Otto-von-Guericke Universität Magdeburg

Fakultät für Informatik
Institut für verteilte Systeme
Arbeitsgruppe Softwaretechnik

Diplomarbeit

Bedeutung und Qualitätseigenschaften des Enterprise Service Bus im Kontext von serviceorientierten Architekturen

Betreuer: **Prof. Dr.-Ing. habil. Reiner R. Dumke**
Otto-von-Guerike Universität Magdeburg, FIN, IVS

Dipl.-Inf. Dmytro Rud
Otto-von-Guerike Universität Magdeburg, FIN, IVS

Verfasser: **Steffen Hiekel**
E-Mail: Steffen.Hiekel@Student.Uni-Magdeburg.de

Magdeburg, den 21. Januar 2007

Inhaltsverzeichnis

Abkürzungsverzeichnis

(UN/)EDIFACT	(United Nations/) Electronic Data Interchange For Administration
ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
B2B	Business-to-Business
BAM	Business Activity Monitoring
BC	Binding Component
BI	Business Intelligence
BIOS	Basic Input Output System
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPMN	Business Process Modeling Notation
CASE	Computer-Aided Software Engineering
CBR	Content Based Routing
CBSE	Component-Based Software Engineering
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
CSV	Comma Separated Values
cXML	Commerce XML
DCOM	Distributed Component Object Model
DIN	Deutsches Institut für Normung e. V.
DoS	Denial of Service
DPF	Distributed Processing Framework
DTD	Document Type Definition
EAI	Enterprise Application Integration
ebXML	Electronic Business XML
EDA	Event Driven Architecture
EDI	Electronic Data Interchange
EI	Enterprise Integration
EJB	Enterprise JavaBeans
EN	Euronorm
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
ESP	Event Stream Processing
FCM-Modell	Factor=Criteria=Metrics=Modell
FTP	File Transfer Protocol

GUI ..... Graphical User Interface
HTTP ..... Hypertext Transfer Protocol
HTTPS ..... Hypertext Transfer Protocol Secure
IDE ..... Integrated Development Environment
IDL ..... Interface Definition Language
IEC ..... International Electrotechnical Commission
IIOP ..... Internet Inter ORB Protocol
ISO ..... International Organization for Standardization
IT ..... Informationstechnologie
J2EE ..... Java 2 Enterprise Edition
JAAS ..... Java Authentication and Authorization Service
JBI ..... Java Business Integration
JCA ..... J2EE Connector Architecture
JCP ..... Java Community Process
JDBC ..... Java Database Connectivitiy
JMS ..... Java Message Service
JMX ..... Java Management Extensions
JNDI ..... Java Naming and Directory Interface
JRE ..... Java Runtime Environment
JSR ..... Java Specification Request
JSSE ..... Java Secure Socket Extension
JVM ..... Java Virtual Machine
LDAP ..... Lightweight Directory Access Protocol
MAC ..... Message Authentication Code
MFL ..... Message Format Laguage
MOM ..... Message Oriented Middleware
NMR ..... Normalized Message Router
OASIS ..... Organization for the Advancement of Structured Information Standards
OLE ..... Object Linking and Embedding
OMG ..... Object Management Group
ORB ..... Object Request Broker
QoP ..... Quality of Protection
QoS ..... Quality of Service
RELAX NG ..... REgular LAnguage for XML Next Generation
RMI ..... Remote Method Invocation
RPC ..... Remote Procedure Call
SAML ..... Security Assertion Markup Language
SCM ..... Supply Chain Management
SE ..... Service Engine
SGML ..... Standard Generalized Markup Language
SLA ..... Service Level Aggreements
SMTP ..... Simple Mail Transfer Protocol
SNMP ..... Simple Network Management Protocol
SOAP ..... Simple Object Access Protocol
SPI ..... Service Provider Interface
SSL ..... Secure Sockets Layer
TLS ..... Transport Layer Security
UBL ..... Universal Business Language
UDDI ..... Universal Description Discovery and Integration
UML ..... Unified Modelling Language

W3C ..... World Wide Web Consortium
WCF ..... Windows Communication Foundation
WS ..... Web Services
WS-BPEL ..... Web Services Business Process Execution Language
WSDL ..... Web Services Description Language
WSFL ..... Web Services Flow Language
WSIL ..... Web Service Inspection Language
WSLA ..... Web Service Level Agreement
XACML ..... eXtensible Access Control Markup Language
xCBL ..... XML Common Business Library
XML ..... Extensible Markup Language
XMPP ..... Extensible Messaging and Presence Protocol
XPath ..... XML Path Language
XQuery ..... XML Query Language
XSL ..... Extensible Stylesheet Language
XSLT ..... XSL Transformations

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren hat sich das Konzept der serviceorientierten Architektur (SOA) zu dem entscheidenden Ziel für moderne Geschäftsanwendungen entwickelt und glaubt man verschiedenen Softwareherstellern, so ist eine SOA bereits heute realisierbar. Den Kern einer solchen Architektur sollen Web Services und der so genannte Enterprise Service Bus (ESB) bilden. Hat sich die Web Service Technologie bereits über mehrere Jahre hinweg entwickelt und etabliert, so steckt der ESB hingegen noch in den Kinderschuhen. Dennoch sehen Analysten aber auch die Industrie selbst den ESB als zukünftige Schlüsseltechnologie für den Aufbau einer SOA.

Mittlerweile hat sich ein umfangreicher Markt an ESB-Produkten entwickelt, die alle gegeneinander konkurrieren. Gepriesen wird – neben seinem im Vergleich zu traditionellen Integrations-Frameworks[1] geringeren Preis – vor allem seine Fähigkeit, Schwachstellen bisheriger Integrationsansätze zu überwinden. Für Kunden oder potenzielle Käufer stellt sich die Frage, was ihnen ein ESB bietet bzw. leisten kann und in wieweit verschiedene ESB-Produkte in Hinblick auf ihre Qualitätseigenschaften miteinander verglichen werden können.

Im Rahmen dieser Arbeit soll daher geklärt werden, was genau unter einem ESB zu verstehen ist und welche möglichen Unterschiede dabei zwischen Forschung und Industrie bestehen. Darüber hinaus sollen in der Softwareindustrie eingesetzte ESB-Implementierungen auf ihre Qualitätseigenschaften hin untersucht werden. Basierend darauf ist ein Modell zur Beurteilung von Qualitätseigenschaften für ESB zu entwickeln und dieses für frei verfügbare ESB-Implementierungen anzuwenden.

[1] Framework bezeichnet eine Rahmenstruktur bzw. ein Gerüst, das als Grundlage für weitere Entwicklungen dient.

1.2 Aufbau

Der Aufbau dieser Arbeit gliedert sich in mehrere Kapitel, wobei dieses Kapitel dazu dient, das Thema zu motivieren und im Weiteren eine grundlegende Einordnung zu liefern. Im Zentrum stehen dabei die SOA sowie dieser vorausgegangene bzw. sich anlehnende, aktuelle Konzepte. Damit sollen Entwicklungen und Parallelen aufgezeigt werden, die helfen, das Konzept des ESB zu verstehen und gleichfalls eine Beschreibung seines Umfeldes liefern. Im zweiten Kapitel geht es um den Begriff ESB und was darunter zu verstehen ist. Dabei interessieren vor allem die Ansichten aus Kreisen der Forschung und damit verbundene Begriffsdefinitionen. Von besonderer Bedeutung sind in diesem Zusammenhang die Fragen nach Aufbau, Eigenschaften und Funktionalitäten eines ESB. Daran anlehnend erfolgt im nächsten Kapitel eine umfangreiche und detaillierte Betrachtung von angebotenen ESB-Produkten. Hierbei soll gezeigt werden, was seitens der Softwareindustrie unter dem Begriff ESB verstanden wird und welche Technologien konkret zum Einsatz kommen. Gleichfalls herauszuarbeiten sind Parallelen sowie mögliche Differenzen zwischen den Sichtweisen in der Forschung und in der Industrie. Das folgende vierte Kapitel konzentriert sich auf die Betrachtung der Qualitätseigenschaften eines ESB. Im Rahmen eines zu erarbeitenden ESB-Qualitätsmodells werden hier allgemeine wie spezielle Qualitätseigenschaften aufgestellt und Kriterien zu deren Bewertung beschrieben. Die Anwendung des ESB-Qualitätsmodells auf frei verfügbare ESB-Implementierungen erfolgt im fünften Kapitel und endet mit einem abschließenden Vergleich. Den Schluss dieser Arbeit bildet eine zusammenfassende Betrachtung der gewonnenen Erkenntnisse sowie ein Ausblick auf zukünftige Entwicklungen und Themenstellungen im Kontext des ESB.

1.3 Einordnung

Im Rahmen einer ersten Einordnung erfolgt an dieser Stelle eine kurze Einführung in das Umfeld heutiger Geschäftsanwendungen und mit ihr verbundener Technologien wie Konzepte.

Rückblickend betrachtet, kann vor allem der Schritt hin zur objektorientierten Programmierung als ein Meilenstein angesehen werden und soll als Ausgangspunkt für die weiteren Betrachtungen dienen. Nicht nur, dass die objektorientierte Programmierung andere Paradigmen vor allem die funktionsorientierte Programmierung weitestgehend abgelöst hat, sie führte auch im Rahmen des Softwareengineerings zu neuen, objektorientierten Konzepten (bspw. der UML[2]). Die wesentlichen Eigenschaften, die für eine objektorientierte Programmierung sprechen und letztlich zu ihrem Durchbruch führten, sind (vgl. [Hei03], S. 8; [Som01], S. 270):

- bessere Beherrschbarkeit von komplexen Softwaresystemen durch Abstraktion, Kapselung und Vererbung,
- höhere Wiederverwendung,
- Vermeidung von redundantem Programmcode (bspw. durch Vererbung),
- daraus resultierend eine bessere Wartbarkeit.

[2] UML – Unified Modelling Language

Charakteristisch für dieses Paradigma ist die Zerlegung der Software in Objekte[3], die über Schnittstellen (Methoden genannt) miteinander kommunizieren. Ein Großteil dessen, was für die Entwicklung moderner Geschäftsanwendungen notwendig ist, steht damit zur Verfügung. Dennoch machte die Entwicklung nicht halt, was nicht zuletzt auf die Defizite des einfachen objektorientierten Paradigmas zurückzuführen ist.

Ein wesentlicher Kritikpunkt ist die Wiederverwendbarkeit, die nicht in dem Maße erreicht wurde, wie man sich das ursprünglich vorgestellt hat. Ursache dafür ist die hohe Granularität der Objekte, die eine Wiederverwendung auf Objektebene erschwert. Darüber hinaus haben sich verschiedene objektorientierte Programmiersprachen etabliert. Kann man auf Objekte ein und derselben Programmiersprache problemlos zugreifen, so ist dies für Objekte verschiedener Programmiersprachen oder unterschiedlicher Architekturen nicht ohne weiteres möglich. Middleware, die zur Behebung dieser Defizite entwickelt wurde wie etwa die Common Object Request Broker Architecture (CORBA), führte zu Verbesserungen und einer gewissen Programmiersprachenunabhängigkeit, änderte aber nichts an der Objektgranularität und den damit verbundenen Problemen (vgl. [Som01], S. 319; [SP01], S. 42 f.).

Ein Konzept, das bessere Wiederverwendungseigenschaften anstrebt, ist die komponentenbasierte Softwareentwicklung[4]. Was im Falle der objektorientierten Softwareentwicklung das Objekt, ist hier die Komponente. Der Begriff der Komponente ist jedoch im Gegensatz zu dem des Objekts wesentlich vielschichtiger. Es lässt sich bspw. bei Siedersleben eine Unterteilung in fachliche[5] und technische Komponenten[6] finden. Definiert werden kann der Begriff Komponente, als eine unabhängige, ausführbare Entität, die über wohldefinierte Schnittstellen eine bestimmte Funktionalität bzw. Dienst anbietet (vgl. [Som01], S. 320). Weiterhin bestehen Komponenten in der Regel aus kleineren Entitäten, wie etwa Objekten oder Modulen und haben daher typischerweise den Charakter von Aggregaten (vgl. [Som01], S. 321; [Sie03], S. 100). Dies unterstreicht ebenso die Definition nach Szyperski, der eine Softwarekomponente als eine Komposition mit vertraglich vereinbarten Schnittstellen ansieht, die unabhängig eingesetzt werden kann und Gegenstand weiterer Kompositionen durch Dritte ist (vgl. [SGM02], S. 41)[7]. Diese Eigenschaft ist es auch, die das Problem der feingranularen Objektstrukturen und der damit verbundenen Wiederverwendungsdefizite behebt. Ein weiterer Unterschied zur objektorientierten Programmierung liegt darin, dass der Fokus bei Komponenten weniger auf dem Nachempfinden realweltlicher Objekte liegt, als vielmehr auf dem Schaffen von Softwarebausteinen bestimmter Funktionalität. Die Anforderungen, die dabei an Komponenten gestellt werden, sind (vgl. [GT00], S. 10 ff.; [Dum02]; [Dum03], S. 351):

- wohldefinierter Zweck,
- Kontextunabhängigkeit,
- Portabilität,
- Unabhängigkeit von der Umgebung (Hard- und Software),
- Ortstransparenz,
- Trennung von Schnittstelle und Implementierung,

[3] Ein Objekt kann definiert werden als eine Entität, die einen Zustand (Menge von Attributen), ein Verhalten (Zustandsänderungen, d. h. Änderungen an den Attributen) und eine Identität besitzt (vgl. [Som01], S. 271; [SP01], S. 35 f.).

[4] Auch als Component-Based Software Engineering (CBSE) bezeichnet.

[5] Komponenten, die Dienste betrieblicher Anwendungsdomänen implementieren (vgl. [Sie03], S. 99 f.).

[6] Komponenten, die Dienste technischer Anwendungsdomänen implementieren und in der Regel zur Unterstützung von Fachkomponenten dienen (vgl. [Sie03], S. 99 f.).

[7] Noch weitere bei Szyperski zu findende Definitionen anderer Autoren sind von gleichbedeutendem Inhalt (vgl. [SGM02], S. 195 ff.).

- Selbstbeschreibungsfähigkeit,
- problemlose sofortige Nutzbarkeit,
- Integrations- und Kompositionsfähigkeit,
- Wiederverwendbarkeit,
- Konfigurierbarkeit/Anpassbarkeit,
- Austauschbarkeit,
- Bewährtheit,
- Vermarktbarkeit.

Mit diesen Zielen verband sich die Vision, dass Software zukünftig aus bereits vorhandenen Komponenten zusammengesetzt, quasi komponiert wird und diese auf eigenen Komponentenmarktplätzen gehandelt werden. Dank der geforderten Architektur- bzw. Implementierungsunabhängigkeit wären Komponenten einfach integrierbar und gegeneinander austauschbar.

Wenngleich die komponentenbasierte Softwareentwicklung einige Wiederverwendungsdefizite beheben und sich verschiedenste Komponenten-Frameworks wie Component Object Model (COM), Enterprise JavaBeans (EJB) und .NET durchsetzen konnten, so wurden dennoch wesentliche Ziele und Visionen nicht oder nur teilweise erreicht. Ein Problem dabei waren die bereits angesprochenen Komponenten-Frameworks selbst, die einen Framework-übergreifenden Komponenteneinsatz kaum unterstützten. Komponenten eines Frameworks konnten nicht problemlos in ein anderes Framework integriert werden und waren somit nicht, wie ursprünglich gefordert, umgebungsunabhängig, austauschbar und problemlos nutzbar. Dies führte dazu, dass sich kaum Komponentenmarktplätze bildeten oder gar etablierten[8]. Eine Vermarktung von Komponenten fand in der Regel nicht statt. Ein weiteres Defizit der komponentenbasierten Softwareentwicklung war die fehlende Integrationsmöglichkeit von Altanwendungen, die ein aufwändiges Software Reengineering (Reverse Engineering und Reimplementierung) erfordert hätte. Dies führte in der Summe dazu, dass auch die komponentenbasierte Softwareentwicklung nicht das Maß an Wiederverwendbarkeit erreichte, wie einst prophezeit.

Einen anderen Ansatz verfolgt die Enterprise Application Integration (EAI), die folgende Kernziele benennt:

- anwendungsübergreifende Integration,
- Integration von Altanwendungen,
- Geschäftsprozessorientierung der Unternehmens-IT[9].

So vielfältig wie das Feld der EAI-Produkte und EAI-Anbieter, ist auch das der Definitionen dieses Begriffs. Die folgenden beiden Definitionen sollen dies veranschaulichen.

> *„EAI-Produkte sind Software, die es erlaubt, die Anwendungen eines Unternehmens zu integrieren. Der Begriff wurde geprägt durch das Bemühen, viele Anwendungen, die nicht für eine Zusammenarbeit entworfen wurden und auch nur*

[8] Derartige Komponentenmarktplätze sind bspw. ComponentSource (`www.componentsource.com`) oder Fashline (`www.flashline.com`) (vgl. [SGM02]).
[9] IT – Informationstechnologie

> *Teilaufgaben von Geschäftsprozessen abdecken, dazu zu bringen, in einheitlichen Geschäftsprozessen zusammenzuspielen. Es geht also darum, heterogene Anwendungen eines Unternehmens so zu integrieren, dass sie sich möglichst so verhalten, als wären sie von Anfang an dafür entworfen worden, die aktuellen Geschäftsprozesse eines Unternehmens zu unterstützen.“*, [Kel02].

> *„EAI ist kein Produkt, sondern ein Ansatz zur Entwicklung integrierter Anwendungssysteme. Dieser Ansatz ist eine Kombination aus Architektur, Technologie und Methodik. Er ist geprägt durch die Verfügbarkeit von Softwareprodukten, die vorgefertigte Integrationslösungen bereitstellen. EAI fokussiert sich auf die Integration von Geschäftsprozessen und Daten, während traditionelle Methoden oft nur datenorientiert sind. EAI beinhaltet die Ziele der Wiederverwendung sowie der Verteilung von Prozessen und Daten. EAI ermöglicht es Nutzern, Anwendungen auch nahezu ohne Kenntnis der technischen Details zu integrieren.“*, [Kai02].

EAI ist folglich kein Produkt, sondern vielmehr ein Konzept, das softwaregestützt umgesetzt werden kann. EAI-Software/-Frameworks sollen dabei helfen, verschiedenste Anwendungen zu verknüpfen und auf diesem Wege das Problem der Architektur- und Implementierungsabhängigkeit zu lösen. Darüber hinaus wird erstmals der Gedanke der (Geschäfts-)Prozessorientierung in den Aufbau der Unternehmens-IT integriert. Weiterhin sollten u. a. noch vorhandene Offline-Schnittstellen, Redundanzen im Bereich von Daten und Funktionalitäten sowie bestehende Wiederverwendungsdefizite endgültig beseitigt werden (vgl. [Sch05a], S. 11).

Bei näherer Betrachtung von EAI stellt sich die Frage, warum sich daneben noch ein anderes Konzept nämlich SOA etablieren konnte und heute droht, EAI nahezu vollständig zu verdrängen. Sind mit EAI nicht sämtliche Integrations- und Wiederverwendungsprobleme gelöst? Leider nein. Trotz des enormen Integrationspotenzials, das sich mit EAI verbindet, ist die Umsetzung dieses Konzepts problembehaftet (vgl. [Cha05a], S. 17; [Sch05a], S. 3):

- EAI-Projekte sind sehr umfangreich und benötigen damit enorme Budgets.
- EAI hat eine zu grobe Granularität für eine effektive Wiederverwendung.
- EAI-Software/-Frameworks sind sehr teuer und basieren im Wesentlichen auf eigenen, proprietären Standards.
- Die Realisierung von EAI mittels monolithisch aufgebauter und in Form einer Hub-and-Spoke Topologie[10] angeordneter Integrationsserver kann sich mit Blick auf Skalierungsproblematiken als Flaschenhals für steigende Leistungsanforderungen erweisen.

Wurden in Unternehmen EAI Produkte verschiedener Anbieter verwendet, führte dies aufgrund proprietärer Standards und Protokolle in der Regel zu Integrationsinseln, d. h. Gruppen aus jeweils miteinander integrierten Anwendungen und im schlimmsten Fall zu einer Vielzahl von Punkt-zu-Punkt Verbindungen[11]. Die Beschränkung auf einen einzelnen Anbieter wiederum führte zu Abhängigkeiten und einer fehlenden Flexibilität. Darüber hinaus

[10] Der Begriff Hub-and-Spoke Topologie beschreibt eine Netzwerktopologie bestehend aus Nabe und Speiche. Mehrere Knoten eines Netzwerks (Spoke/Speiche) sind dabei mit einem zentralen Knoten verbunden (Hub/Nabe). Sie entspricht vom Aufbau her einer Sterntopologie.

[11] Van Huizen und andere sprechen in diesem Zusammenhang auch von einer Architektur des Zufalls (Accidental Architecture), die geprägt ist von Eigenentwicklungen, eingekauften wie übernommenen Anwendungen und Systemen sowie einzelnen Integrationsprojekten (vgl. [Cha04b], S. 28 ff.; [Hui03], S. 28).

erfordern proprietäre Standards Spezialwissen, welches oftmals teuer erworben oder eingekauft werden muss. Abgesehen von den konzeptionellen Zielstellungen von EAI liegt es gerade auch an diesen Defiziten, dass sich EAI für eine unternehmensübergreifende Integration kaum eignet und typischerweise mit sehr hohen Kosten verbunden ist (vgl. [Cha04a], S. 6; [PT05], S. 59 ff.).

Als logisch nächster Schritt aller vorangegangenen Konzepte entstand in den letzten Jahren die serviceorientierte Architektur. Die Überschneidungen, die dabei mit den bereits genannten Konzepten allen voran der komponentenbasierten Softwareentwicklung bestehen, zeigen sich in den folgenden Definitionen:

> *„Service-orientierte Architekturen, kurz SOA, sind das abstrakte Konzept einer Software-Architektur, in deren Zentrum das Anbieten, Suchen und Nutzen von so genannten Diensten*[12] *über ein Netzwerk steht. Dabei werden Dienste fast ausschließlich von Applikationen oder anderen Diensten genutzt. Hierfür ist es unerheblich, welche Hard- oder Software, Programmiersprache oder Betriebssystem die einzelnen Beteiligten verwenden. Unter anderem durch einheitliche Standards sollte es möglich sein, Dienste durch entsprechende Suchfunktionen zu finden, die von einem Anbieter genauso einfach publiziert werden können. [. . .] Im Idealfall ist sogar eine einfache Integration ganzer Anwendungen möglich.“*, [DJMZ05], S. 7.

> *„Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“*, [DJMZ05], S. 11.

> *„Grundlegendes Prinzip einer SOA ist es, Funktionalität als modulare und wiederverwendbare Services zur Verfügung zu stellen. Neue Anwendungen können, dann aus bereits existierenden Services „zusammengesetzt“ werden. Man spricht dabei auch von einer losen Kopplung, da es keine starken logischen oder physischen Abhängigkeiten gibt, und zwar weder zwischen den Services untereinander, noch zwischen den Services und den Anwendungen, in denen sie genutzt werden. Somit ist es auch leicht möglich, laufende Anwendungen durch Austausch einzelner Services zu modifizieren, zu erweitern oder zu optimieren.“*, [Sch05a], S. 8.

Wie in den Definitionen bereits zum Teil schon enthalten, verbinden sich mit einer SOA Zielstellungen wie (vgl. [NL04], S. 54; [PT05], S. 51 f.; [Sch05a], S. 10 f.):

- höhere Wiederverwendung,
- Vermeidung von redundanten Daten und Funktionalitäten,
- Unabhängigkeit von der Umgebung (Hard- und Software),
- Integration von Anwendungen und Diensten,
- lose Kopplung,

[12] Die Begriffe Dienst und Service werden in der Literatur synonym verwendet und sind daher gleichzusetzen.

- einheitliche Standards,
- Vermarktbarkeit von Services,
- an Geschäftsprozessen und deren Änderungen orientierter Aufbau.

Bemerkenswert ist, dass diese Ziele nicht neu sind, sondern schon seit langem existieren. Dennoch sind sie hoch aktuell, was nicht zuletzt auf Defizite der bisherigen Konzepte zurückzuführen ist.

Insgesamt soll SOA zu einer technischen Infrastruktur führen, die es ermöglicht, sich schnell neuen Geschäftsanforderungen anzupassen. Egal ob es sich um Individual- oder Standardsoftware handelt, alles soll sich miteinander integrieren lassen und so Integrationsinseln und Anbieterabhängigkeiten endgültig beseitigen. Nicht zuletzt besteht die Vision von kommerziellen Diensten, die flexibel konsumiert und gegeneinander ausgetauscht werden können. Das Resultat ist eine schon lange angestrebte kosteneffiziente Unternehmens-IT (vgl. [NL04], S. 54).

Neben den bereits genannten Konzepten, bestand schon mit der Entwicklung erster Netzwerke die Notwendigkeit, Programme auch verteilt ablaufen zu lassen. Dies war die Geburtsstunde der Middleware, das heißt Software, die bildlich gesprochen in der Mitte zwischen zwei typischerweise verteilten Softwarekomponenten steht. Sie wird oft auch als glue[13] (Kleber/Kleister) bezeichnet, da sie ähnlich wie dieser Anwendungen verbindet. Als eine erste Middleware entstand der Remote Procedure Call (RPC), der sich, wie sein Name schon sagt, noch stark an der prozeduralen Programmierung orientiert. Mit der objektorientierten Softwareentwicklung folgten auch auf dem Gebiet der Middleware neue Konzepte. Es entstand die objektorientierte Middleware. Vertreter dafür sind CORBA und die Remote Method Invocation (RMI). Eine andere Herangehensweise verfolgt die nachrichtenorientierte Middleware (MOM[14]), die aufgrund ihrer Bedeutung im Kontext von EAI und SOA etwas ausführlicher beschrieben werden soll.

MOM unterscheidet sich von den bereits genannten Middleware-Ansätzen neben einer auf Nachrichten aufgebauten Kommunikation vor allem durch die Fähigkeiten, Nachrichten speichern und verteilen zu können. Die Speicherung von Nachrichten lässt zu, dass Sender und Empfänger nicht notwendigerweise zum selben Zeitpunkt mit dem Netzwerk verbunden sein müssen, d. h. es wird eine asynchrone Kommunikation unterstützt. Mittels Routing[15] können dabei Verbindungen zu einem (unicast) aber auch mehreren (multi- bzw. broadcast) Empfängern aufgebaut werden. Folgende Kommunikationsprotokolle werden von MOM unterstützt (vgl. [Cha04b], 84 ff.):

- Message Passing (Point-to-Point) – direkte Kommunikation zwischen zwei Endpunkten,
- Message Queueing (Store and Foreward) – indirekte Kommunikation über eine Nachrichtenwarteschlange (Queue),
- Publish-and-Subscribe[16] – der Erzeuger (Publisher) stellt (typischerweise mehreren) Konsumenten (Subscriber) Nachrichten über Topics zur Verfügung.

[13] Zu finden bei Schmietendorf ([Sch05b], S. 8), Chappell ([Cha04b], S. 36) u. a.
[14] MOM – Message Oriented Middleware
[15] Unter Routing wird die Wegewahl von Nachrichtenströmen zur Nachrichtenübermittlung verstanden.
[16] Eine Begriffsunterscheidung, die in diesem Zusammenhang vorgenommen wird, betrifft die Nachrichtenwarteschlangen. Diese werden bei unicast verschickten Nachrichten als Queues und bei mulit- oder broadcast verschickten Nachrichten als Topics bezeichnet.

Ein wesentlicher Vorteil, der sich aus diesen Eigenschaften ergibt, ist die Möglichkeit zu einer asynchronen, lose gekoppelten und durch entsprechende Mechanismen dennoch zuverlässigen Kommunikation. Damit verbunden ist eine parallele Verarbeitung und dadurch eine verbesserte Systemverfügbarkeit (vgl. [Cha04b], S. 5, S. 84 ff.).

Einen weiteren Ansatz stellt komponentenbasierte Middleware dar, allen voran die auf der Java 2 Enterprise Edition (J2EE) fußenden EJB. Diese orientiert sich einerseits am Komponentengedanken, übernimmt aber andererseits auch Fähigkeiten nachrichtenbasierter Konzepte. Ihre Bedeutung liegt vor allem im Bereich professioneller[17] Unternehmensanwendungen. Sehr gut unterstützt wird dabei vor allem der Aufbau verteilter, weniger jedoch die Integration bereits bestehender Anwendungen.

Die in dieser Kette wohl jüngste Middleware-Technologie stellen Web Services (WS) dar. Diese basieren im Kern auf folgenden Standards (vgl. [NL04], S. 20 ff.):

- Simple Object Access Protocol (SOAP),
- Web Services Description Language (WSDL),
- Universal Description Discovery and Integration (UDDI).

Die Aufgaben gliedern sich wie folgt. SOAP, ein auf der Extensible Markup Language (XML) und dem Hypertext Transfer Protocol (HTTP) bzw. Simple Mail Transfer Protocol (SMTP) basierendes Protokoll, dient der Übertragung von Nachrichten[18]. Die Schnittstellen zum Senden und Empfangen von Nachrichten werden dabei mittels WSDL (ebenfalls XML-basiert) beschrieben. Dritter wesentlicher Baustein ist ein Verzeichnisdienst zum Eintragen und Finden von Diensten, der so genannte UDDI. Gestützt auf diese Technologien, ermöglichen Web Services die Realisierung einer (vgl. [NL04], S. 20 ff.; [DJMZ05], S. 27 ff.):

- standardbasierten,
- herstellerunabhängigen,
- in Bezug auf Firewalls problemarmen,
- offenen,
- flexiblen Middleware.

Es ist angesichts dieser Vorteile wenig verwunderlich, dass sich Web Services vor allem auf dem Gebiet der Enterprise Integration (EI) zur bevorzugten Middleware entwickeln. Sie ermöglicht eine standardbasierte Integration von Anwendungen und Diensten unterschiedlichster Granularität. Eine unternehmensübergreifende Integration erscheint damit realisierbar. Nicht zuletzt ist es diese Technologie, die von Experten genannt wird, wenn es darum geht, eine SOA zu implementieren.

[17] Unter professionell sind hier höchste Anforderungen an Performanz, Sicherheit und Zuverlässigkeit zu verstehen.

[18] Es ist anzumerken, dass SOAP nicht zwangsläufig an HTTP oder SMTP gebunden ist und deshalb, wie später gezeigt werden wird, auch andere Protokolle als Grundlage dienen können. Jedoch geht eine derartige Verwendung von SOAP über den Kontext von Web Services hinaus und wird aus diesem Grund hier nicht weiter thematisiert.

1.4 Fazit

Eine historische Betrachtung zeigt, dass sich das Konzept der SOA auf verschiedene, vorausgegangene Konzepte stützt und deren logische Weiterentwicklung darstellt. Einige wesentliche Zielstellungen dieser Konzepte konnten jedoch nicht in dem Maße erreicht werden, wie einst vorgestellt. Nun ist es SOA, die diese jahrelang verfolgten Zielstellungen erbt und endlich ganzheitlich umsetzen soll. Der ESB wird in diesem Kontext als das technologische Fundament und zentrale Element einer SOA angesehen. Er ist es also, der das abstrakte Konzept der SOA und damit verbundene Zielstellungen wie Integration unterschiedlichster Anwendungen und Services, Realisierung einer Umgebungsunabhängigkeit (Unabhängigkeit von Hard- und Software), Verknüpfung mittels loser Kopplung und nicht zuletzt eine tatsächliche Wiederverwendung bestehender Software für die Praxis anwendbar machen soll. Ein weiterer wesentlicher Faktor für eine derartige Umsetzung stellen Web Services dar, die sich vor allem in den letzten Jahren rasant verbreitet haben. Im folgenden Kapitel soll gezeigt werden, was genau ein ESB ist bzw. nicht ist, welche Konzepte hinter dem ESB stehen und wie er in Bezug auf bereits existierende Technologien allen voran Web Services eingeordnet werden kann. Weitere Fragestellungen, die im Rahmen dieser Arbeit thematisiert und beantwortet werden sollen, betreffen die Umsetzung des ESB in der Praxis, damit verbundene Technologien, seine Qualitätseigenschaften und den Grad der erfüllten Zielstellungen.

Kapitel 2

Der ESB aus Sicht der Forschung

2.1 Herangehensweise

Fast in jedem Zusammenhang wird heutzutage bei IT-Experten von SOA gesprochen und dies mancher Orten sogar als eine Revolution auf dem Gebiet der EI angesehen. Gleich im nächsten Atemzug fällt dann der Begriff ESB und es heißt, dieser sei der integrale Bestandteil beim Aufbau einer SOA. Weit weniger wortgewaltig drücken sich besagte Experten aus, wenn sie gezwungen werden, den Begriff ESB genau zu definieren oder die einfache Frage, was am ESB neu sei, zu beantworten.

Zur exakten Klärung des Begriffes ESB bedarf es einheitlicher Kriterien. Erst diese erlauben es, unterschiedliche Definitionen miteinander zu vergleichen und zu bewerten. Folgende Kriterien sollen beim Vergleich der anschließenden Definitionen berücksichtigt werden:

- Was ist ein ESB – ein Konzept, ein Softwareprodukt, eine Architektur oder ein Entwurfsmuster?
- Welchem Zweck dient der ESB?
- Was sind die Aufgaben bzw. Eigenschaften des ESB – wodurch wird er charakterisiert?
- Wie lässt sich ein ESB realisieren?
- Wo kommt der ESB zum Einsatz – innerhalb eines Unternehmens oder auch darüber hinaus?
- Welche Rolle spielt der ESB im Kontext von EAI und SOA und wie ist seine Umsetzung in Bezug auf Technologien wie MOM und Integrationsbroker zu sehen?

In einer ersten Annäherung soll der Begriff ESB seiner wortwörtlichen Übersetzung nach charakterisiert werden.

- **Enterprise:** ein Unternehmen, Betrieb oder Firma
- **Service:** ein Dienst
- **Bus:** ein Datenbus

Demnach geht es also darum, mittels eines Datenbusses Dienste für ein Unternehmen zur Verfügung zu stellen. Da dies in Bezug auf die zuvor genannten Kriterien nur wenig Aufschluss bringt, soll an dieser Stelle noch eine andere einführende Definition geliefert werden. Als eine solche kann die nach WIKIPEDIA dienen.

> *„In der Informationstechnologie bezieht sich der Begriff Enterprise Service Bus (ESB) auf eine Softwarearchitektur, die mittels standardbasierter Technologien aus dem Bereich der Middleware Infrastruktur implementiert werden kann und so grundlegende Dienste für komplexere Architekturen über eine ereignisgesteuerte und standardbasierte Nachrichten-Engine*[1] *(dem Bus) zur Verfügung stellt.*
>
> *Ein ESB liefert im Allgemeinen eine Abstraktionsschicht über die Implementierung eines Enterprise Messaging Systems, die es Integrationsspezialisten erlaubt, den Wert eines nachrichtenbasierten Systems zu nutzen ohne Programmcode schreiben zu müssen. Im Gegensatz zum eher klassischen Ansatz der Enterprise Application Integration und einem monolithischen Stack*[2] *innerhalb einer hub-and-spoke Architektur, ist das Fundament eines Enterprise Service Bus aus grundlegenden, in ihre einzelnen Bestandteile zergliederten Funktionen aufgebaut, die soweit erforderlich oder notwendig verteilt eingesetzt und im Einklang zusammenarbeiten können.*
>
> *Der ESB ist nicht die Implementierung einer serviceorientierten Architektur (SOA), verfügt aber über die Eigenschaften mittels derer sich eine solche implementieren ließe. Obwohl weithin angenommen, ist der ESB nicht Web Service-basiert. Der ESB sollte standardbasiert und flexibel sein und dabei viele Transportmedien unterstützen. Eher auf Prinzipien der Enterprise Integration als auf SOA Prinzipien basierend, versucht er die Kopplung zwischen aufgerufenem Dienst und Transportmedium zu beseitigen“*, [Wik06a].

Hierbei wird noch nichts über die Eigenschaften eines ESB gesagt. Diese werden jedoch an späterer Stelle angeführt. Zu den Wichtigsten gehören dabei (vgl. [Wik06a]):

- Nutzung von XML,
- Unterstützung von Web Services,
- Unterstützung von nachrichtenbasierter Kommunikation (synchron, asynchron, Punkt-zu-Punkt und Publish-Subscribe),
- Nutzung von Standardadaptern zur Integration von Altanwendungen,
- Unterstützung von Service Orchestrierung[3] und Service Choreographie[4],
- Unterstützung von Diensten zur Transformation von gesendeten Daten,

[1] Unter einer Engine (wörtlich Motor, Antrieb) wird der ausführende Teil einer Software (Ausführungskomponente) verstanden.

[2] Der Begriff Stack bezeichnet in diesem Zusammenhang eine über mehrere Schichten verteilte Menge von Technologien.

[3] Unter Service Orchestrierung kann die Steuerung von Diensten gemäß übergeordneter Geschäftsprozesse und Prozessmodelle verstanden werden.

[4] Der Begriff Service Choreographie überschneidet sich zum Teil mit dem der Service Orchestrierung, befasst sich jedoch weniger mit Abläufen gemäß von Prozessmodellen und dafür stärker mit der Steuerung und Koordination des notwendigen Nachrichtenflusses. Die Choreographie von Services ist daher eine wichtige Grundlage für deren Orchestrierung innerhalb von (Geschäfts-)Prozessen.

- intelligentes[5], inhaltsbezogenes[6] Routing,
- Unterstützung standardisierter Sicherheitsmodelle.

Bei Anwendung der Definitionskriterien ergibt sich folgendes Bild. Ein ESB ist als erstes eine Softwarearchitektur. Der Zweck dieser Architektur liegt darin, dass sie eine Abstraktionsschicht über ein Enterprise Messaging System bildet, die einerseits die Nutzung nachrichtenbasierter Systeme erleichtern sowie andererseits die Kopplung zwischen Dienst- und Transportmedium aufheben soll. Im Vergleich zur EAI basiert der ESB nicht auf einer Hub-and-Spoke Architektur, sondern auf grundlegenden, in ihre einzelnen Bestandteile zergliederten Funktionen. Weiterhin ist der ESB nicht die Implementierung einer SOA, wenngleich mittels diesem eine solche implementiert werden kann.

Leider unthematisiert bleibt bei dieser Definition das Einsatzfeld des ESB. Es wird keine Aussage getroffen, ob es sich um eine Architektur handelt, die nur innerhalb eines Unternehmens Anwendung findet oder auch darüber hinaus, d. h. unternehmensübergreifend. Ansonsten stellt sich diese Definition als sehr detailliert heraus und kann daher als eine erste Referenz dienen. Wie dies an anderer Stelle, in Forschung und Industrie, gesehen wird, soll im weiteren Verlauf dieses Kapitels umfassend dargelegt werden.

2.2 Definition des ESB in der Forschung

2.2.1 Definition nach Schulte (Gartner)

Die erste Erwähnung des Begriffs ESB stammt aus dem Jahre 2002. Noch im selben Jahr lieferte Schulte (Analyst bei Gartner) diesbezüglich eine erste Definition.

> *„Ein Enterprise Service Bus (ESB) ist eine neue Architektur, die Web Services, nachrichtenorientierte Middleware, intelligentes Routing und Transformation nutzt. ESBs fungieren als ein leichtgewichtiges, allseits verfügbares Integrationsgerüst, durch welches Softwaredienste und Anwendungskomponenten fließen.“*, [Far03].

Schulte sieht demnach den ESB als eine Architektur, die als ein allseits verfügbares Integrationsgerüst für Dienste wie auch Anwendungen dient. Bestandteile dieses Integrationsgerüsts sind Web Services, MOM, intelligentes Routing und Transformation. Sie ähnelt in diesen Punkten der Definition nach Wikipedia und sieht ebenfalls den ESB als eine Architektur.

In Bezug auf andere Kriterien, wie Technologien zur Umsetzung des ESB, Einsatzfeld, Rolle in Bezug auf EAI und SOA, gibt diese Definition nur wenig Aufschluss, weshalb sie an dieser Stelle nicht weiter diskutiert werden soll. Entschuldigende Ursachen für die angesprochenen Defizite mögen in dem Umstand liegen, dass es sich hierbei um die erste Definition dieses Begriffs überhaupt handelt, wie auch in der Ermangelung fehlender Umsetzungen zum damaligen Zeitpunkt.

[5] Nach Chappell können unter intelligentem Routing sämtliche auf Inhalt, Kontext, Geschäftsprozessen oder Prozessorchestrierung basierenden Formen von Routing verstanden werden (vgl. [Cha04b], S. 101). Aufgrund einer fehlenden genaueren Definition seitens Wikipedia kann ein möglicherweise abweichendes Begriffsverständnis nicht geklärt werden.

[6] Inhaltsbezogenes Routing, auch Content Based Routing (CBR) genannt, meint das Routing einer Nachricht gemäß ihrem Inhalt.

2.2.2 Definition nach Kischel (OBJEKTspektrum)

Ein erster größerer Artikel, den OBJEKTspektrum dem ESB widmete, stammt von Kischel aus dem Jahre 2003. Im Rahmen dieses Artikels gibt er die folgende Definition.

> *„Der ESB ist ein messaging-basierter Softwarebus, der die Module eines Verbundes von Anwendungen zu einer verteilten Geschäftsanwendung verbindet. Seine Aufgaben sind:*
>
> - *Verbinden der Anwendungsteile, der Services, zu einer verteilten Geschäftsanwendung,*
> - *Transformation und Zusammenstellen des Contents, den die Services austauschen sowie*
> - *Koordination des Verarbeitungsablaufs (BPM[7]).*
>
> *Zentrale Elemente eines ESB sind:*
>
> - *ein Messaging-System, das die physikalische Ebene des Busses realisiert und die Vielzahl der notwendigen Fähigkeiten, wie z. B. sichere und garantierte Übermittlung, Persistenz bei asynchronen und lose gekoppelten Services bietet und*
> - *ein Distributed Processing Framework (DPF), das als logische Ebene des Busses die Services verknüpft und die Prozesssteuerung übernimmt.*
>
> *Der ESB erfüllt diese Aufgaben durch eine Implementierung auf zwei Ebenen, die Transportschicht und die Prozessschicht.“*, [Kis03], S. 37.

Im Kern stellt der ESB nach Kischel einen nachrichtenbasierten Softwarebus dar. Zweck dieses Softwarebusses ist es, durch Verbindung einzelner Module eines Anwendungsverbundes den Aufbau einer verteilten Geschäftsanwendung zu realisieren. Zu den weiteren Aufgaben neben dem Verbinden von Anwendungen und Services zählen Transformation und Zusammenstellung von Servicedaten (Content) sowie die Koordination des Verarbeitungsablaufs. Im Unterschied zu Schulte aber auch zu Wikipedia wird der ESB nicht allgemein als eine Architektur definiert sondern konkreter als ein Message-basierter, d. h. nachrichtenbasierter Softwarebus. Betrachtet man die Aufgaben, die er nach Kischel zu erfüllen hat – zu verbinden, zu transformieren und zu koordinieren –, so lassen sich diese bereits bei Wikipedia finden. Als zentrale Elemente benennt Kischel ein Messaging-System und ein so genanntes Distributed Processing Framework (DPF) (vgl. [Kis03], S. 37 f.).

Das Messaging-System bildet in diesem Zusammenhang die Transportschicht des ESB und damit die physikalische Ebene zur Realisierung des Busses. Zum Einsatz kommt dabei auf Standards basierende MOM. Als wichtige Standards nennt Kischel Java Message Service (JMS), J2EE Connector Architecture (JCA), SOAP (sowohl SOAP über JMS als auch SOAP über HTTP) und Java Database Connectivitiy (JDBC). Neben diesen Standards besteht die Notwendigkeit, Funktionen aus bestehenden Paketen und Anwendungen heraus zu publizieren und in den ESB einzuhängen. Hierzu müssen Services definiert werden, die dann die Schnittstelle zu den jeweiligen Anwendungen bilden. Dafür ist es unabhängig, aus welchen Quellen (Java, C/C++, Microsoft COM/ActiveX) die Daten stammen. Über bereits angesprochene Adapter und Protokolle werden alle Daten in Nachrichten umgewandelt und im

[7] BPM steht für Business Process Management, zu deutsch (Geschäfts-)Prozessmanagement und umfasst u. a. die Modellierung, Gestaltung, Steuerung, Verwaltung und Verbesserung von (Geschäfts-)Prozessen.

Weiteren an dezidierte Ziele (Queues oder Topics) geschickt. Dort stehen sie der logischen Schicht des ESB zur Verfügung (vgl. [Kis03], S. 38).

Die Prozessschicht des ESB, d. h. die logische Ebene wird mittels DPF realisiert und liegt über der Transportschicht. Der ESB, genauer das DPF stellt sich hier als Vermittler und Koordinator dar, das einzelne Anwendungen und Dienste orchestriert, d. h. deren Verarbeitungsablauf, wie in Abbildung 2.1 dargestellt, dirigiert. Die über die Transportschicht in Warteschlangen (Queues) abgelegten Nachrichten müssen dabei vom DPF an andere Services weitergereicht werden. Grundlage dafür sind Prozessketten, die zuvor mittels entsprechender Werkzeuge (Prozess-Manager oder BPM-Tools) definiert wurden. Weitere Bestandteile dieser Prozessketten sind die beteiligten Services, die Eintrittsendpunkte (Entry Endpoints) aus denen sie ihre Daten beziehen bzw. Austrittsendpunkte (Exit Endpoints) an die bearbeitete Daten weitergereicht werden, die Bedingungen, in welcher Reihenfolge sie durchlaufen werden und das Ziel, wo das Ergebnis zum Abschluss des Prozesses abgelegt werden soll. Die Steuerung ist dabei dynamisch und wählt Services nach semantischen Kriterien aus, d. h. sowohl nach ihrem Status als auch ihrem Inhalt. Das bedeutet, das DPF bildet die Anwendung zur Laufzeit. Dafür erforderlich ist neben der Koordination von Services auch die Fähigkeit zur Transformation von Daten bzw. das Anreichern mit Informationen aus weiteren Quellen. Realisiert wird so etwas mit internen Services zur Transformation, Routing und speziellen Umformungen (vgl. [Kis03], S. 39). Kischel nennt hier (vgl. [Kis03], S. 40):

- Transformations-Services (bei Kischel XFORM genannt), die in XML beschriebene Daten einer Nachricht mittels XSLT[8]-Prozessoren in ein anderes Format übertragen können.
- CBR-Services, die sämtliche Informationen einer Nachricht (Eigenschaften, Kopfdaten (Header Informationen) und Inhalt) dazu heranziehen, um den nächsten Endpunkt festzulegen.
- Custom-Services als spezielle Services, die bspw. mittels entsprechender Parser in der Lage sind, proprietäre Datenformate in ein XML-Format zu übertragen.

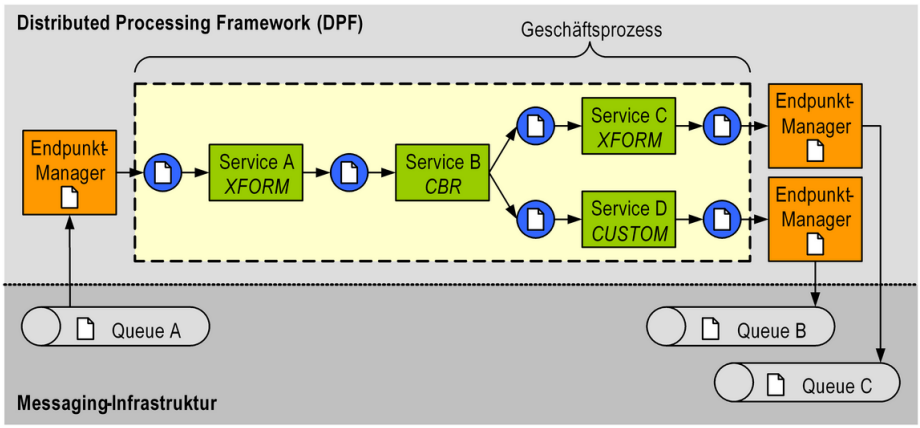

Abbildung 2.1: Enterprise Service Bus (in Anlehnung an [Kis03], S. 38).

[8] XSLT steht für Extensible Stylesheet Language Transformations, d. h. auf der Extensible Stylesheet Language (XSL) aufbauende Transformationen von XML-Dokumenten.

Das DPF und die darunter liegende Messaging-Infrastruktur sind folglich für Kischel integrale Bestandteile eines ESB. Neu im Vergleich zu den bisherigen Definitionen und Charakterisierungen ist vor allem das DPF und insbesondere seine Fähigkeiten, Prozesse bzw. Prozessketten zu unterstützen. Weiterhin taucht bei ihm, das Deployment[9] betreffend, erstmals der Begriff Service-Container auf, den er beschreibt als eine Kapselung für originäre Services sowie Altanwendungen. Eine Konkretisierung dieses Begriffes erfolgt jedoch nicht, weshalb sich seine Bedeutung für den ESB nur schwer abschätzen lässt (vgl. [Kis03], S. 39).

Im Kontext von EAI und SOA bemängelt Kischel, dass übliche Implementierungen entscheidende Qualitäten vermissen lassen. Zu diesen Schwachstellen gehören bspw. (vgl. [Kis03], S. 37):

- **Transaktionsgesicherter Aufruf der Service-Methoden:** Es muss eine definierte Fehlerbehandlung und Protokollierung sichergestellt sein, besonders im Kontext verteilter Geschäftsanwendungen, die unternehmensübergreifende Abläufe, Business-to-Business (B2B) realisieren sollen.
- **Verzögerter und asynchroner Aufruf von Services:** Es kann in dynamisch zur Laufzeit gebildeten Anwendungen keine permanente Verfügbarkeit von Anwendungen und Services, wie sie synchrone Verbindungen erfordern, vorausgesetzt werden. Umso wichtiger ist eine asynchrone Weiterleitung von Nachrichten, die Zwischenspeicherung von Aufrufen und die Fähigkeit wiederholter Übermittlungsversuche.
- **Skalierbarkeit der Anwendung:** Im Umfeld hoch verteilter Anwendungen ist die zu erwartende Verarbeitungslast nur selten im Vorfeld bestimmbar. Vielmehr muss diese dynamisch verteilt werden können.
- **Datensicherheit:** Zum Schutz vor Angriffen gegen kollaborative Anwendungen und die unter ihnen ausgetauschten Daten müssen die häufig hoch sensiblen Geschäftsdaten vor dem Zugriff unberechtigter Dritter geschützt werden.

Ein ESB so Kischel erfüllt diese erweiterten und entscheidenden Anforderungen, indem er u. a. eine entsprechende Messaging-Infrastruktur als Grundlage der physischen Implementierung verwendet. Der ESB stellt hier eine Verbindung aus serviceorientierten Architekturen mit den Fähigkeiten und Qualitäten einer Messaging-Infrastruktur dar. Er kann somit als ein möglicher Ansatz für die Implementierung einer SOA angesehen werden (vgl. [Kis03], S. 39).

Bezüglich des Einsatzgebietes eines ESB gibt Kischel im Gegensatz zu seinen sonstigen Ausführungen nur sehr knapp Auskunft. Im Prinzip nur an einer Stelle, nämlich im Kontext von transaktionsgesicherten Aufrufen, ist von verteilten Geschäftsanwendungen, unternehmensübergreifenden Abläufen und B2B-Anwendungen die Rede. Wenngleich dies nicht unbedingt als ausführlich zu bezeichnen ist, so gibt es zumindest darüber Aufschluss, dass der Fokus über einen reinen unternehmensinternen Einsatz hinausgeht (vgl. [Kis03], S. 37).

In einem abschließenden Vergleich mit den bereits genannten Definitionen insbesondere mit der von Wikipedia muss ein hohes Maß an Konsens festgestellt werden. Nennt Kischel bspw. ein auf nachrichtenorientierter Middleware gestütztes Messaging-System als Fundament für einen ESB, so ist diesbezüglich bei Wikipedia von standardbasierter Middleware

[9] Der Begriff Deployment wird oft in unterschiedlichster Weise verwendet. Die Bedeutungen reichen dabei von Einsatz bzw. Inbetriebnahme über Auslieferung und Verteilung bis hin zu Konfiguration. Zur Klärung dieser Begriffsvielfalt soll er wie folgt definiert werden. Als Deployment kann der Softwaretransfer (Verteilung bzw. Auslieferung) vom Ort der Entwicklung bis hin zum Ausführungsort einschließlich notwendiger Konfigurationen und der anschließenden Inbetriebnahme (Einsatz) verstanden werden.

Infrastruktur die Rede. Einziger wesentlicher Unterschied oder besser Erweiterung zur Definition nach WIKIPEDIA stellt das DPF dar, also eine Komponente zur Prozesssteuerung. Weiterhin werden bspw. mit XSLT-Prozessoren auch konkrete Technologien zur Realisierung genannt, während bei WIKIPEDIA nur allgemein von Orchestrierung, Transformation und Routing die Rede ist.

2.2.3 Definition nach Chappell

Die mit Abstand umfassendste Definition des Begriffs ESB liefert David A. Chappell in seinem Buch „Enterprise Service Bus“ (siehe [Cha04b]), dem wohl zurzeit einzigen Fachbuch, das sich ausschließlich mit dem ESB befasst. Im Prinzip versucht CHAPPELL innerhalb des gesamten Buchs den Begriff ESB zu definieren und zu charakterisieren. Eine erste von ihm gegebene Definition ist die folgende:

> *„Das ESB Konzept ist ein neuer Ansatz in Bezug auf Integration, das das Fundament für ein lose gekoppeltes, hoch verteiltes Integrationsnetzwerk bildet, welches weitaus skalierbarer ist, als beschränkte, Hub-and-Spoke basierte EAI Broker. Ein ESB ist eine standardbasierte Integrationsplattform, die sowohl Nachrichten, Web Services, Datentransformation als auch intelligentes Routing miteinander verbindet, um eine erhebliche Zahl unterschiedlichster Anwendungen entlang erweiterter Unternehmen zuverlässig zu verbinden, ihre Interaktionen zu koordinieren und für transaktionale Integrität zu sorgen. [. . .] Ein erweitertes Unternehmen repräsentiert hierbei eine Organisation und ihre Geschäftspartner, die sowohl durch geschäftliche als auch durch physische Grenzen von einander getrennt sind.“*, [Cha04b], S. 1.

Demnach handelt es sich beim ESB um zweierlei, zum einen um ein Konzept bzw. Ansatz zur Realisierung von Integration, zum anderen um eine Integrationsplattform. Hauptaufgabe des ESB nach CHAPPELL ist es, unterschiedlichste Anwendungen entlang erweiterter Unternehmen zu integrieren. Diese Aufgabe umfasst das Verbinden einer erheblichen Zahl heterogener Anwendungen, das Koordinieren von Interaktionen und die Gewährleistung transaktionaler Integrität. Grundlage eines ESB bilden dabei Nachrichten, Web Services, Dienste zur Datentransformation sowie intelligentes Routing.

Eine weitere kurze Charakterisierung gibt CHAPPELL, wenn er schreibt:

> *„Der ESB ist eine neue Architektur bezüglich Integration, die weltweit in Unternehmen zu erblühen beginnt.“*, [Cha04b], S. 43.

Im Gegensatz zur erstgenannten Definition handelt es sich demnach beim ESB um eine Architektur. Von einem Konzept ist nicht mehr die Rede, jedoch wird erneut von Integration gesprochen. An anderer Stelle wiederum, im Rahmen eines Interviews antwortet CHAPPELL auf die Frage, was der ESB und wie sein Verhältnis zu SOA und Web Services ist:

> *„Ein ESB ist eine Infrastruktur für eine unternehmensweite SOA, die es ermöglicht, ein verteiltes Netzwerk von Applikationen und Services flexibel zu verwalten. Der ESB dient also als Backbone*[10]*, auf dem man eine SOA aufbauen kann.“*, [Cha05a], S. 17.

[10] Backbone bedeutet wortwörtlich Rückgrat und meint in diesem Zusammenhang Gerüst beziehungsweise Grundlage.

Betrachtet man diese Aussagen, so wird die ganze Vielschichtigkeit des Begriffs ESB deutlich. Vom Konzept bzw. Ansatz bis hin zur Architektur und Infrastruktur reichen die Beschreibungen. Einziger Vorteil, der sich daraus ergibt, ist, dass CHAPPELL damit in keinem direkten Widerspruch zu anderen Definitionen steht. Problematisch bleibt jedoch die fehlende Abgrenzung der einzelnen Begriffsdefinitionen zueinander, die eine klare und eindeutige Definition erfordern würde.

