Aus dem Institut für Telematik

der Universität zu Lübeck

Direktor:

Prof. Dr. rer. nat Stefan Fischer

Konzepte und Technologien zur Anwendungs-Integration

Inauguraldissertation

zur

Erlangung der Doktorwürde

der Universität zu Lübeck

- Aus der Technisch-Naturwissenschaftlichen Fakultät -

Vorgelegt von

Herrn Dipl.-Wirt.-Inf. Marcus Tiedemann

aus Bremen

Lübeck, im Oktober 2006

1. Berichterstatter: Prof. Dr. rer. nat Stefan Fischer

2. Berichterstatter: Prof. Dr. rer. nat Volker Linnemann

Datum der mündlichen Prüfung: 24.01.2007

Kurzfassung

Auch wenn Schlagwörter wie Service-orientierte Architektur (SOA) und Enterprise Service Bus (ESB) derzeit in aller Munde sind, ist der zugrunde liegende Gedanke der Anwendungs-Integration keinesfalls neu. Schließlich adressierten beispielsweise Remote Procedue Calls (RPCs), CORBA, RMI oder Datenreplikation genau diese Problematik, nämlich den Datentransfer von einer Anwendung zu einer anderen. Durch diese historisch gewachsene Zahl von Alternativen ergibt sich heute oftmals das Problem der Auswahl der richtigen Integrationsmethode und -technologie, bei der diese Arbeit als Entscheidungs-Leitfaden dienen kann.

Zunächst wird der Themenbereich der Anwendungs-Integration in geeigneter Form untergliedert. Dabei stellt es sich als zweckmäßig heraus, drei sog. Dimensionen abzugrenzen: das zugrunde liegende Konzept, die Art des Zugriffs auf die Anwendungen und die verwendete Technologie.

Nach der Vorstellung einiger Technologie-Beispiele sowie der im Rahmen dieser Arbeit unterschiedenen Integrations-Konzepte (d.h. Datenintegration, Punkt-zu-Punkt-Verbindungen, EAI (Enterprise Application Integration) und SOA) wird ein Vorgehensmodell präsentiert, mit dessen Hilfe die optimale Ausgestaltung der o.g. drei Dimensionen auf möglichst objektive Art und Weise erreicht werden kann.

Den Kern dieses Vorgehensmodells bilden acht Kriterien, die Merkmale einer losen Kopplung. Lose Kopplung bedeutet allgemein, dass zum einen die Verbindung der beteiligten Komponenten so robust (d.h. wenig fehleranfällig) wie möglich ist und zum anderen der Aufrufer einer bestimmten Funktionalität so wenig wie möglich über deren Anbieter wissen muss. Der „Grad der losen Kopplung“ lässt sich unabhängig voneinander durch alle acht Kriterien beeinflussen.

Die weitere Anwendung des Vorgehensmodells gliedert sich in zwei Bereiche: Auf der Soll-Seite werden Anforderungen spezifiziert, die an die Kriterien im konkreten Anwendungsfall gestellt werden, und auf der Ist-Seite wird nach geeigneten Umsetzungsmöglichkeiten gesucht.

Die Anwendung dieses Vorgehensmodells bietet zahlreiche Vorteile. So wird durch die Trennung zwischen Soll- und Ist-Seite erreicht, dass die Spezifikation der Anforderungen vollständig unabhängig z.B. von bestimmten Technologien erfolgt. Darüber hinaus wird durch die auf der Ist-Seite verankerte Unterscheidung der drei Dimensionen sichergestellt, dass nicht z.B. Technologie und Konzept vermischt werden oder ein bestimmtes Konzept von vornherein mit einer Technologie verbunden wird (wie z.B. SOA und Web Services).

Abstract

Even though buzzwords like Service Oriented Architecture (SOA) or Enterprise Service Bus (ESB) presently are widely used, the basic idea of application integration is not new. After all, Remote Procedure Calls (RPCs), CORBA, RMI or data replication, for instance, address exactly the same problem, namely the transfer of data from one application to another. Because of this historically grown number of alternatives, it is difficult today to choose the right integration method and technology. This thesis can help with this choice by providing a decision guideline.

First, the topic of application integration is clustered in a suitable way. For this, it has been found appropriate to define three so called dimensions: the underlying concept, the way of accessing the applications, and the applied technology.

After the presentation of some technology examples and the integration concepts distinguished within this thesis (i.e. data integration, point to point connections, EAI (Enterprise Application Integration) and SOA), a process model is introduced. It allows the design of the three dimensions as objectively as possible.

