basis am Markt verfügbarer Komponenten entwickelt, d.h. im diesem Fall der Einsatz von z.B. Prozessoren und Speicherbausteinen. Für derartige Komponenten finden sich umfangreiche Beschreibungen, die sehr exakt das Verhalten (z.B. mögliche Belegungen von Ein- und Ausgängen) unter definierten Randbedingungen (z.B. Betriebsspannung) beschreiben.

Insbesondere der objektorientierten Softwareentwicklung wurde durch ihre inhärente Fähigkeit der „Vererbung" in den frühen 90er Jahren die Fähigkeit zugeschrieben, hohe Wiederverwendungsraten erreichen zu können. Bereits 1994 stellte jedoch [Udell 1994] die These auf, dass der objektorientierte Ansatz die hohen Erwartungen an Wiederverwendbarkeit von Software nicht erfüllen könne und Softwarekomponenten dafür geeigneter wären. Insbesondere der Verwendung von Komponenten wird so das Potential zugesprochen, die Softwareentwicklung aus der heute zumeist handwerklichen Fertigung zu einer echten Ingenieur - Disziplin zu führen.

Unter [Biggerstaff&Perlis 1989] findet sich die folgende generisch gehaltene Definition für den Begriff der Wiederverwendung:

> *„Wiederverwendung von Software ist das erneute Anwenden von bei der Entwicklung eines Softwaresystems entstandenen Artefakten und angesammeltem Wissen bei der Entwicklung eines neuen Softwaresystems, um den Aufwand zur Erstellung und Pflege dieses neuen Systems zu reduzieren."*

Eine weitere Definition für die Software-Wiederverwendung insgesamt - also für den Reuse-Prozess - von [Ezran 1998] lautet beispielsweise:

> *„Software reuse is the systematic practice of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance."*

Aus beiden Definitionen können folgende Schlussfolgerungen gezogen werden:

Software-Wiederverwendung ist immer in bezug auf den Prozess der Softwareentwicklung zu sehen. Artefakte bzw. Assets schließen eines Vielzahl wiederverwertbarer Komponenten, wie z.B. Anforderungen, Entwurfsmuster oder Implementierungscode, ein. Die Charakterisierung der Wiederverwendung als systematisches Vorgehen unter Verwendung am Markt gehandelter Komponenten (‘stock of building blocks‘) weist auf die Bedeutung eines wohldefinierten Prozesses der komponentenorientierten Softwareentwicklung hin.

Mit einem komponentenbasierten Vorgehen im Rahmen der Softwareentwicklung werden vielfältige Erwartungen verknüpft, wovon einige im folgenden kurz angesprochen werden sollen. Selbstverständlich kann hier kein Anspruch auf Vollständigkeit erhoben werden:

Höhere Qualität und Verfügbarkeit: Es wird von umfangreich getesteten Komponenten ausgegangen, die weitgehend fehlerfrei arbeiten und so zu einer insgesamt höheren Qualität der gesamten Applikation führen.

Arbeitsteilige Softwareentwicklung: Durch den Einsatz wohldefinierter Komponenten besteht die Möglichkeit, Aufgabenstellungen der Softwareentwicklung sowohl innerhalb einer Phase als auch über die Phasen der Softwareentwicklung hinweg örtlich zu verteilten.

Verteilung wirtschaftlicher Risiken: Derzeitige Softwareentwicklungen implizieren hohe Risiken hinsichtlich der Realisierbarkeit fachlicher Anforderungen, da deren Auswirkungen auf die Entwicklung nur grob abgeschätzt werden können.

Spezialisierung: Über den Einsatz einer Komponente können neue Technologien zum Einsatz kommen, die durch das spezielle Wissen des Komponentenentwicklers verantwortet werden, dem Komponentenanwender aber kein entsprechendes Wissen abverlangen.

Verbesserte Erweiterbarkeit: Die Wartung, Pflege und Erweiterung vorhandener Systeme soll durch klar voneinander abgegrenzte, lose gekoppelte Komponenten vereinfacht werden. Eine fehlerhafte Komponente kann durch eine funktionsfähige andere einfach ersetzt werden.

Verkürzung der Entwicklungszeiten: Der Einsatz von Komponenten soll dazu beitragen, derzeit zumeist langwierige Softwarebereitstellungszeiten zu verkürzen und das Risiko hinsichtlich des tatsächlich ausgelieferten Funktionsumfangs der Applikation zu minimieren.

Fachbezogenenheit: Softwareentwicklungen auf der Basis von Komponenten können sich weitgehend auf den fachlichen Kontext der späteren Applikation konzentrieren. Auf diese Weise sollen komplexere fachliche Anforderungen realisiert werden können.

Inwieweit diese Erwartungen durch die derzeit am Markt verfügbaren Technologien tatsächlich abgedeckt werden können, ist derzeit nicht abschließend zu beantworten. Insbesondere die Einschränkung der potentiellen Vielfalt angebotener Funktionen einer Softwarekomponente und deren notwendige Standardisierung scheint hier eines der wesentlichen Erfolgskriterien für den Komponentenansatz zu sein. Dabei handelt es sich jedoch eher um einen menschlichen, als um einen technologischen Aspekt. Derzeit industriell verwendete Komponentenansätze konzentrieren sich insbesondere auf technologische und funktionale Aspekte. Weniger wird darauf eingegangen, wie die Identifikation und Festlegung von Komponentenkandidaten, die Spezifikation von sowohl funktionalen als auch nichtfunktionalen Eigenschaften bzw. ein die konkrete Technologie der Implementierung berücksichtigender Entwurf erfolgen soll. Insbesondere negieren diese den Aspekt der Qualitätsbeschreibung einer konkreten Komponente, weshalb aus Sicht der Autoren auf diesem Gebiet umfangreicher Forschungsbedarf besteht bzw. das Komponentenparadigma bei dieser Fragestellung möglicherweise an seine Grenzen stößt. Im Rahmen dieses Forschungsberichtes wird die Themenstellung einer granularen Qualitätsspezifikation von Softwarekomponenten aufgegriffen. Dafür wird, ausgehend vom generischen Qualitätsmodell der ISO 9126, die Ableitung eines komponentenspezifischen Qualitätsmodells unter Verwendung der GQM-Methode vorgeschlagen. Zur praktischen Umsetzung der Spezifikation wird zum einem die Verwendung einer modellbasierten Notation auf Basis der UML/OCL vorgeschlagen, zum anderen für die Operationali-

sierung konkreter Qualitätskriterien die Möglichkeiten der Verwendung von Software-Metriken aufgezeigt.

2 Komponentenorientierte Softwareentwicklung

2.1 Aspekte der Wiederverwendung von Software

Auf der „Fifth International Conference on Software Reuse 1998“ wurde die Problemstellung der Wiederverwendung nach verschiedenen Aspekten klassifiziert und schematisch in einem sogenannten „Reuse-Diamond“ dargestellt.

- *Strategie und Management* - Unter diesem Aspekt werden z.B. das Herausfinden von Reuse-Strategien und Management-Unterstützung gesehen als auch die Notwendigkeit einer Organisationsentwicklung (häufig als wiederverwendungszentrierte Organisation bezeichnet).

- *People* - In der Literatur finden sich unterschiedliche Rollenmodelle ([Jacobson 1997], [Sodhi 1998], [Coulange 1998]), die in enger Wechselwirkung zur gewählten Organisationsform stehen.

- *Assets* - Als Assets werden alle wiederverwendbaren Produkte, wie z.B. Komponenten, Objekte, Anforderungen, Analyse- und Design-Modelle, Code, Dokumentation, bezeichnet. Synonym für den Begriff der Assets werden häufig sogenannte Artefakte verwendet.

- *Technology* - Bezieht sich vor allem auf Software-Produktions-Umgebungen und verwendete Repositorys zur Speicherung notwendiger Informationen über die SW-Entwicklung als auch entsprechende Komponentendatenbanken, welche wiederverwendbare Assets enthalten.

- *Process* - Der klassisch SW-Entwicklungsprozess unterstützt nur unzureichend die Wiederverwendung. So sind z.B. notwendige Rollen bzw. die personellen Anforderungen nicht definiert.

- *Measurement* - Der Einsatz von Software-Metriken dient vor allem einer möglichen Quantifizierung im gesamten Prozess der Wiederverwendung. Auf diese Problemstellung gehen wir in einem gesonderten Abschnitt detailliert ein.

Im weiteren Verlauf des Preprints soll eine Konzentration auf Softwarekomponenten (Assets) erfolgen, wobei Aspekte des Entwicklungsprozesse (Process) und vor allem der metrikenbasierten Bewertung (Measurement) sowie die Beschreibung von Komponenteneigenschaften im Vordergrund stehen.

Wenngleich im allgemeinen Kontext der Wiederverwendung häufig die Meinung vertreten wird, dass nicht der Quellcode den eigentlich Wert darstellt, sondern eher gute Dokumentationen sowie Analyse- und Design-Modelle einen maßgeblichen Einfluss auf die Wiederverwendbarkeit besitzen, erscheint dieses im Kontext der komponentenbasierten Entwicklung nur bedingt richtig. Aus Sicht der Autoren gilt es, sowohl entsprechende Modelle und Dokumentation als auch die eigentliche quellcodebasierte Komponente verwenden zu können.

2.2 Eigenschaften von Komponenten

2.2.1 Verfügbare Komponenten-Definitionen

In diesem Abschnitt sollen ausgewählte Sichtweisen bzw. Definitionen von Komponenten und Fachkomponenten wiedergegeben werden. Darauf aufbauend soll im kommenden Abschnitt eine Zusammenfassung der Anforderungen an Komponenten erfolgen und eine im Rahmen dieser Forschungsstudie gültige Definition einer Komponente erfolgen.

Unter [Balzert 1996] und in [Orfali 1996] finden sich folgende allgemeine Definitionen für eine Komponente:

> *A component is a piece of software small enough to create and maintain, big enough to deploy and support, and with standard interfaces for interoperability. (Jed Harris in [Orfali 1996])*

> *If the components come with a bad reputation, no one will use them. Therefore, components must be of an extraordinary quality. They need to be well tested, efficient, and well documented... The component should invite reuse. (Ivar Jacobson in [Orfali 1996])*

> *„Ein Halbfabrikat bzw. eine Komponente (componentware) ist also ein abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird. Beim Entwurf des Halbfabrikats wurde auf hohe Wiederverwendbarkeit großer Wert gelegt. [Balzert 1996]*

Unter [Fachkomponenten 2002] findet sich die folgende Definitionen für eine Komponente bzw. Fachkomponente:

> *Eine Komponente besteht aus verschiedenartigen (Software-)Artefakten. Sie ist wiederverwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohldefinierte Schnittstellen zur Verfügung und ist in zur Zeit der Entwicklung in nicht unbedingt vorhersehbarer Kombination mit anderen Komponenten einsetzbar.*

> *Eine Fachkomponente ist eine Komponente, die eine bestimmte Menge von Diensten einer betrieblichen Anwendungsdomäne anbietet.*

Innerhalb dieser Forschungsstudie definieren die Autoren den Begriff der Komponenten wie folgt:

> *Eine Komponente ist eine softwarebasierte, fachbezogene Zusammenfassung von Funktionalität, die nach technologischen Aspekten, wie ihrer einfachen Integrierbarkeit und Nutzbarkeit über jeweils nur eine eindeutige Schnittstelle zu ihrer zugehörigen Anwendung, konzipiert wurde.*

2.2.2 Anforderungen an Komponentenarchitekturen

Unter [Griffel 1998, S. 78] finden sich die folgenden Anforderungen an Komponenten bzw. eine Komponentenarchitektur:

- *Unabhängigkeit von der Umgebung*: Das Zusammenspiel von Komponenten sollte möglichst übergreifend über Programmiersprachen, Betriebssysteme, Netztechnologien und Entwicklungsumgebungen funktionieren.

- *Ortstransparenz*: Eine Komponente sollte nutzbar sein, unabhängig davon, ob sie sich auf einem lokalen Rechner, in einer anderen Prozessumgebung oder im Netzwerk auf einem entfernten Rechner befindet. Die dazu notwendigen Mechanismen sollten für den Benutzer nicht sichtbar sein.

- *Trennung von Schnittstelle und Implementierung*: Die Spezifikation einer Komponente sollte vollständig unabhängig von der Implementation erfolgen können.

- *Selbstbeschreibende Schnittstellen*: Zur Laufzeitkopplung und besseren Wiederverwendbarkeit sollte eine Komponente Auskunft über ihre eigenen Möglichkeiten bzw. Zugriffspunkte geben können.

- *Problemloser sofortige Nutzbarkeit (plug & play)*: Eine Komponente sollte sofort nach Erhalt auf jeder Plattform installiert- und nutzbar sein. Dies impliziert insbesondere eine binäre Unabhängigkeit des Komponentencodes.

- *Integration und Kompositionsfähigkeit*: Eine Komponente sollte zusammen mit anderen Komponenten zu einer neuen funktionsfähigen Komponente integriert werden können.

Im Rahmen des 2. Workshops „Komponentenorientierte betriebliche Anwendungssysteme“ (WKBA 2) im Februar 2000 wurden die folgenden Anforderungen an Fachkomponenten im Rahmen eines "Brainstormings“ formuliert.

- *Richtige Größe der Komponente* im Sinne einer groben Granularität des angebotenen fachlich begründeten Funktionsumfangs der Komponente.

- *Kompositionsfreundlichkeit*, d.h. der Einsatz der Komponente sollte unter vielfältigen Bedingungen (z.B. verschiedene Betriebssysteme) erfolgen können.

- *Kompositionsrobustheit*, potentielle Fehler beim Zusammenspiel der Komponenten mit anderen Softwareteilen sollen weitgehend vermieden werden.

- *Offenheit der Funktionsbeschreibung*, die zu einer Komponente verfügbare Beschreibung sollte sowohl funktionale als auch nichtfunktionale Aspekte berücksichtigen.

- *Wirtschaftlichkeit des Einsatzes*, die Verwendung von Komponenten sollte günstiger sein als die Entwicklung eigener Software.

- *Kapselung der internen Verarbeitungslogik*, dem Anwender einer Komponenten sollte die interne Funktionalität der Komponenten weitgehend verborgen bleiben.

- *Beitrag zum Wettbewerb*, der Einsatz von Komponenten sollte die Entwicklung von fachlich begründeten Alleinstellungsmerkmalen der späteren Anwendung unterstützen.

- *Orthogonalität*, im Sinne des Angebots der richtigen fachlichen Funktionen einer Komponente ohne übermäßige Redundanz zu anderen bereits eingesetzten Komponenten.

- *Toolunterstützung*, der Einsatz der Komponenten hat unter Verwendung von Tools zu erfolgen, die wesentliche Aufgabenstellungen der Entwicklung automatisieren.

- *QoS-Eigenschaften*, die durch die Komponente angebotenen Funktionen sollten die Möglichkeit einer Qualitätsüberwachung bieten.

- *Einhaltung von Standards*, im Umfeld des komponentenorientierten Paradigmas (inkl. verwendeter Middleware) sollten verfügbare Standards eingehalten werden.

- *Domainspezifischer Wiedererkennungswert*, die durch die Komponente angebotenen Funktionen sollten sich auf die Fachebene der späteren Anwendung beziehen.

2.2.3 Ansätze zur Klassifizierung des Komponentenbegriffs

Nachdem unterschiedlichste Definitionen und Anforderungen an Komponenten zur Verdeutlichung der derzeit vorhandenen Sichten aufgezeigt wurden, sollen im folgenden Klassifizierungsansätze für Softwarekomponenten aufgezeigt werden. Eine mögliche Klassifizierung von Komponentenarten finden sich in [Brown 1998] und [Dumke 2001]:

- *Qualifizierte Komponenten (qualified components):* Hierbei sind explizite Schnittstellen definiert, die eine entsprechende Varianz für die Anpassung an vorhandene Architekturbedingungen ermöglichen.

- *Angepasste Komponenten (adapted components):* Diese Komponenten besitzen bereits Techniken, um ggf. auftretende Schnittstellenkonflikte, beispielsweise im Sinne sogenannter Wrapper-Techniken zu lösen.

- *Verbundene Komponenten (assembled components):* Diese Form ist bereits in eine Infrastruktur eingebunden, die selbst die Komposition und Integration unterstützt.

- *Aktualisierte Komponente (updated components):* Die Komponenten dieser Art sind bereits erneuert bzw. die sie nutzende Architektur gewährleistet eine problemadäquate Ersetzung derartiger Komponenten.

Neben diesen eher die Lebenszeit einer Komponente betreffenden Merkmalen können weitere Klassifizierungen gefunden werden, die z.B. fachbezogene oder technologische Merkmale berücksichtigen. Ebenfalls von Bedeutung sind COTS-Komponenten (commercial off the shelf componets). Dazu zählen kommerziell entwickelte Systeme bzw. Systemteile, die in eigene Anwendungen über wohl definierte Schnittstellen eingebunden werden können.

2.3 Prozesse der komponentenorientierten Softwareentwicklung

2.3.1 Übersicht potentieller Prozesse

Es existieren vielfältige und zum Teil recht unterschiedliche Betrachtungen der notwendigen Prozesse für die Organisation, das Management und die Durchführung der Entwicklung von Komponenten bzw. der darauf aufsetzenden komponentenorientierten Softwareentwicklung.

In Abbildung 2.1 bedeutet:

- *Application Family Engineering:* Entwicklung einer Gesamtarchitektur für mehrere verschiedene Applikationen eines Anwendungsbereiches,

- *Component System Engineering:* eigentliche Komponentenerstellung,
  - Gewährleistung von Komponenteneigenschaften vorhandener Softwareteile,

  - Entwicklung im Domain-Modell erkannter Komponenten-Kandidaten,
  - Spezifikation des Komponentenverhaltens,
- *Application Engineering:* Entwicklung/Realisierung der Applikation,
  - Identifizierung der benötigten Komponenten aus fachlicher Sicht,
  - Komposition und Konfiguration wiederverwendbarer Komponenten,
- *Domain Engineering:* schließt Architektur-Entwurf, Anforderungs-Analyse und Software-Entwicklung für eine Familie von Applikationen ein und umfasst Aktivitäten wie:
  - Charakterisierung und Abgrenzung des Problem- bzw. Diskursbereichs,
  - Herausarbeiten der Domain-Anforderungen,
  - Ableitung eines Domain-Modells durch Analyse ähnlicher Systeme,
  - Entwicklung einer Referenz-Architektur für die Domaine.

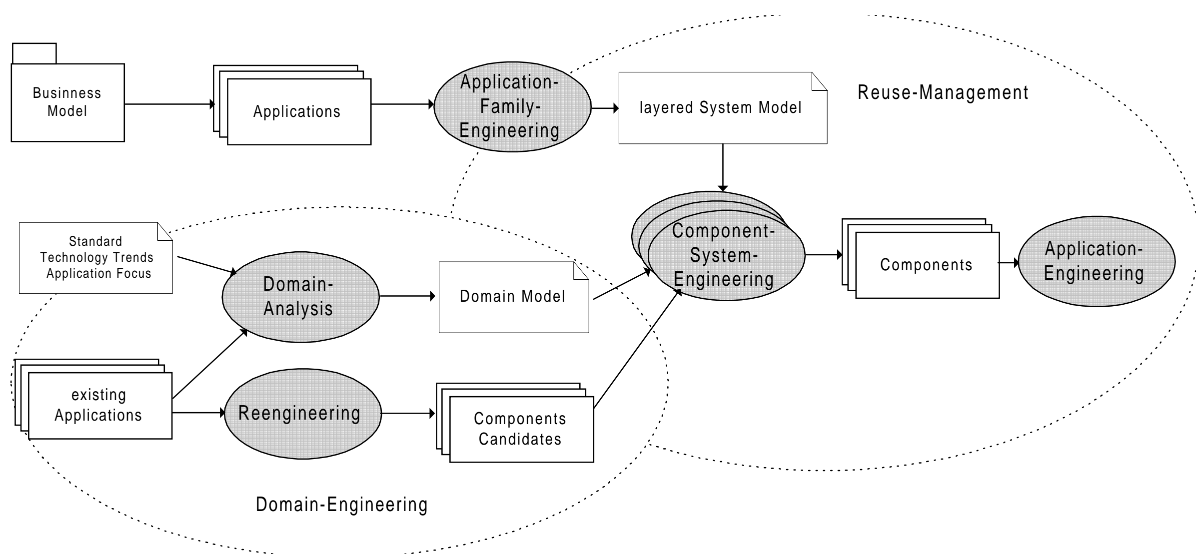

Abbildung 2.1: Prozesse der komponentenorientierten Softwareentwicklung

Sinnvoll ist es, die im Rahmen von Software-Entwicklungs-Projekten entstandenen Komponenten innerhalb einer Komponenten-Datenbank generell zu erfassen. Der Zugriff auf dieses Komponenten-Repository kann mittels einer webgestützten Applikation erfolgen. Die darin enthaltenen Komponenten werden damit allgemein zugänglich gemacht, über deren Eignung für eine generelle Wiederverwendung gibt es aber noch keine Erkenntnisse.

Wichtige Aspekte des Prozesses, wie die Identifikation, Bereitstellung und Verwaltung wiederverwendbarer Komponenten, sowie die Organisation und Einführung einer komponentenorientierten Softwareentwicklung überhaupt, lassen sich dann wie folgt auf die beschriebenen Prozesse des Domain-Engineerings bzw. Reuse-Managements abbilden.

2.3.2 Domain-Engineering

Innerhalb des Domain-Engineerings gilt es, die Identifikation potentieller Komponentenkandidaten in geeigneter Weise zu unterstützen. Dabei können zwei wesentliche Vorgehensweisen verfolgt werden.

- Existierende Anwendungen werden durch gezieltes Reengineering auf potentiell wiederverwendbare Komponenten hin untersucht.

- Im Rahmen einer die Anwendungsdomainen berücksichtigenden Analyse werden Komponentenkandidaten sowohl aus fachlicher aber auch technologischer Sicht identifiziert. Eine mögliche Vorgehensweise dazu findet sich unter [Frank/Jung 2001].

Insbesondere die letztgenannte Aufgabenstellung der Domain-Analyse bietet die Möglichkeit, sogenannte Fachkomponenten, die sich auf die fachlich begründeten Funktionen der späteren Anwendung beziehen, herauszufinden. Dieser Art von Komponenten wird derzeit das größte Potential für eine erfolgreiche Wiederverwendung zugesprochen.

2.3.3 Reuse-Management

Die Aufgaben des Reuse-Managements umfassen neben der eigentlichen Komponentenentwicklung die Verwaltung von innerhalb der Komponenten-Datenbank enthaltenen Komponenten und die Organisation der Wiederverwendung durch Bereitstellung entsprechender Hilfsmittel (z.B. Komponenten-Kataloge). Als wiederverwendbar identifizierte und als Komponenten geeignete Bausteine sollten mittels eines Software-Verteilungssystems bereitgestellt werden. Die entsprechenden Komponenten werden mit den für die Wiederverwendung notwendigen Informationen (allg. Spezifikation – siehe Kap. 5) versehen und ggf. nachbearbeitet (Kapselung, Parametrisierung), um sie für eine Wiederverwendung tauglich zu machen.

Verwaltung

Um die Wiederverwendung von Komponenten zu erleichtern, ist eine straffe und effiziente Verwaltung der Komponenten erforderlich. Es muss vermieden werden, dass die Wiederverwendungsdatenbank in einer unübersichtlichen, unstrukturierten Masse von Komponenten, die ggf. sogar fehlerhaft oder veraltet sind, erstickt. Die Verwaltung muss daher die folgenden Aspekte mit umfassen:

- Verfahren zur Aufnahme neuer Bausteine in die Wiederverwendungsdatenbank,

- Qualitätssicherung aus Sicht der Wiederverwendbarkeit (Stati zur Kennzeichnung des Reifegrades),

- Konfigurationsmanagement,

- Verfahren zur Modifikation bzw. Entfernung von Bausteinen.

Organisation

Wiederverwendung muss organisiert werden, damit sie die Chance erhält, erfolgreich zu sein. Folgende Aspekte sind beispielsweise zu berücksichtigen:

- Erarbeitung einer Klassifizierung und Typisierung von Bausteinen als Strukturierungs-Hilfsmittel,

- Erarbeitung von Kriterien zur Strukturierung des Bausteinkataloges zur Erreichung von Transparenz über die vorhandenen Bausteine,

- Erarbeitung eines typ-spezifischen Pattern für einen leicht verständlichen "Steckbrief" für jeden Baustein (typspezifische, einheitliche, möglichst formalisierte Kurz-Beschreibung der wichtigsten Eigenschaften und Schnittstellen),

- Sicherstellung der Tauglichkeit eines Bausteins für Wiederverwendung (Kriterien, Methodik, Organisation, Verantwortlichkeiten).

Inwieweit die Zuordnung des Component-System-Engineerings zum Reuse-Management sinnvoll ist, kann hier nicht abschließend geklärt werden. Diese Aufgabenstellung erfordert neben domainspezifischem „know how" insbesondere Kenntnisse im Umgang mit konkreten Technologien (z.B. den hier noch weitergehend behandelten Enterprise JavaBeans) zur Implementierung der benötigten Komponenten. In jedem Fall ergeben sich durch das Domain-Engineering und das Reuse-Management verschiedene Ansätze für potentielle Rollen (allg. Aufgabenträger) innerhalb der Identifizierung, Implementierung, Verwaltung und Anwendung von Softwarekomponenten.

2.4 Industrielle Ansätze zu Komponenten-Architekturen

2.4.1 Übersicht möglicher Komponenten-Modelle

Derzeit verfügbare Komponentenansätze konzentrieren sich entweder auf die Bereitstellung eines technologischen Frameworks zur Komponentenerstellung, wie z.B. die im weiteren noch eingehender behandelten Java-Komponenten (JavaBeans bzw. Enterprise JavaBeans) oder auch die COM+/DCOM-Komponenten von Microsoft.

Auf der anderen Seite finden sich domainspezifische Komponentenansätze, die bereits seit mehr als 10 Jahren erfolgreich eingesetzt werden. Entsprechende Beispiele dafür finden sich z.B. im Bereich von softwaregesteuerten Messschaltungen (z.B. VEE - Hewlett-Packard, LabView – National Instruments) oder auch bei der visuellen Entwicklung von IN-Diensten (Intelligent Networks, z.B. T-VoteCall) mittels SIBs (Service Independent Building Blocks) im Rahmen visueller Programmierumgebungen.

Darüber hinaus existiert von der Object Management Group (kurz OMG) mit dem CORBA-Komponentenmodell ein Ansatz zu Standardisierung von Softwarekomponenten.

2.4.2 Java-basierte Komponentenansätze

Sofern der Begriff einer Softwarekomponenten sehr weit gefasst wird, können auch die folgenden, im Rahmen von Java-Klassen-Bibliotheken angebotenen Funktionalitäten als Java-Komponenten aufgefasst werden:

- *JDBC (Java Data Base Connectivity specification):* für die Datenbankanbindung in einer selbst wählbaren Implementationsform,

- *Joe (Java Object Environment):* zum Management der Java-Objekte im Sinne einer speichereffizienten Systemlösung,

- *Java-Swing:* als Erweiterung der java.awt hinsichtlich weiterer GUI-Komponenten und eine Objekttechnologie, die das gleiche Aussehen von Java-Anwendungen auf allen unterstützten Plattformen gewährleistet,

- *JavaTel:* zur Unterstützung der Internet-basierten Telephonie,

- *JavaMedia:* für die Entwicklungs- und Implementationsunterstützung von Multimedia-systemen,

- *JavaEC:* für die Implementation von E-Commerce-Systemen,

- *KVM (KJava-based Virtual Machine):* eine hinsichtlich der Klassenanzahl reduzierte (minimale) Menge von Java-Klassen aus den Bibliotheken java.lang und java.util für die Anwendung für Geräte mit beschränkten Ressourcen,

- *Java Micro Edition:* als Bezeichnung für alle Java-Varianten, die „unterhalb" von Desktop-Anwendungen und „oberhalb" der SmartCard-Programmierung liegen, also z.B. TV-Steuerung, Web-Telefonie, Autosteuerung oder Handheld-Programmierung,

- *JINI:* zur Unterstützung der operativen Vernetzung von Geräten bzw. anderer Netze jeglicher Art zu einem Steuerungs- und Informationssystem.

Diese Beispiele verdeutlichen einerseits die hohe Dynamik auf diesem Gebiet und zeigen andererseits die Orientierung auf die Vervollkommnung der eingesetzten Java-Technologien im Zusammenhang mit der ständigen Verbreitung des Anwendungsbereiches. Im Rahmen dieses Preprints wollen wir uns jedoch auf die folgenden beiden expliziten Java-Komponentenmodelle konzentrieren:

- *JavaBeans* – als wiederverwendbare Komponenten, die innerhalb grafisch orientierter Entwicklungswerkzeuge zur Implementierung von Java-Applikationen bzw. Applets verwendet werden können.

- *Enterprise JavaBeans* - als wiederverwendbare, nicht visuelle Komponenten, die im Rahmen eines Application Servers ablaufen und dessen Dienste zur Verwaltung der Komponenten nutzen.

Obwohl sowohl die JavaBeans als auch Enterprise JavaBeans von SUN Microsystems vorgeschlagene Komponentenmodelle sind, unterscheiden sich diese doch deutlich voneinander. Häufig wird im Zusammenhang mit JavaBeans auch von clientseitigen bzw. bei Enterprise JavaBeans von serverseitigen Java-Komponenten gesprochen. Diese Unterscheidung macht aber nur bedingt Sinn, da auch mit JavaBeans eine Entwicklung für Server durchgeführt werden kann. Der signifikante Unterschied besteht darin, dass im Falle der JavaBeans der Entwickler mit vielen serverseitig notwendigen Diensten (Transaktionssicherung, Persitenz, Security,...), die er selbst implementieren muss, konfrontiert wird. Bei den Enterprise JavaBeans stellen entsprechende Application Server diese Dienste nativ zur Verfügung, d.h. der Entwickler kann sich auf die Implementierung der Fachlogik bzw. Fachdaten konzentrieren.

2.4.3 Microsoft ActiveX und COM+

Neben den hier im Vordergrund stehenden Java-Komponenten bietet Microsoft mit seinen ActiveX Controls sowie der COM+ (Component Object Model) Technologie ebenfalls Komponentenmodelle für die Implementierung grafischer Präsentationskomponenten bzw. serverseitiger Komponenteninfrastrukturen an. ActiveX Controls wurden ähnlich wie die bereits angesprochenen JavaBeans primär als Komponenten für den Bereich grafischer Oberflächen unter Visual Basic, Java oder C++ plaziert. Komponenten auf Basis der COM+ Technologie bieten, aufsetzend auf dem Microsoft Transaction Service (kurz MTS), die Möglichkeit der Implementierung mehrschichtiger verteilter Client/Server-Architekturen. Die ursprünglich aus dem Component Object Model (kurz COM – ohne plus) hervorgegangene Technologie bietet

darüber hinaus weitere Basisdienste, wie z.B. den Zugriff auf Message-orientierte Systeme mittels des Microsoft Message Queue Server (kurz MSMQ) oder aber die Verwendung eines Event Service, an.

Ein Vergleich der COM+ Technologie mit der Technik der Enterprise JavaBeans findet sich unter der folgenden Quellen im Internet: java.sun.com/products/ejb/ejbvscom.html.

2.4.4 Standardisierung des Komponentenansatzes durch die OMG

Die Object Management Group ist vordergründig durch die CORBA-Spezifikation bekannt, die in der Version 3.0 auch einen Ansatz für die Spezifikation von Komponenten (CORBA Component Model – kurz CCM) bietet. Diese programmiersprachen- und betriebssystem-unabhängigen Komponenten orientieren sich weitgehend am EJB-Komponentenansatz. Durch eine CCM-konforme Komponente können im Gegensatz zu EJB-Komponenten mehrere Schnittstellen mittels CORBA-IDL (Interface Definition Lanaguage) spezifizieren, über welche Funktionalität exportiert aber auch importiert werden kann. [Stal 2000]

2.5 Beispiele domainspezifischer Komponentenansätze

Im folgenden soll auf zwei domainspezifische Komponentenmodelle eingegangen werden, welche bereits seit einem Jahrzehnt erfolgreich eingesetzt werden. Beide Komponentenansätze wurden entwickelt, um die Entwicklungszeiten drastisch zu verkürzen bzw. dem Entwickler eine abstrakte Sicht auf die durch die Komponente zur Verfügung gestellte Funktionalität zu ermöglichen. Zusammenfassend sollen charakteristische Merkmale dieser Komponentenmodelle verdeutlicht werden.

2.5.1 Komponentenbasierte Entwicklung von IN-Diensten

Das Intelligente Netzwerk (IN) ist sowohl eine Netzwerk-Architektur als auch eine Technologie zur Entwicklung und Bereitstellung von Diensten in der Telekommunikation. Das IN existiert zusätzlich zum einfachen Telefonnetzwerk und besitzt eine gewisse Steuerfunktion. Ziel ist es, Verbindungsanforderungen (Call‘s) nach einer einfach zu konfigurierenden Logik zu verarbeiten. Die Entwicklung von IN-Diensten (z.B. Televotum „T-Vote Call“) erfolgt auf der Basis von Service Independent Building Blocks (kurz SIBs). SIBs sind wiederverwendbare Grundbausteine, die jeweils eine einzelne komplexe Aktivität beschreiben und in ihrer Komposition einen IN-Dienst repräsentieren. Durch diese Modularität ist es möglich, gleichartige Teilaufgaben verschiedener Dienste an ein und dasselbe SIB zu delegieren und damit bei der Entwicklung neuer Dienste auf bereits bestehende Komponenten zurückzugreifen.

Die Hauptmerkmale der SIBs sind zusammenfassend:

- SIBs modellieren Dienste,
- SIBs sind standardisierte, wiederverwendbare, netztransparente Einheiten,
- SIBs sind unabhängig vom spezifischen Dienst, den sie beschreiben, bzw. von irgendwelchen physikalischen Architekturüberlegungen,
- SIBs besitzen vereinheitliche, stabile Schnittstellen, mit einem oder mehreren Inputs sowie mit einem oder mehreren Outputs.

Durch Kombination dieser wiederverwendbaren SIBs innerhalb grafisch orientierter Entwicklungsumgebungen kann man relativ leicht neue Dienste zusammenstellen, ohne am IN-

System etwas zu ändern. Das derzeitige Problem besteht aber in einer vom jeweiligen Hersteller des Vermittlungssystems, auf welchem der Dienst letztendlich ausgeführt wird, abhängigen Entwicklungsumgebung. Auch die SIBs als Softwarekomponenten können jeweils nur bezogen auf den „Switch-Hersteller“ verwendet werden. Diesbezüglich versuchen Standards, wie z.B. das Komponentenframework JAIN (Java Advanced Intelligent Network), eine Alternative zu bieten. Als ein Quasi-Standard bietet JAIN einen herstellerneutralen Zugriff auf Telekommunikationssysteme, wie z.B. digitale Vermittlungssysteme, an. IN-Software kann so vergleichbar betrieblichen Anwendungsystemen mittels Standard-Technologien, wie z.B. Java-Komponenten, erstellt werden.

2.5.2 Komponentenbasierte softwaregesteuerte Messschaltungen

Ein weiterer domainspezifischer Komponentenansatz bietet sich im Rahmen des ursprünglich von Hewlett Packard entwickelten Programmiersystems VEE (Visual Engineering Enviroment) zur Steuerung von Messgeräten. Diese auf der Basis von Icons (Iconic Programming Language entsprechend [Helsel 1994]) zu verwendende Programmiersprache ermöglicht die grafische Entwicklung softwaregesteuerter Messschaltungen. Mit VEE können Messschaltungen unter Verwendung von physikalisch existierenden Messgeräten als auch die Simulationen von zu entwickelnden Messschaltungen durchgeführt werden. Entwickelte Anwendungen benötigten zur Ausführung die VEE Laufzeitumgebung.