Für ein klareres Bild sorgen hingegen die weiterführenden Betrachtungen von CHAPPELL. Zu den Charakteristiken bzw. Eigenschaften eines ESB gehören nach ihm:

- **Durchdringung und Durchgängigkeit (Pervasivness):** Der ESB kann, wie Abbildung 2.2 (auf Seite 20) zu entnehmen ist, den Kern eines durchgängigen Netzwerks bilden, welches sich über Abteilungen, Geschäftsbereiche, Geschäftspartner und darüber hinaus erstrecken kann. Neben großflächigen Integrationsprojekten ist er nicht zuletzt durch seine Fähigkeit, sich nahezu jeder Integrationsumgebung anzupassen, auch für Integration auf lokaler Ebene geeignet. Applikationen können sich je nach Notwendigkeit in den Bus einklinken, andere Anwendungen und Dienste finden und mit ihnen Daten austauschen. Obwohl Web Services einen integralen Bestandteil des ESB bilden, werden auch andere Protokolle, Nachrichtenumgebungen oder Anwendungsadapter Dritter unterstützt (vgl. [Cha04b], S. 7 f.).
- **Standardbasierte Integration:** Standardbasierte Integration ist das fundamentale Konzept des ESB. Zu diesen Standards, die von einem ESB unterstützt werden müssen, gehören neben Java-basierten Standards wie JMS, JCA auch Microsoft Standards wie COM/DCOM oder .NET. Als wohl wichtigster Standard bzw. Datenformat ist XML zu nennen. Damit einher gehen XSLT, XPath und XQuery zur Datentransformation von XML-Dokumenten. Im Zusammenhang von Web Services sind neben XML noch weitere Standards wie bspw. SOAP (als Protokoll), WSDL (zur Schnittstellenbeschreibung), BPEL4WS[11] (zur Unterstützung von Geschäftsprozessen) als einige der wichtigsten zu nennen. Aufgabe des ESB ist es demnach, auf Industriestandards basierende sowie proprietäre Komponenten über standardisierte Schnittstellen in einer offenen Architektur zu integrieren (vgl. [Cha04b], S. 8 f.).
- **Hoch verteilte Integration und wählbares Deployment:** Ein ESB muss es Diensten ermöglichen, in einer hoch verteilten Art und Weise zusammenzuarbeiten. Dazu gehört auch, dass diese unabhängig voneinander verteilt und skaliert werden können. Insbesondere durch die Implementierung von Integrationsaufgaben in Form von Services wird es möglich, diese frei wählbar auszuliefern und in Betrieb zu nehmen (vgl. [Cha04b], S. 9 f., S. 110).
- **Verteilte Datentransformation:** Da die meisten Anwendungen bzw. Services, die an einen ESB angeschlossen sind, nicht dieselben Datenformate verwenden, müssen diese transformiert werden können. Aus diesem Grund bedarf es spezieller Transformationsdienste, die in der Lage sind, dies für hoch verteilte Anwendungen, d. h. Anwendungen, die an irgend einer Stelle an den ESB angeschlossen sind und auf die von irgend einer anderen Stelle aus zugegriffen wird, zu realisieren. Mit Blick auf Integration ist die Datentransformation immanent wichtig und daher auch eine der Kernaufgaben eines ESB (vgl. [Cha04b], S. 10).
- **Erweiterbarkeit durch überlagerte Services:** Zur Unterstützung unterschiedlichster Integrationsprojekte muss ein ESB um überlagerte Services erweitert werden können. Nur so ist es möglich, spezifischen Anforderungen gerecht zu werden. Beispiele

[11] BPEL4WS steht für den Einsatz der Business Process Execution Language (BPEL) im Kontext von Web Services.

für solche Services sind Software aus dem Bereich BPM zur Bearbeitung von Geschäftsprozessen, Collaboration Server zur Bereitstellung spezieller Dienste im Umgang mit Geschäftspartnern und nicht zuletzt auch spezielle Transformationssoftware von Drittherstellern zur Konvertierung von externen Daten (vgl. [Cha04b], S. 10).

- **Ereignissteuerung:** Innerhalb eines ESB werden Services wie abstrakte Endknoten betrachtet, die auf asynchrone Ereignisse reagieren. Dies bedeutet, dass, sobald eine Nachricht einen Service (Endknoten) erreicht, ein Ereignis ausgelöst und die Nachricht daraufhin weiter bearbeitet wird (vgl. [Cha04b], S. 10).

- **Prozessunterstützung:** Die Prozesssteuerung eines ESB kann von einer einfachen Reihenfolge fester Schritte bis hin zu einer hoch anspruchsvollen Geschäftsprozessorchestrierung mit parallel ausgeführten Prozessen, bedingten Verzweigungen und Zusammenführungen reichen. Diese können durch einfache Metadaten oder spezielle Sprachen wie BPEL4WS gesteuert werden. Herausforderung ist dabei die Realisierung einer hoch verteilten Geschäftsprozessorchestrierung ohne zentrale Steuerungs- oder Ausführungskomponente. Prozesssteuerung umfasst dabei u. a. auch Aspekte wie intelligentes Routing von Nachrichten basierend auf deren Inhalt. Da die Prozesssteuerung des ESB auf einer hohen Abstraktionsebene angesiedelt ist, können hier, ohne Berücksichtigung von Netzwerkgrenzen oder mehrfachen Protokollsprüngen zwischen Anwendungen und Diensten, hoch verteilte Deployment-Topologien realisiert werden (vgl. [Cha04b], S. 11).

- **Sicherheit und Zuverlässigkeit:** Die Verbindungen zwischen einzelnen Knoten des ESB sind Firewall-fähig[12]. Zusätzlich ist er in der Lage, durch harte Authentifizierung[13] und Zugriffsteuerung (Autorisation[14]) für Sicherheit zwischen Anwendungen und ESB sowie auch zwischen einzelnen ESB-Knoten zu sorgen. Zuverlässigkeit wird durch eine hochleistungsfähige MOM als Kern des ESB erreicht. Aufbauend auf diesem Kern werden asynchrone Kommunikation, zuverlässige Versorgung mit Geschäftsdaten und transaktionale Integrität sichergestellt (vgl. [Cha04b], S. 12).

- **Autonomie aber föderierte Umgebung:** Das Ziel einer losen Bindung zwischen Anwendungen und Diensten impliziert ein hohes Maß an Autonomie. Es muss ermöglicht werden, dass einzelne Geschäftseinheiten und -bereiche Kontrolle über ihre eigenen, lokalen IT Ressourcen haben. Dazu gehört, dass seitens des ESB die Installation, Konfiguration, Sicherheit und Management von lokalen Komponenten, Adaptern und Nachrichtenverkehr ermöglicht wird. Darüber hinaus müssen auch die lokalen Integrationsbereiche in ein größeres Integrationsnetzwerk mit integriertem Sicherheitsmodell eingeklinkt werden können, wie in Abbildung 2.3 (auf Seite 20) dargestellt. Die Verteilungseigenschaften des ESB werden durch Abstraktion von physischen Details des Deployments bzw. von zugrunde liegenden Netzwerkprotokollen der Endknoten erreicht. Die Eigenschaften bezüglich Föderierung hingegen basieren auf der Fähigkeit, wahlweise Anwendungsbereiche und Sicherheitsgrenzen trennen oder durchqueren zu können (vgl. [Cha04b], S. 12).

- **Entfernte Konfiguration und Management:** Nicht immer ist es für ein Unternehmen sinnvoll, an allen seiner IT-Standorte auch IT-Personal zu beschäftigen. Beispiele hierfür sind besonders kleinere Vertriebsorganisationen wie etwa Geschäfte oder Filialen. Dennoch besteht die Notwendigkeit, diese im Rahmen eines ESB miteinander zu

[12] Durch den Einsatz von HTTP (etwa mittels SOAP) ist die Durchdringung von Firewalls leichter möglich als bspw. mittels anderer, proprietärer Protokolle.
[13] Authentifizierung bezeichnet den Identitätsnachweis gegenüber einem Dritten (vgl. [Bis03], S. 309; [TS03], S. 369.
[14] Autorisation bezeichnet die sich auf Authentifizierung stützende Vergabe von Rechten bzw. Privilegien (vgl. [TS03], S. 470).

koppeln. Mittels einer Integrationsvorlage, die Schnittstellen, Routing, und Sicherheitsaspekte umfasst, kann eine Verbindung zwischen einer Management-Komponente und mehreren zu verwaltenden Geschäftseinheiten hergestellt werden. Dazu wird die Integrationsvorlage an die zu verwaltenden Geschäftseinheiten verteilt und dort angepasst (vgl. [Cha04b], S. 14 f.).

- **XML als Grundlage:** XML stellt ein ideales Fundament zur Datenrepräsentation dar. Daten, die von einer Unmenge von Anwendungen produziert und konsumiert werden, liegen in fast genauso vielen Daten- wie Paketformaten vor. Wenngleich es durchaus möglich wäre, auch diese Daten mittels ESB zu verschicken, so bietet die Verwendung von XML einige Vorteile. Beispiele hierfür sind die Möglichkeiten mittels spezieller Services, Daten aus verschiedenen Quellen miteinander zu neuen Daten zu kombinieren, Nachrichten zu erweitern, zu transformieren oder einfach nur umzuleiten (vgl. [Cha04b], S. 16).

- **Durchsatz in Echtzeit:** Der ESB ermöglicht den Echtzeitdurchsatz von Daten in dem Sinne, dass Daten ohne große Verzögerung über den ESB verschickt werden können. Heutzutage führen immer noch populäre Methoden wie nächtliche Bulk- und Batchläufe zu Problemen wie Fehleranfälligkeit oder Verzögerungen in der Beschaffung aktueller Informationen. Dies soll mittels ESB überwunden werden (vgl. [Cha04b], S. 16).

- **Operationale Überwachung:** Eine wichtige Eigenschaft eines ESB ist die Möglichkeit, Einblick in den Zustand und Fortschritt von Geschäftsprozessen zu erhalten. Als wesentlicher Vorteil erweist sich hierbei XML als zugrunde liegendes Datenformat. Dank XML ist es leicht möglich, in Daten Einblick zu nehmen bzw. diese mittels anderer Dienste auszuwerten, um so eine genaue Information über den jeweiligen Prozesszustand zu gewinnen. Da zur Bereitstellung dieser Funktionalität – egal ob als Teil des ESB oder in Form von Erweiterungen – nur auf bereits vorhandene Bestandteile der Infrastruktur zurückgegriffen wird, kann auf zusätzliche Produkte aus dem Bereich Business Activity Monitoring (BAM) verzichtet werden. Das Prüfen und Verfolgen von Geschäftsprozessen ist auch dadurch möglich, indem entlang von Prozessketten spezielle Punkte definiert werden können. Sollte eine Nachricht einen solchen Punkt erreichen, wird ein Ereignis ausgelöst und alle für wichtig erachteten Daten (Zustand des Prozesses, spezielle Nachrichteninhalte, etc.) können an anderer Stelle gesammelt und ausgewertet werden. Noch weiterreichender sind fortgeschrittene Dienste zur Sammlung von Daten, zu Anfragemechanismen und zum Reporting, die es darüber hinaus ermöglichen, alle gesammelten Daten in entsprechender Weise zu präsentieren (vgl. [Cha04b], S. 16 f.).

- **Inkrementelle Anpassung:** Eine der primären Eigenschaften, die einen ESB von anderen Konzepten unterscheidet, ist seine Fähigkeit zur inkrementellen Anpassung. Im Gegensatz zum Prinzip „alles oder nichts“ erlaubt er vor allem durch Fähigkeiten wie Autonomie und Föderation ein schrittweises Vorgehen. Diese in Abbildung 2.4 (auf Seite 20) dargestellte schrittweise Integration führt schnell zu integrationsbedingten Mehrwerten, während an weiteren Integrationsphasen gearbeitet wird (vgl. [Cha04b], S. 18).

Ein Punkt, der in diesen Beschreibungen noch nicht auftaucht, sind die schon bei KISCHEL erwähnten Service-Container und so genannten Endpunkte. Diese Endpunkte kapseln Dienste und Anwendungen, die selbst wiederum auf andere Dienste zurückgreifen können. Oft werden sie auch als abstrakte Endpunkte bezeichnet, weil nämlich die gesamte Implementierung ihrer Funktionalität verborgen bleibt. Abstrakte Endpunkte dienen vor allem Werkzeugen in höheren Schichten, z. B. bei der Entwicklung von Prozessabläufen und residieren in entsprechenden Service-Containern. Service-Container sind die physische Repräsentation von

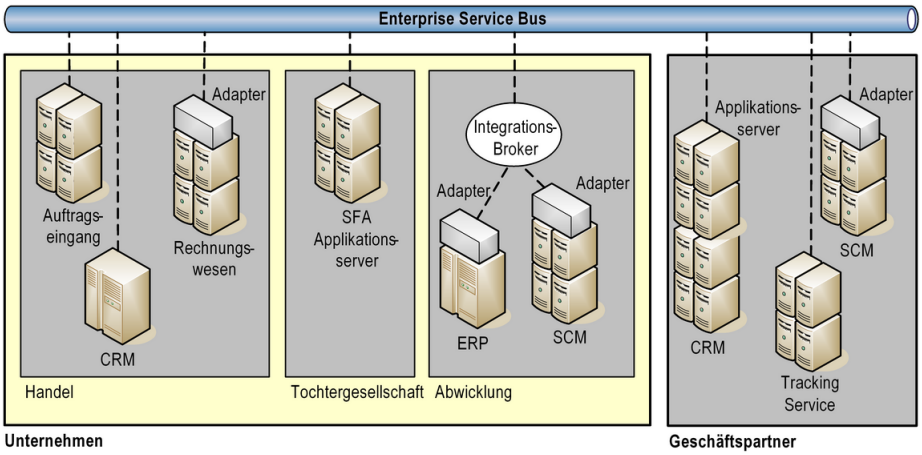

Abbildung 2.2: Enterprise Service Bus (in Anlehnung an [Cha04b], S. 8).

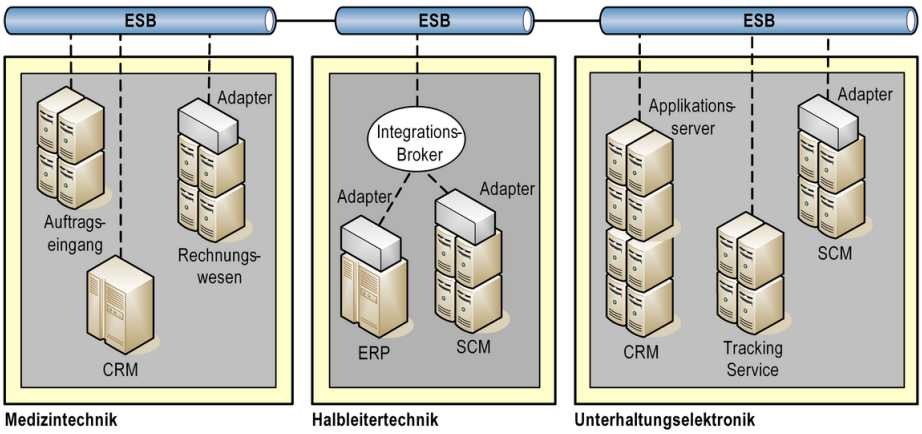

Abbildung 2.3: Föderierter ESB (in Anlehnung an [Cha04b], S. 13).

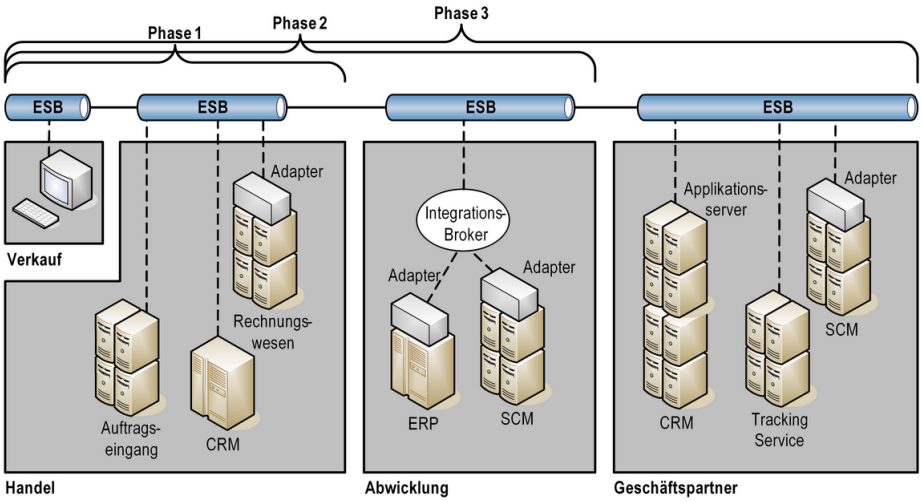

Abbildung 2.4: Inkrementelle Anpassung an den ESB (in Anlehnung an [Cha04b], S. 18).

Endpunkten und stellen die Implementierungen der jeweiligen Service-Schnittstellen bereit. In diesem Punkt besteht eine gewisse Ähnlichkeit zu Containern von Applikationsservern. Neben den Service-Schnittstellen verfügen diese Container über ein Invocation und Management Framework, das sie mit entsprechenden Management-Werkzeugen verbindet und über das Metadaten (Konfiguration, Fehlerbehandlung oder Protokollierung betreffend) ausgetauscht werden können.

In mancherlei Hinsicht stellen diese Service-Container eher den eigentlichen ESB dar als die darunter liegende Middleware. So ist es auch der Container, der die jeweiligen Services mit Daten versorgt. Weiterhin übernimmt dieser Aufgaben wie Lokalisierung, Routing und Service Invocation (d. h. Einbindung von Diensten). Das Management von Servicelebenszyklen[15] gehört dabei genauso zu seinen Aufgaben wie Thread Pooling, dem Bereitstellen mehrerer Instanzen eines Services innerhalb eines Service-Containers. Nicht zuletzt ist es auch der Service-Container, der sich um Quality of Protection (QoP), Sicherheit, Quality of Service (QoS) und Transaktionsverwaltung kümmern muss (vgl. [Cha04b], S. 110 ff.).

Im Zusammenhang mit diesem Konzept verweist Chappell auch auf die Java Business Integration (JBI), eine durch die Softwareindustrie (speziell Sun Microsystems) vorangetriebene Spezifikation, die sich ebenfalls mit der Beschreibung eines Service- und Integrations-Containers befasst. Das mit diesem Meta-Container einhergehende Containermodell soll einerseits bei der Implementierung eines ESB helfen und andererseits den Austausch und das Zusammenspiel von Integrationskomponenten vereinfachen. Der grundlegende Aufbau ähnelt trotz gewisser Eigenheiten dem Ansatz von Chappell. Infolge des praxisgetriebenen Charakters von JBI soll im Rahmen dieses Kapitels auf eine detailliertere Betrachtung verzichtet werden (diese erfolgt im anschließenden Kapitel) (vgl. [Cha04b], S. 184 ff.).

Bezüglich des Anwendungsfeldes des ESB wird von erweiterten Unternehmen gesprochen, d. h. von Organisationen und ihren Geschäftspartnern. Der Fokus des ESB liegt demnach auf einer unternehmensübergreifenden Integration, deren größte Herausforderungen in den geschäftlichen wie physischen Grenzen dieser erweiterten Unternehmen liegen. Infolge von Fusionen, Übernahmen, immer wichtigerem Informationsaustausch zwischen Geschäftspartnern und Management von global verteilten Geschäftseinheiten entsteht ein Zwang zur Verbindung und Integration heterogener IT-Landschaften. In diesem Feld stellt der ESB geeignete Fähigkeiten bereit, bestehende Grenzen und Hindernisse zu überwinden (vgl. [Cha04b], S. 1 und S. 48 ff.).

Nach Chappell bildet der ESB das Rückgrat für den Aufbau einer SOA. Die innerhalb des ESB eingesetzten Konzepte hingegen sind nicht vollkommen neu, sondern basieren im Wesentlichen auf früheren Konzepten, die bereits im Kontext von MOM und EAI Anwendung fanden. Diesbezüglich liefert Chappell, mit der seinem Buch entnommenen Abbildung (Abbildung 2.5), eine Übersicht zur Illustration. Gemäß dieser Darstellung und seinen Beschreibungen sieht er den ESB als eine Weiterentwicklung von MOM und EAI. Beide Konzepte (MOM und EAI) weisen individuelle Defizite auf, die der ESB beseitigt. Von EAI erbt der ESB die Trennung von Anwendungs- und Integrationslogik und von MOM übernimmt er die Fähigkeiten zur asynchronen Nachrichtenübertragung wie lose Kopplung und erlaubt damit eine verteilte Integration (vgl. [Cha04b], S. 4 f.).

[15] Im engeren Sinn und gemäß dem World Wide Web Consortium (W3C) besteht ein Servicelebenszyklus lediglich aus den beiden Hauptzuständen hochgefahren (Up) und heruntergefahren (Down) sowie deren Teilzuständen beschäftigt (Busy), unbeschäftigt (Idle), angehalten (Stopped), bis zur Grenze belastet (Saturated) und fehlerhaft beendet (Crashed) (vgl. [Wor04]). Folgt man jedoch anderen Autoren, so umfasst ein Servicelebenszyklus im erweiterten Sinn zusätzlich Aspekte wie Analyse, Design, Implementierung, Deployment, Test, Konfiguration, Versionierung und Management/Wartung (vgl. [Cha04b], S. 116; [VG06], S. 4.). Im Folgenden wird daher, sofern nicht anderes angegebenen, stets von der erweiterten Sichtweise ausgegangen.

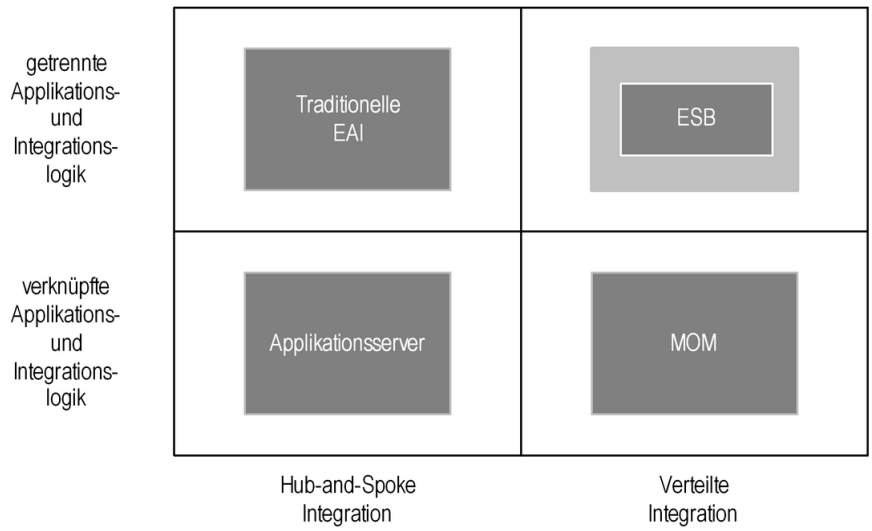

Abbildung 2.5: Integrationsansätze (in Anlehnung an [Cha04b], S. 18).

Als Nachteile traditioneller EAI-Broker im Gegensatz zum ESB sieht Chappell vor allem deren Hub-and-Spoke Architektur. Probleme, die dabei auftreten können, sind eine fehlende regionale Kontrolle von lokalen Integrationsbereichen und Grenzen bei der Durchdringung von physischen Netzwerksegmenten und Firewalls. Die Konsequenz ist, dass selbst wenn sich diese Architektur über organisatorische Grenzen hinweg erstreckt, sie dennoch keine lokale Autonomie einzelner Geschäftsbereiche erlaubt, die diese brauchen, um mindestens teilweise unabhängig voneinander arbeiten zu können. Es ergibt sich ein Konflikt von lokaler Autonomie und zentraler Steuerung (vgl. [Cha04b], S. 33).

Die gesamte Definition nach Chappell stellt sich als äußerst umfangreich und detailliert dar. Vor allem in Bezug auf Eigenschaften und Charakteristiken eines ESB ergänzt er bereits genannte Definitionen ohne mit diesen in Widerspruch zu geraten. Dennoch bleibt der eigentliche Begriff im Kern mehrdeutig. Die Umsetzung eines ESB betreffend konkretisiert er bereits von Kischel erwähnte Endpunkte und Service-Container und hebt ihre Bedeutung im Gesamtkontext hervor. Im Gegensatz zu allen anderen bisherigen Definitionen liegt der Fokus klar auf einer unternehmensübergreifenden Anwendung und Integration. Nichts desto trotz betont Chappell auch die Fähigkeiten des ESB, unternehmensintern, sogar bereichs- oder abteilungsintern eingesetzt zu werden und lokal zu integrieren. Flexibel kann dann diese Integration bis hin zu einer sich weltweit erstreckenden, unternehmensübergreifenden Integration ausgebaut werden. Grundlage dafür bilden Konzepte, die sich schon bei EAI und traditioneller MOM finden lassen, aber noch nicht in der Form verändert, kombiniert und erweitert wurden, wie es beim ESB der Fall ist. Der ESB ist in dieser Reihe der nächste Schritt, der für Chappell die Realisierung einer SOA ermöglicht.

2.2.4 Definition nach Dostal/Jeckle/Melzer/Zengler

Nach Dostal/Jeckle/Melzer/Zengler stellt sich der Begriff ESB wie folgt dar:

> *„Befragt man Experten zum ESB, so bekommt man [...] mindestens zwei verschiedene Antworten. Ein Teil wird den ESB als Bestandteil einer SOA bezeichnen. In diesem Bild wird die Kommunikation in der SOA über eine (zumindest*

> *virtuell) zentrale Komponente mit intelligenten Routing-Fähigkeiten vorgenommen. Dieser Verteiler wird dann als ESB bezeichnet. Der andere Teil bezeichnet den ESB als die logische Weiterentwicklung der SOA. Um ein ereignisgetriebenes Unternehmen in der IT umzusetzen, benötigt man Eigenschaften der SOA wie lose Kopplungen. Der kritische Datenfluss wird vom ESB übernommen. Hier behaupten manche, dass der ESB daher als Backbone einer SOA gesehen werden kann, was wieder nahe bei der ersten Gruppe liegt.“*, [DJMZ05], S. 19.

Problematisch an dieser Definition ist, dass, falls man ihr folgt, ein ESB zweierlei sein kann, einerseits die zentrale Komponente einer SOA und andererseits deren logische Weiterentwicklung. Dies gestehen die Autoren in ihren weiteren Ausführungen auch ein, halten sich jedoch für außer Stande eine exakte Definition zu liefern. Um den Begriff dennoch genauer zu charakterisieren, benennen Sie folgende Aufgaben, die ein ESB erfüllen muss (vgl. [DJMZ05], S. 20 f.):

- Transformation,
- Protokollunabhängigkeit,
- intelligentes Routing,
- Einfachheit,
- Standards.

Mit Transformation ist hier die Umwandlung bzw. Übersetzung verschiedener Datentypen gemeint, die unsichtbar erfolgen soll und damit allen Partnern erlaubt, miteinander zu kommunizieren. Eine ähnliche Aufgabe stellt auch die Gewährleistung von Protokollunabhängigkeit dar, die eine Datenübertragung mittels unterschiedlichster Protokolle unterstützen soll. Während es Transformation und Protokollunabhängigkeit einem Dienst erlauben, sich mit dem Bus zu verbinden, stellt dies noch nicht sicher, dass Daten zwischen beteiligten Partnern ausgetauscht werden können. Hierzu bedarf es dem intelligenten Routing, das ähnlich wie ein Netzwerk-Switch dafür sorgt, dass Datenpakete ihre Adressaten erreichen. Bezüglich der Einfachheit stellen Dostal/Jeckle/Melzer/Zengler die Forderung auf, dass die nicht vermeidbare Komplexität einer solchen Umsetzung gegenüber den beteiligten Diensten verborgen werden soll. Standards, die für Flexibilität, automatische Integration und Herstellerunabhängigkeit sorgen sollen, bilden eine weitere Grundeigenschaft eines ESB (vgl. [DJMZ05], S. 20 f.).

Einziges Manko im Zusammenhang mit den Aufgaben eines ESB ist, dass Dostal/Jeckle/Melzer/Zengler leider nicht nennen, wie genau diese Funktionalität realisiert werden kann, d. h. welche konkreten Technologien Anwendung finden. Eine weitere Schwachstelle dieser Definition ist die unbeantwortete Frage, wo ein ESB zum Einsatz kommt. Äußerst bedauerlich dabei, dass sie gerade in diesem Punkt einigen zuvor genannten Definitionen gleicht.

In Bezug auf andere Konzepte sehen Dostal/Jeckle/Melzer/Zengler den ESB als eine Technologie, die bisherige nachrichtenorientierte Middleware erweitert. In wieweit sich der ESB dabei von dieser abgrenzt und wie genau er diese erweitert, bleibt unthematisiert. Weiterhin unberücksichtigt bleibt auch seine Rolle im Kontext von EAI (vgl. [DJMZ05], S. 21).

Alles in allem ist die Definition nach Dostal/Jeckle/Melzer/Zengler äußerst unbefriedigend. Der Begriff ESB bleibt schwammig und viele Definitionskriterien unberücksichtigt.

Einzig die Aufgaben, die der ESB zu erfüllen hat, geben Aufschluss über seinen Charakter. Verglichen mit anderen Definitionen lassen sich in diesem Punkt große Parallelen feststellen, wenngleich die Sichtweise des ESB als logische Weiterentwicklung von SOA im starken Widerspruch zu diesem steht.

2.2.5 Definition nach Lorenzelli-Scholz (OBJEKTspektrum)

In einem weiteren Fachartikel – zum Thema „Service-orientierte Integration mit einem 'Enterprise Service Bus"' – aus der OBJEKTspektrum ist die folgende Definition entnommen:

> *„Moderne ESBs kann man sich wie einen Middleware-Switch vorstellen. Er funktioniert nach dem Prinzip eines Telekommunikations-Switches, der Anrufe über ein weit verteiltes, heterogenes Netzwerk vermittelt. Diese neue Generation von ESBs vermittelt Nachrichten verschiedener Formate über eine verteilte Infrastruktur heterogener Middleware-Transportprotokolle, wie Tuxedo, MQ, CORBA, TIBCO oder HTTP. Dabei kann sich jeder beliebige Service in die SOA mittels ESB einklinken. Hierfür müssen alle relevanten Plattformen unterstützt werden (von mobilen Endgeräten bis hin zum Mainframe[16]), die notwendige Servicequalität (Sicherheit, Transaktion, Hochverfügbarkeit usw.) muss bereitgestellt werden und nicht zuletzt müssen ESBs natürlich hoch performant sein.*
>
> *Einer der wesentlichen Unterschiede zwischen dem EAI- und dem ESB-Ansatz zeigt sich im Deployment. Gute ESB-Implementierungen unterstützen eine Vielzahl von Deployment-Architekturen und können vor allem ohne eine zentrale Verteilungskomponente auskommen.“*, [LS05].

Anders als bei allen zuvor genannten Definitionen wird der ESB bei Lorenzelli-Scholz als ein Middleware-Switch verstanden[17]. Im Mittelpunkt steht dabei, Nachrichten verschiedener Formate über eine verteilte Infrastruktur heterogener Middleware-Transportprotokolle zu vermitteln. In diesem Zusammenhang äußert Lorenzelli-Scholz weiter, dass es beim ESB nicht darum geht, neue Middleware oder Protokolle einzuführen, sondern vielmehr vorhandene Infrastrukturen so gut wie möglich zu nutzen. Weiterhin besitzt ein ESB auch keinen internen Bus, dieser ist vielmehr ein rein virtuelles Konstrukt, das durch Vermitteln von Nachrichten entsteht (vgl. [LS05], S. 21).

Zur Erfüllung dieser Aufgabe muss der ESB konkrete Integrationsfunktionen wie die Unterstützung für verschiedene Transportprotokolle (wie Tuxedo, WebSphere MQ, CORBA, TIBCO Rendezvous oder HTTP) und Plattformen, die Gewährleistung der notwendigen Servicequalität (Sicherheit, Transaktionssicherheit, Hochverfügbarkeit u. a.) und nicht zuletzt auch hohe Performanz vorweisen können. Eine konkrete Technologie, die Lorenzelli-Scholz in diesem Zusammenhang nennt, ist WSDL. In diese sollen andere, bereits bestehende Schnittstellendefinitionen (IDL[18]-Dateien, TIBCO Nachrichtenbeschreibungen oder COBOL Copybooks) überführt und in einem nächsten Schritt um Aspekte wie Sicherheit, Routing, Bindung, Transaktionskontrolle und Zuverlässigkeit erweitert werden. Es wird in diesem Zusammenhang ein entsprechender Service-Designer angesprochen, eine einer Entwicklungsumgebung ähnliche oder darin integrierte grafische Komponente (GUI[19]) zur Erstellung von Diensten. Wie dieser arbeitet, zeigt das im Anhang A zu findende Beispiel einer

[16] Großrechner
[17] Es wird zwar bereits bei Dostal/Jeckle/Melzer/Zengler von einem Netzwerk-Switch gesprochen, jedoch nur im Zusammenhang mit Routing und nicht allgemein in Bezug auf den ESB.
[18] IDL – Interface Definition Language
[19] GUI – Graphical User Interface

IDL-Datei eines CORBA-Services (siehe Listing A.1), die mittels Service-Designer in eine WSDL-Datei transformiert wurde (siehe Listing A.2).

Bezüglich des Einsatzgebietes eines ESB gibt Lorenzelli-Scholz keinerlei Auskunft, ob dieses innerhalb eines Unternehmens oder auch darüber hinaus angesiedelt ist. Es wird jedoch gesagt, dass das technische Einsatzgebiet eines ESB vom mobilen Endgerät bis hin zum Mainframe reicht.

Im Vergleich zu EAI sieht Lorenzelli-Scholz die Unterschiede zum ESB vor allem im Deployment. Dieser kann ohne zentrale Verteilungskomponente auskommen und zusätzlich unterschiedlichen Deployment-Szenarien gerecht werden. Ob eingebettet im Client (universeller Client), falls serverseitig keine Änderungen möglich sind, ob als Middleware Migration oder einfach zur Bedienung unterschiedlicher Protokolle (universeller Endpunkt), alles ist mit dem ESB realisierbar (siehe Abbildung 2.6) (vgl. [LS05], S. 21).

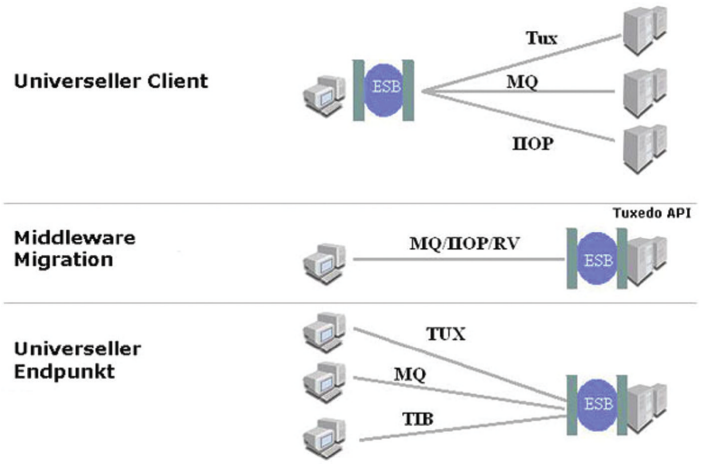

Abbildung 2.6: Deployment-Szenarien des ESB (Quelle: [LS05], S. 20).

Im Kontext von SOA wird der ESB als Lieferant konkreter Integrationsfunktionen und als optimale Grundlage gesehen, um eine solche zu implementieren. Dreh- und Angelpunkt beim Aufbau einer SOA ist für Lorenzelli-Scholz weiterhin eine einheitliche und allgemein verständliche Schnittstellenbeschreibung (vgl. [LS05], S. 19 ff.).

Abgesehen vom Fokus dieser Definition, der stark auf Schnittstellenbeschreibung und WSDL liegt, werden zusätzlich Deployment-Szenarien beschrieben, die in dieser Form bei anderen Definitionen nicht genannt wurden. Der ESB nach Lorenzelli-Scholz gleicht dabei einem Netzwerk-Switch, das auf bereits vorhandener Middleware aufbaut. Ein in einigen Punkten durchaus zutreffender Vergleich, der einer Betrachtung des ESB als Middleware-Infrastruktur nicht zugegen läuft.

2.2.6 Definition nach Rieks (IM)

In der IM aus dem ersten Quartal 2006 macht Rieks bezüglich ESB die folgenden Äußerungen:

> *„Der Kern der handelsüblichen SOA-Produkte ist der so genannte Enterprise Service Bus (ESB). Der ESB ist eine Komponente, die auf einer hohen Abstraktionsebene in der Lage ist, Services bereitzustellen, Datenflüsse zu steuern und gleichzeitig entsprechende Sicherheits- und Management-Features für diese Services anbietet.“*, [Rie06].

Ein ESB nach Rieks ist demnach eine Komponente, die den Kern handelsüblicher SOA-Produkte bildet. Diese Antwort wirft die Frage auf, was Rieks unter „handelsüblichen SOA-Produkten“ versteht. Leider bleibt er die Antwort schuldig. Beantwortet werden, wenngleich auch nur sehr kurz, die Aufgaben, die ein ESB zu erfüllen hat. So muss er Services bereitstellen, Datenflüsse steuern und gleichzeitig entsprechende Sicherheits- und Management-Features anbieten. Obwohl wenig konkret, orientieren sich die hier genannten Aufgaben an denen, die bereits an anderer Stelle etwa bei Kischel oder Chappell angegeben wurden.

Zusammenfassend gesehen, erweist sich die Definition nach Rieks als äußerst knapp. Fragen nach Einsatzgebiet, Parallelen und Unterschieden zu anderen Konzepten (MOM und EAI) oder Technologien zur Realisierung bleiben offen. Einzig reizvoll ist die Betrachtung des ESB als zentrale Komponente einer SOA Implementierung, die verglichen mit anderen Definitionen eher zu einem Widerspruch führt. Ist der ESB bspw. nach Chappell ein Konzept, eine Architektur bzw. eine Infrastruktur, so weist er nach Rieks einen viel stärker ausgeprägten Produktcharakter auf und kann auch als ein solches verstanden werden.

2.2.7 Definition nach Tieke (InformationWeek)

Als eine weitere in ihrer Form sehr kompakte Definition ist die folgende nach Tieke zu sehen:

> *„Der ESB ermöglicht die Integration verschiedener Ansätze und Kommunikationstechniken sowie die Integration von Legacy- und Standardsoftware. Darüber hinaus implementiert er Querschnittsfunktionen, die in einer größeren SOA wichtig sind, etwa für Adressierung und Sicherheit. [...]*
>
> *Er bringt eine Vielzahl von Service-Nehmern und -Gebern mit unterschiedlichen Qualitätsanforderungen und -angeboten zur Laufzeit zusammen. Dazu geben beide Seiten dem ESB über eine so genannte Policy bekannt, welche Anforderungen sie im Hinblick auf die Service-Ausführung stellen oder erfüllen können.“*, [Tie06].

Tieke trifft weder in seiner Definition noch im Rahmen des von ihm verfassten Artikels eine explizite Aussage darüber, was ein ESB ist. Die Frage, ob es sich demnach überhaupt um eine Definition handelt, liegt auf der Hand. Dennoch charakterisiert er den Begriff ESB auf eine Weise, die sich von anderen durchaus unterscheidet. Aufgaben des ESB nach Tieke sind die Integration verschiedener Ansätze und Kommunikationstechniken sowie die Integration von Legacy- und Standardsoftware. Bei einer genauen Betrachtung führt dies bereits in die Richtung einer Meta-Integration, wird aber nicht genauer thematisiert. Ebenfalls hervor sticht die Erwähnung von unterschiedlichen Qualitätsanforderungen und -angeboten, der mittels ESB zu verbindenden Service-Nehmer und -Geber. Hier ist es eine so genannte Policy, über die Anforderungen dem ESB und den darüber verbundenen Partnern mitgeteilt werden. Wie genau sich eine solche Policy realisieren lässt, wird von Tieke leider nicht beschrieben. Trotzdem stellt dieser Aspekt eine Ergänzung zu den bisher genannten Eigenschaften eines ESB dar.

Im Kontext von SOA sieht Tieke den ESB als Teil einer solchen. Dies wird an keiner Stelle explizit genannt, kann jedoch indirekt der Definition entnommen werden. Gleichsam mit

anderen sehr knappen Definitionen wird auch bei TIEKE nicht auf weiterführende Kriterien (Einsatzgebiet, MOM, EAI oder Technologien zur Implementierung) eingegangen, weshalb die Definition in diesen Punkten als ungenügend zu bezeichnen ist (vgl. [Tie06], S. 12).

2.2.8 Definition nach Vollmer/Gilpin (Forrester)

Im Rahmen einer Marktanalyse von Forrester aus dem zweiten Quartal des Jahres 2006 definieren VOLLMER/GILPIN den ESB als:

> *„Softwareinfrastruktur, die wiederverwendbare Business Services (geschäftsrelevante Dienste) auf breiter Fläche Nutzern, Anwendungen, Geschäftsprozessen und anderen Diensten zur Verfügung stellt.“*, [VG06], S. 2.

Beim ESB handelt es sich demzufolge um eine Infrastruktur für Software. Diese Infrastruktur soll geschäftsrelevante Dienste auf breiter Fläche zur Verfügung stellen. Weiterhin ist von Nutzern, Anwendungen, Geschäftsprozessen und anderen Diensten die Rede, welche sich dieser Infrastruktur bedienen. Zur Fragestellung, welche Aufgaben ein ESB ihrer Ansicht nach zu erfüllen hat, bezeichnen sie folgende Eigenschaften und Funktionalitäten als typisch (vgl. [VG06], S. 2. f.):

- **Kommunikationsinfrastruktur:** Der ESB stellt eine Kommunikationsinfrastruktur bereit, die Kommunikationsstile wie Publish-and-Subscribe, synchron/asynchron und ereignisgesteuerte Kommunikation ermöglicht. Hierzu zählt auch die Aufgabe, mittels Adapter unterschiedlichste Verbindungen aufbauen zu können.
- **Routing und Versionierung:** Zu den primären Aufgaben eines ESB gehört ein typ-, inhalts- und wegplanbasiertes Routing. Darüber hinaus gehört hierzu die Fähigkeit zur Schnittstellenversionierung, die insbesondere im Zusammenhang mit versionsgerechtem Routing von Bedeutung ist.
- **Transformation und Mapping:** Angefangen vom Mapping einfacher Datentypen bis hin zu umfangreichen Transformationen (Aggregation und Dekomposition von Dienstinhalten und ihrer Semantik) muss all dies von einem ESB unterstützt werden.
- **Service Orchestrierung, Aggregation und Prozessmanagement:** Zu diesem Aufgabenfeld gehört zum einen, die Fähigkeit kleinere Dienste zu größeren Diensten zu aggregieren und zum anderen, eine sich auf Prozessdefinitionen stützende Orchestrierung von Diensten zu ermöglichen. Grundlage für Orchestrierung und Prozessmanagement können hier auf BPEL basierende Laufzeitumgebungen bilden.
- **Transaktionsverwaltung:** Im Bereich Transaktionsverwaltung wird vom ESB erwartet, mittels einer Ereignissteuerung doppelte, sich ausschließende oder andere inkonsistente Transaktionen zu erkennen. Eine harte Transaktionsverwaltung, wie sie bspw. eng gekoppelte Applikationsserver zur Verfügung stellen, wird nicht gefordert oder nur dann, wenn der ESB überwiegend zur Vermittlung synchroner Verbindungen genutzt wird. Ein Problem stellen hier fehlende Standards aus dem Bereich SOA-basierter Transaktionsverwaltung dar.
- **Sicherheit:** Der ESB muss Sicherheitskonzepte unterstützen können. Eine Möglichkeit stellt die Bereitstellung zusätzlicher Infrastruktur zur Definition und Umsetzung dieser Konzepte dar. Eine andere ist das Einklinken in eine bereits vorhandene Sicherheitsinfrastruktur.

- **Dienstgüte (QoS):** In Bezug auf Dienstgüte kann der ESB Nachrichten in Message Queues halten, Operationen wiederholen, falls Fehler auftreten, alternative Server zur Gewährleistung von Ausfallsicherheit nutzen und einige weitere Schritte mehr, die eine Einhaltung von Dienstgütevereinbarungen (SLA[20]) sicherstellen.
- **Dienstverzeichnis (Service Registry) und Metadatenmanagement:** Zur Unterstützung von Servicelebenszyklen sind Metadaten (Anforderungen, Design, Implementierung, Metriken, Management, Versionierung und Konfiguration) notwendig, die über die mittels WSDL beschreibbaren hinausgehen. Für das Management dieser und anderer Metadaten muss der ESB über entsprechende Registries und Repositories verfügen. Die Unterstützung von UDDI ist hier eine Mindestanforderung.
- **Monitoring und Management:** Neben der Aufgabe, das Management eines ESB so weit wie möglich zu automatisieren, muss es dennoch für Menschen möglich sein, Probleme zu untersuchen, Ursachen zu finden und entdeckte Fehler zu korrigieren. Als ein Weg dies zu unterstützen, kann die Protokollierung und Protokollverwaltung angesehen werden.
- **Unterstützung von Servicelebenszyklen:** Der Lebenszyklus eines Dienstes (Service Life Cycle) umfasst Aspekte, die von Entwicklung, Wiederverwendung und Integration bis hin zu Deployment, Management und Optimierung reichen. All dies sollte ein ESB bspw. mittels etablierter Entwicklungsumgebungen wie Eclipse[21] unterstützen.

Trotz der recht kurzen Definition des eigentlichen Begriffs ESB sind die weiteren Äußerungen von Vollmer/Gilpin bezüglich des Aufgabengebietes eines ESB sehr umfangreich und detailliert. Es werden nicht nur verschiedene Aufgaben und Eigenschaften aufgelistet, sondern darüber hinaus technische Möglichkeiten zur Realisierung genannt. Dazu zählen BPEL, WSDL, UDDI, Repositories für Metadaten oder Eclipse als Entwicklungsumgebung. Trotz der Tatsache, dass viele der von Vollmer/Gilpin beschriebenen Aufgaben bereits genannt wurden, vertiefen sie dennoch andere Definitionen und stellen, vor allem Versionierung und Servicelebenszyklen betreffend, wichtige Erweiterungen dar.

Wie schon andere Autoren gehen Vollmer/Gilpin ebenfalls nicht auf ein mögliches Einsatzfeld eines ESB ein. Ähnlich untransparent bleibt die Einordnung des ESB im Vergleich zu MOM, EAI und SOA. Einzig die beschriebene Kommunikationsinfrastruktur lässt auf den Einsatz von MOM im Kern des ESB schließen.

Insgesamt betrachtet ergänzen Vollmer/Gilpin die Aussagen vorheriger Autoren, insbesondere hinsichtlich der Aufgaben eines ESB. Mit ihrer Begriffsdefinition, den ESB als Infrastruktur betrachtend, haben sie eine ähnliche Sichtweise wie Chappell und stehen in keinem größeren Widerspruch zu anderen Definitionen. Ebenso ist es mit dem von ihnen genannten Hauptzweck des ESB, nämlich über eine entsprechende Infrastruktur Dienste, Anwendungen und Geschäftsprozesse bereitzustellen. Eine Betrachtung, die sich mit anderen deckt.

2.3 Vergleich der Definitionen

Im Rahmen eines Vergleichs bzw. einer Zusammenfassung der genannten Definitionen sollen die zu Beginn aufgestellten Definitionskriterien herangezogen werden. An erster Stelle stand

[20] SLA – Service Level Agreements

[21] Eclipse ist eine freie Entwicklungsumgebung (Integrated Development Environment – IDE), die besonders im Bereich der Java-basierten Anwendungsentwicklung verbreitet ist. Eclipse selbst ist ebenfalls in Java implementiert.

die Frage, was genau unter dem Begriff ESB verstanden werden kann. Wie die vorangegangenen Definitionen gezeigt haben, ist dies nicht eindeutig, wenngleich viele der Definitionen große Parallelen aufweisen. Es wird von Architektur, Konzept und Infrastruktur gesprochen und dann wieder von einem Bestandteil einer SOA. Typisches Beispiel sind die Äußerungen von Chappell, der den Begriff auf vielerlei Weise definiert (siehe Kapitel 2.2.3). Einzig überraschend ist, dass der ESB nirgends direkt als Produkt betrachtet wird, obwohl nicht wenige Produkte genau unter dieser Bezeichnung vertrieben werden.

Wie kann man dieses vielschichtige Bild ordnen? Eine Möglichkeit besteht darin, klar zwischen zweierlei Dingen zu unterscheiden, zum einen der Softwarearchitektur ESB – als ein Konzept auf dem Weg hin zu einer SOA – und zum anderen dem ESB als Implementierung dieser Architektur. Diese Unterscheidung ermöglicht, verschiedene Begriffsdefinitionen besser einzuordnen. Die Betrachtung des ESB als Architektur, Konzept oder Ansatz lässt sich der ersten Sichtweise zuordnen und Begriffe wie Software-/Nachrichtenbus, Bestandteil einer SOA oder Middleware-Switch eher der zweiten. In Folge der Dualität dieses Begriffs und zur Unterscheidung der beiden unterschiedlichen Betrachtungen ergibt sich die Notwendigkeit einer klaren sprachlichen Trennung. Für den weiteren Verlauf dieser Arbeit wird daher unter der Bezeichnung ESB eine Softwarearchitektur verstanden (als logisches Konzept) und unter ESB-Implementierung dessen physische Umsetzung (in Form von ESB-Produkten).

Weniger gespalten sind die Ansichten bezüglich des Zwecks, den ein ESB zu erfüllen hat. Es ist die Integration, genauer die Integration von unterschiedlichsten, hoch verteilten Anwendungen und Diensten im Umfeld heterogener Systeme und Plattformen.

Auf Grundlage der umfangreichen Äußerungen von Kischel und Chappell in Bezug auf die Aufgaben und funktionalen Eigenschaften eines ESB werden die wesentlichen in kurzer Form zusammenfassend aufgelistet. Dazu gehören:

- ***standardbasierte Kommunikation und Integration,***
- ***Unterstützung und Abbildung von (Geschäfts-)Prozessen bzw. Prozessketten,***
- ***Service Orchestrierung,***
- ***Service Choreographie,***
- ***Transformation,***
- ***intelligentes Routing,***
- ***Konfiguration, Monitoring und Management von Services,***
- ***Management von Servicelebenszyklen,***
- ***Sicherheit (QoP),***
- ***Dienstgüte (QoS),***
- ***Transaktionsverwaltung,***
- ***flexibles Deployment,***
- ***Skalierbarkeit,***
- ***Föderation und Erweiterbarkeit.***

Zur Erreichung der Zielstellungen und gemäß den aufgestellten Anforderungen muss der ESB neben XML als zentralem Standard und natives Datenformat eine Vielzahl weiterer Standards sowie proprietäre Protokolle und Formate unterstützen. Einige von diesen wurden bereits an verschiedenen Stellen genannt und sollen nun geordnet werden. Aufgrund der großen Anzahl an Standards, Protokollen und Formaten können nur die wesentlichsten und wichtigsten genannt werden. Besonders das breite Feld der WS-Standards entzieht sich einer vollständigen Betrachtung. Zum Zwecke einer besseren Gliederung wird die folgende Einteilung vorgenommen:

- **Web Service Standards (WS-Standards):** bspw. BPEL4WS, WS-Coordination oder WSDL (siehe Anhang Tabelle B.1).
- **XML-basierte Standards**[22]**:** XML Schema, XPath oder XSLT (siehe Anhang Tabelle B.2)[23].
- **Protokoll Standards:** bspw. HTTP, SMTP oder SOAP (siehe Anhang Tabelle B.3).
- **Java-basierte Standards:** bspw. JCA, JMS oder JMX (siehe Anhang Tabelle B.4).
- **Proprietäre Standards:** bspw. .NET, SonicMQ oder Webpsphere MQ (siehe Anhang Tabelle B.5).
- **Sonstige Standards:** bspw. CSV, IDL oder UDDI (siehe Anhang B Tabelle B.6).

Sämtliche als relevant betrachteten Standards sowie deren Bedeutung sind gemäß den Verweisen und aufgrund ihres Umfangs im Anhang B verzeichnet.

Der Aufbau eines ESB gliedert sich in zwei Schichten. Eine, die physische Schicht des ESB, basierend auf entsprechender nachrichtenbasierter Infrastruktur (bei Kischel als Transportschicht bezeichnet) und eine darüber liegende Prozessschicht (bei Kischel als DPF bezeichnet). Als ein weiterer wesentlicher Bestandteil stellen sich in diesem Zusammenhang die so genannten Service-Container dar. Sie bilden, von der darunter liegenden Middleware-Infrastruktur abgesehen, den eigentlichen ESB. Viele Aufgaben, die ein ESB zu erfüllen hat, werden mittels dieser Container realisiert. Weiterhin sind es die Service-Container, die Dienste und Anwendungen kapseln und in Form abstrakter Endpunkte (Eintritts- wie Austrittsendpunkte) der Prozessschicht zur Verfügung stellen. Der beispielhafte Aufbau eines solchen Containers (nach Chappell) ist in Abbildung 2.7 dargestellt. Sehr gut zu sehen ist darin das Management Framework, das zur Steuerung der ESB-Service-Container dient. Die dafür notwendige Kommunikation (sowohl für Management als auch für Metadatenaustausch) wird über JMX realisiert. Das Ansprechen der innerhalb dieser Container residierenden Services erfolgt über Eintritts- und Austrittsendpunkte und ggf. notwendige Adapter.

In Bezug auf Konzepte wie EAI, SOA und Technologien wie MOM und Integrationsserver ist der ESB in folgender Weise einzuordnen. Von MOM und Integrationsservern (EAI) erbt der ESB Fähigkeiten zur Nachrichtenübermittlung, losen Kopplung, Integration und Trennung von Anwendungs- und Integrationslogik, kombiniert und erweitert diese aber so, dass er Schwächen dieser Konzepte beseitigt. Die Summe der dadurch bereitgestellten Eigenschaften und Funktionalitäten befähigt den ESB, eine zentrale Rolle bei der Implementierung einer SOA einzunehmen.

[22] Hierzu werden auch Standards gerechnet, die als solche nicht XML-konform sind, jedoch hauptsächlich im Kontext von XML Anwendung finden.

[23] Viele dieser Standards gehen entweder auf das World Wide Web Consortium (W3C) oder auf die Organization for the Advancement of Structured Information Standards (OASIS) als Standardisierungsgremium zurück.

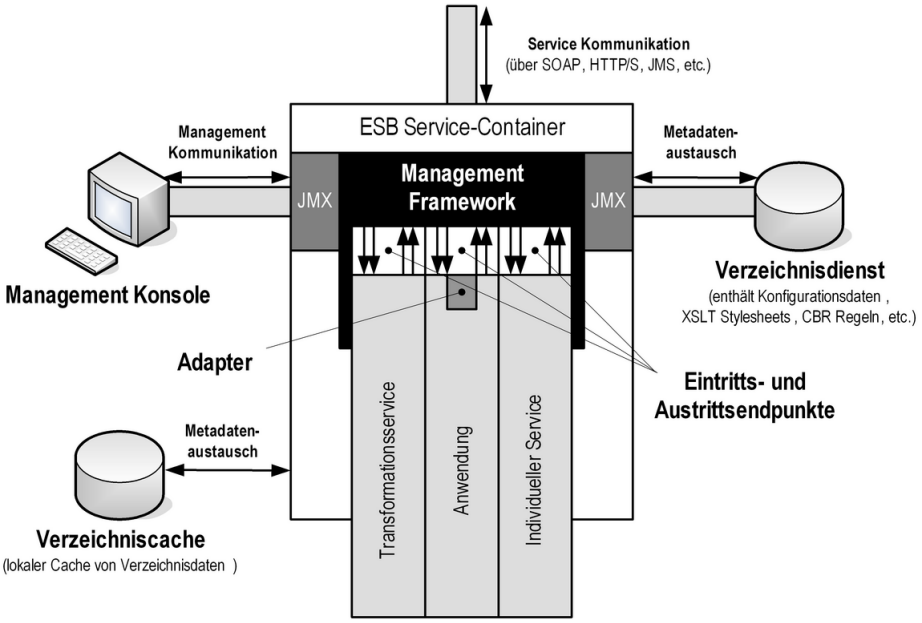

Abbildung 2.7: ESB-Service-Container (in Anlehnung an [Cha04b], S. 113).