The core of this process model is formed by eight criteria, the attributes of loose coupling. Loose coupling generally means that, on the one hand, the connection of the participating components is as robust (i.e. not error-prone) as possible, and, on the other hand, that the invoker of certain functionality has to know as few as possible about its provider. The "degree of loose coupling" can be affected independently by all of these eight criteria.

The further use of the process model is divided into two areas: At the "target" site ("Soll") requirements are specified, that the criteria have to fulfill in a concrete scenario, and at the "actual" site ("Ist") suitable implementation possibilities are searched.

The appliance of this process model offers numerous advantages. Because of the separation of the "target" and "actual" site, it can be achieved, that the specification of the requirements is realized independently e.g. of certain technologies. Furthermore, the distinction between the three dimensions fixed at the "target" site assures that e.g. technology and concept are not mixed and no concept is associated with a certain technology from the outset (like e.g. SOA and Web Services).

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1: Einleitung

Informationstechnologie ist heutzutage ein unverzichtbarer Bestandteil einer jeden Organisation. In den Arbeitsabläufen von Industrie-Unternehmen, Behörden etc. unterstützen IT-Systeme die Arbeit der Menschen oder automatisieren Vorgänge.

Allerdings arbeiten die wenigsten Systeme (wenn überhaupt) völlig autark und unabhängig von anderen. Die Zusammenarbeit kann sich darauf beschränken, dass Anwendungslogik, die einmal entwickelt wurde, an anderer Stelle wieder benutzt wird, kann sich aber auch bis zur Ausgestaltung fachlicher Prozesse entlang der kompletten Wertschöpfungskette erstrecken.

Der Terminus „Anwendungs-Integration“ wird in dieser Arbeit als Oberbegriff für all diese Bemühungen gebraucht. Die Zielsetzung ist aber je nach Anwendungsfall unterschiedlich und kann in vier Bereiche aufgeteilt werden. Dies ist Gegenstand von 1.1.

In 1.2 wird anschließend dargelegt, welche Problemstellung diese Arbeit adressiert, und in 1.3 wird ihr genauer Aufbau skizziert.

1.1 Zielsetzung von Anwendungs-Integration

Um Missverständnisse zu vermeiden, muss zunächst auf die Bedeutung der Begriffe Enterprise Application Integration (EAI)[1] und Anwendungs-Integration im Rahmen dieser Arbeit hingewiesen werden.

Im täglichen Sprachgebrauch wird EAI zuweilen als Oberbegriff über alle Varianten der Integration von Anwendungen gebraucht (z.B. als Bezeichnung „EAI Competence Center“ innerhalb einer Unternehmens-Organisation). Theoretisch, d.h. vom Begrifflichen her, ist dagegen im Grunde auch nichts einzuwenden. Allerdings wird EAI in der Praxis sehr viel häufiger mit der sogenannten Hub-and-Spokes-Architektur („Nabe und Speichen“), die unter dem Namen EAI bekannt geworden ist, verbunden. Diese Bedeutung wird auch im Rahmen dieser Arbeit verwendet; als Oberbegriff für alle Varianten der Integration wird der weniger vorbelastete Terminus „Anwendungs-Integration“ gebraucht.

[1] Dieses Konzept wird in Kapitel 4 noch detailliert behandelt; eine kurze Beschreibung ist zudem in Anhang A zu finden.

Unabhängig von der konkreten Methode oder Technologie werden mit der Integration von Anwendungen zwei grobe Ziele verfolgt, nämlich

- das Einsparen von Kosten (für die Erstellung und Wartung von Schnittstellen) und
- die Erhöhung der Flexibilität (bezüglich des Austauschs von einzelnen Systemen oder der Änderung von Geschäftsprozessen).

Diese beiden Grobziele können auf vier grundlegend verschiedenen Wegen bzw. in vier Bereichen erreicht werden, die es sich näher zu betrachten lohnt. Diese Zielbereiche, die in 1.1.1 bis 1.1.4 näher erläutert werden, sind im Einzelnen [FT06]:

1. Schnittstellen
2. Geschäftsprozesse
3. Anwendungsentwicklung
4. Investitionsschutz

Die Zielbereiche sind voneinander weitestgehend unabhängig. Das heißt, dass je nach Integrations-Szenario ein anderes Ziel oder eine andere Kombination von Zielen maßgeblich ist. Diese Zieldefinition bzw. -auswahl ist der erste Schritt auf dem Weg zur Empfehlung der geeigneten Integrations-Methode und -Technologie.