Alle Elemente dieser Sprache werden dem Entwickler als visuelle Komponenten (repräsentiert durch Icons bzw. Sinnbilder) angeboten, an welche die nutzbaren Funktionen direkt gekoppelt sind. Diese Komponenten werden durch folgende 4 Grundeigenschaften gekennzeichnet:

- die Dateneingänge,
- die Datenausgänge,
- die Steuereingänge und
- die Ereignisausgänge.

Unter VEE können die folgende Typen von Komponenten verwendet werden:

I/O Objekte: (Ein- Ausgabeobjekte) dienen dem Zusammenarbeiten mit angeschlossenen Messgeräten, der Verwendung von Druckern, zum Benutzen von VEE fremden Programmcode (z.B. Nutzung von C-Funktionen) als auch dem Schreiben und Lesen in ein File. Interessant ist auch der Busmonitor, der zur Analyse des Datenstroms von im Rechnersystem vorhandenen Interfaces genutzt werden kann.

Computation Objekte: dienen der Datenanalyse und Datenaufbereitung. Es werden z.B. elementare und höhere mathematische Funktionen, Statistik und Wahrscheinlichkeitsrechnung, Digitalfilter, digitale Signalverarbeitung, höhere Analysis und Matrix Operationen für die Applikationsentwicklung unter VEE zur Verfügung gestellt.

Display Objekte: werden für die Datendarstellung in einer VEE-Applikation verwendet. Es ist unter anderem möglich, alphanumerische, grafische (XY-Diagramme, Polarkoordinaten, Signaldarstellung) und anwenderdefinierbare (z.B. Streifendrucker, Zeigerinstrument...) grafische Objekte in einer VEE-Anwendung zu verwenden.

Flow Control Objekte: um einen Programmablauf zu definieren, ist es möglich, Elemente der Flusssteuerung unter VEE zu nutzen, so z.B. Wiederholungs- und Schleifenkonstrukte als auch Anweisungen der bedingten Verzweigungen. Ebenso ist es möglich, Prüfschritte (Test Sequencing) im Sinne eines Debugging in die Applikation einzubinden.

Daten- und Signalgenerierung: hier handelt es sich um virtuelle Objekte, die Daten (z.B. Zufallszahlen) und Signale (z.B. Funktions- und Frequenzgeneratoren) erzeugen können.

Die Frage der Datenkonvertierung wird unter VEE weitgehend durch die Entwicklungsumgebung durchgeführt. Werden verschiedene Typen von Daten (z.B. Integer und Real) mathematisch verknüpft, so konvertiert VEE automatisch die Integerzahl in eine Realzahl und erzeugt ein Ergebnis vom Datentyp Real.

Um ein Programm zu erstellen, werden aus den unter VEE verfügbaren Objekten die für die Applikation benötigten auf der Programmieroberfläche von VEE grafisch eingebunden und die Daten- und Steuerflüsse zwischen den Objekten durch Verbindungslinien ("Verdrahtung") gekennzeichnet. Sinnvoll ist auch die Generierung einer Panel-Ansicht (Aggregation von Objekten), welche nur die Programmelemente der Applikation anzeigt, die dem User zur Verfügung gestellt werden sollen. Dadurch kann der User nicht in die fertige Anwendung eingreifen (Änderungen durchführen) und eventuell Fehlerquellen hineinbringen. In einer dann nur für den Entwickler zugänglichen Detailansicht können weiterhin Änderungen durchgeführt werden.

2.5.3 Nominale Bewertung der Ansätze

Beide vorgestellten domainspezifischen Ansätze besitzen die folgenden Merkmale:

- Komponenten besitzen einen klar abgegrenzten Funktionsumfang hinsichtlich der Anwendungsdomäne.

- Zur Ausführung wird eine entsprechenden Laufzeitumgebung benötigt, die entsprechende Basisdienste zur Verfügung stellt.

- Die dargestellten Komponenten können auf unterschiedlichen Betriebssystemen zur Ausführung gebracht werden.

- Im Rahmen eines Sinnbildes (kurz Icon) erfolgt eine Visualisierung der wesentlichen fachlich begründeten Kernfunktionen der Komponente

- In beiden Fällen konnte eine grafisch orientierte Implementierung der späteren Applikation erfolgen, so dass der Entwickler von softwaretechnische Details verschont blieb.

- Beide Ansätze haben einen Bezug auf die Domain der Nachrichtentechnik, möglicherweise ist in dieser Ingenieurdisziplin der Komponentenbegriff sehr ausgereift.

- Ebenfalls besteht die Möglichkeit, auf der Basis von Komponenten weitere Komponenten mit neuen Eigenschaften zu assemblieren.

- Die Komponentenansätze kapseln vollständig die interne Verarbeitungslogik und können nur einer begrenzten und vorher definierten Anpassung unterzogen werden.

- Beide Komponentenansätze zielen darauf ab, dass die Entwicklung der Applikation durch Spezialisten der betroffenen Anwendungsdomäne selbst erfolgen kann.

Die vorgenannten Bewertungskriterien decken sich zum Teil mit den unter Abschnitt 2.2 aufgezeigten Anforderungen an Komponentenarchitekturen. Bei den im folgenden tiefergehend untersuchten Java-Komponenten sollten diese Kriterien für einen erfolgreichen Einsatz berücksichtigt werden.

3 JavaBean-Komponenten

3.1 JavaBean Spezifikation

3.1.1 JavaBeans im Überblick

Die Java-Technologie bietet mit dem JavaBean-Modell einen möglichen Ansatz für eine komponentenorientierte Softwareentwicklung auf der Basis der Programmiersprache Java. Unter einer JavaBean kann eine Sammlung von Java-Klassen und Ressourcen-Files verstanden werden, welche die Vorteile der Java-Technologie, wie z.B. Plattformunabhängigkeit, Verteilbarkeit oder auch die mögliche Interaktion mit „Nicht-Java-Anwendungen“, berücksichtigt. Der Java-Sprachumfangs wurde dafür nicht geändert, wohl aber das JDK[1] ab der Version 1.1 um ein weiteres Package (java.beans.*) ergänzt. Dieses Package enthält spezielle Klassen und Interfaces für die Entwicklung von JavaBeans.

Die Firma SUN definiert die Eigenschaften einer JavaBean folgendermaßen:

> *„JavaBeans components, or Beans, are reusable software components that can be manipulated visually in a builder tool. Beans can be combined to create traditional applications, or their smaller web-oriented brethren, applets. In additional, applets can be designed to work as reusable Beans.“*

Entsprechend der von SUN ausgegebenen Spezifikation handelt es sich bei Java-Beans also um wiederverwendbare Softwarekomponenten, deren Instanzen im Rahmen von Builder-Tools visuell mit Hilfe von Sinnbildern (Icons) angezeigt werden. Darüber hinaus besteht die Möglichkeit der Anpassung von Beans hinsichtlich angebotener und als veränderbar deklarierter Eigenschaften im Rahmen entsprechender grafischer Dialoge. Auf der Basis der Kombination von JavaBeans können sowohl eigenständige Java-Applikationen als auch Java-Applets erstellt werden bzw. es ist wiederum die Erstellung einer neuen Bean möglich. Unterschieden werden Beans mit einer grafischen Repräsentation (typischerweise GUI-Elemente) und „unsichtbare“ Beans, welche keine Bildschirmrepräsentation haben und z.B. als „shared“ Ressourcen oder aber für den direkten Zugriff auf Services (z.B. Datenbanken) dienen. Abbildung 3.1 gibt einen Überblick über die wesentlichen Eigenschaften der Technologie.

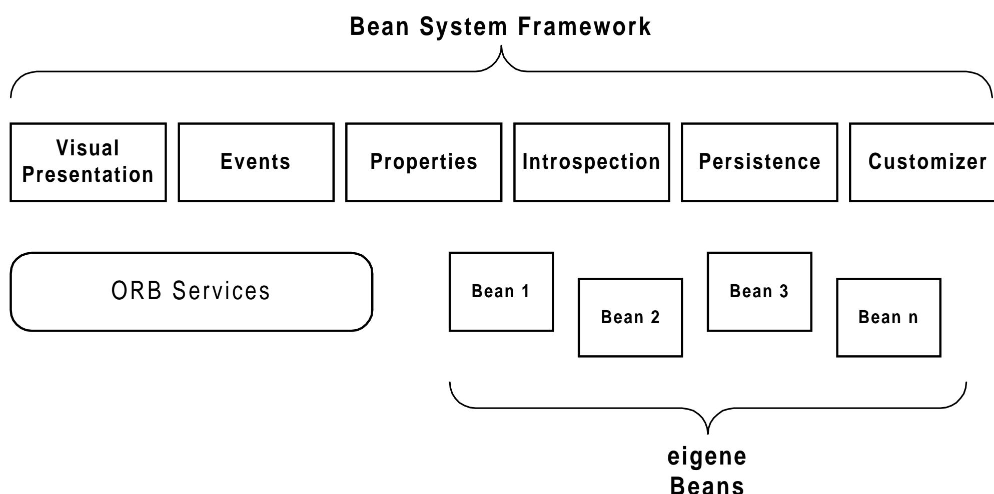

Abbildung 3.1: JavaBean Framework ([Orfali 1998] S. 659)

[1] JDK Java Development Kit

3.1.2 Grundprinzipien der JavaBean-Technologie

Im folgenden werden die grundsätzlichen Merkmale des JavaBean-Komponentenmodells erläutert:

- *Properties (Eigenschaften):* Properties folgen der aus der Objektorientierung bekannten Idee der Kapselung, d.h. Eigenschaften der Bean können nur über wohldefinierte Schnittstellen kontrolliert und verändert werden.

  Die Festlegung von Properties erfolgt über Methodenpaare, innerhalb der entsprechenden Java-Klasse, die das „Setzen„ und „Lesen“ einer Eigenschaft (Attribut) der Bean erlauben. Damit der Bean-Builder die Properties später als solche erkennt und ein Interface für sie anbieten kann, muss deren Definition einer bestimmten Syntax entsprechen, welche als *Signature Pattern* bezeichnet wird.

  Aufbau von Signature pattern für Properties: *getProperty() / setProperty()*

  Unterschieden werden Standard-, Indexed-, Bound- und Constaint-Properties. Während Standard Properties einen einzelnen Wert repräsentieren, sind die Indexed-Properties ein Array/Feld gleicher Datentypen. Bound-Properties kombinieren das Property mit einem Event (Property Change Event), welches z.B. bei einer Propertyänderung ausgelöst wird. Die Contraint-Properties bieten die Möglichkeit, anderen Beans die Änderung eines Property mitzuteilen und eine Zustimmung abzuwarten. Erfolgt diese nicht, geht das Property in seinen vorhergehenden Zustand zurück.

- *Introspection (Selbstbeobachtung):* Diese sowohl implizit, als auch explizit angebotene Eigenschaft bietet die Möglichkeit, dass Java-Beans zur Laufzeit dem Builder-Tool Auskunft hinsichtlich ihrer möglichen Funktionen und Eigenschaften geben können. Speziell können die als public definierten Methoden, Properties und ausführbare Ereignisse abgefragt werden. Auf dieser Grundlage kann innerhalb entsprechender Bean-Builder-Tools auf einen direkten Quellcode-Zugriff verzichtet werden.

  Die implizite Introspection-Funktion basiert auf einem einfachen Reflection-Mechanismus, der alle in der JavaBean angebotene Methoden liefert. Mit Hilfe von Design Patterns wird dann auf Eigenschaften, Ereignisse etc. geschlossen.

  Der explizite Ansatz funktioniert über das BeanInfo Interface, das wie ein Filter arbeitet. Auf Anfrage werden Deskriptor-Objekte geliefert, welche die gewünschten Informationen enthalten.

- *Customisation (Anpassung):* Im Rahmen von Bean-Builder-Tools können existierende JavaBeans entsprechend den aktuellen Erfordernissen durch Verändern der Properties visuell angepasst werden. Hinsichtlich der Erweiterbarkeit folgen diese der objektorientierten Philosophie mit Polymorphie und Kapselung, wofür das Interface *Customizer* im Rahmen der JavaBean zu spezifizieren ist.

- *Event-Handling (Ereignisse):* Beans können 'Ereignisse' auf der Basis des ab der JDK 1.1 Version angebotenen Delegation-Event-Handling-Modell (mittels Event-Sources und E-vent-Listenern) untereinander austauschen. Innerhalb der Bean-Builder-Tools können E-vent-Sources mit –Listenern grafisch verbunden werden.

  Zum Beispiel könnte die Änderung eines Property anderen interessierten Beans über das Auslösen eines „PropertyChangeEvent“ mitgeteilt werden. Dafür müssen innerhalb des

Event-auslösenden Beans die interessierten Listener registriert werden. Die das Event verarbeitenden Beans müssen ihrerseits das Interface *java.beans.PropertyChangeListener* implementieren.

- *Persistenz (Speicherung):* Die Eigenschaften (Properties und Daten der Komponente) der angepassten und verbundenen Bean-Instanzen müssen für die spätere Ausführung der Applikation dauerhaft gespeichert werden können.

  Wird z.B. eine Komponente visuell innerhalb eines Bean-Builders hinsichtlich der angebotenen Properties geändert, muss die Möglichkeit der persistenten (dauerhaften) Abspeicherung der alten und neu konfigurierten Werte gegeben sein. Diese Eigenschaft ist insbesondere für die Wahrung von Konsistenzbedingungen notwendig. Stellt die geänderte Eigenschaft keine sinnvolle Konfiguration dar, sollte der Ausgangszustand wiederhergestellt werden können.

3.1.3 Funktion der BeanInfo-Klasse

Bean-Builder analysieren alle angebotenen public-Methoden und Events einer Bean und bieten diese innerhalb des Builder-Tools an. Bedingt durch die typischerweise genutzte Vererbung für die Entwicklung einer neuen JavaBean können so eine Vielzahl „ererbter Methoden" angeboten werden, die beim Komponentenbenutzer eher Verwirrung auslösen, als zum einfachen Umgang mit der Komponente beizutragen. Um hier eine Vereinfachung herbeizuführen, wurde das Konzept der BeanInfo-Klassen eingeführt, die vergleichbar einem Filter nur solche Properties, Methoden und Ereignisse einem Bean-Builder-Tool bekannt macht, die tatsächlich zur Nutzung der Komponente notwendig und sinnvoll sind.

Wird durch den Komponentenentwickler keine eigene „Bean-Info" Klasse angelegt, wird grundsätzlich „SimpleBeanInfo" verwendet und alle Methoden, Properties und Events werden für die Verarbeitung „durchgelassen". Neben dieser Filterfunktion kann in der BeanInfo-Klasse auch ein mit der Bean korrespondierendes „Icon" (Sinnbild) festgelegt werden, das in einem Builder-Tool die grafische Repräsentation der Bean übernimmt.

- *Bean Descriptor* – macht Aussagen über die JavaBeans selbst und verweist auf eine zur Bean-Klasse zugeordnete Customizer-Klasse.

  Durch den expliziten Einsatz einer Customizer-Klasse lassen sich komplexere Customizer-Dialoge (Eigenschafts-Editoren) erstellen als diese standardmäßig verwendet werden. Die Verbindung zur JavaBean erfolgt über die Bean-Info-Klasse.

- *Property Descriptor* – Festlegung der Properties, die innerhalb der Bean-Builder zur Verfügung stehen sollen.

- *Method Descriptor* - Festlegung der Methoden, die innerhalb der Bean-Builder zur Verfügung stehen sollen.

- *Event Set Descriptor* - Festlegung der Events (Ereignisse), die innerhalb der Bean-Builder zur Verfügung stehen sollen.

Der Dateiname der BeanInfo-Klasse muss sich immer aus dem Dateinamen des entsprechenden JavaBeans und „BeanInfo" zusammensetzen. Nur so kann gewährleistet werden, dass die verwendeten Builder-Tools, wie z.B. das BDK, Informationen der BeanInfo-Klasse auswerten können.

3.1.4 Verteilung von JavaBeans

Die breite Verbreitung von Java begründete sich in der Vergangenheit insbesondere auf die Verwendung innerhalb HTML-Files eingebetteter Applets. Vor dem JDK 1.1 waren für die Übertragung eines Applets und der dazu notwendigen Dateien vom Webserver zum Browser vielfache HTTP-Transaktionen erforderlich. Mit der Einführung von JAR-Dateien, die über den gleichnamigen jar-Befehl des JDK erstellt werden, besteht jetzt die Möglichkeit, alle Dateien eines Applets zusammenzufassen und mit einer HTTP-Transaktion zu übertragen, was zu erheblichen Performancesteigerungen führen kann.

Die Verteilung von Java-Beans erfolgt ebenfalls innerhalb von JAR-Files (Java Archive), welche technisch dem ZIP-Format entsprechen und so auch durch Programme, wie z.B. WinZip, verarbeitet werden können. Ein solches JAR-File kann eine oder mehrere Beans und die dafür notwendigen Files (Klassen-, Ressourcen-, HTML-Files) inklusive Verzeichnisstruktur enthalten. Neben der Möglichkeit, Dateien zusammenzufassen, besteht die Möglichkeit, den enthaltenen Dateien digitale Signaturen mitzugeben.

Ebenfalls im JAR-File steht eine sogenannte Manifest-Datei, die einen Index über die Klassen der enthaltenen Java-Beans und den eigentlichen Einsprung (Start-Klasse) enthält. Im folgenden ist ein Auszug einer Manifest-Datei wiedergegeben:

```
Manifest-Version: 1.0

Name: sunw/demo/juggler/Juggler.class
Java-Bean: True
Digest-Algorithms: SHA MD5
SHA-Digest: Y3coWI0goXPl7ug2/bl9MGTKP1Q=
MD5-Digest: ajTYJMBtosaSpBNOjJ+FXQ==
```

3.2 Beispiel der Entwicklung einer JavaBean

3.2.1 Spezifikation der Komponente

Wie bei jeder Softwareentwicklung sollte am Beginn eine Analyse der innerhalb der Komponenten abzubildenden fachlichen Funktionalität stehen, gefolgt von einem Design, das auch programmtechnische Aspekte, wie z.B. die Verwendung von Wrapper-Klassen, berücksichtigt. Für diese Aufgabenstellung kann die Notation der UML mit ihren verschiedenen Diagrammtypen (z.B. UseCase-, Sequenz-, Klassen-, State-Diagramm) sinnvoll verwendet werden. Da das Ziel dieses Skriptes nicht in der Verdeutlichung der objektorientierten Modellierung besteht, soll eine einfache verbale Beschreibung der im folgenden dargestellten Java-Bean an dieser Stelle ausreichen.

Als Beispiel soll eine Stoppuhr implementiert werden, die später als Komponente zur Verfügung steht und über folgende Eigenschaften verfügt.

- Start der Zeitmessung,
- Stop der Zeitmessung,
- Rückstellen des Zählers auf Null,
- Belegen des Zählers mit einem Initialwert,
- Ausgabe der aktuellen Zeit im Format MM:SS.

3.2.2 Implementierung funktionaler Eigenschaften

Der Quelltext der Stoppuhr-Komponente enthält sowohl ein entsprechendes Property zum setzen des Anfangszustandes des Zeitzählers auf Sekunden-Basis und Events, die jede Sekunde an entsprechend registrierte Listener (Klassen die Ereignisse verarbeiten) gesendet werden können.

```
// Info: Beispiel-Bean einer Stopuhr
// by © 1999 Jens Lezius and Andreas Schmietendorf
// Otto-von-Guericke-Universität Magdbeurg

package Beans;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.Serializable;
```

Die folgende Bean3 wird von Panel abgeleitet und implementiert die Interfaces Serializable für die Realisierung der Persistenzeigenschaften einer Bean und Runnable für die Verwendung innerhalb des Bean-Threads.

```
public class Bean3 extends Panel implements Serializable,
Runnable{
     BorderLayout borderLayout1 = new BorderLayout();
     TextField textField1 = new TextField();
     Thread t = new Thread(this);
     private String text = "leer";
     public String i ;
     public int a =0;
     private boolean ind=false;
     private transient PropertyChangeSupport proper-
tyChangeListeners = new PropertyChangeSupport(this);
```

Über den parameterlosen Konstruktor wird die Bean bzw. die Objektinstanz initialisiert. Die eigentlichen Eigenschaften werden in der Methode jbInit() festgelegt, die innerhalb des Konstruktur aufgerufen wird.

```
//parameterloser Konstruktor
  public Bean3() {
     try  {
                jbInit();
     }
     catch (Exception ex) {
               ex.printStackTrace();
     }
  }

  //Initialisierung der Bean-Komponente
  private void jbInit() throws Exception {
     this.setSize(new Dimension(186, 24));
     textField1.setText("Bean3");
     this.setLayout(borderLayout1);
     this.add(textField1, BorderLayout.NORTH);
  }
```

Der folgende Thread realisiert die eigentliche Zeitzählung, wobei das Propertie „Text“ durch die Methode „setText“ entsprechend dem Ergebnis der Hochzählung gesetzt wird.

```
//Thread für die Ausführung starten
public void run(){
  //Endlosschleife
   for (;;){
             ind=true;
             a=a + 1;
             i=""+ a;
             setText(i);
             textField1.setText(text);
        try{
             Thread.sleep(1000);
        }catch
             (InterruptedException ex){}
   }
}
```

Diese Methode realisiert den eigentlichen Start der Stoppuhr und soll in einem Bean-Builder-Tool zur Verfügung stehen.

```
// Start der Uhr
public void start(){
   if (ind) {
             t.resume();
    }
   else{
             t.start();
             setText(i);
   }
}
```

Diese Methode realisiert den eigentlichen Stopp der Stoppuhr und soll ebenfalls im Bean-Builder-Tool zur Verfügung stehen.

```
// Stopp der Uhr
public void stop(){
   t.suspend(); // Thread beenden
}
```

Diese Methode realisiert das Rücksetzen der Stoppuhr auf 0 soll ebenfalls im Bean-Builder-Tool verwendet werden können.

```
// Rücksetzen auf 0
  public void reset(){
     a= 0;
      i="0";
     setText(i);
     textField1.setText(text);
  }
```

Die folgenden beiden Methoden entsprechen dem Entwurfsmuster (Methoden-Signatur) für Java-Beans und realisieren so das Property dieser Bean. Ebenso werden entsprechende Listener (z.B. eine andere Bean) über Änderungen des Property durch ein entsprechedes Event „propertyChangeListener“ informiert.

```
  //Propertie für Text setzen und lesen
  public void setText(String newText) {
     String  oldText = text;
     text = newText;
    //Auslösung eines entsprechenden Event
     propertyChangeListeners.firePropertyChange("text",
oldText, newText);
  }

  public String getText() {
     return text;
  }
```

Die folgenden beiden Methoden realisieren in der späteren Bean-Applikation die Registrierung von auf Events wartenden Listenern.

```
  //entfernen eines registrierten Listener
  public  synchronized  void  removePropertyChangeListe-
ner(PropertyChangeListener l) {
     propertyChangeListe-
ners.removePropertyChangeListener(l);
  }

  //registrieren eines registrierten Listener
  public   synchronized   void   addPropertyChangeListe-
ner(PropertyChangeListener l) {
     propertyChangeListeners.addPropertyChangeListener(l);
  }
}
```

3.2.3 Aufbau und Funktion der BeanInfo-Klasse

Zur implementierten Bean3 wurde die Klasse „Bean3BeanInfo.java“ definiert werden, überschrieben wurden nur die Methoden für den Method- und Property-Descriptor, der Event - Deskriptor blieb unverändert:

```
package Beans;

import java.beans.*;
import java.lang.reflect.*;
public class Bean3BeanInfo extends SimpleBeanInfo {
  Class beanClass = Bean3.class;
  //Icon-Name zur Bean in String-Variable übergeben
  String iconColor16x16Filename = "bspicon.gif";
  public Bean3BeanInfo() {
  }
```

Der folgende Property-Deskriptor legt die angebotenen Properties - hier nur das Property „Text“ - fest.

```
  public PropertyDescriptor[] getPropertyDescriptors() {
    try
    {
      PropertyDescriptor pds = new PropertyDescriptor
```

```
    ("Text",Bean3.class);
    pds.setDisplayName("Text");
    pds.setShortDescription("Texteingabe möglich");
    PropertyDescriptor[] pda = {pds};
    return pda;
  }
  catch (IntrospectionException e)
  {
    return null;
  }
}
```

Der folgende Method-Deskriptor legt die angebotenen Methoden der Bean, hier die Methoden „start“, „stop“ und „reset“, fest.

```
public MethodDescriptor[] getMethodDescriptors() {
  //First find the "method" object.
  Class args[] = {};
  Method m, m1, m2;
  Try
    {
      m = Bean3.class.getMethod("start", args);
      m1 = Bean3.class.getMethod("stop", args);
      m2= Bean3.class.getMethod("reset", args);
    }
  catch (Exception ex)
    {
    // "should never happen"
    throw new Error("Missing method: " + ex);
    }

  // Now create the MethodDescriptor array:
  MethodDescriptor result[] = new MethodDescriptor[3];
  result[0] = new MethodDescriptor(m);
  result[1] = new MethodDescriptor(m1);
  result[2] = new MethodDescriptor(m2);
  return result;
  }
```

In der folgenden Methode getIcon wird das konkret zu verwendende Icon festgelegt, dabei können verschiedene Formate genutzt werden (hier nur 16 x 16).

```
public java.awt.Image getIcon(int iconKind) {
    switch (iconKind) {
    case BeanInfo.ICON_COLOR_16x16:
      return iconColor16x16Filename
      != null ? loadImage(iconColor16x16Filename) : null;
    case BeanInfo.ICON_COLOR_32x32:
     ...
    }
    return null;
  }
}
```

3.3 Entwicklungen/Montage mit JavaBeans

3.3.1 Konzept einer bean-basierten Entwicklung

Die Verwendung von JavaBeans erfolgt innerhalb grafikorientierter Werkzeuge, wie dem BDK von SUN, dem Java-Studio von SUN oder Visual-Age von IBM. Zu den beiden erstgenannten Tools sollen einige signifikante Eigenschaften aufgezeigt werden. Wenngleich das explizite Interesse an den JavaBeans zum Zeitpunkt der Erstellung dieses Preprints deutlich nachgelassen hat, soll die komponentenorientierte und vor allem grafisch orientierte Entwicklung mittels dieser Komponenten dennoch hier skizziert werden. Auch wenn die im folgenden dargestellten Werkzeuge nur noch eine geringe praktische Relevanz haben, wird diese Form der Entwicklung aus Sicht der Autoren weiter an Bedeutung gewinnen. Der Begründung für diese Hypothese liefern insbesondere die in Abschnitt 2.5 skizzierten domainspezifischen Komponentenansätze, die seit vielen Jahren erfolgreich angewendet werden.

3.3.2 Referenzimplementierung BDK

Das von SUN für die Endmontage von JavaBeans vorgeschlagene BDK (Bean Developer Kit) kann als Referenzimplementierung eines Bean-Builders betrachtet werden. Es versteht sich dabei nicht als Entwicklungsumgebung für die Implementierung von Java-Source-Code. Vielmehr vereinigt es Hilfsprogramme und Dokumentationen sowie eine ausschließlich visuell arbeitende Entwicklungsumgebung für die Montage (Composition) von JavaBeans zu kompletten Applikationen.

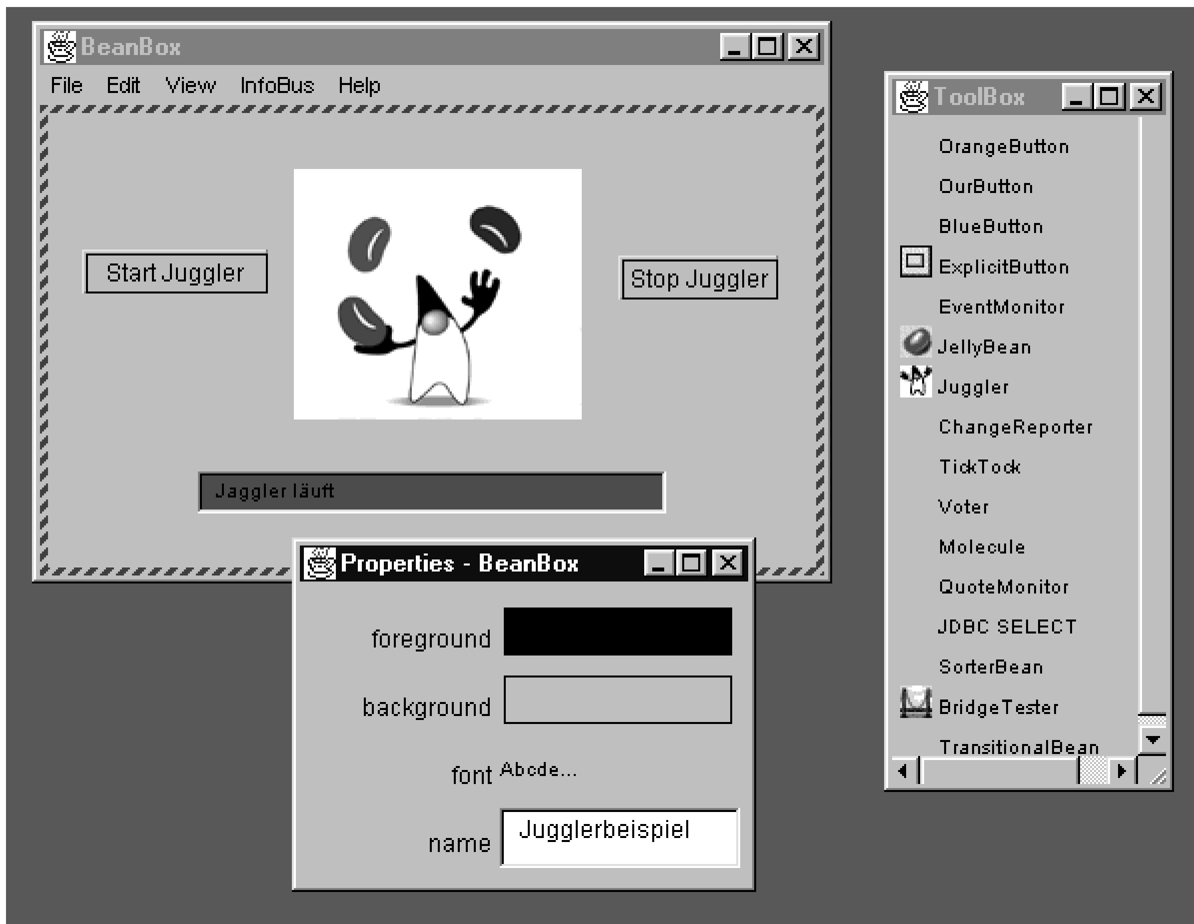

Abbildung 3.2: BDK - Bean Development Kit

Innerhalb dieser Entwicklungsumgebung stehen nach dem Start des BDK[2] die folgenden drei grafischen Dialoge zur Verfügung:

- *BeanBox:* entspricht der grafischen Präsentation der späteren Anwendung. Für die Anwendung benötigte Beans werden mittels der Maus per „Drag & Drop“ auf diesem Window plaziert.

- *ToolBox:* bietet alle für die Montage verfügbaren JavaBeans auf der Basis des korrespondierenden Sinnbildes (Icon) an. Dabei stehen sowohl die „native“ mit dem BDK ausgelieferten Beans als auch nachträglich explizit durch den Nutzer importierte JavaBeans zur Verfügung.

- *Propertie:* nachdem eine Bean innerhalb der Bean-Box plaziert wurde, erfolgt, sofern dieses „markiert“ ist, eine Anzeige der gebotenen Properties über einen entsprechenden Dialog. Nach dem Start der BDK wird der Propertie-Dialog der BeanBox angezeigt, der ebenfalls auf der Basis einer JavaBean implementiert wurde.

Über die Menüleiste der BeanBox können Funktionen zur Ereignisverknüpfung zwischen JavaBeans aufgerufen werden. Es besteht die Möglichkeit, neue JavaBeans in das BDK zu importieren, ebenso kann die montierte Applikation gespeichert werden oder auch die fertige Java-Applikation (z.B. als Applet) herausgeneriert werden.

3.3.3 Entwicklung unter dem Java-Studio

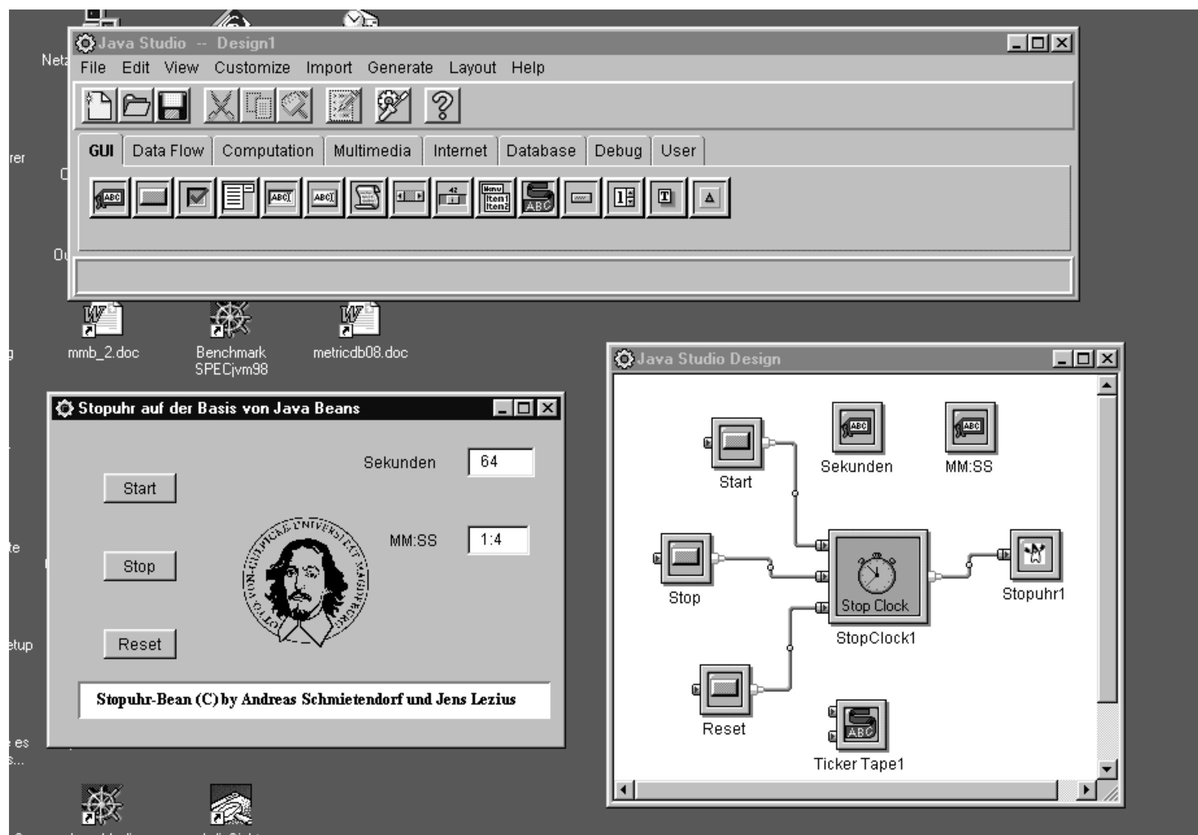

Abbildung 3.3: Design der Anwendung unter dem Java-Studio

[2] Start des BDK erfolgt durch Ausführung von: LW:\BDK\beanbox\run.bat

Das Java-Studio (siehe Abbildung 3.3) ist eine rein visuelle Entwicklungsumgebung, die dem Entwickler bereits in der Grundversion einen vorgefertigten Satz von JavaBeans (GUI-Elemente, ODBC-Datenbankanbindung, Berechnungen,...) zur Verfügung stellt. Bevor auch eine selbstentwickelte JavaBean verwendet werden kann, muss dieses über die Import-Schnittstelle in die Entwicklungsumgebung eingebunden werden. Bei diesem Vorgang können die angebotenen Properties und public definierten Methoden der JavaBean ausgewählt (d.h. eine Einschränkung der verfügbaren Methoden der Komponente ist auch auf diese Weise möglich) sowie die Anordnung der Ein- (links) und Ausgänge (rechts) sowie Trigger (oben) am korrespondierenden Icon festgelegt werden. Zur Entwicklung einer JavaBean-basierenden Anwendung sind die folgenden Schritte notwendig:

1. Analyse der fachlichen Anforderungen an die spätere Applikation. Auf der Basis von Use Cases (siehe entsprechendes UML-Diagramm) sind fachlich begründete Anwendungsszenarien der späteren Nutzer zu entwerfen.