2.4 Fazit

Betrachtet man die verschiedenen Definitionen, so kann eines festgestellt werden. Trotz eines vielfältigen Begriffsverständnisses besteht im Wesentlichen Konsens über den Zweck und die Aufgaben des ESB, nämlich auf MOM aufbauend eine entsprechende Integrationsinfrastruktur bereitzustellen. Ähnlich verhält es sich mit seinen Eigenschaften, wenngleich diese unterschiedlich detailliert beschrieben werden. Zurückkommend auf den eigentlichen Begriff ESB muss unterschieden werden zwischen dem ESB als Architekturkonzept und dem ESB als Implementierung einer solchen Architektur. Beides wird als ESB bezeichnet, ist aber aus wissenschaftlicher Sicht klar voneinander zu trennen. Aus diesem Grund werden im Rahmen dieser Arbeit die beiden Begriffe ESB (Architekturkonzept) und ESB-Implementierung verwendet. Als Hauptaufgaben bzw. Eigenschaften einer ESB-Implementierung können neben der standardbasierten Kommunikation und Integration vor allem Orchestrierung, Choreographie, Umsetzung von Prozessketten, Transformation und Routing genannt werden. Die Frage betreffend, wie eine Umsetzung dieser Aufgaben erfolgt und welche weiteren Technologien neben den bereits genannten eingesetzt werden, ist Kern des nächsten Kapitels.

Kapitel 3

Der ESB aus Sicht der Softwareindustrie

3.1 Herangehensweise

Neben der in Kapitel 2 beschriebenen Sicht der Forschung auf den ESB wird nun eine Betrachtung des ESB in der Softwareindustrie vorgenommen. Es sind weniger Begriffsdefinitionen der Softwarehersteller als vielmehr die angebotenen Produkte (ESB-Implementierungen), die dabei verglichen und bewertet werden sollen. Der Markt angebotener ESB-Produkte ist genauso mannigfaltig, wie die Zahl der Hersteller derartiger Software. Werden an einer Stelle dem Namen nach reine ESB vertrieben, so werden an anderer Stelle Produkte beworben, die einen ESB beinhalten oder sogar über erweiterte ESB verfügen sollen. Es ist daher auch an dieser Stelle zweckmäßig, analog zu der Vorgehensweise bei der Untersuchung unterschiedlicher ESB Definitionen, exakte Vergleichskriterien aufzustellen. Solche Kriterien sind:

- **Bestandteile:** Aus welchen Komponenten setzt sich der ESB zusammen?
- **Aufbau:** Welche Architektur liegt dem ESB zugrunde?
- **Funktionsumfang:** Welche Funktionen stellt der ESB bereit?
- **Flexibilität:** Wie anpass- und kombinierbar sind ESB?
- **Definitionskonformität:** In wieweit erfüllt der ESB die Forderungen der Definition?
- **Qualitätseigenschaften:** Über welche Qualitätseigenschaften verfügt der ESB?

Eine genaue Betrachtung des ESB-Marktes zeigt dessen rasantes Wachstum mit der Folge, dass zunehmend etablierte IT-Anbieter wie bspw. IBM, Sun Microsystems und nicht zuletzt Microsoft ein starkes Interesse bekunden. Mit immensen Budgets wird hier der Versuch unternommen, auch in diesem Markt Fuß zu fassen und bisherige ESB Spezialisten zu verdrängen. Die Qualität dieser Produkte bzw. deren Eignung als ESB ist dabei fraglich, soll jedoch im Rahmen dieser Arbeit genauer untersucht werden. Eine möglichst umfassende, jedoch nicht den Anspruch auf Vollständigkeit erhebende Liste von Anbietern und Produkten ist im Anhang C Tabelle C.1 zu finden. Aufgrund dieser Vielzahl und um den Rahmen dieser Arbeit einzuhalten, musste eine Auswahl erfolgen. Berücksichtigt werden soll dabei:

- Rolle des Anbieters/Unternehmens bei der Entstehung des ESB Marktes,
- Bedeutung des Anbieters/Unternehmens auf dem Markt angebotener ESB- und SOA-Produkte,
- Bedeutung des Anbieters/Unternehmens auf dem gesamten IT-Markt[1],
- Rolle des Anbieters/Unternehmens im Open Source[2] Umfeld.

In einer Marktstudie von 2005 sieht Forrester den ESB Markt in zwei Segmente geteilt. Ein Segment besteht aus Anbietern von einfachen ESB, d. h. von Produkten, die sich auf die Funktionalität eines ESB beschränken und ein anderes Segment aus umfassenden bzw. erweiterten ESB also Produkten, die in der Regel weit über den Funktionsumfang herkömmlicher ESB hinausgehen. Die Ursachen für eine solche Zweiteilung sind vor allem auf die unterschiedlichen Hintergründe der jeweiligen Softwarehersteller zurückzuführen. Im ersten Segment sind überwiegend ESB Spezialisten vertreten, deren Wurzeln häufig im Bereich Web Service-basierter Infrastrukturen zu finden sind. Unternehmen, die hier eine Rolle spielen, sind vor allem BEA Systems, Cape Clear, Fiorano, IONA Technologies, PolarLake und Progress Software (ehemals Sonic Software) (vgl. [VG05], S. 4 ff.).

Das zweite Segment bildet sich aus Anbietern kompletter Integrationsplattformen und basiert sehr oft auf deren früheren EAI- oder Integrationsprodukten. Genannt werden können hier BEA Systems, Oracle, Sun Microsystems, TIBCO und webMethods. Anzumerken ist, dass BEA Systems als Anbieter einfacher und umfassender ESB in beiden Segmenten vertreten ist. Welche Bedeutung die genannten Unternehmen in den jeweiligen Marktsegmenten haben, ist Abbildung D.1 und Abbildung D.2 (siehe Anhang D) zu entnehmen (vgl. [VG05], S. 4 ff.).

In einer neueren Studie aus dem Jahr 2006 sieht Forrester Cape Clear Software, BEA Systems, Fiorano, IBM, IONA Technologies, Polar Lake, Progress Software und Software AG als die Hauptvertreter für ESB-Produkte. Unberücksichtigt bleiben in dieser Studie die Anbieter erweiterter ESB bzw. Integrationsplattformen mit ESB-Komponente. Im Vergleich zur Studie aus dem Jahr 2005 sind zwei neue Anbieter dazugekommen. Diese sind IBM und mit Software AG auch ein deutsches Unternehmen (vgl. [VG06], S. 4 ff.). Wie Forrester diese Unternehmen bezüglich Marktpräsenz, Leistungsangebot und Strategie einschätzt, ist in Abbildung D.3 (siehe Anhang D) dargestellt.

Auf Grundlage der Marktanalysen von Forrester und basierend auf den zuvor genannten Kriterien sollen im weiteren Verlauf dieser Arbeit die Produkte folgender Softwarehersteller und -anbieter betrachtet werden[3]:

- **Progress Software:** Der erste auf dem Markt verfügbare ESB stammt von Progress Software (ehemals Sonic Software). Das rasante Wachstum dieser Branche insbesondere in frühen Jahren ist nicht zuletzt auf dieses Unternehmen zurückzuführen. Seither gehört Progress Software im Bereich ESB zu einem etablierten Unternehmen, das wie die Branche selbst noch immer im Wachstum begriffen ist (vgl. [VG06], S. 11).
- **Fiorano Software:** Fiorano Software gehört wie Progress Software mit zu einem der ersten Unternehmen auf dem ESB-Markt. Als eher kleiner Softwarehersteller ist es dennoch gelungen, sich mit Erfolg am Markt zu behaupten (vgl. [VG06], S. 11).

[1] Als ein Indikator dienen hier wirtschaftliche Kennzahlen wie Marktkapitalisierung und Umsatz. Die konkreten Daten eines Großteils der ESB-Anbieter sind (soweit vorhanden) im Anhang in Tabelle C.2 zu finden.
[2] Als Open Source wird Software bezeichnet, deren Quellen in offener Form vorliegen.
[3] Aufgrund des Umfangs werden einige Produkte erst in Kapitel 5 im Rahmen der Bewertung von frei verfügbaren ESB-Implementierungen näher betrachtet.

- **Cape Clear:** Dritter im Bunde der sehr frühen ESB-Anbieter und unter diesen wohl erfolgreichster ist Cape Clear. Mit einer gewachsenen Angebotspalette und der Unterstützung für Service Orchestration kann Cape Clear, gemessen an seiner Größe, eine beachtliche Marktpräsenz vorweisen (vgl. [VG06], S. 10).

- **IONA Technologies**: IONA Technologies hat als langjähriger Middleware Anbieter 2004 erstmals auch einen ESB in sein Produktportfolio aufgenommen. Gestützt auf das Know how aus dem Bereich der Middleware verfügt man mit Artix über einen durchaus leistungsfähigen ESB. Zusätzlich und im Gegensatz zu manch anderem ESB-Anbieter wird mit Celtix auch eine Open Source ESB-Implementierung angeboten. Einige kleinere Schwächen verhindern jedoch den Sprung an die oberste Spitze (vgl. [VG06], S. 11).

- **BEA Systems:** Mit BEA Systems und ihrem BEA AquaLogic Service Bus hat im Jahre 2005 erstmals einer der größeren Softwarehersteller den Schritt auf den ESB-Markt gewagt. Trotz einiger kleiner Schwächen im Vergleich zu anderen, ausgereifteren Produkten hat BEA ziemlich schnell den Weg an die Spitze dieses Marktes gefunden (vgl. [VG06], S. 10).

- **Oracle:** Als wohl größter Anbieter auf dem Markt der erweiterten ESB kann Oracle gesehen werden. Mit einer aggressiven Strategie versucht Oracle seine hervorgehobene Stellung auf dem Markt angebotener Integrationslösungen nun auch auf den ESB-Markt auszudehnen (vgl. [VG05], S. 12).

- **MuleSource:** MuleSource bietet mit Mule (einem im Jahre 2003 ins Leben gerufenen Projekt) eine Open Source ESB-Implementierung an. Sowohl Dank eines reichhaltigen Funktionsumfangs als auch mehrjähriger Erfahrung gehört Mule wohl zu den ausgereifteren quellcodeoffenen ESB-Implementierungen.

- **Microsoft:** Beim wohl größten heutigen Softwareunternehmen[4] Microsoft bleibt die Entwicklung im Bereich SOA und ESB nicht unbeobachtet. Obwohl man hier noch über keinen ESB als solchen verfügt, glaubt man doch, mit dem BizTalk Server diesem sehr nahe zu kommen (vgl. [Nic05]; [Mic06b]).

- **Sun Microsystems:** Neben vielen anderen renommierten Softwareherstellern versucht auch Sun Microsystems sich auf dem Gebiet von Integrationssoftware zu positionieren. In dieses Konzept gehört der Aufkauf von SeeBeyond und seiner ICAN Technologie (vgl. [VG05], S. 12). Eine weitere ESB-Implementierung von Sun ist der Java-basierte Open ESB. Zielstellung, dieser sich noch in der Entwicklung befindenden ESB-Implementierung ist, fundierend auf JBI, einen quellcodeoffenen und hochleistungsfähigen ESB zu entwickeln (vgl. [Sun06a]).

- **JBoss ESB:** JBoss, als langjähriger Anbieter des gleichnamigen und ebenfalls quellcodeoffenen Applikationsservers, versucht mit dem JBoss ESB seine Produktpalette um eine ESB-Implementierung zu erweitern. Unter den bereits genannten Open Source ESB-Implementierungen befindet sich der JBoss ESB trotz eines ausgereiften Konzepts ähnlich wie der Open ESB noch in einem sehr frühen Entwicklungszustand und liegt daher auch in keiner endgültigen Release[5] vor.

[4] Gemessen an der Marktkapitalisierung (siehe Tabelle C.2 im Anhang).
[5] Bezeichnet eine neue Produktversion, die zur Veröffentlichung freigegeben wurde.

3.2 JBI Spezifikation

Bevor auf konkrete ESB-Implementierungen einzelner Softwarehersteller eingegangen werden soll, erfolgt an dieser Stelle eine Darstellung der schon bei Chappell erwähnten JBI Spezifikation. Obwohl unter der Federführung von Sun Microsystems entwickelt, ist sie für verschiedene Implementierungen und damit auch in allgemeiner Weise von Bedeutung. Entstanden ist diese Spezifikation im Rahmen des Java Community Process (JCP) und wird daher auch als Java Specification Request (JSR) 208 bezeichnet.

JBI ist in erster Linie eine Spezifikation bezüglich der Architektur von Service-Containern. Abbildung 3.1 zeigt ihren beispielhaften Aufbau. Zu den drei wichtigsten Bestandteilen gehören ein Component Framework, ein Normalized Message Router (NMR) und ein Management Framework. Im Zusammenhang mit dem Component Framework wird zwischen zwei Komponententypen unterschieden. Der erste Typ, die Service Engine Components (SE), sind für Geschäftslogik verantwortlich und können von einfachen Transformationen über die Realisierung komplexer BPEL-basierter Geschäftsprozesse bis hin zur Bereitstellung sonstiger Anwendungslogik reichen. Daneben gibt es Binding Components (BC), deren Hauptaufgabe in der Bereitstellung unterschiedlicher Bindungen (bindings) und Transportprotokolle besteht. Sie werden in der Regel von verschiedenen SE genutzt und stellen Stellvertreter (Proxies) für externe wie interne Services dar. Das Ziel, das mit diesem Aufbau verfolgt wird, ist, Geschäftslogik von Infrastrukturdetails zu entkoppeln. Gleichfalls entkoppelnd wirken soll der Einsatz von Standard Service Provider Interfaces (SPI), die SE und BC zwischengeschaltet sind. Beide Komponententypen SE und BC werden mittels WSDL beschrieben und können entweder als Dienstanbieter oder Dienstkonsumenten auftreten. Die dafür notwendige Kommunikation erfolgt in einem normalisierten Nachrichtenformat bestehend aus Eigenschaften (Message Properties), Inhalt/Nutzlast (Message Payload) und Anhängen (Message Attachments) mittels Normalized Message Router. Gesteuert werden JBI-Container über das bereits genannte Management Framework. Dieses erlaubt es über JMX sowohl die JBI-Laufzeitumgebung (den JBI-Container selbst) als auch die darin gehaltenen Services zu verwalten (vgl. [Che05], S. 15 ff.;[Kin05]).

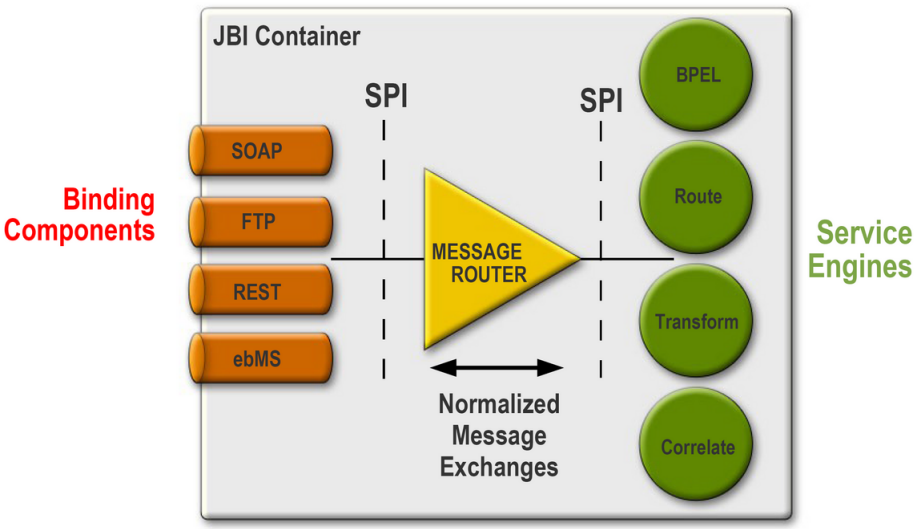

Abbildung 3.1: Service-Container nach JBI (Quelle: [Che05], S. 16).

Diese standardisierte Architektur gestattet es, relativ einfach sowohl SE als auch BC zwischen unterschiedlichen JBI-Implementierungen auszutauschen. Denkbar wird damit sogar, eine gesamte Integrationsinfrastruktur auf leichte Weise zwischen unterschiedlichen JBI-konformen ESB-Implementierungen zu portieren. Zu bereits heute verfügbaren JBI-basierten ESB-Implementierungen gehören Celtix, Mule, ServiceMix und der Open ESB.

3.3 ESB nach Progress Software (ehemals Sonic Software)

Im Jahre 2002 tauchte der Begriff ESB erstmals bei Sonic Software, heute Progress Software, auf. Progress Software war somit der erste Anbieter einer ESB-Implementierung, dem Sonic ESB. Mit der Sonic ESB Produktfamilie wurde um den Sonic ESB herum eine Reihe von Komplementärprodukten aufgestellt, die jeweils Teilaspekte einer umfassenden ESB-Implementierung abdecken (siehe Abbildung 3.2). Zu den *Bestandteilen* dieser Produktfamilie gehört die Sonic Integration Workbench IDE[6], der Sonic Orchestration Server, der Sonic XML Server, der Sonic Database Service und der eigentliche Sonic ESB (vgl. [Son06b], S. 9).

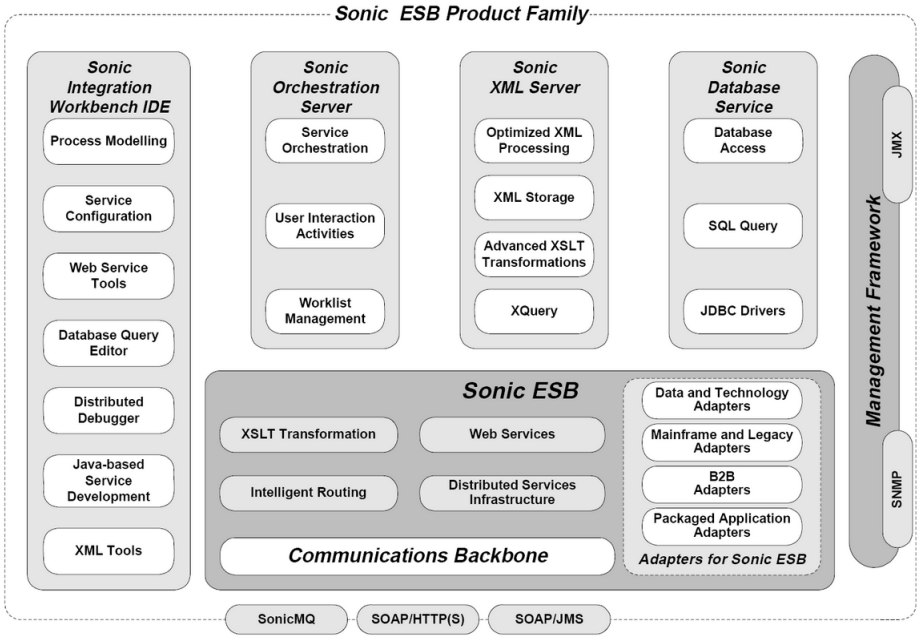

Abbildung 3.2: Sonic ESB Produktfamilie (Quelle: [Son06b], S. 9).

Die Aufgabe der Sonic Integration Workbench IDE ist es, eine werkzeugbasierte grafische Unterstützung zur Prozessmodellierung, Konfiguration, Testdurchführung/Debugging und Verteilung/Auslieferung zu liefern. Dafür werden verschiedene Editoren bereitgestellt. Die Sonic Integration Workbench IDE fußt auf Eclipse (als Plugin[7] integriert) und unterstützt

[6] IDE steht für Integrated Development Environment und meint eine integrierte Entwicklungsumgebung, d. h. ein Werkzeug aus dem Bereich Computer-Aided Software Engineering (CASE).

[7] Ein Plugin ist eine für ein bestimmtes Programm entwickelte, erweiternde Komponente, die problemlos in dieses integriert (eingeklinkt) werden kann.

sämtliche Produkte der Sonic ESB Produktfamilie. Zu den Stärken gehört u. a. die grafische Unterstützung (speziell in Bezug auf die Konfiguration), die den nötigen Programmieraufwand auf ein Minimum reduziert und die Möglichkeit eines Debuggings von Prozessen, die sich über mehrere und hoch verteilte Services erstrecken können (vgl. [Son06b], S. 18 ff.).

Der Sonic Orchestration Server erweitert den Sonic ESB und seine Fähigkeiten zum intelligenten Routing um die Modellierung, Automation und Management komplexer, zustandsbehafteter Geschäftsprozesse entlang erweiterter Unternehmen. Während der gesamten Laufzeit eines solchen zustandsbehafteten und möglicherweise lang andauernden Prozesses, kann auf die darin enthaltenen Geschäftsdaten zugegriffen werden. Diese liegen typischerweise in XML vor und werden in einem integrierten Datenspeicher gehalten. Die im Sonic Orchestration Server enthaltene Ausführungsumgebung unterstützt die gleichzeitige Ausführung mehrerer Prozesse und für jeden Prozess die Ausführung mehrerer Instanzen dieses Prozesses (vgl. [Son06b], S. 16).

Mit dem Sonic XML Server wird ein hochleistungsfähiges Management zur Bearbeitung, Speicherung und Anfrage von XML-Daten zur Verfügung gestellt. Darüber hinaus kann er zur Steigerung der Leistungsfähigkeit (Performanz), zur Unterstützung einer Ereignissteuerung in Bezug auf Geschäftsprozesse oder als Grundlage für Protokollierung und Überwachung entlang des Sonic ESB verteilt und eingesetzt werden. Kern des Sonic XML Servers ist eine Engine zur XML-Bearbeitung, die eine erweiterte und standardgerechte Datentransformation etwa mittels XQuery oder XSLT unterstützt. Auf diesem Weg wird ein effizienter Austausch von XML-Daten zwischen verschiedenen Partnern ermöglicht, genau wie diese auch zur Laufzeit analysiert werden können (vgl. [Son06b], S. 17).

Eine weitere Komponente bildet der Sonic Database Service, der den Zugriff und die Wiederverwendung von Datenquellen im Rahmen des ESB vereinfachen soll. Erleichtert wird vor allem die Konfiguration und Ausführung von Anfragen, Updates und gespeicherten Prozeduren (Stored Procedures). Die Ausführung von Datenbankoperationen kann dabei durch Nachrichten oder eine Zeitsteuerung ausgelöst werden. Abschließend wird das Ergebnis einer solchen Operation zu einem XML-Dokument transformiert und in Form einer Nachricht bereitgestellt. Technische Grundlage für den Aufbau von Datenbankverbindungen bilden entsprechende JDBC Treiber (vgl. [Son06b], S. 15).

Das eigentliche Kernelement dieser Produktfamilie ist der Sonic ESB. Wie man den bisherigen Ausführungen entnehmen konnte, werden bei Sonic viele Funktionalitäten in Form von komplementären Produkten ausgegliedert. Zu den Aufgaben, die im eigentlichen ESB verbleiben, gehören die Transformation von Daten, intelligentes Routing, Integration von Adaptern zum Anschluss von unterschiedlichsten (proprietären) Anwendungen und Systemen sowie die Bereitstellung einer verteilten Service Infrastruktur auf Basis eines entsprechenden Kommunikationsgerüsts (Communication Backbone) (vgl. [Son06b], S. 9).

Der grundsätzliche *Aufbau* des Sonic ESB ähnelt dem, wie er bereits im vorangegangenen Kapitel beschrieben wurde. Über entsprechende Services werden (wie in Abbildung 3.3 gezeigt) unterschiedliche Anwendungen und Systeme (J2EE Anwendungen, Legacy Systeme, .NET Anwendungen, Partnersysteme) miteinander verknüpft. Der einzelne Service dient hierbei (soweit erforderlich) als Kapselung (Wrapper) für die jeweilige Anwendung oder das Hintergrundsystem. Web Services müssen dabei nicht gekapselt werden, sondern können direkt angeschlossen werden. Ein solcher Anschluss erfolgt über Service-Container. In diesen residieren die jeweiligen Services und werden hier auch verwaltet. Über entsprechende MOM, die das Gerüst für die Datenübertragung bildet (Datenübertragungsbackbone), erfolgt dann der eigentliche Nachrichtenaustausch (vgl. [Son05a], S. 1 ff.).

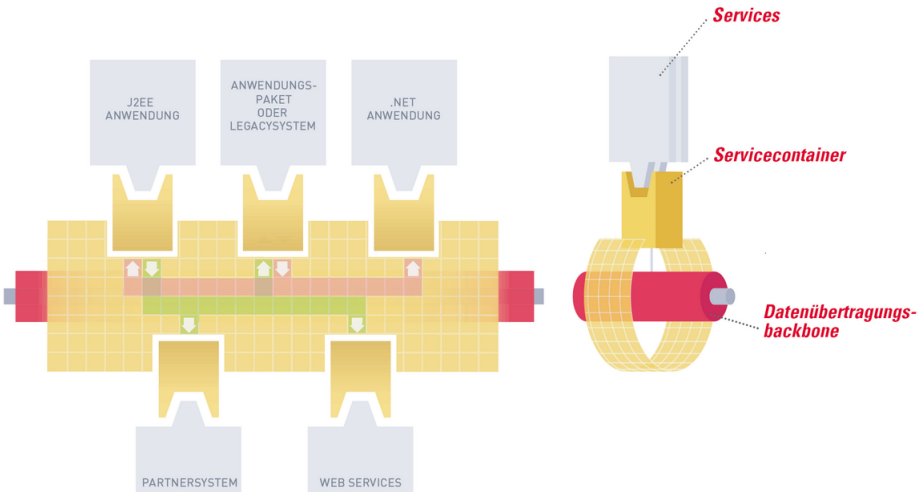

Abbildung 3.3: Sonic ESB-Architektur (Quelle: [Son05a], S. 5).

Zentrales Element des Sonic ESB sind die jeweiligen Services, die entweder in direkter Form vorliegen oder als Proxies[8] Verbindungen zu anderen IT-Ressourcen kapseln. Services können vom Typ her sehr unterschiedlich sein. Es gibt Services zur Mediation[9], wie XML Transformation oder CBR, benutzerspezifische Services und erweiterte Services etwa für Orchestrierung, unternehmensübergreifende Zusammenarbeit (Collaboration), XML-Verarbeitung oder Datenbankanbindung. Hervorzuheben, weil bisher wenig thematisiert, sind die Services zur Orchestrierung und unternehmensübergreifenden Zusammenarbeit. Im Bereich der Orchestrierung ist der Ausgangspunkt die Definition von Aktivitäten eines Prozesses. Hierfür können mittels der Sonic Integration Workbench IDE UML Aktivitätsdiagramme modelliert und in Form von Services umgesetzt werden. Darüber hinaus werden seitens des Sonic Orchestration Server auch BPEL4WS-basierte Prozessmodelle unterstützt. Auf dem Gebiet der unternehmensübergreifenden Zusammenarbeit geht es vor allem um die Unterstützung von Protokollen aus dem B2B-Bereich, wie etwa RosettaNet und ebXML aber auch BPEL4WS. Ähnlich wie bei der Orchestrierung werden auch hier Prozesse durch klar definierte Aktivitäten spezifiziert (Aktivitätsdiagramme) (vgl. [Son05b], S. 14 ff.; [Son06b], S. 10).

Neben den Services gibt es auch Service-Container, die als Verwaltungsschnittstelle für Services dienen und für diese eine entsprechende Umgebung bereitstellen. Hauptaufgaben dieser Container sind die Verwaltung der Servicelebenszyklen oder auch die Verwaltung mehrerer Threads (Instanzen) eines Services aus Gründen der Skalierung. Sie sind leichtgewichtig und erfordern zur Bereitstellung von Services nicht die Installation großer, monolithischer Applikationsserver (vgl. [Son05b], S. 12; [Son06b], S. 10).

Die eigentliche Kommunikation wird durch einen multiprotokollfähigen Kommunikationsbroker (SonicMQ) übernommen, der die Infrastruktur zum Nachrichtentransport bildet. Durch die Möglichkeit zur Bildung von Clustern[10] stellt er sich als überaus zuverlässig und skalierbar dar. Als Protokolle werden JMS, HTTP, HTTPS und SOAP unterstützt, wobei JMS bevorzugt wird. Nachrichten können damit asynchron und dem Prinzip der losen Kopplung

[8] Als Proxy wird Software bezeichnet, die stellvertretend für andere agiert. In der Regel wird dabei der anfallende Datenverkehr an die, mittels Proxy vertretene, Software weitergereicht.
[9] Mediation bezeichnet in diesem Zusammenhang die Integration und Aggregation von Schnittstellen, die über einen Mediator in einer vereinheitlichenden Weise angesprochen werden können.
[10] Ein Computercluster oder einfach Cluster bezeichnet einen Verbund bzw. eine Gruppe miteinander vernetzter Rechner, die eine logische Einheit bilden (vgl. [Ben04], S. 3).

folgend verschickt werden. Mittels Queues und Topics erfolgt dann die Zustellung zu den jeweiligen Empfängern. Eine weitere Aufgabe des Kommunikationsbrokers ist die Gewährleistung der notwendigen Sicherheit (vgl. [Son05b], S. 19 f.; [Son06b], S. 11). Dazu gehören (vgl. [Son05b], S. 25):

- **Authentifizierung:** Die Authentifizierung betrifft Sender wie Empfänger einer Nachricht. Beide können sich entweder mittels Benutzername und Passwort authentifizieren oder dies über externe Sicherheitsinstanzen tun (Security Repositories).
- **Autorisation:** Mittels ACL[11] werden Nutzer bzw. Nutzergruppen autorisiert, Nachrichten bestimmter Absender zu erhalten oder an bestimmte Empfänger zu senden.
- **Verschlüsselung:** Die Verschlüsselung kann zum einen auf der Protokollebene (SSL oder HTTPS) als auch auf der Nachrichtenebene durch eine gesonderte Verschlüsselungskomponente erfolgen. Unterstützt werden verschiedene Verschlüsselungsalgorithmen so auch der Advanced Encryption Standard (AES).
- **Integrität:** Die Nutzung von Message Authentication Codes (MAC) stellt die Integrität von Nachrichten sicher.

Im Rahmen der Unterstützung von Web Services wird WSDL zur Schnittstellenbeschreibung verwendet. Diese Beschreibung kann entweder in einem ESB-Repository oder in einer UDDI-Registry gespeichert werden. Weitere Standards, die durch den Kommunikationsbroker unterstützt werden, sind WS-ReliableMessaging, WS-Adressing, WS-Security und WS-Policy (vgl. [Son05b], S. 22 f.).

Das Zusammenspiel der einzelnen vorgestellten Komponenten des Sonic ESB ist in Abbildung 3.4 dargestellt. Darüber hinaus enthalten ist eine der Überwachung dienende Management Konsole. Sie ist Teil des so genannten Management Frameworks, welches zum Sonic ESB dazugehört. Dieses stellt mittels JMX und SMTP den Rahmen zu dessen Steuerung und seiner Komponenten bereit (vgl. [Son06b], S. 9).

Der *Funktionsumfang* erstreckt sich von zahlreichen, unterstützten Standards über Transformation, intelligentem Routing bis hin zu Sicherheit und Skalierbarkeit. Nahezu alle Aufgaben einer ESB-Implementierung kann der Sonic ESB lösen, einzig auf die Punkte Service Choreographie und Transaktionsverwaltung wird nicht konkret eingegangen. Gründe dafür mögen sein, dass die Service Choreographie der Service Orchestrierung zugerechnet wurde und dass eine Transaktionsverwaltung über die MOM-Plattform SonicMQ bzw. beschränkt mit der Umsetzung von JMS und WS-ReliableMessaging erreicht werden kann[12].

In Puncto *Flexibilität* stellt der Sonic ESB mit seinen mehr als 300 Adaptern, die von der Unterstützung von ERP[13]-, SCM[14]- oder CRM[15]-Systemen (Oracle, Peoplesoft, SAP) über Nachrichtenadapter (MSMQ, Oracle AQ) und Adapter für Mainframe- und Altanwendungen bis hin zur Integration spezieller Datenformate reichen (EDI, cXML, xCBL), größtmögliche Konnektivität bereit. Weiterhin werden zahlreiche Betriebssysteme unterstützt (Microsoft Windows, Solaris/SunOS, Red Hat Linux, u. a.), die in Bezug auf die Verteilung und Skalierung der leichtgewichtigen Service-Container ein Höchstmaß an Flexibilität bieten. Auf

[11] ACL – Access Control List
[12] JMS bietet mittels XAConnection eine Unterstützung für verteilte Transaktionen und kann genau wie WS-ReliableMessaging zumindest einen zuverlässigen Nachrichtentransport sicherstellen.
[13] ERP – Enterprise Resource Planning
[14] SCM – Supply Chain Management
[15] CRM – Customer Relationship Management

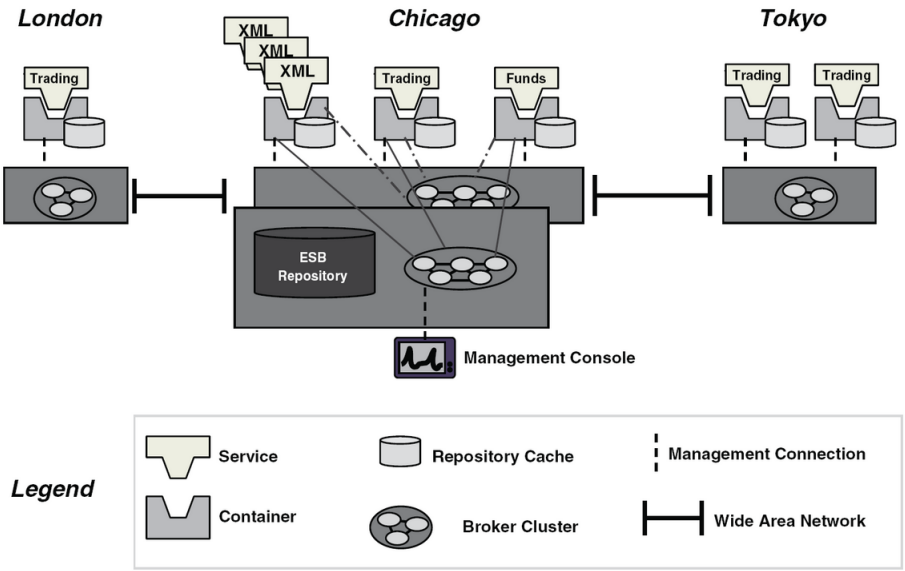

Abbildung 3.4: Sonic ESB Beispiel (Quelle: [Son05b], S. 37).

gleiche Weise flexibel skalierbar und dementsprechend leistungsfähig sind die Kommunikationsbroker zum eigentlichen Nachrichtentransport (vgl. [Son06a], S. 2 f.).

Der Aufbau wie auch die durch die Produktfamilie abgedeckten Funktionalitäten stehen in keinem Widerspruch zur Definition eines ESB, weshalb der Sonic ESB (bzw. die gesamte Produktfamilie) folglich als *definitionskonform* zu bezeichnen ist.

Als besondere *Qualitätseigenschaften* hervorzuheben sind die grafische Unterstützung, welche den Einsatz des Sonic ESB erleichtert und den Programmieraufwand auf ein Minimum reduziert, die umfangreichen Sicherheitsmaßnahmen, die Qualität der Unterstützung von (Geschäfts-)Prozessen sowie auch die professionellen Möglichkeiten zur XML-Bearbeitung/-Verwaltung. Betrachtet man die Sonic ESB Produktfamilie im Ganzen und den Sonic ESB im Speziellen, so ergibt sich das Bild einer als ausgereift und leistungsfähig zu bezeichnenden ESB-Implementierung, die überzeugen kann.

3.4 ESB nach Fiorano

Wie auch bei anderen traditionellen ESB-Anbietern hat man bei Fiorano weit mehr im Produktangebot als eine einfache ESB-Implementierung. Der Fiorano ESB ist daher nur eine Teilkomponente innerhalb der Fiorano SOA Plattform. Diese Plattform, in deren Zentrum der Fiorano ESB steht, soll sowohl den Aufbau einer SOA als auch den einer EDA[16] unterstützen. Zu den weiteren Bestandteilen gehören FioranoMQ, Fiorano BPEL und Fiorano Business Components & Adapters.

Diese *Bestandeile* sowie ihr Aufbau sind in Abbildung 3.5 dargestellt. Unterschieden werden dabei drei wesentliche Schichten. Die unterste Schicht (Distributed Enterprise IT Infrastructure) stellt die unterschiedlichen, physichen IT-Ressourcen dar. Diese reichen von Daten-

[16] EDA – Event Driven Architecture

banken, ERP-Systemen bis hin zu Altanwendungen oder Web Portalen. Darüber liegt die eigentliche ESB Schicht, die mittels Adaptern in der Lage ist, Anwendungen zu verbinden und ggf. in Form von Services zu kapseln. Die ESB Schicht stellt eine Abstraktion von den tatsächlichen physischen Ressourcen dar und führt zu einer logischen Sichtweise über die vorhandenen Ressourcen. Realisiert wird diese Abstraktion durch den Fiorano ESB bzw. die integrierte, JMS-basierte FioranoMQ als entsprechende Kommunikations- und Nachrichteninfrastruktur. Ebenfalls beteiligt sind die mittels Fiorano Business Components & Adapters bereitgestellten Adapter zur Anbindung heterogener IT-Ressourcen. Eine weitere Abstraktionsstufe bildet die Prozess- und Verwaltungsschicht (Process Management and Monitoring). Auf dieser Ebene geht es vor allem um die Unterstützung und Verwaltung von Geschäftsprozessen wie auch die Überwachung und Kontrolle des darunter liegenden ESB über entsprechende Managementschnittstellen. Als Werkzeuge eingesetzt werden auf dieser Ebene bspw. die Fiorano Process Orchestration Tools und der Fiorano BPEL Editor (vgl. [Fio05b], S. 1 ff.).

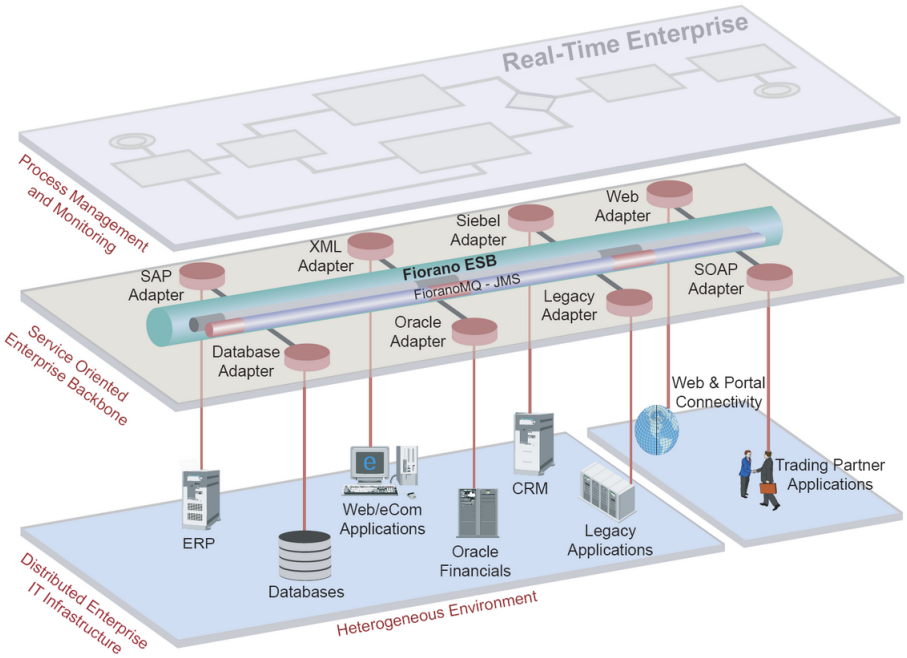

Abbildung 3.5: Fiorano SOA Plattform (Quelle: [Fio05b], S. 1).

Neben dem Fiorano ESB ist parallel dazu FioranoMQ als wesentliche Kernkomponente der gesamten Fiorano SOA Plattform zu nennen. FioranoMQ bildet das erforderliche Gerüst zum Austausch von Nachrichten entlang des ESB. Zu den Fähigkeiten gehören (vgl. [Fio04b], S. 1 f.):

- **Standardbasierter Nachrichtentransport:** Zu den durch FioranoMQ unterstützten Nachrichten- und Transportstandards gehören JMS, JCA, XML-basierter Nachrichtenaustausch, SOAP, HTTP, HTTPS und die Anbindung anderer nachrichtenbasierter Infrastruktur wie MSMQ, Websphere MQ oder TIBCO Rendezvous.
- **Hohe Verfügbarkeit:** Die Möglichkeit, FioranoMQ in einem Cluster zu betreiben, gewährleistet eine anpassbare Lastverteilung und zugleich eine gesteigerte Ausfallsicherheit.

- **Sicherheit:** Die notwendige Sicherheit wird auf dreierlei Wegen gewährleistet. Erstens wird mittels Java-basierter API[17] die Authentifizierung und Autorisation sichergestellt. Der Anschluss externer LDAP[18] Verzeichnisse oder Datenbanken wird in diesem Zusammenhang ebenfalls unterstützt. Die zweite Komponente betrifft den sicheren Transport, der sich bspw. mittels JSSE[19] oder SSL und HTTPS realisieren lässt. Abschließend kann zusätzlich eine Nachrichtenverschlüsselung erfolgen, um Datensicherheit zu gewährleisten.

Die Verwaltung erfolgt mit dem FioranoMQ Administration Tool, das eine GUI zur Steuerung sämtlicher FioranoMQ-Installationen bereitstellt. FioranoMQ als skalierbares bzw. über mehrere Server verteilbares Kommunikationsgerüst bildet damit das Fundament des Fiorano ESB (vgl. [Fio04b], S. 3).

Eine weitere Kernkomponente der Fiorano SOA Plattform ist Fiorano BPEL, eine BPEL-basierte Komponente zur Geschäftsprozessunterstützung. Diesbezüglich ist man bei Fiorano nicht nur in der Lage, eine Orchestrierung von Web Services zu bewerkstelligen, sondern darüber hinaus auch mittels JCA gekapselte Anwendungen zu orchestrieren. Zur weiteren Unterstützung wird mit dem Fiorano BPEL Studio ein grafischer Editor zur Entwicklung BPEL-gestützter Prozessketten bereitgestellt. Obwohl dabei versucht wird, die technischen Umsetzungsdetails zu verbergen, können diese über andere Sichten ebenfalls mit dem Fiorano BPEL Studio bearbeitet werden. Dies gestattet, dass sowohl Gestalter von Geschäftsprozessen als auch Personal zur technischen Umsetzung mit ein und dem selben Werkzeug arbeiten können. Zu bemerken ist zusätzlich, dass auf Fiorano BPEL basierende Prozesse sich über eine Vielzahl verschiedener Rechner erstrecken können wie auch die Ausführungskomponenten flexibel entweder in einzelnen JVM[20] oder in entsprechenden Applikationscontainern (J2EE-Container) residieren können. Derartig verteilte BPEL-Ausführungsumgebungen können dann wiederum mittels JMS, JCA, Web Services, Websphere MQ, TIBCO Rendezvous oder anderer entsprechender Middleware angesprochen und so in einen ESB integriert werden (vgl. [Fio05a], S. 1 f.).

Zur Anbindung von unterschiedlichen Anwendungen und Systemen werden mit den Fiorano Business Components & Adapters entsprechende Konnektoren bereitgestellt. Diese gliedern sich in (vgl. [Fio06]):

- Dateiadapter zum Lesen, Schreiben und Überwachen von einfachen Dateien und als Zugriff auf das lokale Dateisystem verteilter Anwendungen.
- Datenbankadapter als Verbindung zu unterschiedlichen, relationalen Datenbanken (Oracle, MS SQL Server, IBM DB2, etc.).
- Adapter für geschlossene Anwendungen wie von SAP, Siebel, Peoplesoft und anderen.
- Adapter für vorkonstruierte Services aus den Bereichen inhaltsbezogenes Routing, Überbrückung von Protokollen, Transformation, Verschlüsselung und weiterer unterstützender Funktionalitäten.

Darauf aufbauend sind andere Anwendungen der Fiorano SOA Plattform (speziell der Fiorano ESB) in der Lage, sich vielfältigsten Bedürfnissen und Einsatzszenarien anzupassen sowie ein Höchstmaß an Konnektivität zu erreichen.

[17] API – Application Programming Interface
[18] LDAP – Lightweight Directory Access Protocol
[19] JSSE – Java Secure Socket Extension
[20] JVM – Java Virtual Machine

Der Aufbau der Fiorano SOA Plattform zeigt, dass zahlreiche Funktionalitäten einer ESB-Implementierung bereits durch andere Komponenten abgedeckt werden. Beantwortet werden muss daher die Frage, welche Funktionen im Rahmen der Fiorano SOA Plattform dem Fiorano ESB zukommen und wie diese realisiert werden.

Der grundsätzliche *Aufbau* des Fiorano ESB, der anderen ESB-Implementierungen durchaus ähnelt, ist Abbildung 3.6 zu entnehmen. Gezeigt wird hier, wie er verschiedene Anwendungen (ERP-Systeme, .NET Anwendungen, Web Services, Alt- und J2EE Anwendungen) über ebenso verschiedene Protokolle (JCA, JMS, SOAP und HTTP) entlang einer zuverlässigen, asynchronen und sicheren Nachrichteninfrastruktur miteinander verbindet (vgl. [Fio04a], S. 1).

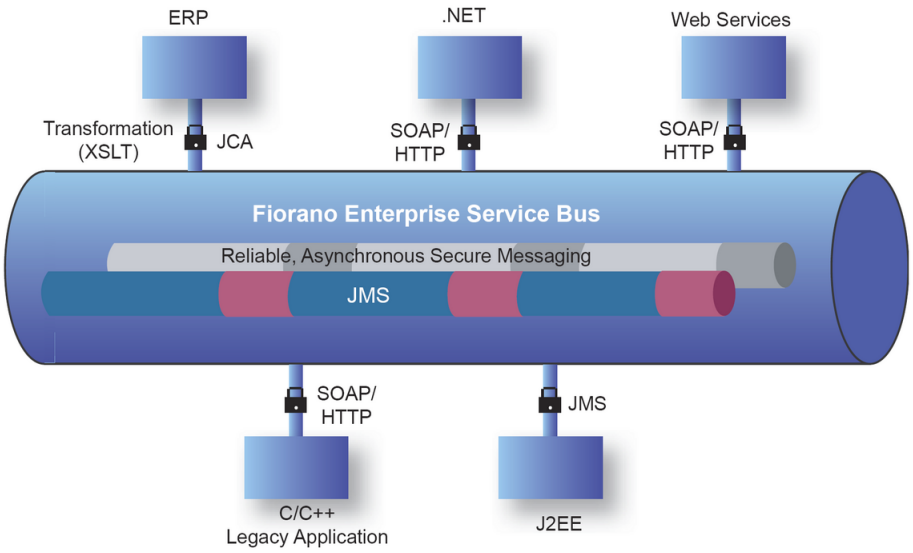

Abbildung 3.6: Fiorano Enterprise Service Bus (Quelle: [Fio04a], S. 1).

Die Stärken des Fiorano ESB liegen in den Bereichen Ereignissteuerung, Management von Servicelebenszyklen, Überwachung, Protokollierung und Standardfunktionalitäten wie Transformation und intelligentes Routing. Mittels Ereignissteuerung ist es möglich, Abläufe, die bspw. in Form von Prozessketten formuliert sind, gestützt auf Standards wie BPEL technisch umzusetzen und auf die zugrunde liegende Middleware zu übertragen. Dazu gehört ebenfalls, Prozesse und darüber die betreffenden Services orchestrieren sowie dynamisch (zur Laufzeit) auf Ereignisse reagieren zu können. Weiterhin werden das Management von Lebenszyklen und die Versionierung von Prozessen wie auch Services unterstützt. Leider wird dieser Punkt nicht genauer beschrieben und einzig auf eine Umsetzung mittels Prozessprofilen verwiesen. Festgestellt werden kann dennoch, dass bei Fiorano stets von der obersten Ebene, d. h. von der Prozessebene ausgehend, Funktionalitäten bereitgestellt werden und diese dann mit geringem Aufwand auf untergeordnete Ebenen übertragen werden können. Dies ermöglicht auch ohne technische Detailkenntnisse, Funktionen und Leistungen des Fiorano ESB nutzen zu können. Bezüglich der Kontrolle von Prozessen, Services und Nachrichten ist der Fiorano ESB in der Lage, ereignisgesteuert beteiligte Partner oder andere Stellen über kritische Fehler zu benachrichtigen, dynamisch Nachrichten umzuleiten, zu verfolgen, zu protokollieren oder zu überwachen. Hinzu kommen die Unterstützung für intelligentes Routing, anspruchsvolle Transformationsmöglichkeiten von XML-Daten sowie die Fähigkeit, diese dynamisch validieren zu können. Zu den Standards, die durch den Fiorano ESB selbst unterstützt werden, gehören als Transport- und Protokollstandards z. B. JMS, JCA, SOAP,

HTTP und HTTPS, als Standards zur Transformation XSLT, XPath und XQuery und als Standards aus dem Bereich Web Services WSDL, UDDI und BPEL4WS. Hinzu kommen bereits erwähnte und in Fiorano Business Components & Adapters enthaltene Konnektoren, die zusammen mit dem Fiorano ESB vertrieben werden (vgl. [Fio04a], S. 1 ff.).

Der mit dem Fiorano ESB und mit der Fiorano SOA Plattform bereitgestellte *Funktionsumfang* ermöglicht nahezu alle in Kapitel 2.2 beschriebenen Aufgaben und Eigenschaften eines ESB zu erfüllen. Defizite liegen lediglich im Bereich der Dienstgüte und grafischen Unterstützung. Im Gegensatz zu anderen ESB-Implementierungen, die eine Verwaltung und Überwachung von Dienstgütevereinbarungen (SLA) ermöglichen, ist dies mit dem Fiorano ESB nicht möglich. In ähnlicher Weise sind Defizite in der grafischen Unterstützung auszumachen. Hier gibt es in Form des Fiorano BPEL Studios eine äußerst umfangreiche GUI zum Erstellen und Verwalten von Prozessen und Services sowie zu deren Orchestrierung. Was fehlt bzw. nicht beschrieben wird, ist eine Komponente zur Visualisierung des dynamischen Verhaltens des ESB und der beteiligten Services. Eine weitere Schwäche liegt in der Darstellung der eigentlichen Architektur des Fiorano ESB. Zwar spricht man bei Fiorano von einer flexiblen und verteilten Anordnung von Services, mit der Fähigkeit, diese separat konfigurieren und verwalten zu können. Nicht konkretisiert wird jedoch, ob diesbezüglich leichtgewichtige Service-Container zum Einsatz kommen bzw. auf welche konkreten Softwarekomponenten sich der Fiorano ESB stützt. Darin ein reiht sich die Frage nach einem zentralen Repository zur Metadatenverwaltung bzw. einer Registry als Verzeichnis für vorhandene Services, Anwendungen oder Systeme. Ob diese fehlen oder lediglich auf eine Beschreibung verzichtet wurde, kann leider nicht genauer geklärt werden.

Aufgrund der Tatsache, dass kaum Äußerungen über den konkreten Aufbau des Fiorano ESB gemacht werden, fällt eine Bewertung des Fiorano ESB in Bezug auf *Definitionskonformität* schwer. Ausgehend von den vorhandenen Beschreibungen seitens Fiorano und die Tatsache berücksichtigend, dass keine offensichtlichen Widersprüche zur genannten ESB-Definitionen erkennbar sind, kann von einer (zumindest beschränkten) Definitionskonformität ausgegangen werden. Gestützt wird diese Annahme durch die Installationsbeschreibung der Fiorano SOA Plattform bzw. des Fiorano ESB, die von einzelnen und separaten ESB Endpunkten (ESB Peers) spricht. Ähnlich schwer fällt die Bewertung der *Flexibilität* des Fiorano ESB. Für ein Mindestmaß an Flexibilität sprechen die Unterstützung von vielen Betriebssystemen (Windows, Linux, HP-UX, Solaris/SunOS, i5/OS und Mac OS) sowie die mitgelieferten Konnektoren. Unklar bleibt aber, wie umfangreich und flexibel sich eine Installation im Detail darstellt (vgl. [Fio04a], S. 4).

Besondere *Qualitätseigenschaften* des Fiorano ESB genauer der Fiorano SOA Plattform liegen auf dem Gebiet der (Geschäfts-)Prozessunterstützung. Mit reichhaltigen Werkzeugen können Prozessketten ohne umfangreiche technische Kenntnisse entwickelt und verwaltet werden. Ähnlich hervorzuheben ist die ausgereifte Ereignissteuerung, die notwendige Voraussetzungen für die Umsetzung einer EDA schafft.

Im Gesamtkontext stellt sich die Fiorano SOA Plattform mit ihren Produkten (speziell dem Fiorano ESB) als überaus leistungsfähig dar, insbesondere vom Standpunkt des unterstützten Funktionsumfangs. Ebenso positiv sind die Beziehungen dieser Komponenten untereinander zu bewerten. Es treten kaum Überschneidungen in den bereitgestellten Funktionalitäten auf, es sind vielmehr abgestimmte und sich ergänzende Beziehungen festzustellen. Über den Weg komplementärer Produkte wird ein Großteil an ESB-Funktionalität aus der eigentlichen ESB-Implementierung ausgegliedert, steht aber prinzipiell zur Verfügung. Bis auf einzelne kleinere Schwächen und der offenen Frage nach dem konkreten Aufbau des Fiorano ESB steht zusammen mit der Fiorano SOA Plattform und den angebotenen Standards ein ausgereiftes Produktportfolio auf dem Weg hin zu einer SOA bereit.

3.5 ESB nach Cape Clear

Der von Cape Clear angebotene Cape Clear ESB gehört zu den im Gesamtmarkt am stärksten vertretenen ESB-Implementierungen, der sich jedoch in gewisser Weise von anderen Implementierungen (etwa denen von Sonic und Fiorano) unterscheidet. Dennoch sind durchaus Parallelen vorhanden, auf die zu Beginn eingegangen werden soll. Ähnlich wie beim Sonic ESB bzw. beim Fiorano ESB werden auch bei Cape Clear neben der eigentlichen ESB-Implementierung noch weitere, komplementäre Komponenten vertrieben bzw. zusammen mit dem Cape Clear ESB gebündelt und ausgeliefert. Zu diesen erweiterten Bestandteilen gehören das Cape Clear Studio, Manager mit BAM-Unterstützung, Data Transformer und Orchestrator. Sie realisieren bereits ein Großteil erweiterter ESB-Funktionalitäten, so dass sich der eigentliche Cape Clear ESB auf Kernaufgaben beschränken und fokussieren kann.

Den allgemeinen Aufbau der Cape Clear ESB Plattform und ihre *Bestandteile* zeigt Abbildung 3.7. Neben den bereits genannten Komponenten lassen sich unter den Punkten Cape Clear ESB (Cape Clear EBS Server) und Service Bereitsteller (Service Enabler) die zentralen Bestandteile finden. Sehr gut dargestellt sind die alle Komponenten der Cape Clear ESB Plattform flankierenden grafischen Werkzeuge Cape Clear Studio und Cape Clear Manager. Ebenso enthalten sind diverse angeschlossene Anwendungen und Systeme, die von Portalen, Webanwendungen (Web Server), Applikationsservern (Application Server) über Mehrkanalschnittstellen (Mulit-Channel Gateway) und Web Services bis hin zu Datenbanken (Database), Mainframes und geschlossenen Anwendungen (Packaged Applications) reichen (vgl. [Cap06c], S. 1 ff.; [Cap06i], S. 15 f.).