1.1.1 Schnittstellen

Die Problematik der sogenannten „Spaghetti-Architektur“ (d.h. einem sehr schwer zu übersehenden Geflecht von Punkt-zu-Punkt-Verbindungen[2]) war der Auslöser für die Entwicklung der ersten EAI-Ansätze. Deren Ziel, das auch heute noch aktuell ist, war die Reduzierung der Anzahl und Heterogenität der Schnittstellen zwischen den Anwendungen zum Zwecke einer besseren Übersicht und Wartbarkeit. Außerdem soll dadurch eine automatisierte und schnellere Kommunikation zwischen diesen Anwendungen ermöglicht werden.

1.1.2 Geschäftsprozesse

Ein Problem in den meisten Unternehmen ist die Kommunikation zwischen dem Fachbereich und den ihn unterstützenden IT-Stellen. Auf Seiten des Fachbereichs werden die Anforderungen in der Sprache der jeweiligen Problemdomäne (Geschäftsobjekte, Aktivitäten, Rollen usw.) formuliert. Die IT-Spezialisten dagegen denken in Anwendungen, die im Normalfall nicht ohne weiteres einem dedizierten fachlichen Problem zugeordnet werden können und

[2] Detailliertere Informationen hierzu folgen in Kapitel 4.

umgekehrt. Durch eine Kapselung der Anwendungen in fachlich abgegrenzte Services (Dienste) ist es möglich, den bislang fehlenden „gemeinsamen Nenner“ für die Kommunikation zu finden.

Diese Service-Orientierung bietet zudem die Möglichkeit, einen Geschäftsprozess durch die Aneinanderreihung von Service-Aufrufen abzubilden. Auf die fachliche Modellierung des Prozesses hat dies zwar keine Auswirkungen – dies ist nach wie vor die Aufgabe des Fachbereichs und nicht der IT –, aber die Abbildung des fachlichen Prozesses auf die unterstützenden Anwendungen wird durch diese zusätzliche Abstraktionsschicht wesentlich erleichtert.

Darüber hinaus kann Anwendungs-Integration einen wesentlichen Beitrag zur Erhöhung der Flexibilität bei Änderungen des fachlichen Prozesses leisten. Während ein solches Szenario auch heute noch zumeist unweigerlich zur Umprogrammierung von Anwendungen führen muss, soll es in Zukunft ausreichen, die Services als Ganzes in einer anderen Reihenfolge zu verknüpfen. Deren interne Funktionalität bleibt davon aber unberührt.

Ferner wird es auf diese Weise möglich, ganze Services von externen Anbietern zu kaufen und zu nutzen, die nur auf Service-Ebene in die eigene IT-Landschaft integriert, nicht aber im Rechenzentrum zum Laufen gebracht werden müssen. Dass die tatsächliche Umsetzung dieser Möglichkeit auch unternehmenspolitisch uneingeschränkt gewünscht ist, bedeutet deren Erwähnung an dieser Stelle natürlich nicht. Diese Überlegungen sind auch nicht Gegenstand dieser Arbeit.

1.1.3 Anwendungsentwicklung

Die Kapselung von Anwendungsfunktionalität in wiederverwendbare Komponenten (welcher Art und Größe auch immer) ermöglicht nicht nur deren Zusammenschaltung zu Prozessen, sondern auch die Nutzung der vorhandenen Funktionalität in anderen Anwendungen. Die Idealvorstellung ist, dass im Zuge der Entwicklung einer neuen Anwendung nur die Logik, die es zumindest im eigenen Unternehmen noch nicht gibt, programmiert werden muss, aber alle anderen Bestandteile in Form eines Aufrufs von Komponenten aus bestehenden Applikationen eingebunden werden.

1.1.4 Investitionsschutz

Durch Anwendungs-Integration können viele bestehende (Legacy-) Anwendungen, die aufgrund von veralteter Technologie eigentlich abgelöst werden müssten, weiter genutzt werden. Ermöglicht werden kann dies zum einen durch Standardisierung der Zugriffs-Schnittstelle(n), zum anderen durch eine geeignete Kapselung und demzufolge die Nutzung der Altanwendung als „Black Box“, die sich nach außen genauso verhält wie eine neuere Anwendung.

1.2 Zielsetzung dieser Arbeit

In frühen Zeiten der Informatik standen für die Kommunikation zwischen Anwendungen lediglich traditionelle Medien wie Papier und Bleistift oder Telefon zur Verfügung. Mit anderen Worten, die Daten mussten manuell aus einem System extrahiert und in ein anderes eingefügt werden. Doch schon bald entstanden z.B. mit dem SUN RPC (Remote Procedure Call), der Daten-Replikation oder dem Datei-Transfer über Netzwerke Mechanismen zum automatischen Austausch von Daten. Im Laufe der Zeit wurden neuere Technologien und Konzepte entwickelt, wie beispielsweise Java RMI oder CORBA. Das vorläufige Ende dieser Entwicklung stellen die Service-orientierte Architektur (SOA), der Enterprise Service Bus (ESB) und Web Services dar.