2. Identifikation der aus fachlicher Sicht benötigten Komponenten und Modellierung des komponentenbasierten Anwendungsdesigns mittels einer grafischen Notation, wie z.B. den UML-Packetdiagrammen.

3. Auswahl von JavaBeans, welche die fachlich begründeten Funktionen der Anwendung realisieren können. Ggf. sind Funktionen, die nur durch JavaBeans abgedeckt werden können, durch eine individuelle Implementierung zu realisieren.

4. Implementierung der Applikation:

   - Innerhalb eines Design-Windows erfolgt die grafische Anordnung der JavaBeans, d.h. die korrespondierenden Icons werden per „Drag and Drop“ plaziert.

   - Über die rechte Maustaste bzw. per Menüauswahl können Dialoge der zur JavaBean verfügbaren Properties, wie in Abbildung 3.4 und 3.5 dargestellt, aufgerufen werden. Innerhalb dieser Dialoge lassen sich Properties konfigurieren, die z.B. die Größe, Farbe oder aber vordefinierte Ausgabetexte betreffen.

   - Der nächste Schritt für das Anwendungsdesign betrifft die „Verdrahtung“ der angebotenen Connects, wobei die „Bubble“-Hilfe mit der Angabe des Methodennamens am jeweils durch die Maus gekennzeichneten Connect sehr hilfreich ist.

   - Über einen Menüeintrag „generate“ bietet sich im Java-Studio die Möglichkeit der Generierung eines Applets, einer Applikation oder einer neuen Bean, vergleichbare Funktionen werden auch in anderen Bean-Builder-Tools angeboten.

5. Test und Einführung der Applikation (wird hier nicht weiter betrachtet)

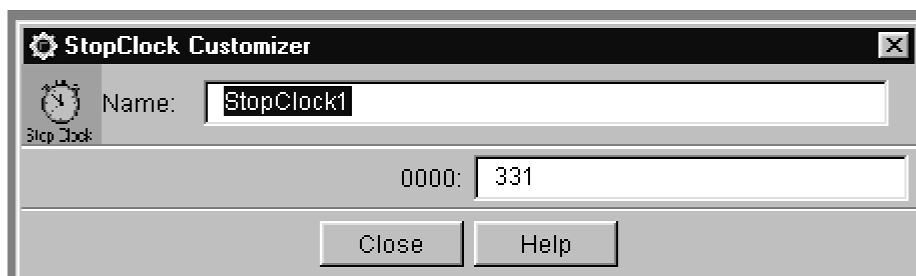

Abbildung 3.4: Customizer-Dialog zum setzen und lesen des vereinbarten Properties

Abbildung 3.4 zeigt den Customizer-Dialog der StopClock-Bean zum Setzen bzw. Lesen des vereinbarten Properties „0000" (entspricht fachlich dem Setzen der Stopuhr auf einen definierten Anfangszustand). Grundlage dafür sind die verwendete Methoden-Signatur des Java-Bean-Modells „PUBLIC VOID SETTEXT(STRING NEWTEXT){...}" bzw. „PUBLIC VOID GETTEXT(STRING NEWTEXT){...}". (siehe auch Quelltext zur JavaBean im Abschnitt 3.2)

Abbildung 3.5 zeigt einen komplexen Customizer-Dialog der ebenfalls in der Anwendung verwendeten Laufschrift-Bean. Ein derartiger Dialog muss explizit entwickelt werden, hier reichen die standardmäßig verwendeten Methoden-Signaturen nicht aus.

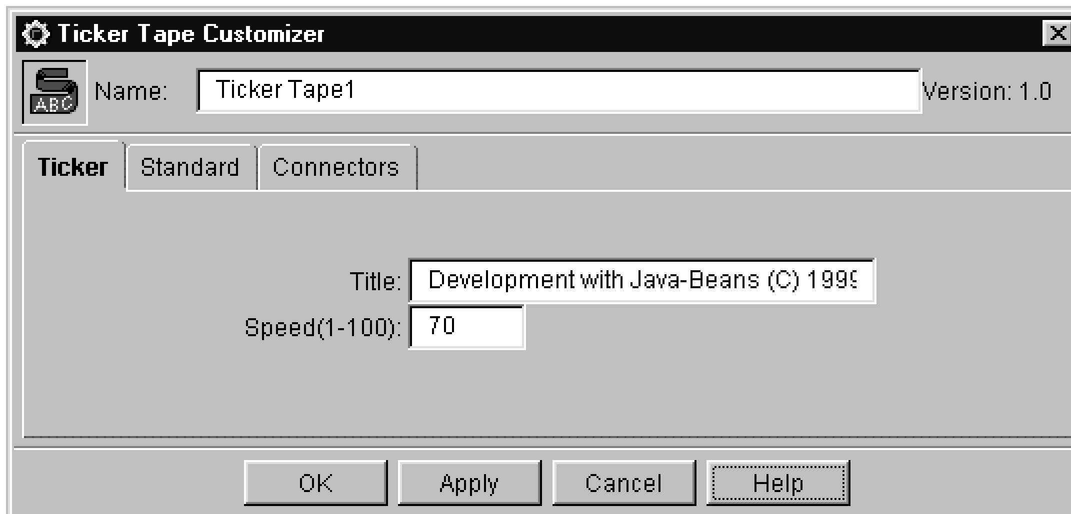

Abbildung 3.5: Konfigurationsdialog

Wird in einem Bean-Builder z.B. das Event „ActionPerformed" einer Buttons-Bean (z.B. in Abbildung 3.3 der Button „Reset") ausgewählt und visuell mit einer Methode der StopClock-Bean, wie z.B. „reset( )", verbunden, generiert der Bean-Builder eine sogenannte Adapterklasse, die das Event der Button-Bean an die verarbeitende Methode der Bean3 weiterleitet. Das folgenden Listing zeigt den Quellcode dieser generierten Adapterklassen.

```
// Automatically generated event hookup file.

package tmp.sunw.beanbox;
import Beans.Bean3;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ___Hookup_15a98e19ab implements java.awt.event.ActionListener,
java.io.Serializable {

    // Ziel an welches das Event weitergeleitet werden kann
    public void setTarget(Beans.Bean3 t) {
        target = t;
    }

    // Event-Verarbeitung welche die Methode der Ziel-Bean aufruft
    public void actionPerformed(java.awt.event.ActionEvent arg0) {
        target.reset();
    }

    private Beans.Bean3 target;
}
```

Bemerkung: Die Quellen wurden dem BDK entnommen, da das Java-Studio keinen Zugriff auf die Java-Quellen ermöglicht.

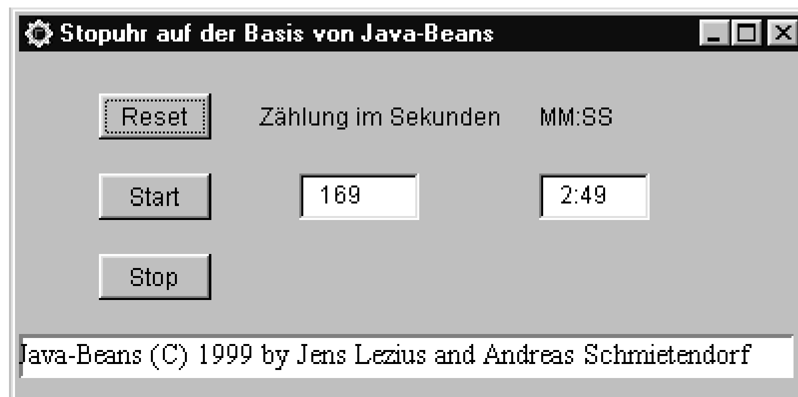

Abbildung 3.6: Java-Bean basierte Applikation unter dem Java-Studio

Abbildung 3.6 zeigt die fertige Java-Applikation inkl. grafischer Repräsentation (Bean 3 als Textfeld mit aktuellen Wert „169“) der verwendeten Beans.

3.4 Zusammenfassende Betrachtungen

Wenngleich die JavaBeans eine interessante Vorgehensweise zur visuellen Entwicklung von Anwendungen darstellen, reichen die derzeitigen Möglichkeiten dieser Technologie für die Entwicklung komplexer Client/Server Anwendungen bei weitem nicht aus. Vordergründig wurde diese Technologie für die Implementierung von Anwendungsteilen mit einer grafischer Präsentation entwickelt. Dennoch bestünde theoretisch auch die Möglichkeit, serverseitig ablaufende Anwendungen mittels Java-Bean-Komponenten zu realisieren. Diesbezüglich bestehen jedoch vielfältige Problemstellungen, welche durch die Technologie der JavaBeans derzeit nicht abgedeckt werden und durch den Entwickler selbst gelöst werden müssten, wie z.B.:

- Fehlende explizite Unterstützung einer mehrstufigen Client/Server-Architektur, hier werden dem Entwickler alle Aufgabenstellungen der Verteilung überlassen.
- Eine Transaktionssicherung der durch die Komponente angebotenen Funktionen muss vom Entwickler selbst vorgenommen werden.
- Die persistente Abspeicherung von Daten im Rahmen eines externen Datenbankmanagementsystems ist durch den Entwickler unter Verwendung von Low-Level-Interfaces (z.B. JDBC) selbst vorzunehmen.
- Es besteht keine Unabhängigkeit der Komponente von der konkreten Location, wo diese ablaufen, die Verwendung von Funktionen anderer Komponenten muss während der Entwicklung explizit berücksichtigt werden.
- Die durch eine JavaBean angebotene Schnittstelle kann sehr „breit“ ausfallen (d.h. mit einer Vielzahl angebotener Funktionen), was sich nachteilig auf deren Wiederverwendung auswirkt.
- Die verteilte Entwicklung von Anwendungen (d.h. die Berücksichtigung von Rollen wie z.B. Komponentenproduzenten und Komponentenkonsumenten) wird nicht explizit unterstützt.

Die vorgenannten Nachteile der Java-Bean-Technologie waren der Grund für die Entwicklung eines weiteren Komponentenmodells, der *Enterprise JavaBeans* (ebenfalls von SUN), die im folgenden Kapitel einer eingehenden Analyse unterzogen werden.

4 Enterprise JavaBeans

4.1 J2EE und EJB Spezifikation – ein Überblick

4.1.1 Bestandteile der J2EE-Spezifikation

Bei der Java 2 Enterprise Edition (kurz J2EE) handelt es sich um eine Plattform, die verschiedenste Technologien miteinander vereint. Erreicht wird so ein Architekturvorschlag für die Entwicklung internetbasierter mehrstufige Client/Server-Anwendungen, die insbesondere den Bereich von E-Buisiness-Anwendungen abdecken sollen. Diese Architektur soll die Implementierung skalierbarer, integrierter und verteilter Anwendungen unter Berücksichtigung eines heterogenen Umfeld unterstützen. Dabei soll durch das Angebot standardisierter Schnittstellen der Zugriff auf bereits vorhandene Systeme, wie z.B. SAP bzw. Host-Systeme, ermöglicht werden. Die J2EE-Spezifikation und auf dieser Basis implementierte Applikations-Server können so auch als Integrationsplattformen verstanden werden, die den Zugriff auf Legacy-Systeme, verteilte Objekte via CORBA, die Verwendung Message-orientierter Middleware- und Web-Services, aber auch den Zugriff auf Microsoft DCOM-Architekturen erlaubt. Durch diesen Architekturvorschlag soll der Entwickler von zeitaufwendigen Detailproblemen der Programmierung entlastet werden und sich voll auf die fachlich begründeten Funktionen der späteren Anwendung konzentrieren können. Abbildung 4.1 gibt eine grobe Übersicht zu den vielfältigen Einsatzszenarien dieser Architektur.

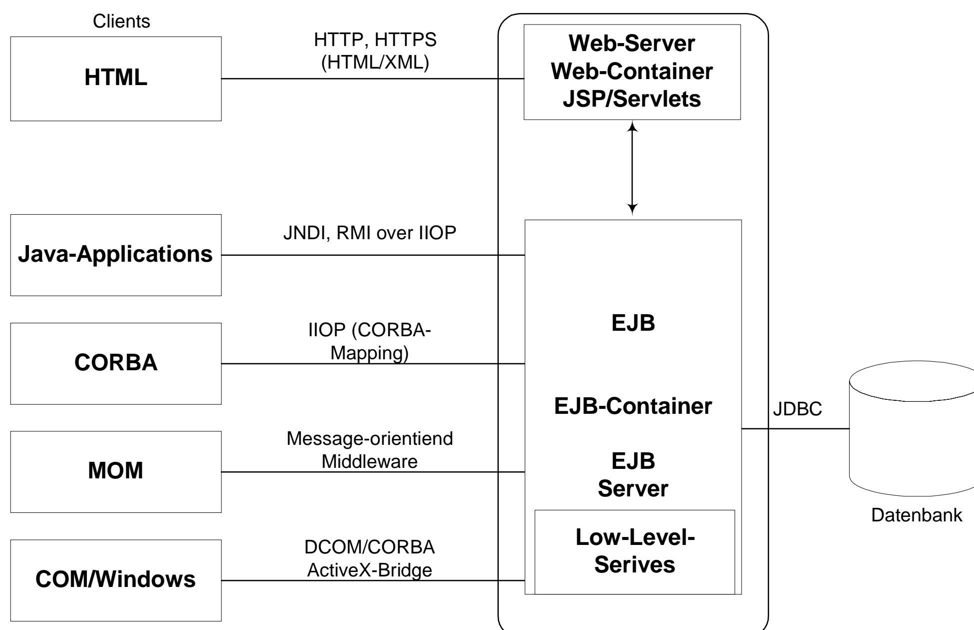

Abbildung 4.1: Möglichkeiten EJB-basierter Architekturen

Als wesentlichste funktionale Eigenschaften dieser Plattform sind die im Rahmen entsprechender Container ablaufenden EJBs (Enterprise JavaBeans), JSP's (Java Server Pages) und Servlets zu nennen. Während die EJBs eine Komponentenarchitektur zur Abstraktion serverseitig verwalteter Daten und Funktionen anbieten, können Servlets als ein primär funktionsorientiertes serverseitiges Komponentenmodell betrachtet werden die insbesondere im Rahmen entsprechender Web-Server (speziellen Web-Containern) ablaufen. Im Kontext mit den JSP's (JavaServer Pages) besteht der primäre Einsatz von Servlets in der Möglichkeit einer dynamischen Generierung von Webseiten unter Verwendung des HTTP-Protokolls.

Die bereits angesprochenen Container bieten im Falle der EJB-Komponenten sogenannte Laufzeitdienste an. Diese beziehen sich auf die Transaktionssicherung, die Zugriffssicherheit, den entfernten Zugriff auf die EJB-Komponenten über RMI (Remote Invocation Interface), die Verwaltung des Lebenszyklus der Komponenten und zu nutzende Persistenzmechanismen bzw. ein entsprechendes Pooling der genutzten Datenbankverbindungen. J2EE setzt dafür auf der Java 2 Standard Edition (kurz J2SE), welche die Kernfunktionalitäten des JDK (Java Development Kit) anbietet, auf. Dementsprechend bietet sich der Zugriff auf CORBA-, RMI-, JDBC- und JNDI-Funktionalitäten. Darüber hinaus können Applets und JavaBeans im Rahmen einer J2EE-konformen Anwendung verwendet werden, wobei diesen Komponenten insbesondere Aufgabenstellungen innerhalb entsprechender Client-Applikationen zukommen.

4.1.2 Zielstellungen der EJB-Architektur

Innerhalb der Spezifikation von SUN Microsystems finden sich die folgenden sehr allgemeinen Zielstellungen der EJB-Architektur. Die jeweils aktuellen Versionen finden sich unter der folgenden URL: *http://java.sun.com/products/ejb/docs.html*

1 Die EJB-Architektur bietet eine standardisierte Komponenten-Architektur für die Entwicklung verteilter objektorientierter Geschäftsapplikationen mit Hilfe der Programmiersprache Java. Die Entwicklung kann durch die Kombination vorgefertigter Komponenten verschiedener Hersteller erfolgen.

2 Die EJB-Architektur vereinfacht die Entwicklung der Applikation erheblich, da der Entwickler sich auf die Geschäftslogik konzentrieren kann und von solchen Details wie z.B. Transaktionssicherung und Persistenzmanagement weitgehend entlastet wird.

3 Eine EJB-Komponente folgt der Java-Philosopie „Write Once, Run Anywhere“ und kann somit auf unterschiedlichsten Herstellerplattformen zur Ausführung gebracht werden, Voraussetzung ist lediglich eine J2EE-konforme Umgebung.

4 Die EJB-Architektur berücksichtigt den kompletten Life-Cycle eines IT-Systems von der Entwicklung über die Installation bis hin zu Aspekten der Laufzeitumgebungen.

5 Die EJB-Architektur definiert einen entsprechenden Rahmen für die Entwicklung EJB-kompatibler Werkzeuge und Komponenten.

6 Die EJB-Architektur ist kompatible zu existierenden Applikation-Server-Plattformen, somit können existierende Produkte um J2EE-kompatible Funktionalitäten erweitert werden.

7 Die EJB-Architektur soll kompatibel zu anderen Programmierschnittstellen (API) der Java Programmiersprache sein.

8 Die EJB-Architektur soll die Interoperabilität zwischen Enterprise JavaBeans und Anwendungen, die nicht in Java realisiert wurden, gewährleisten.

9 Die EJB-Architektur ist kompatibel zu ausgewählten Teilen der CORBA-Architektur (Schwerpunkt: Kommunikationsprotokoll IIOP) und ermöglicht so die Interoperabilität in heterogenen Systemlandschaften.

Entsprechend den vorgestellten allgemeinen Zielstellungen, die weitgehend unabhängig von einer konkreten Version der EJB-Spezifikation sind, kann die EJB-Architektur sowohl als herstellerunabhängiger Vorschlag für mehrschichtige Client/Server-Applikationen als auch Integrationsplattform in heterogenen Systemumgebungen verstanden werden.

4.1.3 Das EJB-Komponentenmodell

Im folgenden ist die Definition der *Enterprise-Java-Beans-Architektur* wiedergegeben:

> *The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification. [SUN 1999]*

Bei den Enterprise JavaBeans Komponenten handelt es sich um kein Produkt, sondern um eine Spezifikation von Sun Microsystems, die eine Komponentenarchitektur definiert. Hersteller, wie z.B. Inprise, BEA, Sun, Oracle und IBM, bieten kommerzielle Produkte an, die dieser Spezifikation genügen. Bei diesen Produkten handelt es sich zum einem um sogenannte Applikation Server bzw. Container, welche die Laufzeitumgebung zur Ausführung der EJB-Komponenten bereitstellen und Entwicklungswerkzeuge die verschiedenen Aufgabenstellungen der Entwicklung und Einführung entsprechender komponentenbasierter Anwendungen unterstützen. Die Verwendung von EJBs wird insbesondere durch die Vorzüge der Wiederverwendung, einer arbeitsteiligen Softwareentwicklung und der Nutzung durch den Container angebotener Basis-Dienstleistungen motiviert.

- *Component Pooling und Lifecycle Management*: EJB's werden in entsprechenden Pools verwaltet und der Lebenszyklus durch den J2EE-Server gemanagt.
- *Client Session Management*: durch einen Client referenzierte EJB's behalten ihrer Zustand bei, auch wenn zwischenzeitlich durch einen anderen Client verwendet wurden.
- *Database Connection Pooling*: Datenbank-Ressourcen und die entsprechenden Verbindungen zu diesen werden durch den J2EE-Server verwaltet und potentiellen Interessenten bei Bedarf automatisch zugeteilt. Dieses Vorgehen ermöglicht eine effiziente Verarbeitung und die Einsparung von Lizenzen.
- *Transaktions Management*: Methoden der EJB können unter die Transaktionssicherung des Applikation-Servers gestellt werden
- *Security*: Authentifizierung und Zugriffskontrolle
- *Persistence*: (d.h. die dauerhafte Speicherung von Daten) kann durch den Container selbst realisiert werden.

Die durch Sun Microsystems vorgeschlagene Spezifikation der Enterprise JavaBeans bietet die Grundlage zur objektorientierten und komponentenbasierten Entwicklung verteilter Geschäftsapplikationen unter Verwendung der Programmiersprache Java. Dementsprechend bieten sich alle Vorteile, aber auch Nachteile der Programmiersprache Java im Umfeld der EJB-Entwicklung.

Die derzeit in der Version 2.0 vorliegende EJB-Spezifikation bietet eine Komponentenarchitektur für verteilte Anwendungen. Waren die JavaBeans mehr der Client-Seite und dort insbesondere grafischen Benutzerschnittstellen vorbehalten, so bietet die EJB-Spezifikation eine plattformneutrale Grundlage für Komponenten auf der Serverseite. EJB-Komponenen benötigen zur Ausführung einen EJB-Container (Laufzeitumgebung), welcher wiederum innerhalb eines entsprechenden Applikation-Servers zur Verfügung steht. Die im folgenden kurz angesprochenen Services werden den EJB-Komponenten über Standardprogrammierschnittstellen zur Verfügung gestellt:

Mit der Entlastung des Komponentenentwicklers von grundlegenden Diensten, wie der Transaktionssteuerung oder der Persistenz entsprechender Daten, ergeben sich Restriktionen für die Entwicklung der EJBs, wovon im folgenden einige aufgeführt sind.

- EJBs können keinen Gebrauch von der Multithreaded API machen,
- Klassen des AWT-Packages (Ein- und Ausgabeoperationen) können nicht genutzt werden,
- die Verwendung des I/O-Packages ist nicht möglich,
- kein Zugriff und Modifizierung der Security-Einstellungen.

4.1.4 J2EE-konforme Programmierschnittstellen

Zur Entwicklung J2EE-konformer Applikationen stehen dem Entwickler eine Vielzahl entsprechender Programmierschnittstellen zur Verfügung, durch die auf die Dienste des Containers zugegriffen werden kann. Diese sind durch die Anbieter J2EE-konformer Application Server entsprechend zu unterstützen. Im folgenden sollen die Schnittstellen bzw. durch sie adressierte Dienste kurz hinsichtlich ihrer wesentlichen Funktionalität charakterisiert werden, angegeben wurden darüber hinaus die Pakete, in denen sich die Java-Klassen befinden:

- *JNDI* – Das „Java Naming and Directory Interface" API bietet Java-Applikationen den einheitlichen Zugriff auf Namens- und Verzeichnisdienste wie LDAP, DNS, NIS, NDS, oder auch COS Naming. Über die Verwendung von JNDI können zum einem Referenzen auf im Netzwerk verteilte Software-Komponenten (natürlich auch Objekte) erhalten werden, zum anderen kann z.B. die Ausfallsicherheit J2EE-konformer Applikationen über die Verwendung der JNDI-Funktionalitäten realisiert werden.

  Pakete/Klassen von JNDI finden sich unter: `javax.naming`

- *JDBC* – Das Java Database Connectivity API bietet den einheitlichen Zugriff auf derzeit zumeist relationale Datenbanksysteme, wie z.B. Oracle, SQL-Server oder auch DB2. Dafür werden durch die Hersteller der Datenbankmanagement-Systeme entsprechende Treiber zur Verfügung gestellt. Bei den JDBC-Treibern werden 4 Typen unterschieden. Über dieses Interface kann sowohl die Struktur bzw. Schemadefinition der Datenbank (z.B. CREATE TABLE studenten) verwaltet werden, als auch auf die Inhalte der Datenbank zugegriffen werden (z.B. SELECT * FROM studenten WHERE alter > 30).

  Pakete/Klassen von JDBC finden sich unter: `javax.sql`

- *JMS* – Das Java Message Service API bietet die Möglichkeit der asynchronen Kommunikation (Nachrichten) von Java-Anwendungen untereinander bzw. zu Nicht-Java-Anwendungen über nachrichtenorientierte Middleware (kurz MOM). Insbesondere im Bereich sogenannter EAI-Systeme (Enterprise Application Integration) spielt diese Technologie eine zunehmende Rolle, lassen sich doch so heterogene Systeme lose und flexible miteinander koppeln.

  Pakete/Klassen von JMS finden sich unter: `javax.jms`

- *JTA/JTS* – Das Java Transaction API ermöglicht eine transaktionsorientierte Verarbeitung innerhalb der J2EE-konformenen Anwendungssysteme, wobei webbasierte Clients nicht erfasst werden können. Der Java Transaction Service (konform zum CORBA Transaction Service) bietet die Möglichkeit einer verteilten Transaktionsverarbeitung unter Verwendung des XA-Protokolls.

  Pakete/Klassen von JTA finden sich unter: `javax.transaction`

  Pakete/Klassen von JTS finden sich unter: `javax.transaction.xa`

- *Java IDL* – Die Java Interface Definition Lanaguage bietet J2EE-Anwendungen die Möglichkeit auf entfernte CORBA-Objekte bzw. –Dienste via IIOP zuzugreifen und bildet Eine der wesentlichsten Voraussetzungen zur Realisierung von Connectivity und Inrteroperabilität zu Systemen, die über CORBA angesprochen werden können.

  Pakete/Klassen von JavaIDL finden sich unter: `org.omg.CORBA`
- *JavaMail* – Das Java Mail API bietet den einheitlichen Zugriff auf verschiedenste Internet-Mail Protokolle an, so dass Mails erstellt, gesendet und empfangen werden können. Über das Java Activation Framework können Java-Applikation alle MIME-Type (Multipurpose Internet Mail Extension) kompatiblen Daten verarbeiten.

  Pakete/Klassen von JNDI finden sich unter: `javax.jms`

Auf die Darstellung der primär im Fokus stehenden J2EE-Komponenten mit der Enterprise JavaBeans-API, der Java Servlet-API sowie die Verwendung von Java Server Pages wurde an dieser Stelle nicht explizit eingegangen, da diese im folgenden vertiefend dargestellt werden.

4.1.5 Verteilte Softwareentwicklung durch das EJB-Rollenmodell

Vergleichbar der industriellen Fertigung besteht auch in der Softwareentwicklung der Bedarf einer arbeitsteiligen Entwicklung, wobei die jeweils Beteiligten sich hochgradig auf bestimmte Aufgaben/Rollen spezialisieren können. Dieser Notwendigkeit trägt das EJB-Modell durch die Aufteilung der Entwicklungsverantwortlichkeiten Rechnung. Auf der Basis wohldefinierter Schnittstellen können konkrete Aufgaben an verschiedene Unternehmen oder auch Personen innerhalb einer Organisation verteilt werden und so insgesamt eine effiziente Entwicklung unterstützt werden. Die zum Zeitpunkt der Erstellung dieses Buches gültige EJB-Spezifikation 2.0 enthält dafür die folgenden 7 Rollen:

Server Provider

Der Verantwortungsbereich des Server-Providers liegt in der Entwicklung/Bereitstellung des Application Servers, der die Laufzeitumgebung für EJB-Container bietet. Die Funktionalität des Application Servers berücksichtigt unter anderem die Prozess- und Threadverwaltung, notwendige Netzwerkanbindungen oder auch die Verwaltung unterliegender Hard- und Softwareressourcen. Die Anbieter kommen zumeist aus dem Bereich der Betriebssystem-, Entwicklungsumgebungs- und Datenbank-Hersteller.

Container-Provider

Die Idee besteht darin, Laufzeitumgebungen für Enterprise JavaBeans, sogenannte EJB-Container, getrennt vom Applikation-Server anbieten zu können. Im Schwerpunkt berücksichtigt der EJB-Container Funktionalitäten der Transaktionssicherung, der Security und Datenbankanbindung zur Realisierung der Persistenzmechanismen (Container Managed Persistence). Da die EJB-Architektur das Zusammenwirken von EJB-Container und unterliegendem Server nicht detailliert beschreibt, kommen potentielle Anbieter zumeist aus einem Haus.

Bean Provider

Der Bean Provider realisiert die Entwicklung der EJB-Komponenten, die dann in einem entsprechenden Container ablaufen können. Auf der Basis von Entity Beans (persistente Komponenten), Session Beans (transiente Komponenten) und Message Driven Beans (Komponenten zur Verarbeitung von Nachrichten) wird die benötigte fachliche Funktionalität entworfen und implementiert. Durch das Angebot vorgefertigter Services soll der Entwickler technische

Aspekte weitgehend außer acht lassen können und sich auf die Geschäftslogik konzentrieren. Dieser Idealvorstellung stehen in der Praxis häufig Probleme, wie die ungenügende Performance der genutzten Persistenzservices des Containers, gegenüber.

Im Einzelnen sind vom Bean Provider folgende Aktivitäten wahrzunehmen:

- Programmierung des Home-Interfaces der Bean,
- Programmierung des Component-Interfaces der Bean, in dem die für die Applikation benötigten Business-Methoden deklariert sind,
- Programmierung der Implementation-Klasse, die alle deklarierte Business-Methoden in das Remote-Interface implementiert,
- Liefern des Deployment-Deskriptors, der Informationen über die Komponente selbst sowie über externe Abhängigkeiten enthält,
- Paketieren der Bean-Interfaces, der Implementation-Klasse und des Deployment-Deskriptors in eine Datei im jar-Format (jar steht für Java-Archiv).

Application Assembler

Der *Application Assembler* (dt. etwa Anwendungsentwickler oder Bean-Monteur) ist ein Domain-Experte, der über ausreichendes Wissen der durch die Applikation zu unterstützenden Geschäftsprozesse besitzt. Auf der Basis fertiger, von Bean-Providern gelieferter EJB-Komponenten und weiterer, für die Anwendung benötigter Komponenten (z.B. Java Server Pages und Servlets) entwickelt er die eigentliche Anwendung. Seine Arbeit dokumentiert der Application Assembler ebenso wie der Bean Provider im Deployment-Deskriptor. Dabei werden insbesondere Informationen zur Transaktionsverarbeitung, Beziehungen zu anderen Komponenten oder auch Security-Einstellungen hinterlegt. Damit ist für den Bean Deployer klar, wie er die Komponenten innerhalb der Laufzeitumgebung zu installieren hat.

Bean Deployer

Der EJB-Deployer (dt. etwa EJB-Installateur) nimmt eine oder mehrere EJB-jar-Dateien und installiert sie in einer bestimmten Umgebung, zu welcher der EJB-Server und der EJB-Container gehören. Das dafür notwendige Wissen bezieht sich insbesondere auf die technischen Eigenschaften des verwendeten Application Servers. Dazu werden in der Regel die Tools des EJB-Server- bzw. Container-Providers verwendet. Der Deployer muss zusätzlich dafür sorgen, dass alle im Deployment-Deskriptor definierten Abhängigkeiten aufgelöst und die benötigten Ressourcen (z.B. in der Applikation verwendete Datenbanken) bereitgestellt werden.

Persistence Manager Provider

Diese mit der EJB-Spezifikation 2.0 eingeführte Rolle übernimmt das Mapping der Persistenzeigenschaften auf ein entsprechendes Datenbank-Management-System. Dementsprechend obliegt ihm die Konfiguration des Deployment-Deskriptors der betroffenen Entity-Beans. Dabei sind insbesondere eine abstrakte Schema-Beschreibung sowie die Beziehungen zwischen den Enterprise JavaBeans untereinander bzw. zu anderen Objekten festzulegen. Auf diese Weise kann beschrieben werden, wie das objektrelationale Mapping durchzuführen ist.

Systemadministrator

Der Systemadministrator ist für die Infrastruktur und für das Netzwerk des laufenden Systems verantwortlich. Das schließt insbesondere die Betreuung und Wartung des EJB-Servers, aber zumeist auch der genutzten Datenbankserver ein. Im einzelnen sind dabei Aufgabenstellungen der Softwareinstallation, der Nutzerverwaltung, der Realisierung von Verfügbarkeitsanforderungen, der Analyse verbrauchter Ressourcen (inkl. Performance-Monitoring) und des Tunings wahrzunehmen.

Die derzeitige Praxis besteht darin, die vorgestellten Rollen im wesentlichen auf drei relevante zu reduzieren:

- EJB-Entwickler (incl. Bean-Provider, Application-Assembler und Deployer),
- EJB-Administrator (Deployer und System Administrator),
- EJB-Server-Provider (enthält: Container-, Server und Persistence Manager-Provider).

Die Gründe dafür liegen unter anderem im noch nicht ausreichend vorhandenen Komponenten-Markt und der zum Teil schwierigen Abgrenzung durchzuführender Aufgabenstellungen der einzelnen Rollen. Insbesondere die in der EJB-Spezifikation 2.0 hinzugekommene Rolle des Persistenz-Managers trägt der hohen Komplexität einer effizienten Datenbank-Abbildung Rechnung.

4.2 Typen von Enterprise JavaBeans

4.2.1 Entwicklung der EJB-Spezifikation

Die EJB-Spezifikation wurde 1998 mit der Version 1.0 durch SUN Microsystems erstmals publiziert. Im Jahr 1999 folgte die Version 1.1 und im Jahr 2000 die Version 2.0. Im folgenden sollen ausgewählte Eigenschaften bzw. vorhandene Unterschiede zwischen den jeweiligen Version kurz vorgestellt werden.

EJB-Spezifikation Version 1.0

- Bean-Typen: Session Beans, Entity Beans (optional)
- Aufruf von Funktionen: synchron
- Deployment-Deskriptor: serialisierte Klasse
- Kommunikationsprotokoll: Java-RMI (Remote Method Invocation)
- Benötigtes JDK: Version 1.1

EJB-Spezifikation Version 1.1

- Bean-Typen: Session Beans, Entity Beans
- Aufruf von Funktionen: synchron
- Deployment-Deskriptor: XML-konforme Datei
- Kommunikationsprotokoll: Java-RMI over IIOP und CORBA/IIOP (optional)
- Benötigtes JDK: Version 1.2

EJB-Spezifikation Version 2.0

- Bean-Typen: Session Beans, Entity Beans, Message-driven Beans

- Aufruf von Funktionen: synchron und asynchron
- Deployment-Deskriptor: XML-konforme Datei
- Kommunikationsprotokoll: Java-RMI over IIOP und CORBA/IIOP
- Datenbank-Mapping EJB QL (neue Rolle des Persistenz-Managers)
- Relationen zwischen EJB's
- Benötigtes JDK: Version 1.3.

In keiner der bisher veröffentlichten Spezifikationen wird von einer komponentenbasierten Vererbung gesprochen, d.h. wenngleich innerhalb einer Komponente die Eigenschaften der Vererbung entsprechend dem Java-Sprachstandard genutzt werden kann, so ist dieses auf der Ebene der Komponenten nicht möglich. Im Vordergrund der EJB-Spezifikation stehen die unterschiedlichen Typen angebotener EJB-Komponenten. Abbildung 4.2 zeigt die derzeit verfügbaren Typen.