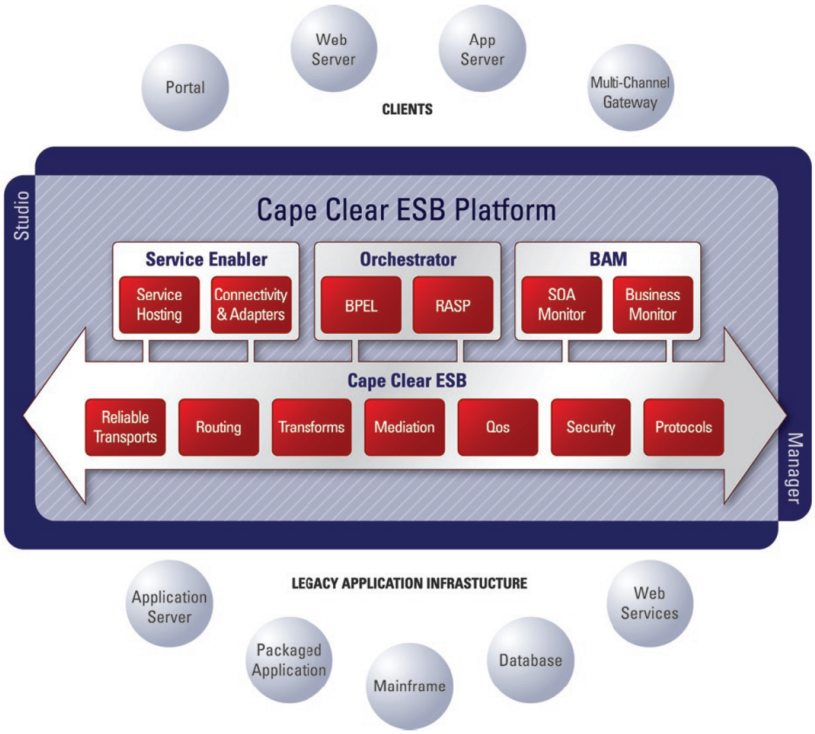

Abbildung 3.7: Cape Clear ESB Plattform (Quelle: [Cap06c], S. 4).

Das Cape Clear Studio ist eine erste Teilkomponente und der Anfangspunkt für die Entwicklung von SOA-basierten Anwendungen auf der Grundlage von Eclipse. Die Unterstützung

reicht von der Entwicklung neuer Services über deren Orchestrierung bis hin zur Definition und Umsetzung anspruchsvoller Transformationen. Speziell bei der Entwicklung von Services wird (zumindest im Java Umfeld) sowohl ein Top-Down als auch ein Bottom-Up Vorgehen unterstützt. So können ausgehend von entsprechenden Servicebeschreibungen (WSDL-/XML Schema Dateien) Klassen und Methoden generiert (Top-Down) oder umgekehrt diese sehr leicht in Web Services überführt werden (Bottom-Up). Generell unterstützt wird der Bottom-Up Ansatz, der es ermöglicht, relativ einfach andere Anwendungen oder Systeme über geführte Menüs zu integrieren und in Form von Services zu kapseln. Auf ähnliche Weise grafisch unterstützt werden umfangreiche Möglichkeiten zur Transformation, die von der Überführung proprietärer Formate nach XML bis hin zu weiterführenden XML-Transformationen reichen. Weitere Funktionen umfassen die Modellierung von Geschäftsprozessen und die damit eng verbundene Orchestrierung von Services. Daran an knüpft das weitestgehende Verbergen von technischen Details, mit dem Ziel, den Fokus auf die Geschäftslogik und weniger auf Fragen zur Infrastruktur zu lenken. Hervorzuheben ist eine ebenfalls integrierte Umgebung, die den Test (bspw. automatisch generierte JUnit[21] Tests) von Services und Prozessen ermöglicht. Im Kontext von Servicelebenszyklen werden diese durch den Cape Clear ESB und das Cape Clear Studio nahezu vollständig abgedeckt. Das reicht von der Modellierung, Entwicklung, Integration, Test und Deployment von Services bis hin zu einem ausfallfreien erneuten Deployment (Hot Re-Deployment). Abschließend kann man feststellen, dass mit dem Cape Clear Studio eine umfangreiche GUI für alle Komponenten des Cape Clear ESB geliefert wird, die von einem reichhaltigen Funktionsumfang und einer Integration in Eclipse als weit verbreitete und flexible Entwicklungsumgebung profitiert (vgl. [Cap06h]).

Es ist wenig verwunderlich, dass die Beschaffung von Daten und speziell deren strukturierte und allen Partnern verständliche Darstellung eine Kernaufgabe bei der Integration von Services, Anwendungen und Systemen darstellt. Übernommen wird diese Aufgabe vom Cape Clear Data Transformer. Als weitere Komponente der Cape Clear ESB Plattform ermöglicht er den Zugang zu Daten, die in Altanwendungen, proprietären Systemen oder in sonstigen Datenquellen gehalten werden. Ausgestattet mit der Fähigkeit diese Daten nach XML zu überführen, können darauf aufbauend ebenso mittels XSLT weiterführende Transformationen vorgenommen werden. Die Bereitstellung derartiger Transformationen erfolgt über entsprechende, an den Cape Clear ESB angeschlossene Services. Zu beachten ist, dass es beim Data Transformer weniger darum geht, über Adapter einen Zugang zu anderen Systemen herzustellen, als über bereits vorhandene Zugänge einen XML-basierten Zugriff auf heterogene Daten zu ermöglichen. Konfiguriert wird der Data Transformer über das bereits erwähnte Cape Clear Studio (vgl. [Cap06b]; [Cap06g], S. 1 ff.).

Eine weitere wichtige Aufgabe stellt die Orchestrierung von Prozessen und Services dar. Hierzu wird von Cape Clear ein so genannter Orchestrator angeboten. Er dient der Umsetzung von Geschäftsprozessen mittels BPEL. Angesteuert wird der Cape Clear Orchestrator, wie andere Komponenten auch, über das Cape Clear Studio als GUI zur Modellierung der umzusetzenden Abläufe. Die grafische Wiedergabe der auf BPEL basierenden Prozesse stellt sich als ein mächtiges Werkzeug dar, wenn es darum geht, Geschäftsprozesse gegenüber weniger technisch geprägten Anwendern zu veranschaulichen oder diese zu befähigen, eigenständig Geschäftsprozesse modellieren und umsetzen zu können. Diese Fähigkeiten sind es, die eine Übersetzung von geschäftsrelevanten Anforderungen in technische Implementierungen unterstützen sollen. Neben der Modellierung von Geschäftsprozessen gilt es, deren Verwaltung und Überwachung im laufenden Betrieb nicht zu vernachlässigen. Cape Clear bietet dazu mit dem Cape Clear Manager eine Web Browser-gestützte Konsole an, die ein derartiges Management im Sinne von BAM unterstützt. Mit dem Cape Clear Orchestra-

[21] JUnit ist ein Framework zum Testen von Java-Programmen, das speziell für Komponententests (Unit Tests) konzipiert wurde.

tor wird eine Komponente bereitgestellt, die Geschäftsprozesse und damit verbundene, zum Teil lang andauernde Transaktionen umsetzen kann. Neben dem Debugging von Prozessen stehen Möglichkeiten bereit, blockierte oder fehlgeschlagene Prozesse wieder aufzunehmen. Abgerundet wird das Bild des Cape Clear Orchestrators durch die Fähigkeit zur Bildung von Clustern. Diese gewährleisten die Erfüllung höchster Anforderungen bezüglich Skalierbarkeit und Ausfallsicherheit erfolgskritischer Geschäftsprozesse (vgl. [Cap06f]).

Wie bereits angesprochen, ist das Management eines ESB und seiner beteiligten Komponenten im laufenden Betrieb von zentraler Bedeutung. Zur Umsetzung damit einhergehender Aufgaben wird der Cape Clear Manager bereitgestellt. Das Spektrum der angebotenen Funktionalitäten reicht hierbei von der Administration der Infrastruktur und Services über BAM, Überwachung, Diagnose und Integration mit anderen Managementwerkzeugen. Auf dem Gebiet der Infrastrukturadministration können relativ einfach ESB-Server, Cluster und zusätzliche Infrastruktur wie Adapter, Transporte oder andere nachrichtenbasierte Systeme verwaltet und angeschlossen werden. Die eigentliche Serviceverwaltung umfasst im Kern das Management von Servicelebenszyklen und gestattet bspw. über alternativ ansteuerbare Serviceimplementierungen eine ausfallfreie Wartung von Services. Zusätzlich neben administrativen Tätigkeiten, die sich mit dem Cape Clear Manager bewerkstelligen lassen, gibt es umfangreiche Möglichkeiten, den laufenden Betrieb und das Innenleben des Cape Clear ESB zu überwachen. Geschehen kann das über die grafische Darstellung von Meta- und Laufzeitdaten sowie eine Unterstützung zur Protokollierung und Ausgabe von Warnungen und Fehlern. Eine spezielle Teilkomponente ist in diesem Kontext der Cape Clear Orchestration Manager. Dieser liefert mit der Einsichtnahme in den Fortschritt von Geschäftsprozessen zur Laufzeit, der Darstellung von Geschäftsprozesshistorien und einer hohen Granularität (bis hin zu einzelnen Nachrichten) umfangreiche Funktionen aus dem Bereich BAM. Weiterhin ist es über entsprechende JMX- bzw. SNMP[22]-basierte Schnittstellen möglich, jeden im Rahmen des Cape Clear ESB beteiligten Cape Clear Server und die darin gehaltenen Services unabhängig vom Cape Clear Manager zu steuern und das von jeder Managementplattform aus, die diese Standards (JMX oder SNMP) unterstützt (z. B. HP OpenView, CA Unicenter oder IBM Tivoli). Hervorgehoben werden kann, dass damit das notwendige Maß an Flexibilität geboten wird, das im Kontext eines föderierten Betriebs etwa mit anderen ESB-Implementierungen unerlässlich ist. Mit dem Cape Clear Manager wird ein äußerst leistungsfähiges Administrations- und Managementwerkzeug zur Verfügung gestellt, das die ohnehin schon als sehr gut zu bezeichnende grafische und werkzeugbasierte Unterstützung des Cape Clear ESB (etwa durch das Cape Clear Studio) noch weiter erhöht (vgl. [Cap06e]).

Im Kern stellt sich der Cape Clear ESB vom *Aufbau* her als ein Netzwerk aus verteilten Servern dar. Diese können entweder eigenständig oder innerhalb von Applikationsservern betrieben werden. Sie besitzen einen Service-Container, innerhalb dessen die bereitgestellten Services gehalten werden. Über entsprechende Adapter ist es möglich, Anwendungen und Systeme an einen ESB-Server anzuschließen und in Form von Services zu exportieren. Cape Clear spricht in diesem Zusammenhang von einem so genannten Service Enabler. Diese Komponente liefert eine native Unterstützung von Web Services und damit verbundenen Protokollen wie SOAP, HTTP oder HTTPS. Gleichfalls als Protokolle unterstützt werden SMTP oder FTP. Eine eigenständige nachrichtenbasierte Infrastrukturplattform wird nicht mitgeliefert, jedoch können solche dank der Unterstützung von JMS ebenso angesprochen werden. Es ist darauf hinzuweisen, dass das Konzept des Cape Clear ESB gerade für diese zweite Variante (bereits vorhandene Infrastruktur zu nutzen und nicht neu einzuführen) geschaffen wurde. Über diesen Weg können verteilte ESB-Server gestützt auf SOAP über HTTP genauso wie mittels Websphere MQ oder JBossMQ miteinander kommunizieren. In diesem

[22] SNMP – Simple Network Management Protocol

Zusammenhang werden eine Reihe von WS-Standards so z. B. WS-ReliableMessaging, WS-Adressing und WS-Policy unterstützt (vgl. [Cap06i], S. 12 ff.; [Cap06j], S. 6 ff.).

Neben dem Transport von Nachrichten erfolgen im Cape Clear ESB zusätzlich notwendige Transformationen, intelligentes Routing und die Umsetzung von Sicherheitspolitiken. Die Transformationen stützen sich auf XSLT, XPath bzw. XQuery als Standards zur Manipulation XML-basierter Daten. Zur Übersetzung anderer Formate steht der bereits angesprochene Cape Clear Data Transformer bereit. Intelligentes Routing betreffend können Nachrichten entweder gestützt auf BPEL aber auch inhaltsabhängig oder gemäß Header Informationen weitergeleitet werden. Zum Cape Clear ESB hinzu kommt eine Datenbank, die zur Umsetzung einer UDDI-basierten Registry als Verzeichnis für vorhandene Services dient. Es ist dabei möglich und wird sogar empfohlen, die mitgelieferte Datenbank durch andere leistungsfähigere Datenbanken (Oracle, MS SQL Server oder MySQL) zu ersetzen. Ebenfalls benötigt wird diese Datenbank zur Speicherung anfallender Metadaten (vgl. [Cap06i], S. 13 f.; [Cap06k], S. 2).

Ein neben der Architektur gleichfalls wichtiger Gesichtspunkt betrifft Maßnahmen zur Sicherheit. Das bei Cape Clear verfolgte Sicherheitskonzept umfasst die Punkte (vgl. [Cap06d], S. 6):

- Authentifizierung,
- Autorisation,
- Vertraulichkeit,
- Integrität,
- Verfügbarkeit.

Die Umsetzung dieser Ziele erfolgt über spezielle dem Kommunikationsfluss zwischengeschaltete Sicherheitskomponenten (als so genanntes Interceptor Framewok bezeichnet). Ihre Aufgabe ist es, wie in Abbildung 3.8 dargestellt, zuvor definierte Sicherheitspolitiken modular zu implementieren. Es wird dabei versucht, Services um den Aspekt der Sicherheit zu entlasten. Dazu werden Sicherheitspolitiken separat definiert und dann mit bereits bestehenden Services verknüpft. Entwickler von Services müssen sich dadurch nicht länger um Aspekte der Sicherheit kümmern, sondern können diese Aufgabe an Sicherheitsexperten delegieren, die selbst wiederum kein Wissen über die konkreten Serviceimplementierungen haben müssen (vgl. [Cap06d], S. 7 ff.).

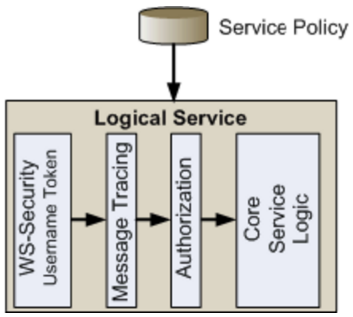

Abbildung 3.8: Cape Clear Sicherheitsframework (Quelle: [Cap06d], S. 9).

In Bezug auf Authentifizierung kann entweder ein mitgelieferter dateibasierter Identitätsspeicher verwendet werden oder ebenso auf Mechanismen wie LDAP Verzeichnisserver, Authentifizierungsdatenbanken oder andere über JAAS[23] ansteuerbare, externe Identitätsmanagementlösungen zurückgegriffen werden. Die zur Authentifizierung verwendeten Mechanismen basieren hierbei auf einfacher Authentifizierung über HTTP oder auch auf fortgeschrittenen Standards wie SSL, SAML[24] oder XML-basierten digitalen Signaturen. Die mit der Authentifizierung verbundene Rechtevergabe und -zuweisung, also Autorisation, kann über XACML[25] oder den Rückgriff auf Berechtigungs- bzw. ACL-gestützte Systeme von Drittanbietern erfolgen. Vertraulichkeit als weiterer Gesichtspunkt findet auf zwei verschiedenen Ebenen statt, zum einen auf Transportebene (SSL bzw. TLS) und zum anderen auf Nachrichtenebene. Bezüglich Integrität kann diese in ähnlicher Weise mit SSL bzw. TLS auf der Transportebene oder mittels digitaler Signaturen auf der Nachrichtenebene erfolgen. Zum Schutz vor zufällig oder mutwillig herbeigeführten Verfügbarkeitsproblemen werden mehrere davor schützende Mechanismen bereitgestellt. Standardmäßig erfolgt eine Validierung aller eingehenden Nachrichten (und ggf. deren Ablehnung). Es können Proxyserver oder Router zum Schutz vor DoS[26]-Angriffen zwischengeschaltet werden. Über Warteschlangen wird außerdem der Transport von Nachrichten von ihrer Bearbeitung entkoppelt und spezielle Abfangmechanismen des Interceptor Frameworks erlauben die zu bearbeitende Nachrichtenlast zu beschränken. Nicht zuletzt können Obergrenzen für den Datenumfang von Nachrichten gesetzt werden. In Fragen der Kompatibilität mit anderen, eventuell bereits bestehenden Sicherheitskomponenten werden zusätzlich Standards wie SAML und WS-Security unterstützt. Mit diesen weitestgehend auf Standards basierenden Mechanismen verfügt Cape Clear wohl über eines der ausgereiftesten Sicherheitsframeworks und deckt notwendige Sicherheitsaspekte bestmöglich ab (vgl. [Cap06d], S. 11 ff.).

Zurückkommend auf die gesamte Cape Clear Plattform besteht der durch Teilkomponenten bereitgestellte *Funktionsumfang* aus:

- Unterstützung und Abbildung von (Geschäfts-)Prozessen bzw. Prozessketten,
- Service Orchestrierung/Choreographie,
- Transformation,
- Konfiguration, Monitoring und Management von Services,
- Management von Servicelebenszyklen,

Folgende weitere Kernfunktionen liefert hierbei der eigentliche Cape Clear ESB:

- standardbasierte Kommunikation und Integration,
- intelligentes Routing,
- Sicherheit (QoP),
- Dienstgüte (QoS),
- Transaktionsverwaltung,

[23] JAAS – Java Authentication and Authorization Service
[24] SAML – Security Assertion Markup Language
[25] XACML – eXtensible Access Control Markup Language
[26] Denial of Service (wörtlich Dienstverweigerung) bezeichnet den (überlastungsbedingten) Ausfall eines Dienstes.

- flexibles Deployment,
- Skalierbarkeit,
- Föderation und Erweiterbarkeit.

In Bezug auf *Flexibilität* lässt der Cape Clear ESB kaum Wünsche offen. An erster Stelle können hier umfangreiche Adapter genannt werden. Diese reichen von Datenbankadaptern (für Oracle, MS SQL Server, Sybase, etc.), Datenformatadaptern (XML, CSV, EDI/EDIFACT, Textdateien, EXCEL spreadsheets, usw.) über Anwendungsadapter (Java, J2EE, CORBA und COBOL) bis zu Nachrichtenadaptern zum Anschluss JMS-basierter Nachrichtensysteme wie etwa Websphere MQ oder JBossMQ. Zusätzlich können ebenso JCA-kompatible Adapter von Drittanbietern verwendet werden. Abgesehen davon erweist sich der Cape Clear ESB auch im Zusammenspiel mit nachrichtenorientierter Middleware als äußerst flexibel. Im Gegensatz zu anderen ESB-Anbietern verzichtet Cape Clear mit seiner ESB-Implementierung darauf, eine eigenständige nachrichtenbasierte Kommunikationsplattform mit auszuliefern. Mit dieser Strategie will man vermeiden, dass Kunden dazu gezwungen werden, eine bestimmte Kommunikationsplattform zu verwenden. Die Philosophie von Cape Clear ist dabei, keine neue nachrichtenbasierte Infrastruktur in einem Unternehmen einzuführen, als vielmehr die zumeist bereits vorhandene Infrastruktur bestmöglich zu nutzen. Als leider nachteilig könnte sich dies auswirken, falls noch keine solche Infrastruktur vorhanden sein sollte und eine vom Cape Clear ESB gelieferte native Unterstützung für SOAP über HTTP basierend auf WS-Standards wie WS-ReliableMessaging, WS-Security u. a. als nicht ausreichend erachtet wird. Weitere Einschränkungen betreffen die unterstützten Plattformen, zu denen lediglich Windows Server, Windows XP, Red Hat Linux und Solaris/SunOS gehören[27]. Es ist dafür aber möglich, den Cape Clear ESB auf diesen Plattformen in bereits vorhandene Applikationsserver zu integrieren (BEA WebLogic Server, IBM Websphere, JBoss, etc.) und auf diesem Weg anwendungsnah zu betreiben (vgl. [Cap06k], S. 1 f.; [Cap06a], S. 1 f.).

Eine weitere noch zu beantwortende Frage betrifft die *Definitionskonformität* des Cape Clear ESB. Wie ist hier das Prinzip verteilter ESB-Server hinsichtlich der geforderten leichtgewichtigen Service-Container einzuordnen? Wenngleich Cape Clear von Servern spricht, so sind diese dennoch nicht so umfangreich wie Applikationsserver oder Integrationsserver. Eine Tatsache, die dies bekräftigt, ist die Möglichkeit, einen ESB-Server innerhalb eines Applikationsservers zu betreiben. Darüber hinaus verfügt der ESB-Server neben der nativen Unterstützung von Web Services über keine nachrichtenbasierte Infrastrukturplattform, die dem Prinzip leichtgewichtiger Service-Container möglicherweise entgegen wirken könnte. Dieser Sichtweise folgend, kann der Cape Clear EBS-Server durchaus als leichtgewichtiger Service-Container angesehen werden, zumal er in der Lage ist, flexibel und granular nur die gerade benötigten Funktionalitäten bereitzustellen (im Gegensatz zu monolithischen Integrationsservern).

Hervorragende *Qualitätseigenschaften* des Cape Clear ESB, genauer der Cape Clear ESB Plattform bieten an erster Stelle die beiden Werkzeuge Cape Clear Studio und Cape Clear Manager. Diese gestatten es, auf nahezu sämtliche angebotene Funktionalitäten grafisch unterstützt zuzugreifen und so auch Servicelebenszyklen nahezu vollständig zu verwalten. In der Summe liefern diese Werkzeuge über die Darstellung statischer wie auch dynamischer Eigenschaften eine ganzheitliche Sichtweise auf den Cape Clear ESB, die diesen von anderen Implementierungen abhebt. Eine weitere Stärke liegt auf dem Gebiet der Sicherheit. Ähnlich umfangreich und leistungsfähig wie die grafische Unterstützung ist das bereitgestellte Sicherheitsframework. Aspekte wie Authentifizierung, Autorisation und Vertraulichkeit

[27] Eine Java Laufzeitumgebung (Java Runtime Environment – JRE) in der Version 1.4 oder höher ist zwingend erforderlich.

spielen hierbei eine elementare Rolle und werden zusätzlich von Konzepten zum Schutz von Integrität und Verfügbarkeit ergänzt.

Trotz der Fragezeichen hinter einer fehlenden nachrichtenbasierten Infrastruktur ist dieses Vorgehen unter dem Gesichtspunkt der Flexibilität geradezu zwingend. Cape Clear vermeidet Dinge neu einzuführen, die in den allermeisten Fällen bereits vorhanden sind. Vielmehr liefert man nur das, was wirklich benötigt wird. Cape Clear spricht in diesem Zusammenhang von einer infrastrukturerkennenden (infrastructure-agnostic) Architektur, die bereits existierende Ressourcen (nachrichtenbasierte Infrastruktur, Applikationsserver, etc.) bestmöglich nutzen soll. Überhaupt kann man feststellen, dass die gesamte Cape Clear ESB Plattform am Ziel weitreichender Flexibilität ausgerichtet ist. Das beginnt bei zahlreichen Adaptern, setzt sich fort bei Eclipse- und Web-basierten grafischen Werkzeugen und endet bei offenen Schnittstellen (gestützt auf JMX bzw. SNMP) zur Verwaltung der beteiligten ESB-Komponenten. Wenngleich eine umfangreichere Plattformunterstützung wünschenswert wäre, werden zumindest gängige Betriebssysteme unterstützt. Mit diesem Ansatz und unter Berücksichtigung des angebotenen Leistungsspektrums ist die hervorragende Stellung von Cape Clear im gesamten ESB-Markt kaum verwunderlich.

3.6 ESB nach BEA

BEA Systems verfügt mit dem BEA AquaLogic Service Bus nicht nur über eine ESB-Implementierung, sondern mit der BEA AquaLogic Produktfamilie auch über weitere, dazu komplementäre Produkte. Diese sind die BEA AquaLogic User Interaction, BPM Suite, Data Services Platform, Enterprise Security, Enterprise Repository und Service Registry.

Die *Bestandteile* einer SOA bzw. eines erweiterten ESB nach BEA gliedern sich wie schon bei anderen Anbietern in einzelne Schichten. Eine erste derartige Schicht ist die BEA AquaLogic User Interaction. Wie der Name bereits andeutet, geht es um die Einbeziehung der Nutzer und deren Interaktion mit einer SOA. Mit BEA AquaLogic User Interaction werden Werkzeuge zur Schaffung von Firmenportalen, kollaborativen Arbeitsgruppen und zusammengesetzten Anwendungen bereitgestellt, die alle auf einer Service-Infrastruktur aufbauen. Auf diesem Weg soll es Mitarbeitern, Geschäftspartnern oder Kunden erleichtert werden, Zugriff auf relevante Daten und Systeme zu erhalten (vgl. [BEA06k]).

Die BEA AquaLogic BPM Suite als weitere Komponente dieser Produktfamilie dient hauptsächlich der Verwaltung von Geschäftsprozessen. Dies umfasst deren Modellierung, Implementierung, Ausführung und Überwachung. Somit soll der gesamte Lebenszyklus eines Geschäftsprozesses und dessen Optimierung unterstützt werden. Zwei Kernbestandteile sind dabei das BEA AquaLogic BPM Studio als GUI für eine UML-basierte Modellierung, Simulation und Verteilung von Geschäftsprozessen sowie der BEA AquaLogic BPM Enterprise Server, dessen Aufgaben die Orchestrierung aller Prozesse und Ressourcen (Menschen, Organisationen, Anwendungen und Systeme), die Kontrolle von Geschäftsregeln und die Überwachung sowie Fehlerbehandlung von Prozessen sind. Unterstützt und ausgeführt werden können dabei sämtliche durch BPEL beschriebene Prozesse (vgl. [BEA05a], S. 2 ff.; [BEA06a]).

Eine weitere Komponente bzw. Schicht der BEA AquaLogic Produktfamilie ist die BEA AquaLogic Data Services Platform. Im Kern geht es hierbei darum, Datenbankverbindungen zu kapseln. Abbildung 3.9 zeigt den Aufbau dieser Schicht (Enterprise Data Services Layer), die einen vereinfachten Zugriff auf heterogene Echtzeitdaten erlaubt. Der Aufbau zeigt Parallelen zum Aufbau eines ESB, befasst sich aber einzig mit dem Zugriff auf Daten, die aus unterschiedlichsten Quellen stammen können. Als Quellen genutzt werden können

gängige relationale Datenbanken (Zugriff auf Tabellen, Sichten und Stored Procedures), nicht relationale Datenquellen wie einfache Dateien und XML-Dateien oder mittels Web Services auch Daten aus geschlossenen Anwendungen (SAP, Oracle, Siebel u. a.). Zugegriffen werden kann auf diese Daten mittels Java/SDO, ADO.NET, Web Services und nicht zuletzt auch mit einer JDBC Schnittstelle. Ein wesentlicher Nutzen dieser Komponente besteht in der schnellen und grafisch unterstützten Entwicklung von Sichten und Anfragen für hoch verteilte Daten (vgl. [BEA06b], [BEA06c] S. 3 f.).

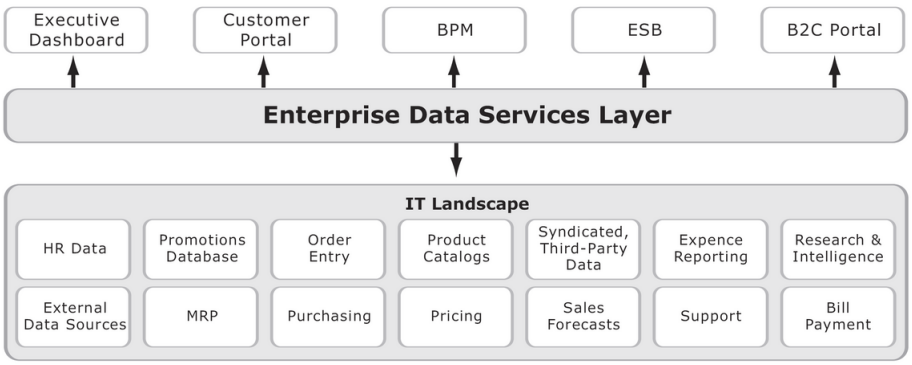

Abbildung 3.9: BEA AquaLogic Data Services Platform (Quelle: [BEA06c], S. 2).

Mit der BEA AquaLogic Enterprise Security werden vor allem Berechtigungen, Sicherheitspolitiken und Sicherheitsmaßnahmen durch ein eigenständiges Produkt abgedeckt. Hauptaufgaben sind Authentifizierung, Autorisation, Rechtevergabe und Abbildung von Rollen. Bereitgestellt werden diese Funktionen durch einen Berechtigungsserver und einen Administrationsserver. Der Administrationsserver dient der Definition von Politiken für Zugangsberechtigungen bzw. Rollen und verfügt über Repositories für Sicherheitspolitiken. Damit ist er die zentrale Komponente zur Steuerung des Berechtigungsservers. Dieser setzt die definierten Politiken um und unterstützt Standards wie XACML und SAML. Anzumerken ist, dass sich BEA AquaLogic Enterprise Security hauptsächlich auf Berechtigungen konzentriert und weniger bzw. gar nicht auf Aspekte wie Integrität oder Verschlüsselung (vgl. [BEA06f]; [BEA06g], S. 1 ff.).

Zur professionellen Verwaltung von anfallenden Metadaten wird das BEA AquaLogic Enterprise Repository angeboten. Dabei sollen sämtliche im Rahmen von Servicelebenszyklen (Planung, Design, Implementierung/Produktion, Versionierung und Ablösung) anfallenden Metadaten durch eine zentrale Komponente verwaltet und bereitgestellt werden. Dazu gehören weiterhin die automatische Erfassung von Metadaten durch Analyse von Konfigurationsdateien und Integration in den Erstellungsprozess, die Umsetzung und Verwaltung von Politiken zu Sicherheit, Qualitätsstandards, Unternehmensrichtlinien, etc. und die Unterstützung bzw. Integration von Entwicklungsumgebungen wie Eclipse oder Microsoft Visual Studio. Zu den technischen Voraussetzungen gehören ein Applikationsserver (BEA WebLogic, IBM Websphere oder Apache Tomcat) und eine entsprechende Datenbank (Oracle, Microsoft SQL Server oder IBM DB2), auf denen das BEA AquaLogic Enterprise Repository aufsetzt (vgl. [BEA06d], [BEA06e], S. 1 ff.).

Die BEA AquaLogic Service Registry dient als eigenständige Komponente für das Suchen, Finden und Veröffentlichen von Services. Im Zentrum steht dabei eine entsprechende Registry Datenbank (basierend auf Oralce, IBM DB2, MS SQL Server, Sybase, etc.), die zur Verwaltung der anfallenden Metadaten (WSDL, XML Schemata, BPEL u. a.) dient. Neben den bereits genannten Aufgaben erfolgt auch eine aktive Unterstützung von Servicelebens-

zyklen in der Form, dass Änderungen an Services mittels Benachrichtigungen an Nutzer oder Prozesse weitergegeben werden. BEA AquaLogic Service Registry stellt sich als UDDI v3-konform dar und setzt damit auf einen anerkannten Standard für Serviceverzeichnisse. Es ist dadurch für jeden UDDI Client möglich, mit der BEA AquaLogic Service Registry zusammenzuarbeiten (vgl. [BEA06i], [BEA06j], S. 1 ff.).

Betrachtet man den Aufbau der BEA AquaLogic Produktfamilie im Ganzen, so ist zu bemängeln, dass Schnittstellen der einzelnen Produkte zueinander kaum oder gar nicht zu existieren scheinen oder einfach nicht erwähnt werden. Es entsteht daher der Eindruck, dass es sich um voneinander unabhängige bzw. sich überschneidende Produkte und Funktionalitäten handelt. Selbst das Zusammenspiel mit dem BEA AquaLogic Service Bus, der nach BEA das eigentliche Infrastrukturgerüst einer SOA bildet, wird nicht erwähnt. Im Weiteren wird daher auf eine ausführlichere Betrachtung von anderen Produkten der BEA AquaLogic Produktfamilie abgesehen und lediglich der BEA AquaLogic Service Bus näher und umfangreich beschrieben.

Der *Aufbau* des eigentlichen BEA AquaLogic Service Bus ist in vier wesentliche Bestandteile gegliedert (siehe Abbildung 3.10). Diese sind Service Management, Message Brokering, Configuration Framework und Security Framework. Der Wichtigste ist das Message Brokering, also das Vermitteln und Versenden der Nachrichten und dabei das Service Messaging. Nachrichten können entweder mit SOAP und WSDL beschriebene XML-Daten, reine XML-Daten, Message Format Laguage (MFL) Daten, Daten im Textformat oder reine Binärdaten sein. In Abhängigkeit des jeweiligen Datenformats stehen dann unterschiedliche Transportprotokolle/-mechanismen bereit. Dazu gehören JMS, HTTP, HTTPS, SMTP, FTP und der Dateizugriff[28]. Ebenfalls abhängig vom Datenformat ist die interne Codierung der Daten (text/XML, text/plain oder binary/octet-stream). Über entsprechende Adapter ist es auch möglich, proprietäre Produkte und Protokolle wie Websphere MQ (ehemals MQ Series), TIBCO oder Tuxedo zu nutzen. Weitere Funktionen sind inhaltsbezogenes Routing, Transformation und Fehlerbehandlung. Gesteuert werden diese und andere Funktionen mit dem Project Explorer, der als eingebaute grafische Oberfläche zur Steuerung und Verwaltung des ESB dient. So können hier mittels XQuery oder XSLT Transformationen definiert und ihre Position bzw. Rolle im gesamten Nachrichtenfluss festgelegt werden. Darüber hinaus möglich sind Nachrichtenveränderung (einfügen, löschen und ersetzen von Daten innerhalb einer Nachricht), Nachrichtenvalidierung (Prüfung von XML-Daten gegen eine WSDL- bzw. XML-Schema-Datei), inhaltsbezogenes Routing (dazu gehört auch die Berücksichtigung unterschiedlicher Sicherheitsanforderungen) und Fehlerbehandlung. In Bezug auf Fehlerbehandlung stehen Möglichkeiten wie die Wahl eines alternativen Endpunktes, das Versenden einer Fehlernachricht, die Protokollierung von Fehlern zur Erstellung späterer Reports oder die Weiterverarbeitung der Nachricht nach erfolgter Kontextanpassung bereit (vgl. [BEA05b], S. 7 ff.; [BEA06h], S. 1-7 ff.).

Die Verwaltung und Konfiguration des ESB fällt dem Configuration Framework zu. Dieses wird über die BEA AquaLogic Service Bus Console bzw. ein Change Center gesteuert und erfüllt Aufgaben wie die Speicherung von Metadaten, Registrierung von Services (lokal oder mittels BEA AquaLogic Service Registry), Validierung von Services und Durchführung von Metadatenänderungen. Die Speicherung von anfallenden Metadaten kann über Ordner und Verzeichnisse erfolgen, die unternehmens-, abteilungs- oder projektweit zugeordnet werden. In Bezug auf Integrität der darin abgelegten Daten werden Änderungen an diesen (soweit möglich) automatisch validiert und erst danach durchgeführt. Weiterhin ist neben der Verwendung der lokalen Registry bzw. der BEA AquaLogic Service Registry im Prinzip die Nutzung sämtlicher UDDI-konformer Registries möglich. Über diese Registries können

[28] Gemeint ist damit der Zugriff (Lesen und Schreiben) auf Dateien innerhalb eines Dateisystems, der sich im Falle eines Netzwerkdateisystems auch über größere Entfernungen und Ressourcen hinweg erstrecken kann.

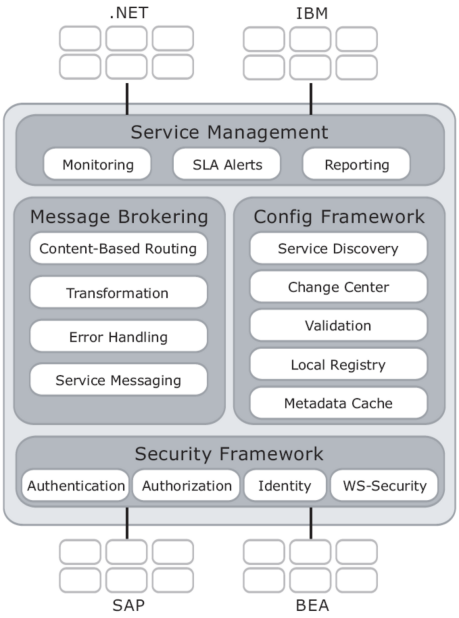

Abbildung 3.10: BEA AquaLogic Service Bus (Quelle: [BEA05b], S. 7).

Services registriert, gesucht und importiert werden. Ebenfalls unterstützt wird die Versionierung von Services, d. h. dass parallel mehrere Versionen eines Services existieren können (vgl. [BEA05b], S. 7 ff.; [BEA06h], S. 1-7 ff., S. 3-10 ff., S. 4-1 ff.).

Alle Fragen zur Sicherheit werden durch das Security Framework abgedeckt. Dieses realisiert Anforderungen wie Authentifizierung, Autorisation und Verschlüsselung auf mehreren Ebenen. Die erste Ebene bildet die Nutzerverwaltung, also das Anlegen und Ändern von Nutzern, Nutzergruppen, Rollen und Rechten. Ein weiterer Punkt ist die Sicherheit auf der Transportebene, die bspw. mittels HTTPS realisiert wird und so Authentifizierung und Integrität sicherstellen kann. Letzte Ebene ist die Nachrichtensicherheit. Hier kann das Signieren und Verschlüsseln von Nachrichten erfolgen, das sich auf Standards wie SAML, WS-Security und WS-Policy stützt (vgl. [BEA05b], S. 24 f.; [BEA06h], S. 3-1 ff.).

Ein letzter großer Bereich des BEA AquaLogic Service Bus ist das Service Management, das die Punkte Monitoring, SLA-Warnungen und Reporting umfasst. Unter den Punkt Monitoring fällt das Sammeln und Bereitstellen von Daten und Statistiken zur Laufzeit. Über eine anpassbare Oberfläche (BEA AquaLogic Service Bus Console Dashboard) erfolgt in einem weiteren Schritt die flexible Darstellung und Auswertung dieser Daten. Fehler oder Probleme, die möglicherweise auftreten, können somit frühzeitig erkannt und isoliert werden. Weiterhin, die Qualität von Services betreffend, erlaubt der BEA AquaLogic Service Bus entsprechende SLA-Warnungen auszugeben. Werden bspw. kritische Werte für Fehlschlagrate, Fehleranzahl, Validierungsfehler oder Antwortzeit überschritten, so werden Fehlernachrichten unterschiedlicher Priorität erzeugt und an zuvor definierte Empfänger verschickt. Als nachteilig erweist sich jedoch, dass keine Maßnahmen enthalten bzw. dokumentiert sind, wie überhaupt SLA ausgehandelt, definiert, gespeichert oder überwacht werden können und wie bei Nichteinhaltung verfahren wird. Eine weitere, jedoch weniger kritische Funktionalität ist das Reporting. Hierbei ist es möglich, Nachrichten bzw. Nachrichtenteile zu filtern und den jeweiligen Inhalt zur Erstellung späterer Reports zu speichern. Über ein enthaltenes grafisches Reportingmodul können dann Reports in unterschiedlicher Detailgenauigkeit betrachtet werden (vgl. [BEA05b], S. 24 f.; [BEA06h], S. 3-1 ff.).

Die notwendige Infrastruktur zur Umsetzung des BEA AquaLogic Service Bus ist der BEA WebLogic Server, der die Laufzeitumgebung bereitstellt. Dabei besteht die Wahl, den ESB als einzelnen Server oder als ein, mehrere Server umfassendes Cluster zu betreiben. In beiden Fällen fungiert aber immer genau ein Server als Administrationsserver, dessen Aufgabe die zentrale Verwaltung (des Clusters) darstellt. Änderungen an der Konfiguration oder anderen Metadaten werden über diesen Server propagiert. Obwohl die Verwendung von Applikationsservern höchste Anforderungen in Sachen Verfügbarkeit, Skalierbarkeit und Zuverlässigkeit erfüllen kann, entspricht sie aber weniger dem Konzept leichtgewichtiger Service-Container. Die entsprechend notwendige Messaging Infrastruktur ist damit bereits integriert, weshalb sich das bei BEA verfolgte Konzept eines ESB als eher monolithisch darstellt (vgl. [BEA06h], S. 1-7 f.).

Der *Funktionsumfang* des eigentlichen BEA AquaLogic Service Bus umfasst wie bereits angesprochen:

- standardbasierte Kommunikation und Integration,
- Transformation,
- inhaltsbezogenes Routing,
- Konfiguration, Monitoring und Management von Services,
- Management von Servicelebenszyklen (speziell Verwaltung von Metadaten und Versionierung),
- Sicherheit (QoP),
- Dienstgüte (QoS),
- Skalierbarkeit.

In Sachen *Flexibilität* müssen infolge der bereits angesprochenen Architektur Abstriche gemacht werden. Zwar werden zahlreiche Betriebssysteme (Microsoft Windows, Red Hat Linux, Solaris/SunOS und HP-UX) unterstützt, jedoch ist stets die Verwendung des BEA WebLogic Servers notwendig. Trotzdem können über Adapter andere, proprietäre Anwendungen und Protokolle angesprochen werden, auch eine Kapselung dieser über den Einsatz von Proxy Services ist möglich. Ein Mindestmaß an Flexibilität wird somit gewährleistet.

Zusammenfassend betrachtet erfüllt der BEA AquaLogic Service Bus im Kern lediglich essentielle Anforderungen und Funktionalitäten eines ESB. Weiterführende Funktionen wie die Abbildung von Geschäftsprozessen, Service Orchestrierung oder Service Choreographie sind mit dem BEA AquaLogic Service Bus erst über den Einsatz komplementärer Produkte möglich. Der Aspekt der Föderation und Erweiterbarkeit des BEA AquaLogic Service Bus taucht an keiner Stelle auf und, so ist zu vermuten, er spielt bei BEA nur eine untergeordnete Rolle. Über besondere *Qualitätseigenschaften* verfügt der BEA AquaLogic Service Bus kaum und bezüglich seiner *Definitionskonformität* muss die Frage erlaubt sein, in wieweit er der in Kapitel 2.3 beschriebenen Architektur entspricht. So beruht der BEA AquaLogic Service Bus weniger auf leichtgewichtigen, einfach verteil- und einsetzbaren Service-Containern als vielmehr auf den Applikationscontainern des BEA WebLogic Servers. Eine weitere Schwäche stellt die fehlende Möglichkeit einer direkten Datenbankanbindung bspw. mittels JDBC dar. Datenbankanbindungen sind zwar möglich aber nur über den BEA WebLogic Server als entsprechende Mittelschicht. Ein letzter Punkt betrifft die BEA AquaLogic Produktfamilie. Wie andere Anbieter auch versucht BEA, eine Vielzahl komplementärer Produkte zum BEA AquaLogic Service Bus zu vertreiben. Diese besitzen aber sich teilweise überschneidende

Funktionalitäten und werden darüber hinaus losgelöst voneinander betrachtet und vertrieben. Ein von einer Produktfamilie zu erwartender Gesamtzusammenhang und speziell das Zusammenspiel mit dem BEA AquaLogic Service Bus als Kernkomponente fehlt, weshalb sich auch der ESB nach BEA als nicht vollends ausgereift darstellt.

3.7 ESB nach Oracle

Ähnlich wie andere ESB-Hersteller bietet auch Oracle nicht nur eine reine ESB-Implementierung, sondern in Form der Oracle SOA Suite eine Reihe von Produkten rund um den Oracle ESB an. Zu diesen erweiterten *Bestandteilen* gehören Oracle BPEL Process Manager, BAM, Business Rules, Web Service Manager, Service Registry, JDeveloper und Connectivity. Diese ergänzen den Oracle ESB und bieten einige der von einer ESB-Implementierung geforderten Funktionen und darüber hinaus noch weitere an (vgl. [Ora06d], S. 1).

Der Oracle BPEL Process Manager stützt sich, wie der Name bereits sagt, auf BPEL zur Beschreibung von Geschäftsprozessen. Seine GUI, der Process Designer wird in Form eines Plugins zum JDeveloper (der dafür benötigt wird) bereitgestellt. Darüber hinaus gibt es eine Web-basierte Konsole für Management, Administration oder Debugging. Die Hauptkomponente ist die Oracle BPEL Engine, die als eigenständiger Server BPEL-Prozesse ausführt und steuert. Sie verfügt weiterhin über eigenständige Transformationsmöglichkeiten (mittels XSLT, XPath und XQuery) und kann über JMS, JCA oder Web Services entweder innerhalb eines ESB aber auch separat in anderen Einsatzumgebungen betrieben werden (vgl. [Ora04], S. 1 f.).

Oracle BAM dient in diesem Zusammenhang zur ereignisgesteuerten und nachrichtenbasierten Überwachung von Geschäftsprozessen. Das bedeutet konkret, Service Levels und Leistungsindikatoren von Prozessen mittels Übersichtsanzeigen, Warnmeldungen oder Reports in Echtzeit zu überwachen. Folglich wird nicht nur das Einhalten von SLA ermöglicht, sondern auch ein Blick auf Unternehmensabläufe und -prozesse in Echtzeit (vgl. [Ora06a], S. 1 ff.).

Neben dem Oracle Process Manager und Oracle BAM gibt es mit Oracle Business Rules eine weitere Komponente zum Thema Geschäftsprozesse. Ihre Aufgabe besteht im Wesentlichen darin, Regeln und Politiken für Geschäftsprozesse zu definieren und umzusetzen. So gehören neben einem Werkzeug zur Beschreibung von Geschäftsregeln auch ein SDK[29] und eine auf Java gestützte Engine zur Umsetzung dieser Regeln mit dazu. Kritisch hinterfragt werden muss, warum das Werkzeug zur Beschreibung von Geschäftsregeln eigenständig arbeitet und nicht in den Oracle JDeveloper integriert ist und noch wichtiger, wie die eigentliche Engine und das dazugehörige Repository (zur Speicherung der Geschäftsregeln) in einer serviceorientierten Weise genutzt werden können. Überhaupt ist ein Zusammenspiel im Kontext einer SOA nicht klar erkennbar und daher zu bemängeln (vgl. [Ora06b], S. 1 f.).

Der bereits angesprochene Oracle JDeveloper dient als grafisch unterstützte Entwicklungsumgebung und ist für eine Vielzahl von Anwendungsszenarien geeignet. Diese reichen von der Anwendungsentwicklung (auf Basis von Java, J2EE, SQL usw.) bis hin zur Entwicklung und Kontrolle von Services sowie deren Lebenszyklen[30] (vgl. [Ora06d], S. 4).

Die Oracle Service Registry und Oracle Connectivity sind erweiterte Komponenten der Oracle SOA Suite, die im Gegensatz zu anderen nicht in sinnvoller Weise eigenständig betrieben werden können. Die Oracle Service Registry stellt den zentralen Anlaufpunkt dar, um Ser-

[29] Software Development Kit

[30] Nach Oracle gehört dazu Modellierung, Implementierung, Deployment, Test und Überwachung von Services.

vices zu veröffentlichen und sie Nutzern bzw. anderen Diensten bereitzustellen. Neben dem Suchen und Finden von Services (auf Basis von UDDI), geht es gleichfalls darum, weitere, im Rahmen einer SOA anfallenden Metadaten zu erfassen und zu verwalten. Oracle Connectivity hingegen dient als Sammlung von Adaptern und gewährleistet die notwendige Konnektivität des Oracle ESB. Das Angebot an Adaptern reicht von Java-basierten Adaptern (JCA und JMS), über Adapter für Mainframe- und geschlossene Anwendungen (SAP, Peoplesoft, Siebel, CICS, VSAM u. a.), Datenbankadapter (etwa für Oralce, IBM DB2 oder MS SQL Server) bis zu Kommunikationsadaptern (bspw. Oracle AQ, TIBCO Rendezvous oder WebSphere MQ). Hinzu kommt eine Unterstützung von B2B-Protokollen und -Formaten (wie EDI, RosettaNet, ebXML oder UBL) (vgl. [Ora06d], S. 4).

Die wichtigste Komponente neben dem Oracle ESB ist der Oracle Web Services Manager, der sich bei genauer Betrachtung als ein halber ESB darstellt. Ihm kommt die Aufgabe zu, fast sämtliche für Web Services relevanten Politiken (zu den Themen Sicherheit, Identifikation, Zugriffskontrolle, Protokollierung und Validierung) zu verwalten und umzusetzen. Sein Aufbau ist in Abbildung 3.11 dargestellt. Ausgangspunkt für die Umsetzung von Politiken ist der Policy Manager, der über eine Konsole angesprochen wird. Hier können Politiken definiert, gespeichert und an Gateways oder Agenten ausgeliefert werden. Gateways, die (mehreren) Web Services vorgeschaltet werden, dienen wie Agenten der Umsetzung von Politiken. Agenten sind im Gegensatz zu Gateways granularer aufgebaut und werden auf eine direktere Art und Weise in Web Services integriert (residieren am selben Ort). Die Überwachung von Gateways und Agenten sowie eine Visualisierung in Form von Zustandsübersichten erfolgt über einen Monitor. Dies geschieht in Echtzeit und gestattet bei Überschreitung von Grenzwerten Warnungen auszugeben oder weitere Maßnahmen einzuleiten (vgl. [Ora05a], S. 1 ff.).

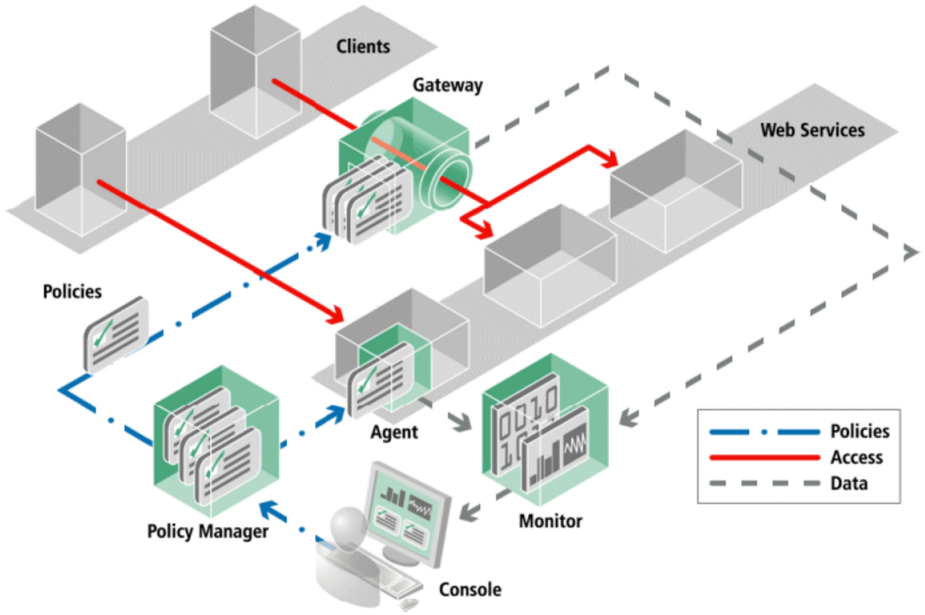

Abbildung 3.11: Oracle Web Services Manager (in Anlehnung an [Ora05a], S. 5).

Schaut man sich den Umfang der Politiken und die Fähigkeiten speziell der Gateways an, so entpuppen sich diese als halbe Service-Container. Sie dienen neben der Umsetzung von Politiken ebenso der Virtualisierung von Services, dem Routing von Nachrichten und der Transformation von unterschiedlichen Datenformaten. Weiterhin können Sicherheitsmaßnahmen für Zugriffskontrolle und Rollenvergabe (Autorisation), Authentifizierung und Verschlüsselung

über Standards (SAML, WS-Security) sowie externe Anwendungen (LDAP, COREid, Netegrity) realisiert werden. Abgesehen von diesen Sicherheitspolitiken und -maßnahmen können natürlich auch Dienstgüten gemäß SLA überwacht werden. Abgerundet wird das Spektrum angebotener Funktionen durch die verschiedenen Möglichkeiten zur Nachrichtenübertragung (synchron/asynchron), zu Nachrichtenformaten (XML-basiert mit oder ohne SOAP) und zu Nachrichtenprotokollen (HTTP, HTTPS, JMS, WebSphere MQ, etc.) (vgl. [Ora05b]).

Der Oracle ESB ist trotz der sich überschneidenden Funktionen zum Oracle Web Service Manager der Kern der Oracle SOA Suite. Er präsentiert sich als multiprotokollfähige Integrationsplattform, deren *Aufbau* im Kern aus einer leistungsfähigen Nachrichteninfrastruktur in Form vom MOM (Oracle Enterprise Messaging Service/Oracle AQ) besteht. Über Standards wie JMS und JCA ist es auch möglich, andere MOM wie Sonic MQ, WebSphere MQ oder TIBCO Rendezvous anzusprechen und einzubinden. Offen bleiben jedoch Fragen zum Aufbau des ESB selbst. So bleibt unklar, in wieweit das Konzept leichtgewichtiger Service-Container bei Oracle Berücksichtigung findet. Es muss vielmehr davon ausgegangen werden, dass es solche Container bei Oracle nicht gibt und anstelle dieser Applikationscontainer Verwendung finden. Dafür spricht, dass der Oracle ESB komplett in Oracles Applikationsserver (Oracle AS) integriert ist. Aufgrund fehlender Informationen kann über die Frage, wie genau sich diese Integration darstellt, nur spekuliert werden (vgl. [Ora06c], S. 1).

Klarer als der Aufbau des Oracle ESB kann der bereitgestellte *Funktionsumfang* beschrieben werden. Zu den Funktionen, die sowohl durch Komponenten der Oracle SOA Suite als auch durch den Oracle ESB zur Verfügung gestellt werden, gehören (vgl. [Ora06c], S. 1 f.):

- standardbasierte Kommunikation und Integration,
- Unterstützung und Abbildung von (Geschäfts-)Prozessen bzw. Prozessketten,
- Transformation,
- intelligentes Routing,
- Konfiguration, Monitoring und Management von Services,
- Management von Servicelebenszyklen,
- Sicherheit (QoP),
- Dienstgüte (QoS),
- Transaktionsverwaltung,
- Skalierbarkeit,
- Föderation und Erweiterbarkeit.

Zur Frage hinsichtlich der *Definitionskonformität* sind beim Oracle ESB verschiedene Aspekte zu berücksichtigen. Auf der einen Seite steht eine ESB-Implementierung, die wesentliche funktionale Anforderungen erfüllt. Den Kern und in soweit besteht ebenso Konsens, bildet eine auf MOM aufbauende Nachrichteninfrastruktur. Den weiteren Aufbau des ESB betreffend und das ist die andere Seite, erweist sich dieser als weniger definitionskonform. Gestützt auf einen Applikationsserver entspricht er weniger dem Konzept leichtgewichtiger Service-Container und stellt sich damit auch in Sachen Deployment und geforderter *Flexibilität* als weniger leistungsfähig heraus. Bedingt durch diesen als schwergewichtig zu bezeichnenden Aufbau, können selbst zahlreiche Adapter die damit verbundenen Defizite nicht vollständig ausgleichen.

Die *Qualitätseigenschaften* der von Oracle angebotenen ESB-Implementierung (im erweiterten Sinne) werden allen voran durch das reichhaltige Funktionsangebot und die umfangreichen Werkzeuge bestimmt. Andererseits stellt gerade das auch die größte Schwäche dieser Implementierung dar. Bedingt durch die Art und Weise wie Funktionen bereitgestellt werden, nämlich durch separate, eigenständige und nur teilweise auf Zusammenarbeit abgestimmte Komponenten, ergibt sich das Bild einer noch nicht vollends ausgereiften Lösung in Sachen ESB. Insbesondere die sich überschneidenden Funktionen einzelner Komponenten fallen negativ ins Gewicht. Wiederum positiv zu sehen sind die angebotenen Möglichkeiten in Bezug auf Dienstgüte (und deren Überwachung gemäß SLA) sowie vielerlei Unterstützung im B2B-Umfeld. Hier unterscheidet sich der Oracle ESB auf positive Art von vielen anderen angebotenen Implementierungen.