Auch wenn natürlich die neueren Konzepte und Technologien viele Möglichkeiten und Vorteile im Vergleich zu den älteren mitbringen, so bedeutet das nicht, dass sie in jeder Situation die richtige Lösung sind. Vielmehr hat so gut wie jede Alternative Vor- und Nachteile, die sie in speziellen Szenarien u.U. zur einzigen brauchbaren Möglichkeit, unter anderen Rahmenbedingungen aber gänzlich unbrauchbar machen.

Diese Auswahl an Möglichkeiten führt aber auch zu Problemen: Wie findet man für das individuelle Integrations-Problem bzw. die Schnittstelle, die man gerade entwerfen muss, die optimale Methode und Technologie? Wie kann man herausfinden, ob die gewählte Umsetzung einer bestehenden Schnittstelle noch die beste Wahl ist? Und wenn nicht, ist eine Umstellung dieser doch grundsätzlich funktionierenden Schnittstelle sinnvoll?

Die Beantwortung dieser Fragen erfolgt in der Praxis zumeist äußerst subjektiv [FT06]. Man orientiert sich beispielsweise an

- aktuellen „Hypes“, deren Vorteile man gar nicht benötigt, deren Nachteile z.B. bezüglich Performance oder Implementierungs-Aufwand man aber trotzdem in Kauf nimmt
- Gewohnheiten, d.h. man macht es „so wie immer“, ohne neue Entwicklungen zu berücksichtigen
- kurzfristig vorhandenem Personal und dessen persönlichen Vorlieben
- dem „Weg des geringsten Widerstandes“, d.h. es wird die Technologie verwendet, die im ersten Schnelltest die wenigsten Probleme bereitet.

Allerdings ist die Wahl der optimalen Lösung alles andere als trivial, in jedem Falle keine einfache „wenn, dann“-Entscheidung.

Das zentrale Ziel dieser Arbeit besteht folglich darin, ein Vorgehensmodell auszuarbeiten, auf dessen Basis die Auswahl von Integrations-Methoden und -Technologien wesentlich objektiver und fundierter als bisher sowie mehr auf den konkreten Anwendungsfall bezogen erfolgen kann.

1.3 Aufbau dieser Arbeit

Um zwischen verschiedenen Alternativen auswählen zu können, ist es zunächst notwendig, die gesamte Thematik der Anwendungs-Integration geeignet zu untergliedern. Dies ist Gegenstand von Kapitel 2. Hierbei werden zunächst vier Gliederungen aus der Literatur vorgestellt; nachdem deren Nachteile aufgezeigt wurden, folgt schließlich die Präsentation eines neuen Ansatzes, der sich in drei Dimensionen untergliedert, welche die Basis für die weiteren Untersuchungen im Rahmen dieser Arbeit bilden.

Die Grundlage einer jeden Lösung zur Anwendungs-Integration bildet eine geeignete Technologie. Damit ist an dieser Stelle nicht die Hardware oder das Betriebssystem gemeint, sondern Middleware-Technologien zur Kommunikation zwischen verschiedenen Systemen oberhalb der Transportschicht des ISO/OSI-Schichtenmodells, die in Kapitel 3 erläutert werden. Es wurde bewusst darauf geachtet, dieses Kapitel nicht zur „Produktschau“ verkommen zu lassen, d.h. nur fertige Produkte verschiedener Hersteller vorzustellen, sondern der Fokus liegt auf der Präsentation verschiedener Basis-Technologien.

Mit diesen Technologien alleine kann zwar eine Kommunikation zwischen Anwendungen realisiert werden, aber eine strategisch ausgerichtete Anwendungs-Integration bedarf eines Konzepts oder einer Methode, die prinzipiell unabhängig von speziellen Technologien ist. In Kapitel 4 werden vier verschiedene Konzepte erläutert und gleichzeitig jeweils die Frage der Möglichkeiten einer technologischen Umsetzung auf der Basis von Kapitel 3 diskutiert.

Kapitel 3 und 4 ergeben zusammen den „Werkzeugkasten“, d.h. verschiedene Möglichkeiten zur Realisierung einer Anwendungs-Integration. Kapitel 5 enthält schließlich den Kern dieser Arbeit, nämlich das Vorgehensmodell, auf dessen Basis die für eine individuelle Situation geeignete Zusammenstellung dieser „Werkzeuge“ ermittelt werden kann.