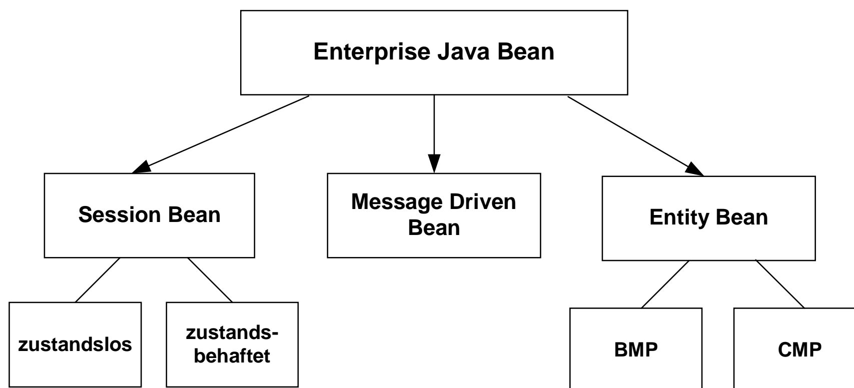

Abbildung 4.2: Komponenten-Typen in der EJB-Spezifikation Version 2.0

Der folgende tabellarische Vergleich zeigt zusammenfassend die unterschiedlichen Zielstellungen der EJB-Typen:

Tabelle 1: Vergleich der EJB-Komponententypen

Merkmal	Session Bean	Entity Bean	Message Driven Beans
Aufgabe	Repräsentiert einen Server-seitigen Dienst, der Aufgaben für einen Client ausführt	Repräsentiert ein Business-Objekt mit dauerhaft gespeicherten Daten	Ähnelt vom Verhalten her zustandslosen Session Beans, dient dem asynchronen Auslösen von Aktionen durch den JMS
Zugriff	Stellvertreter des Nutzers; private Ressource, steht ihm exklusiv zur Verfügung	Zugriff durch mehrere Nutzer; zentrale Ressource	Besitzen kein Home- und Remote-Interface (Message Listener)
Persistenz	nicht persistent; wenn der verbundene Client oder der Server terminiert ist, so ist auch die Bean nicht mehr verfügbar	persistent; Speicherung des Zustandes auf einem persistenten Speichermedium; Wiederherstellung der Bean zu einem späteren Zeitpunkt	nicht persitent

Merkmal	Session Bean	Entity Bean	Message Driven Beans
Unter-typen	*zustandslos*: Speichern von einem Methodenaufruf zum nächsten, aber keine Daten *zustandsbehaftet*: Speichern von Daten über mehreren Methodenaufrufe hinweg; können persistent gespeichert werden, sind aber nicht über einen Primärschlüssel ansprechbar	*BMP*: Bean selbst verantwortlich, dass ihre Daten persistent gemacht werden *CMP*: EJB-Container ist für die Persistenz der Daten verantwortlich	keine

4.2.2 Aufgaben des Home und Remote-Interfaces

Der Zugriff eines Clients auf die Funktionalitäten eines EJB's (speziell Entity- und Session-Beans) erfolgt über wohldefinierte Schnittstellen. Im einzelnen

- bietet das Remote-Interface die fachlich begründeten Funktionalitäten des EJB's an,
- ermöglicht das Home-Interface die Steuerung/Kontrolle des EJB-Lebenszyklus.

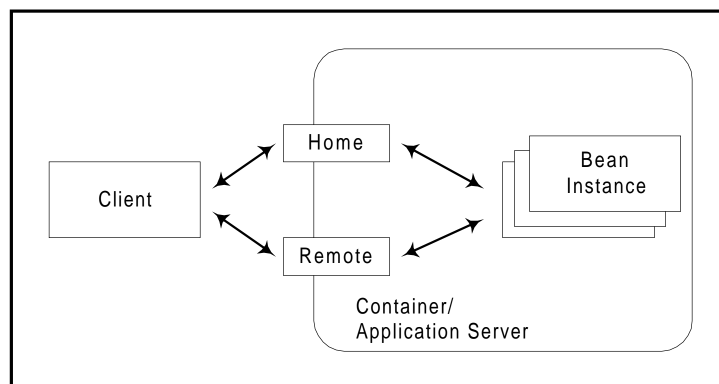

Abbildung 4.3: Zugriff auf die Bean-Instanzen

Damit ein Client auf eine konkrete EJB-Instanz zugreifen kann, benötigt er eine Referenz auf das entsprechende Home-Interface. Diese erhält der Client zum Zeitpunkt der Ausführung über den Namens- und Verzeichnis-Dienst JNDI (Java Naming and Directory Interface). Um dieses zu ermöglichen, ist jede durch einen Client referenzierbare Bean bei einem entsprechenden JNDI-Server registriert.

Der Zugriff auf die Methoden einer Bean-Instanz kann niemals direkt durch einen Client erfolgen. Dafür wird ein EJB-Objekt (im englischsprachigen Raum zumeist als „*glue*" bezeichnet) zwischen Client und Enterprise JavaBean verwendet, welches die Funktionalitäten des Component-Interfaces anbietet. Über dieses mit Hilfe des EJB-Containers verwaltete Interface wird ein Methodenaufruf an die eigentliche Bean-Instanz delegiert. Die sich daraus ergebenden Vorteile beziehen sich auf die Ortstransparenz einer Bean-Instanz (d.h. der Client muss nicht wissen, auf welchem physikalischen Server die Komponente abläuft), die Trennung von Implementierung der Funktionalität und Deklaration innerhalb der verwendeten Schnittstelle und die Möglichkeit der Einführung eines automatisierten Managements (Transaktionsverwaltung, Security, Persistenz,…).

4.2.3 Zustandslose und zustandsbehaftete Session Beans

Eine Session Bean realisiert die Funktionalität eines im Rahmen der Applikation zu berücksichtigenden Geschäftsprozesses und darin enthaltener Anwendungsfunktionen. Ein typisches Beispiel dafür ist die Abbildung der Funktionen eines Warenkorbs innerhalb eines webbasierten Verkaufsportals. Die fachlich begründeten Funktionen bestehen z.B. darin, Produkte in diesen Korb zu legen, die Gesamtkosten aller eingelegten Produkte zu ermitteln oder aber auch Produkte zu entfernen.

Es werden zwei Arten von Session Beans unterschieden:

- Die *Stateful Session Bean* (SFS) besitzt einen sogenannten Conversational State, einen zustandsbehafteten Status. Dadurch wird eine solche Bean über ihren gesamten Lebenszyklus einem Client exklusiv zur Verfügung gestellt. Bei der Erzeugung der Instanz muss deshalb auch eine Stateful Session Bean initialisiert werden. Über einen Aktivierungs- und Passivierungsmechanismus sorgt der EJB-Container dafür, dass nicht genutzte Instanzen in einen zweiten Speicher (z.B. Festplatte) ausgelagert werden können, um so einen Überlauf an Bean-Instanzen im EJB-Container zu vermeiden. Der Zustand wird in der Regel nur im Hauptspeicher gehalten (oder temporär auf die Platte ausgelagert), überlebt allerdings einen Absturz oder Neustart des Servers nicht.

- Eine *Stateless Session Bean* (SLS) ist eine zustandslose Instanz, muss nicht initialisiert werden und steht einem Client nur für die Zeit eines Methodenaufrufes zur Verfügung. Ein wesentlicher Unterschied liegt aber in der Tatsache, dass eine Stateless Session Bean keine Aktivierung bzw. Passivierung benötigt. Durch die nur kurzfristige Bindung der Instanz an den Client kann diese Instanz nach Abarbeitung des Methodenaufrufes an jeden beliebigen Client weitergereicht werden, um dessen Anforderung zu bearbeiten. Daraus folgt, dass weitaus weniger Instanzen vorhanden sein müssen, als Clients angeschlossen sind.

Zusammenfassend sollte festgehalten werden, dass eine Stateful Session Bean Daten (d.h. die darin enthaltenen Objektattribute) zwischen Client-Zugriffen speichern kann, eine Stateless Session Bean dagegen nicht.

4.2.4 Persistenzbehaftete Entity Beans

Eine Entity Bean repräsentiert entsprechend seinem Namen die Sicht auf dauerhaft gespeicherte Daten (Entitäten). Dafür besitzt es einen eindeutigen Schlüssel zur Identifizierung und erlaubt den gemeinsamen bzw. konkurrierenden Zugriff mehrerer Clients. Eine Entity Bean existiert so auch nach einem serverseitigen Zusammenbruch, da es unter Transaktionskontrolle des Application Servers bzw. des Datenbank-Servers steht.

Die Entity-Beans (sie erben die Fähigkeiten aus javax.ejb.EntityBean) bieten den Zugriff auf innerhalb eines Datenbanksystems persistent gespeicherte Datenobjekte. Derzeit werden die Persistenzeigenschaften zumeist auf relationalen Datenbankmanagement-Systemen (z.B. Microsoft SQL-Server, Oracle, IBM DB2) abgebildet. Mittels des JDBC-Interfaces kann eine vom konkreten Datenbanksystem völlig losgelöste Implementierung der Entity-Beans erfolgen, wodurch eine volle Datenbankportabilität erreicht werden kann. Wird ein derartiges Vorgehen gewählt, spricht man von sogenannten CMP-Beans (Container Managed Persistence), d.h. erst beim Vorgang des Deployments werden die Persistenzeigenschaften konkreter Attribute der Beans festgelegt. Dafür müssen diese Attribute innerhalb des Deployment-Deskriptors explizit festgelegt werden.

Ebenfalls möglich ist es, die Persistenzeigenschaften unter die Kontrolle des Entwicklers zu stellen. In diesem Fall wird von sogenannten BMP-Beans (*Bean Managed Persistence*) gesprochen. Der Entwickler hat hier innerhalb der Komponenten die datenbankspezifischen Zugriffsoperationen selbst zu implementieren. Darunter fallen der Auf- und Abbau einer Datenbankverbindung und die auf der Datenbank auszuführenden Operationen (z.B. SQL-Statements). Insbesondere die Abbildung auf Datenbanksysteme (z.B. objektorientierte Datenbanksysteme), die nicht über das JDBC-Interface angesprochen werden können, kann so unterstützt werden. Die Gefahr eines solchen Vorgehens besteht allerdings in der nicht mehr vorhandenen Unabhängigkeit vom konkret eingesetzten Datenbanksystem, ebenfalls nicht mehr nutzbar ist die Zugriffsoptimierung des Application Servers.

4.2.5 Nachrichten gesteuerte Beans (Message Driven Beans)

Mit den Message Driven Beans wird ein weiterer EJB-Typ eingeführt, der für die Bearbeitung von über den JMS (Java Message Service) eingehenden Nachrichten verwendet werden kann. Die API des JMS bietet dem Java-Entwickler die Möglichkeit eines standardisierten und herstellerneutralen Zugriffs auf sogenannte MOM-Architekturen (Message orientierte Middleware), wie z.B. IBM MQSeries. Auf diese Weise kann von einem JMS-Client (z.B. Session Bean) eine Nachricht asynchron in eine entsprechende Warteschlange eingestellt werden. Der Message Broker übernimmt dann die Verantwortung für die Zustellung dieser Message an die Warteschlange des Zielsystems. Aus dieser kann dann mittels eines auf der Warteschlange des Zielsystems registrierten Message Driven Beans (Message Listener) die entsprechende Nachricht herausgelesen und weiterverarbeitet werden. Anders als bei einem synchronen Aufruf kann der JMS-Client sofort weiterarbeiten und wird nicht bis zu einer eingehenden Antwort des Nachrichtenempfängers blockiert.

Im Gegensatz zu den bereits dargestellten EJB-Typen verfügen Message Driven Beans über kein Home- und kein Remote-Interface. Dementsprechend können sie nicht durch ein Client angesprochen werden

Die EJB-2.0-Spezifikation unterstützt demnach die JMS-Integration in folgender Weise:

- Die Verwendung der JMS-API erfolgt typischerweise innerhalb einer Session Bean zum Senden einer entsprechenden Nachricht über den Message-Broker.

- Die Message Driven Bean dient als Empfänger der Nachricht. Dafür werden mittels eines entsprechenden Listeners potentielle Warteschlangen auf eintreffende Nachrichten hin überwacht.

Eine JMS-konforme Nachricht besteht aus einem Header und einem Body. Während der Header entsprechende Routing-Informationen und Meta-Daten enthält, handelt es sich beim Body um den eigentlichen Inhalt der Nachricht. Der Inhalt der Nachricht kann z.B. ein serialisiertes Objekt, einfacher Text oder auch eine XML-Datei sein.

Die Transaktionssicherung der Message Driven Beans kann sowohl durch die EJB (CMT) selbst, als auch durch den Container realisiert werden. Im Falle der Realisierung durch den Container kann der Effekt einer Endlosschleife auftreten, wenn nämlich ein entsprechendes Rollback durch den Container ausgelöst wird. In diesem Fall verbleibt die Nachricht in der Warteschlange des Message Brokers und wird erneut an die Message Driven Beans versendet, ein abermaliges Rollback führt dann zu einer entsprechenden Endlosschleife.

4.2.6 Aufgaben des Deployment-Deskriptors

Der Vorgang des Deployments (Einführung/Verteilung) der Enterprise JavaBeans bezeichnet die Installation einer EJB-Komponente innerhalb eines entsprechenden Containers. Dieser stellt die zur Ausführung der Komponente benötigte Umgebung (Services und Ressourcen) bereit. Für das Deployment wird ein sogenannter Descriptor verwendet. Dieser erlaubt die Anpassung vielfältiger Eigenschaften (z.B. Transaktionsverhalten, Persistenz, Security,...). Wurden die darin enthaltenen Informationen innerhalb der EJB-Spezifikation 1.0 noch auf entsprechende serialisierte Klassen verteilt, erfolgte ab der EJB-Version 1.1 die Verwendung eines sogenannten XML-Deployment-Descriptors. Dieser kann auf unterschiedliche Weise erstellt werden. Zum einem über die Verwendung eines allgemein zugänglichen Editors bzw. Command-Line-Interface oder aber im Rahmen durch die Application Server angebotener Werkzeuge (typischerweise Wizards), die den Deployment-Vorgang durch ein visuelles Interface unterstützen. Die derzeitigen Erfahrungen zeigen jedoch, dass eine manuelle Nachbearbeitung des Deployment-Descriptors durchaus erforderlich sein kann.

Auf der Basis des Deployment-Descriptors soll eine Anpassung entsprechender Attribute der EJB‘s ermöglicht und ein „Blackbox-Reuse“ unterstützt werden. Der Deployment-Descriptor kann als „Beipackzettel“ für eine EJB oder für eine aus mehreren EJBs zusammengesetzten Applikation betrachtet werden.

Folgende Eigenschaften (Attribute) sind im Deployment-Descriptor konfigurierbar:

- Sicherheitsattribute,
- Transaktionsattribute,
- Umgebungsvariablen,
- Verknüpfungen mit anderen Komponenten,
- Verknüpfungen zu Datenquellen.

Die Grobstruktur eines EJB-Deployment-Descriptors besteht aus:

- Struktur-Sektion (Basis- und Umgebungsinformationen),
- Beschreibungssektion (verwendete EJB-Namen und ggf. –Icons)
- Assemblierungssektion (Zugriffsrechte, Transaktionsattribute,...).

4.3 Beispiel der Implementierung eines EJB’s

4.3.1 J2EE Referenz-Umgebung

Das Software Development Kit „Java™ 2 SDK, Enterprise Edition 1.3“ enthält neben den eigentlichen Java-Klassen auch eine umfangreiche Sammlung von Werkzeugen für die Entwicklung und Ausführung von J2EE-kompatiblen Anwendungen. Mit dieser Referenzumgebung werden primär zwei Zielstellungen verfolgt. Zum einen wird potentiellen Anbietern eine Richtlinie hinsichtlich der benötigten Funktionalität von J2EE-konformen Applikations-Servern und Containern zur Verfügung gestellt, zum anderen bietet sich eine kostenfreie Laufzeitumgebung, die ein Erlernen der J2EE-Funktionalität ermöglicht. Die kommerzielle Verwendung dieser Umgebung ist dabei untersagt.

Neben dem J2EE-konformen Server, der durch den Aufruf von „j2ee –option“ von der Kommandozeile aus gestartet werden kann, enthält die Referenzumgebung auch die Cloudscape-Datenbank zur Gewährleistung der Persistenz verwendeter Entity-Beans. Diese Datenbank kann von der Kommandozeile aus gestartet bzw. gestoppt werden, ebenso besteht die Möglichkeit eines SQL-basierten Zugriffs.

Im folgenden sollen die Werkzeuge der J2EE-Umgebung kurz vorgestellt werden:

- Administration Tool (Kommandozeile)

  Das “j2eeadmin” Werkzeug wird zum Hinzufügen und Entfernen verschiedenster Ressourcen verwendet. Dieses können z.B. JDBC-Datenbanktreiber, allgemeine Datenquellen (z.B. OODBMS), JMS-Ziele (Queue oder Topic) oder auch Connection Factories sein.

- Application Deployment Tool (Kommandozeile oder GUI)

  Das „deploytool“ Werkzeug unterstützt die Erzeugung von J2EE-Komponenten (Packeten), den Vorgang der Konfiguration und das eigentliche Einbringen der kompletten Applikaiton in den J2EE-Server.

- Key Tool (Kommandozeile)

  Das „keytool“ Werkzeug kann zum Erzeugen von public und private Schlüsseln (keys) und zur Generierung von X509 kompatiblen Zertifikaten verwendet werden. Darüber hinaus wird die Verschlüsselung mittels RSA-Algorithmus unterstützt.

- Realm Tool (Kommandozeile)

  Das „realmtool“ Werkzeug erlaubt das Management von Nutzern des J2EE-Servers.

- Verifier (Kommandozeile oder GUI)

  Das “verifier”-Werkzeug prüft J2EE-Applikationen und -Komponenten auf die Einhaltung der J2EE-Spezifikation. Möglich ist die Prüfung von J2EE-Applikationen (EAR File), von Enterprise JavaBeans (EJB JAR File), von Web-Komponenten (WAR File) und J2EE-Client-Applikationen (JAR File). Es kann sowohl von der Kommandozeile, mit einer eigenständigen GUI oder aber innerhalb des Deployment Tool verwendet werden.

- Packager (Ausgeführt von der Kommandozeile)

  Das “packager” Werkzeug ermöglicht die Erzeugung von J2EE-kompatiblen Komponenten. Möglich ist die Erzeugung der folgenden Pakete:

  - Enterprise JavaBean JAR File
  - Web Applikation WAR File
  - Applikation Client JAR File
  - J2EE Applikation EAR File
  - Ressource Adapter RAR File

Darüber hinaus erlaubt dieses Werkzeug das Setzen von für die Ausführung benötigten Informationen einer J2EE-Anwendung.

- Cleanup Script (Ausgeführt von der Kommandozeile)

  Das "cleanup"-Werkzeug löscht alle innerhalb des J2EE-Applikation-Servers installierten Anwendungen und die dazugehörigen log-Files. Gelöscht werden nicht die entsprechenden Archive (JAR, WAR, EAR).

Für die praktische Arbeit werden insbesondere der Applikations-Server, die Datenbank und Werkzeuge für die Installation (Deployment) von EJB-Komponenten. Die folgende Übersicht beschreibt kurz die enthaltenen Werkzeuge.

Eine ausführliche Dokumentation der im Java 2 SDK (Enterprise Edition 1.3) enthaltenen Werkzeuge findet sich unter folgenden Verzeichnis des SDK: ..\j2sdkee1.3\doc\index.html

4.3.2 Schritte zur Implementierung und Ausführung eines EJB's

Abbildung 4.4 verdeutlicht die notwendigen Schritte zur Implementierung und Ausführung eines EJB's bzw. der Fertigstellung einer EJB-basierten Applikation. Bei diesen Schritten erfolgt zuerst die Implementierung der Java-Quellcodedateien (*.java) innerhalb eines Editors bzw. einer Entwicklungsumgebung, wie z.B. dem JBuilder von Borland. Nach dem Übersetzen (Compilerlauf) der Java-Quellen in entsprechende Java-Bytecode (*.class) erfolgt der eigentliche Vorgang der Paketierung, wobei eine entsprechende *.jar-Datei erzeugt wird. Dieses Paket enthält zusätzlich zu den Java-Klassen den bereits vorgestellten XML-Deployment-Descriptor und eine sogenannte Manifest-Datei (*.mf). Sowohl der Deployment-Descriptor als auch die Manifest-Datei werden bei Verwendung der hier betrachteten J2EE-Referenzumgebung automatisch erzeugt.

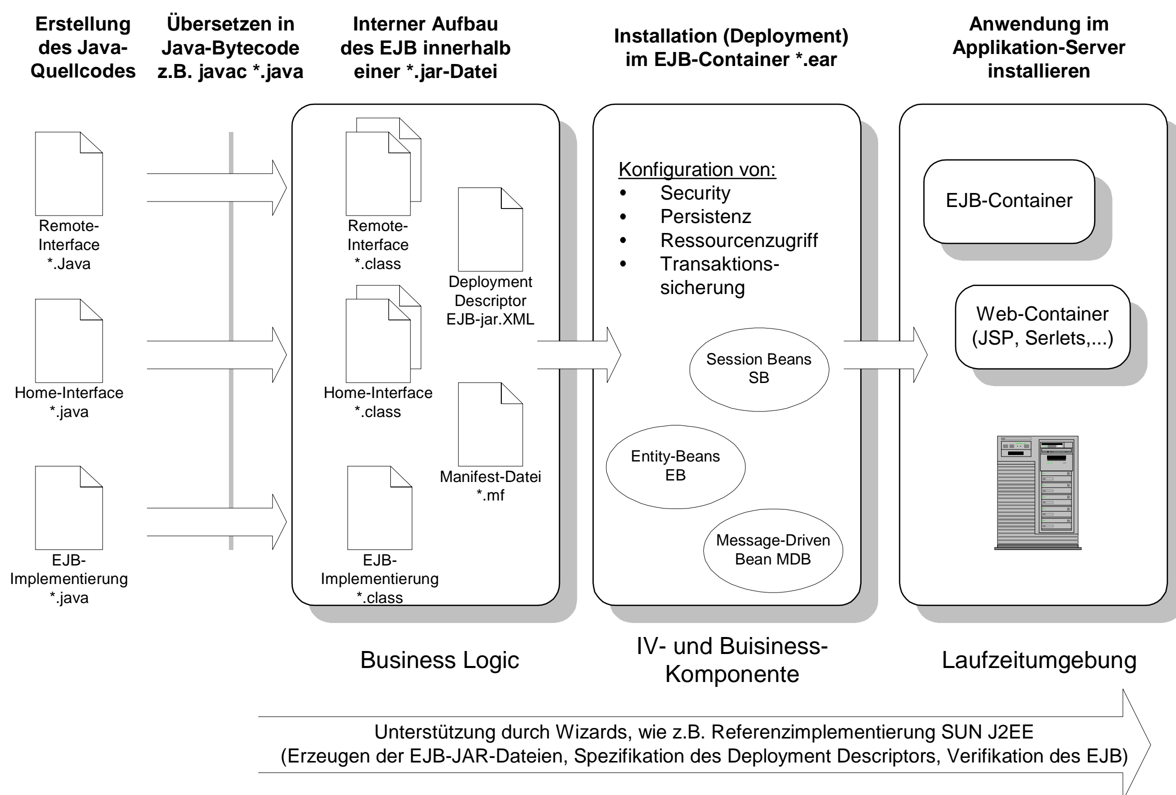

Abbildung 4.4: Übersicht zur Vorgehensweise der Entwicklung/Verteilung

4.3.3 Implementierung der Server-Anwendung

Im folgenden soll die Implementierung einer einfachen Server-Anwendung, die durch genau eine EJB hinsichtlich der gegebenen Funktionalität repräsentiert wird, vorgestellt werden. Ziel soll es sein, eine Stateless Session Bean zu entwickeln, das zwei Methoden anbietet. Eine Methode dient der Umrechnung von DM in EURO, die andere soll EURO in DM umrechnen. Anhand dieses Beispiels sollen die Schritte bis zur lauffähigen Applikation erklärt werden.

Home-Interface der EJB

Das Home-Interface übernimmt die Steuerung des Life-Cycle des EJB. Im einzelnen sind dieses die Erzeugung (create), die Beseitigung (remove) und die Lokalisierung (find) einer EJB-Komponente.

```
package example1;

/**
 * Title:        EJB book examples
 * Description:  Home-Interface der Bean
 * Copyright:    Copyright (c) 2002
 * Company:      private
 * @author schmietendorf/dimitrov/dumke
 * @version 1.0
 */

import java.rmi.*;
import javax.ejb.*;

public interface EuroCalcHome extends EJBHome {

  public EuroCalcRemote create() throws CreateException, RemoteException;
}
```

Da es sich um eine Stateless Session Bean handelt, besitzt die create-Methode keine Parameter. Dem Aspekt einer potentiell verteilten Anwendung trägt die Ausnahmebehandlung beim Erzeugen der EJB Rechnung. Die `javax.ejb.CreateException` wird bei Problemen des Containers „geworfen“ (throw), im Falle von Problemen beim entfernten Methodenaufruf die `java.rmi.RemoteException.`

Remote-Interface der EJB

Dieses Interface spezifiziert die nach außen sichtbaren fachlichen Funktionalitäten der EJB-Komponente, bei dem hier gewählten Beispiel die Funktionen euro_to_dm und dm_to_euro. Potentielle Fehler beim Aufruf dieser Funktionen werden wiederum durch eine entsprechende Ausnahmebehandlung abgefangen.

```
package example1;

/**
 * Title:        EJB book examples
 * Description:  Funktion eines Session Beans (Währungsberechnung)
 * Copyright:    Copyright (c) 2002
 * Autoren:      dimitrov/dumke/schmietendorf
 * Version:      1.0
 */

import java.rmi.*;
import java.lang.*;
```

```
import javax.ejb.*;

public interface EuroCalcRemote extends javax.ejb.EJBObject {

  // Umrechnung Euro-Betrag in DM
  public double euro_to_dm(double amount) throws RemoteException;

  // Umrechnung DM-Betrag in Euro
  public double dm_to_euro(double amount) throws RemoteException;
```

Implementierung der Bean-Klasse

Die Implementierung der durch die Komponenten gebotenen Funktionen erfolgt innerhalb dieser Klasse. Neben den fachlich begründeten Funktionen benötigen die EJB-Komponenten entsprechende Funktionen (`ejbCreate(), ejbRemove(); ejbActivate(), ejbPassivate()`) zur Steuerung des Lebenszyklus.

```
package example1;

/**
 * Title:        EJB book examples
 * Description:  Funktion eines Session Beans (Währungsberechnung)
 * Copyright:    Copyright (c) 2002
 * Autoren:      dimitrov/dumke/schmietendorf
 * Version:      1.0
 */

import java.rmi.*;
import javax.ejb.*;
import javax.naming.*;


public class EuroCalc implements SessionBean {

  private SessionContext sessionContext;

  //Umrechnungskurs der DM zu einem Euro
  double changerate = 1.95583;

  //Berechnung des Währungsbetrages in DM
  public double euro_to_dm(double amount) throws ArithmeticException {
    double dm = 0;
    dm = amount*changerate;
    System.out.println(dm);
    return dm;
  }

  //Berechnung des Währungsbetrages in Euro
  public double dm_to_euro(double amount) throws ArithmeticException {
    double euro = 0;
    euro = amount/changerate;
    System.out.println(euro);
    return euro;
  }

  //Methoden zur Steuerung des Lebenzyklus des EJB
  public void ejbCreate(){}
  public void ejbRemove(){}
  public void ejbActivate(){}
  public void ejbPassivate(){}
```

```
//Ermöglicht den Zugriff auf den Kontext des EJB
public void setSessionContext(SessionContext context){
sessionContext = context;
}
```

Nach der Compilierung dieses Java-Source-Codes sollten die entsprechenden Klassen für den weiteren Vorgang des Deployments zur Verfügung stehen. Für die Implementierung der beiden Interfaces (Home und Remote) wird kein Code benötigt, da dieser während des Deloyments automatisch generiert wird.

4.3.4 Verteilung (Deployment) der Server-Anwendung

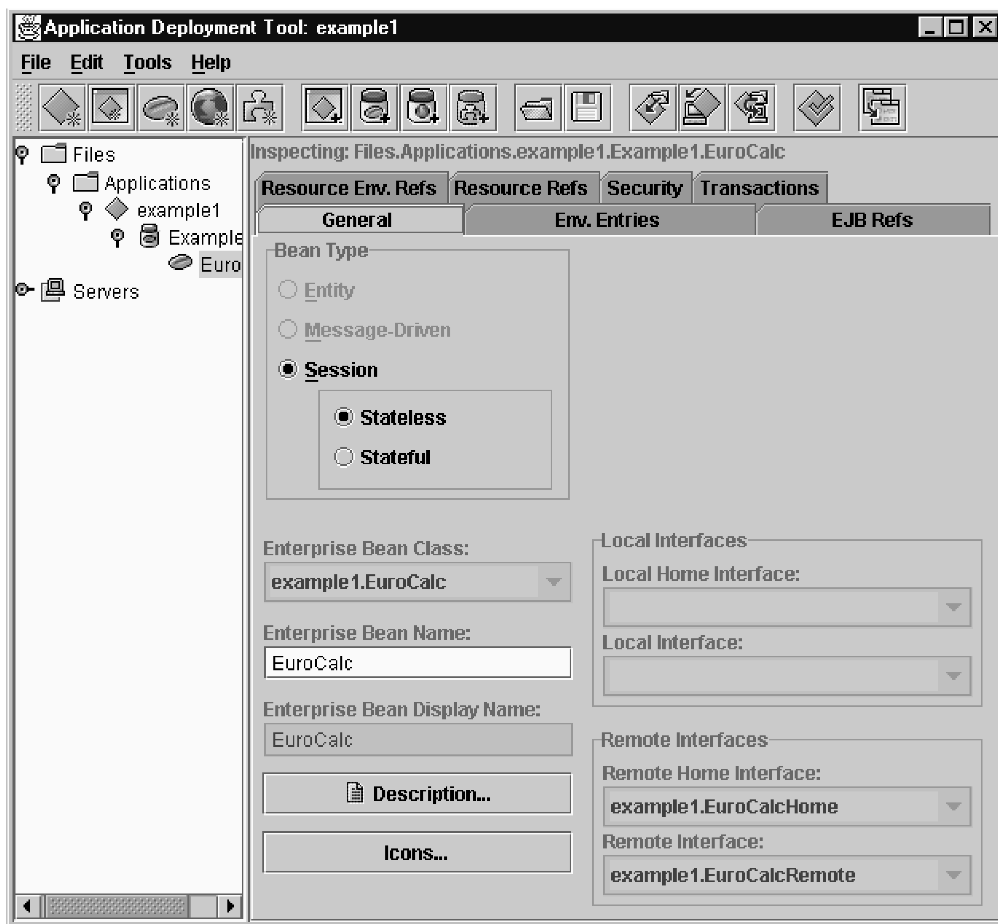

Abbildung 4.5: Oberfläche des Deployment-Tools

Nach der Implementierung der Server-Anwendung erfolgt im nächsten Schritt das sogenannte Deployment. Im Falle der hier verwendeten J2EE-Referenzumgebung bieten sich die folgenden Grundfunktionalitäten,die innerhalb des Deployments durchzuführen sind:

- Aufsetzen J2EE-Komponenten-basierter Applikationen,
- Erzeugen und Konfigurieren von EJB-Komponenten auf Basis der Java-Klassen,
- Erzeugen und Konfigurieren von Web-Komponenten (Servlets und JSP's),
- Erzeugen von Client-Anwendungen für den Zugriff auf die EJB-Anwendungen,
- Speichern der kompletten Applikation auf externen Speicher,
- Hinzufügen und Entfernen von JAR-, WAR-, EAR-, RAR-Archive,

- Auslösen der Installation (deployment) der J2EE-Komponenten im Application Server,
- Analyse der Komponenten auf J2EE-Konformität.

Für diese Aufgabenstellung bietet die Referenzumgebung das bereits vorgestellte „deploytool“, versehen mit einer grafischer Nutzeroberfläche, an. Nach dem Start bietet sich die folgende Oberfläche, die hier bereits mit der eingebrachten EJB EuroCalc versehen ist.

Abbildung 4-6 zeigt eine Applikation mit einer installierten EJB (EuroCalc), welches die Funktionalität der Währungsumrechnung realisiert. Um diese EJB innerhalb des Application Servers zu installieren, ist im ersten Schritt eine neue Applikation anzulegen und danach mittels des sogenannten EJB-Wizard die neue EJB zu erzeugen. Im folgenden werden alle dabei zu durchlaufenden Dialoge wiedergegeben.

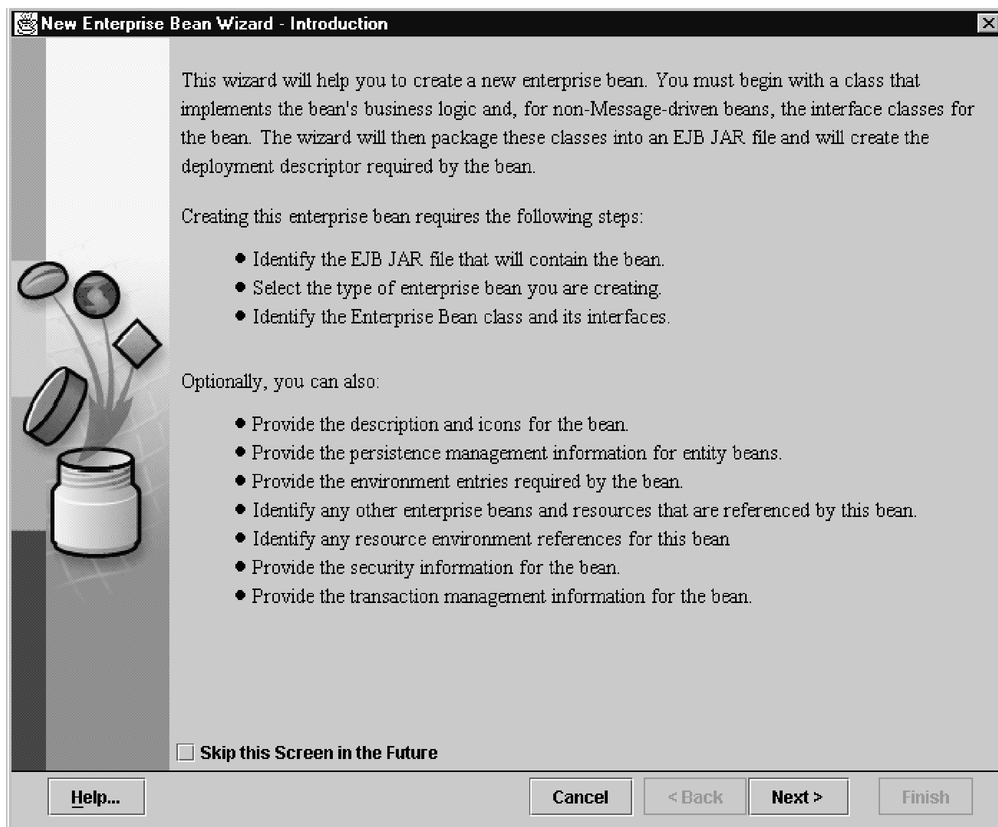

Abbildung 4.6: Eingangsdialog des EJB-Wizard

Der Eingangsdialog des EJB-Wizard (Abbildung 4.6) gibt eine kurze Übersicht zur gebotenen Funktionalität dieses Werkzeugs.

- Festlegung des Namens der Java-Archive und darin enthaltener Klassen und Interfaces,
- Definition des konkreten EJB-Typs (SessionBean, EntityBean, MessageDrivenBean),
- Festlegung potentieller Referenzen der betrachteten EJB-Komponente zu anderen EJBs oder aber benötigten Ressourcen,
- Festlegung von Persistenz-, Security- und Transaktionseigenschaften,
- Festlegung von Umgebungsvariablen des EJB’s.