Im Ganzen gesehen, stellt sich die Lösung von Oracle zum Thema ESB als sehr durchwachsen dar. Es gibt zwar eine umfangreiche Menge an einzelnen Komponenten und damit verbundenen Funktionen, doch sind diese nicht in dem Maße aufeinander abgestimmt, wie man es von einem homogenen Produkt bzw. Produktfamilie erwarten würde. Positiv zu nennen sind zahlreiche Werkzeuge, allen voran der JDeveloper als leistungsfähige Entwicklungsumgebung. Dessen Anbindung an den Oracle ESB ist als sehr gut zu bezeichnen und damit ein positiver Aspekt bei dessen Bewertung. Weniger gut ist die Tatsache, dass andere Werkzeuge einzeln angeboten werden und nicht miteinander (oder in den JDeveloper) integriert sind. So ergibt sich ein buntes Sammelsurium an Werkzeugen, das sich in der Praxis als wenig übersichtlich darstellen dürfte. Ähnlich verhält es sich mit den Komponenten selbst, die sich teilweise in Sachen Funktionalität überschneiden und häufig einen sehr eigenständigen Charakter aufweisen. Besonders stark fällt dies beim Oracle Web Services Manager ins Gewicht, dessen Überschneidungen zum Oracle ESB gravierend sind. Den Schlusspunkt bildet eine Architektur, die nur bedingt als definitionskonform zu bezeichnen ist und vor allem bezüglich Flexibilität Defizite aufweist.

3.8 ESB nach MuleSource

Mule, eine im Sinne von Open Source und der MuleSource Public License freie ESB-Implementierung, gehört schon seit längerer Zeit zum Kreis angebotener ESB-Implementierungen. Im Gegensatz zu seinen kommerziellen Pendants besteht Mule lediglich aus zwei größeren Komponenten. Diese *Bestandteile* sind einerseits Mule als Implementierung des ESB sowie andererseits Mule IDE als grafische Entwicklungsumgebung und Schnittstelle zum ESB. Mule wird hierbei als eine Nachrichtenplattform angesehen, deren Aufbau der ESB-Architektur folgt. Er verfügt über UMO[31]-Container (mit Service-Containern vergleichbar und teilweise auch als solche bezeichnet) und im Kern über einen Nachrichtenbus, der für Nachrichtentransport und Routing zuständig ist. Den prinzipiellen Aufbau zeigt Abbildung 3.12. Darin dargestellt wird die Fähigkeit, zahlreiche Anwendungen, Systeme, Web Services und fortführende Funktionalitäten (BPEL, E-Mail, Instant Messaging, etc.) an den ESB anzuschließen (vgl. [Mul06e]).

Bevor mit Mule der eigentliche ESB näher betrachtet wird, soll an dieser Stelle kurz die Mule IDE vorgestellt werden. Sie ist in Form eines Plugins für Eclipse realisiert und dient der Steuerung von Mule. Im Vergleich zu den Entwicklungsumgebungen bzw. grafischen Schnittstellen anderer (kommerzieller) ESB-Anbieter erweist sich die Mule IDE als rudimentär. Zu diesen rudimentären Fähigkeiten gehört das Bereitstellen von serviceähnlichen Komponenten in entsprechenden Containern und deren Konfiguration über Skripte. Dies alles geschieht

[31] UMO – Universal Message Object

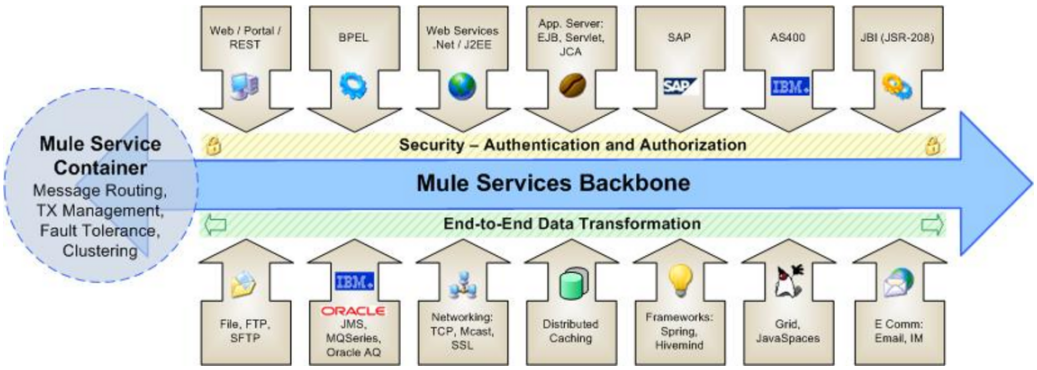

Abbildung 3.12: Mule ESB (Quelle: [Mul06b]).

auf einer sehr implementierungsnahen Ebene, die ein hohes Maß an technischem Verständnis und Kenntnissen erfordert. Sonstige grafische Unterstützung, die über einen einfachen Texteditor hinausgeht, wird kaum angeboten. Ebenso fehlt eine Entwicklungsunterstützung von (Geschäfts-)Prozessen, Service Orchestrierung oder Service Choreographie. Eine letzte große Schwachstelle ist die fehlende Möglichkeit, auf einfache Weise bereits vorhandene Services suchen, finden und einbeziehen zu können (vgl. [Mul06f]).

Neben der Mule IDE stellt sich der zentrale *Aufbau* von Mule in folgender Weise dar. Im Mittelpunkt stehen so genannte Universal Message Objects (UMO), die in Containern (Mule Manager Instances) residieren (siehe Abbildung 3.13). Sie kapseln externe Anwendungen und stellen die notwendige Integrationslogik bereit. Über HTTP, SMTP bzw. eine entsprechende JMS-basierte Nachrichteninfrastruktur (Apache ActiveMQ) erfolgt dann der Nachrichtenaustausch. Ebenso möglich ist es, mittels JMS auch andere MOM (Oracle AQ, SonicMQ, Websphere MQ, etc.) zu nutzen (vgl. [Mul06c]).

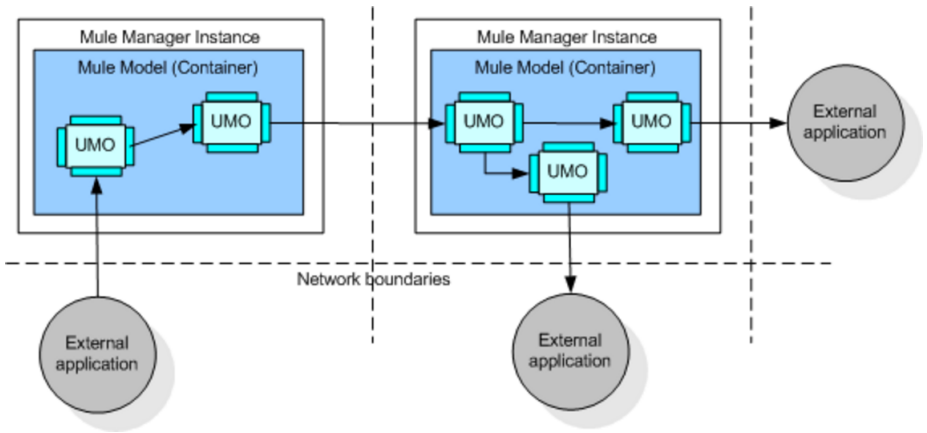

Abbildung 3.13: Mule UMO-/Service-Container (Quelle: [Mul06c]).

Der Weg einer über Mule verschickten Nachricht wird in Abbildung 3.14 gezeigt. Message Receiver (als Teil eines Transport Providers) dienen hier als Adapter zu externen Anwendungen, Systemen oder Services und können über das vom jeweiligen Transport Provider bereitgestellte Protokoll kommunizieren. Mule liefert Transport Provider für SOAP, HTTP, SMTP, JMS u. a. Protokolle. Sie dienen als Kapselung und reichen Nachrichten an den Inbound Router weiter. Dieser entscheidet, an welche UMO Component eine Nachricht weitergeleitet

werden soll. Ebenfalls möglich ist es Nachrichten durch den Inbound Router zu filtern, aufzuteilen oder zu aggregieren. Nachdem die Nachricht die UMO Component erreicht hat und dort verarbeitet wurde, erfolgt in der Regel das Versenden einer Antwortnachricht. Hierfür wird die Nachricht an den Outbound Router gesendet und von dort aus an möglicherweise verschiedene Transport Provider (UMO Components oder externe Anwendungen) weitergeleitet. Über einen Dispatcher erfolgt in einem letzten Schritt die Zuteilung der Nachrichten an die jeweiligen Empfänger (vgl. [Mul06a]).

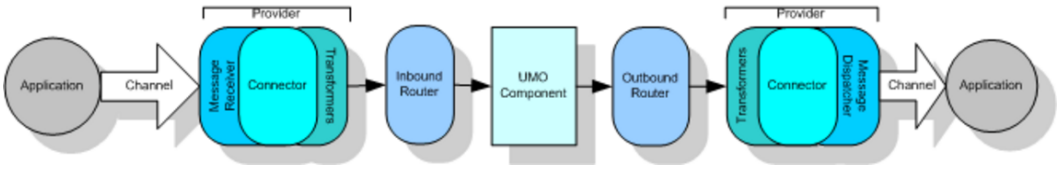

Abbildung 3.14: Mule Nachrichtentransport (Quelle: [Mul06a]).

Das Management von Mule bzw. der UMO-Container erfolgt im Wesentlichen über Skripte. Zusätzlich wird ein Configuration Grapher angeboten, der in der Lage ist, Konfigurationsdateien grafisch aufbereitet in Form von Diagrammen darzustellen. Eine weitere Möglichkeit zur Konfiguration stellt eine über JMX ansprechbare Managementschnittstelle dar. Sie erlaubt neben der Verwaltung des gesamten UMO-Containers auch die Verwaltung von Lebenszyklen einzelner UMO Components (vgl. [Mul06d]).

Obwohl es keine grafische Unterstützung für die Entwicklung von Geschäftsprozessen gibt, so ist es trotzdem möglich diese umzusetzen. Dazu wird durch PXE[32] eine BPEL-Engine bereitgestellt, mit der basierend auf BPEL beschriebene Geschäftsprozesse ausgeführt werden können. Der *Funktionsumfang* von Mule umfasst darüber hinaus (vgl. [Mul06e]):

- standardbasierte Kommunikation und Integration,
- bedingte Unterstützung und Abbildung von (Geschäfts-)Prozessen bzw. Prozessketten,
- Transformation,
- intelligentes Routing,
- Management von UMO Components (Services) und deren Lebenszyklen,
- Sicherheit (QoP),
- Transaktionsverwaltung,
- flexibles Deployment,
- Skalierbarkeit,
- Föderation und Erweiterbarkeit.

Die *Flexibilität* von Mule und gleichzeitig eine besondere *Qualitätseigenschaft* zeigt sich in den verschiedenen Deployment-Topologien. So unterstützt Mule neben der typischen Bus-Topologie auch Topologien wie Punkt-zu-Punkt oder Hub-And-Spoke. Dadurch stellt er sich,

[32] FiveSight Process eXecution Engine

hinsichtlich der Flexibilität, als äußerst anpassungsfähig dar. Eine Steigerung dieser Anpassungsfähigkeit ergibt sich durch die Möglichkeit, unterschiedliche Topologien miteinander zu verbinden. Auf diesem Weg ist es möglich, auch komplexere Nachrichtenverbindungen aufzubauen. Weiterhin als positiv herauszustellen sind die zahlreichen Plattformen (Windows und verschiedene Unix- und Linuxderivate), für die Mule (in Java implementiert) verfügbar ist (vgl. [Mul06e], [Mul06c]).

Zum Thema *Definitionskonformität* stellt sich jedoch die Frage, ob Mule tatsächlich ein ESB ist. Festgehalten werden kann, dass Mule zum einen auf den Prinzipien einer ESB-Architektur zum anderen aber auf dem Gedanken der Objektorientierung aufgebaut ist. So findet die Integration über Nachrichtenobjekte statt und weniger über Services. Diese Tatsache, die sich bei einem ersten Blick als eine reine Definitionsfrage darstellt, entpuppt sich bei genauerer Betrachtung durchaus als folgenreich. Zwar können verschiedenste Nachrichten und Nachrichtenformate genutzt und mit XML Schema, XSLT und XQuery gleichfalls validiert und transformiert werden, dennoch fehlt es gerade im Bereich der WS-Standards an Unterstützung. Im Vergleich zu anderen Implementierungen zeigt sich, dass Mule nicht primär für den Aufbau einer SOA entworfen wurde. Sieht man davon ab, stellt sich Mule trotz alledem als eine vollwertige und im Wesentlichen definitionskonforme ESB-Implementierung dar.

Bezüglich des Aufbaus erweist sich Mule als äußerst flexibel, hat aber auch Schwächen. Diese liegen vor allem im Leistungsumfang, der mit dem von kommerziellen Implementierungen nur bedingt mithalten kann. Größtes Manko in diesem Zusammenhang ist eine fehlende Registry zur Verwaltung der vorhandenen bzw. integrierten Services und Nachrichtenobjekte. Hier wird das Konzept der Nachrichten austauschenden Objekte schnell zum Problem, da etwa eine darauf aufbauende Registry fehlt und die Nutzung UDDI-konformer Registries nicht möglich ist. Ähnlich unbefriedigend stellt sich die grafische und werkzeugbasierte Unterstützung dar, die trotz Mule IDE noch nicht den Leistungsumfang aufweist, den man von einer professionellen Implementierung erwarten würde. Können diese Schwächen und Leistungsdefizite in Zukunft behoben werden, so entsteht mit Mule eine ESB-Implementierung, die den Vergleich mit kommerziellen Produkten nicht zu scheuen braucht.

3.9 ESB nach Microsoft

Microsoft gehört zu den Softwareunternehmen, die kein direkt als ESB bezeichnetes Produkt vertreiben. Vielmehr ist man hier der Meinung, die Funktionalitäten und Anforderungen, die an einen ESB gestellt werden, mit bereits vorhandenen Produkten abdecken zu können. Die Produkte, die dabei eine Rolle spielen, sind der Microsoft BizTalk Server, ein Integrationsserver mit Geschäftsprozessunterstützung und die Windows Communication Foundation (WCF) (ehemals Indigo Plattform), eine Framework zur Entwicklung sicherer und zuverlässiger Web Services. Der BizTalk Server stellt sich in diesem Rahmen als Integrationsplattform mit Geschäftsprozessunterstützung dar, die es erlaubt, Systeme von SAP, Siebel, Peoplesoft bis hin zu MSMQ und Websphere MQ miteinander zu verbinden. Die WCF hingegen ermöglicht die notwendige Serviceunterstützung (Sicherheit, Zuverlässigkeit und Interoperabilität von Services) in Form eines eigenständigen Frameworks. Dessen umfangreiche Standards aus dem Bereich Web Services wie SOAP, WSDL, XSLT, XPath, WS-Adressing, WS-Policy, WS-Security, WS-Trust, WS-SecureConversation, WS-ReliableMessaging, WS-AtomicTransaction oder WS-Coordination ermöglichen dem BizTalk Server darauf aufbauend eine serviceorientierte Integration (vgl. [Nic05]; [Mic06b]).

Bezüglich einer einheitlichen Definition des Begriffs ESB kritisiert Microsoft, dass ihrer Meinung nach eine solche fehlt und der Begriff ESB keine Architektur, sondern lediglich eine Klasse von Produkten bezeichnet, deren Ursprung sich typischerweise im Bereich von EAI

oder MOM finden lässt. Daran angelehnt definiert man bei Microsoft den ESB als eine Sammlung von Funktionalitäten wie Kommunikation, Routing, Transformation und Serviceunterstützung, die eine Integration der IT-Ressourcen entlang von Geschäftsprozessen innerhalb einer Organisation ermöglichen. Eine Definition, die ganz bewusst den Aspekt einer zugrundeliegenden Architektur ausblendet und so die von Microsoft angebotene Lösung in keinem großen Widerspruch zu anderen ESB-Implementierungen erscheinen lässt (vgl. [Nic05]; [Mic06b]; [IDG06a]).

Dieser Denkweise folgt ebenso die Entscheidung bei Microsoft, ganz bewusst auf den Begriff ESB zu verzichten. Dies hat verschiedene Ursachen. Ein Grund ist, dass man der Meinung ist, weit mehr an Funktionalität anzubieten, als es bei traditionellen ESB-Implementierungen der Fall ist. Zu diesen gehören etwa eine umfassende Unterstützung von Geschäftsprozessen, BAM, Verwaltung von Geschäftsregeln und die Möglichkeit zur Abbildung von Geschäftsbeziehungen mit zahlreichen Geschäftspartnern (vgl. [Nic05]; [Mic06b]). Ein weiterer möglicher Grund ist der, dass man die Popularität des ohnehin starken ESB-Markts nicht noch weiter steigern und auch nicht in einen direkten Konkurrenzkampf mit den dort vertretenen Anbietern treten möchte. Zu befürchten wäre dann nämlich, dass potenzielle Kunden Vergleiche mit anderen ESB-Produkten machen würden und sich möglicherweise sehr schnell Defizite an dem bei Microsoft verfolgten Konzept zeigen würden.

Bei einem Blick auf die Funktionalitäten von BizTalk Server und WCF scheinen kaum Wünsche offen zu bleiben. Dennoch, betrachtet man nämlich die bei Microsoft verfolgte Architektur und die Architektur eines ESB wie sie in Kapitel 2.3 beschrieben ist, ergeben sich massive Unterschiede. So stellt sich der BizTalk Server als ein mächtiger und skalierbarer Integrationsserver dar, der aber sämtliche Integrationsfunktionalitäten monolithisch kapselt und diese nicht wie ein definitionskonformer ESB in Form von flexibel separierbaren, leichtgewichtigen Service-Containern bereitstellt. Würde man bspw. eine XSLT-basierte Transformationskomponente benötigen, so wäre dies nur im Rahmen eines nahezu kompletten BizTalk Servers möglich. Im Vergleich dazu ist ein auf leichtgewichtigen Service-Containern beruhender ESB in der Lage, einzelne Funktionalitäten sehr leicht zu entkoppeln und flexibel, den jeweiligen Anforderungen entsprechend, zu skalieren. Egal ob es sich um Funktionalitäten wie inhaltsbezogenes Routing, Transformation oder Adapter für proprietäre Anwendungen und Formate handelt, alle können trotz ihrer logischen Verknüpfungen physisch unabhängig voneinander betrieben und einzeln unterschiedlichen Leistungsanforderungen angepasst werden. Dies gestattet z. B., dass ein und dieselbe Funktionalität durch mehrere Service-Container und mehrere Services innerhalb dieser bereitgestellt werden kann, unabhängig davon, dass diese Funktionalität auf der logischen Ebene nur an einer Stelle angeboten wird. Derartige maßgeschneiderte Anpassungen sind in der Form mit einem bzw. mehreren BizTalk Servern nicht oder zumindest nur sehr bedingt möglich. Wenngleich sich bspw. auch der BizTalk Server auf einzelne Funktionalitäten reduzieren lässt und diese flexibel zur Verfügung stellen kann, so erweist er sich dennoch wesentlich umfangreicher als ein leichtgewichtiger Service-Container. Zusätzlich ist er mit der Beschränkung auf das Microsoft Server Betriebssystem und der Abhängigkeit zum Microsoft SQL Server bei weitem nicht so flexibel und würde mit dem zugrundeliegenden Lizenzmodell wahrscheinlich auch wesentlich teurer sein als vergleichbare ESB-Implementierungen. Es ist leicht erkennbar, dass der BizTalk Server nicht für das Konzept der leichtgewichtigen Service-Container geschaffen wurde, sondern sich vielmehr als ein skalierbarer und nur bedingt verteilbarer Integrationsserver (Microsoft spricht von einer verteilten Hub-and-Spoke Architektur) mit umfangreicher Serviceunterstützung darstellt (vgl. [Cha05b]; [Mic06a]).

Die Frage, die sich abschließend stellt, ist die, wie der Ansatz von Microsoft im Ganzen zu bewerten ist. Zum einen ist festzuhalten, dass nahezu alle Funktionalitäten, über die eine ESB-Implementierung verfügen sollte, vorhanden sind. Dazu gehören (vgl. [Nic05]; [Mic06b]):

- standardbasierte Kommunikation und Integration,
- Unterstützung und Abbildung von (Geschäfts-)Prozessen bzw. Prozessketten,
- Service Orchestrierung/Choreographie,
- Transformation,
- intelligentes Routing,
- Konfiguration und Management von Services,
- bedingte Unterstützung von Servicelebenszyklen,
- Sicherheit (QoP),
- Dienstgüte (QoS),
- Transaktionsverwaltung,
- Skalierbarkeit.

Mit diesen Eigenschaften kann sich der BizTalk Server sehr wohl mit anderen ESB-Implementierungen messen. Einzige Schwächen liegen (wie bereits zuvor erwähnt) im Bereich des flexiblen Deployments und einer daraus resultierenden beschränkten Möglichkeit zur Föderation und Erweiterbarkeit. Zum anderen entspricht die konkrete Implementierung von Microsoft nicht dem im Kapitel 2.3 beschriebenen Aufbau eines ESB. Abgesehen von den sich daraus ergebenden negativen Konsequenzen kann der Ansatz von Microsoft demnach nicht als definitionskonform angesehen werden. Folglich ist die Kombination von Microsoft BizTalk Server und WCF nicht als eine ESB-Implementierung im engeren Sinne zu betrachten. Was bleibt, sind eine Menge von Funktionalitäten, die anderen ESB-Implementierungen durchaus ebenbürtig sind, jedoch in letzter Konsequenz keine ESB-konforme Architektur.

3.10 ESB nach Sun Microsystems

Bei Sun Microsystems wurde mit dem Zukauf von SeeBeyond Technology Software Corp. (ein Anbieter von Integrationssoftware) der Versuch unternommen, auf dem hart umkämpften, aber dennoch lukrativen Markt der Integrationslösungen Fuß zu fassen. Die von SeeBeyond entwickelte Integrated Composite Application Network Suite (ICAN Suite) bildet dabei das technologische Rückgrat und fließt in unterschiedliche Produkte, so auch in die Sun Java ESB Suite, ein. Die Sun Java ESB Suite ist eine neue Teilkomponente der Sun Java Composite Application Platform Suite (CAPS Suite), welche als eine umfassende Lösung zur Umsetzung einer SOA angeboten wird. Zu den Fähigkeiten und Zielen gehören dabei, die Wiederverwendung von bereits existierenden Anwendungen zu steigern, die Bereitstellung neuer Services zu unterstützen und die Fähigkeit, Alt- oder geschlossene Anwendungen schnell zu integrieren. Darüber hinaus werden komplementäre Funktionalitäten wie Geschäftsprozessmanagement (BPM) zur Orchestrierung komplexer, personen-, systeme-, services- und unternehmensumfassender Prozesse (basierend auf BPMN[33] und BPEL), nachrichtenbasierte Kommunikation, reichhaltige Transformationsmöglichkeiten und zahlreiche Konnektoren zur Anbindung unterschiedlichster Anwendungen und Systeme (Datenbanken, geschlossene Anwendungen, objekt-, nachrichten- oder webbasierte Protokolle und Technologien) angeboten. Abgerundet wird das Leistungsspektrum der CAPS Suite durch leistungsfähige Werkzeuge aus dem

[33] BPMN – Business Process Modeling Notation

Bereich BAM, dem B2B-Umfeld sowie einer Vielzahl von Servern (Portal-, Applikations-, Web-, Web Proxy- und Verzeichnisserver) (vgl. [Sun06b]; [Sun06c]; [ISI06], S. 1).

Es scheint sich damit um eine wirklich umfassende Infrastruktur zur Realisierung moderner Geschäftsanwendungen zu handeln, die laut Sun in der Lage sein soll, eine SOA zu realisieren. Bei näherer Betrachtung entpuppen sich jedoch die einzelnen Komponenten als eigenständige Produkte, deren integriertes Zusammenspiel nicht immer klar erkennbar ist. Die Rolle, die dabei die angebotene ESB Suite einnimmt, bleibt unklar. Es wird diesbezüglich wiederum auf Fähigkeiten wie BPM, offene Standards, Integrationsfähigkeit, nachrichtenbasierte Kommunikation, Transformation und ein reichhaltiges Angebot von Adaptern verwiesen, dennoch findet kaum mehr als eine Aneinanderreihung von Schlagwörtern statt. Der Aufbau einer ESB-Implementierung oder ob überhaupt eine solche enthalten ist, wird nicht beschrieben. Vielmehr wird auf einzelne Produkte verwiesen, die jeweils unterschiedliche Funktionalitäten eines ESB liefern sollen. Diese sind dazu durchaus in der Lage, jedoch entspricht ihr Aufbau nicht der Architektur und dem Konzept eines ESB, wie es in Kapitel 2.3 beschrieben wurde. Weiterhin bleibt zu konstatieren, dass, vermutlich bedingt durch den erst vor kurzem erfolgten Zukauf von SeeBeyond, es die CAPS Suite und die darin enthaltene ESB Suite nicht schaffen, sich als homogene Software-Suites mit ineinandergreifenden Werkzeugen zu präsentieren. Ein zusätzliches Fragezeichen ist hinter die ESB Suite und ihrem Zusammenhang zum ebenfalls von Sun unterstützten Open ESB zu setzen. Sieht man von diesen Schwächen ab, bleiben aber zahlreiche und überaus leistungsfähige Werkzeuge übrig, die ähnlich wie bei Microsoft ein reichhaltiges Potenzial vorweisen können (vgl. [Sun06b]; [Sun06c]).

Der Open ESB ist das eigentliche Fragezeichen bei Sun Microsystems. Ins Leben gerufen und unterstützt wurde diese ESB-Implementierung als eine freie Referenzimplementierung der von Sun entwickelten JBI Spezifikation. Ziel des Open ESB ist es, über JBI-konforme Service-Container eine freie ESB-Implementierung bereitzustellen. Dazu soll es möglich sein, mehrere JBI-Container miteinander zu verbinden und mittels JMX über einen zentralen Administrationsserver zu verwalten (siehe Abbildung 3.15). Hinzu kommt die Sun Java System Message Queue, die die Grundlage der nachrichtenbasierten Kommunikationsinfrastruktur bildet. Diese erlaubt eine leistungsfähige und zuverlässige Kommunikation und stellt die Verbindung zu den einzelnen JBI-Containern her. Über stellvertretende Bindungen (Proxy Binding) ist es möglich, verteilte Endpunkte anzusprechen und zum Zweck der Lastverteilung oder im Rahmen von Lebenszyklen flexibel auszutauschen. Weiterhin mitgeliefert werden mit dem Open ESB J2EE-, BPEL- sowie auch XSLT-Ausführungskomponenten (in Form von Service Engines) und Binding Components für JMS, SOAP über HTTP und Dateien (über Dateisystem oder FTP). Das derzeitig angebotene Leistungsspektrum des Open ESB ist eher als prototypisch zu bezeichnen und zeigt im Vergleich zu professionellen ESB-Implementierungen erhebliche Defizite (bspw. fehlende Adapter für gängige Anwendungen und Systeme, fehlende grafische Unterstützung oder fehlende Sicherheitsmaßnahmen) (vgl. [Ade06], S. 10 f.; [Sun06a]).

Dennoch kann die Umsetzung von JBI als standardisiertes Containermodell mit Blick auf die Verknüpfung und Föderation unterschiedlicher ESB-Implementierungen als wegweisend erachtet werden. JBI bietet in diesem Kontext die Chance, hinausgehend über die Integration von Anwendungen, Systemen und Protokollen, das Prinzip der Integration und Standardisierung auf eine nächst höhere Abstraktionsebene, nämlich auf die Ebene der Integrationsinfrastruktur und des ESB selbst zu übertragen. Die Forderung würde also nicht mehr nur lauten, dass ein ESB in der Lage sein muss, Standards zu unterstützen, sondern seine Architektur selbst müsste standardkonform (bspw. gemäß JBI) aufgebaut sein. Offen bleibt trotz alledem der Zusammenhang des Open ESB zur ebenfalls von Sun angebotenen ESB Suite und welche Rolle ihm nach dem Zukauf von SeeBeyond zugedacht ist. Im Rahmen eines weiteren Vorantreibens der JBI ist jedoch auch zukünftig mit einer starken Unterstützung und Weiterentwicklung des Open ESB zu rechnen.

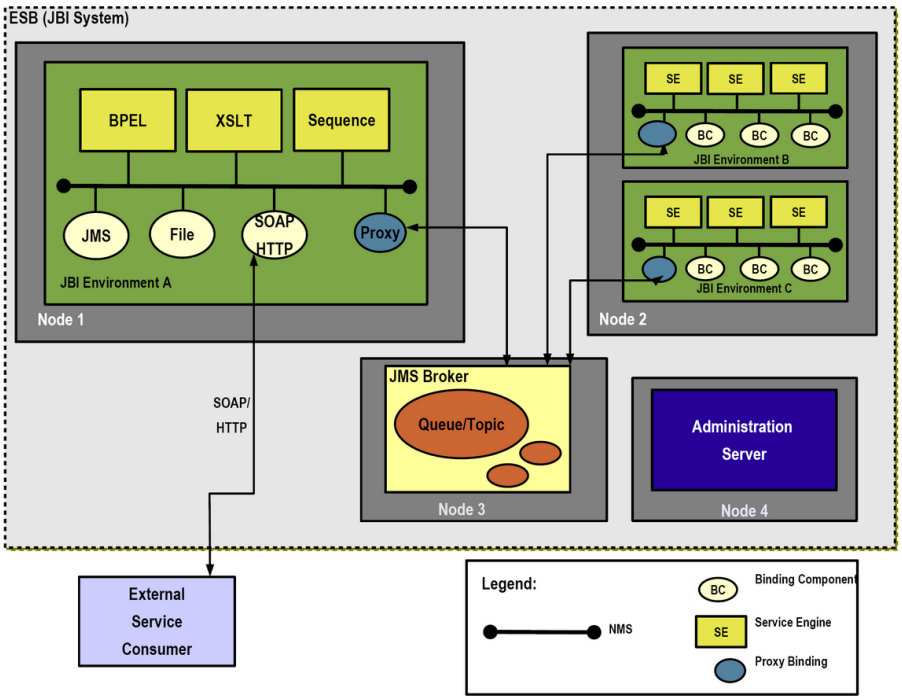

Abbildung 3.15: Aufbau des Open ESB (Quelle: [Ade06], S. 11).

3.11 ESB nach Red Hat (JBoss)

Der von JBoss entwickelte JBoss ESB befindet sich unter den im Rahmen dieser Arbeit genannten ESB-Implementierungen wohl noch im frühesten Entwicklungszustand. Eine stabile Version ist noch nicht verfügbar, einzig eine erste Beta-Version. Trotzdem soll er im Kontext einer zukünftig weiteren, freien ESB-Implementierung kurz betrachtet werden.

Den Kern des JBoss ESB bilden wiederum entsprechende Service-Container, deren Aufgabe u. a. darin besteht, das Management von Servicelebenszyklen zu übernehmen. Explizite Zielstellung ist es beim JBoss ESB, eine Interoperabilität zu anderen ESB-Implementierungen zu ermöglichen. Die zu erreichende Grenze dieser Interoperabilität und Flexibilität besteht darin, dass letztendlich jedwede Komponente des ESB gegen andere austauschbar sein soll. Eine derartig föderierbare Infrastruktur soll bspw. mittels Java Business Integration (JBI) unterstützt werden. Weiterführende Aspekte oder Standards, die eine Umsetzung dieses Ziels aufzeigen, werden jedoch nicht genannt. Weiterhin ist bei JBoss sogar die Rede davon, nicht nur einen Bus sondern mehrere verschiedene je nach Zweck und Priorität zu verwenden. Leider wird auch dieser Aspekt nicht ausführlicher erläutert, weshalb er an dieser Stelle auch nur genannt werden kann (vgl. [JBo05], S. 2 ff.; [JBo06], S. 1 ff.).

Die Aufgaben, die der JBoss ESB zu erfüllen hat, orientieren sich an den Fähigkeiten traditioneller EAI-Produkte (BPM, Bereitstellung von Verbindungen/Adaptern zu unterschiedlichsten Anwendungen und Systemen, Transaktionsverwaltung, Nachrichtentransport oder Metadatenverwaltung) und darüber hinausgehender, ESB-spezifischer Funktionalitäten, wie Orchestrierung (BPEL4WS), Transformation, Management von Servicelebenszyklen, QoS (Transaktionsverwaltung und Fehlerbehandlung) und QoP (Verschlüsselung und Sicherheit). Speziell den Sicherheitsaspekt betreffend werden Access Controll Lists (ACL) zur Zugangs-

kontrolle und Rechtevergabe sowie WS-Standards (WS-Security, WS-Trust) verwendet (vgl. [JBo05], S. 4; [JBo06], S. 2 f.).

Eine abschließende Bewertung des JBoss ESB fällt insbesondere dadurch schwer, da noch keine stabile Version vorliegt. Bei einer Betrachtung des zugrunde liegenden Konzepts hingegen muss eine große Übereinstimmung zu dem in Kapitel 2.3 dargelegten Aufbau und Funktionalitäten eines ESB festgestellt werden. Der mit dem JBoss ESB verfolgte Ansatz erscheint folglich als vielversprechend. Einzige Frage bleibt, in wieweit eine stabile Version des JBoss ESB es schafft, dies umzusetzen.

3.12 Vergleich der ESB-Implementierungen

Bei einem Blick auf die im Rahmen dieses Kapitels betrachteten ESB-Implementierungen fällt auf, dass die professionellen (kommerziellen) ESB-Implementierungen sehr häufig einen überaus reichhaltigen Funktionsumfang bereithalten. Die damit einhergehende hohe Leistungsfähigkeit wird dabei sehr oft in Form von komplementären Komponenten/Produkten realisiert. Diese stammen zum Teil von vergangenen Entwicklungen aus den Bereichen MOM und Integrationsserver (EAI) ab und wurden an heutige ESB-Implementierungen angepasst. Von Vorteil ist, dass sie dadurch als ausgereift und stabil gelten können und in aller Regel äußerst umfangreich sind. Es muss aber ebenso festgehalten werden, dass sich diese Einzelkomponenten/-produkte an manchen Orten als derart selbständig darstellen, dass sich das Zusammenspiel untereinander nicht immer als reibungsfrei erweist. Besonders nachteilig ist dies, wenn davon der eigentliche ESB betroffen ist. Gründe hierfür liegen in einer nahezu eins zu eins Übernahme von Produkten und Funktionen, die gemäß anderer Paradigmen und Konzepte entwickelt wurden und nun zur Unterstützung einer SOA herhalten sollen.

Der Aufbau der unterschiedlichen ESB-Implementierungen stellt sich im Vergleich zueinander trotz individueller Eigenheiten als sehr ähnlich dar. In der Regel folgt er dem Konzept der Service-Container, auch wenn diese nicht immer als solche bezeichnet werden und nicht immer so leichtgewichtig sind wie in Kapitel 2.3 gefordert. Einzige grobe Ausnahme ist Microsoft, deren BizTalk Server sich mehr als ein um Web Services erweiterter Integrationsserver darstellt. Zugute halten kann man Microsoft, dass sie das eigene Konzept nicht als ESB bezeichnen, wenngleich man der Meinung ist, weit mehr als eine ESB-Implementierung anzubieten. Ein weiterer Gesichtspunkt betrifft die zugrundeliegende Nachrichteninfrastruktur. Wenn einige Anbieter (Progress Software oder Fiorano) mit ihren ESB-Implementierungen gleich eine MOM-Plattform mitliefern, so distanzieren sich andere Anbieter von diesem Konzept (allen voran Cape Clear). Es wird dann darauf verwiesen, dass im Sinne größt möglicher Flexibilität der ESB in der Lage sein muss, sämtliche bereits vorhandene Nachrichteninfrastruktur nutzen zu können ohne eine neue einzuführen. Welcher dieser beiden Ansätze der bessere ist, hängt jedoch stark vom jeweiligen Einsatzumfeld ab, zumal die ESB-Implementierungen mit eigener MOM-Infrastruktur gleichfalls in der Lage sind, andere MOM anzusprechen und zu nutzen.

Auf dem Markt der freien ESB-Implementierungen erweist sich Mule trotz einzelner Schwächen und einem teilweise abweichenden Verständnis über den Aufbau und serviceorientierten Charakter eines ESB als vergleichsweise ausgereift. Im Zusammenhang mit vielen anderen freien ESB-Implementierungen und ihrem Aufbau (Mule ist hier die Ausnahme) spielt JBI eine große Rolle. Diese Spezifikation beschreibt ein Service-Container-Modell, das dem Prinzip leichtgewichtiger Service-Container folgt, wenngleich sich der interne Aufbau, von dem im Kapitel 2.3 beschriebenen, im Detail unterscheidet. Zielstellung dieser Spezifikation und gleichzeitig deren Bedeutung ist, den Aufbau einer ESB-Implementierung selbst zu standardisieren. Der Vorteil, der sich aus einer solchen Standardisierung ergibt, ist die Fähigkeit,

einfach und flexibel Integrationslogik austauschen und verknüpfen zu können. Der ESB, der Integrationslogik in Form von Services bereitstellt, und JMX als Standard für die Managementkommunikation zielen bereits in diese Richtung. JBI geht jedoch in seinen Standardisierungsbemühungen noch weiter und fokussiert eine über Anwendungs- und Systemintegration hinausgehende Integration von Integrationsinfrastrukturen. Das Ziel könnte folglich mit Metaintegration umschrieben werden. Einher geht damit eine noch weiterreichende Föderationsmöglichkeit auch von unterschiedlichen ESB-Implementierungen, die in dem Maße gegenwärtig noch nicht möglich ist.

Die nachfolgende Tabelle 3.1 zeigt eine zusammenfassende Übersicht der zuvor beschriebenen ESB-Implementierungen und ihre wesentlichen Eigenschaften.

ESB-Implementierung	**SOA-Plattform**	**MOM-Plattform**	**leichtgewichtige Service-Container**	**grafische Benutzerschnittstelle (GUI)**	**Managementschnittstelle/-en**	**Sicherheitsmechanismen**	**Geschäftsprozessunterstützung**	**umfangreiche Adapter**	**Service-Registry/-Repository**
Sonic ESB	+	+	+	+	+	+	+	+	+
Fiorano ESB	+	+	?	+	+	+	+	+	?
Cape Clear ESB	+	—	+	+	+	+	+	+	+
BEA AquaLogic Service Bus	+	+	—	+	+	+	+	+	+
Oracle ESB	+	+	—	+	+	+	+	+	+
Mule	—	+	+	+	+	+	+	—	—
Microsoft BizTalk Server	+	+	—	+	+	+	+	+	+
Sun Java ESB Suite	+	?	?	?	?	?	?	?	?
Open ESB	—	?	+	?	∗	?	+	∗	?
JBoss ESB	—	?	+	?	∗	?	∗	∗	?

\+ – vorhanden ? – keine Angabe
— – nicht vorhanden ∗ – geplant

Tabelle 3.1: Vergleich der ESB-Implementierungen.

Ein weiterer Aspekt ist die Nutzung von XML als Standard zur Beschreibung von Daten. Vor allem für den Datenaustausch innerhalb eines heterogenen Umfelds ist dieses Format geradezu prädestiniert und gestattet unterschiedlichste Formen der Transformation, Interpretation und Validierung. Darüber hinaus erlauben XML- also textbasierte Datenströme bspw. im Zusammenspiel mit einer ereignisgesteuerten Architektur (EDA), wie sie eine ESB-Implementierung bereitstellen kann, die Entwicklung vollkommen neuer Konzepte. Ein Beispiel hierfür ist das so genannte Event Stream Processing (ESP), eine Technik aus dem Bereich der Business Intelligence (BI). Während bisherige Konzepte sich vor allem auf Data-Warehouse-Systeme stützen, setzt ESP im Kern auf dem ESB, d. h. auf einer im Idealfall bereits vorhandenen Infrastruktur auf. Im Gegensatz zur Verwaltung und Analyse riesiger Datenmengen und -silos werden mit ESP lediglich Datenströme ausgewertet. Diese Auswertung findet zur Laufzeit statt und liefert damit Echtzeitdaten, auf die ebenso in Echtzeit reagiert werden kann. Zum Einsatz kommen dabei musterbasierte Analysen, die Folgeereignisse (Serviceaufrufe, Alarmmeldungen oder E-Mail Nachrichten) auslösen sowie mittels spezieller Werkzeuge visualisiert werden können. Abbildung E.1 im Anhang E zeigt dazu eine Darstellung der einzelnen Schichten einer ESP-Engine und Abbildung E.1 dessen Einbettung in den ESB. Während traditionelle BI-Werkzeuge vor allem zur Unterstützung längerfristiger Entscheidungen dienen, ermöglicht ESP kurzfristig und sogar automatisch auf geschäftsrelevante Ereignisse reagieren zu können. ESP erweist sich somit als eine fundamentale Ergänzung bisheriger BI-Ansätze und zeigt die Bedeutung, die ESB-Implementierungen bereits heute auf diesem Gebiet haben (vgl. [Pal05], S. 1 ff.).

Zu den Punkten, die es neben ESP in Zukunft noch stärker zu unterstützen gilt, gehört allen voran der Einsatz des ESB im Rahmen eines unternehmensübergreifenden Anwendungsumfeldes. Obwohl einige Anbieter durchaus verschiedene B2B-Anbindungen (ebXML, RosettaNET, etc.) bereithalten, sind weiterreichende Konzepte die Ausnahme. Nur vereinzelt können Dienstgütevereinbarungen (SLA) realisiert werden. Serviceorientierte Kostenmodelle sowie die dynamische Wahl von Serviceangeboten liegen noch im Bereich zukünftiger Entwicklungen. Eine Erweiterung heutiger ESB-Implementierungen auf diesen wichtigen Gebieten ist zwingend notwendig, auch um den Anforderungen einer SOA in einem noch stärkerem Maße gerecht zu werden.

3.13 Fazit

Im Rahmen dieses Kapitels konnte gezeigt werden, dass die von der Softwareindustrie angebotenen Implementierungen in ihrer Mehrheit ein mit der Forschung übereinstimmendes Verständnis von einem ESB haben. Viele der kommerziellen ESB-Implementierungen erweisen sich zudem als äußerst mächtig, wenngleich die Fülle an Funktionen teilweise zu Lasten der Handhabbarkeit und Überschaubarkeit geht. Unterschiede, soweit diese vorhanden sind, betreffen im Wesentlichen zwei Fragen. Einerseits in wieweit eine ESB-Implementierung über entsprechende MOM verfügen sollte und andererseits wie serviceorientiert ein ESB aufgebaut sein muss. Welche möglichen Folgen dies auf die Qualitätseigenschaften einer ESB-Implementierung hat und wie das angebotene Leistungsspektrum quantifiziert werden kann, soll mit dem im nächsten Kapitel beschriebenen Qualitätsmodell bewertet werden.

Kapitel 4

Qualitätseigenschaften des ESB

4.1 Herangehensweise

Zielstellung dieses Kapitels ist es, ein Modell zur Bewertung von Qualitätseigenschaften für ESB-Implementierungen aufzustellen. Es wurde bereits in den vorangegangenen Kapiteln auf Eigenschaften, die sich mit einem ESB verbinden, eingegangen. Diese sollen nun in Bezug auf Qualität geordnet und in ein entsprechendes Modell integriert werden. Eine erste Frage, die sich hinsichtlich der Qualitätseigenschaften stellt, ist die nach der Definition des Begriffs Qualität. Nach DIN EN ISO 9000:2000 kann Qualität allgemein angesehen werden als:

> *„Vermögen einer Gesamtheit inhärenter Merkmale eines Produktes, Systems oder Prozesses, zur Erfüllung von Forderungen von Kunden und anderen interessierten Parteien.“*, [KB06], S. 170.

Der Begriff Qualität bezieht sich hier auf Produkte, Systeme und Prozesse. Wie sind hier ESB-Implementierungen einzuordnen? Der Begriff ESB-Implementierung bezeichnet, wie ausführlich in Kapitel 2.3 dargelegt, die Implementierung einer Softwarearchitektur. Demnach geht es bei einer Qualitätsbewertung von ESB-Implementierungen weniger um Prozessqualität als vielmehr um Produkt- und Systemqualität.

Mittels des Factor-Criteria-Metrics-Modells (FCM-Modell) sollen darauf aufbauend Qualitätsmerkmale (Factors) bestimmt werden. In einem weiteren Schritt schließt sich daran die Bestimmung von Teilmerkmalen (Criteria) an. Den so aufgestellten Teilmerkmalen sind dann in einem dritten und letzten Schritt Qualitätsmetriken (Metrics) zuzuordnen, die eine Bewertung von Teil- und Qualitätsmerkmalen ermöglichen.

4.2 Allgemeine Qualitätseigenschaften

Als Quellen derartiger, allgemeiner Qualitätskriterien für Softwaresysteme bzw. -produkte sollen sowohl die Ausführungen von Siedersleben zu diesem Thema als auch die Norm ISO/IEC 9126 herangezogen werden. Zu den Kriterien, die bei beiden Quellen zu finden sind und im Rahmen eines ESB-Qualitätsmodells Anwendung finden sollen, gehören (vgl. [Sie03], S. 92 f.; [Wik06b]):

- **Performanz:** Bezüglich Performanz sind vor allem zwei Kriterien zu nennen. Das sind zum einen die Antwortzeit und zum anderen der Durchsatz. Die Antwortzeit bemisst die Zeitspanne, die nach Absenden einer Nachricht bis zum vollständigen Erhalt einer Antwort vergeht. Der Durchsatz hingegen beschreibt die Datenmenge, die innerhalb einer bestimmten Zeitspanne bearbeitet bzw. übertragen werden kann. Weiterhin kann in der Anzahl der parallel ausführbaren Prozesse noch ein drittes Kriterium gesehen werden.
- **Sicherheit:** Mit Blick auf die Sicherheit eines Systems gilt es vor allem gegen unautorisierten Zugriff geschützt zu sein, d. h. Aspekte wie Vertraulichkeit und Authentizität sicherzustellen. Ein Gesichtspunkt, der in diesem Kontext nicht vernachlässigt werden darf, ist der Schutz vor zufälligen oder beabsichtigten Angriffen.
- **Funktionsumfang/Funktionalität:** Hierbei geht es um die Frage, wie viele der geforderten bzw. zuvor definierten Funktionen tatsächlich erbracht werden.
- **Benutzbarkeit/Anwendbarkeit:** Eines der wohl subjektiveren Kriterien stellt die Benutzbarkeit dar. Dieses Kriterium betrifft vor allem die Benutzerschnittstelle und in welchem Maße der Benutzer bei der Bewältigung seiner Aufgaben unterstützt wird. Darüber hinaus gilt es, die notwendige Einarbeitungszeit bzw. Lernkurve zu berücksichtigen. Mit anderen Worten, welchen Aufwand erfordert der Einsatz der Software von den Benutzern.

Neben diesen bei SIEDERSLEBEN als harte Kriterien bezeichneten Eigenschaften, werden diese unter dem Gesichtspunkt der Flexibilität ergänzt durch weiche, weniger gut quantifizierbare Kriterien. Dazu gehören (vgl. [Sie03], S. 94 f.; [Wik06b]):

- **Testbarkeit und Integrierbarkeit:** Von Bedeutung ist hierbei die Fragestellung, ob und inwieweit ein schrittweiser Test (Komponente für Komponente) bis hin zu einem Test des gesamten Systems möglich ist. Obwohl ein derartiges Vorgehen keine absolute Garantie für Fehlerfreiheit liefert, so besteht wenigstens eine große Wahrscheinlichkeit, grobe Fehler frühzeitig zu entdecken.
- **Wartbarkeit:** Da nahezu jedes System in irgendeiner Weise Fehler aufweist, ist es notwendig, dass diese ohne großen Aufwand gefunden und repariert werden können. Eine wesentliche Bedingung ist weiterhin, dass eine derartige Reparatur nicht zu neuen Fehlern an anderer, unerwarteter Stelle führen darf.
- **Plattformunabhängigkeit/Portierbarkeit:** Plattformunabhängigkeit beschreibt die Eigenschaft von Software, unter verschiedenen Hard- und Softwareumgebungen (Plattformen[1]) lauffähig zu sein. In diesem Zusammenhang ist unter Portierbarkeit zu verstehen, ob und wie ein System im Rahmen einer neuen Umgebung (organisatorische Umgebung, Hardware- oder Softwareumgebung) angepasst und eingesetzt bzw. in diese übertragen werden kann. Portabilität kann demnach als Grad der Plattformunabhängigkeit angesehen werden.
- **Skalierbarkeit:** Skalierbarkeit bezeichnet die Eigenschaft eines Systems, sich ändernden Leistungsanforderungen anpassen zu können. In der Regel betrifft dies steigende Leistungsanforderungen, die nicht dazu führen dürfen, dass der Ressourcenverbrauch überproportional ansteigt oder gar der Ausfall des Systems droht. Vielmehr geht es darum, Ressourcen flexibel zu handhaben und gemäß den Leistungsanforderungen anpassen (erweitern) zu können.

[1] Mit Plattform ist hier die Kombination aus Hardware und hardwarenaher Systemsoftware wie BIOS (Basic Input Output System) oder Betriebssystem gemeint, die zur Ausführung anderer Software dient (vgl. [VAC+05], S. 63).

- **Wiederverwendbarkeit:** Dieses Kriterium gibt Aufschluss darüber, in wieweit es ein System ermöglicht, vorhandene Software (Anwendungen, Komponenten oder einzelne Bausteine) in einem anderen logischen Kontext (anderes Projekt, anderer Geschäftsprozess, etc.) wiederzuverwenden.

Es muss beachtet werden, dass alle genannten Kriterien (harte wie weiche) nur von allgemeiner Natur sind und daher nicht genau auf die speziellen Eigenschaften eines ESB eingehen. Folglich ist es erforderlich, sie um diese speziellen Qualitätseigenschaften zu ergänzen.

4.3 Spezielle Qualitätseigenschaften

An einen ESB bzw. eine ESB-Implementierung werden, wie in Kapitel 2.3 beschrieben, umfangreiche Anforderungen bezüglich der bereitzustellenden Funktionalitäten gestellt. Einige von diesen sind von zentraler Bedeutung und unterscheiden den ESB von anderen Integrationsansätzen. Aus diesem Grund können sie als spezielle Qualitätseigenschaften einer ESB-Implementierung angesehen werden. Dazu gehören:

- ***Unterstützung von Standards,***
- ***intelligentes Routing,***
- ***Transformation.***

Basierend auf diesen speziellen Qualitätseigenschaften sowie auf Basis der zuvor genannten allgemeinen Qualitätseigenschaften soll im folgenden Abschnitt ein ESB-Qualitätsmodell aufgestellt werden.

4.4 ESB-Qualitätsmodell

4.4.1 Performanz

Zur Bestimmung der Performanz sind insbesondere zwei Maße von Bedeutung. Zum einen ist das die Antwortzeit und zum anderen der Durchsatz. Die Antwortzeit (Response Time) bezeichnet hierbei die Zeitspanne, die zwischen dem Absenden einer Nachricht und dem vollständigen Erhalt einer Antwort vergeht. Sie umfasst sowohl Reaktions-, Lauf- als auch Bearbeitungszeiten (vgl. [DAT06]). Langendörfer/Schnor sehen in der Antwortzeit ferner die Aufenthaltszeit eines Auftrags innerhalb eines Systems bis zum Eintreffen des Resultats an einer Ausgabeschnittstelle (vgl. [LS06], S. 250).

Weiterhin spielt beim ESB die Anzahl der parallel ausführbaren Prozesse eine zu berücksichtigende Rolle. Ursache dafür ist der Zusammenhang, der zwischen dieser Größe und der Antwortzeit besteht. So ist es leicht verständlich, dass mit zunehmender Zahl paralleler Prozesse ein Rückgang der Antwortzeit zu erwarten ist, jedoch damit kein Rückgang der Performanz verbunden sein muss. Deshalb kann bei einer Bewertung der Antwortzeit die Anzahl der parallelen Prozesse nicht außer Acht gelassen werden.

Im Gegensatz dazu ist unter Durchsatz (Throughput) das Verhältnis zwischen versendeter bzw. verarbeiteter Datenmenge je Zeiteinheit zu verstehen. Er kann außerdem als Leistung

bzw. Geschwindigkeit eines Systems angesehen werden, mit der eine bestimmte Arbeitslast bewältigt wird (vgl. [CDK02], S. 65; [LS06], S. 250). Es ergibt sich demnach:

$$Durchsatz = \frac{Datenmenge}{Zeiteinheit}$$

Diese allgemeine Form trägt jedoch nur bedingt den Charakteristiken einer ESB-Implementierung Rechnung. Hierzu gilt es, den Begriff Datenmenge genauer zu definieren. Es wurde bereits an verschiedenen Stellen darauf hingewiesen, dass sich die Kommunikation eines ESB im Wesentlichen auf Nachrichten stützt. Diese, bspw. mittels SOAP gekapselten Nachrichten, stellen demnach die zu verarbeitenden Daten dar. Folglich ist es weniger interessant welche Datenmenge, als vielmehr welche Nachrichtenmenge von einer ESB-Implementierung verarbeitet werden kann. Es wird deshalb an Stelle des allgemeinen Durchsatzes (auch als Datendurchsatz bezeichnet) der Nachrichtendurchsatz verwendet. Über die Nachrichtenmenge, also die Anzahl der verarbeiteten Nachrichten, erfolgt außerdem eine Entkopplung von der tatsächlichen Datenmenge. Der Nachrichtendurchsatz lässt sich somit im Gegensatz zum Datendurchsatz beschreiben als:

$$Nachrichtendurchsatz = \frac{Nachrichtenmenge}{Zeiteinheit}$$

Um diese drei Größen für ESB-Implementierungen anwenden zu können, sind jedoch noch weitere Einschränkungen zu berücksichtigen. Gemeinsam sind sie in starkem Maße von äußeren (exogenen) Faktoren wie Netzwerk-/Rechenkapazität, verwendete Plattformen, gewählte Deployment-Topologie und nicht zuletzt von der jeweiligen Belastung (in Form zu verarbeitender Nachrichten/Daten) abhängig. Diese Faktoren, die von der eigentlichen ESB-Implementierung unabhängig sind, werden im Folgenden als Szenario bezeichnet. Ein Szenario dient dazu, basierend auf annähernd gleichen äußeren Bedingungen, Antwortzeiten, Anzahl paralleler Prozesse und Durchsatzraten vergleichbar zu machen. Aus diesem Grund werden für ein gemeinsames Szenario die Antwortzeit als AZ, die Anzahl der parallelen Prozesse als PP und der Nachrichtendurchsatz als ND definiert.

In Folge des dominierenden Szenarioeinflusses wird weiterhin auf eine qualifizierende Bewertung gemessener Antwortzeiten, Anzahl paralleler Prozesse und Durchsatzraten (bspw. mittels eines Notensystems) verzichtet, zumal eine derartige Bewertung nur relativ zu den jeweiligen ESB-Implementierungen und dem gewählten Szenario zu sehen wäre.

4.4.2 Sicherheit

Ein wichtiger Gesichtspunkt im Kontext des ESB betrifft die Sicherstellung und Gewährleistung hoher Sicherheitsanforderungen. Diese begründen sich in der Nutzung offener Netze (Internet), der über Unternehmensgrenzen hinaus reichenden Kommunikation und dem geschäftsrelevanten Inhalt. Folgende wesentliche Sicherheitsaspekte sollen Berücksichtigung finden (vgl. [Bis03], S. 4 ff.; [NL04], S. 124 f.):

- **Vertraulichkeit:** Unter Vertraulichkeit ist der Schutz von Informationen und Ressourcen gegenüber unberechtigten Dritten zu verstehen. Teilaspekte der Vertraulichkeit betreffen Authentifizierung, Autorisation und Verschlüsselung.
- **Integrität:** Integrität bezieht sich auf die Glaub- bzw. Vertrauenswürdigkeit von Daten oder Ressourcen. Diese können als integer gelten, sofern sichergestellt ist, dass

sowohl ihr Inhalt als auch die Identität des Verfassers/Absenders nicht geändert wurde (mutwillig oder zufällig). Darunter fällt wiederum der Aspekt der Authentifizierung.

- **Verfügbarkeit[2]:** Verfügbarkeit ist die Fähigkeit eines Systems, seine Informationen oder Ressourcen in gewünschter Weise nutzen zu können. Dieser Aspekt ist aus dem Grunde wichtig, da nicht verfügbare Informationen oder Ressourcen sich schlimmsten Falls wie gar keine Informationen oder Ressourcen auswirken und somit zu Sicherheitsrisiken führen können. Diesbezüglich sind vor allem der Ausfall eines Systems und die damit verbundenen Konsequenzen für die (potenziellen) Nutzer von Bedeutung.

Zur Bewertung der unterstützten Sicherheitsaspekte wird in diesem Zusammenhang *SI* als ordinalskalierte Metrik definiert. Der Wertebereich von *SI* liegt zwischen eins und fünf. Die einzelnen Ausprägungen von *SI* ergeben sich in folgender Weise.

Interpretation von *SI*:

- $\boldsymbol{SI = 1}$ („sehr gut“): Zu allen genannten Sicherheitsaspekten und ggf. noch zu weiteren (etwa Nichtabstreitbarkeit der Urheberschaft (Non-Repudiation of Origin) oder Single-Sign-On) gibt es Mechanismen und Technologien, mit denen diese umgesetzt werden können.
- $\boldsymbol{SI = 2}$ („gut“): Bis auf Verfügbarkeit werden die beiden anderen Aspekte Vertraulichkeit und Integrität sowie auch darin enthaltene Gesichtspunkte wie Authentifizierung, Autorisation und Verschlüsselung durch entsprechende Maßnahmen abgedeckt.
- $\boldsymbol{SI = 3}$ („befriedigend“): Sämtliche Teilaspekte der Vertraulichkeit (Authentifizierung, Autorisation und Verschlüsselung) werden unterstützt (ggf. sogar durch mehrere Mechanismen bzw. Technologien).
- $\boldsymbol{SI = 4}$ („ausreichend“): Es werden mindestens auf dem Gebiet der Verschlüsselung entsprechende Mechanismen und Technologien bereitgestellt.
- $\boldsymbol{SI = 5}$ („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

4.4.3 Funktionsumfang/Funktionalität

In Anlehnung an die bereits im Kapitel 2.3 von einem ESB bzw. einer ESB-Implementierung geforderten Kernfunktionalitäten soll nun eine entsprechende Metrik definiert werden, die eine exakte Quantifizierung erlauben soll. Zu den geforderten Funktionen gehören:

- Unterstützung und Abbildung von (Geschäfts-)Prozessen bzw. Prozessketten,
- Service Orchestrierung,
- Service Choreographie,
- Konfiguration, Monitoring und Management von Services,
- Management von Servicelebenszyklen,

[2] Der Aspekt der Verfügbarkeit wird an dieser Stelle nur allgemein und aus dem Blickwinkel der Sicherheit heraus betrachtet.

- Dienstgüte (QoS),
- Transaktionsverwaltung,
- flexibles Deployment,
- Föderation und Erweiterbarkeit.

Hierbei muss beachtet werden, dass andere wesentliche Funktionalitäten einer ESB-Implementierung (Standardunterstützung, Transformation, Routing, Sicherheit und Skalierbarkeit) ausgeklammert wurden, da diese in gesonderter Weise betrachtet werden. Aus der Anzahl der unterstützten Funktionalitäten ergibt sich die verhältnisskalierte Metrik FU mit einem Wertebereich von null bis neun. Neben dieser quantifizierenden Bewertung soll mit FU' eine qualifizierende Bewertung in Form einer ordinalskalierten Metrik ermöglicht werden. Der Wertebereich von FU' liegt zwischen eins und fünf, wobei die einzelnen Ausprägungen von FU' in folgendem Zusammenhang zu FU zu sehen sind.

Interpretation von FU':

- $\boldsymbol{FU' = 1}$ („sehr gut“): Es werden alle neun geforderten Funktionalitäten bereitgestellt ($FU = 9$).
- $\boldsymbol{FU' = 2}$ („gut“): Es werden mindestens sieben der genannten Funktionalitäten bereitgestellt ($FU \geq 7$). Weiterhin müssen zu diesen wenigstens Geschäftsprozessunterstützung, Service Orchestrierung/Choreographie, Management von Servicelebenszyklen, Transaktionsverwaltung und flexibles Deployment gehören.
- $\boldsymbol{FU' = 3}$ („befriedigend“): Es werden mindestens fünf der genannten Funktionalitäten bereitgestellt ($FU \geq 5$). Weiterhin müssen zu diesen wenigstens Transaktionsverwaltung und flexibles Deployment gehören.
- $\boldsymbol{FU' = 4}$ („ausreichend“): Es werden mindestens drei der genannten Funktionalitäten bereitgestellt ($FU \geq 3$).
- $\boldsymbol{FU' = 5}$ („ungenügend“): Es werden weniger als drei der genannten Funktionalitäten bereitgestellt ($FU < 3$).

4.4.4 Benutzbarkeit

Obwohl die Bewertung der Benutzbarkeit (Usability) eigentlich nur in Form eigenständiger Modelle und entsprechender Nutzbarkeitstests (Usability-Tests) möglich ist, soll dennoch im Rahmen dieses Modells eine grobe Bewertung vorgenommen werden. Grundlage für diese Bewertung bildet die Annahme, dass mit steigendem Umfang bereitgestellter grafischer und assistentgestützter Werkzeuge und Schnittstellen (GUI) sich gleichfalls die Benutzbarkeit erhöht (d. h. eine direkte Proportionalität besteht). Begründet werden kann diese Annahme dadurch, dass bei grafischer Aufbereitung von Informationen im Gegensatz zu einer rein textbasierten Darstellung eine anschaulichere und leichtere Verarbeitung möglich ist (weshalb bspw. auch bei der Darstellung statistischer Daten grafische Darstellungsformen weit verbreitet sind). Als ähnlich unterstützend können Assistenten (Wizards) angesehen werden, deren Zweck darin besteht, komplexere Aufgaben automatisch oder halbautomatisch auszuführen und die Komplexität dieser Aufgaben vor dem Nutzer zu verbergen. Diesbezüglich sind folgende Aufgaben als relevant einzuschätzen:

- Design und Spezifikation von (Service-)Schnittstellen,

- Modellierung von Geschäftsprozessen,
- Orchestrierung und Choreographie von Services,
- Transformation,
- Routing,
- Deployment,
- Management und Überwachung von Services und Servicelebenszyklen,
- Suchen und Finden von Diensten,
- Konfiguration und Management des ESB.

Die Anzahl der durch grafische Komponenten bzw. Assistenten unterstützten Aufgabenbereiche findet sich in der verhältnisskalierten Metrik *BE* wieder (Wertebereich zwischen null und neun). Weiterhin wird mit *BE'* eine ordinalskalierte Metrik definiert, die einer qualifizierenden Bewertung der Benutzbarkeit dient. Der Wertebereich von *BE'* liegt zwischen eins und fünf. Zwischen *BE* und *BE'* besteht der folgende Zusammenhang.

Interpretation von *BE'*:

- ***BE'* = 1** („sehr gut"): Es werden sämtliche genannten Aufgabenbereiche durch grafische oder assistentgestützte Werkzeuge unterstützt ($BE = 9$).
- ***BE'* = 2** („gut"): Es werden mindestens sieben der genannten Aufgabenbereiche durch grafische oder assistentgestützte Werkzeuge unterstützt ($BE \geq 7$). Zu diesen gehören wenigstens eine Unterstützung für Design und Spezifikation von (Service-)Schnittstellen, für Modellierung von Geschäftsprozessen, für Deployment sowie auch für Konfiguration und Management des ESB.
- ***BE'* = 3** („befriedigend"): Es werden mindestens fünf der genannten Aufgabenbereiche durch grafische oder assistentgestützte Werkzeuge unterstützt ($BE \geq 5$). Zu diesen gehören wenigstens eine Unterstützung für Design und Spezifikation von (Service-)Schnittstellen sowie auch für Konfiguration und Management des ESB.
- ***BE'* = 4** („ausreichend"): Es werden mindestens drei der genannten Aufgabenbereiche durch grafische oder assistentgestützte Werkzeuge unterstützt ($BE \geq 3$).
- ***BE'* = 5** („ungenügend"): Es werden zu weniger als drei der genannten Aufgabenbereiche grafische oder assistentgestützte Werkzeuge bereitgestellt ($BE < 3$).

Ein zu nennender Kritikpunkt in Bezug auf die zugrunde liegende Annahme betrifft die Grenze der Anwendbarkeit. Wenngleich sich eine grafische und werkzeugbasierte Unterstützung bei einem vertretbaren Funktionsumfang durchaus positiv auswirken wird, so ist in gleicher Weise anzunehmen, dass mit immer weiter steigendem Funktionsumfang und damit steigender Komplexität wiederum negative Auswirkungen auf die Benutzbarkeit festgestellt werden können. Eine derartige Obergrenze wird im Rahmen dieses Modells nicht berücksichtigt, sollte aber im Rahmen der Interpretation bedacht werden[3].

[3] Eine ESB-Implementierung, die dieser Grenze sehr nahe kommt bzw. diese zum Teil schon überschreitet, ist der Oracle ESB (siehe Kapitel 3.7)

4.4.5 Testbarkeit, Integrierbarkeit

Hier geht es vor allem um das Kriterium der Testunterstützung, also in wieweit eine ESB-Implementierung in der Lage ist, den schrittweisen Test und die Integration einzelner Services zu ermöglichen. Im Mittelpunkt stehen dabei vor allem dynamische und integrationsbezogene Testverfahren. Statische Testverfahren spielen eine untergeordnete Rolle, da bis auf Schnittstellenbeschreibungen und Programmcode zur Implementierung der Services, die eigentliche Geschäftslogik in der Regel nicht innerhalb der Service-Container residiert bzw. deren Quellcode verborgen bleibt. Folgende Maßnahmen sollte eine ESB-Implementierung berücksichtigen:

- statische Tests[4],
- Black-Box Tests von Services,
- White-Box Tests von Services[4],
- Integrationstests über mehrere Services hinweg,
- Testdatengeneratoren,
- Debugging von Services,
- Ablaufverfolgung (Tracing) von Services,
- Benchmarking mittels Stress-/Lasttests,
- Laufzeitanalysen (Profiling).

Die Anzahl der unterstützten bzw. bereitgestellten Testmaßnahmen wird durch die verhältnisskalierte Metrik *TI* beschrieben. Dieser ist wiederum eine weitere ordinalskalierte Metrik in Form von *TI'* zugeordnet. Der Wertebereich von *TI* liegt zwischen null und neun, der von *TI'* zwischen eins und fünf. *TI'* ist in folgender Weise von *TI* abhängig.

Interpretation von *TI'*:

- ***TI'* = 1** („sehr gut“): Es werden mindestens sieben der genannten Testmaßnahmen angeboten ($TI \geq 7$). Zu diesen gehören wenigstens statische, Black-Box, White-Box und Integrationstests sowie das Debugging und die Ablaufverfolgung von Services.
- ***TI'* = 2** („gut“): Es werden mindestens fünf der genannten Testmaßnahmen angeboten ($TI \geq 5$). Zu diesen gehören wenigstens statische, Black-Box und Integrationstests.
- ***TI'* = 3** („befriedigend“): Es werden mindestens vier der genannten Testmaßnahmen angeboten ($TI \geq 4$). Zu diesen gehören wenigstens statische und Black-Box Tests.
- ***TI'* = 4** („ausreichend“): Es werden mindestens drei der genannten Testmaßnahmen angeboten ($TI \geq 3$).
- ***TI'* = 5** („ungenügend“): Es werden weniger als drei der genannten Testmaßnahmen angeboten ($TI < 3$).

[4] Dies betrifft den innerhalb der Service-Container residierenden Programmcode.

4.4.6 Wartbarkeit

Die Wartbarkeit einer ESB-Implementierung ist gerade für die Umsetzung einer SOA von zentraler Bedeutung. In besonderem Maße gilt dies mit Blick auf das Ziel einer SOA, flexibel und schneller auf neue Anforderungen reagieren zu können, als dies andere Konzepte ermöglichen. Betrachtet werden sollen hierbei weniger Aspekte wie Analysierbarkeit, Änderbarkeit, Stabilität oder Testbarkeit (wie sie in der ISO/IEC-9126 genannt werden) als vielmehr die Unterstützung von verschiedenen Wartungsaktivitäten. Zu diesen können die Folgenden gerechnet werden (vgl. [Dum03], S. 104 ff.):

- **Erweiterung:** Eine ESB-Implementierung muss in der Lage sein, auf einfache Art und Weise um Service-Container als auch um weitere Services (Geschäfts- und Integrationslogik) erweitert werden zu können, ohne dass dies an anderen Stellen zu Fehlern oder Inkonsistenzen führt. Die Ursache für Erweiterungen bilden in der Regel neue funktionale Anforderungen.
- **Anpassung:** Neben der Erweiterung muss eine ESB-Implementierung ebenso flexibel an neue systembezogene Anforderungen angepasst werden können.
- **Korrektur:** Aufgrund der Komplexität von Geschäftsprozessen und damit verbundener Abläufe können trotz umfangreicher Tests Fehler nie ganz ausgeschlossen werden. Im Rahmen der Korrektur muss eine ESB-Implementierung daher in der Lage sein, Fehler frühzeitig zu erkennen und zu melden sowie geeignete Maßnahmen zur Korrektur bereitzustellen.
- **Verbesserung:** Die Verbesserung betrifft vor allem die qualitativen Anforderungen, die an eine ESB-Implementierung und deren Services gestellt werden. Gerade mit Blick auf Dienstgüten (QoS) gilt es, schnell neuen Anforderungen gerecht zu werden. Hier sollte eine ESB-Implementierung über ein entsprechendes Portfolio von Maßnahmen verfügen.
- **Vorbeugung:** Viele der im Sinne einer Vorbeugung relevanten Maßnahmen sind bereits in den genannten Kriterien enthalten. Noch nicht enthalten sind Maßnahmen, die aufzeigen, an welchen Stellen es sinnvoll ist über Erweiterungen, Anpassungen, Korrekturen oder Verbesserungen nachzudenken, also vorbeugend Wartungsaktivitäten durchzuführen.

Die Bewertung der Wartbarkeit erfolgt über die ordinalskalierte Metrik *WA*, deren Wertebereich zwischen eins und fünf liegt. Die genannten Kriterien werden dabei in folgender Weise berücksichtigt.

Interpretation von *WA*:

- $\boldsymbol{WA = 1}$ („sehr gut“): Es werden alle genannten Wartungsaktivitäten durch entsprechende Maßnahmen oder Werkzeuge unterstützt.
- $\boldsymbol{WA = 2}$ („gut“): Es werden mindestens zu Erweiterung, Anpassung, Korrektur und Verbesserung Maßnahmen oder Werkzeuge angeboten.
- $\boldsymbol{WA = 3}$ („befriedigend“): Es werden mindestens zu Erweiterung, Korrektur und Verbesserung Maßnahmen oder Werkzeuge angeboten.
- $\boldsymbol{WA = 4}$ („ausreichend“): Es werden mindestens zu Erweiterung und Korrektur Maßnahmen oder Werkzeuge angeboten.

- ***WA* = 5** („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

4.4.7 Plattformunabhängigkeit/Portierbarkeit

Trotz der Tatsache, dass ESB-Implementierungen Anwendungen und Systeme für unterschiedliche Umgebungen nutzbar (portierbar) machen sollen, betrifft der Aspekt der Portierbarkeit aus verschiedenen Gründen auch ESB-Implementierungen selbst. Zum einen handelt es sich bei ESB-Implementierungen genauso um Softwareprodukte, weshalb dafür geltende Qualitätsmerkmale angewendet werden können und zum anderen liegt die Bedeutung ebenso in der Natur der mit einem ESB verfolgten Ziele. So muss sich der ESB in einem zumeist äußerst heterogenen Umfeld behaupten und ist daher darauf angewiesen, möglichst plattformunabhängig eingesetzt werden zu können.

Kriterien für Plattformunabhängigkeit und Portierbarkeit sind zum einen die Anzahl der unterstützten Plattformen und zum anderen der Arbeitsaufwand, der benötigt wird, um eine Portierung[5] durchzuführen. Problematisch ist in diesem Kontext die Messung des benötigten Portierungsaufwands, da dieser in großem Maße sowohl von vorhandenen personellen und Hardwareressourcen als auch von den jeweiligen Ausgangs- und Zielplattformen abhängig ist. Leichter und objektiver messbar ist die Anzahl der unterstützten Plattformen. Wenngleich auf dieser Basis nur bedingt Rückschlüsse auf eine allgemeine Portierbarkeit möglich sind, soll aufgrund der geschilderten Problematik und der damit verbundenen Ungenauigkeiten von einer Bestimmung des benötigten Portierungsaufwands abgesehen werden. Allgemein betrachtet kann davon ausgegangen werden, dass mit der Anzahl der unterstützten Plattformen der Grad der Plattformunabhängigkeit und damit gleichfalls die Portierbarkeit steigt.

Als Metrik für die Plattformunabhängigkeit soll die Anzahl der unterstützten Plattformen, genauer die Anzahl der unterstützten Betriebssysteme verwendet werden[6]. Es handelt sich um eine verhältnisskalierte Metrik mit einem Wertebereich von null bis 23, die im Weiteren als *PU* bezeichnet wird. Grundlage bilden die in Tabelle 4.1 aufgeführten und als wesentlich anzusehenden Betriebssysteme.

Betriebssystemkategorien	**Betriebssysteme**
Microsoft:	Windows 2000 Windows 2000 Server Windows XP Windows Server 2003
Linux:	Debian Red Hat Enterprise Linux SUSE Linux Enterprise Slackware
Apple:	Mac OS X Mac OS X Server

[5] Portierung bezeichnet die Übertragung von Software von einer auf eine andere Plattform.
[6] Auf eine Betrachtung unterschiedlicher Hardwarearchitekturen (x86, x86-64, IA32, IA64, PPC, SPARC, etc.) wird verzichtet, da diese in aller Regel in einem direkten Zusammenhang zur verwendeten Systemsoftware stehen.

Unix:	AIX FreeBSD HP-UX NetBSD SCO OpenServer SCO UnixWare Solaris/SunOS OpenBSD IRIX
Mainframe:	BS2000/OSD i5/OS (ehemals OS/400) Unisys OS2200 VMS/OpenVMS z/OS (ehemals OS/390)

Tabelle 4.1: Betriebssysteme (vgl. [Ope06]; [IDG06b]; [Gui06]).

Eine wichtige Komponente einer ESB-Implementierung stellen grafische Werkzeuge dar. Zwar wird diese Funktionalität nicht direkt gefordert (siehe Kapitel 2.3), spielt jedoch zumindest indirekt mit Blick auf die Nutzbarkeit (Usability) eine gewichtige Rolle. Von den meisten ESB-Anbietern werden separate Komponenten bzw. Produkte als GUI zum ESB mitgeliefert. Wenngleich sie in der Regel losgelöst von der eigentlichen ESB-Implementierung betrieben werden und demzufolge oft andere Betriebssystemanforderungen stellen, können sie dem ESB trotzdem im weiteren Sinn zugeordnet werden. Aus diesem Grund soll mit *PUG* eine weitere verhältnisskalierte Metrik für die Anzahl der unterstützten Plattformen seitens der grafischen Komponenten definiert werden.

Obwohl in *PU* und *PUG* wichtigste Betriebssysteme einfließen, bleiben andere unberücksichtigt. Außerdem werden alle Betriebssysteme ungewichtet und unabhängig von den jeweiligen Betriebssystemfamilien betrachtet. Aus diesem Grund und zum Zweck einer qualitativen Bewertung der Plattformunabhängigkeit werden zwei weitere ordinalskalierte Metriken *PU'* und *PUG'* mit einem Wertebereich von eins bis fünf definiert. Deren jeweilige Ausprägungen können wie folgt interpretiert werden.

Interpretation von *PU'*:

- ***PU'* = 1** („sehr gut"): Es werden jeweils mindestens drei den Betriebssystemkategorien Microsoft, Unix und Linux zuordenbare Betriebssysteme unterstützt[7]. Zusätzlich werden mindestens zwei Betriebssysteme der Kategorie Mainframe und ein Betriebssystem der Kategorie Apple unterstützt.
- ***PU'* = 2** („gut"): Es werden jeweils mindestens zwei den Betriebssystemkategorien Microsoft, Unix und Linux zuordenbare Betriebssysteme unterstützt[7]. Weiterhin müssen insgesamt mindestens acht der genannten Betriebssysteme unterstützt werden ($PU \geq 8$).
- ***PU'* = 3** („befriedigend"): Es wird jeweils mindestens ein den Betriebssystemkategorien Microsoft, Unix und Linux zuordenbares Betriebssystem unterstützt[7]. Weiterhin müssen insgesamt mindestens fünf der genannten Betriebssysteme unterstützt werden ($PU \geq 5$).

[7] Dies können im Einzelfall auch Betriebssysteme sein, die nicht in Tabelle 4.1 aufgeführt sind.

- ***PU'* = 4** („ausreichend“): Es werden mindestens drei der genannten Betriebssysteme unterstützt ($PU \geq 3$), wobei diese aus mindestens zwei verschiedenen Betriebssystemkategorien stammen müssen.
- ***PU'* = 5** („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

Interpretation von *PUG'*:

- ***PUG'* = 1** („sehr gut“): Es werden jeweils mindestens zwei den Betriebssystemkategorien Microsoft, Unix, Linux und Apple zuordenbare Betriebssysteme von grafischen Komponenten unterstützt[7].
- ***PUG'* = 2** („gut“): Es wird jeweils mindestens ein den Betriebssystemkategorien Microsoft, Unix, Linux und Apple zuordenbares Betriebssystem von grafischen Komponenten unterstützt[7]. Weiterhin müssen insgesamt mindestens fünf der genannten Betriebssysteme unterstützt werden ($PUG \geq 5$).
- ***PUG'* = 3** („befriedigend“): Es werden mindestens fünf der genannten Betriebssysteme von grafischen Komponenten unterstützt ($PUG \geq 5$).
- ***PUG'* = 4** („ausreichend“): Es werden mindestens drei der genannten Betriebssysteme von grafischen Komponenten unterstützt ($PUG \geq 3$).
- ***PUG'* = 5** („ungenügend“): Es werden weniger als drei der genannten Betriebssysteme von grafischen Komponenten unterstützt ($PUG < 3$).

Im Zusammenhang mit der Bewertung der Plattformunabhängigkeit und Portierbarkeit ist es notwendig, eine weitere ESB-spezifische Einschränkung vorzunehmen. Wie bereits in Kapitel 3 beschrieben, sind viele ESB-Implementierungen aus einzelnen Komponenten aufgebaut bzw. werden durch komplementäre Produkte ergänzt. Soweit diese Komponenten oder Produkte der jeweiligen ESB-Implementierung direkt zuzuordnen sind oder zur Erbringung des in Kapitel 2.3 geforderten Funktionsumfangs dienen, müssen auch ihre betriebssystemspezifischen Anforderungen berücksichtigt werden. Aus diesen Gründen wird festgelegt, dass bei Abweichungen der unterstützten Betriebssysteme seitens zuvor genannter ESB-Bestandteile lediglich die gemeinsame Schnittmenge als Bewertungsgrundlage herangezogen wird. Davon nicht betroffen sind Komponenten oder Produkte zur grafischen Unterstützung, die wie zuvor beschrieben, in gesonderter Form betrachtet werden.

4.4.8 Skalierbarkeit

Für eine ESB-Implementierung ist die Skalierbarkeit von zentraler Bedeutung, zumal sich der ESB gerade durch diese Eigenschaft von anderen Integrationslösungen abhebt. Es wird zwischen zwei Arten unterschieden, zum einen die vertikale und zum anderen die horizontale Skalierbarkeit. Unter vertikaler Skalierbarkeit[8] wird die Erweiterung lokaler Ressourcen verstanden, also eine Leistungssteigerung innerhalb eines Knotens[9]. Im Gegensatz dazu ist

[8] Im Kontext des ESB wird unter vertikaler Skalierbarkeit weniger die Fähigkeit zur Steigerung von Hardwareressourcen als vielmehr zur Steigerung von Softwareressourcen innerhalb eines Knotens/Service-Containers verstanden. Das wäre bspw. das Hinzufügen weiterer Services und damit verbundener Geschäfts- bzw. Integrationslogik.

[9] Ein (Netzwerk-)Knoten (Node) bezeichnet eine physische oder logische Einheit, die über ein gewisses Quantum an Rechenkapazität oder anderen Ressourcen verfügt. Im Falle des ESB sind dies typischerweise die Service-Container.

mit horizontaler Skalierbarkeit die Erweiterung der Ressourcen mittels weiterer Knoten gemeint. Eine diesbezüglich leistungsfähige ESB-Implementierung sollte demnach beide Arten der Skalierbarkeit gemäß der folgenden Kriterien unterstützen:

- vertikale Skalierung zur Steigerung der Anwendungsfunktionalität,
- vertikale Skalierung zur Steigerung der Integrationsfunktionalität,
- horizontale Skalierung zur Steigerung der Ausfallsicherheit,
- horizontale Skalierung zur Steigerung der Performanz.

Als Metrik zur Bewertung dieser Skalierbarkeitskriterien wird *SK* definiert. *SK* ist ordinalskaliert mit einem Wertebereich von eins bis fünf.

Interpretation von *SK*:

- ***SK* = 1** („sehr gut"): Neben einer vollständigen vertikalen Skalierung ist gleichfalls eine vollständige horizontale Skalierung möglich. Das bedeutet im Falle der horizontalen Skalierung, dass sowohl die Steigerung der Ausfallsicherheit gemäß Ausweichmechanismen (Failover) als auch die Steigerung der Performanz im Sinne einer Lastverteilung (Load Balancing) möglich ist. Lastverteilung heißt dabei, dass gleiche Funktionalität redundant über mehrere Knoten (typischerweise Service-Container) verteilt ist und je nach Belastung eine Auswahl des entsprechenden Knotens erfolgt.
- ***SK* = 2** („gut"): Neben einer vollständigen vertikalen Skalierung ist eine horizontale Skalierung zur Steigerung der Ausfallsicherheit gemäß Ausweichmechanismen (Failover) möglich, d. h. beim Ausfall eines (primären) Services oder Service-Containers kann mindestens ein weiterer (sekundärer) Service bzw. Service-Container dessen Aufgabe übernehmen.
- ***SK* = 3** („befriedigend"): Es ist eine vollständige vertikale Skalierung möglich, in Form der Erweiterung einzelner Service-Container um zusätzliche Services (Anwendungsfunktionalität) und auch in Form zusätzlicher Integrationsfunktionalität (bspw. zur Realisierung von erforderlichen Dienstgüten).
- ***SK* = 4** („ausreichend"): Es ist mindestens eine vertikale Skalierung in Form der Erweiterung einzelner Service-Container um zusätzliche Services (Anwendungsfunktionalität) möglich.
- ***SK* = 5** („ungenügend"): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend") erforderlich sind.

4.4.9 Wiederverwendbarkeit

Wie bereits in den Kapiteln 1 und 2 betont wurde, stellt die Wiederverwendung von Software ein schon seit langem angestrebtes Ziel dar. SOA und speziell der ESB sollen helfen, diesem Ziel noch stärker Rechnung zu tragen, als das bereits durch bisher bestehende Konzepte der Fall ist. Neben vielen verschiedenen Wiederverwendungsarten (siehe [Dum03], S. 356 ff.) sind im Kontext des ESB die folgenden von größerer Bedeutung (vgl. [Dum03], S.357 f.):

- **Black-Box-Wiederverwendung:** Einzelne Komponenten/Services werden als abgeschlossene Einheiten betrachtet und können als solche wiederverwendet werden.

- **Explizite Wiederverwendung:** Einzelne Komponenten/Services können exakt identifiziert und somit wiederverwendet werden.
- **Externe Wiederverwendung:** Einzelne Komponenten/Services können auch außerhalb des Bereichs der realisierten Software genutzt werden.
- **Implizite Wiederverwendung:** Einzelne Komponenten/Services werden indirekt wiederverwendet, ohne dass dies nach außen direkt erkennbar ist.
- **Offensive Wiederverwendung:** Die Entwicklung von Software und damit verbundene Softwareanforderungen orientieren sich an bereits vorhandenen Komponenten/Services.

Die Aufgabe einer ESB-Implementierung besteht nun darin, diese verschiedenen Wiederverwendungsarten bestmöglich zu unterstützen. Wie gut sich diese Unterstützung im Einzelfall darstellt, soll mit der ordinalskalierten Metrik WV gemessen werden. Der Wertebereich von WV liegt zwischen eins und fünf, wobei die jeweiligen Merkmalsausprägungen folgende Bedeutung haben.

Interpretation von WV:

- $\boldsymbol{WV = 1}$ („sehr gut“): Neben Black-Box-, expliziter, impliziter und offensiver Wiederverwendung ist durch Unterstützung globaler Registries zusätzlich eine externe Wiederverwendung möglich. Weiterhin wird im Sinne einer nachhaltigen Wiederverwendung die Versionierung von Services unterstützt.
- $\boldsymbol{WV = 2}$ („gut“): Sowohl Black-Box-, explizite als auch implizite Wiederverwendung von Services sind möglich. Weiterhin wird über Registries zum Veröffentlichen, Beschreiben, Suchen und Finden von Services eine offensive Wiederverwendung unterstützt.
- $\boldsymbol{WV = 3}$ („befriedigend“): Neben Black-Box- und expliziter Wiederverwendung von Services wird eine implizite Wiederverwendung bspw. durch andere Services unterstützt.
- $\boldsymbol{WV = 4}$ („ausreichend“): Mindestens Black-Box- und explizite Wiederverwendung von Services sind möglich.
- $\boldsymbol{WV = 5}$ („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

4.4.10 Standardunterstützung

Als zentrale Aufgaben einer ESB-Implementierung wird sowohl in der Forschung (siehe Kapitel 2) als auch in der Industrie (siehe Kapitel 3) die Unterstützung von Standards angesehen. Diese, so die Ansicht, ermöglichen die Integration von heterogenen Anwendungen und Systemen ohne auf herstellerspezifische Beschränkungen einzugehen. Der Fokus liegt dabei auf offenen Standards, d. h. Standards die nicht an einen speziellen Hersteller gebunden, sondern frei verfügbar sind. Solche und weitere Standards wurden bereits in Kapitel 2.3 aufgeführt (siehe dazu auch Anhang B) und sollen nun im Zuge einer Bewertung in folgende Kategorien eingeteilt werden (siehe Tabelle B.7 im Anhang B):

- **Grundlegende Standards:** Standards, die für eine ESB-Implementierung unverzichtbar sind.

- **Aufbauende Standards:** Aufbauende Standards gehen über grundlegende Standards hinaus, sind aber im Rahmen einer leistungsfähigen ESB-Implementierung als wichtig und notwendig anzusehen.
- **Fortgeschrittene Standards:** Fortgeschrittene Standards gehen noch weiter und können neben Kernfunktionalitäten auch darüber hinausgehende Funktionen betreffen.
- **Ergänzende Standards:** Ergänzende Standards sind typischer Weise proprietäre Standards oder Standards, die als eine mögliche Ergänzung angesehen werden können.

Die verhältnisskalierte Metrik ST bestimmt hierbei die Anzahl der von einer ESB-Implementierung unterstützten Standards. Sie setzt sich aus den Teilmetriken ST_G, ST_A, ST_F und ST_E zusammen und stellt die Summe dieser dar. Die Teilmetriken wiederum entsprechen der Anzahl unterstützter Standards in der jeweiligen Kategorie (G – grundlegende Standards, A – aufbauende Standards, F – fortgeschrittene Standards und E – ergänzende Standards.)

Um die mit der Metrik ST (und seinen Teilmetriken) verbundene Qualität besser bewerten zu können, wird eine zusätzliche, ordinalskalierte Metrik ST' mit einem Wertebereich von eins bis fünf definiert, die von ST bzw. dessen Teilmetriken abhängig ist.

Interpretation von ST':

- $\boldsymbol{ST' = 1}$ („sehr gut“): Es werden sämtliche grundlegende Standards unterstützt ($ST_G = 7$). Weiterhin werden mindestens zehn aufbauende, 15 fortgeschrittene und fünf ergänzende Standards unterstützt ($ST_A \geq 10$, $ST_F \geq 15$, $ST_E \geq 5$).
- $\boldsymbol{ST' = 2}$ („gut“): Es werden sämtliche grundlegende Standards unterstützt ($ST_G = 7$). Weiterhin werden mindestens fünf aufbauende, fünf fortgeschrittene und mindestens ein ergänzender Standard unterstützt ($ST_A \geq 5$, $ST_F \geq 5$, $ST_E \geq 1$).
- $\boldsymbol{ST' = 3}$ („befriedigend“): Es werden mindestens fünf grundlegende Standards unterstützt ($ST_G \geq 5$) , zu denen wenigstens HTTP, SOAP, WSDL und XML gehören. Weiterhin werden mindestens drei aufbauende, drei fortgeschrittene und ggf. ergänzende Standards unterstützt ($ST_A \geq 3$, $ST_F \geq 3$).
- $\boldsymbol{ST' = 4}$ („ausreichend“): Es werden mindestens fünf grundlegende Standards unterstützt ($ST_G \geq 5$), zu denen wenigstens HTTP, SOAP, WSDL und XML gehören. Gegebenenfalls werden weitere Standards unterstützt.
- $\boldsymbol{ST' = 5}$ („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

4.4.11 Routingfähigkeiten

Ein neben Standards gleichfalls wichtiger Punkt betrifft das Routing (auch als intelligentes Routing bezeichnet). Diesbezüglich werden an ESB-Implementierungen höchste Anforderungen gestellt, die es zu bewerten gilt. Zur Beurteilung der Routingfähigkeiten einer ESB-Implementierung können die folgenden Kriterien herangezogen werden:

- Routing auf Basis von Header Informationen (Daten im Nachrichtenkopf),

- Routing auf Basis des Nachrichten-/Datenformats,
- Routing auf Basis des Nachrichteninhalts (Content Informationen),
- Routing auf Basis von (Geschäfts-)prozessen bzw. Prozessabläufen (etwa BPEL).

Zur Bewertung dieser Fähigkeiten wird die ordinalskalierte Metrik *RO* definiert, deren Wertebereich zwischen eins und fünf liegt. Die einzelnen Ausprägungen von *RO* haben folgende Bedeutung.

Interpretation von *RO*:

- ***RO* = 1** („sehr gut“): Alle zuvor beschriebenen Routingvarianten werden unterstützt.
- ***RO* = 2** („gut“): Sowohl Header Informationen als auch Nachrichtenformat und -inhalt können für das Routing genutzt werden.
- ***RO* = 3** („befriedigend“): Neben dem Routing über Header Informationen ist entweder ein Routing gemäß Nachrichtenformat oder gemäß Nachrichteninhalt möglich.
- ***RO* = 4** („ausreichend“): Es ist mindestens Routing auf Basis von Header Informationen möglich.
- ***RO* = 5** („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

4.4.12 Transformationsfähigkeiten

Eine dritte wesentliche Kernaufgabe einer ESB-Implementierung neben Standardunterstützung und Routing stellt die Transformation von Daten dar. Diese Aufgabe hat auf Grund des heterogenen Anwendungsumfelds von ESB-Implementierungen große Bedeutung. Als ein Qualitätskriterium dient der Umfang der angebotenen Transformationsfähigkeiten. Dazu gehören:

- Transformationen mittels XSLT,
- Transformationen mittels XQuery,
- Transformationen von proprietären/nicht XML-basierten Formaten in nicht proprietäre/XML-basierte Formate,
- Transformationen von unterschiedlichen Formaten in (andere) proprietäre/nicht XML-basierte Formate,
- Transformationen auf Basis mehrerer ggf. unterschiedlich formatierter Quellen (zum Zweck der Aggregation).

Die ordinalskalierte Metrik zur Bewertung der genannten Transformationsfähigkeiten wird als *TR* mit einem Wertebereich von eins bis fünf definiert.

Interpretation von *TR*:

- ***TR* = 1** („sehr gut“): Alle genannten Transformationsmöglichkeiten werden unterstützt.

- ***TR* = 2** („gut“): Fast alle Transformationsmöglichkeiten bis auf Transformationen von unterschiedlichen Formaten in (andere) proprietäre/nicht XML-basierte Formate werden unterstützt.
- ***TR* = 3** („befriedigend“): Sowohl XSLT als auch XQuery können zur Transformation genutzt werden und Transformationen von proprietären/nicht XML-basierten Formaten in nicht proprietäre/XML-basierte Formate sind möglich.
- ***TR* = 4** („ausreichend“): Es sind mindestens Transformationen auf Basis von XSLT oder XQuery möglich.
- ***TR* = 5** („ungenügend“): Es werden weniger Bedingungen erfüllt, als für die Bewertung 4 („ausreichend“) erforderlich sind.

4.5 Zusammenfassung

Zum Zwecke einer Übersicht und Zusammenfassung sind nachfolgend in Tabelle 4.2 sämtliche Qualitätskriterien samt Qualitätsmaßen/-metriken und Wertebereichen aufgeführt.

Qualitätskriterien	**Qualitäts-maße /-metriken**	**Skalierung**	**Werte-bereiche**	**Maß-einheit**
Performanz	*AZ*	verhältnisskaliert	$[0 \ldots \infty]$	ms
	PP	verhältnisskaliert	$[1 \ldots \infty]$	–
	ND	verhältnisskaliert	$[0 \ldots \infty]$	$\frac{N}{min}$
Sicherheit	*SI*	ordinalskaliert	[1, 2, . . . 5]	–
Funktionsumfang	*FU*	verhältnisskaliert	[0, 1, . . . 9]	–
	FU'	ordinalskaliert	[1, 2, . . . 5]	–
Benutzbarkeit	*BE*	verhältnisskaliert	[0, 1, . . . 9]	–
	BE'	ordinalskaliert	[1, 2, . . . 5]	–
Test-/Integrierbarkeit	*TI*	verhältnisskaliert	[0, 1, . . . 9]	–
	TI'	ordinalskaliert	[1, 2, . . . 5]	–
Wartbarkeit	*WA*	ordinalskaliert	[1, 2, . . . 5]	–
Plattformunabhängigkeit	*PU*	verhältnisskaliert	[0, 1, . . . 24]	–
	PU'	ordinalskaliert	[1, 2, . . . 5]	–
	PUG	verhältnisskaliert	[0, 1, . . . 24]	–
	PUG'	ordinalskaliert	[1, 2, . . . 5]	–

Skalierbarkeit	SK	ordinalskaliert	[1, 2, ... 5]	–
Wiederverwendbarkeit	WV	ordinalskaliert	[1, 2, ... 5]	–
Standardunterstützung	ST_G	verhältnisskaliert	[0, 1, ... 7]	–
	ST_A	verhältnisskaliert	[0, 1, ... 18]	–
	ST_F	verhältnisskaliert	[0, 1, ... 24]	–
	ST_E	verhältnisskaliert	[0, 1, ... 23]	–
	ST	verhältnisskaliert	[0, 1, ... 72]	–
	ST'	ordinalskaliert	[1, 2, ... 5]	–
Routingfähigkeiten	RO	ordinalskaliert	[1, 2, ... 5]	–
Transformationsfähigkeiten	TR	ordinalskaliert	[1, 2, ... 5]	–

Tabelle 4.2: Zusammenfassung der Qualitätsmaße/-metriken.

Aufbauend auf diesen Maßen und Metriken wird eine Bewertung von ESB-Implementierungen unter dem Gesichtspunkt ausgewählter Qualitätskriterien möglich. Sie bilden damit ein Qualitätsmodell, welches im folgenden Kapitel für die Bewertung ausgewählter ESB-Implementierungen angewandt werden soll.

Kapitel 5

Bewertung von verfügbaren ESB-Implementierungen

5.1 Herangehensweise

Wie bereits im Kapitel 3 gezeigt werden konnte, ist das Feld von ESB-Produkten sehr weit und vielschichtig. Dieser großen Vielfalt Rechnung tragend, werden beispielhaft nur die folgenden drei unterschiedlichen ESB-Implementierungen untersucht[1]:

- *Artix*, eine kommerzielle ESB-Implementierung von IONA Technologies.
- *Mule*, eine freie ESB-Implementierung von MuleSource, Inc. (siehe Kapitel 3.8).
- *ServiceMix*, eine weitere freie ESB-Implementierung.

Die Anwendung des Qualitätsmodells soll in der Art erfolgen, dass Schritt für Schritt einzelne Qualitätsaspekte betrachtet und dazugehörige Qualitätsmaße und -metriken in gleicher Weise für alle zu untersuchenden ESB-Implementierungen angewendet werden. Dies hat den Vorteil, dass unterschiedliche Qualitätseigenschaften einzelner ESB-Implementierungen sehr gut miteinander verglichen werden können.

5.2 Anwendung des Qualitätsmodells

5.2.1 Performanz

Zur Beschreibung der Performanz mittels der beiden Maße Antwortzeit (AZ) und Nachrichtendurchsatz (ND) sowie der Metrik Anzahl paralleler Prozesse (PP) ist zunächst ein

[1] Auswahlgrundlage bilden die folgenden Prämissen. Die jeweilige ESB-Implementierung muss in einer stabilen Version vorliegen und zumindest für eine Evaluation verfügbar sein. Außerdem soll gezeigt werden, dass sich das entwickelte ESB-Qualitätsmodell nicht nur für die in Kapitel 3 beschriebenen ESB-Implementierungen anwenden lässt.

entsprechendes Szenario zu definieren. Die dafür verwendete Hard- und Systemsoftware (siehe Tabelle F.1) sowie die gewählte Deployment-Topologie (siehe Abbildung F.1) sind im Anhang dargestellt. Weiterhin lässt sich das gewählte Szenario in folgender Weise beschreiben. Innerhalb eines lokalen Netzwerks (Local Area Network – LAN) liegen zwei verschiedene Rechner mit jeweils einem Service-Container und wiederum jeweils einem Service vor. Beide zusammen bilden eine als rudimentär anzusehende SOA-Implementierung auf Basis einer ESB-Architektur. Einer der beiden Rechner fungiert hierbei als Client und der andere als Server. Das bedeutet konkret, dass die an den Service des Client-Knotens gestellten Anfragen an den Service des Server-Knotens weitergereicht, dort bearbeitet und von da aus wieder zum Client zurückgeschickt werden. Der Aufruf des Client-Services erfolgt dabei über JMeter[2], welches sich ebenfalls auf dem Client-Rechner befindet. Die Interaktionskette einer einzelnen Anfrage zeigt das im Anhang zu findende Sequenzdiagramm (siehe Abbildung F.2). Gleichfalls ist die Schnittstellenbeschreibung der Services (für Client und Server identisch) im Anhang verzeichnet (siehe Listing F.1). Zur Durchführung der Performanzmessung wurde weiterhin mittels JMeter ein entsprechender Testfall (Lasttreiber) konstruiert[3]. Dieser ist so definiert, dass zyklisch und parallel Anfragen (SOAP-Nachrichten) an den Service des Clients gestellt und während dessen AZ und ND gemessen werden[4]. Hierfür verwendet JMeter Threads[5], deren Anzahl im Rahmen des Szenarios sukzessive gesteigert wird. Die aktuelle Anzahl der Threads repräsentiert hierbei den Wert für PP. Sollte die von einem Thread verschickte Anfragenachricht nicht korrekt beantwortet werden, so wird dieser beendet und die entsprechende Anfrage von der Messung ausgeschlossen. Das führt dazu, dass sich mit der Zeit die Anzahl der Threads (Anzahl paralleler Anfragen) und damit PP auf einem bestimmten Niveau einpegelt. In gleicher Weise verhält sich der Nachrichtendurchsatz. Der im Anhang zu findende Screenshot (siehe Abbildung F.3) zeigt hierzu die Durchführung der Messung und den ansteigenden Verlauf des Nachrichtendurchsatzes. Die Abbruchbedingung des Testfalls ist erreicht, wenn der Wert von ND innerhalb einer Minute um weniger als 5 Nachrichten (N) schwankt. Die Durchschnitte der bis zu diesem Zeitpunkt gemessenen Werte für AZ und ND sowie der erreichte Wert für PP stellen das Messergebnis des Testfalls (Szenarios) dar und sind für Artix, Mule und ServiceMix in Tabelle 5.1 (siehe Kapitel 5.3, S. 95) aufgeführt. Hierbei hat sich gezeigt, dass Artix als deutlich leistungsfähiger einzustufen ist als Mule und ServiceMix. Während Mule einen Nachrichtendurchsatz von 6.024,75 Nachrichten pro Minute ($\frac{N}{min}$) erzielte und ServiceMix von 5.514,16 $\frac{N}{min}$, schaffte Artix mit 10.820,67 $\frac{N}{min}$ knapp das Doppelte. Die dabei aufgetretene höhere Antwortzeit von Artix (1.084 ms) resultierte vor allem daraus, dass Artix in der Lage war, parallel 214 Anfrageprozesse stabil zu bearbeiten, Mule ($AZ = 522\ ms$) und ServiceMix ($AZ = 553\ ms$) hingegen jeweils nur 51. Angesichts dessen ist Artix als performanteste Implementierung einzustufen, gefolgt von Mule und danach ServiceMix. Die grafische Darstellung von AZ und ND ist im Anhang zu finden (siehe Abbildung F.4, Abbildung F.5, Abbildung F.6).

5.2.2 Sicherheit

Ein Aspekt, der im Umfeld professioneller Geschäftsanwendungen von besonderer Bedeutung ist, betrifft die Sicherheit und damit verbundene Mechanismen und Technologien zur Umsetzung. Die Begründung für die Wichtigkeit dieses Qualitätskriteriums liegt im Anwendungsfeld einer ESB-Implementierung, das weniger innerhalb geschlossener Netze liegt,

[2] JMeter ist ein Programm, das hauptsächlich zum dynamischen Testen und Messen von Web-/Netzwerkanwendungen gedacht ist.

[3] Eine XML-basierte Beschreibung des Testplans (umfasst die Testfälle aller drei Implementierungen) ist dem Anhang zu entnehmen (siehe Listing F.4).

[4] Listing F.2 und F.3 zeigen die Quelltexte von Anfrage- und Antwortnachricht.

[5] Ein Thread stellt im Rahmen des Testfalls/Szenarios einen einzelnen Anfrageprozess dar, der zyklisch wiederholt wird und jeweils genau eine Anfrage stellt.

sondern vielmehr darüber hinaus reicht. Trotz dieser Tatsache verfügt unter den drei ESB-Implementierungen einzig Artix über Sicherheitsmechanismen, die bei Mule und ServiceMix komplett fehlen. Zu diesen gehören bei Artix die Authentifizierung und Verschlüsselung mittels SSL/TLS sowie darüber hinaus auch eine Unterstützung von HTTPS. Ebenso wird dadurch die Nachrichtenintegrität gewährleistet. Bezüglich Web Services gibt es zusätzlich eine Unterstützung für WS-Security und damit weitere Mechanismen bezüglich Authentifizierung und Integrität. Ebenso gibt es mit der Common Secure Interoperability (CSI) auch Sicherheitsmechanismen für CORBA-Verbindungen. Der mit der Authentifizierung eng verbundene Aspekt der Autorisation wird bei Artix über ein spezielles Sicherheitsframework realisiert, welches Verbindung zu einem externen Sicherheitsprovider aufnimmt und darüber eine Rechtevergabe vornimmt. Somit erstrecken sich die angebotenen Sicherheitsmechanismen über Vertraulichkeit (Authentifizierung, Autorisation und Verschlüsselung) bis hin zu Integrität und führen zu einer Bewertung mit „gut“. Artix ist folglich in Fragen der Sicherheit als überdurchschnittlich leistungsfähig anzusehen im Gegensatz zu den diesbezüglich vollkommen unzureichenden ESB-Implementierungen Mule und ServiceMix (beide nur „ungenügend“). Eine Übersicht der jeweiligen Werte von *SI* zeigt Tabelle 5.1 (siehe Kapitel 5.3, S. 95).

5.2.3 Funktionsumfang

Bei einer Betrachtung des bereitgestellten Funktionsumfangs gibt es verschiedene zu berücksichtigende Teilaspekte. An erster Stelle steht die Abbildung und Umsetzung von Geschäftsprozessen. Artix bietet hier im Gegensatz zu Mule und ServiceMix nicht nur eine BPEL-Engine an, sondern mit dem Artix Designer auch eine GUI für die Modellierung von Geschäftsprozessen. Über die jeweiligen BPEL-basierten Prozessbeschreibungen ist es weiterhin möglich, Services zu orchestrieren bzw. zu choreographieren. Das Management von Services und Service-Containern ist in sehr beschränktem Maße mit allen ESB-Implementierungen möglich. Hierfür gibt es JMX-Schnittstellen, die ein grundlegendes Management ermöglichen. Allein Artix bietet über eine Management Konsole weiterführende Verwaltungsmöglichkeiten an, die bis hin zu einer Unterstützung von Servicelebenszyklen reichen. Mule und ServiceMix unterstützen Servicelebenszyklen gleichfalls, jedoch nicht über eine GUI. Zur Gewährleistung geforderter Dienstgüten (QoS) werden von allen Implementierungen Maßnahmen bspw. zu Sicherheit, Verfügbarkeit und Zuverlässigkeit angeboten, verzichtet wird jedoch auf deren Überwachung. So können keine Grenzen für Dienstgüten definiert werden und auf eine Unterstützung von Dienstgütevereinbarungen (SLA) wird in gleicher Weise verzichtet. Über entsprechende Transaktionsmanager bieten sowohl Artix als auch Mule und ServiceMix die Möglichkeit der Transaktionsverwaltung, jedoch sind lediglich Artix und Mule in der Lage, einen Transaktionskontext über mehrere Services hinweg zu ermöglichen. Kaum Unterschiede gibt es beim Deployment. Hier stellen sich alle Implementierungen als flexibel dar und erlauben eine Installation auf Client, Anwendungssystem/Applikationsserver oder als eigenständiger Netzwerkknoten. Ähnlich sieht es bei Föderation und Erweiterbarkeit aus. Relativ einfach können neue Services hinzugefügt und die jeweiligen Implementierungen um weitere Service-Container sowie darin gekapselte Geschäfts-/Integrationslogik erweitert werden. Ebenfalls prinzipiell möglich ist die Föderation mit anderen ESB-Implementierungen. Dies erfordert jedoch, dass eine möglichst große Menge gemeinsamer Standards unterstützt wird, da ansonsten eine Zusammenarbeit nur schwer möglich und wenig sinnvoll ist. Etwas anders sieht das bei ServiceMix aus, da dieser auf JBI basiert. Sollte dieser mit einer anderen ebenfalls JBI-konformen ESB-Implementierung föderiert werden, ist es prinzipiell egal, welche Standards beide ESB-Implementierungen unterstützen. Ursache dafür ist die Fähigkeit, Binding Components und Service Engines austauschen und diese damit beiden ESB-Implementierungen zur Verfügung stellen zu können. Wenngleich alle drei Implementierungen recht viele der geforderten Funktionalitäten vorweisen können, wirkt die Art der

Umsetzung bei Artix am ausgereiftesten. An Stellen, wo Mule und ServiceMix nur einzelne oder wenige Maßnahmen bereithalten, bietet Artix in der Regel oft verschiedene Möglichkeiten der Umsetzung an. Zu Recht schneidet der mit „sehr gut“ bewertete Artix besser als die mit „gut“ bewerteten Implementierungen Mule und ServiceMix ab. Trotz alledem sind die betrachteten Implementierungen durchaus noch verbesserungsfähig, speziell in Bereichen wie Dienstgütevereinbarungen oder B2B-Anbindungen. Tabelle 5.1 zeigt die jeweiligen Werte für *FU* und *FU'* (siehe Kapitel 5.3, S. 95). Weiterhin ist eine grafische Darstellung von *FU* im Anhang enthalten (Abbildung F.7).

5.2.4 Benutzbarkeit

Unter dem Gesichtspunkt der Benutzbarkeit treten zwischen den einzelnen ESB-Implementierungen große Unterschiede zu Tage. Während ServiceMix auf jegliche Hilfen wie grafische Werkzeuge oder sonstige Assistenten verzichtet, bietet Artix zu fast jeder der genannten Aufgaben eine Unterstützung an. Im Zentrum stehen dabei zum einen der Artix Designer (eine Erweiterung der Eclipse IDE) und zum anderen die Artix Management Console, die im Wesentlichen der Verwaltung der Services und des ESB dient. Einzig in Sachen Suchen und Finden von Services treten Defizite auf, wobei hier das Problem eher in einem fehlenden Repository bzw. einer fehlenden Service-Registry zu sehen ist. Alles in allem erreicht Artix die Bewertung „gut“, ServiceMix hingegen nur „ungenügend“. Mule ordnet sich mit seinen Fähigkeiten zwischen den beiden recht unterschiedlich abschneidenden Implementierungen ein und kann mit dem angebotenen Leistungsumfang als „befriedigend“ eingestuft werden. Ein Defizit bei Mule ist die fehlende Unterstützung für stärker dynamisch geprägte Aufgaben, speziell das Management und die Überwachung der Services und des ESB betreffend. Die konkreten Werte von *BE* und *BE'* sind in Tabelle 5.1 (siehe Kapitel 5.3, S. 95) zu finden und eine grafische Darstellung von *BE* liefert Abbildung F.8 im Anhang.

5.2.5 Testbarkeit, Integrierbarkeit

Zur Frage nach der Test- und Integrierbarkeit entsteht der Eindruck, dass bei keiner der drei ESB-Implementierungen ein Konzept vorhanden ist. Während ServiceMix eine Testunterstützung vollständig schuldig bleibt und diesbezüglich keinerlei Unterstützung anbietet, ist eine solche wenigstens in sehr geringem Maße bei Artix und bei Mule vorhanden. Beide Implementierungen bieten über grafische Entwicklungsumgebungen die Möglichkeit für statische Tests und Artix sogar die Möglichkeit zum Debugging. Dennoch ist Mule im Gegensatz zu Artix etwas besser zu bewerten, da dieser neben einem rudimentären Benchmarking gleichfalls die Durchführung von Black-Box Tests ermöglicht. Somit kann Mule als einzige ESB-Implementierung mit „ausreichend“, Artix und ServiceMix hingegen nur mit „ungenügend“ bewertet werden. Einen Aspekt, den jedoch alle Implementierungen ausklammern, ist die Integrierbarkeit, also ein schrittweiser Test einzelner Services bis hin zu einem Test des Gesamtsystems oder zumindest einzelner, miteinander integrierter Prozessabläufe. Die erreichten Werte von *TI* und *TI'* zeigt Tabelle 5.1 (siehe Kapitel 5.3, S. 95). Eine grafische Darstellung von *TI* liefert Abbildung F.9 im Anhang.

5.2.6 Wartbarkeit

Bezüglich der Wartbarkeit stellen sich alle drei ESB-Implementierungen als leistungsfähig heraus. Das betrifft in besonderem Maße die Punkte Erweiterung und Anpassung. Es ist leicht möglich, weitere Services oder dadurch gekapselte Anwendungen und Systeme hinzu-

zufügen sowie flexibel Änderungen an der gewählten Deployment-Topologie vorzunehmen. Ebenfalls zum Tragen kommt die Unterstützung unterschiedlichster Plattformen (auf die später noch genauer eingegangen wird), die es erlaubt, auch nachträglich Plattformänderungen ohne großen Aufwand realisieren zu können. Weniger leistungsfähig sind die Korrekturverfahren ausgeprägt. Zwar gibt es bei allen Implementierungen die Möglichkeit, fehlerhafte Nachrichten zu erkennen und entsprechend darauf zu reagieren (bspw. durch so genanntes Failover Routing oder Fehlerbenachrichtigung), doch wären weitere Maßnahmen bezüglich der eigentlichen Korrektur wünschenswert. Neben Erweiterungen oder Anpassungen sind Verbesserungen ein weiterer wichtiger Gesichtspunkt. Hier müssen Artix, Mule und ServiceMix jedoch unterschiedlich betrachtet werden. Während Artix vor allem von seinem Funktionsumfang profitiert und im Rahmen dessen unterschiedliche Dienstgüten und qualitative Anforderungen erfüllen kann, gleichen Mule und ServiceMix dies auf eine andere Art aus. Mule und ServiceMix, deren beider Quellcode frei verfügbar ist und ServiceMix mit einer JBI-konformen Architektur, erlauben noch weiterreichende Anpassungs- und Erweiterungsmöglichkeiten als Artix und darüber wiederum Möglichkeiten zur Verbesserung. Ihr geringerer Funktionsumfang wird also durch eine offenere Architektur in gewisser Weise kompensiert. Nicht ausgeglichen werden kann das Fehlen von vorbeugenden Maßnahmen. Nur Artix bietet in Form einer Management Konsole Werkzeuge an, die es erlauben, frühzeitig Probleme oder Defizite ausfindig zu machen. Deshalb erreicht Artix als einzige Implementierung die Bewertung „sehr gut“, Mule und ServiceMix lediglich ein „gut“. Dennoch können alle ESB-Implementierungen zeigen, worin die Stärken eines ESB liegen, nämlich Anforderungsänderungen (funktionaler, systembezogener oder qualitativer Art) schnell und flexibel umzusetzen. Die Werte von *WA* sind nochmals in Tabelle 5.1 aufgeführt (siehe Kapitel 5.3, S. 95).