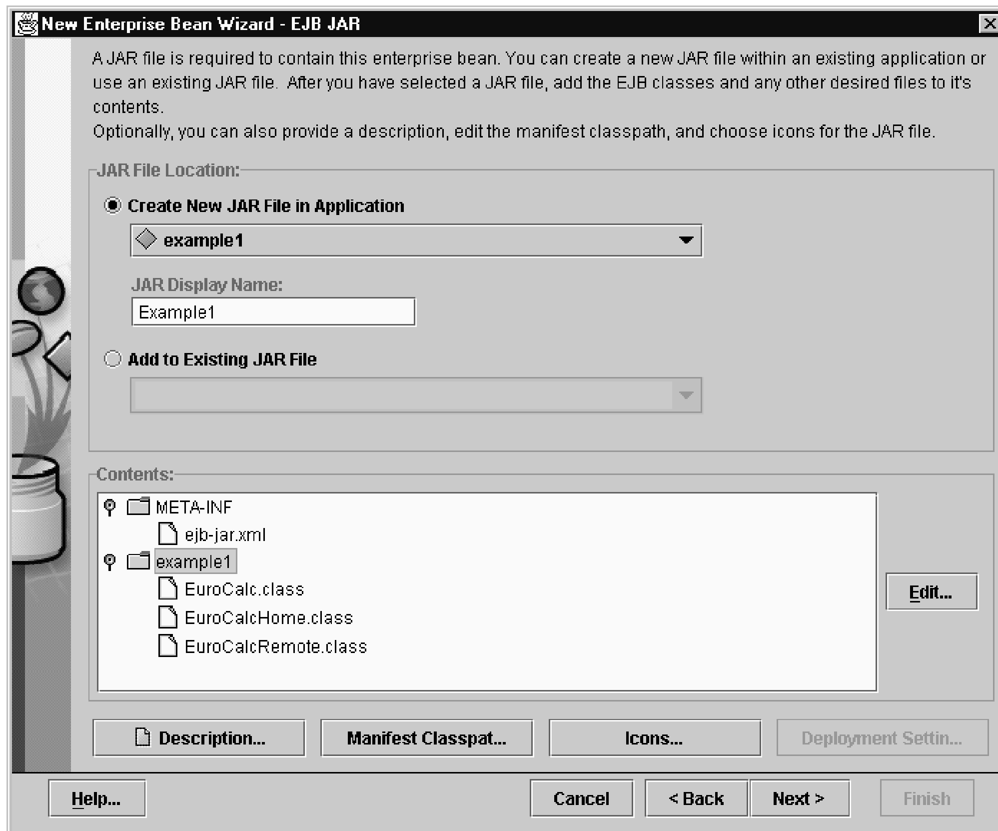

Abbildung 4.7: Erzeugen eines JAR-Files innerhalb der Applikation (example1)

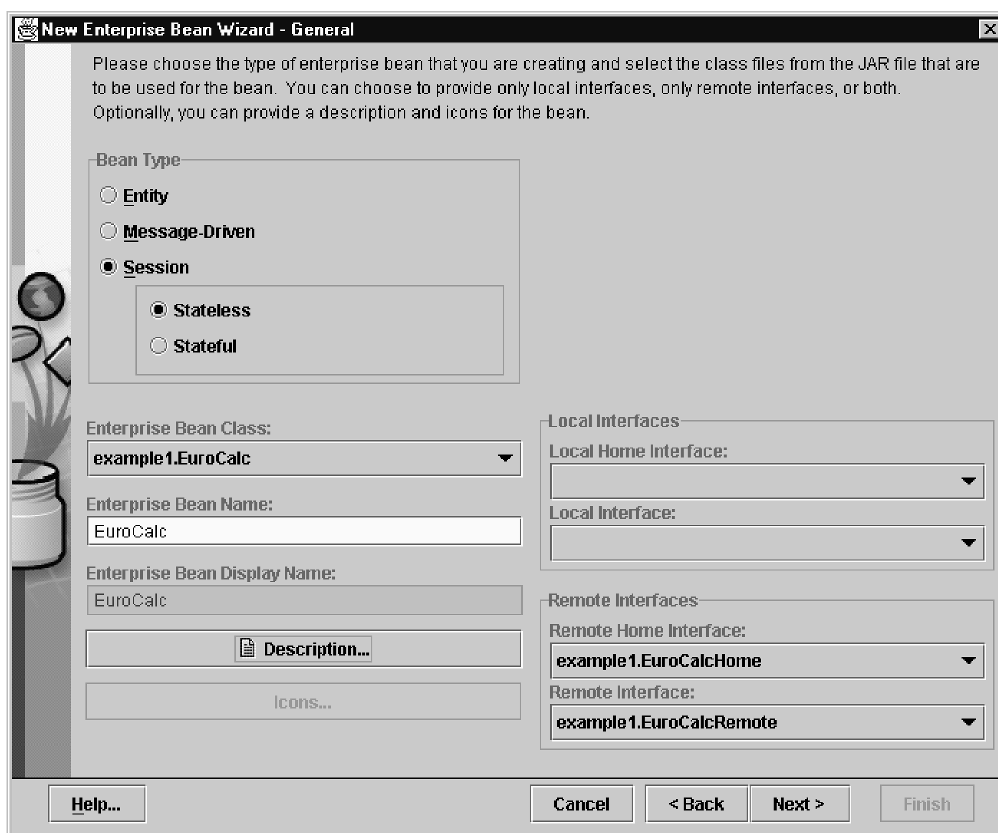

Abbildung 4.8: Konfiguration des EJB

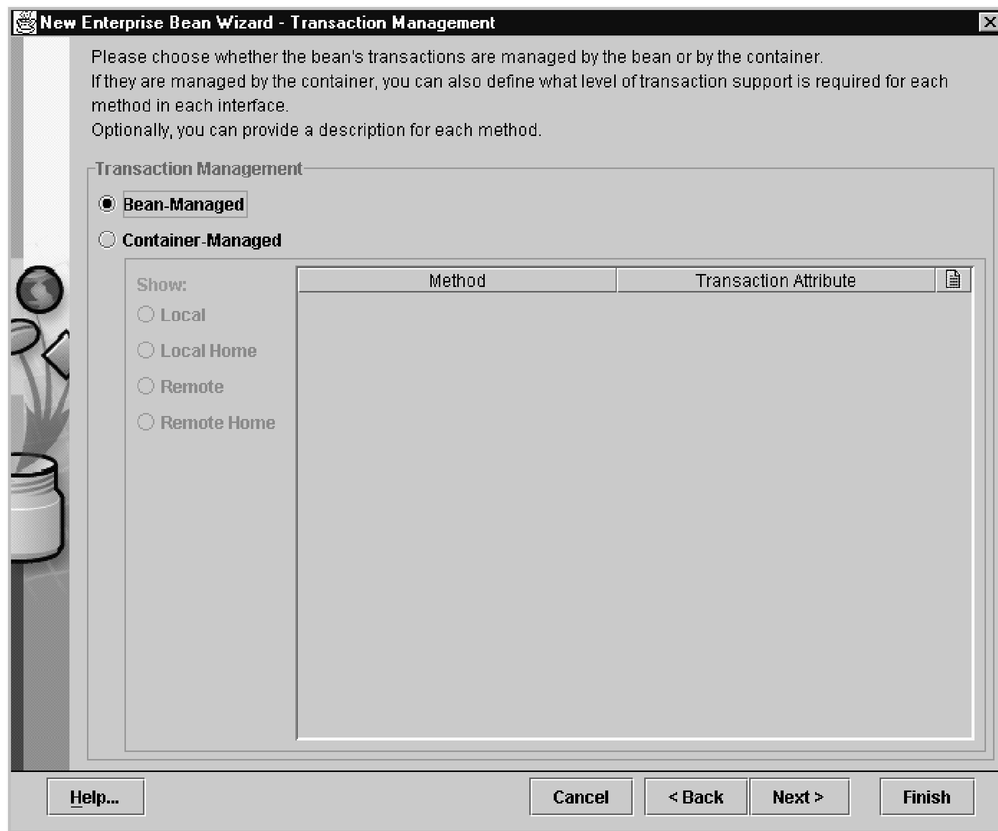

Abbildung 4.9: Festlegen der Transaktionseigenschaften

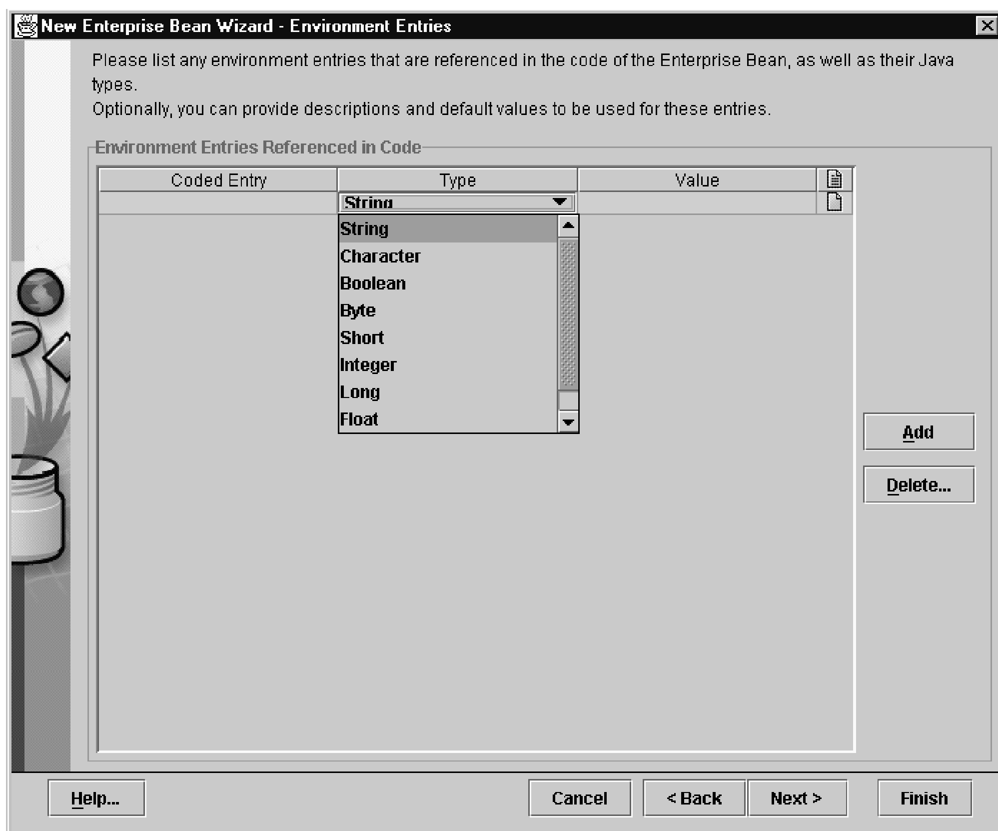

Abbildung 4.10: Festlegen von Umgebungsvariablen

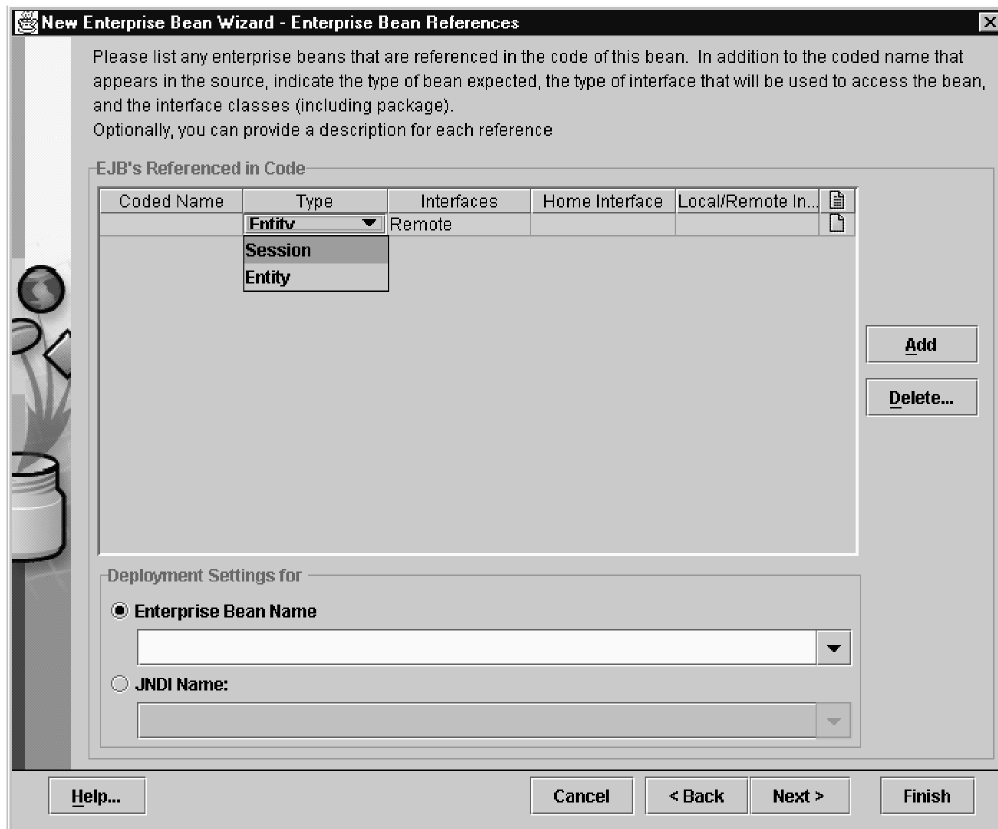

Abbildung 4.11: Konfiguration durch die EJB referenzierter EJB‘s

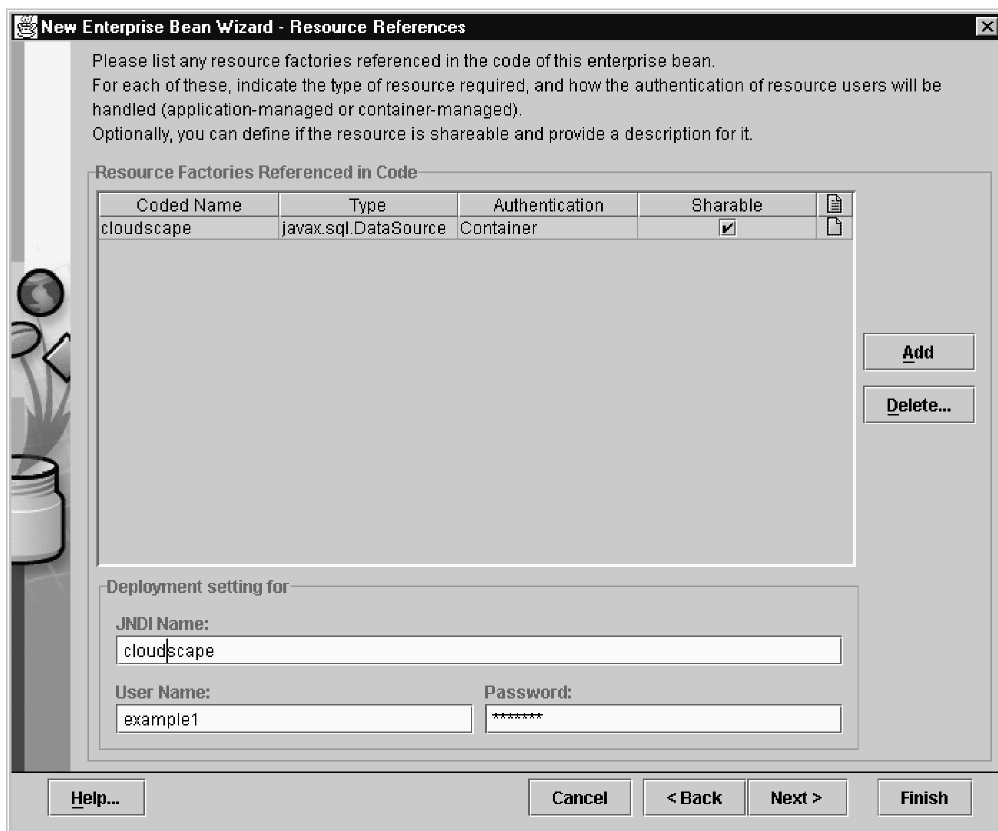

Abbildung 4.12: Festlegung genutzter Ressourcen wie z.B. eine Datenbank

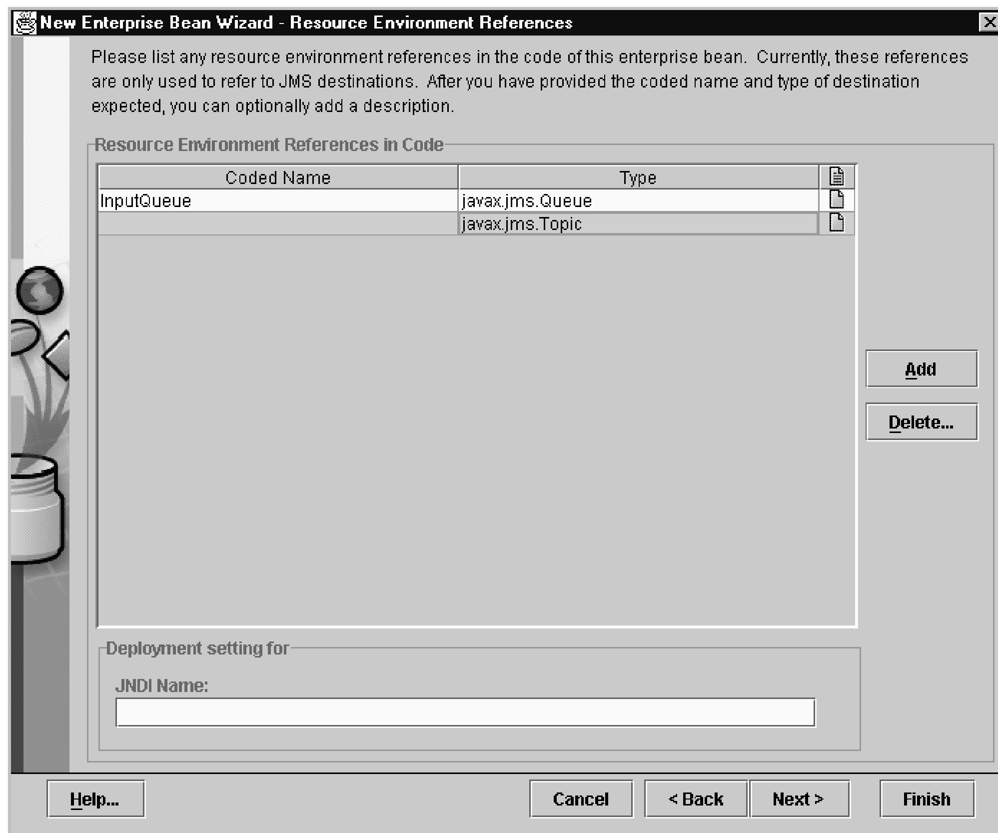

Abbildung 4.13: Festlegung von JMS-Referenzen

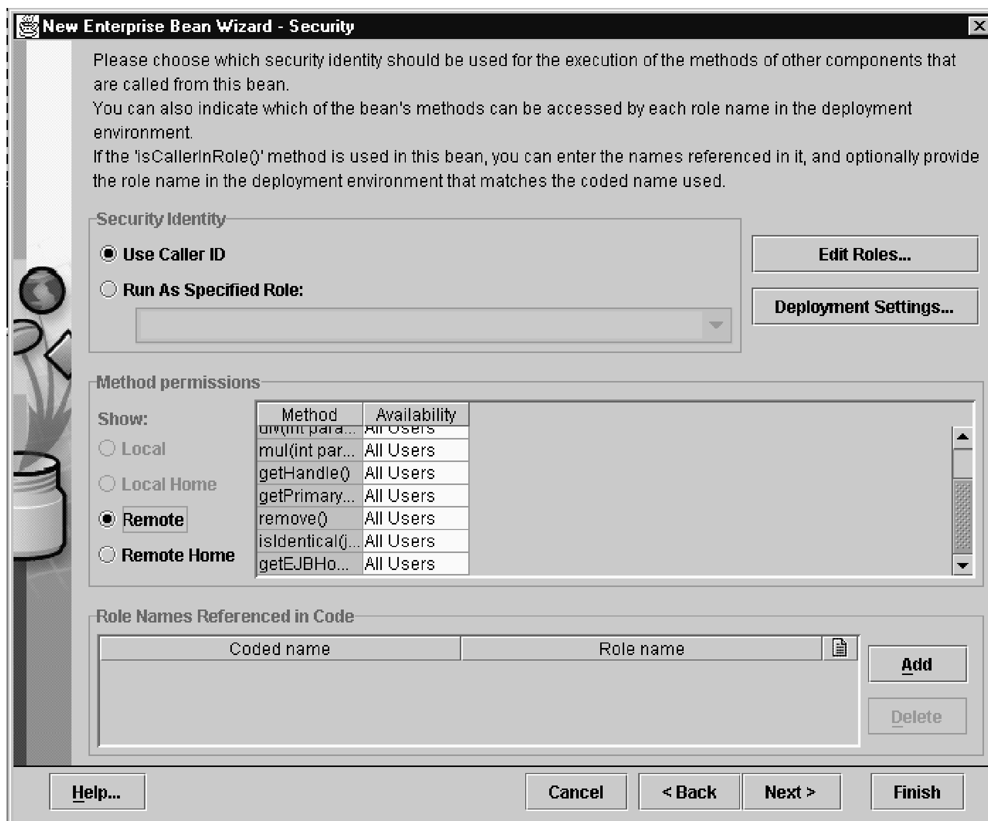

Abbildung 4.14: Festlegung von Security-Einstellungen

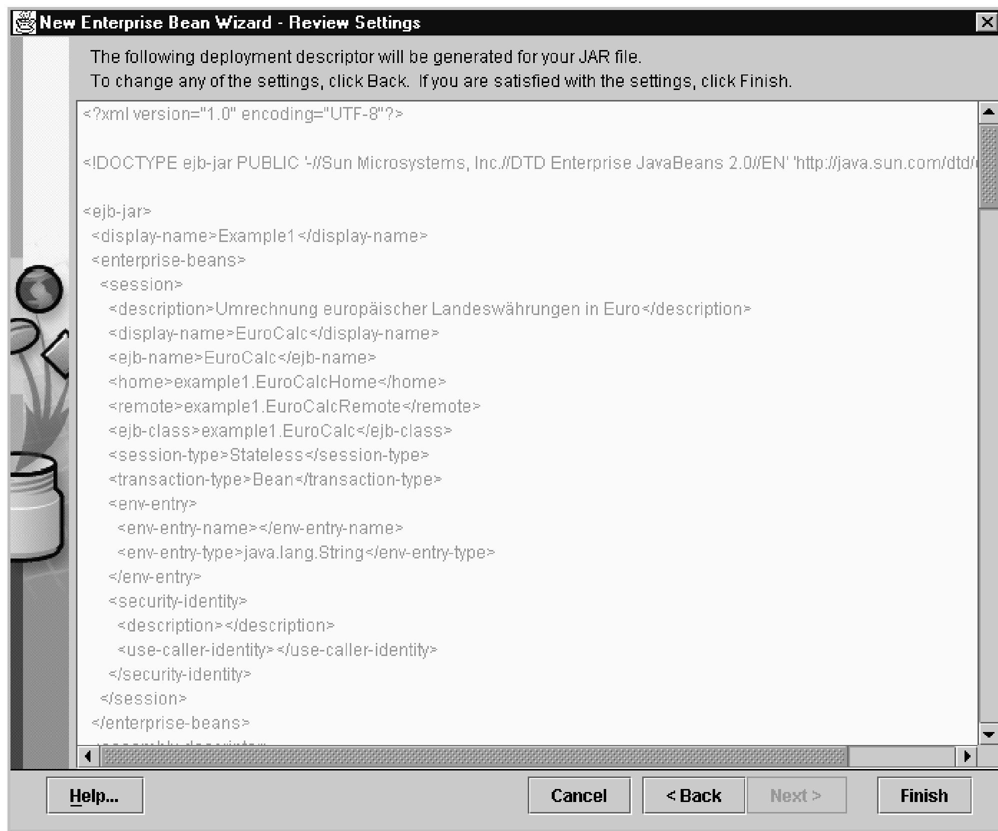

Abbildung 4.15: Ausgabe des erzeugten Deployment-Descriptors

Im folgenden wird der erzeugte XML-Deployment-Descriptor noch einmal aufgezeigt, enthalten sind dabei nur die Tags, welche zur Ausführung unbedingt benötigt werden:

```
<?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>
  <description>no description</description>
  <display-name>Ejb1</display-name>
  <enterprise-beans>
    <session>
            <display-name>Euro Calculator</display-name>
            <ejb-name>EuroCalc</ejb-name>
            <home>example1.EuroCalcHome</home>
            <remote>example1.EuroCalcRemote</remote>
            <ejb-class>example1.EuroCalc</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Der <*enterprise-beans*>-Tag kann neben den oben dargestellten <session>-Tags ebenfalls die Tags <*entity*> bzw. <*message-driven*> enthalten. Weitere Informationen innerhalb des Deployment-Desciptors betreffen Angaben zum Transaktionsverhalten, zur Sicherheit oder auch zu den verwendeten Ressourcen.

Nachdem diese Dialoge bearbeitet wurden, kann der eigentliche Vorgang der Installation der Applikation der Referenzumgebung innerhalb des J2EE-Serves ausgelöst werden.

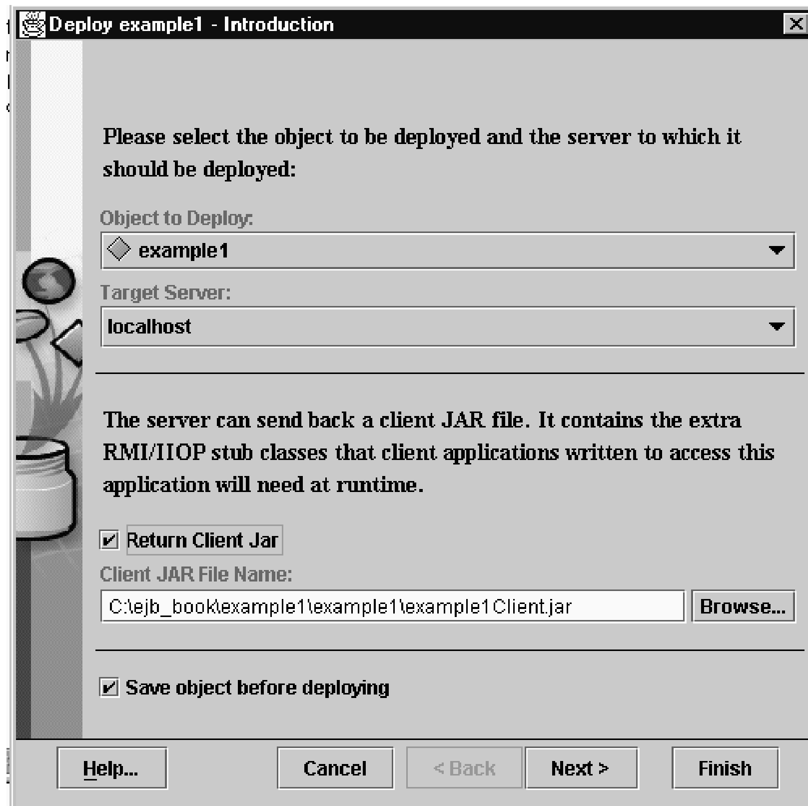

Abbildung 4.16: Deployment der Anwendung

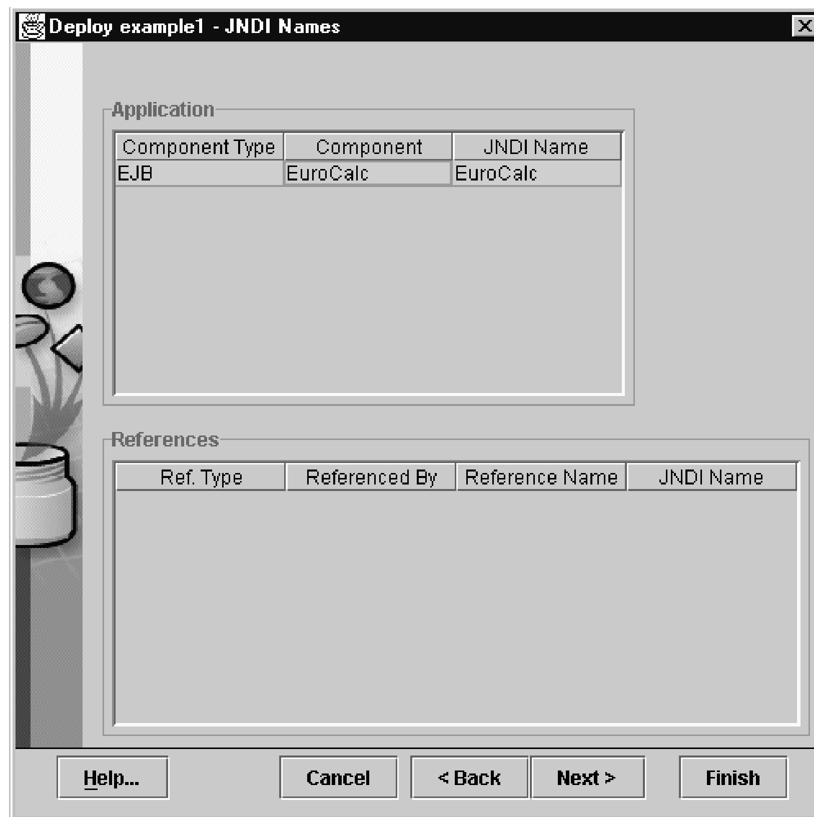

Abbildung 4.17: Festlegung der JNDI-Namen der Applikation und benötigter Referenzen

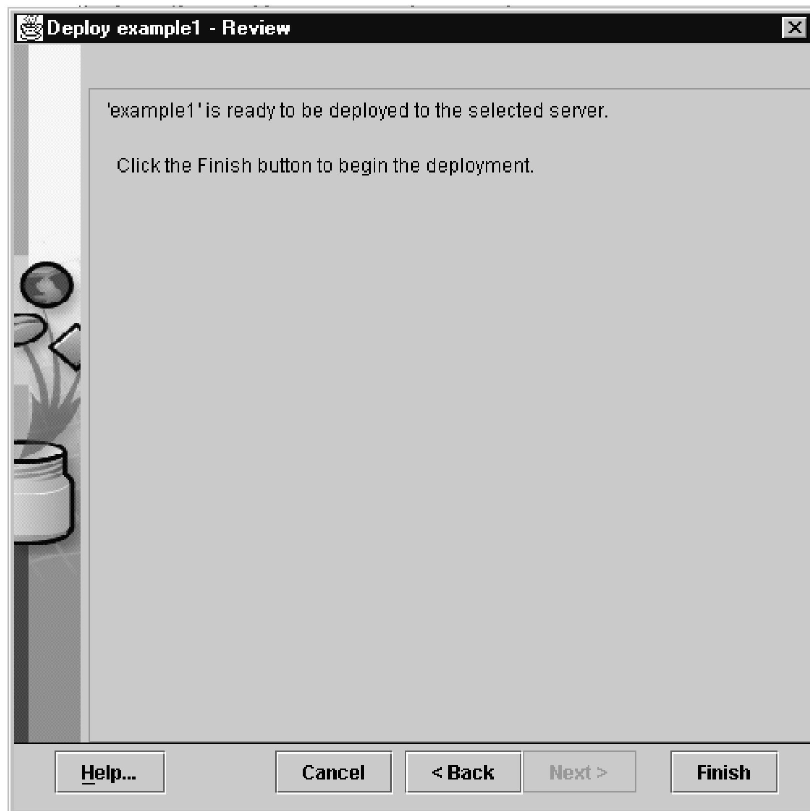

Abbildung 4.18: Bereit zur Installation der Applikation

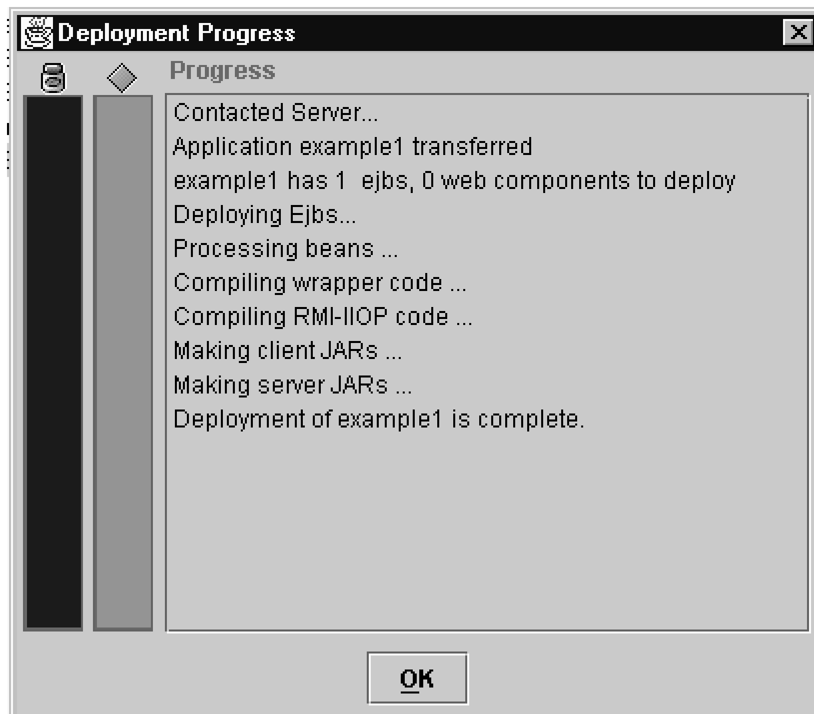

Abbildung 4.19: Dialog während und nach Beendigung der EJB-Installation

4.3.5 Implementierung der Client-Anwendung

Um die vorhergehend im Rahmen des Applikations-Servers installierte (deployed) Anwendung auch zur Ausführung zu bringen, wird ein entsprechender Client benötigt, der auf die Methoden der Anwendung und letztendlich der EJB-Komponente zugreifen kann.

```
public class EuroCalcClient {

  Object object = null;
  EuroCalcHome home = null;
  EuroCalcRemote remote = null;

  public EuroCalcClient() {
  }

  public void getBean()
  {
    try
      {
        InitialContext ctx = new InitialContext();
        //JNDI, nachsehen ob die Bean existiert
        object =  ctx.lookup("EuroCalc");
        System.out.println("object = ctx.lookup, wurde ausgefuehrt! \n");
        home  =  (EuroCalcHome)PortableRemoteObject.narrow(object,  EuroCal-
cHome.class);
        System.out.println("Zuweisung nach home, erledigt! \n");
        remote = home.create();
        System.out.println("Bean Reference: "+ remote +"\n");

      }
      catch(Exception ex){
      System.out.println("Fehler in getBean()! \n");
      ex.printStackTrace();
      }
  }

  public void umrechnung(){
     try{
      remote.dm_to_euro(50000);
      System.out.println("50000 DM sind " +
             remote.dm_to_euro(50000));
      remote.euro_to_dm(50000);
      System.out.println("50000 DM sind " +
             remote.euro_to_dm(50000));

     }catch (Exception exc){
       exc.printStackTrace();
    }
  }

  public static void main(String[] args) {
    EuroCalcClient rechnerClient = new EuroCalcClient();
    rechnerClient.getBean();
    rechnerClient.umrechnung();
 }
}
```

Im folgenden soll die Interaktion der wichtigsten Elemente einer EJB aus Sicht eines auf die Funktionen zugreifenden Clients noch einmal exemplarisch verdeutlicht werden.

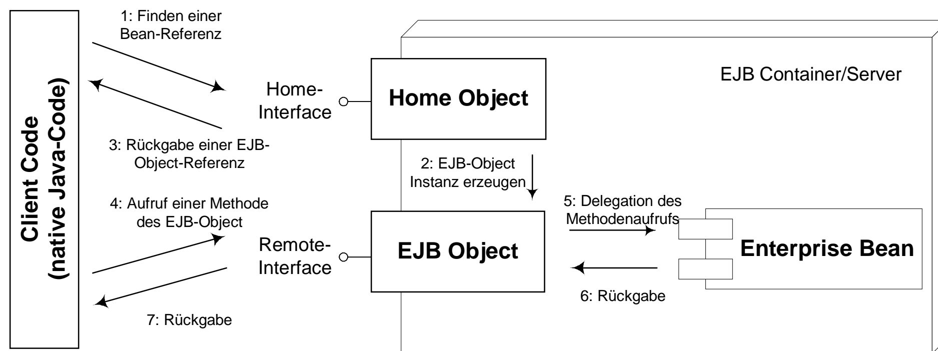

Abbildung 4.20: Interaktionsschritte beim Zugriff auf eine EJB-Komponente

1. Schritt: *Finden einer Referenz auf das Home Object der benötigten Bean.* Zuvor muss der aufrufende Client die im EJB-Container installierte Bean (Home Object) über JNDI finden (lookup „Home“). Nachfolgend werden die einzelnen Codefragmente für diesen Schritt vorgestellt.