5.2.7 Plattformunabhängigkeit/Portierbarkeit

In Bezug auf Plattformunabhängigkeit und Portierbarkeit erweisen sich alle ESB-Implementierungen als ähnlich leistungsstark. Leider verfügt lediglich Artix über eine Mainframeunterstützung (z/OS), die bei Mule und ServiceMix fehlt. Aus diesem Grund kann Artix mit „gut“ bewertet werden und auch besser abschneiden als die Konkurrenz. Mule und ServiceMix (beide mit „befriedigend“ bewertet) muss man jedoch zugute halten, dass sie im Gegensatz zu Artix auch unter Mac OS X und dessen Servervariante genutzt werden können. Weiterhin ist zu beachten, dass bei allen ESB-Implementierungen lediglich die explizit ausgewiesenen Betriebssysteme berücksichtigt wurden. Vor allem Mule und ServiceMix mit ihrem offenen und auf Java basierenden Quellcode dürften jedoch potenziell noch weitaus mehr Plattformen unterstützen (sofern eine Java-Laufzeitumgebung vorhanden ist). Die Plattformunabhängigkeit grafischer Werkzeuge und Komponenten kann nur für Artix und Mule angegeben werden, da ServiceMix über keine derartigen Komponenten verfügt. Im Gegensatz zur eigentlichen ESB-Implementierung ist, hinsichtlich der grafischen Unterstützung, Mule leicht im Vorteil. Ursache dafür ist, dass Mule (mit „gut“ bewertet) im Gegensatz zu Artix (mit „befriedigend“ bewertet) auch in diesem Bereich eine Unterstützung für Mac OS X und Mac OS X Server liefert. Zusammenfassend sind die konkreten Werte für *PU*, *PU'*, *PUG* und *PUG'* in Tabelle 5.1 aufgeführt (siehe Kapitel 5.3, S. 95). Eine grafische Darstellung von *PU* und *PUG* liefert Abbildung F.10 im Anhang.

5.2.8 Skalierbarkeit

Die Skalierbarkeit betreffend sind sowohl Artix als auch Mule und ServiceMix in der Lage, sämtliche Forderungen vertikaler Skalierbarkeit zu erfüllen. Es können einzelne Service-Con-

tainer flexibel um Services sowie auch um andere, zusätzliche Funktionalitäten (die bspw. Service-Container gegenüber Services bereitstellen) erweitert und angepasst werden. Anders sieht es im Hinblick auf die horizontale Skalierbarkeit aus. Hier fällt vor allem Mule auf, dessen horizontale Skalierungsmechanismen sich noch in der Entwicklung befinden und ein großes Manko darstellen. Im Gegensatz dazu verfügen Artix und ServiceMix durchaus über entsprechende Fähigkeiten, die beiden eine horizontale Skalierung zur Steigerung von Performanz und Ausfallsicherheit gestatten. Hierfür wird bei Artix ein spezieller Locator Service genutzt, der es ermöglicht, unterschiedliche Instanzen eines Services zu einem logischen Service zusammenzufassen. In Abhängigkeit verfügbarer Service-Instanzen erfolgt dann deren Auswahl. Bei ServiceMix hingegen werden komplette Service-Container zu einem Cluster verbunden, die alle über die gleichen Service-Instanzen verfügen. Ähnlich wie bei Artix können auch darüber je nach Verfügbarkeit unterschiedliche Service-Instanzen genutzt werden. Aus den genannten Gründen können Artix und ServiceMix mit „sehr gut“, Mule hingegen nur mit „befriedigend“ bewertet werden. Die Werte von SK sind nochmals in Tabelle 5.1 (siehe Kapitel 5.3, S. 95) aufgeführt.

5.2.9 Wiederverwendbarkeit

In Sachen Wiederverwendbarkeit unterscheiden sich alle drei ESB-Implementierungen nur minimal. Gemeinsam können deren Fähigkeiten mit „befriedigend“ bewertet werden. Grund für keine bessere Bewertung sind fehlende Registries, die es gestatten, über den Kontext einzelner Service-Container hinaus, sämtliche an den ESB angeschlossene Services suchen, finden, veröffentlichen und einbinden zu können. Ein großer Mangel, der in diesem Zusammenhang zu Tage tritt, ist die fehlende Unterstützung von UDDI, einem zentralen Web Service Standard. Einzig Artix unterstützt UDDI und kann damit, zumindest externe UDDI-Verzeichnisse/-Registries und darin eingetragene Services ansprechen und nutzen. Über einen speziellen Locator Service können mit Artix außerdem Services registriert werden. Diese Registrierung dient jedoch im Wesentlichen einer dynamischen Adressauflösung und damit nicht dem Suchen und Finden von Services, wie es eine offensive Wiederverwendung erfordern würde. Weiterhin wird das Thema Versionierung von allen ESB-Implementierungen gleichermaßen unzureichend berücksichtigt. In der Regel und das ist zu kritisieren, wird es nahezu vollständig dem Anwender überlassen. Auf daraus resultierende Wiederverwendungsprobleme sei an dieser Stelle nur verwiesen (siehe [DE05], S. 72 ff.). Eine Auflistung der einzelnen Werte für WV zeigt Tabelle 5.1 (siehe Kapitel 5.3, S. 95).

5.2.10 Standardunterstützung

Die Standardunterstützung der untersuchten ESB-Implementierungen stellt sich in gewisser Weise als Schwachstelle heraus. Obwohl Mule und ServiceMix hier mit „befriedigend“ und Artix sogar mit „gut“ bewertet werden können, gibt es dennoch einige Defizite. Zu diesen gehören bei Mule und ServiceMix an erster Stelle eine fehlende Unterstützung von UDDI. Ein derartiger Mangel ist durchaus als gravierend anzusehen, da UDDI nicht nur ein Grundpfeiler der Web Service Technologie darstellt, sondern darüber hinaus auch für die Umsetzung einer SOA von Bedeutung ist. Überhaupt muss gleichermaßen festgestellt werden, dass WS-Standards nur sehr beschränkt und B2B-Standards im Prinzip gar nicht unterstützt werden. Ausdruck findet dies in der nur sehr geringen Anzahl angebotener aufbauender und speziell fortgeschrittener Standards. Hier gibt es für alle ESB-Implementierungen einen großen Nachholbedarf. Positiv fällt die Unterstützung von Mule in Bezug auf proprietäre MOM auf, weshalb Mule hinsichtlich der erweiterten Standards sogar besser abschneidet als Artix. Insgesamt gesehen liegt Artix vor Mule gefolgt von ServiceMix. Alle Werte für ST_G, ST_A, ST_F,

ST_E, ST und ST' sind Tabelle 5.1 zu entnehmen (siehe Kapitel 5.3, S. 95). Eine grafische Darstellung von ST_G, ST_A, ST_F, ST_E und ST liefert Abbildung F.10 im Anhang.

5.2.11 Routingfähigkeiten

Im Hinblick auf die Routingfähigkeiten der einzelnen ESB-Implementierungen werden trotz unterschiedlicher Umsetzungen sämtliche Forderungen erfüllt. Sowohl Artix als auch Mule und ServiceMix können mit „sehr gut“ bewertet werden. Besonders hervorzuheben ist, dass alle drei Implementierungen BPEL unterstützen. Sie sind dadurch in der Lage, ein Routing von Nachrichten entsprechend definierter (Geschäfts-)Prozesse durchzuführen. Andere Formen des Routings besonders inhaltsbezogenes Routing werden über Skripte realisiert, die sich im Wesentlichen auf XPath, zum Teil aber auch auf eigene Skript- bzw. Regel-Engines (Groovy, Drools) stützen. Die bereits genannten Werte für *RO* sind nochmals in Tabelle 5.1 aufgeführt (siehe Kapitel 5.3, S. 95).

5.2.12 Transformationsfähigkeiten

Wie bereits an anderen Stellen zuvor, sind die Unterschiede der drei ESB-Implementierungen auch im Bereich der Transformationsfähigkeiten nur sehr gering. So erlauben alle Implementierungen, Transformationen an XML-basierten Nachrichten bzw. Daten mittels XSLT durchzuführen. Leider nicht möglich ist die Nutzung von XQuery, um bspw. Anfragen an XML-Dokumente zu stellen und über diesen Weg Transformationen durchzuführen. Aus diesem Grund können Artix, Mule und auch ServiceMix nur mit „ausreichend“ eingestuft werden. Weitere Schwächen liegen bei allen Implementierungen in einer äußerst unzureichenden Unterstützung proprietärer bzw. nicht XML-basierter Formate. Weder CSV-Dateien, EDI-Formate oder Formate anderer geschlossener Anwendungen werden direkt unterstützt. Sieht man davon ab, werden zumindest mit der Fähigkeit zur Aggregation von Nachrichten und Daten (von allen Implementierungen angeboten) noch zusätzliche Transformationsmöglichkeiten bereitgestellt. Die Werte für *TR* sind gleichfalls in Tabelle 5.1 aufgeführt (siehe Kapitel 5.3, S. 95).

5.3 Vergleich

Die Qualitätseigenschaften der untersuchten ESB-Implementierungen unterscheiden sich in vielen Punkten nur geringfügig und können oft als vergleichbar angesehen werden. Dennoch gibt es Unterschiede, wie Tabelle 5.1 und die Häufigkeitsverteilungen der ordinalskalierten Metriken (siehe Abbildung F.12 im Anhang) zeigen. Dem kann entnommen werden, dass vor allem der von IONA angebotene Artix gegenüber Mule und ServiceMix in der Mehrzahl bessere Bewertungen erreicht. Außerdem verfügt er gerade auf wichtigen Gebieten wie Sicherheit, Funktionsumfang, Benutzbarkeit und nicht zuletzt auch Performanz über höhere Qualitätseigenschaften und kann zu Recht als die qualitativ beste der drei untersuchten Implementierungen angesehen werden. Nicht so klar ist das Bild im Vergleich zwischen Mule und ServiceMix. Obwohl ServiceMix im Gegensatz zu Mule häufiger mit „sehr gut“ bewertet wurde, erhielt umgekehrt Mule die Bewertung „ungenügend“ nicht so häufig wie ServiceMix. Weiterhin erzielen Mule und ServiceMix in vielen wichtigen Punkten oft übereinstimmende Bewertungen und haben gleichfalls zum Teil große Defizite. So fehlt ServiceMix an erster Stelle eine grafische Unterstützung und Mule die Fähigkeit zur horizontalen Skalierbarkeit. Insgesamt können daher beide Implementierungen als qualitativ ebenbürtig angesehen wer-

den. Im Vergleich zu Artix fehlen Mule und ServiceMix speziell Sicherheit, Funktionsumfang und Benutzbarkeit betreffend noch ein gewisses Maß an Leistungsumfang. Oft kann Artix bezüglich einzelner Qualitätskriterien verschiedene Umsetzungsmöglichkeiten bereitstellen, während Mule und ServiceMix dies nicht erlauben. Eine zusammenfassende Darstellung aller Qualitätsmaße und -metriken ist Tabelle 5.1 zu entnehmen.

Qualitäts-kriterien	**Qualitäts-maße/ -metriken**	**Artix**	**Mule**	**ServiceMix**
Performanz	AZ	1.084 ms	522 ms	553 ms
	PP	214	51	51
	ND	10.820,67 $\frac{N}{min}$	6.024,75 $\frac{N}{min}$	5.514,16 $\frac{N}{min}$
Sicherheit	SI	gut	ungenügend	ungenügend
Funktions-umfang	FU	9	8	7
	FU'	sehr gut	gut	gut
Benutzbarkeit	BE	8	6	0
	BE'	gut	befriedigend	ungenügend
Test-/Integrier-barkeit	TI	2	3	0
	TI'	ungenügend	ausreichend	ungenügend
Wartbarkeit	WA	sehr gut	gut	gut
Plattform-unabhängigkeit	PU	10	10	10
	PU'	gut	befriedigend	befriedigend
	PUG	9	10	—
	PUG'	befriedigend	gut	—
Skalierbarkeit	SK	sehr gut	befriedigend	sehr gut
Wiederver-wendbarkeit	WV	befriedigend	befriedigend	befriedigend
Standard-unterstützung	ST_G	7	6	6
	ST_A	8	7	6
	ST_F	6	3	3
	ST_E	6	9	7
	ST	27	25	19
	ST'	gut	befriedigend	befriedigend
Routing-fähigkeiten	RO	sehr gut	sehr gut	sehr gut
Transformations-fähigkeiten	TR	ausreichend	ausreichend	ausreichend

Tabelle 5.1: Qualitätsmaße und -metriken der untersuchten ESB-Implementierungen.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde zunächst dargelegt, wie der Begriff ESB einzuordnen und zu verstehen ist. Hierzu erfolgte im ersten Kapitel ein historischer Abriss. Dieser zeigte neben den Entwicklungen verschiedener Konzepte und Technologien auch die damit verbundenen und oft nicht hinreichend erfüllten Zielstellungen. In dieser Kette von Weiterentwicklungen gehört die SOA zu den jüngeren Konzepten, deren Realisierung mit einem ESB ermöglicht werden soll. SOA und speziell der ESB erben viele der bereits seit langem existierenden Zielstellungen, unter denen die Wiederverwendbarkeit eine der wichtigsten ist. Die zentrale Aufgabe eines ESB ist es folglich, diese Zielstellungen im Rahmen einer SOA umzusetzen.

Das sich anschließende Kapitel diente dazu, auf der bereits erfolgten Einordnung aufbauend, den Begriff ESB noch näher und speziell aus der Sicht der Forschung zu charakterisieren. Von Interesse war hierbei neben dem Vergleich unterschiedlicher Definitionen, auch bestehende Zusammenhänge zu anderen bereits beschriebenen Konzepten und Technologien aufzuzeigen. An erster Stelle sind diesbezüglich Web Services und MOM zu nennen. Es stellte sich heraus, dass gerade diese von den meisten Autoren als zentrale technologische Grundlage eines ESB angesehen werden. Als weiteres Ergebnis konnte trotz unterschiedlicher Sichtweisen oft ein großes Maß an Konsens festgestellt werden, wenngleich zum Teil mit abweichenden Prämissen und Bezeichnungen gearbeitet wurde. Ein Hauptunterschied, der zwischen den einzelnen Definitionen zutage trat, betraf das Verständnis des ESB als Architekturkonzept bzw. als dessen Implementierung. Dies berücksichtigend wurde neben dem Begriff ESB, der Begriff ESB-Implementierung eingeführt und beide gesondert definiert. Des Weiteren wurden die zum Teil in den Definitionen enthaltenen und als wesentlich zu sehenden Aufgaben und funktionalen Eigenschaften beschrieben. Sie bildeten zusammen mit getroffenen Begriffsdefinitionen die Grundlage für die weiteren Untersuchungen.

Eine praxisorientierte Sichtweise fokussierend, erfolgte im dritten Kapitel die Auseinandersetzung mit dem ESB aus dem Blickwinkel der Softwareindustrie. Von zentraler Bedeutung war dabei die Frage nach dem wie, also der konkreten Umsetzung des Architekturkonzepts ESB. Ähnlich wie bereits im vorangegangenen Kapitel waren auch hier die Übereinstimmungen größer als die Unterschiede. Vor allem das Konzept der bereits im zweiten Kapitel beschriebenen ESB-Service-Container fand sich sehr häufig wieder. Weiterhin wurde deutlich, dass mit dem ESB bestehende Konzepte und Technologien nicht ersetzt sondern vielmehr ergänzt werden. Der ESB spielt in diesem Umfeld die Rolle eines Mediators, der dezentral

unterschiedliche Anwendungen und Systeme verbindet und sowohl Integrations- als auch Geschäftslogik in Form von Services bereitstellt. Außerdem konnte festgestellt werden, dass viele ESB-Implementierungen oft über weiterführende bzw. komplementäre Komponenten verfügen und mit diesen geforderte oder gar zusätzliche Funktionen anbieten. Ein in Teilen abweichendes Verständnis zeigten die Implementierungen von Cape Clear, MuleSource und Microsoft. So will man bei Cape Clear die eigene ESB-Implementierung als MOM-neutrale und damit weitaus flexiblere Implementierung verstanden wissen und distanziert sich von anderen Anbietern, die ihre ESB-Implementierungen mit eigener MOM verknüpfen. Mule hingegen unterscheidet sich in der Form von anderen Implementierungen, indem an Stelle von Services so genannte UMO Komponenten (serviceähnliche, javabasierte Nachrichtenobjekte) eingesetzt werden. Microsoft schließlich glaubt mit einem um Web Services erweiterten Integrationsserver weitaus leistungsfähiger zu sein als vergleichbare ESB-Implementierungen. Nichts desto trotz stellten sich die meisten ESB-Implementierungen als definitionskonform heraus und sind durchaus in der Lage, den im vorherigen Kapitel genannten Aufgaben und Anforderungen gerecht zu werden.

Aufbauend auf den umfangreichen Betrachtungen verschiedener ESB-Implementierungen diente das vierte Kapitel dazu, ein Qualitätsmodell zu entwickeln, das einen Vergleich unterschiedlicher ESB-Implementierungen ermöglichen soll. Diesbezüglich wurden sowohl allgemeine als auch spezielle, sich aus den funktionalen ESB-Anforderungen ableitende Qualitätseigenschaften definiert. Gemäß dieser Qualitätsfaktoren erfolgte eine Zuordnung von entsprechenden Qualitätskriterien, aus denen dann die konkreten Qualitätsmaße und -metriken abgeleitet wurden. Außerdem erfolgte, soweit möglich und sinnvoll, eine Zuordnung weiterer ordinalskalierter Metriken. Dies hat den Zweck, neben einer rein quantifizierenden Bewertung einzelner Qualitätsfaktoren und -kriterien außerdem eine qualifizierende Bewertung in Anlehnung an ein Notensystem vornehmen zu können. Die Summe der somit aufgestellten Qualitätsmetriken und -maße stellt das Qualitätsmodell dar.

Abschließend erfolgte im fünften und vorletzten Kapitel die Anwendung des Qualitätsmodells für drei verschiedene ESB-Implementierungen. Zu diesen gehörten Artix, Mule und ServiceMix. Während Artix eine kommerzielle ESB-Implementierung darstellt, handelt es sich bei Mule und ServiceMix um freie, quellcodeoffene ESB-Implementierungen. Unter den beiden quellcodeoffenen Implementierungen stellte sich Mule als leistungsfähiger und von höherer Qualität als der JBI-konforme ServiceMix heraus. Dennoch reichen auch die Fähigkeiten von Mule hinsichtlich Leistung und Qualität nicht an die von Artix heran. Artix als einzige kommerzielle Implementierung kann somit die höheren Anschaffungskosten in Form besserer Qualitätseigenschaften rechtfertigen.

Zusammenfassend lässt sich sagen, dass der ESB als Konzept und ESB-Implementierungen als dessen Umsetzung weit über Web Service Wrapper hinausgehen und keinesfalls als funktional beschränkte Integrationsserver zu sehen sind. Vielmehr ist der angebotene Funktionsumfang anderen Integrationslösungen durchaus ebenbürtig. Neu im Vergleich zu Integrationsservern und dem Konzept der EAI ist jedoch die Verteilung und dezentrale Anordnung nicht nur von Geschäfts- sondern auch von Integrationslogik. Im Zentrum stehen dabei leichtgewichtige und flexibel erweiterbare ESB-Service-Container, die basierend auf zahlreichen Standards, Adaptern und entsprechender Kommunikationsinfrastruktur (Web Services, MOM, etc.) den Weg für eine schrittweise Integration freimachen. Nicht zuletzt ist sogar die Entwicklung neuer Konzepte auf anderen Gebieten, wie bspw. ESP, dieser Architektur und dem konsequenten Einsatz von XML zu verdanken. Im Umfeld kommerzieller ESB-Implementierungen fällt jedoch auf, dass viele eine ausgeprägte herstellerspezifische Handschrift tragen. Je nachdem aus welchem Umfeld (Integrationsserver, MOM, Applikationsserver usw.) der Hersteller stammt, wird davon die angebotene ESB-Lösung oft beeinflusst, im schlimmsten Fall gar dominiert. Weitere Defizite liegen auf dem Gebiet der unternehmensübergreifenden Integration und der Föderation. Obwohl oft verschiedene Maßnahmen

zur Gewährleistung einer hohen Dienstgüte vorhanden sind und zum Teil die Möglichkeit zu deren Überwachung besteht, fehlt es oft an der Unterstützung B2B-spezifischer Standards. Ebenso selten ist die Unterstützung von Dienstgütevereinbarungen. Dies zeigt, dass trotz aller anders lautender Beteuerungen der Fokus heutiger ESB-Implementierungen noch sehr stark auf einer rein unternehmensinternen Integration liegt. Gleichermaßen deutlich wird das auf dem Gebiet der Föderation. Trotz vieler Standards kann hier oft nur auf die Schnittmenge gemeinsamer Standards zurückgegriffen werden und herstellerspezifische Eigenheiten sind oft die Regel (z. B. die eingesetzte MOM). Ein weiterer Schwachpunkt wird bei der Betrachtung der kommerziellen Nutzung von Services offensichtlich. Es fehlt sowohl an etablierten Standards als auch an einer durchaus möglichen Unterstützung bereits existierender Servicemarktplätze (bspw. StrikeIron). Sieht man von diesen Mängeln ab, sind heutige ESB-Implementierungen trotz alledem als ein gangbarer Weg zur Realisierung einer SOA und damit verbundener Zielstellungen (siehe Kapitel 1.3) anzusehen.

6.2 Ausblick

Die Beantwortung der Frage, welche künftigen Entwicklungen im Umfeld des ESB zu erwarten sind, liefert der ESB selbst. Dessen oberste Maxime ist stets die Unterstützung von Standards, weshalb deren Weiterentwicklung eine große Rolle spielen wird. Das Ziel kann dabei jedoch nicht sein, immer weitere, sich teilweise überschneidende Standards zu definieren (wie etwa im Umfeld von Web Services geschehen), sondern muss darin liegen, gleichartige Standards zu vereinen und zu etablieren. Die besten offenen Standards nützen nur wenig, wenn sie parallel neben einer Vielzahl gleichartiger Standards existieren und infolge einer geringen Verbreitung wieder einen proprietären Charakter annehmen. Nötig und für die Zukunft anzustreben ist daher eine Konsolidierung von Standards, die das Ausschöpfen noch weiteren Integrationspotenzials verspricht.

Ein anderes Ziel, das in gleicher Weise dieser Maxime folgt, ist eine Standardisierung des ESB selbst. Dazu gehört nicht nur eine weitreichende Standardunterstützung, sondern darüber hinaus auch den Aufbau und die Architektur des ESB zu standardisieren. Ansätze für eine Vereinheitlichung sind hier allen voran JBI aber auch JMX, die sowohl zur Standardisierung von ESB-Service-Containern als auch bezüglich des Austauschs anfallender Metadaten und der Managementkommunikation dienen. Angesichts der Tatsache, dass es zukünftig noch stärker darum gehen wird, nicht nur Anwendungen und Systeme, sondern auch gewachsene Integrationsinfrastrukturen zu integrieren, wird die Bedeutung derartiger Meta-Standards deutlich. Das Ziel lautet also ganzheitliche Integration gestützt auf Standards.

Mit Blick auf das Ziel einer SOA ist der ESB als ein geeignetes Fundament anzusehen, weshalb bei einer weiter steigenden Verbreitung von SOA auch mit einer zunehmenden Verbreitung von ESB-Implementierungen zu rechnen sein wird. Dennoch gibt es offene Problemstellungen. So ist zu fragen, in wieweit ein ESB zukünftig noch stärker als heute die Integration kommerzieller Services unterstützen wird. Damit einher geht, ein noch größeres Augenmerk gerade auf die unternehmensübergreifende Integration zu legen. Am Horizont steht hierbei das Ziel, dynamisch Verträge und Zahlungsmodelle bezüglich Services zu vereinbaren und genauso dynamisch und flexibel Services im Rahmen eines ESB zu integrieren, zu nutzen oder gegen andere auszutauschen. Das macht jedoch Konzepte erforderlich, die über heute schon verfügbare Dienstgütevereinbarungen (SLA) oder UDDI-basierte Verzeichnisdienste hinausgehen. Nicht zuletzt wird es damit auch am ESB liegen, in wieweit sich der heutige Servicemarktplatz im Vergleich zum eher erfolglosen Komponentenmarktplatz durchsetzen kann.

Anhang A

Transformation von Schnittstellenbeschreibungen

```
interface HelloWorld {
   string sayHi ();
   string greetMe (in string user);
};
```

Listing A.1: IDL-Datei (Quelle: [LS05], S. 6.).

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:corba="http://schemas.iona.com/bindings/corba"
   xmlns:corbatm="http://www.iona.com/cdr_over_iiop"
   xmlns:references="http://schemas.iona.com/references"
   xmlns:tns="http://www.iona.com/cdr_over_iiop"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.iona.com/cdr_over_iiop"
   targetNamespace="http://www.iona.com/cdr_over_iiop"
   name="cdr_over_iiop">
   <types>
      <schema targetNamespace="http://www.iona.com/cdr_over_iiop"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
         <element name="HelloWorld.sayHi.return"
            type="xsd:string"/>
         <element name="HelloWorld.greetMe.user"
            type="xsd:string"/>
         <element name="HelloWorld.greetMe.return"
            type="xsd:string"/>
      </schema>
   </types>
   <message name="HelloWorld.sayHi"/>
   <message name="HelloWorld.sayHiResponse">
      <part name="return"
         element="corbatm:HelloWorld.sayHi.return"/>
   </message>
   <message name="HelloWorld.greetMe">
      <part name="user" element="corbatm:HelloWorld.greetMe.user"/>
   </message>
   <message name="HelloWorld.greetMeResponse">
      <part name="return"
         element="corbatm:HelloWorld.greetMe.return"/>
   </message>
   <portType name="HelloWorld">
      <operation name="sayHi">
         <input name="sayHi" message="corbatm:HelloWorld.sayHi"/>
         <output name="sayHiResponse"
            message="corbatm:HelloWorld.sayHiResponse"/>
      </operation>
      <operation name="greetMe">
         <input name="greetMe"
            message="corbatm:HelloWorld.greetMe"/>
```

```
        <output name="greetMeResponse"
           message="corbatm:HelloWorld.greetMeResponse"/>
     </operation>
  </portType>
  <binding name="HelloWorldBinding" type="corbatm:HelloWorld">
     <corba:binding repositoryID="IDL:HelloWorld:1.0"/>
     <operation name="sayHi">
        <input/>
        <output/>
     </operation>
     <operation name="greetMe">
        <input/>
        <output/>
     </operation>
  </binding>
  <service name="HelloWorldService">
     <port name="HelloWorldPort"
        binding="corbatm:HelloWorldBinding">
        <corba::address
           location="file:../../hello_world_service.ior"/>
     </port>
  </service>
</definitions>
```

Listing A.2: WSDL-Datei (Quelle: [LS05], S. 6.).

Anhang B

Standards

Standard	**Beschreibung**
BPEL4WS (neu WS-BPEL)	Die Business Process Execution Language (BPEL) for Web Services bzw. Web Services Business Process Execution Language ermöglicht die Beschreibung und Umsetzung von Geschäftsprozessen auf Grundlage von Web Services und BPEL im Sinne einer Service Orchestrierung. Sie stammt von der Web Services Flow Language (WSFL) und XLANG ab.
WS-Addressing	WS-Addressing ist eine Spezifikation zur Beschreibung und Identifikation von Endpunkten und ist für andere WS-Standards unverzichtbar.
WS-Agreement	WS-Agreement erlaubt die Definition von Vereinbarungen zu Dienstgüten zwischen zwei Parteien.
WS-Atomic Transaction	WS-Atomic Transaction gewährleistet eine Transaktionssicherheit auf dem Prinzip ganz oder gar nicht. Durch Sperrung von Ressourcen sollten damit jedoch nur kleine also atomare Transaktionen durchgeführt werden.
WS-BusinessActivity	Zielstellung von WS-BusinessActivity ist die Gewährleistung einer Transaktionssicherheit von lang anhaltenden Transaktionen.
WS-Coordination	WS-Coordination liefert Koordinationsprotokolle bezüglich der Konversation zwischen Web Services.
WS-Eventing	Mit WS-Eventing soll eine asynchrone, über Benachrichtigungen (Events) gesteuerte Kommunikation für Web Services ermöglicht werden.
WS-Federation	WS-Federation soll die Föderation, d. h. den Verbund bzw. die Vereinigung von mehreren Web Services über unterschiedliche (Sicherheits-)Domänen hinweg ermöglichen.

WS-Inspection/ WSIL	WS-Inspection/Web Service Inspection Language dient der Beschreibung von registrierten Web Services über XML-Dokumente.
WS-Notification	WS-Notification gestattet die Übertragung von Nachrichten gemäß Publish-and-Subscribe für Web Services.
WS-Policy	Mit WS-Policy können Anforderungen, Fähigkeiten und Zusicherungen an einen Web Service beschrieben werden.
WS-Reliability	WS-Reliability soll gewährleisten, Nachrichten zuverlässig (garantierte Zustellung, kein doppeltes Versenden und geordnete Reihenfolge) zu übertragen.
WS-ReliableMessaging	WS-ReliableMessaging soll in gleicher Weise wie WS-Reliability einen zuverlässigen Nachrichtenaustausch gewährleisten.
WS-SecureConversation	Bei WS-SecureConversation geht es um den Aufbau eines sicheren Nachrichtenkontextes, innerhalb dessen mehrere Nachrichten ausgetauscht werden können ohne diese jeweils einzeln verschlüsseln zu müssen.
WS-Security	WS-Security ist ein die Sicherheit von Web Services betreffender Standard und stützt sich auf XML Encryption und XML Signature.
WS-Transaction	WS-Transaction wurde abgelöst durch WS-Atomic Transaction und WS-BusinessActivity.
WS-Trust	WS-Trust baut auf WS-Security auf und fokussiert sich auf den Austausch und die Etablierung von Vertrauensbeziehungen (im Sinne einer Autorisation) mittels so genannter Sicherheitstoken.
WSDL	Die Web Services Description Language ist ein XML-basierter Standard zur Beschreibung von Web Services bzw. deren Schnittstellen.
WSLA	Web Service Level Agreement gestattet die Festlegung von Dienstgütevereinbarungen (SLA) für Web Services. Dazu gehört die Definition, Umsetzung und Überwachung von Dienstgütevereinbarungen.

Tabelle B.1: Web Service Standards (WS-Standards).

Standard	**Beschreibung**
cXML	Commerce XML ist ein Standard aus dem Bereich B2B und dient vor allem dem Austausch von geschäftsrelevanten Dokumenten im Umfeld von Beschaffung und Zulieferung.
DTD	Die Document Type Definition ist eine Deklaration innerhalb von SGML- und XML-Dokumenten über deren Struktur und Aufbau. In Folge einer steigenden Verbreitung von XML Schema oder auch RELAX NG nimmt die Bedeutung von DTD ab, zumal die DTD selbst XML-basiert aufgebaut ist.

ebXML	Electronic Business XML stellt eine Familie von Standards dar, die den technischen Rahmen zur Nutzung von XML für elektronische Geschäftsprozesse bilden soll.
RELAX NG	Die REgular LAnguage for XML Next Generation ist eine Schemasprache für XML, die in ihrem Funktionsumfang zur WSDL sehr ähnlich ist, jedoch eine nicht ganz so weite Verbreitung vorweisen kann. RELAX NG selbst ist wie WSDL auch XML-basiert.
RosettaNet	RosettaNet ist ein Sammlung von Standards aus dem B2B-Umfeld, die bspw. die Unterstützung unternehmensübergreifender Geschäftsprozesse anstrebt.
SAML	Die Security Assertion Markup Language ist eine XML-basierte Auszeichnungssprache zur Authentifizierung und Autorisation.
UBL	Die Universal Business Language ist eine auf ebXML und damit auch auf XML gestützte Spezifikation zur Beschreibung von elektronischen Geschäftsdokumenten.
XACML	Die eXtensible Access Control Markup Language ist eine XML-basierte Beschreibungssprache für Politiken zur Zugangskontrolle (Autorisation).
xCBL	Die XML Common Business Library ist eine Sammlung von XML-Spezifikationen im Umfeld von B2B und basiert zu großen Teilen auf EDI-Standards.
XML Encryption	XML Encryption ist eine Spezifikation zur Verschlüsselung von XML-Dokumenten. Es wird sowohl die Verschlüsselung des gesamten Dokuments, einzelner Elemente als auch einzelner Werte ermöglicht.
XML Schema	XML Schema ist ein Empfehlung des W3C bezüglich der Definition von XML-Dokumenten bzw. XML-Dokumentenstrukturen. Im Gegensatz zur Document Type Definition (DTD) ist XML Schema selbst auch XML-basiert.
XML Signature	XML Signature ist eine Spezifikation bezüglich der Signatur von XML-Dokumenten.
XMPP	Das Extensible Messaging and Presence Protocol ist ein auf XML gestützter Standard für Instant Messaging.
XPath	Die XML Path Language ist eine vom W3C entwickelte Anfragesprache, um Teile eines XML-Dokuments anzusprechen.
XQuery	Die XML Query Language ist eine vom W3C entwickelte Anfragesprache für Sammlungen von XML-Daten und lehnt sich an die Structured Query Language (SQL) an. XQuery selbst ist nicht XML-konform.
XSLT	Extensible Stylesheet Language (XSL) Transformations ist eine XML-basierte Sprache, die zur Transformation von XML-Dokumenten genutzt werden kann.

Tabelle B.2: XML-basierte Standards.

Standard	Beschreibung
FTP	Das File Transfer Protocol ist ein Netzwerkprotokoll zur Dateiübertragung speziell in TCP/IP-Netzwerken.
HTTP	Das Hypertext Transfer Protocol ist ein auf TCP/IP aufsetzendes Protokoll zum Austausch von Daten (Webseiten) über ein Netzwerk (typischerweise Internet).
HTTPS	Das Hypertext Transfer Protocol Secure ist die Erweiterung von HTTP um eine zusätzliche Sicherheitsschicht (zur Authentifizierung und Verschlüsselung). Realisiert wird diese Schicht in der Regel über SSL.
IIOP	Das Internet Inter ORB Protocol ist ein Protokoll bezüglich eines entfernten Methodenaufrufs bzw. entfernten Objektaustauschs im Umfeld von CORBA.
SMTP	Das Simple Mail Transfer Protocol in ein Protokoll zum Austausch von elektronischen Nachrichten (E-Mails) innerhalb eines Netzwerks.
SNMP	Das Simple Network Management Protocol dient zur Überwachung und Steuerung von Elementen innerhalb eines Netzwerks.
SOAP	Das Simple Object Access Protocol ist ein netzwerkbasiertes Protokoll, das typischerweise zum Austausch von XML-basierten Daten/Nachrichten verwendet wird. SOAP bildet dabei die Grundlage für den Aufbau weiter reichender Kommunikationsinfrastrukturen. Im Umfeld von Web Services stellt HTTP bzw. HTTPS ein typisches Basisprotokoll für SOAP dar.
SSL/TLS	Der Secure Sockets Layer bzw. dessen Nachfolger, die Transport Layer Security, sind ein Verschlüsselungsprotokoll für Datenübertragungen in Netzwerken (hauptsächlich im Internet).

Tabelle B.3: Protokoll Standards.

Standard	Beschreibung
EJB	Enterprise JavaBeans bilden das zentrale Komponentenmodell der Java 2 Enterprise Edition (J2EE).
JAAS	Der Java Authentication and Authorization Service besteht aus einer Sammlung von API, die Services gestattet sich zu authentifizieren und Rechtevergaben umzusetzen.
JBI	Die Java Business Integration ist, wie bereits ausführlich in Kapitel 3.10 beschrieben, eine Spezifikation über den Aufbau von ESB-Service-Containern.

JCA	Die J2EE Connector Architecture ist eine Java-basierte Programmierschnittstelle (API – Application Programming Interface) zur Einbindung von Ressourcenadaptern in die Kommunikation mit externen Anwendungen und Systemen.
JDBC	Die Java Database Connectivity ist eine Java-basierte Programmierschnittstelle (API) zur Anbindung von Datenbanken.
JMS	Der Java Message Service ist eine Java-basierte Programmierschnittstelle (API – Application Programming Interface) zur Unterstützung von MOM.
JMX	Die Java Management Extensions ist eine Spezifikation bezüglich der Überwachung und des Managements von Java-Anwendungen.
JNDI	Java Naming and Directory Interface ist eine Programmierschnittstelle für Namens- und Verzeichnisdienste.
JSSE	Die Java Secure Socket Extension ist eine Sammlung von Paketen zur Umsetzung von SSL und TLS mittels Java.
RMI	Die Remote Method Invocation bezeichnet den entfernten Zugriff auf Methoden aus einer Java-Umgebung und dient dem Austausch von Objekten. Sie kann folglich auch als objektorientierte Middleware angesehen werden.

Tabelle B.4: Java-basierte Standards.

Standard	**Beschreibung**
.NET	.Net ist ein neueres Framework von Microsoft, das Programmierern umfangreiche vorgefertigte Bibliotheken bereithält und ältere Konzepte wie COM/DCOM ablösen soll. Unterstützt werden dabei unterschiedliche Microsoft-spezifische Programmiersprachen.
ActiveX/OLE	ActiveX oder ehemals Object Linking and Embedding ist ein von Microsoft entwickeltes Komponentenmodell, das vor allem bei Internetanwendungen verbreitet ist.
BEA MessageQ	MOM-Plattform von BEA.
COM	Das Component Object Model ist eine von Microsoft entwickelte Komponentenplattform/-framework.
DCOM	Das Distributed Component Object Model ist ein auf COM aufsetzendes Protokoll, mit dem in einem Netzwerk verteilte Komponenten miteinander kommunizieren können.
FioranoMQ	MOM-Plattform von Fiorano.
MSMQ	MOM-Plattform von Microsoft.
Oracle AQ	MOM-Plattform von Oracle.

SonicMQ	MOM-Plattform von Sonic Software.
TIBCO Rendezvous	TIBCO Rendezvous ist eine MOM-Plattform aufbauend auf einem Nachrichtenbus (The Information Bus – TIB).
Websphere MQ	MOM-Plattform von IBM.

Tabelle B.5: Proprietäre Standards.

Standard	**Beschreibung**
COBOL Copybook	COBOL Copybooks dienen zur Beschreibung von Datenformaten in COBOL.
CORBA	Die Common Object Request Broker Architecture ist eine von der Object Management Group (OMG) entwickelte objektorientierte Middleware.
CSV	Comma Separated Values bezeichnet ein Datenformat basierend auf Textdateien. Die einzelnen Werte sind dabei durch Kommata, Semikola, Doppelpunkte oder sonstige Zeichen voneinander getrennt (daher wird teilweise auch von Character Separated Values gesprochen).
EDI	Electronic Data Interchange (wörtlich elektronischer Datenaustausch) bezeichnet ein Klasse von Formaten und Verfahren zum Austausch strukturierter Nachrichten. (UN/)EDIFACT (United Nations/Electronic Data Interchange For Administration, Commerce and Transport) ist hierfür ein typischer Vertreter, der speziell im Geschäftsverkehr verwendet wird.
IDL	Die Interface Definition Language dient im Kontext von CORBA als Sprache zur Schnittstellenbeschreibung.
UDDI	Universal Description, Discovery and Integration ist ein Verzeichnisdienst zum Registrieren von Web Services.

Tabelle B.6: Sonstige Standards.

Grundlegende Standards $\boldsymbol{ST}_G$	**Aufbauende Standards** $\boldsymbol{ST}_A$	**Fortgeschrittene Standards** $\boldsymbol{ST}_F$	**Ergänzende Standards** $\boldsymbol{ST}_E$
DTD	FTP	.NET	ActiveX/OLE
HTTP	HTTPS	BPEL4WS (neu WS-BPEL)	BEA MessageQ
SOAP	JMS	CORBA	COBOL Copybook
UDDI	SMTP	CSV	COM
WSDL	SSL/TLS	cXML	DCOM
XML	WS-Addressing	ebXML	EJB
XML Schema	WS-Atomic Transaction	EDI/(UN/)EDIFACT	FioranoMQ
	WS-BusinessActivity	IDL	JAAS
	WS-Coordination	IIOP	JBI
	WS-Notification	JMX	JCA
	WS-Reliability	RosettaNet	JDBC
	WS-ReliableMessaging	SAML	JNDI
	WS-Security	UBL	JSSE
	WS-Transaction	WS-Agreement	MSMQ
	WSLA	WS-Eventing	Oracle AQ
	XPath	WS-Federation	RELAX NG
	XQuery	WS-Inspection/WSIL	RMI
	XSLT	WS-Policy	SMNP
		WS-SecureConversation	SonicMQ
		WS-Trust	SonicMQ
		XACML	TIBCO Rendezvous
		xCBL	Websphere MQ
		XML Encryption	XMPP
		XML Signature	

Tabelle B.7: Standardkategorien.

Anhang C

ESB-Anbieter

ESB-Anbieter	**ESB-Produkte**
BEA Systems, Inc.	BEA AquaLogic Service Bus
Boston, Corp.	ChainBuilder ESB
Cape Clear Software, Inc.	Cape Clear ESB
Fiorano Software, Inc.	Fiorano ESB
IBM, Corp.	WebSphere Enterprise Service Bus
Iona Technologies	Artix, Celtix
JBoss, a divison of Red Hat, Inc.	JBoss ESB
Logic Blaze, Inc.	FUSE
Microsoft, Corp.	BizTalk Server 2006
MuleSource, Inc.	Mule
ObjectWeb Consortium	PEtALS
Oracle, Corp.	Oracle Enterprise Service Bus
PolarLake Ltd.	PolarLake Integration Suite
Progress Software, Corp. (Sonic Software, Corp.)	Sonic ESB
SeeBeyond Technology, Corp.	eInsight Enterprise Service Bus
Software AG	Enterprise Service Integrator
SpiritSoft	SpiritWave Real-Time ESB
Sun Microsystems, Inc.	Sun Java ESB Suite

The Apache Software Foundation	ServiceMix
TIBCO Software, Inc.	TIBCO BusinessWorks
webMethods, Inc.	webMethods Fabric

Tabelle C.1: ESB-Anbieter und -Produkte.

ESB-Anbieter	**Marktkapitalisierung (zum 23.11.2006)**	**Jahresumsatz (letztes Geschäftsjahr)**
Microsoft, Corp.	294,13 Mrd. US-Dollar	45,35 Mrd. US-Dollar
IBM, Corp.	140,87 Mrd. US-Dollar	89,59 Mrd. US-Dollar
Oracle, Corp.	102,14 Mrd US-Dollar	15,20 Mrd.US-Dollar
Sun Microsystems, Inc.	19,67 Mrd. US-Dollar	13,53 Mrd. US-Dollar
BEA Systems, Inc.	5,49 Mrd.US-Dollar	1,3 Mrd.US-Dollar
Red Hat, Inc.	3,36 Mrd. US-Dollar	335,51 Mio. US-Dollar
Software AG	1,53 Mrd. Euro (ca. 1,98 Mrd. US-Dollar)	438,03 Mio. Euro (ca. 576,25 Mio US-Dollar)
TIBCO Software, Inc.	1,96 Mrd.US-Dollar	490,66 Mio US-Dollar
Progress Software, Corp.	1,17 Mrd.US-Dollar	411,57 Mio. US-Dollar
webMethods, Inc.	410,70 Mio US-Dollar	209,71 Mio US-Dollar
Iona Technologies	200,06 Mio US-Dollar	74,14 Mio US-Doallar

Tabelle C.2: ESB-Anbieter Unternehmensdaten (vgl. [OnV06]; [Yah06]).

Anhang D

Marktstudien nach Forrester

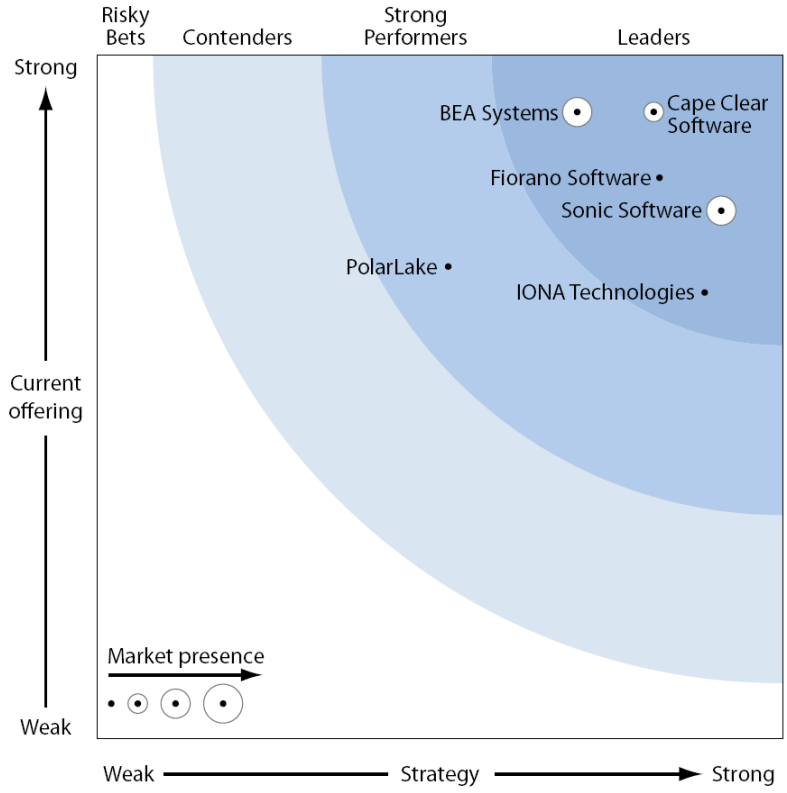

Abbildung D.1: Forrester Wave: Enterprise Service Bus, ESB Suites, Q4 '05 (Quelle: [VG05], S. 8).

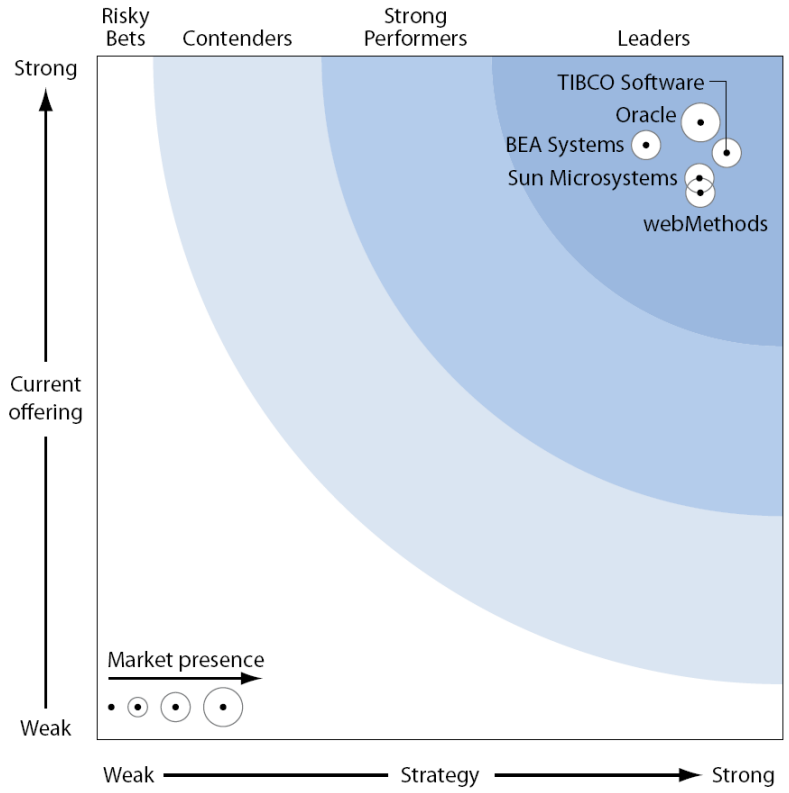

Abbildung D.2: Forrester Wave: Enterprise Service Bus, Comprehensive ESB Suites, Q4 '05 (Quelle: [VG05], S. 11).

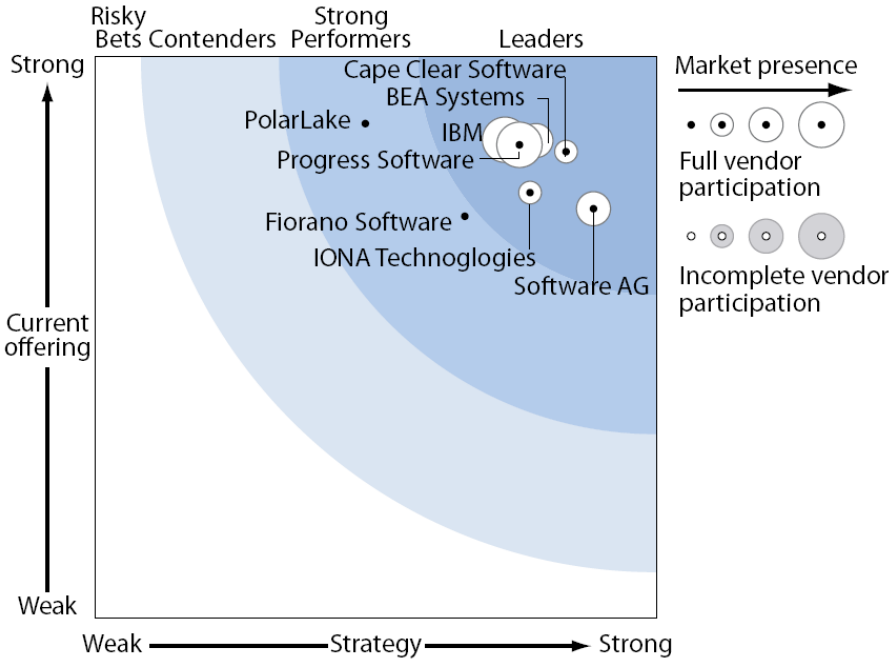

Abbildung D.3: Forrester Wave: Enterprise Service Bus, Q2 '06 (Quelle: [VG05], S. 9).

Anhang E

Event Stream Processing

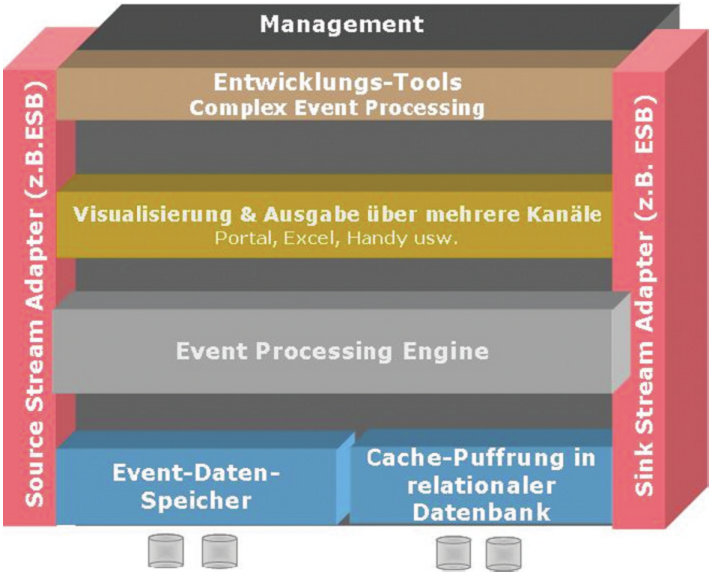

Abbildung E.1: Aufbau und Bestandteile einer ESP-Engine (Quelle: [Pal05], S. 3).

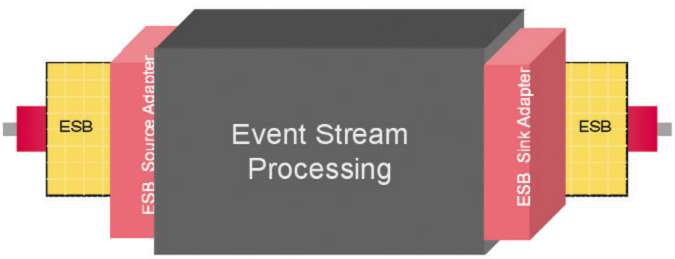

Abbildung E.2: Einbettung einer ESP-Engine (Quelle: [Pal05], S. 4).

Anhang F

Anwendung des ESB-Qualitäts-modells

Hardware	Client	Server
Prozessor (CPU)	AMD Duron, 1300 MHz	AMD Duron, 1300 MHz
Arbeitsspeicher (RAM)	1024 MB, PC-2100 DDR-SDRAM	1024 MB, PC-2100 DDR-SDRAM
Netzwerk	LAN, 100 Mbit/s	LAN, 100 Mbit/s
Betriebssystem	Windows XP – Professional Edition	Windows XP – Professional Edition
Java Version	J2SE Runtime Environment Version 5.0 (1.5.0_09)	J2SE Runtime Environment Version 5.0 (1.5.0_09)

Tabelle F.1: Plattformeigenschaften des gewählten Szenarios.

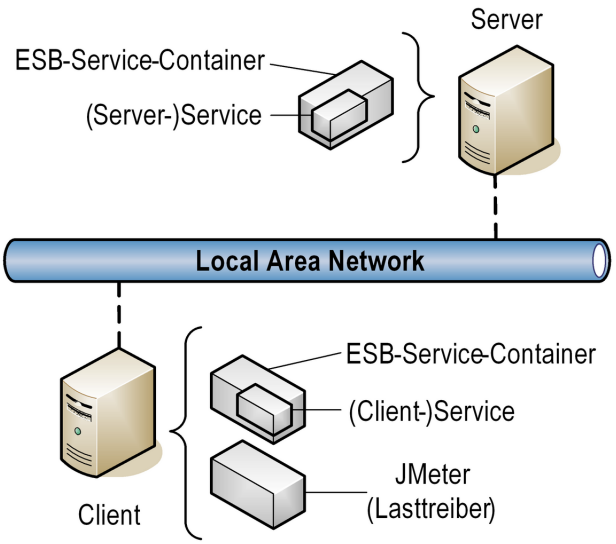

Abbildung F.1: Deployment-Topologie des gewählten Szenarios.

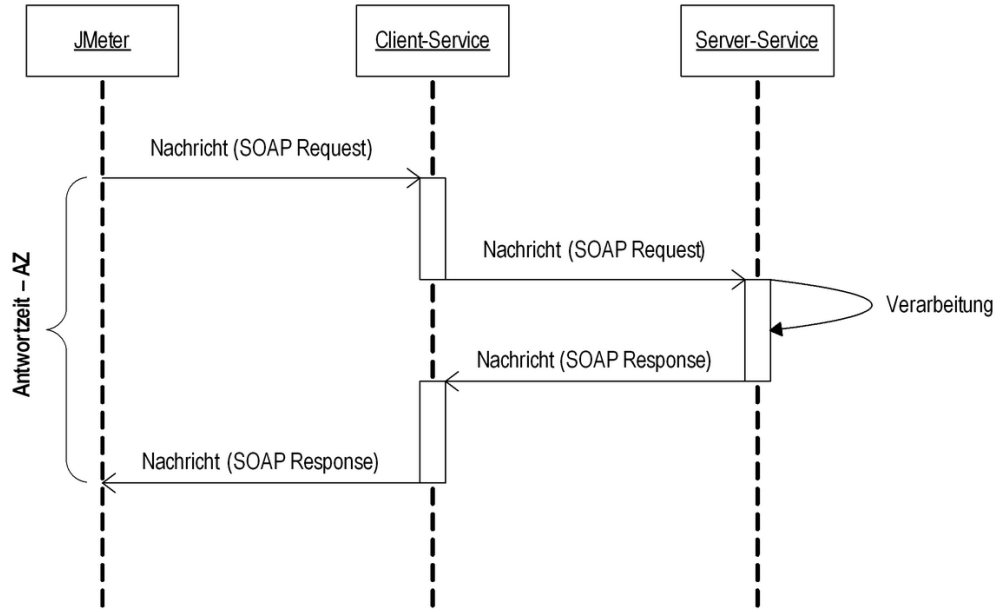

Abbildung F.2: Interaktion eines Aufrufs.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ESBDateTime"
   targetNamespace="http://localhost/esb_datetime"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://localhost/esb_datetime"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:types>
      <schema targetNamespace="http://localhost/esb_datetime"
         xmlns="http://www.w3.org/2001/XMLSchema">
         <element name="responseType" type="xsd:dateTime"/>
         <element name="requestType" type="xsd:string"/>
      </schema>
   </wsdl:types>
   <wsdl:message name="getESBDateTimeRequest">
      <wsdl:part element="tns:requestType" name="theRequest"/>
   </wsdl:message>
   <wsdl:message name="getESBDateTimeResponse">
      <wsdl:part element="tns:responseType" name="theResponse"/>
   </wsdl:message>
   <wsdl:portType name="ESB_PortType">
      <wsdl:operation name="getESBDateTime">
         <wsdl:input message="tns:getESBDateTimeRequest"
            name="getESBDateTimeRequest"/>
         <wsdl:output message="tns:getESBDateTimeResponse"
            name="getESBDateTimeResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="ESB_Binding" type="tns:ESB_PortType">
      <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getESBDateTime">
         <soap:operation soapAction="" style="document"/>
         <wsdl:input name="getESBDateTimeRequest">
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="getESBDateTimeResponse">
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="ESBDateTimeService">
      <wsdl:port binding="tns:ESB_Binding" name="SoapPort">
         <soap:address location="http://localhost:19000"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Listing F.1: WSDL-Schnittstellenbeschreibung.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:m1="http://localhost/esb_datetime"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <m1:requestType> </m1:requestType>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing F.2: SOAP-Nachricht (Anfrage).

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:m1="http://localhost/esb_datetime"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <m1:responseType>2006-12-30T11:04:59.687+01:00</m1:responseType>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing F.3: SOAP-Nachricht (Antwort).

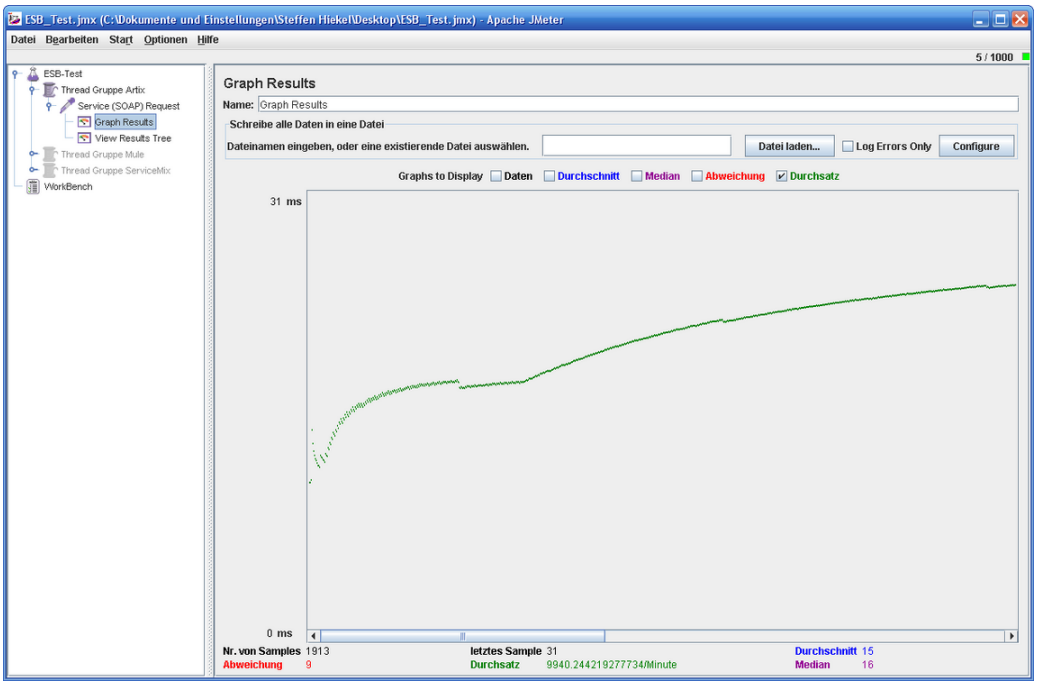

Abbildung F.3: Screenshot von JMeter.