```
// Erzeugen des Kontext unter Berücksichtigung des genutzten Applikation-
Servers/Container (ein innerhalb InitialContext übergebener Parameter
kann auf den Pfad des Applikations-Servers verweisen)

InitialContext ctx = new InitialContext(h);

//JNDI, auffinden der Bean über JNDI

object = ctx.lookup("EuroCalc");

//Typumwandlung

EuroCalcHome home = (EuroCalcHome) PortableRemoteObject.narrow(object,
EuroCalcHome.class);
```

2. Schritt: *EJB-Object Instanz erzeugen.* Es wird ein entsprechendes EJB-Objekt durch das Home-Object (Delegation der eigentlichen create-Methode vom Client) erzeugt oder auf ein vorhandenes referenziert.

```
// Folgende beim Client aufgerufene Methode des Home-Interfaces bewirkt
die Erzeugung eines EJB-Objektes (Funktionalität des Remote-Interfaces).

remote = home.create();
```

3. Schritt: *Rückgabe einer EJB-Object-Referenz.* Von diesem Zeitpunkt kann der Client (über die remote-Instanz) die Funktionalität der EJB nutzen.

4. Schritt: Aufruf einer im EJB-Objekt deklarieren Methode.

```
public void umrechnung(){

// die hier genutzte remote-Instanz enthält Methoden zur EURO/DM-
Umrechnung.

remote.dm_to_euro(50000);

remote.euro_to_dm(50000);
```

5. Schritt: *Delegation des Methodenaufrufs.* Über das EJB-Object wird der Aufruf an die eigentliche Implementierung der Enterprise JavaBean weitergeleitet.

Die Schritte 6 und 7 übergeben jeweils das Ergebnis der Methodenausführung an den Aufrufenden (EJB Objekt bzw. Client).

4.4 Transaktionssicherung in EJB-Umgebungen

Entsprechend [Wloka 1995] kann unter einer Transaktion eine Folge von Operationen verstanden werden, welche die Datenbank in ununterbrechbarer Weise von einem konsistenten Zustand in einen nicht notwendig verschiedenen konsistenten Zustand überführt. Mit den Transaktionen sind die folgenden Eigenschaften, die auch als ACID-Prinzip bezeichnet werden, „nativ“ verbunden.

- *Atomarität:* Eine Transaktion wird entweder vollständig oder gar nicht ausgeführt, undefinierbare Zwischenzustände werden ausgeschlossen. Für diese Aufgabe werden die konsistenten Zustände einer Datenbank im „Before Image“ gespeichert, auf dieses kann durch Rollback der Datenbank zurückgesetzt werden.
- *Consistency:* Vor und nach dem Ausführen einer Transaktion befindet sich die Datenbank in einem konsistenten Zustand.
- *Isolation:* Parallel ablaufende Transaktionen laufen unabhängig voneinander getrennt ab, eine gegenseitige Beeinflussung wird durch das Datenbanksystem ausgeschlossen.
- *Dauerhaftigkeit:* Die Ergebnisse einer erfolgreich ausgeführten Transaktion werden dauerhaft im sogenannte „After Image“ gespeichert und sind auch in Problemfällen wiederherstellbar.

Datenbank-Managementsysteme bieten von sich heraus eine Transaktionssicherung an. Im Umfeld verteilter Anwendungen reicht dieses nicht aus, da nur die Datenbank, aber nicht die Funktionen während der Verarbeitung berücksichtigt werden.

Im Falle der EJB-Technologie kann die Transaktionssicherung zum einem durch den Container, zum anderen durch die EJB-Komponenten selbst gemanagt werden. Die präferierte Vorgehensweise besteht in der erstgenannten Variante, deren Funktionalität im folgenden weiter betrachtet werden soll. Innerhalb des Deployment Descriptors können dafür die folgenden Transaktionsattribute verwendet werden, um festzulegen, in welcher Form die Transaktionssicherung bei einem Aufruf von Methoden des Home- bzw. Remote-Interfaces der entsprechenden EJB erfolgen soll. Auf diese Weise wird der Entwickler von umfangreichen Aufgabenstellungen der Transaktionssicherung, die er sonst selbsttätig erledigen müsste, befreit.

- *Required* – Diese Konfiguration führt zum Start einer Transaktion für den Fall, dass der Methodenaufruf einer EJB-Komponente nicht schon im Transaktionskontext steht.
- *Requires New* – Beim Methodenaufruf erfolgt in jedem Fall der Start einer Transaktion, eine schon laufende Transaktion wird vorübergehend beendet.
- *Supports* – Bei dieser Einstellung übernimmt der Container den bereits vorhandenen Zustand, d.h. es erfolgt eine Transaktionssicherung, wenn der Aufruf bereits unter Transaktionsverwaltung steht, andernfalls erfolgt keine Transaktionssicherung.
- *Not Supported* – Diese Einstellung führt dazu, dass die betroffene Bean an keiner Transaktion teilnehmen kann.
- *Mandatory* – Der aufrufende Client und die vom Aufruf betroffenen Methoden stehen unter einen gemeinsamen Transaktionsverwaltung.
- *Never* – Beim Aufruf von Methoden einer solchen EJB innerhalb einer laufenden Transaktion wird eine Ausnahme geworfen.

4.5 EJB-basierte Application Server

Theoretisch sollten EJB-Container auf jedweden Application Servern ablauffähig sein. Da aber die Schnittstelle zwischen Server und Containern innerhalb der EJB-Spezifikation nicht eindeutig festgelegt wurde, implementieren die Hersteller eigene Schnittstellen, die zumeist proprietäre Erweiterungen enthalten. Aus diesem Grund können EJB-Container häufig nicht ohne weiteres auf Serversystemen unterschiedlicher Hersteller zum Ablaufen gebracht werden. Dementsprechend wird das Produkt Application Server mit den darin ablaufenden Containern, welche die eigentliche Laufzeitumgebung für die Komponenten zur Verfügung stellen, zumeist gleichgesetzt. Allgemein bietet ein EJB-Applikation-Server die folgende Funktionalität an:

- Laufzeitumgebung für EJB- und JSP/Servlet-Container,
- Thread- und Prozessmanagement für Container, die parallel auf dem Server laufen,
- Verwaltung der Ressourcen des Betriebssystems,
- Lastverteilung, Clustering, und Ausfallsicherheit.

Im folgenden finden sich einige der derzeit über 40 am Markt existierenden Application-Server-Produkte. An dieser Stelle soll nicht der Versuch einer Bewertung unternommen werden, da diese nur eine extrem kurzfristige Gültigkeit aufweist. Eine Übersicht zum derzeitigen „State of the art“ bei Application Servern findet sich unter [Merkle 2000]

IBM	WebSphere
SilverStream	SilverStream Application Server
Netscape	Netscape Application Server
SUN	J2EE-Referenz-Implementierung
Sybase	Enterprise Application Server
Gemstone	Gemstone/J
BEA	WebLogic
Borland	Borland Application Server
ProSyst	EnterpriseBeans Server
IONA	Orbix E2A™ J2EE Technology Edition
JBoss	*Open Source Applikation Server*

Als Auswahlkriterien für einen Application Server können unter anderem die folgenden Kriterien herangezogen werden (die komplette Kriterienliste ist in der Anlage einzusehen):

- Handling des Kommunikations-Overheads zwischen EJB-Komponenten,
- eingehaltene Standards,
- Art und Weise der Deploymentunterstützung,
- Integration von Entwicklung und Deployment,
- Unterstützte Clustering- und Verfügbarkeitsfunktionalitäten,
- Administration und Konfiguration des Applikation Servers,
- Position des Anbieters am Markt inklusive angebotenem Support.

5 Beschreibung und Bewertung von Komponenten

5.1 Spezifikation von Komponenten

Die Spezifikation einer Komponente dient der vollständigen und widerspruchsfreien Beschreibung einer Softwarekomponente. Für den späteren Komponenten-Konsumenten ist primär das an der Schnittstelle der Komponente vorhandene Verhalten von Interesse, während den Komponenten-Entwickler auch interne Eigenschaften interessieren. Es stellt sich hier die Frage, inwieweit durch interne Kenngrößen der Komponente mittels eines empirischen Bewertungsmodells auf das externe Verhalten geschlussfolgert werden kann. Mittels der Spezifikation sollen potentielle Anwender der Komponente in die Lage versetzt werden, diese innerhalb eigener Programme verwenden und auf deren Dienste zugreifen zu können. Sehr allgemein können die folgenden Anforderungen an eine Beschreibung von Komponenten formuliert werden:

- Widerspruchsfreiheit und Eindeutigkeit,
- Verwendung von wenigen Notationen,
- Gütesiegel im Falle einer COTS-Komponente (siehe auch [SEI 2001]),
- Bezug auf die abgebildete Domain bei Fachkomponenten,
- Umsetzbar mit minimalen Aufwand,
- Konvergenz innerhalb betroffener Ingenieur-Disziplinen.

In der Literatur finden sich vielfältige und vor allem firmenspezifische Ansätze zur Beschreibung von Softwarekomponenten. Als Beispiele seinen [Schmietendorf/Dumke 2000, S.104] oder aber [Griffel 1998, S. 55] genannt. An dieser Stelle wollen wir uns bewusst an einem in [Turowski 1999] begründeten Ansatz orientieren, der in [Fachkomponenten 2002] einer Standardisierung[3] unterzogen wurde. Im Gegensatz zu diesem Ansatz, der im Kontext mit den sogenannten Fachkomponenten ausschließlich die Außensicht der Komponente berücksichtigt, soll im Rahmen dieses Preprints teilweise auch die Innensicht der Komponente von Interesse sein, um die bereits dargestellten Möglichkeiten einer empirischen Bewertung zu demonstrieren. In [Fachkomponenten 2002] werden die folgenden Beschreibungsebenen zur Spezifikation einer Fachkomponente vorgeschlagen:

- *Schnittstellenebene* – Auf dieser Ebene gilt es, die durch die Komponente öffentlich angebotenen Dienste hinsichtlich der verwendeten Signatur, übertragender Parameter inklusive der dabei verwendeten Datentypen und die potentiellen Fehlermeldungen zu spezifizieren. Darüber hinaus sind potentielle Dienste anzugeben, welche die Komponente zum Erbringen ihrer Leistung benötigt.

- *Verhaltensebene* – Die Verhaltensebene dient der Beschreibung von Vor- und Nachbedingungen konkreter Dienste der Komponente. Eine Vorbedingung kann dabei zum Beispiel die Belegung konkreter Parameter der Schnittstellendefinition sein. Darüber hinaus können Invarianten für interne Zustände der Komponente vergeben werden.

- *Abstimmungsebene* – Die Abstimmungsebene dient der Festlegung von Reihenfolgebeziehungen zwischen den Diensten verschiedener Komponenten als auch den Diensten innerhalb einer Komponente. Entsprechende Informationen zu dieser Ebene zeigen Szenarien der Interaktion mit anderen Komponenten.

---

[3] Memorandum des GI-AK 5.10.3 zur Vereinheitlichung der Spezifikation von Fachkomponenten

- *Qualitätsebene* – Ziel dieser Ebene ist es, die qualitativen Eigenschaften (z.B. Antwortzeit und Durchsatz) der Komponente und ihrer angebotenen Dienste zu erfassen. Häufig wird in diesem Zusammenhang auch von den nichtfunktionalen Eigenschaften gesprochen. Im Rahmen dieses Preprints wollen wir uns insbesondere auf diese Spezifikationsebene konzentrieren und entsprechende Ansätze bzw. Forschungsbedarfe aufzeigen.

- *Terminologieebene* – Diese Ebene regelt die semantischen Eigenschaften der Komponente in Bezug auf die verwendeten Begrifflichkeiten in Bezug auf die unterstützte Anwendungsdomain. Insbesondere für Fachkomponenten ist diese Festlegung wichtig, regelt diese doch, was z.B. unter einem Datenobjekt „Kunden“ aus fachlicher Sicht zu verstehen ist.

- *Aufgabenebene* – Ziel dieser Ebene ist die Darstellung der durch die Komponente realisierbaren Funktionalitäten aus fachlicher Sicht. Dabei können sowohl die Komposition, als auch Dekomposition von Aufgabenstellungen im Rahmen eines komplexen Gesamtsystems berücksichtigt werden.

- *Vermarktungsebene* – Für den Handel von Komponenten (COTS) besteht der Bedarf, allgemeine technische-, betriebswirtschaftliche- und organisatorische Informationen (z.B. Ansprechpartner für den Fehlerfall und Reaktionszeiten) vorzuhalten. Ziel dieser Ebene ist es, entsprechende Kataloge von Komponenten zu unterstützen.

Es sei darauf verwiesen, dass im praktischen Umfeld eine exakte Abgrenzung der Beschreibungsebenen häufig Schwierigkeiten bereitet bzw. aus Gründen des Aufwands nicht alle explizit verwendet werden können. Darüber hinaus unterstützen die innerhalb dieser Studie im Vordergrund stehenden Java-Komponenten (speziell Enterprise JavaBeans) diese Ebenen zum Teil nur implizit bzw. überlassen es dem Entwickler, diese Informationen der Komponente beizulegen.

5.2 Qualitätsbewertung von Komponenten

Im folgenden wollen wir uns vornehmlich auf die Spezifikation der nichtfunktionalen Eigenschaften einer Komponente konzentrieren, welche auf der Qualitätsebene festgehalten werden. Beispiele hierfür sind die Verfügbarkeit, die Wartbarkeit oder auch das Performanceverhalten eines durch die Komponente angebotenen Dienstes. Eine Spezifikation auf dieser Ebene sollte die Qualitätskriterien, verwendbare Messgrößen, Methoden zu deren Quantifizierung und ggf. Service Level (Qualitätsgrenzen zur Laufzeit), die durch die Komponenten einzuhalten sind, berücksichtigen. Darüber hinaus ist festzulegen, in welcher Form diese Informationen dem Benutzer einer Komponente zur Verfügung gestellt werden.

Die Erfassung der Qualitätseigenschaften einer Komponente kann entweder zum Zeitpunkt der Ausführung über die Aufnahme dynamischer Größen, wie z.B. des Durchsatzes und des Antwortzeitverhaltens konkreter Funktionen, oder aber anhand statischer Eigenschaften, z.B. Umfang, Anzahl der angebotenen Funktionen, Kopplungsbeziehungen, Testabdeckung, erfolgen, die auf der Grundlage eines entsprechenden empirischen Erfahrungshintergrunds auf Qualitätseigenschaften der Komponente schließen lassen.

Insbesondere die dynamischen Größen sind prinzipiell von den entsprechenden Randbedingungen (Hauptspeicher, Prozessor, DBMS etc.) abhängig, unter denen die Fachkomponente zur Ausführung gebracht wird. Diese Randbedingungen müssen konkretisiert werden, um zu objektiven Aussagen bezüglich der Qualitätseigenschaften einer Komponente zu gelangen. Dabei ist zu beachten, dass bei verschiedenen Komponenten eventuell verschiedene Randbedingungen von Relevanz sind. Beispielsweise können bestimmte Fachkomponenten

dingungen von Relevanz sind. Beispielsweise können bestimmte Fachkomponenten sehr stark von der Leistungsfähigkeit der Datenbank abhängig sein, andere wiederum können stark abhängig von der Netzwerkkapazität oder von der Prozessorleistung sein. Daher erscheint es notwendig, die zu spezifizierenden Randbedingungen offen zu lassen.

Zur Bestimmung der Qualitätseigenschaften bei konkreter Definition von Randbedingungen sind weiterhin zwei verschiedene Möglichkeiten denkbar:

1. Definition einer Referenzumgebung: Bei dieser Variante wird eine Referenzumgebung definiert, auf der prinzipiell die Qualitätseigenschaften bestimmt werden. Problem: Dieses Vorgehen ist aufwendig. Ferner können die Referenzumgebungen schnell veralten und somit die Relevanz (in Bezug auf den speziellen Einsatz der Komponente) der gewonnenen Messwerte in Frage stellen.

2. Sammlung von Messwerten bei konkreten Komponenten-Installationen: Bei dieser Variante wird nicht eine ausschließliche und explizite Referenzumgebung definiert. Statt dessen werden viele Messungen der Qualitätseigenschaften unter verschiedenen Randbedingungen vorgenommen. Alle diese Messfälle werden in der Spezifikation gesammelt. Problematisch hierbei ist die Form der Datenerhebung.

Durch die Berücksichtigung der vorher dargestellten Randbedingungen erschien eine Festlegung konkret festzulegender Qualitätskriterien innerhalb des Memorandums nicht zielführend, da so die Universalität der Spezifikation stark eingeschränkt wird. Aus diesem Grund soll ein grober Rahmen zur Vorgehensweise der Qualitätsspezifikation festgelegt werden, der durch den jeweiligen Komponentenhersteller entsprechend konkretisiert wird.

5.2.1 Schritte zur Qualitätsspezifikation

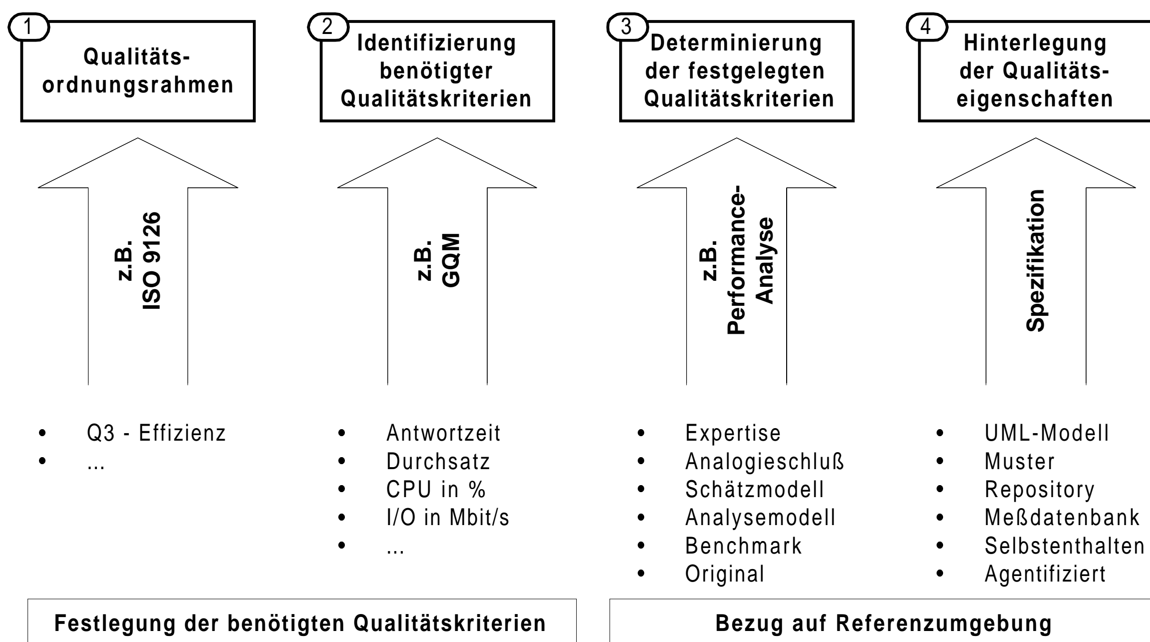

Abbildung 5.1: Schritte zur Qualitätsspezifikation [Schmietendorf/Dumke 2001]

Abbildung 5.1 zeigt die notwendigen Schritte zur Qualitätsspezifikation einer Fachkomponente, wobei die Schritte 1 und 2 von einer im vorhergehenden Abschnitt aufgezeigten Referenz-

umgebung weitgehend unabhängig sind, die Schritte 3 und 4 nur im Kontext mit einer entsprechenden Referenzumgebung ausgeführt werden können.

Um der Allgemeingültigkeit des hier vorgestellten Ansatzes gerecht zu werden, sollen diese Schritte in den folgenden Abschnitten weiter erläutert werden, wobei die Schritte 3 und 4 beispielhaft für die Qualitätsklasse der Effizienz in den folgenden Abschnitten erläutert werden.

5.2.2 Qualitätsklassen als Ordnungsrahmen (Schritt 1)

Der Begriff der Qualität eines Softwareproduktes im allgemein als auch im besonderen in Bezug auf Komponenten führt zu vielfältigen Interpretationsmöglichkeiten. Dementsprechend ist es notwendig, ein konkretes Qualitätsmodell festzulegen, um den Qualitätsbegriff zu operationalisieren. Für diese Aufgabenstellung haben sich sogenannte FCM[4]-Modelle (siehe dazu auch Darstellungen in [Balzert 1998]) herausgebildet, welche ausgehend von Qualitätsfaktoren entsprechende Qualitätskriterien festlegen und für deren Quantifizierung entsprechende Metriken vorschlagen.

Mit der ISO 9126 findet sich ein Standard für ein Qualitätsmodell entsprechend dem FCM-Ansatz. Im folgenden soll dieser als Orientierungshilfe herangezogen werden, um Qualitätsklassen zur Bewertung von Komponenten zu bilden. Mit der Einführung der folgenden Qualitätsklassen (siehe auch [Schmietendorf/Scholz 2000]) im Rahmen der Spezifikation von Softwarekomponenten ergibt sich die Möglichkeit einer granularen Berücksichtigung von Qualitätseigenschaften konkreter Komponenten.

- *Übertragbarkeit Q1:* Anpassbarkeit, Installierbarkeit;
- *Anwendbarkeit Q2:* Erlernbarkeit, Beherrschbarkeit, Verständlichkeit;
- *Effizienz Q3:* Raumbezogen, Zeitbezogen, Ressourcenbezogen;
- *Funktionalität Q4:* Angemessenheit, Interoperabilität, Genauigkeit;
- *Zuverlässigkeit Q5:* Fehlertoleranz, Fehlerhäufigkeit, Fehlerverfolgbarkeit;
- *Wartbarkeit Q6:* Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit.

Eine konkrete Komponente kann so z.B. die Qualitätseigenschaften Q1 und Q3 aufweisen, d.h. innerhalb der Spezifikation befinden sich Aussagen zu genau diesen Qualitätseigenschaften, nicht aber zu allen weiteren. Wie letztendlich die konkrete Ausprägung der Spezifikation in Bezug auf genau eine Qualitätsklasse erfolgt, soll in einem weiteren Schritt mit Hilfe der GQM-Methode geklärt werden.

5.2.3 Komponentenspezifische Qualitätsmodelle (Schritt 2)

Für die eigentliche Identifizierung der für eine konkrete Komponente zu spezifizierenden Qualitätseigenschaften kann z.B. das GQM-Pardigma (Goal Question Metric) verwendet werden [Solingen/Berghout 1999]. Dieses bietet eine methodische Vorgehensweise zur Erstellung eines entwicklungsspezifischen Qualitätsmodells und kann so auch für die Komponententechnologie herangezogen werden. Dafür führt die GQM-Methode, ausgehend von zu formulierenden Qualitätszielen (im speziellen Fall der Qualität können diese der ISO 9126 entnommen werden) und den Fragen, wie diese erreicht werden können, zu den für die Quantifizierung bzw. Beantwortung der Fragen notwendigen Messgrößen. Für die Beantwortung der identifi-

[4] factor criteria metrics model

zierten Fragestellungen werden typischerweise die folgenden Erfolgskriterien in Anlehnung an [Dumke 2001] herangezogen:

- *Sichtweise:* Komponentenentwickler, Komponentenanwender, Auftraggeber,...
- *Anwendungsbereich:* spezielles Projekt bzw. ausgewählte Produktklasse,
- *Zweck:* Analyse, Verständnis,...
- *Kontext:* z.B. im Rahmen eines ausgewählten Entwicklungsteams.

Darüber hinaus sind im Rahmen des GQM-Ansatzes Aufgabenstellungen der effizienten Messwerterfassung, der Ergebnisinterpretation und der Validation durchzuführen (siehe dazu z.B. [Dumke 2001, S. 204]). Auf diese Themenstellung soll im Rahmen dieses Beitrags allerdings nicht weiter eingegangen werden.

Für speziell die Qualitätsklasse Q3 (Effizienz) ergibt sich so die folgende Ausprägung:

Goal (Ziel)

Das determinierte Performanceverhalten der durch eine Komponente angebotenen Funktionen ist für deren erfolgreichen Einsatz von entscheidender Bedeutung. (z.B. bei Komponenten im Bereich der Telekommunikation, Banken, Militär).

Question (Frage)

Welche Kenngrößen werden zur Erfassung des zeit- und raumbezogenen Performanceverhaltens benötigt?

Metric (Messgröße)

Für die Erfassung des nach außen sichtbaren Performanceverhaltens werden die Größen Antwortzeit und Durchsatz einer konkreten Funktion der Komponente herangezogen.
Darüber hinaus besteht die Notwendigkeit, den für die Erbringung dieser Größen benötigten Ressourcenverbrauch (CPU, I/O, Softwareservices) festzulegen, das entsprechende Lastprofil (Auftragsarten sind im zeitlichen Verlauf zu berücksichtigen), die verwendete Hard- und Softwarearchitektur und die Leistungseigenschaften der betrachteten Gesamtarchitektur (Netzwerke, Rechner, Services,...) zu erfassen, was der bereits angesprochenen Referenzumgebung entspricht.

Für speziell die Qualitätsklasse Q3 wird die Vorgehensweise zur Ermittlung der vorgenannten Messgrößen unter Abschnitt 5.5 dargestellt.

5.2.4 Determinieren der festgelegten Qualitätskriterien (Schritt 3)

Die Quantifizierung der Qualitätskriterien einer Komponente kann nur unter Anwendung eines konkreten Verfahrens bzw. Methode erfolgen, wobei aus Gründen der Effektivität in jedem Fall eine werkzeuggestützte Vorgehensweise anzustreben ist. Im folgenden seien allgemeine Beispiele solcher Methoden in Bezug auf die verwendeten Qualitätsklassen genannt:

- *Übertragbarkeit Q1:* Im speziellen Fall einer Enterprise JavaBean könnten dafür z.B. Werkzeuge (Parser) zur Analyse der Konformität zur EJB-Spezifikation eingesetzt werden.

- *Anwendbarkeit Q2:* Evaluierung mittels z.B. eines Fragenkatalogs zur Handhabbarkeit der Komponente. Dieser wird potentiellen Benutzern zur Verfügung gestellt.

- *Effizienz Q3:* Verwendung von Methoden der Performanceschätzung, Performancemodelle, Performancebenchmark und entsprechender Profiler-Werkzeuge.
- *Funktionalität Q4:* Durchführung entsprechender Funktionstests im Rahmen der Komponentenentwicklung.
- *Zuverlässigkeit Q5:* Gewinnung von Erfahrungswerten anhand eingesetzter Komponenten innerhalb des Wirkbetriebs.
- *Wartbarkeit Q6:* Software-Messung zur Gewinnung statischer Quellcodemetriken mittels z.B. des Logiscope- oder des McCabe-Tools.

In Anlehnung an den von der IEEE vorgegebenen Standard können die grundlegenden Messstrategien der Evaluierung (z.B. Fragebogen), Merkmalsschätzung (z.B. formelbasierte Darstellung potentieller Zusammenhänge), modellbezogene Messung sowie die direkte Messung zum Einsatz kommen. Die Quantifizierung der Qualitätskriterien einer Komponente kann nur unter Anwendung eines konkreten Verfahrens bzw. Methode erfolgen. Für den speziellen Fall der Effizienz (Qualitätsklasse Q3) kann die allgemeine Vorgehensweise unter Abschnitt 5.5 nachvollzogen werden.

5.2.5 Spezifikation der Qualitätseigenschaften (Schritt 4)

Aufgrund der vielfältig einsetzbaren Notationen zur Hinterlegung der qualitativen Eigenschaften einer Komponente ist es nicht sinnvoll, eine primäre Notation vorzugeschlagen. Dementsprechend können nur grobe Vorschläge unterbreitet werden, wie die gewonnenen Qualitätsmerkmale als Spezifikation der Komponente verwendet werden können.

- Beschreibung im Rahmen der Komponentenmodellierung,
- Beschreibung als Entwurfsmuster (vgl. EJB-Pattern),
- Beschreibung im Rahmen eines Komponenten-Repository,
- Hinterlegung der Qualitätseigenschaften in Messdatenbanken,
- Selbstenthaltene Beschreibung von Komponenteneigenschaften
  - z.B. innerhalb des Deployment Descriptors,
  - als flache Datei im Rahmen der Archiv-Datei,
- Agentifizierte Komponenten
  - Möglichkeiten der dynamischen Reaktion auf z.B. Leistungsengpässe,
  - Auslesen qualitativer Eigenschaften aus zentralen Repositories.

5.3 Ausgewählte Beispiele UML-basierter Spezifikationen

5.3.1 Möglichkeiten einer UML-basierten Spezifikation

Sollen Qualitätseigenschaften einer Komponente im Rahmen der UML-Notation festgehalten werden, bieten sich dafür insbesondere die Interaktions-, Zustands-, Paket- und Verteilungsdiagramme an. Um diese Diagramme für die Qualitätsebene zu verwenden, sind die nativen UML-Erweiterungsmöglichkeiten (Stereotypes, Annotationen, Constraints, Tagged Values) heranzuziehen. Entsprechende Vorschläge zur Verwendung der UML-Notation im Kontext mit dem Effizienzverhalten (Qualitätsklasse Q3) von Software-Artefakten finden sich unter [Schmietendorf/Dimitrov 2001].

Im folgenden sollen die Möglichkeiten dieser Notation in Bezug auf die Spezifikation von Komponenten skizziert werden:

- *Interaktionsdiagramm*: erlaubt die Darstellung von Interaktionen zwischen Komponenten-Instanzen. In deren Rahmen kann zum einem das Auftragsprofil, das zeitliche Verhalten von Komponenten-Methoden und der Ressourcenbedarf von Komponenten-Instanzen mittels Constraints, Tagged Values und Annotationen festgehalten werden.

- *Zustandsdiagramm:* erlaubt z.B. die Erfassung des Lebenszyklus einer Komponente, wofür Zustände (Knoten) und Zustandsübergänge (Kanten) mit entsprechenden Übergangs-Bedingungen verwendet werden. Dementsprechend bietet sich mit diesem Modell eine direkte Beziehung zur Markov'schen Modellierung, welche für die modellbasierte Performanceanalyse einer Komponente herangezogen werden kann.

- *Paketdiagramm:* erlaubt die atomare Darstellung der Komponente selbst. Durch sowohl Annotationen als auch Constraints lassen sich das abstrakte Ressourcenverhalten sowie statische Eigenschaften spezifizieren, wie z.B. Granularität und Kopplungsgrade einer Komponente.

- *Verteilungsdiagramm:* bietet den direkten Bezug zu Hard- und Softwareressourcen der Laufzeitumgebung, sofern explizite Angaben zu den benötigten Ressourcen mittels Annotationen und Sterotypen möglich sind.

5.3.2 Beispiele UML-basierten Spezifikation

Abbildung 5.2 zeigt die Spezifikation von zeitlichen Vorgaben (Ausführungszeiten und Latenzzeiten im Rahmen eines UML-Interaktionsdiagramms.

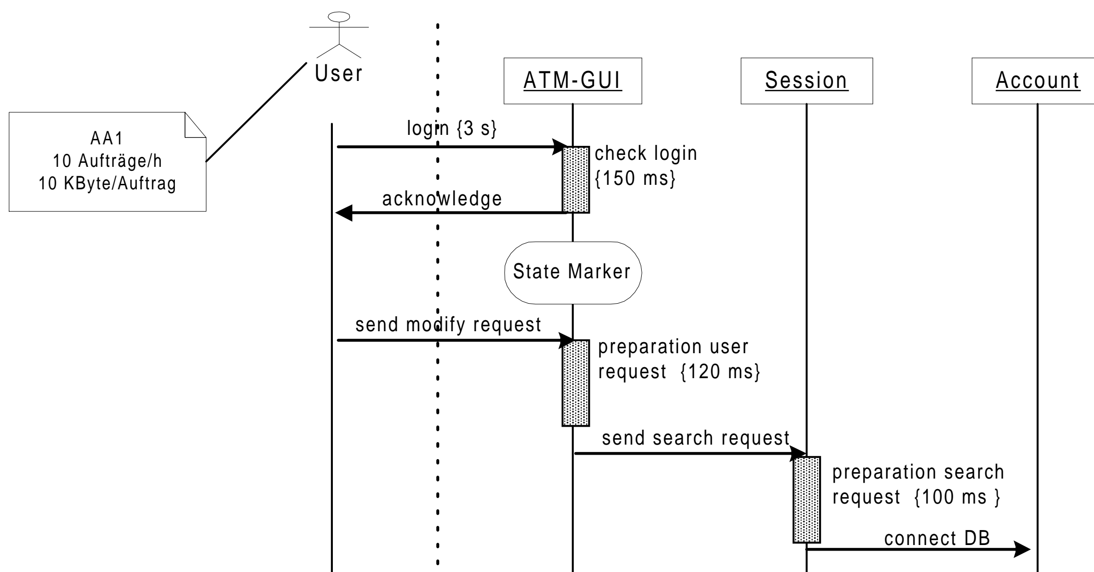

Abbildung 5.2: Spezifikation im Rahmen von Sequenzdiagrammen

Abbildung 5.3 zeigt die vorgeschlagene Vorgehensweise unter Zuhilfenahme restriktiver Stereotypen und Annotationen im Rahmen von Verteilungsdiagrammen:

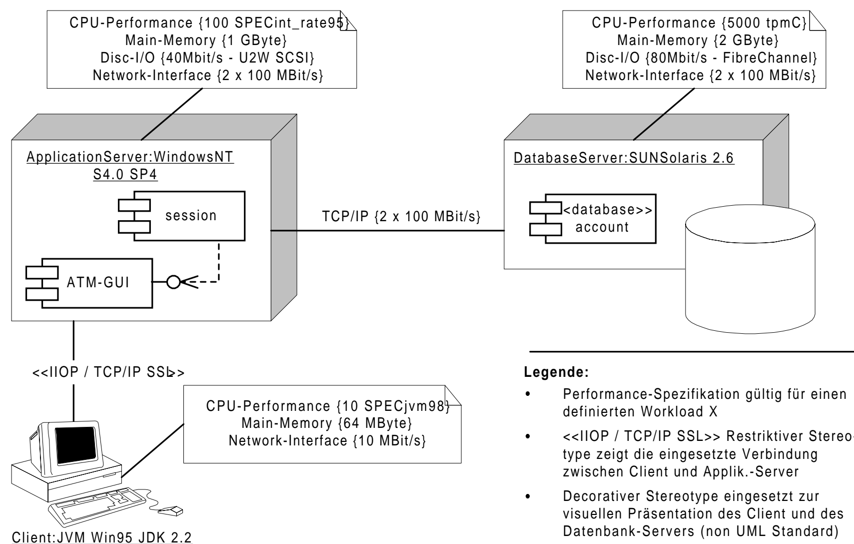

Abbildung 5.3: erweiterte UML-Verteilungsdiagramme

Die Object Constraint Language OCL bietet die Möglichkeit, Rahmenbedingungen bzw. Voraussetzungen, die eine Komponente zur Erbringung ihrer geforderten funktionalen und qualitativen Eigenschaften benötigt, festzulegen. Im folgenden sei beispielhaft der Ressourcenbedarf einer konkreten Methode an I/O-Operationen mittels der OCL-Notation dargestellt:

```
context Komponente inv I/O-Bedarf
     self.show() < 100 NW-I/O
```

Die OCL bietet darüber hinaus die Möglichkeiten, Performanceanforderungen im Sinne von Antwortzeit- und Durchsatz-Schranken zu formulieren. Sinnvoll ist es, diese aus dem Modell direkt in die betroffene Komponenten zu überführen, wobei im speziellen Fall der EJB-Komponenten eine Abbildung im Rahmen des sogenannten XML-Deployment-Deskriptor vorgenommen werden kann. Diese Informationen können so insbesondere zur Festlegung von SLA‘s (Service Level Agreement) bzw. SLO‘s (Service Level Objectives) im Rahmen des Wirkbetriebs herangezogen werden können. Inwieweit die Vereinbarung komponentenbezogener SLA’s sinnvoll ist, hängt zum einem von der Granularität der betrachteten Komponente, dem mit einem unperformanten Funktionsverhalten einhergehenden Risiko und den ggf. erforderlichen dynamischen Reaktionen (z.B. Loadbalancing) auf Performanceengpässe ab.

5.3.3 Messwertbezogene Spezifikationsansätze

Eine weitere Möglichkeit zur Erfassung qualitativer Eigenschaften einer Komponente besteht in der allgemeinen Verwendung von Softwaremetriken (über Performancemetriken hinausgehend), die qualitative Eigenschaften einer Komponente widerspiegeln. Dabei handelt es sich um die bereits angesprochenen statischen Eigenschaften der Komponente. Die Zielstellung besteht darin, Komponenten anhand charakteristischer Messgrößen hinsichtlich ihres qualitativen Verhaltens zu bewerten. Auf der Basis empirisch gewonnener Schwellwerte (Metriken) für „qualitativ hochwertige“ Softwarekomponenten lassen sich von Seiten des Komponentenentwicklers Komponenteneigenschaften durch den Einsatz von Programmierrichtlinien verifizieren bzw. im Rahmen der Beschreibung für den späteren Komponentenkonsumenten festhalten. Die dabei verwendeten Messgrößen können ein unterschiedliches Skalenniveau (nominal-, ordinial-, intervall-, ratio-skaliert) aufweisen. Ein höheres Skalenniveau impliziert sowohl einen größeren Informationsgehalt dieser Metriken als auch die Möglichkeit, höherwertige statistische Operationen über diesen anzuwenden.

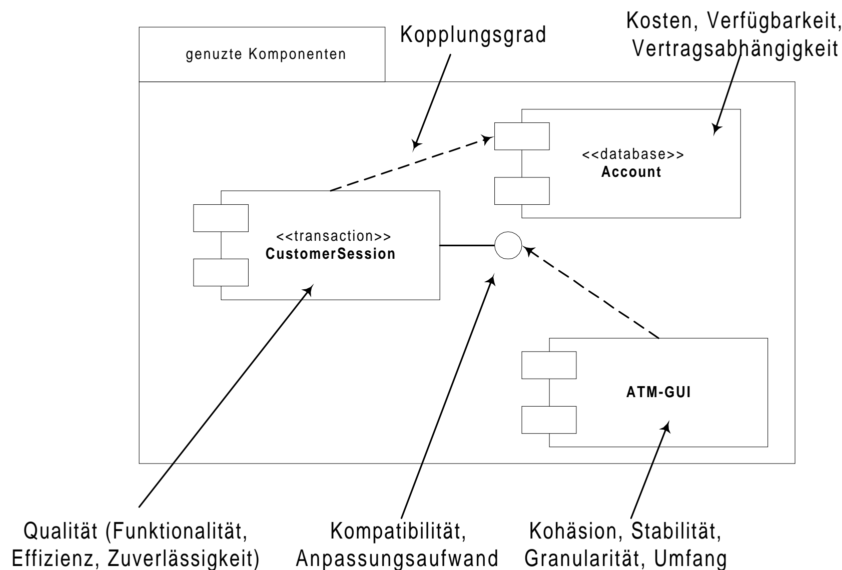

Abbildung 5.4: Messansätze im Rahmen einer Komponentenarchitektur

Abbildung 5.4 zeigt potentielle Messansätze im Umfeld von Softwarekomponenten. Im Abschnitt 5.4 und 5.5 soll auf ausgewählte Beispiele solcher Messansätze noch vertiefend eingegangen werden. Im Rahmen einer UML-basierten Spezifikation der betrachteten Komponente sollten metrikenbasierte Bewertungsansätze in Form von Annotationen bzw. im formaleren Sinne durch OCL-konforme Ausdrücke hinterlegt werden. Derartige Beschreibungselemente dienen in der derzeitigen Form der UML ausschließlich der Dokumentation potentieller Anforderungen an das Verhalten der Komponente. Eine Verifikation dieser Daten, im Sinne der Übereinstimmung des komponentenorientierten Entwurfs mit den tatsächlichen Bedingungen der späteren Anwendung, auf Basis einer z.B. simulativ durchgeführten Analyse, ist derzeit unmittelbar nicht möglich. Dies erfordert die Überführung in adäquate Analysemodelle.

5.3.4 Abschließende Anmerkungen

Die vorhergehend dargestellten Spezifikationsansätze auf Basis der UML konzentrierten sich primär auf die Abbildung qualitativer Aspekte im Bereich der Performanceeigenschaften einer Komponente. Allgemein kann festgestellt werden, dass derzeit kein standardisierter Ansatz zur Modellierung von Komponenten existiert. Dieser Umstand hat aus Sicht der Autoren maßgeblich zur noch unzureichend durchgeführten Wiederverwendung im Umfeld der industriellen Softwareentwicklung beigetragen. Die Verwendung der UML-Notation im Umfeld eines komponentenbasierten Softwareengineerings kann für folgende Aufgabenstellungen zum Einsatz kommen:

- Modellierung des Diskursbereiches (Daten und Funktionen) der späteren Komponente im Rahmen einer fachlich orientierten Analyse, wobei die Aufgabenstellung der Komponentenidentifizierung und ggf. Kopplung zu anderen Komponenten im Vordergrund steht.
- Modellierung von Design- und Implementierungsaspekten der in Entwicklung befindlichen Komponente.
- Dokumentation zur Unterstützung einer effizienten Wiederverwendung der Komponente im Rahmen eigener Anwendungen.

Die häufig im Umfeld der UML-basierten Modellierung erhobene Anforderung, aus dem Komponenten-Design automatisiert Quellcode abzuleiten, ist aus Sicht der Autoren zwar sehr löblich, aber derzeit doch recht fragwürdig. Im Vordergrund einer Modellierung steht die

Abstraktion vom Original. Es gilt, Fragen der Funktionalität und Qualität des späteren Systems mit geringerem Aufwand klären zu können, als dies am Original möglich wäre. Darüber hinaus wird, wie in jeder Ingenieurdisziplin üblich, eine vereinfachende Darstellung komplexer Zusammenhänge mittels verschiedener Sichten und Strukturierungsebenen auf das spätere System benötigt. Nur so können komplexe Systeme arbeitsteilig in Teams entwickelt werden.

5.4 Empirische Analysen verfügbarer Bean-Komponenten

5.4.1 Hintergründe zur Metriken-Anwendung

Die Analyse statischer Komponenteneigenschaften auf der Basis der Vermessung von ca. 50 Java-Komponenten (sowohl JavaBeans als auch EJB's) aus dem industriellen und akademischen Umfeld führte zu einem ersten Entwurf einer entsprechenden Programmierrichtlinie, die sowohl durch Komponentenentwickler als auch Komponentenanwender verwendet werden kann (siehe dazu [Lezius 2001]). Der Aufbau dieser Richtlinie enthält eine Darstellung des Qualitätsmerkmals, das entsprechende Kriterium, einsetzbare Metriken und Schwellwerte zur Bewertung einer konkreten Größe. Als Beispiele seien in Bezug auf die Wiederverwendbarkeit folgende Metriken genannt:

- *Komponentengröße (z.B. Klassen, eLoC),*
- *Kopplungsmaße (z.B. Fan In/Out),*
- *Abstraktion der Komponente (z.B. Vererbungstiefe),*
- *Dokumentation der Schnittstellen (z.B. Verhältnis eLoC zu Kommentaren),*
- *Komplexität der Komponenten (hier McCabe konkreter Methoden),*
- *Granularität (z.B. Anzahl zur Verfügung gestellter fachlicher Funktionen).*

Ein anderer Input für die Einschätzung der Qualität kommerziell angebotener Komponenten (COTS - commercial of the shelf) besteht in der Verwendung indirekter Faktoren, die sich auf das Prozess- und Ressourcenniveau der jeweiligen Komponentenhersteller beziehen. Als Beispiel für ein dafür verwendbares Prozessmodell sei das CMM (Capability Maturity Model) genannt. Inwiefern dieses auch die Aspekte einer komponentenorientierten Softwareentwicklung berücksichtigt, kann derzeit nicht abschließend geklärt werden. Erste Anhaltspunkt dazu finden sich im CMMI unter [Ahern 2001]. Im folgenden sollen die EJB-Komponenten auf der Basis zweier Bewertungsansätzen analysiert werden. Zum einen werden die bereits unter Abschnitt 5.1 eingeführten Spezifikationsebenen aus [Fachkomponenten 2002] für die allgemeine Bewertung herangezogen, zum anderen bestehende EJBs auf der Basis statischer Quellcodemetriken analysiert.

5.4.2 Berücksichtigung der Spezifikationebenen

Die ordinale bzw. plazierende Bewertung des EJB-Komponentenansatzes erfolgt hinsichtlich der Zweckmäßigkeit des Einsatzes der vorgeschlagenen Spezifikationsebenen, der expliziten Berücksichtigung innerhalb der EJB-Komponenten bzw. potentieller Widersprüche zu diesen.

- *Schnittstellenebene* – Der Zugriff auf die Funktionen einer EJB-Komponente kann über das Remote- und Home-Interface bzw. ab der EJB-Spezifikation 2.0 auch über entsprechende lokale Ausprägungen dieser Interfaces erfolgen. Aus fachlicher Sicht kommt dem Remote-Interface die ausschließliche Bedeutung zu. Im folgenden wird die Implementierung eines Remote-Interfaces beispielhaft wiedergegeben.

```
public interface EuroCalcRemote extends javax.ejb.EJBObject {
```

```
// Umrechnung Euro-Betrag in DM
public double euro_to_dm(double amount) throws RemoteException;

// Umrechnung DM-Betrag in Euro
public double dm_to_euro(double amount) throws RemoteException;
```

Die Verwendung der OMG IDL entsprechend [Fachkomponenten 2002] zur Spezifikation auf Schnittstellenebene bringt bei der Verwendung von EJBs aus unserer Sicht keine signifikanten Vorteile, da die Generierung von Stubs bzw. Skeletons bereits eine EJB-inhärente Funktionalität ist.

- *Verhaltensebene* – Die Formulierung von Vor- und Nachbedingungen bzw. Invarianten kann mittels der vorgeschlagenen Notation (außerhalb des EJB) durchgeführt werden. Die Sicherstellung dieser Festlegungen kann sowohl programmtechnisch innerhalb des Remote-Interfaces, als auch dynamisch durch Verwendung von Umgebungsvariablen zur wertmäßigen Belegung spezifizierter Invarianten der EJB-Komponenten erfolgen.

- *Abstimmungebene* – Die Beziehungen zwischen den Diensten verschiedener EJB-Komponenten werden innerhalb des Deployment-Descriptors beschrieben. Festlegungen zur Reihenfolge, wie dieses die Spezifikation vorschreibt, erfolgen dabei jedoch nicht. Dementsprechend sind, sofern diese Angaben zur fehlerfreien Erledigung der Aufgabenstellung einer Fachkomponente benötigt werden, diese in externen Dokumenten, entsprechend dem Vorschlag der Spezifikation, niederzulegen.

- *Qualitätsebene* – Die Spezifikation der Qualitätseigenschaften einer EJB-Komponente ist derzeit ein völlig ungelöstes Problem. Die Spezifizierung dieser Eigenschaften erfordert, wie unter [Schmietendorf/Dumke 2001] dargestellt, die Analyse der Komponenten im Rahmen einer entsprechenden Referenzumgebung. Aussagen zu qualitativen Eigenschaften einer EJB-Komponente sind in externen Dokumenten zu pflegen.

- *Terminologie-, Aufgaben- und Vermarktungsebene* – Da für die EJB-Komponenten keine Spezifikation verfügbar ist, welche die Inhalte dieser Ebenen aufgreift, können die vorgeschlagene Notationen theoretisch ohne Änderungen zum Einsatz kommen.

Die derzeitige EJB-Spezifikation 2.0 unterstützt nur ausgewählte Merkmale der ersten drei Spezifikationsebenen explizit, die weiteren können verwendet werden, schlagen sich jedoch nur implizit innerhalb der EJB-Komponenten nieder. Insgesamt kann festgestellt werden, dass die Spezifizierung entsprechend [Fachkomponenten 2002] nur für ausreichend große Komponenten überhaupt sinnvoll ist, für einzelne EJB-Komponenten erscheint dies keineswegs zielführend, da vom Aufwand her nicht zu vertreten. Wir wollen daher im folgenden Abschnitt eine metrikenbasierte Analyse einzelner EJB bzw. mehrerer EJB, die über eine sogenannte Session Facade (J2EE-Pattern) gekapselt werden, durchführen.

5.4.3 Metriken-basierte Analyse elementarer EJB

Eine wesentliche Rolle für die erfolgreiche Wiederverwendung von EJB-Komponenten spielt deren Granularität, also die Größe einer solchen Komponente. In diesem Zusammenhang liegt es nahe, Kriterien für die „richtige“ Größe von EJB-Komponenten über empirische Untersuchungen vorhandener Komponenten zu finden.

Für die metrikenbasierte Analyse wurden 24 EJBs (20 Entity-Beans und 4 Session-Beans) herangezogen. Im folgenden sollen dementsprechend 24 EJB-Komponenten, welche eine

EAI[5]-Integrationsapplikation zur Steuerung und Überwachung des Bereitstellungsprozesses von Netzwerkprodukten der Telekommunikation realisieren, untersucht werden. In einem ersten Ansatz wurden die Bestandteile dieser Komponenten hinsichtlich ihres Umfangs (d.h. effektiver Lines of Code - eLoC) untersucht.

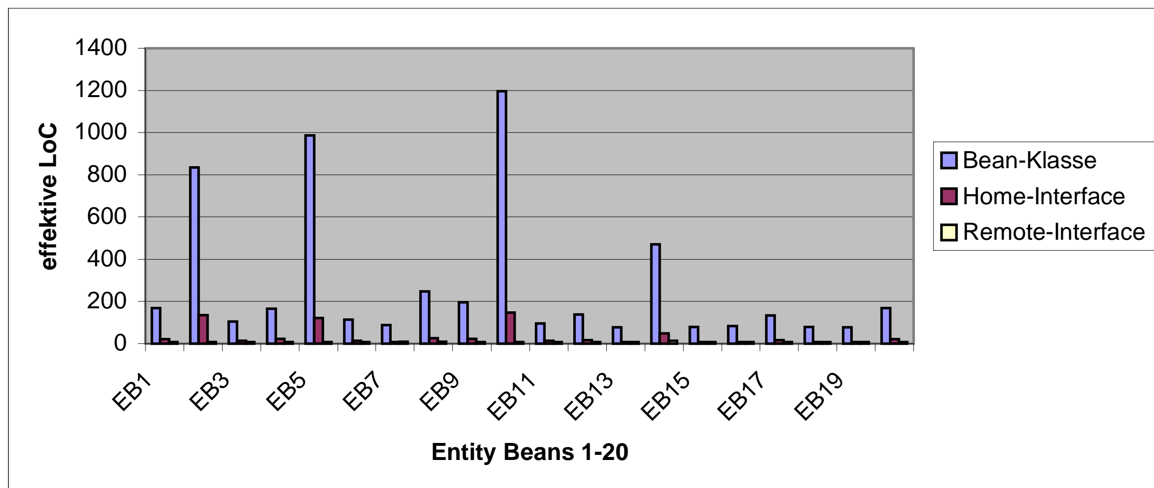

Abbildung 5.5: Umfangsmetriken von EntityBean-Komponenten (kurz EB)

In Abbildung 5.5 kann der Umfang von 20 untersuchten EJB-Komponenten (hier vom Typ Entity-Beans) nachvollzogen werden. Auffällig ist, dass der größte Teil der Komponenten bei der Bean-Klasse einen Umfang von ca. 100 bis 200 eLoC aufweist. Die Größe des Home-Interfaces korreliert sehr stark mit der Größe der Bean-Klasse im Verhältnis von 1:8 bei einem Korrelationskoeffizienten von $r = 0.985$, wie in Abbildung 5.6 ersichtlich. Das Remote-Interface weist eine Größe von durchschnittlich 7 eLoC auf.

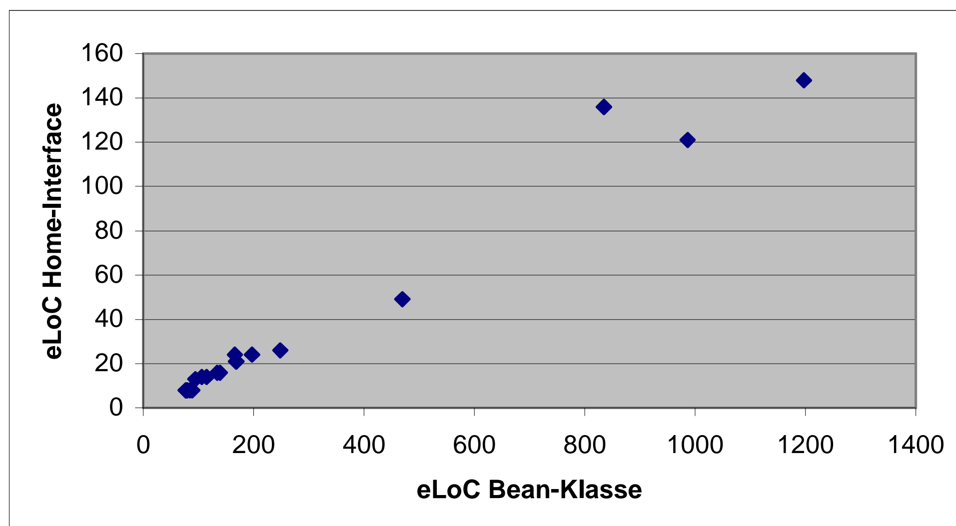

Abbildung 5.6: Zusammenhang zwischen Bean-Klasse und Home-Interface

Im folgenden sollen 4 SessionBeans untersucht werden. Während die Größe der Bean-Klasse, welche die fachliche Funktionalität realisiert, zwischen minimal 164 eLoC und maximal 756 eLoC schwankt, bleibt der Umfang der Home- bzw. Remote-Interfaces mit ca. 6 eLoC bzw.

[5] Enterprise Application Integration

10 eLoC weitgehend konstant. Durch die EJB-Komponenten werden innerhalb des Remote-Interfaces die folgende Anzahl von Diensten angeboten:

- Session Bean 1: 4 angebotene Dienste
- Session Bean 2: 3 angebotene Dienste
- Session Bean 3: 5 angebotene Dienste
- Session Bean 4: 6 angebotene Dienste

Der Anzahl der angebotenen Dienste kann durchaus als schmale und übersichtliche Schnittstelle angesehen werden, welche dem 7±2 Gesetz von G. A. Miller (1956) folgt.

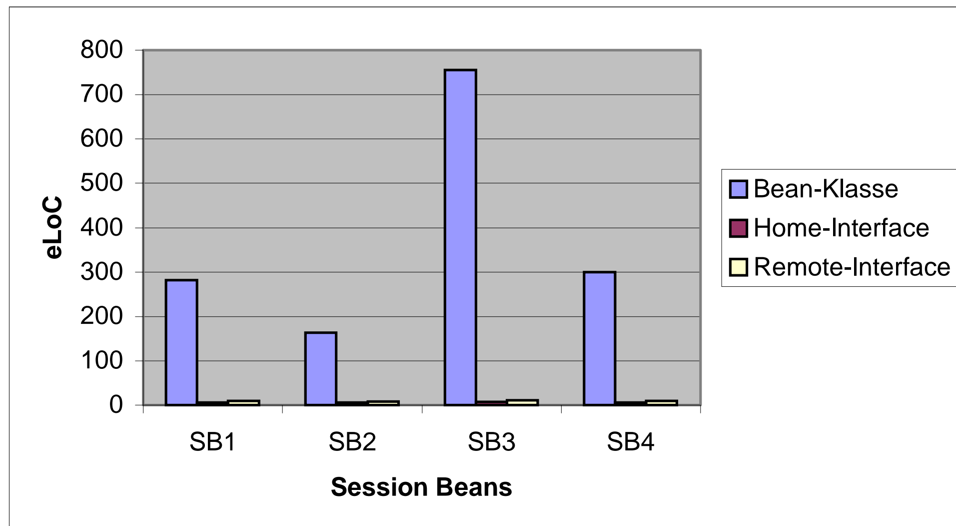

Abbildung 5.7: Umfangsanalyse von Session Beans (kurz SB)

5.4.4 EJB-basierte Fachkomponenten durch Design-Pattern

Die vorhergehende Analyse elementarer EJBs zeigte ausgewählte Eigenschaften dieser Komponenten auf. Es sei darauf verwiesen, dass hier selbstverständlich noch keine statistische Sicherheit aufgrund der geringen Anzahl analysierter Komponenten erzielt werden konnte. Dennoch zeigen sich erste Trends, welche die in [Schmietendorf et al. 2000], [Lezius 2001], [Beneken 2002] getätigten Aussagen weitgehend bestätigen. Insbesondere die geringe Größe und die hohen Abhängigkeiten von anderen Komponenten legen den Schluss nahe, dass nicht jede EJB automatisch eine Fachkomponente ist bzw. überhaupt sein kann. Erst der Einsatz entsprechender EJB-Muster erlaubt tatsächlich die Entwicklung von Fachkomponenten. In diesem Zusammenhang hat sich insbesondere das Muster Session Facade durchgesetzt.

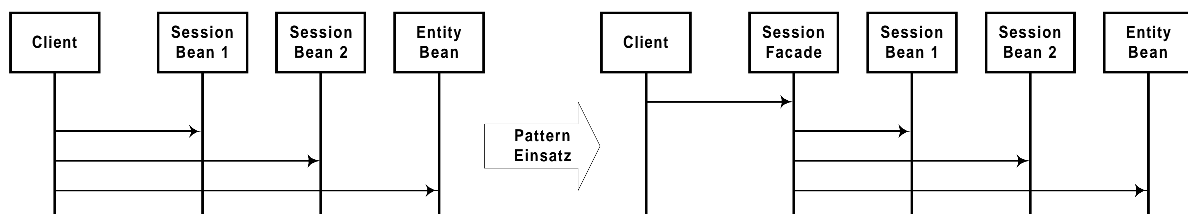

Abbildung 5.8: Einsatz des Session Facade Muster

Das Session-Facade-Muster (siehe auch Abbildung 5.8) unterstützt die Entwicklung wiederverwendbarer grobgranularer Fachkomponenten, reduziert potentiellen Netzwerkverkehr zwischen Client und Server und bietet darüber hinaus die Möglichkeit eines zentralisierten Transaktions- und Security-Managements. Innerhalb der im vorhergehenden Abschnitt analysier-

ten Session Beans werden entsprechend dem Entwurfsmuster Session Facade mehrere Entity Beans referenziert, so dass der Zugriff auf die komplette Fachkomponente nur noch über die bereits dargestellten Session Beans erfolgt. Die folgende Tabelle gibt eine zusammenfassende Sicht auf die Anzahl der jeweils referenzierten Komponenten (hier Entity Beans) und deren Gesamtumfang in eLoC wieder:

Tabelle 2: Übersicht der Session Facade Fachkomponenten (SB und EB)

Fachkomponenten mit Session Facade	Anzahl angebotener Dienste	Umfang der Session Beans in eLoC	Anzahl referenzierter Entity Beans	eLoC aller Entity Beans
Fachkomp. 1 Session Bean 1	4	281	5	1020
Fachkomp. 2 Session Bean 2	3	164	4	772
Fachkomp. 3 Session Bean 3	5	756	16	4290
Fachkomp. 4 Session Bean 4	4	300	7	1301

Es zeigt sich, dass sich zwischen dem Umfang der Session Beans und der Anzahl, aber auch dem Umfang in eLoC der referenzierten Entity Beans ein Zusammenhang ergibt.

Abbildung 5.9 zeigt diesen Zusammenhang noch einmal deutlicher auf, wobei sich für die 4 Stichproben ein Korrelationskoeffizient von r = 0,99, d.h. eine sehr hohe Korrelation ergibt.

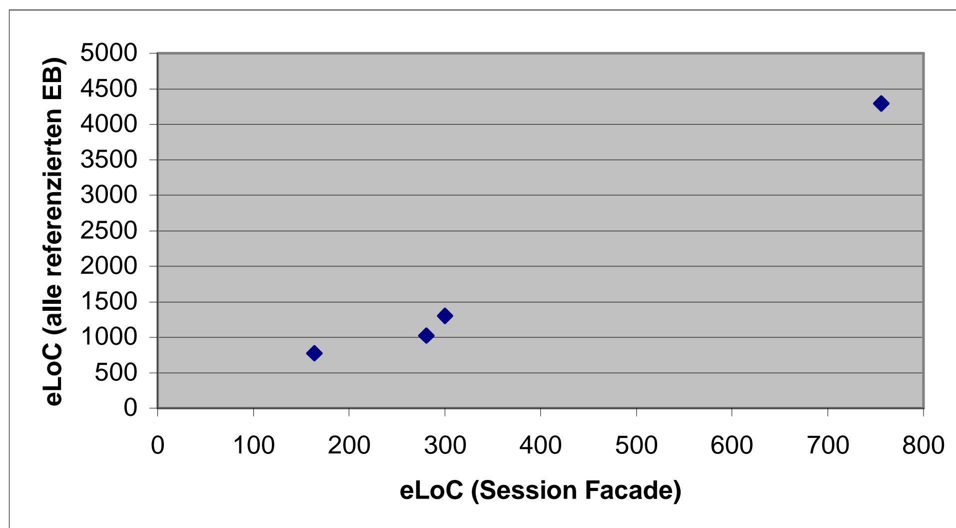

Abbildung 5.9: Session-Facade und Summe der Entity Beans

5.4.5 Potentielle Design-Empfehlungen

In Auswertung der durchgeführten Untersuchungen könnten erste Design-Empfehlungen[6] für Fachkomponenten, welche auf der Grundlage von EJB-Komponenten erstellt werden, folgendermaßen aussehen:

[6] Aufgrund der geringen statistischen Sicherheit der durchgeführten Untersuchungen sind diese mehr als Ansätze für weitere Untersuchungen, denn feststehende Erkenntnisse zu werten.

- Die Schnittstelle zu einer Fachkomponente wird immer auf der Grundlage einer Session Facade Bean und deren Home- bzw. Remote-Interface gebildet.
- Schnittstellen innerhalb der Fachkomponente zwischen verwendeten EJB sollten als lokale Interfaces realisiert werden und von außen nicht sichtbar sein.
- Das Remote-Interface sollte nicht mehr als 7±2 Dienste enthalten, welche sowohl den Zugriff auf die Komponente als auch die Bündelung externer Abhängigkeiten regeln.
- Der Umfang einer Fachkomponente sollte mindesten 1000 bis einige 10.000 eLoC groß sein und auf mehreren EJB-Komponenten basieren.
- Zwischen dem Umfang der Session Facade und den dahinter liegenden EJB-Komponenten sollte sich ein Verhältnis von 1 : 5 bis 1 : 10 einstellen.

Die durchgeführten empirischen Analysen zur Granularität von EJB-Komponenten stellen einen ersten Ansatz dar, um potentielle Bewertungskriterien von Fachkomponenten zu gewinnen. Zur effektiven Verwendung der Ergebnisse von Softwaremessungen werden empirisch untersetzte Bewertungsmodelle benötigt, die derzeit noch nicht zur Verfügung stehen. Das hier verwendete basiert noch auf sehr vagen Annahmen und muß selbstverständlich verbreitert werden. In diesem Zusammenhang ist insbesondere die Verwendung weiterer Softwaremetriken zur statischen, aber auch dynamische Analyse von EJB-Komponenten geplant.

Über die Verwendung von Softwaremetriken kann das Design einer Fachkomponente analysiert werden, nicht jedoch die Feststellung, dass es sich um eine Fachkomponente handelt. Inwieweit es sich bei einer EJB bzw. der Aggregation mehrerer EJBs über eine Session Facade um eine Fachkomponente handelt, hängt selbstverständlich nicht vom Umfang oder von der Anzahl verfügbarer Schnittstellen ab, sondern vielmehr von der fachlichen Zweckbestimmung. Da hier jedoch semantische Aspekte im Vordergrund stehen, die einer Formalisierung nur schwer zugänglich sind, kann die Festlegung, ob es sich um eine Fachkomponente handelt, nur durch den Komponentenentwickler bzw. den Anwender (Assembler) selbst durchgeführt werden. Im Ergebnis der Untersuchungen kann festgehalten werden, dass nur die über eine Session Facade (Session Bean) referenzierten EJB-Komponenten wirklich die Möglichkeit zur Implementierung von Fachkomponenten bieten, keinesfalls aber eine elementare EJB.

5.5 Performance Analysen im Umfeld der EJB-Technologie

5.5.1 Zielstellung einer Performanceanalyse

Die derzeit am Markt verfügbaren Softwarekomponenten bieten zumeist keine Aussagen hinsichtlich ihres Performanceverhaltens während der Programmausführung. Eine kostengünstige Abklärung potentieller Performance-Probleme ist in den frühen Phasen einer komponentenorientierten Entwicklung daher nicht möglich. Probleme mit der Performance des Systems werden so erst beim Abnahmetest oder während der Systemeinführung erkannt, wodurch aufwendige Nacharbeiten, die sich typischerweise über alle Phasen der Softwareentwicklung erstrecken, notwendig werden. Es stellt sich die Frage, ob unter solchen Bedingungen Komponenten überhaupt die Erwartungen an höhere qualitativen Eigenschaften des endgültigen Softwareproduktes erfüllen können oder aber das Komponentenparadigma insbesondere in diesem Bereich seine Grenzen erreicht hat. Bereits im Jahr 1996 findet sich innerhalb eines technischen Reports des Software Engineering Institute (SEI) dazu die folgende Feststellung [Klein 1996]:

> *„No standard of practice exist for how to evaluate the performance of a software component. As reliance on COTS increases, the risk of discovering performance*

problems after it is too late may increase for systems for which performance predictability is critical."

Bis zum heutigen Tag konnte diese Problemstellung keiner befriedigenden Lösung zugeführt werden. Wenngleich wie unter [Poulin 2001] die Aufnahme und Bereitstellung von Metriken zur Quantifizierung der Performanceeigenschaften von Softwarekomponenten immer wieder angemahnt wird, fehlt es jedoch zumeist an einer unter wirtschaftlichen Gesichtspunkten effizient umsetzbaren Vorgehensweise zur Ermittlung dieser Größen.

Um die qualitativen Eigenschaften der Effizienz einer Komponente überhaupt im Rahmen der Spezifikation angeben zu können, müssen diese durch konkrete Verfahren der Performanceanalyse (z.B. Benchmarks, modellbasierte Analysen, Schätzverfahren) ermittelt werden. Die Motivation zur Performance-Analyse von Komponenten soll durch das folgende Zitat von [Harmon 2001] unterstrichen werden:

> *Fifteen years ago, it was said that 80% of an application performance depended on the implementation algorithm. Today, we can say that 80% of an application performance depends on the interface design of its components.*

Die bisher im Umfeld der EJB-Technoloigie existierenden Performanceanalysen konzentrieren sich vordergründig auf die folgenden kurz umrissenen Zielstellungen:

- Performance-Analyse der Laufzeitumgebung von Komponenten, im Fall der EJBs wären dies der Container bzw. Application Server,
- Performance-Analyse des Vorgangs zur Installation (Deployment) von EJB-Komponenten innerhalb entsprechender Container,
- Performance-Analyse und –Spezifikation von einzusetzenden Komponenten, d.h. welche leistungsrelevanten Eigenschaften besitzt eine Komponente.

Im Rahmen dieses Preprints sollen die vorgenannten Zielrichtungen und die dazu derzeit vorhandenen Forschungsarbeiten dargestellt werden.

Im folgenden sind einige ausgewählte Fragestellungen für eine Performanceanalyse aufgezeigt:

- Wie verhalten sich EJB-Server unter der Last steigender Client-Zugriffe und wie werden dabei die verfügbaren Hard- und Software-Ressourcen ausgenutzt?
- Welche Einflüsse auf die Performance hat die Verwendung der unterschiedlichen EJB-Typen (Entity-Beans, Session-Beans, Message Driven Beans)?
- Welchen Einfluss hat die Lokalität des Methodenaufrufs auf die Performance. Zu Berücksichtigen sind Methoden-Aufrufe im gleichen Container/EJB-Server, zu verschiedenen Containern innerhalb des selben EJB-Servers oder aber zu anderen EJB-Servern.
- Bringt es Vorteile, den EJB-Server als Teil eines Datenbankservers zu integrieren und wenn ja, wie können diese quantifiziert werden?
- Wie können Hard- und Software-Systeme für die EJB-Plattform bemessen werden, um den Nutzeranforderungen gerecht zu werden?

5.5.2 Allgemeine Vorgehensweise zur Performanceanalyse

Für die Beschreibung der Performance-Charakteristik einer Softwarekomponente werden die folgenden Informationen benötigt:

- *Standardlastprofil:* Die durch die Komponente angebotenen Kernfunktionen sind hinsichtlich ihrer Aufrufhäufigkeit und der dabei übergebenen Datenmengen zu charakterisieren. Hierbei sollte primär die typische Verwendung einer Komponente berücksichtigt werden.

- *Ressourcenverbrauch:* Unter dem gegebenen Lastprofil ist der Ressourcenverbrauch sowohl von anderen softwaretechnischen Komponenten bzw. Services als auch hinsichtlich der in Anspruch genommenen Hardwareressourcen (z.B. CPU-, I/O-, Speicher) festzulegen.

- *Performance-Verhalten:* Das zeitliche Verhalten der im Lastprofil festgelegten Kernfunktionen der Komponente (Antwortzeit und Durchsatz) ist zu dokumentieren. Darüber hinaus sind Angaben über das interoperable Verhalten der Komponente zu anderen Softwarekomponenten und Services darzustellen.

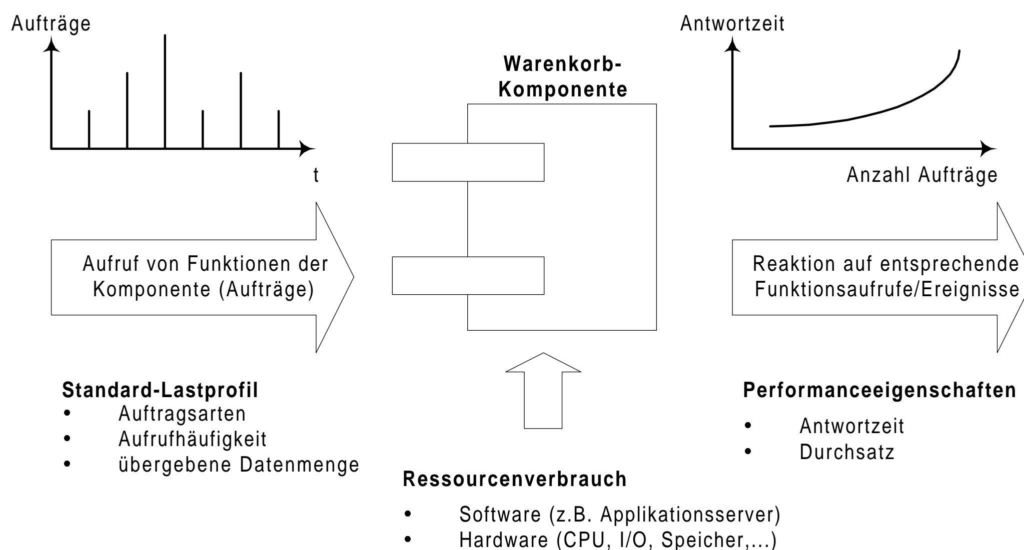

Abbildung 5.10: Elemente der Performance-Beschreibung [Schmietendorf/Scholz 2000]

Für die Ermittlung dieser Angaben ist die Ausführung von Benchmarks erforderlich, wie unter (Sametinger 1997) kurz angeregt. Ähnlich der Beschreibung elektronischer Baugruppen werden dabei für eine konkrete Softwarekomponente umfangreiche Kennlinienfelder ermittelt und innerhalb entsprechender Datenblätter zur Verfügung gestellt. Für eine effiziente Benchmark-Ausführung sollten Standardtestumgebungen und Lasttreiber verwendet werden, die wiederholende Funktionsaufrufe eines zu definierenden bzw. zu konfigurierenden Lastmixes erlauben. Methodisch können die folgenden Schritte dabei unterschieden werden:

1. Entwicklung eines Lastprofils, d.h. Identifizierung der wesentlichen durch die Komponenten angebotenen Funktionen (Methoden) und deren Aufrufverhalten.

2. Bereitstellung einer Laufzeitumgebung für die Komponenten, u.U. Installation weiterer notwendiger Komponenten und Services im Rahmen der Testumgebung.

3. Bei Zugriff der Komponenten auf einen Datenbestand sind entsprechende Testdaten bereitzustellen.

4. Atomare Ausführung der zu untersuchenden Funktionen der Komponenten und Aufzeichnung der Aufrufe durch einen Monitor

5. Sukzessive Modifikation der aufgezeichneten Funktionsaufrufe, so dass unterschiedliche Datenbereiche bei der Ausführung genutzt werden.

6. Konfiguration des Lasttreibers entsprechend dem für die Komponente ermittelten relevanten Lastprofil.

7. Ausführung des Benchmarks und Aufnahme von internen und externen Performance-Metriken.

8. Festhalten der Ergebnisse in entsprechenden Datenblättern zur Beschreibung der Performance-Eigenschaften der vermessenen Komponente.

Bei dieser Vorgehensweise gestaltet sich die Determinierung des Ressourcenbedarfs problematisch. Erfolgt ein Bezug auf direkte Hardwareeigenschaften des gewählten Referenzsystems, wie z.B. den CPU- oder Speicherverbrauch, sind die Performance-Aussagen nur für die im Test verwendete Hardware gültig. Eine Übertragbarkeit auf andere Systeme ist damit nur schwer möglich. Eine bedingte Alternative stellt der Bezug auf Messgrößen dar, die beispielsweise durch Standard-Benchmarks (z.B. SPEC jbb2000 - Java Business Benchmark) ermittelt werden.

5.5.3 Verfügbare Arbeiten

Verfügbare modell- und messwertbasierte Performanceanalysen zur EJB-Technologie finden sich sowohl im universitären als auch industriellen Bereich. Im folgenden werden die Inhalte dreier ausgewählte Studien kurz angesprochen:

- Im Rahmen des EJB-Vergleichsprojekts [Procházka 2000] erfolgten Benchmarks-Tests der Funktionalität, Interoperabilität und Performance. In den Performancetest werden insbesondere die Verteilungsmechanismen und der Overhead der Transaktionsverarbeitung einbezogen.

- Mit dem ECperfTM steht ein Benchmark zur Verfügung, der die Bewertung der Skalierbarkeit und Performance von EJB-1.0-kompatiblen Containern und Servern ermöglicht. Der ECperfTM testet insbesondere die Eigenschaften des Containers in Bezug auf den Ressourcenbedarf, Connection-Pooling oder auch die Aktivierung und Deaktivierung von EJB-Komponenten bzw. die Nutzung entsprechender Caching-Mechanismen. [Subramanyam 2000]

- In [Llado 2000] finden sich Ansätze zur modellbasierten Performance-Analyse ausgewählter Aspekte der EJB-Technologie, die sowohl analytisch als auch simulativ einer Lösung zugeführt werden. Primär beziehen sich diese auf die Wahrscheinlichkeit der gegenseitigen Blockierung gleichzeitig ausgeführter Methodenaufrufe konkreter EJB-Instanzen innerhalb des selben Containers und die dabei erreichbare Performance (Antwortzeit/Durchsatz). Im Rahmen von [Lüthi 2001] werden diese Ansätze durch die Verwendung intervallbasierter Modellparameter zur Berücksichtigung unscharfer Modelleingangsgrößen weiter vertieft.

- Unter [Schill 2000] finden sich Performanceaussagen zu unterschiedlichen Applikations-Servern, die einem Benchmark mittels einer Anwendung aus dem Versicherungsbereich unterzogen wurden. Durch den parallelen Start entsprechender Testclients, die webbasierte Interaktionen gegenüber der EJB-basierten Applikation simulieren, wurden Performancegrößen, wie Ausführungszeiten, an unterschiedlichen Messpunkten der gesamten Applikation gewonnen.

Die vorgestellten Arbeiten beschäftigen sich vordergründig mit den Performanceeigenschaften des EJB-Servers (Container) und einer darin zur Ausführung gebrachten Applikation, den zur Installation (Deployment) von EJB-Komponenten innerhalb entsprechender Container benötigten Zeiten und ausgewählten Detailtechniken, die Einfluss auf das spätere Performanceverhalten haben. Aussagen zum Performanceverhalten atomarer EJB-Komponenten bzw. methodische Vorgehensweisen zur Ermittlung dieser Größen finden sich derzeit nicht in der einschlägigen Literatur.

5.5.4 EJB-Komponenten-Performance

Ein Ansatz zur Performanceanalyse von EJB-Komponenten findet sich unter [Nakonz 2001]. Die folgende Abbildung 5.11 zeigt die Oberfläche des dort entwickelten Benchmarks zur Vermessung unterschiedlicher EJB-Typen. Die Zielstellung dieses Benchmarks bestand in der Identifizierung potentieller Einflusskriterien auf die Performance von EJB-Komponenten.

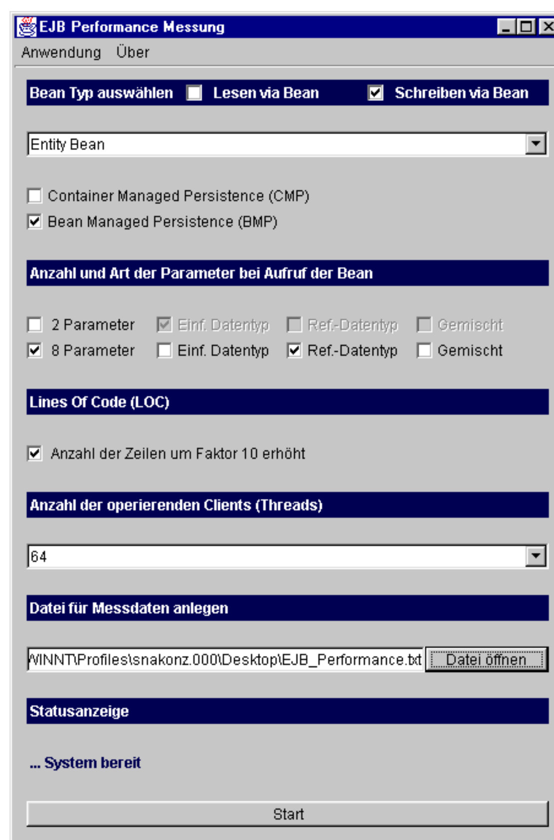

Abbildung 5.11: Oberfläche zur Konfiguration des EJB-Benchmark

Berücksichtigt werden die EJB-Typen Session Beans und Entity Beans. Im Rahmen des Benchmarks können diese Komponenten einer Variation hinsichtlich ihrer Größe unterzogen, die Anzahl und Art der übergebenen Parameter variiert sowie die auf die Komponente parallel zugreifenden Clients skaliert werden. Darüber hinaus können die Persistenzeigenschaften

(Zugriffe auf das verwendete Datenbanksystem) im Falle der Verwendung von Entity Beans durch den Container oder aber die Komponente selbst verwaltet werden.

Abbildung 5.12 zeigt die Initialisierungszeiten der unterschiedlichen EJB-Typen. Darüber hinaus wurde die Größe der EJBs (Umfang in Lines of Code) variiert.

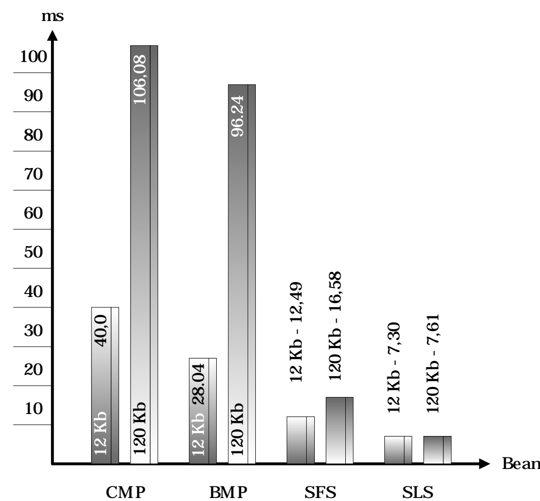

Abbildung 5.12: EJB-Initialisierungszeiten

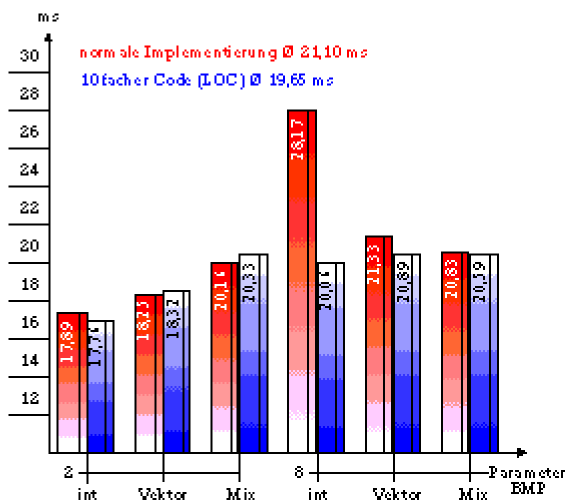

Abbildung 5.13: Schreiben über ein Entity Bean (BMP)

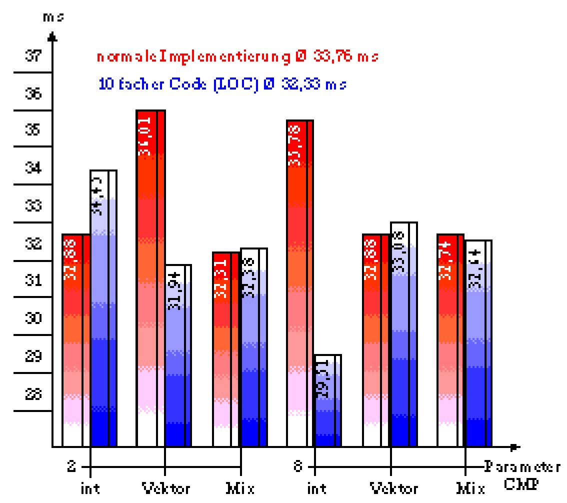

Abbildung 5.14: Schreiben über ein Entity Bean (CMP)

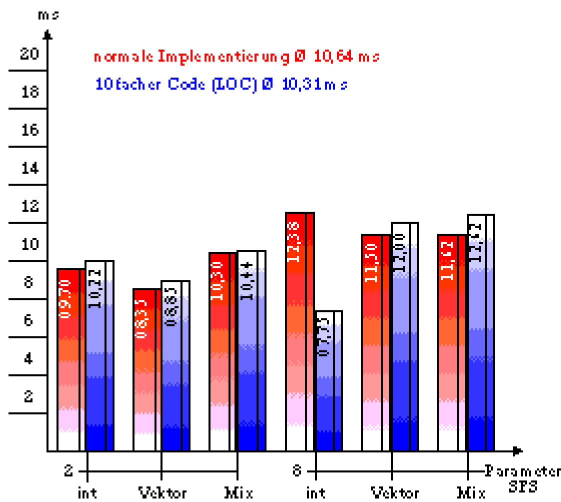

Abbildung 5.15: Schreiben über ein Session Bean (SFS)

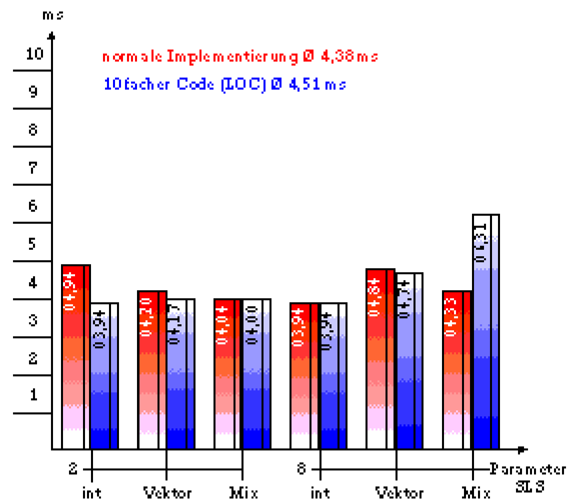

Abbildung 5.16: Schreiben über ein Session Bean (SLS)

Der EJB-Performance-Benchmark wurde innerhalb einer entsprechenden Testinstallation zur Ausführung gebracht, im folgenden ist die Konfiguration und die dabei genutzte Hardware angegeben.

- Client-System (Last-Treiber): CPU Pentium II 333 Mhz, 256 MByte RAM, 6 GB Festplatte IDE
- Application Server (BEA Weblogic): CPU Pentium II 333 Mhz, 384 MByte RAM, 6 GB Festplatte IDE
- Database (Oracle 8i): CPU PIII 500 Mhz, 500 Mbyte RAM, 20 GB Festplatte IDE

Berücksichtigt werden im Benchmark Session- (SFS[7] und SLS[8]) und Entity-Beans (CMP[9] und BMP[10]). Weitere Benchmarkergebnisse beziehen sich auf die Deserialisierung von Bean-Instanzen sowie lesende und schreibende Zugriffe auf die Komponenten unter Variation der Anzahl übergebender Parameter. Die Abbildungen 5.13 bis 5.16 zeigen ausgewählte Ergebnisse der Benchmarkausführung.

Die vorgestellten Arbeiten zeigen erste Ansatzpunkte für mögliche Performanceanalysen der EJB-Technologie.

- Die Zeitdauer für die Instanziierung eines Entity-Bean hängt maßgeblich von dessen Größe ab; bei einer Session-Bean spielt die Größe der Bean-Klasse dagegen keine Rolle.
- Session-Beans benötigen eine wesentlich kürzere Initialisierungszeit im Vergleich zu Entity-Beans.
- Einen geringen Einfluss auf die Performance besitzen die der Bean übergebenen Parameter und die Tatsache, ob es sich um einen schreibenden oder lesenden Zugriff handelt.

---

[7] SFS Statefull Session Bean
[8] SLS Stateless Session Bean
[9] Container Managed Persistence
[10] Bean Managed Persistence

- Die umfangreiche Verwendung von Entity-Beans impliziert bei den derzeit verfügbaren Container- bzw. Applicationserver-Implementierungen Performancerisiken.

Weitere Aspekte beziehen sich auf die Hinterlegung von Performanceeigenschaften, z.B. innerhalb der Komponente selbst, und die Berücksichtigung geforderter Service-Level im Rahmen des Performance-Managements. Aussagen dazu finden sich z.B. unter [Schmietendorf/Dumke 2001]. Die am Entwicklungszentrum Berlin durchgeführten Performance-Analysen zeigten bereits vielfältige Problemstellungen bei der Performance-Analyse atomarer EJB-Komponenten auf. Beispielhaft seien der damit verbundene Aufwand, die erreichbare Genauigkeit bei der Aufnahme der Performance-Metriken, die Verallgemeinerung der Ergebnisse, die Parallelisierung der Zugriffe auf die Bean-Instanz oder auch die Granularität der untersuchten Komponenten selbst genannt. Aus unserer Sicht kann festgestellt werden, dass eine Performance-Analyse einzelner EJB-Komponenten nicht zielführend ist, da diese vom Aufwand her nicht zu vertreten ist.

6 Zusammenfassung

Der hier vorliegende Preprint gibt nach einer einführenden Darstellung der Grundlagen einer komponentenbasierten Softwareentwicklung einen Einblick in die Technologie der Java-Komponenten. Dabei werden sowohl die JavaBeans als auch die Enterprise JavaBeans im Detail vorgestellt und anhand eines konkreten Beispiels die Themenstellung der Entwicklung komponentenbasierter Anwendungen verdeutlicht. Innerhalb des Kapitels 5 wird dann auf im wissenschaftlichen Kontext derzeit bearbeitete Themenstellungen zur Beschreibung, Spezifikation und Bewertung von Komponenten eingegangen. Dabei werden ausgewählte Ergebnisse empirischer Analysen der statischen Eigenschaften von EJB-Komponenten und die Themenstellung der Performance-Analyse von EJB-Komponenten aufgegriffen. Insbesondere im Umfeld der Spezifikation qualitativer Eigenschaften einer Komponente (z.B. Performance) zeigen sich die Grenzen des Komponentenparadigmas. Wenngleich theoretisch die Spezifikation der qualitativen Eigenschaften einer konkreten Komponente unter Berücksichtigung diverser Randbedingungen (z.B. benötigte Hard- und Softwarekomponenten, Lastprofil,...) möglich wäre, erscheint ein derartiges Vorgehen unter wirtschaftlichen Gesichtspunkten nicht zweckmäßig. Aus Sicht der Autoren ist hier der Einsatz agentifizierter Komponenten sinnvoll, welche über die Fähigkeit der Wahrnehmung ihrer Umgebung auf ggf. auftretende Performanceeinbußen dynamisch reagieren können. Somit ist die Frage der Qualität einer Komponente nur im Kontext der Gesamtarchitektur, in der sie abläuft, zu beantworten.

7 Quellenverzeichnis

[Fachkomponenten 2002] Ackermann, J.; Brinkop, F.; Conrad, S.; Fettke, P.; Frick, A.; Glistau, E.; Jaekel, H.; Klein, U.; Kotlar, O.; Loos, P.; Mrech, H.; Ortner, E.; Overhage, S.; Sahm, S.; Schmietendorf, A.; Teschke, T.; Turowski, K.: Vorschlag zur Vereinheitlichung der Spezifikation von Fachkomponenten. Memorandum des GI-Arbeitskreises 5.10.3, Februar 2002

[Coulange1998] Coulange, B.: Software Reuse. Springer-Verlag, London 1998

[Ezran 1998] Ezran, M., Morisio, M., Tully, C.: Practical Software Reuse: The essential Guide. Paris: Freelife Publ., 1998

[Jacobson 1997] Jacobson, I, Griss, M., Jonsson, P.: Software Reuse (Architecture, Process and Organization for Business Success), Reading/MA: Addison-Wesley 1997

[Poulin 1997] Poulin, J.S.: Measuring Software Reuse. Principles, Practices, and Economic Models. Reading/MA: Addison-Wesley 1997

[Sodhi1998] Sodhi, J.; Sodhi, P.: Software Reuse. Domain Analysis and Design Processes. New York ...: Mc Graw-Hill 1998

[Udell 1994] Udell, J.: Component software. BYTE (5)1994 19, S. 46 - 55

[Klein 1996] Klein, M.: State of the Practice Report. Problems in the Practice of Performance Engineering, Technical Report, No. SEI-95-TR-020, Software Engineering Institute, Pittsburgh, 1996

[Llado 2000] Lladó, C. M.; Harrison, P. G.: Performance Evaluation of an Enterprise JavaBean Server Implementation, in Proceedings of WOSP2000, Ottawa/Canada, September 2000

[Lüthi 2001] Lüthi, J.; Lladó, C. M.: Interval Parameters for Capturing Uncertainties in an EJB Performance Model, in Proc. of SIGMETRICS 2001/PERFORMANCE 2001, Special Issue of Performance Evalution Review, Vol. 29 No. 1, Cambridge/MA, USA

[Procházka 2000] Procházka, M.; Tuma, P.; Pospíšil, R.: Enterprise JavaBeans Benchmarking, Faculty of Mathematics and Physics, Department of Software Engineering, Charles University Prag/Czech Republic, 2000

[Schill 2000] Schill, A.; Neumann, O.: EJB-Server-Vergleich, in Proc. Hochschul-/Industrie-Konferenz „Middleware“, TU Dresden, 8. December 2000

[Subramanyam 2000] Subramanyam, S.; Sucharitakul, A.: The ECPerfTM, J2EETM Server Performance and Scalability Workload, White Paper, URL: java.sun.com

[Merkle 2000] Merkle, B.; Johann, M.: Produktüberblick von EJB-basierten Applikations-Servern. Java Spektrum, Ausg. 2, März/April 2000

[Beneken 2002] Beneken, G.; Schamper, M.: Qualität ist das beste Rezept – Komponenten mit J2EE-Pattern. Java Spektrum, Ausg. 2, Febr./März 2002

[Orfali 1996] Orfali, R.; Harkey, D.; Edwards, J.: The Essential Client/Server Survival Guide. Second Edition, John Wiley & Sons: New York, 1996

[Orfali 1998] Orfali, R.; Harkey, D.: Client/Server Programming with JAVA and CORBA. Second Edition, John Wiley & Sons: New York, 1998

[Ahern 2001] Ahern, D.M.; Clouse, A.; Turner, R.: CMMI Distilled. A Practical Introduction to Integrated Process Improvement. Addison-Wesley: Boston, San Francisco, New York, … 2001

[Balzert 1996] Balzert, H.: Lehrbuch der Software-Technik. Software-Entwicklung. Spektrum Akademischer Verlag GmbH: Heidelberg, Berlin, 1996

[Biggerstaff/Perlis 1989] Biggerstaff, T., Perlis, A.: Software Reusability. Band I und II in der Frontier Series. ACM Press, 1989

[Balzert 1998] Balzert, H.: Lehrbuch der Software-Technik. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag GmbH: Heidelberg, Berlin, 1998

[Brown 1998] Brown, A. W.; Wallnau, K. C.: The curent state of CBSE. IEEE Software, S. 37-46, September/October 1998

[Dumke 2001] Dumke, R.: Software Engineering – Eine Einführung für Informatiker und Ingenieure, Vieweg-Verlag, 2001

[Frank/Jung 2001] Frank, U.; Jung, J.: Prototypische Vorgehensweise für den Entwurf anwendungsnaher Komponenten S. 57-74, in Proc. zur Verbundtagung VertIS 2001, Universität Bamberg, 2001

[Griffel 1998] Griffel, F.: Componentware. Konzepte und Techniken eines Softwareparadigmas. dpunkt-verlag: Heidelberg, 1998

[Harmon 2001] Harmon, P.; Rosen, M.; Guttman, M.: Developing E-Business Systems and Architectures, Morgan Kaufmann Publishers, San Francisco 2001

[Helsel 1994] Helsel, R.: Cutting Your Test Development Time with HP VEE, Pretice Hall, Englewoods Cliffs, New Jersey, 1994

[Lezius 2001] Lezius, J.: Qualitätsbewertung von Softwarekomponenten auf der Basis von Metriken. Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2001 (fachliche Betreuung am Entwicklungszentrum Berlin der T-Nova)

[McIlroy 1968] McIlroy, M. D.: Mass Produced Software Components, in Naur, P.; Randell, B. (Hrsg.): Software Engineering: Report on a Conference by the NATO Science Commitee, S138, Brüssel 1968

[Nakonz 2001] Nakonz, S.: Benchmarking verteilter Komponentensysteme. Diplomarbeit, Fachhochschule für Technik und Wirtschaft Berlin, 2001 (fachliche Betreuung am Entwicklungszentrum Berlin der T-Nova)

[Schmietendorf et al. 2000] Schmietendorf, A.; Dimitrov, E.; Dumke, R.; Foltin, E.: Konzeption und erste Erfahrungen einer Metriken-basierten Software Wiederverwendung. In:

R. Dumke, F. Lehner: Software-Metriken. Deutscher Universitäts-Verlag/Gabler Edition Wissenschaft: Wiesbaden, 2000, S. 95

[Schmietendorf/Dimitrov 2001] Schmietendorf, A; Dimitrov, E: Possibilities of performance modelling with UML. In R. Dumke et. al.: Performance Engineering. State of the art and current trends. LNCS 2047, Springer 2001

[Schmietendorf/Dumke 2000] Schmietendorf, A.; Dumke R.: Metriken-basierte Bewertung von Software-Komponenten. In: Proc. CONQUEST 2000, Nürnberg, 14./15.9.2000, S. 104

[Schmietendorf/Dumke 2001] Schmietendorf, A.; Dumke, R.: Spezifikation von Softwarekomponenten auf Qualitätsebene. In Turowoski, K. (Hrsg.): Tagungsband zum 2. Workshop Modellierung und Spezifikation von Fachkomponenten (im Rahmen der vertIS 2001) S. 113-123, Universität Bamberg, Oktober 2001

[Schmietendorf/Scholz 2000] Schmietendorf, A.; Scholz A.: Spezifikation der Performance - Eigenschaften von Softwarekomponenten. In: Proc. Modellierung und Spezifikation von Fachkomponenten, Workshop im Rahmen der MobIS 2000, Siegen, S. 41

[SEI 2001] COTS Usage Risk Evaluation, http://www.sei.cmu.edu/cbs/cure-one-pager.html

[Solingen/Berghout 1999] Solingen, v. R.; Berghout, E.: The Goal/Question/Metric Method. McGraw Hill Verlag 1999

[Stal 2000] Stal, M.: Reich der Mitte, Die Komponententechnologien COM+, EJB und „CORBA-Components“, OBJECTspektrum 3/2000 S. 26-33

[SUN 1999] SUN Microsystems: Enterprise JavaBeans Specification, v 1.1, 1999

[Wloka 1995] Wloka, U.: Datenbanken. 12. Lehrbrief: Sicherung der Integrität in Datenbanken. VMS-Verlag: Hamburg, Dresden 1995

[Turowski 1999] Turowski, K.: Ordnungsrahmen für komponentenorientierte betriebliche Anwendungssysteme, In: Turowski, K. (Hrsg.): Tagungsband 1. Workshop komponentenorientierte betriebliche Anwendungssysteme, Magdeburg, 1999

8 Abkürzungen

EJB	Enterprise JavaBeans
EB	Entity-Bean
MDB	Message Driven Beans
SB	Session-Bean
DD	Deployment-Deskriptor
BMP	Bean Managed Persistence
CMP	Container Managed Persistence
ORB	Object Request Broker
CORBA	Common Object Request Broker Architecture
JVM	Java Virtual Machine
HTTP	Hyper Text Transfer Protocol
XML	Extensible Markup Language
UML	Unified Modelling Language
OMG	Object Management Group
FCM	Factor Criteria Metrics Model
GQM	Goal Question Metrics

Anlage: Bewertungskriterien für Application Server

Zur Auswahl eines J2EE-konformen Application Servers sollten entsprechende Kriterien aufgestellt werden, die sowohl technische als auch kommerzielle Aspekte berücksichtigen. Auf der Basis einer Multifaktorenanalyse können die identifizierten Kriterien mit Hilfe einer ordinalen Skalierung (z.B. von 1 bis 5) einer Bewertung unterzogen werden. Aufbauend auf dieser Bewertung kann eine Gesamtbewertung durchgeführt werden, wobei die einzelnen Kriterien nochmals einer Wichtung unterzogen werden können. Die im folgernde dargestellte Bewertungsmatrix enthält eine entsprechende Vorgehensweise, wobei ausgewählte Kriterien zur Auswahl eines Application Servers aus technischer und wirtschaftlicher Sicht hinterlegt wurden.

Tabelle 3: Multifaktorenanalyse auf der Basis technischer Kriterien

KRITERIUM	ERLÄUTERUNG	GEWICHT	BEWERTUNG
Standard-konformität	Hält das angebotene Produkt den aktuellen J2EE-Standard ein?		
Java-Version	Welche Java-Version wird für die Verwendung des Application Servers benötigt, können neue Versionen entsprechend angepasst werden?		
Persistenz-mechanismen	Welche Persistenz-Services werden unterstützt? - JDBC-Konformität - Eigene Datenbanktreiber für z.B. objektorientierte Systeme		
Transaktions-sicherung	Werden verteilte Transaktionen auf der Basis des Java Transaction Service (kurz JTS) durch das Produkt unterstützt?		
Middleware-konformität	Welche Middlewareansätze werden durch den Applikation-Server unterstütz? - CORBA (Common Object Request Broker Architecture) - MOM (Message orientierte Middleware)		
Entwicklungs-plattform	Durch welche Entwicklungsumgebungen wird der Application Server primär unterstützt?		
Integration mit den Werkzeugen für die Modellierung	Wie können Design-Modelle mittels einer Notation, wie z.B. UML, innerhalb der EJB-basierten Implementierung weiter verwendet werden. - Generieren entsprechender Quellcoderahmen für die im Modelle spezifizierten EJB-Komponenten. - Unterstützung eines Round-Trip Engineerings, d.h. Änderungen am Quellcode führen auch zu Änderungen innerhalb des Design-Modells.		
Integration mit den Werkzeugen für die	In welcher Form können Entwicklungsumgebung und Application Server miteinander integriert werden?		

KRITERIUM	ERLÄUTERUNG	GEWICHT	BEWERTUNG
Implementierung	- Wizard-Unterstützung für die Entwicklung, Paketierung und Deployment von EJB-Komponenten. - Integration der Entwicklung und des Deployments, automatische Vorgehensweisen sind hier von Vorteil. - Unterstützung der Vorgehensweise bei der statischen und dynamischen Fehlersuche.		
Performance des Application Servers	Hier sollten sowohl Erfahrungswerte aus durchgeführten Projekten, aber auch die auf Basis des ECPerf-Benchmarks gewonnenen Aussagen zum Performance-Verhalten des Application Servers herangezogen werden.		
Tuning-Möglichkeiten des Application Servers	Welche Möglichkeiten des Tuning bietet der Application Server, sinnvoller Weise sollte durch den Hersteller ein entsprechender Tuning-Leifaden zur Verfügung gestellt werden.		
Qualitative Eigenschaften des Produktes. Performance Sicherheit Skalierbarkeit ...	Optimierte Verarbeitungsmechanismen wie z.B.: - Versetztes Rückschreiben von Daten in die Datenbank, so können lang laufende Transaktionen innerhalb des Hauptspeichers durchgeführt werden und ein ressourcenintensiver Zugriff auf externen Speicher (Platte) vermieden werden. - Cache-Funktionen innerhalb des Hauptspeichers. Signifikante Performancevorteile bietet ein caching von Daten im Hauptspeicher gegenüber einem sogenannten „pass-though“-Vorgehen, wo Änderungen sofort auf die Festplatte geschrieben werden. - Skalierbekit des Application Servers bei Lasterhöhung, anzustreben ist eine lineare Skalierung. Bei den heutigen zeigt sich allerdings, dass nur ca. 4 Prozessoren effizient unterstützt werden können. - Angebotene Funktionen, die eine Hochverfügbarkeit unterstützen, dabei sollte auch die Interaktion mit den benötigten Betriebs- und Hardwaresystemen berücksichtigt werden. - Unterstützte Security-Funktionen, wie z.B. die Verwendung benutzerbezogener Access Control Lists (ACL) bzw. die Unterstützung des Java Authentification and Authorisation Service (kurz JAAS).		
Möglichkeiten des Systemmanagements	Wie können Application Server basierte Lösungen in bestehende Systemmanagement-Infrastrukturen eingebunden werden. Zu berücksichtigen sind unter anderem die folgenden Aspekte: - Nutzer-Administration - Fehlererkennung und Fehlerbehandlung - Monitoring der Ausführung		

KRITERIUM	ERLÄUTERUNG	GEWICHT	BEWERTUNG
	- Backupt und Revovery inkl. Aussagen zu den benötigten Zeiten eines Wiederanlaufs - Benötigte Zeiten für das Deployment		
Vorhandene Infrastruktur	Wie sieht die derzeitige Infrastruktur des eigenen Unternehmens aus? - Eingesetzte Datenbankmanagementsysteme und Verwendbarkeit unter den analysierten Application Servern - Eingesetzte Webserver und deren weitere Verwendung unter dem avisierten Application Servern		

Neben einer Bewertung der technischen Eigenschaften können ausgewählte Aspekte auch im Rahmen praktischer Tests erprobt werden. Um den Aufwand für solche Tests in Grenzen zu halten sollten diese erst nach einer Vorauswahl auf der Basis einiger weniger Produkte durchgeführt werden.

Tabelle 4: Multifaktorenanalyse auf der Basis wirtschaftliche Kriterien

KRITERIEN	ERLÄUTERUNG	GEWICHT	BEWERTUNG
Referenzprojekte	Können Referenzen zum erfolgreichen Einsatz des Produktes angegeben werden?		
Open Source	Besteht die Möglichkeit eines Zugriffs auf den Quellcode des Produktes, so dass ggf. eine eigenständige Wartung vorgenommen werden kann?		
Schulung	Welche Formen der Schulung werden für das Produkt angeboten? - Computer based Training - Webbasierte Schulungsinhalte - Art und Umfang von Trainingsunterlagen - Reputation der Schulungszentren		
Support	Existiert ein Online-Support der potentielle Fragen zum Produkt beantworten kann? Wie schnell können technische Fragen zum Produkt beantwortet werden? Wie sieht die Wartung/Pflege des Produktes aus, d.h. wie schnell werden Fehler behoben oder aber neu benötigte Funktionen zur Verfügung gestellt?		
Testlizenzen	Besteht die Möglichkeit das Produkt im Rahmen einer kostenfreien Teststellung hinsichtlich seiner Funktionalitäten zu verifizieren?		

KRITERIEN	**ERLÄUTERUNG**	**GEWICHT**	**BEWERTUNG**
Position des Anbieters zu Standards	Beteiligt sich der Anbieter an entsprechenden Aktivitäten zur Entwicklung von Standards im Umfeld des Produktes?		
Partnerschaften und Kooperationen	Mit welches Anbietern arbeitet der Hersteller des Produktes zusammen		
Wirtschaftliche Situation des Anbieters	Welche wirtschaftliche Position besitzt das Unternehmen am Markt? - Umsatzzahlen der vergangenen 2 Jahre - Stand der Aktie im Falle einer Aktiengesellschaft		
Abhängigkeit vom Anbieter	Wie hoch ist die schon vorhandene Abhängigkeit von den in Frage kommenden Anbietern? Hier sollte sowohl eine Bewertung der Vorteile aber auch potentiellen Nachteile durchgeführt werden.		
Roadmap	Existiert eine Roadmap zum Produkt, die aufzeigt, wann bestimmte Versionen verfügbar sind und welche neuen Funktionen damit verbunden sind.		

Neben der eigenen Bewertung von Markt-Produkten können auch die Einschätzungen entsprechender Analysten, wie z.B. der Gartner- oder der Meta-Group, herangezogen werden. Die dort durchgeführten Einschätzungen orientieren sich vordergründig an den wirtschaftlichen Kriterien, technische Details werden häufig nur im Sinne einer Standard-Konformität betrachtet.