```
<jmeterTestPlan version="1.2" properties="1.8">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="ESB-Test"
      enabled="true">
      <elementProp name="TestPlan.user_defined_variables"
         elementType="Arguments" guiclass="ArgumentsPanel" testclass="Arguments"
         testname="Benutzer definierte Variablen" enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
      <boolProp name="TestPlan.serialize_threadgroups">true</boolProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <stringProp name="TestPlan.comments"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
         testname="Thread Gruppe Artix" enabled="false">
        <longProp name="ThreadGroup.start_time">1166447600000</longProp>
        <stringProp name="ThreadGroup.delay"></stringProp>
        <stringProp name="ThreadGroup.duration"></stringProp>
        <stringProp name="ThreadGroup.num_threads">1000</stringProp>
        <boolProp name="ThreadGroup.scheduler">false</boolProp>
        <elementProp name="ThreadGroup.main_controller"
         elementType="LoopController" guiclass="LoopControlPanel"
         testclass="LoopController" testname="Wiederholungs Kontroller"
         enabled="true">
          <intProp name="LoopController.loops">-1</intProp>
          <boolProp name="LoopController.continue_forever">false</boolProp>
        </elementProp>
        <longProp name="ThreadGroup.end_time">1166447600000</longProp>
        <stringProp name="ThreadGroup.on_sample_error">stopthread</stringProp>
        <stringProp name="ThreadGroup.ramp_time">2400</stringProp>
      </ThreadGroup>
      <hashTree>
        <WebServiceSampler guiclass="WebServiceSamplerGui"
         testclass="WebServiceSampler" testname="Service (SOAP) Request"
         enabled="true">
          <stringProp name="HTTPSampler.path">/ESBService</stringProp>
          <stringProp name="HTTPSampler.method">POST</stringProp>
          <stringProp name="WebServiceSampler.proxy_host"></stringProp>
          <stringProp name="HTTPSampler.protocol">http</stringProp>
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments">
```

```
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="WebServiceSampler.proxy_port"></stringProp>
      <stringProp name="HTTPSampler.port">19001</stringProp>
      <stringProp name="WebServiceSampler.use_proxy">false</stringProp>
      <stringProp name="Soap.Action"></stringProp>
      <stringProp name="WebServiceSampler.xml_data_file"></stringProp>
      <stringProp name="WebServiceSampler.read_response">true</stringProp>
      <stringProp name="WebServiceSampler.xml_path_loc"></stringProp>
      <stringProp name="WebserviceSampler.wsdl_url">
        http://localhost:19001/</stringProp>
      <stringProp name="WebServiceSampler.memory_cache">true</stringProp>
      <stringProp name="HTTPSampler.domain">localhost</stringProp>
      <stringProp name="HTTPSamper.xml_data">
        &lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;
          standalone=&quot;no&quot;?&gt;
        &lt;SOAP-ENV:Envelope
          xmlns:xs=&quot;http://www.w3.org/2001/XMLSchema&quot;
          xmlns:m1=&quot;http://localhost/esb_datetime&quot;
          xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
          xmlns:SOAP-ENV=
            &quot;http://schemas.xmlsoap.org/soap/envelope/&quot;&gt;
        &lt;SOAP-ENV:Body&gt;
        &lt;m1:requestType&gt; &lt;/m1:requestType&gt;
        &lt;/SOAP-ENV:Body&gt;
        &lt;/SOAP-ENV:Envelope&gt;
      </stringProp>
    </WebServiceSampler>
    <hashTree>
      <ResultCollector guiclass="GraphVisualizer"
        testclass="ResultCollector" testname="Graph Results"
        enabled="true">
        <objProp>
          <value class="SampleSaveConfiguration">
            <time>true</time>
            <latency>true</latency>
            <timestamp>true</timestamp>
            <success>true</success>
            <label>true</label>
            <code>true</code>
            <message>true</message>
            <threadName>true</threadName>
            <dataType>true</dataType>
            <encoding>false</encoding>
            <assertions>true</assertions>
            <subresults>true</subresults>
            <responseData>false</responseData>
            <samplerData>false</samplerData>
            <xml>true</xml>
            <fieldNames>false</fieldNames>
            <responseHeaders>false</responseHeaders>
            <requestHeaders>false</requestHeaders>

            <responseDataOnError>false</responseDataOnError>
            <saveAssertionResultsFailureMessage>false
                 </saveAssertionResultsFailureMessage>
            <assertionsResultsToSave>0</assertionsResultsToSave>
          </value>
          <name>saveConfig</name>
        </objProp>
        <stringProp name="filename"></stringProp>
        <boolProp name="ResultCollector.error_logging">false</boolProp>
      </ResultCollector>
      <hashTree/>
    </hashTree>
  </hashTree>
  <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
     testname="Thread Gruppe Mule" enabled="false">
    <longProp name="ThreadGroup.start_time">1166447600000</longProp>
    <stringProp name="ThreadGroup.delay"></stringProp>
    <stringProp name="ThreadGroup.duration"></stringProp>
    <stringProp name="ThreadGroup.num_threads">1000</stringProp>
    <boolProp name="ThreadGroup.scheduler">false</boolProp>
    <elementProp name="ThreadGroup.main_controller"
     elementType="LoopController" guiclass="LoopControlPanel"
     testclass="LoopController" testname="Wiederholungs Kontroller"
     enabled="true">
      <intProp name="LoopController.loops">-1</intProp>
      <boolProp name="LoopController.continue_forever">false</boolProp>
    </elementProp>
```

```
  <longProp name="ThreadGroup.end_time">1166447600000</longProp>
  <stringProp name="ThreadGroup.on_sample_error">stopthread</stringProp>
  <stringProp name="ThreadGroup.ramp_time">2400</stringProp>
</ThreadGroup>
<hashTree>
  <WebServiceSampler guiclass="WebServiceSamplerGui"
   testclass="WebServiceSampler" testname="Service (SOAP) Request"
   enabled="true">
    <stringProp name="HTTPSampler.path">/ESBUMO</stringProp>
    <stringProp name="HTTPSampler.method">POST</stringProp>
    <stringProp name="WebServiceSampler.proxy_host"></stringProp>
    <stringProp name="HTTPSampler.protocol">http</stringProp>
    <elementProp name="HTTPsampler.Arguments" elementType="Arguments">
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="WebServiceSampler.proxy_port"></stringProp>
    <stringProp name="HTTPSampler.port">19002</stringProp>
    <stringProp name="WebServiceSampler.use_proxy">false</stringProp>
    <stringProp name="Soap.Action"></stringProp>
    <stringProp name="WebServiceSampler.xml_data_file"></stringProp>
    <stringProp name="WebServiceSampler.read_response">true</stringProp>
    <stringProp name="WebServiceSampler.xml_path_loc"></stringProp>
    <stringProp name="WebserviceSampler.wsdl_url">
      http://localhost:19002/</stringProp>
    <stringProp name="WebServiceSampler.memory_cache">true</stringProp>
    <stringProp name="HTTPSampler.domain">localhost</stringProp>
    <stringProp name="HTTPSamper.xml_data">
      &lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;
        standalone=&quot;no&quot;?&gt;
      &lt;SOAP-ENV:Envelope
        xmlns:xs=&quot;http://www.w3.org/2001/XMLSchema&quot;
        xmlns:m1=&quot;http://localhost/esb_datetime&quot;
        xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
        xmlns:SOAP-ENV=
          &quot;http://schemas.xmlsoap.org/soap/envelope/&quot;&gt;
      &lt;SOAP-ENV:Body&gt;
      &lt;m1:requestType&gt; &lt;/m1:requestType&gt;
      &lt;/SOAP-ENV:Body&gt;
      &lt;/SOAP-ENV:Envelope&gt;
    </stringProp>
  </WebServiceSampler>
  <hashTree>
    <ResultCollector guiclass="GraphVisualizer"
      testclass="ResultCollector" testname="Graph Results"
      enabled="true">
      <objProp>
        <value class="SampleSaveConfiguration">
          <time>true</time>
          <latency>true</latency>
          <timestamp>true</timestamp>
          <success>true</success>
          <label>true</label>
          <code>true</code>
          <message>true</message>
          <threadName>true</threadName>
          <dataType>true</dataType>
          <encoding>false</encoding>
          <assertions>true</assertions>
          <subresults>true</subresults>
          <responseData>false</responseData>
          <samplerData>false</samplerData>
          <xml>true</xml>
          <fieldNames>false</fieldNames>
          <responseHeaders>false</responseHeaders>
          <requestHeaders>false</requestHeaders>
          <responseDataOnError>false</responseDataOnError>
          <saveAssertionResultsFailureMessage>false
            </saveAssertionResultsFailureMessage>
          <assertionsResultsToSave>0</assertionsResultsToSave>
        </value>
        <name>saveConfig</name>
      </objProp>
      <stringProp name="filename"></stringProp>
      <boolProp name="ResultCollector.error_logging">false</boolProp>
    </ResultCollector>
    <hashTree/>
  </hashTree>
</hashTree>
<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
   testname="Thread Gruppe ServiceMix" enabled="false">
```

```
  <longProp name="ThreadGroup.start_time">1167523680000</longProp>
  <stringProp name="ThreadGroup.delay"></stringProp>
  <stringProp name="ThreadGroup.duration"></stringProp>
  <stringProp name="ThreadGroup.num_threads">1000</stringProp>
  <boolProp name="ThreadGroup.scheduler">false</boolProp>
  <elementProp name="ThreadGroup.main_controller"
   elementType="LoopController" guiclass="LoopControlPanel"
   testclass="LoopController" testname="Wiederholungs Kontroller"
   enabled="true">
    <intProp name="LoopController.loops">-1</intProp>
    <boolProp name="LoopController.continue_forever">false</boolProp>
  </elementProp>
  <longProp name="ThreadGroup.end_time">1166407680000</longProp>
  <stringProp name="ThreadGroup.on_sample_error">stopthread</stringProp>
  <stringProp name="ThreadGroup.ramp_time">2400</stringProp>
</ThreadGroup>
<hashTree>
  <WebServiceSampler guiclass="WebServiceSamplerGui"
   testclass="WebServiceSampler" testname="Service (SOAP) Request"
   enabled="true">
    <stringProp name="HTTPSampler.path">/ESBService</stringProp>
    <stringProp name="HTTPSampler.method">POST</stringProp>
    <stringProp name="WebServiceSampler.proxy_host"></stringProp>
    <stringProp name="HTTPSampler.protocol">http</stringProp>
    <elementProp name="HTTPsampler.Arguments" elementType="Arguments">
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <stringProp name="WebServiceSampler.proxy_port"></stringProp>
    <stringProp name="HTTPSampler.port">19003</stringProp>
    <stringProp name="WebServiceSampler.use_proxy">false</stringProp>
    <stringProp name="Soap.Action"></stringProp>
    <stringProp name="WebServiceSampler.xml_data_file"></stringProp>
    <stringProp name="WebServiceSampler.read_response">true</stringProp>
    <stringProp name="WebServiceSampler.xml_path_loc"></stringProp>
    <stringProp name="WebserviceSampler.wsdl_url">
      http://localhost:19003/</stringProp>
    <stringProp name="WebServiceSampler.memory_cache">true</stringProp>
    <stringProp name="HTTPSampler.domain">localhost</stringProp>
    <stringProp name="HTTPSamper.xml_data">
      &lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;
        standalone=&quot;no&quot;?&gt;
      &lt;SOAP-ENV:Envelope
        xmlns:xs=&quot;http://www.w3.org/2001/XMLSchema&quot;
        xmlns:m1=&quot;http://localhost/esb_datetime&quot;
        xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;
        xmlns:SOAP-ENV=
          &quot;http://schemas.xmlsoap.org/soap/envelope/&quot;&gt;
      &lt;SOAP-ENV:Body&gt;
      &lt;m1:requestType&gt; &lt;/m1:requestType&gt;
      &lt;/SOAP-ENV:Body&gt;
      &lt;/SOAP-ENV:Envelope&gt;
    </stringProp>
  </WebServiceSampler>
  <hashTree>
    <ResultCollector guiclass="GraphVisualizer"
      testclass="ResultCollector" testname="Graph Results"
      enabled="true">
      <objProp>
        <value class="SampleSaveConfiguration">
          <time>true</time>
          <latency>true</latency>
          <timestamp>true</timestamp>
          <success>true</success>
          <label>true</label>
          <code>true</code>
          <message>true</message>
          <threadName>true</threadName>
          <dataType>true</dataType>
          <encoding>false</encoding>
          <assertions>true</assertions>
          <subresults>true</subresults>
          <responseData>false</responseData>
          <samplerData>false</samplerData>
          <xml>true</xml>
          <fieldNames>false</fieldNames>
          <responseHeaders>false</responseHeaders>
          <requestHeaders>false</requestHeaders>
          <responseDataOnError>false</responseDataOnError>
          <saveAssertionResultsFailureMessage>false
            </saveAssertionResultsFailureMessage>
```

```
              <assertionsResultsToSave>0</assertionsResultsToSave>
            </value>
            <name>saveConfig</name>
          </objProp>
          <stringProp name="filename"></stringProp>
          <boolProp name="ResultCollector.error_logging">false</boolProp>
        </ResultCollector>
        <hashTree/>
      </hashTree>
    </hashTree>
  </hashTree>
</hashTree>
</jmeterTestPlan>
```

Listing F.4: Testplan des Lasttreibers.

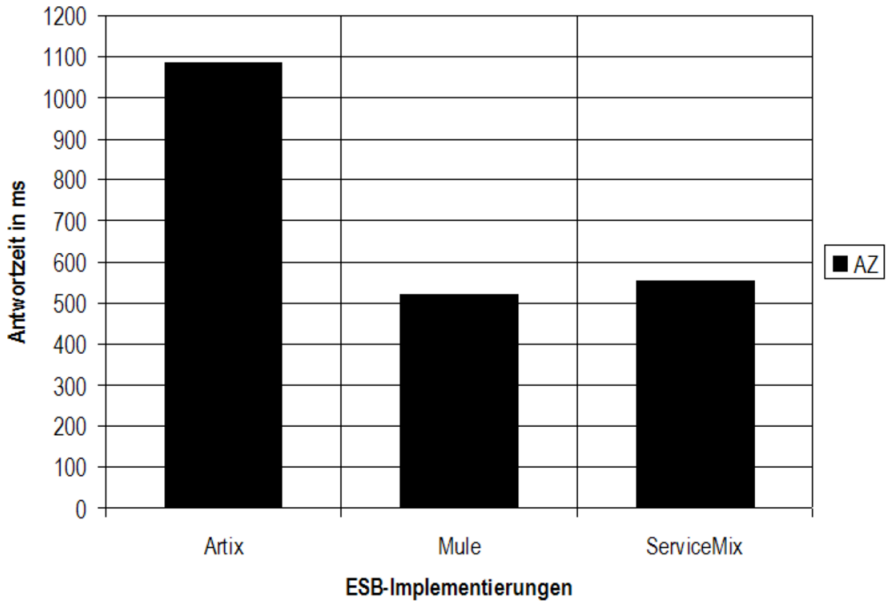

Abbildung F.4: Antwortzeiten (AZ) im Vergleich.

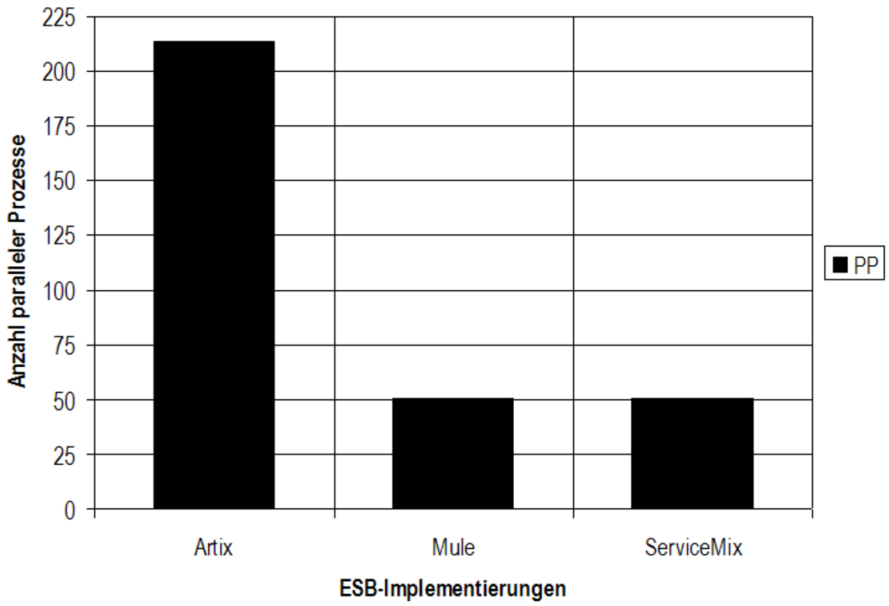

Abbildung F.5: Anzahl pralleler Prozesse (*PP*) im Vergleich.

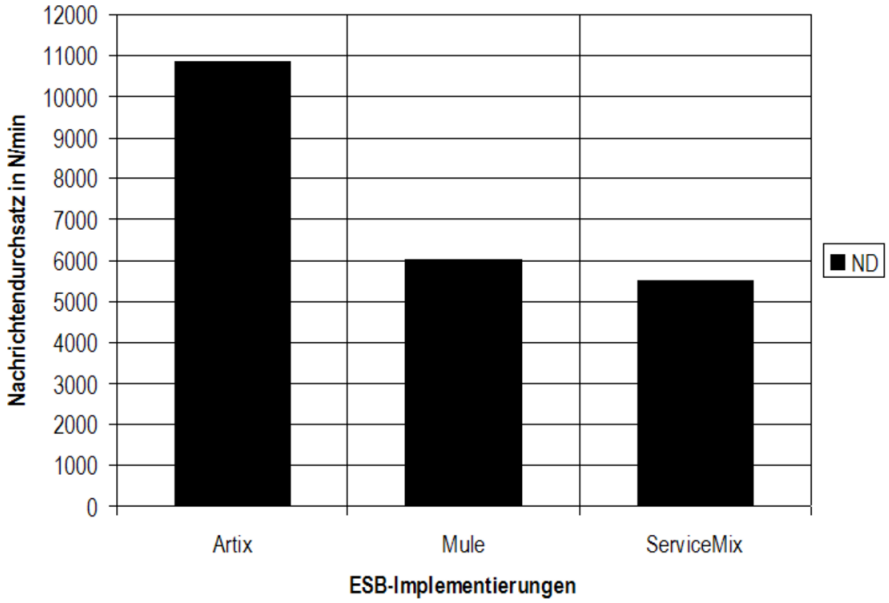

Abbildung F.6: Nachrichtendurchsätze (*ND*) im Vergleich.

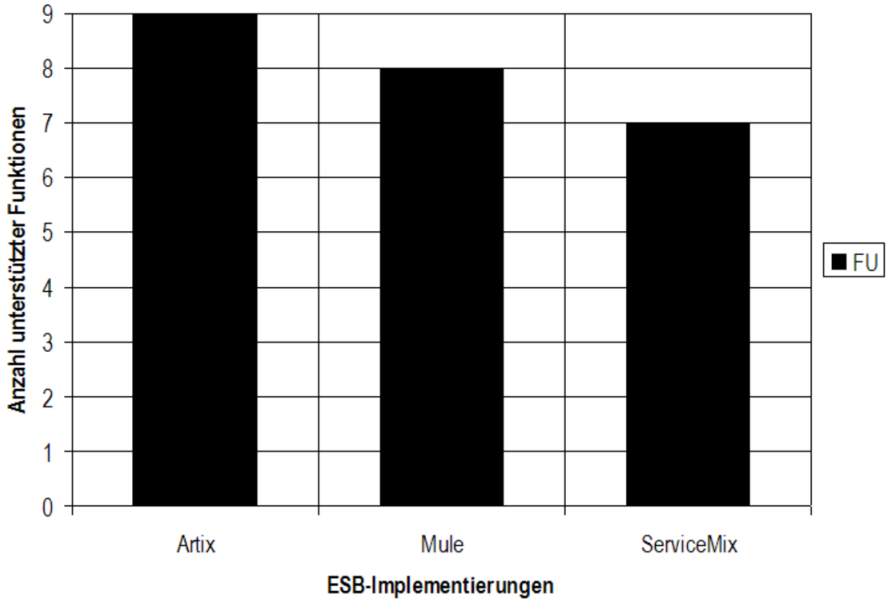

Abbildung F.7: *FU* im Vergleich.

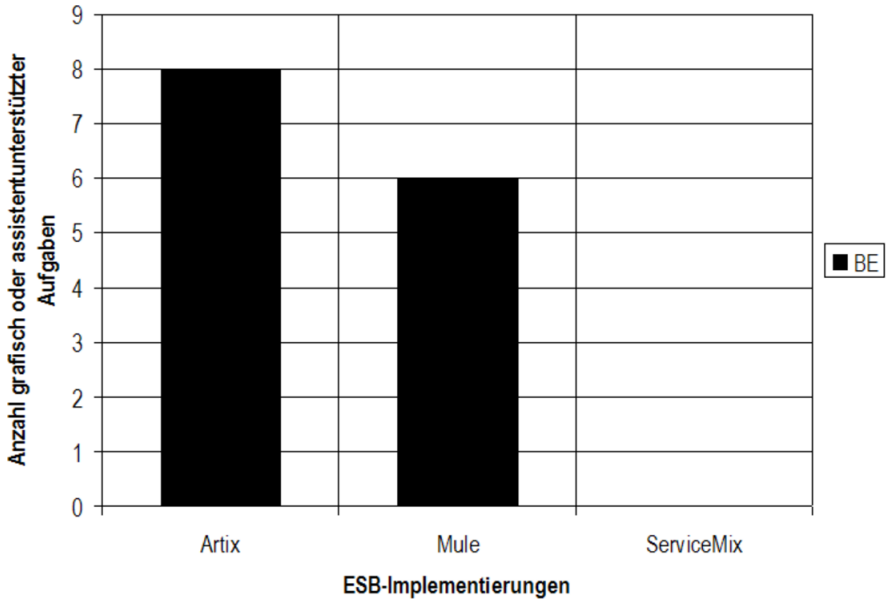

Abbildung F.8: *BE* im Vergleich.

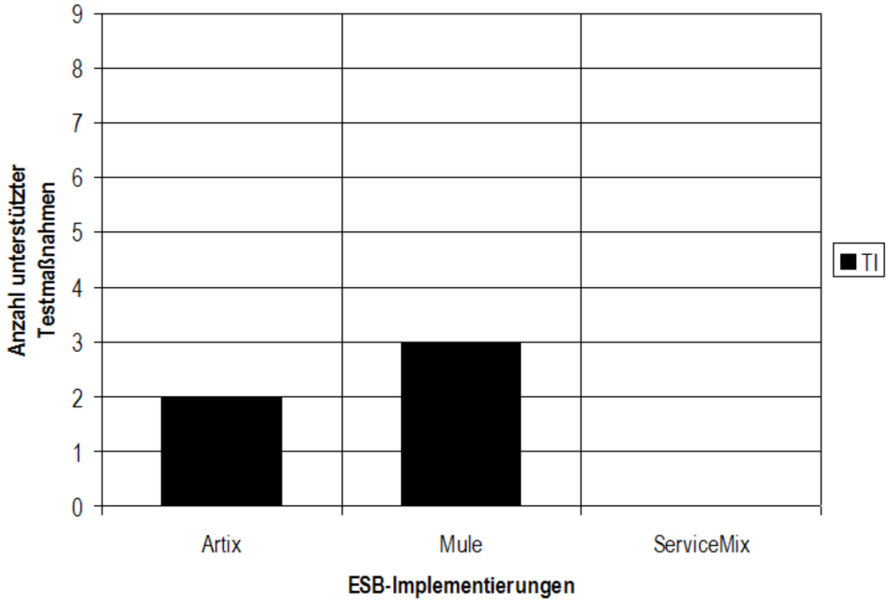

Abbildung F.9: *TI* im Vergleich.

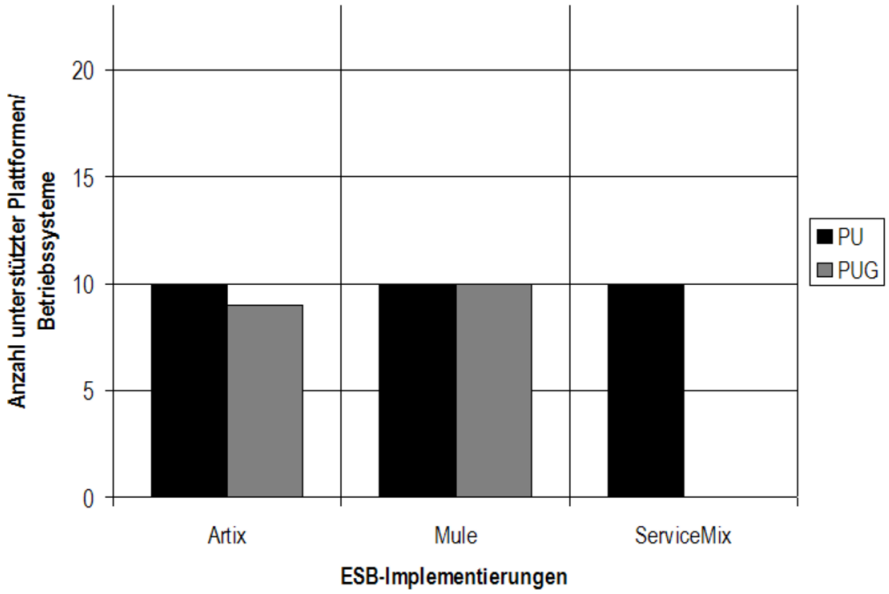

Abbildung F.10: *PU* und *PUG* im Vergleich.

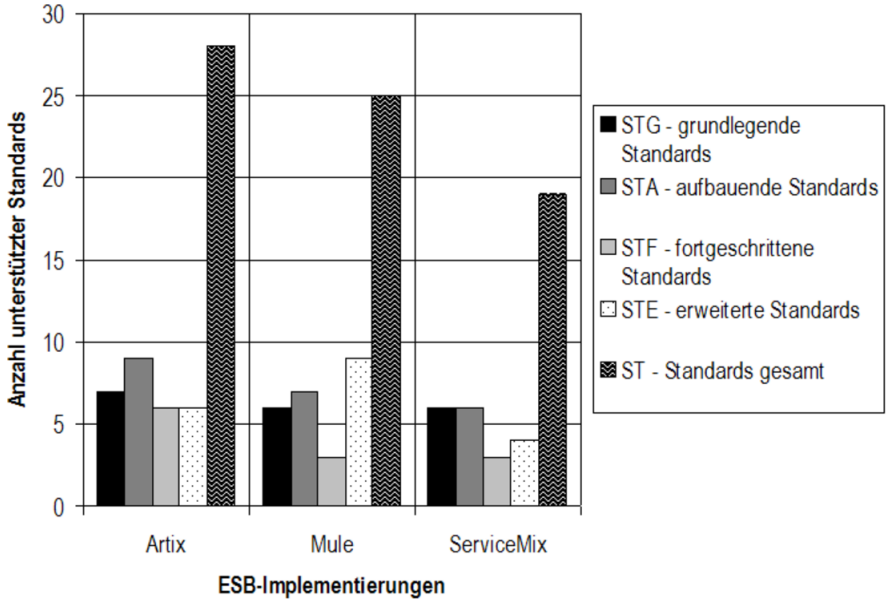

Abbildung F.11: ST_G, ST_A, ST_F, ST_E und ST im Vergleich.

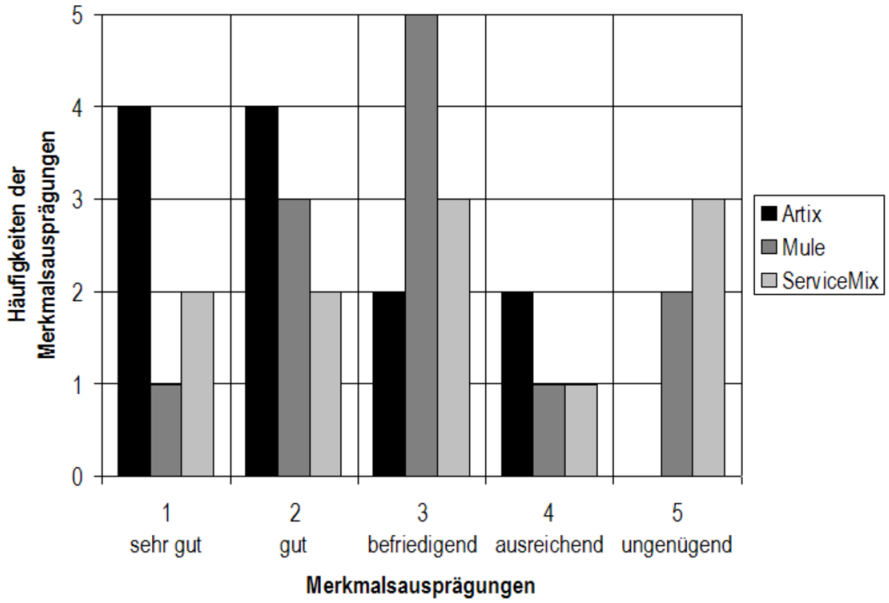

Abbildung F.12: Häufigkeitsverteilung der Qualitätsmetriken.

Literaturverzeichnis

[Ade06] Adelhardt, D. (2006).
Die Sun Application Server Familie – Basis einer SOA.
http://de.sun.com/company/events/2006/software/pdf/applicationserver_isv.pdf.
Online Ressource, Abruf: 18.11.2006.

[BEA05a] BEA Systems, Inc. (Hrsg.) (2005).
BEA AquaLogic™BPM Suite – Product Data Sheet.
http://eudownload.bea.com/eu/products/aqualogic/bpm_suite/BEA_AquaLogic_BPM_Suite_ds.pdf.
Online Ressource, Abruf: 30.10.2006.

[BEA05b] BEA Systems, Inc. (Hrsg.) (2005).
BEA AquaLogic™Service Bus – BEA White Paper.
http://www.bea.com/content/news_events/white_papers/BEA_AL_Service_Bus_wp.pdf.
Online Ressource, Abruf: 04.11.2006.

[BEA06a] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™BPM Suite.
http://de.bea.com/products/aqualogic/bpm_suite/index.jsp.
Online Ressource, Abruf: 30.10.2006.

[BEA06b] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Data Services Platform.
http://de.bea.com/products/aqualogic/data_services/.
Online Ressource, Abruf: 30.10.2006.

[BEA06c] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Data Services Platform – Product Data Sheet.
http://eudownload.bea.com/eu/products/aqualogic/data_services_platform/BEA_AQL_DataSvs_ds.pdf.
Online Ressource, Abruf: 30.10.2006.

[BEA06d] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Enterprise Repository.
http://www.bea.com/framework.jsp?CNT=overview.htm&FP=/content/products/aqualogic/repository/.
Online Ressource, Abruf: 04.11.2006.

[BEA06e] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Enterprise Repository – Product Data Sheet.
`http://www.bea.com/content/news_events/white_papers/BEA_AL_Enterprise_Repository_ds.pdf`.
Online Ressource, Abruf: 04.11.2006.

[BEA06f] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Enterprise Security.
`http://de.bea.com/products/aqualogic/security/product_overview.jsp`.
Online Ressource, Abruf: 04.11.2006.

[BEA06g] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Enterprise Security – Product Data Sheet.
`http://eudownload.bea.com/eu/products/aqualogic/security/BEA_AQL_ES_ds.pdf`.
Online Ressource, Abruf: 04.11.2006.

[BEA06h] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Service Bus – Concepts and Architecture.
`http://edocs.bea.com/alsb/docs25/pdf/concepts.pdf`.
Online Ressource, Abruf: 04.11.2006.

[BEA06i] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Service Registry.
`http://www.bea.com/framework.jsp?CNT=overview.htm&FP=/content/products/aqualogic/service_registry/`.
Online Ressource, Abruf: 04.11.2006.

[BEA06j] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™Service Registry – Product Data Sheet.
`http://www.bea.com/content/news_events/white_papers/BEA_AQL_ServiceReg_ds.pdf`.
Online Ressource, Abruf: 04.11.2006.

[BEA06k] BEA Systems, Inc. (Hrsg.) (2006).
BEA AquaLogic™User Interaction.
`http://de.bea.com/products/aqualogic/user_interaction/`.
Online Ressource, Abruf: 30.10.2006.

[Ben04] Bengel, G. (2004):
Grundkurs Verteilte Systeme. 3. Aufl.
Wiesbaden : Vieweg Verlag, 2004.

[Bis03] Bishop, M. (2003):
Computer Security.
Upper Saddle River, NJ, USA : Pearson Education, 2003.

[Cap06a] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear 6.7 a simpler platform for SOA.
`http://capeclear.com/download/whitepapers/CC67_Datasheet.pdf`.
Online Ressource, Abruf: 18.11.2006.

[Cap06b] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear Data Transformer.
`http://www.capeclear.com/products/data_transformer.shtml`.
Online Ressource, Abruf: 18.11.2006.

[Cap06c] Cape Clear Software, Inc. (Hrsg.) (2006).
Cape Clear ESB Platform, Version 6.7.
`http://www.capeclear.com/download/whitepapers/CapeClear67.pdf`.
Online Ressource, Abruf: 18.11.2006.

[Cap06d] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear ESB Security.
`http://capeclear.com/download/whitepapers/CapeClearESBSecurity4.pdf`.
Online Ressource, Abruf: 18.11.2006.

[Cap06e] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear Manager.
`http://www.capeclear.com/products/manager.shtml`.
Online Ressource, Abruf: 18.11.2006.

[Cap06f] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear Orchestrator.
`http://www.capeclear.com/products/orchestrator.shtml`.
Online Ressource, Abruf: 18.11.2006.

[Cap06g] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear Software – Data Transformer.
`http://capeclear.com/download/whitepapers/DTDataSheet.pdf`.
Online Ressource, Abruf: 18.11.2006.

[Cap06h] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear Studio.
`http://www.capeclear.com/products/studio.shtml`.
Online Ressource, Abruf: 18.11.2006.

[Cap06i] Cape Clear Software Inc. (Hrsg.) (2006).
Cape Clear's Enterprise Service Bus (ESB).
`http://www.capeclear.com/download/whitepapers/ESB_Whitepaper.pdf`.
Online Ressource, Abruf: 06.10.2006.

[Cap06j] Cape Clear Software Inc. (Hrsg.) (2006).
Concepts.
`http://developer.capeclear.com/files/CC66manuals/Concepts.pdf`.
Online Ressource, Abruf: 18.11.2006.

[Cap06k] Cape Clear Software Inc. (Hrsg.) (2006).
Release Notes.
`http://developer.capeclear.com/files/CC67manuals/Release_Notes.pdf`.
Online Ressource, Abruf: 18.11.2006.

[CDK02] Coulouris, G.; Dollimore, J.; Kindberg, T. (2002):
Verteilte Systeme – Konzepte und Design. 3. Aufl.
München : Pearson Education, 2002.

[Cha04a] Chande, S. (2004).
ESB: Enterprise Service Bus.
`http://www.cs.helsinki.fi/u/chande/courses/cs/WSA/presentations/L10_EnterpriseServiceBus.pdf`.
Online Ressource, Abruf: 25.09.2006.

[Cha04b] Chappell, D. A. (2004):
Enterprise Service Bus.
Sebastopol, CA, USA, et al. : O'Reilly Media, Inc., 2004.

[Cha05a] Chappell, D. A. (2005):
Enterprise Service Bus – Interview mit Dave Chappell.
In: JavaSPEKTRUM, Heft Nr. 3/2005, S. 17–18.

[Cha05b] Chappell, D. A. (2005).
ESB vs Biztalk Debate Gets Heated in Barcelona.
`http://www.oreillynet.com/xml/blog/2005/11/esb_vs_biztalk_debate_gets_hea.html`.
Online Ressource, Abruf: 05.11.2006.

[Che05] Cheung, P. (2005).
Implementing SOA with JBI.
`http://sun.hongkongmarket.net/javachina/download/PaulCheung-JBI-SOA-03.pdf`.
Online Ressource, Abruf: 18.11.2006.

[DAT06] DATACOM Buchverlag GmbH (Hrsg.) (2006).
Antwortzeit.
`http://www.itwissen.info/definition/lexikon//__response%20time%20_antwortzeit.html`.
Online Ressource, Abruf: 25.12.2006.

[DE05] Doble, A.; Elting, A. (2005):
Versionierung von Services: SOA für Fortgeschrittene.
In: OBJEKTspektrum, Heft Nr. 3/2005, S. 72 – 75.

[DJMZ05] Dostal, W.; Jeckle, M.; Melzer, I.; Zengler, B. (2005):
Service-orientierte Architekturen mit Web Services.
München : Spektrum Akademischer Verlag, 2005.

[Dum02] Dumke, R. (2002).
Komponenten und CBSE.
`http://ivs.cs.uni-magdeburg.de/~dumke/CBSE/CBSE01.html`.
Online Ressource, Abruf: 28.09.2006.

[Dum03] Dumke, R. (2003):
Software Engineering – Eine Einführung für Informatiker und Ingenieure: Systeme, Erfahrungen, Methoden, Tools. 4. Aufl.
Wiesbaden : Vieweg Verlag, 2003.

[Far03] Farges, N. (2003).
Enterprise Service Bus (ESB): Lasting concept or latest buzzword?
`http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci913058,00.html`.
Online Ressource, Abruf: 23.10.2006.

[Fio04a] Fiorano Software Inc. (Hrsg.) (2004).
The Fiorano Enterprise Service Bus.
`http://www.fiorano.com/whitepapers/fiorano_esb_brochure.pdf`.
Online Ressource, Abruf: 11.11.2006.

[Fio04b] Fiorano Software Inc. (Hrsg.) (2004).
FioranoMQ.
http://www.fiorano.com/whitepapers/fioranomq_brochure.pdf.
Online Ressource, Abruf: 11.11.2006.

[Fio05a] Fiorano Software Inc. (Hrsg.) (2005).
Fiorano BPEL.
http://www.fiorano.com/whitepapers/fiorano_bpel_brochure.pdf.
Online Ressource, Abruf: 11.11.2006.

[Fio05b] Fiorano Software, Inc. (Hrsg.) (2005).
Fiorano SOA 2006 Platform.
http://www.fiorano.com/whitepapers/fiorano_soa_brochure.pdf.
Online Ressource, Abruf: 11.11.2006.

[Fio06] Fiorano Software Inc. (Hrsg.) (2006).
Adapters from Fiorano.
http://www.fiorano.com/products/fesb/adapters/fioranoadapters.htm.
Online Ressource, Abruf: 11.11.2006.

[GT00] Gruhn, V.; Thiel, A. (2000):
Komponentenmodelle.
München et al. : Pearson Education, 2000.

[Gui06] Guild Companies, Inc. (Hrsg.) (2006).
Solaris 10 Breaks Through 6 Million Shipment Barrier.
http://www.itjungle.com/tug/tug110206-story03.html.
Online Ressource, Abruf: 02.12.2006.

[Hei03] Heisel, M. (2003).
Objektorientierte Softwareentwicklung.
http://ivs.cs.uni-magdeburg.de/~heisel/oose/einf2.pdf.
Online Ressource, Abruf: 25.09.2006.

[Hui03] Huizen, G. V. (2003):
Serviceorientierte Architekturen.
In: JavaSPEKTRUM, S. 28–29.

[IDG06a] IDG Business Verlag GmbH, München (Hrsg.) (2006).
Microsoft präsentiert ESB-Anleitung für die SOA.
http://www.computerwoche.de/soa-trends/582163/.
Online Ressource, Abruf: 05.11.2006.

[IDG06b] IDG Communications (Hrsg.) (2006).
Server Operating Systems Buying Guide.
http://www.pcworld.idg.com.au/index.php/id;1327251104.
Online Ressource, Abruf: 02.12.2006.

[ISI06] ISIS Medien (Hrsg.) (2006).
SeeBeyond (Deutschland) GmbH/Sun Microsystems.
http://www.isis-specials.de/profile_pdf/1s263_eai.pdf.
Online Ressource, Abruf: 05.11.2006.

[JBo05] JBoss, a divison of Red Hat, Inc. (Hrsg.) (2005).
JBoss ESB – SOA everywhere!
`http://labs.jboss.com/file-access/default/members/jbossesb/freezone/resources/presentations/JBossESBoverview.pdf`.
Online Ressource, Abruf: 25.09.2006.

[JBo06] JBoss, a divison of Red Hat, Inc. (Hrsg.) (2006).
JBoss ESB requirements and architecture.
`http://labs.jboss.com/file-access/default/members/jbossesb/freezone/resources/project/JBossESBArchitecture.pdf`.
Online Ressource, Abruf: 25.09.2006.

[Kai02] Kaib, M. (2002):
Enterprise Application Integration – Grundlagen, Integrationsprodukte, Anwendungsbeispiele.
Wiesbaden : Deutscher Universitäts-Verlag, 2002.

[KB06] Kamiske, G. F.; Brauer, J.-P. (2006):
Qualitätsmanagement von A bis Z. 5. Aufl.
München – Wien : Carl Hanser Verlag, 2006.

[Kel02] Keller, W. (2002):
Enterprise Application Integration – Erfahrungen aus der Praxis.
Heidelberg : Dpunkt Verlag, 2002.

[Kin05] Kinnumpurath, M. (2005).
JBI – A Standard-based approach for SOA in Java.
`http://www.theserverside.com/tt/articles/content/JBIforSOA/article.html`.
Online Ressource, Abruf: 18.11.2006.

[Kis03] Kischel, S. (2003):
Der „Enterprise Service Bus“: Implementierung einer service-orientierten Architektur.
In: OBJEKTspektrum, Heft Nr. 2/2003, S. 37–40.

[LS05] Lorenzelli-Scholz, D. (2005):
Service-orientierte Integration mit einem „Enterprise Service Bus“.
In: OBJEKTspektrum, Heft Nr. 6/2005, S. 18–22.

[LS06] Langendörfer, H.; Schnor, B. (2006):
Anwendungsorientierte Informatik.
München – Wien : Carl Hanser Verlag, 2006.

[Mic06a] Microsoft, Corp. (Hrsg.) (2006).
Microsoft BizTalk Server 2006 - Systemanforderungen.
`http://www.microsoft.com/germany/biztalk/uebersicht/systemanforderungen.mspx`.
Online Ressource, Abruf: 10.11.2006.

[Mic06b] Microsoft, Corp. (Hrsg.) (2006).
Microsoft on the Enterprise Service Bus (ESB).
`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/47850cbd-63ed-4370-a467-6bd320636902.asp`.
Online Ressource, Abruf: 05.11.2006.

[Mul06a] MuleSource, Inc. (Hrsg.) (2006).
Mule – Architecture Guide.
`http://mule.mulesource.org/wiki/display/MULE/Architecture+Guide`.
Online Ressource, Abruf: 04.12.2006.

[Mul06b] MuleSource, Inc. (Hrsg.) (2006).
Mule – Home.
`http://mule.mulesource.org/wiki/display/MULE/Home`.
Online Ressource, Abruf: 04.12.2006.

[Mul06c] MuleSource, Inc. (Hrsg.) (2006).
Mule – Introduction.
`http://mule.mulesource.org/wiki/display/MULE/Introduction`.
Online Ressource, Abruf: 07.12.2006.

[Mul06d] MuleSource, Inc. (Hrsg.) (2006).
Mule – JMX Management.
`http://mule.mulesource.org/wiki/display/MULE/Jmx+Management`.
Online Ressource, Abruf: 04.12.2006.

[Mul06e] MuleSource, Inc. (Hrsg.) (2006).
Mule – Overview.
`http://mule.mulesource.org/wiki/display/MULE/Overview`.
Online Ressource, Abruf: 04.12.2006.

[Mul06f] MuleSource, Inc. (Hrsg.) (2006).
Mule IDE.
`http://mule.mulesource.org/wiki/display/MULE/Mule+IDE`.
Online Ressource, Abruf: 04.12.2006.

[Nic05] Nicholson, M. (2005).
Is BizTalk Server an ESB?
`http://dnjonline.com/article.aspx?ID=dec05_biztalk`.
Online Ressource, Abruf: 05.11.2006.

[NL04] Newcomer, E.; Lomow, G. (2004):
Understanding SOA with Web Services.
Upper Saddle River, NJ, USA et al. : Pearson Education, 2004.

[OnV06] OnVista Media GmbH (Hrsg.) (2006).
OnVista: Wertpapieranalyse.
`http://www.onvista.de/`.
Online Ressource, Abruf: 23.11.2006.

[Ope06] Operating System Documentation Project (Hrsg.) (2006).
Informationsübersicht der Betriebssysteme.
`http://www.operating-system.org/betriebssystem/_german/os-uebersicht.htm`.
Online Ressource, Abruf: 02.12.2006.

[Ora04] Oracle, Corp. (Hrsg.) (2004).
Oracle BPEL Process Manager.
`http://www.oracle.com/technology/products/ias/bpel/pdf/oracle_bpel_process_manager_datasheet.pdf`.
Online Ressource, Abruf: 04.12.2006.

[Ora05a] Oracle, Corp. (Hrsg.) (2005).
An Introduction to Oracle Web Services Manager.
`http://www.oracle.com/technology/products/webservices_manager/pdf/oracle_wsm_402_wp.pdf`.
Online Ressource, Abruf: 04.12.2006.

[Ora05b] Oracle, Corp. (Hrsg.) (2005).
Oracle Web Services Manager – Feature Overview.
`http://www.oracle.com/technology/products/webservices_manager/htdocs/owsm_10gr2_fov.html`.
Online Ressource, Abruf: 04.12.2006.

[Ora06a] Oracle, Corp. (Hrsg.) (2006).
Oracle BAM.
`http://www.oracle.com/technology/products/integration/bam/pdf/oracle-bam-datasheet.pdf`.
Online Ressource, Abruf: 04.12.2006.

[Ora06b] Oracle, Corp. (Hrsg.) (2006).
Oracle Business Rules.
`http://www.oracle.com/technology/products/ias/business_rules/pdf/dataSheet.pdf`.
Online Ressource, Abruf: 04.12.2006.

[Ora06c] Oracle, Corp. (Hrsg.) (2006).
Oracle Enterprise Service Bus.
`http://www.oracle.com/technology/products/integration/esb/pdf/ds_esb_v10_1_3.pdf`.
Online Ressource, Abruf: 04.12.2006.

[Ora06d] Oracle, Corp. (Hrsg.) (2006).
Oracle SOA Suite.
`http://www.oracle.com/technologies/soa/oracle-soa-suite-datasheet.pdf`.
Online Ressource, Abruf: 04.12.2006.

[Pal05] Palmer, M. (2005):
Event Stream Processing - Tools für eine ereignisgesteuerte service-orientierte Architektur.
In: OBJEKTspektrum, S. 1–6. Sonderheft SOA.

[PT05] Pulier, E.; Taylor, H. (2005):
Understanding Enterprise SOA.
Greenwich, CT, USA : Manning Publications, 2005.

[Rie06] Rieks, M. (2006):
Serviceorientierte Architekturen – Architekturparadigma für die Integration von IT-Systemen.
In: IM – Information Management & Consulting, 21. Jg., Heft Nr. 1/2006, S. 6–10.

[Sch05a] Schmietendorf, A. (2005).
Die industrielle Sicht auf SOA.
`http://ivs.cs.uni-magdeburg.de/~schmiete/lehre/vorlesung/paper/ws05/unimd_soa_part2_v1.pdf`.

Online Ressource, Abruf: 03.10.2006. Im Rahmen der Vorlesung – „Etablierung serviceorientierter Architekturen mit Web Services“.

[Sch05b] Schmietendorf, A. (2005).
Die industrielle Sicht auf SOA.
`http://ivs.cs.uni-magdeburg.de/~schmiete/lehre/vorlesung/paper/ws05/unimd_soa_part4_v1.pdf`.
Online Ressource, Abruf: 24.11.2006. Im Rahmen der Vorlesung – „Etablierung serviceorientierter Architekturen mit Web Services“.

[SGM02] Szyperski, C.; Gruntz, D.; Murer, S. (2002):
Component Software.
London et al. : Pearson Education, 2002.

[Sie03] Siedersleben, J. (Hrsg.) (2003):
Softwaretechnik – Praxiswissen für Software-Ingenieure.
München – Wien : Carl Hanser Verlag, 2003.

[Som01] Sommerville, I. (2001):
Software Engineering.
München et al. : Pearson Education, 2001.

[Son05a] Sonic Software Corp. (Hrsg.) (2005).
Sonic ESB – Datenblatt.
`http://www.sonicsoftware.com/products/docs/sonic_esb_de.pdf`.
Online Ressource, Abruf: 27.10.2006.

[Son05b] Sonic Software Corp. (Hrsg.) (2005).
Sonic ESB: An Architecture and Lifecycle Definition.
`http://www.sonicsoftware.com/products/whitepapers/docs/esb_architecture_definition.pdf`.
Online Ressource, Abruf: 29.10.2006.

[Son06a] Sonic Software Corp. (Hrsg.) (2006).
Adapters for Sonic ESB – Datasheet.
`http://www.sonicsoftware.com/products/docs/adapters_sonic_esb.pdf`.
Online Ressource, Abruf: 29.10.2006.

[Son06b] Sonic Software Corp. (Hrsg.) (2006).
Introducing the Sonic ESB Product Family.
`http://www.sonicsoftware.com/products/documentation/docs/soa_overview.pdf`.
Online Ressource, Abruf: 27.10.2006.

[SP01] Stevens, P.; Pooley, R. (2001):
UML - Softwareentwicklung mit Objekten und Komponenten.
München : Pearson Education, 2001.

[Sun06a] Sun Microsystems, Inc. (Hrsg.) (2006).
Project Open ESB Starter Kit.
`http://java.sun.com/integration/openesb/starterkit.jsp`.
Online Ressource, Abruf: 18.11.2006.

[Sun06b] Sun Microsystems, Inc. (Hrsg.) (2006).
Sun Java Composite Application Platform Suite.
`http://www.sun.com/software/javaenterprisesystem/integration_suite/`.
Online Ressource, Abruf: 05.11.2006.

[Sun06c] Sun Microsystems, Inc. (Hrsg.) (2006).
Sun Java ESB Suite.
`http://www.sun.com/software/javaenterprisesystem/integration_suite/esb_suite/index.xml`.
Online Ressource, Abruf: 05.11.2006.

[Tie06] Tieke, T. (2006):
Die Schritte zur SOA.
In: InformationWeek, Heft Nr. 9/10/2006, S. 10–12.

[TS03] Tanenbaum, A.; van Steen, M. (2003):
Verteilte Systeme – Grundlagen und Paradigmen.
München et al. : Pearson Education, 2003.

[VAC+05] Vogel, O.; Arnold, I.; Chughtai, A.; Ihler, E.; Mehling, U.; Neumann, T.; Völter, M.; Zdun, U. (2005):
Software-Architektur – Grundlagen, Konzepte, Praxis.
München : Spektrum Akademischer Verlag, 2005.

[VG05] Vollmer, K.; Gilpin, M. (2005).
The Forrester Wave: Enterprise Service Bus, Q4 2005.
`http://www.oracle.com/corporate/analyst/reports/infrastructure/fm/ESB_Wave_Final.pdf`.
Online Ressource, Abruf: 11.10.2006.

[VG06] Vollmer, K.; Gilpin, M. (2006).
The Forrester Wave: Enterprise Service Bus, Q2 2006.
`http://www.capeclear.com/download/reports/The_Forrester_Wave_Enterprise_Service_Bus_Q22006.pdf`.
Online Ressource, Abruf: 09.10.2006.

[Wik06a] Wikipedia – Wikimedia Foundation Inc. (Hrsg.) (2006).
Enterprise service bus.
`http://en.wikipedia.org/wiki/Enterprise_service_bus`.
Online Ressource, Abruf: 08.10.2006.

[Wik06b] Wikipedia – Wikimedia Foundation Inc. (Hrsg.) (2006).
ISO/IEC 9126.
`http://de.wikipedia.org/wiki/ISO/IEC_9126`.
Online Ressource, Abruf: 19.10.2006.

[Wor04] World Wide Web Consortium (Hrsg.) (2004).
Web Service Management: Service Life Cycle.
`http://www.w3.org/TR/wslc/`.
Online Ressource, Abruf: 05.10.2006.

[Yah06] Yahoo! Inc. (Hrsg.) (2006).
Yahoo! Finance.
`http://finance.yahoo.com/`.
Online Ressource, Abruf: 23.11.2006.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Steffen Hiekel
Magdeburg, den 21. Januar 2007