Nach dem Ausblick in Kapitel 6 werden in Anhang A einige zentrale Fachbegriffe als Glossar nochmals kurz erklärt. Diese Ausführungen sollten auch schon vor der Lektüre der detaillierteren Ausführungen insbesondere in Kapitel 4 verständlich sein und können somit für eine grobe Orientierung des Lesers hilfreich sein.

1.4 Zusammenfassung des eigenen Beitrags

Der in diese Arbeit eingeflossene eigene Beitrag besteht aus mehreren Aspekten:

- Herausarbeitung der unterschiedlichen Zielsetzungen der Anwendungs-Integration
- Analyse bestehender Gliederungen dieser Thematik und die Erarbeitung eines eigenen, neuen Ansatzes
- Definition und Abgrenzung verschiedener Integrations-Konzepte und Analyse der jeweiligen Vor- und Nachteile sowie Einsatzbedingungen
- Untersuchung eines möglichen Zusammenspiels von EAI und SOA sowie der Problematik von untergeordneten Services
- Erarbeitung eines Vorgehensmodells als Entscheidungs-Leitfaden für die Bestimmung der optimalen Integrations-Lösung in Abhängigkeit der jeweils vorliegenden Ausgangssituation

Teilergebnisse hiervon wurden in den folgenden Beiträgen veröffentlicht:

- Marcus Tiedemann, Volker Wittig, Stefan Rougier: Die Gestaltung einer unternehmensweiten, flexiblen Software-Architektur auf Basis eines selbst entwickelten Zonen- und Schichten-Modells, in: Schelp, Winter (Hrsg.): Auf dem Weg zur Integration Factory: Proceedings der DW2004 – Data Warehousing und EAI, Heidelberg 2005
- Marcus Tiedemann: Handling subordinate services in a service-oriented architecture, in: Althoff u.a. (Hrsg.): WM2005: Professional Knowledge Management: Experiences and Visions, Kaiserslautern 2005
- Marcus Tiedemann: Die Identifikation geeigneter Technologien für Integrations-Szenarien auf Basis des Grades der losen Kopplung, in: Lehner, Nösekabel, Kleinschmidt (Hrsg.): Multikonferenz Wirtschaftsinformatik 2006, GITO-Verlag (Berlin) 2006
- Stefan Fischer, Marcus Tiedemann: Auswahl von Methoden und Technologien für die Integration von Anwendungen, in: WISU – Das Wirtschaftsstudium 6/2006, S. 812-817

Kapitel 2: Die Gliederung der Thematik der Anwendungs-Integration

Bevor verschiedene Konzepte oder Technologien zur Anwendungs-Integration miteinander verglichen werden können, ist es zunächst notwendig, eine geeignete Gliederung dieser Thematik zu finden. Dies ist der Inhalt dieses Kapitels.

Dass der Integrationsbegriff weitaus älter ist als die aktuell diskutierten Konzepte zur Anwendungs-Integration (wie z.B. SOA), wird am Beispiel des Integrationsbegriffs nach Heilmann verdeutlicht.

Anschließend werden mit den Gliederungen von Keller, Linthicum und Kaib drei aktuellere Ansätze vorgestellt, bevor am Ende dieses Kapitels die eigene Gliederung präsentiert wird, die dieser Arbeit zugrunde liegt.

2.1 Der Integrationsbegriff nach Heilmann

Heilmann beschäftigte sich mit dem Begriff der Integration bereits 1989 [Hei89], also deutlich vor der Blütezeit des klassischen EAI-Ansatzes. Dennoch findet man hier viele Begriffe wieder, die später unter dem Namen EAI eine Rolle gespielt haben und auch heute noch spielen.

Wie in Abbildung 1[3] ersichtlich, unterscheidet Heilmann vorwiegend zwischen der Integrationsform (also dem integrierenden Objekt) und dem Integrationsbereich. Darüber hinaus erwähnt sie die Integrationsreichweite (d.h. inner- oder zwischenbetrieblich) sowie die Ausrichtung. Eine horizontale Ausrichtung bedeutet dabei eine Verknüpfung von Daten und Funktionen einzelner Geschäftsbereiche, bei der vertikalen Integration steht die Verbindung verschiedener Hierarchie-Ebenen im Vordergrund. Der Begriff der temporalen Integration bezieht sich auf die zeitliche Abstimmung von Anwendungen, die an einem Prozess beteiligt sind.

[3] Diese Abbildung zeigt nicht die Original-Grafik von Heilmann [Hei89], sondern eine von Linß [Linß95] überarbeitete Version, die inhaltlich weitgehend identisch, aber bezüglich der Darstellung verbessert ist.

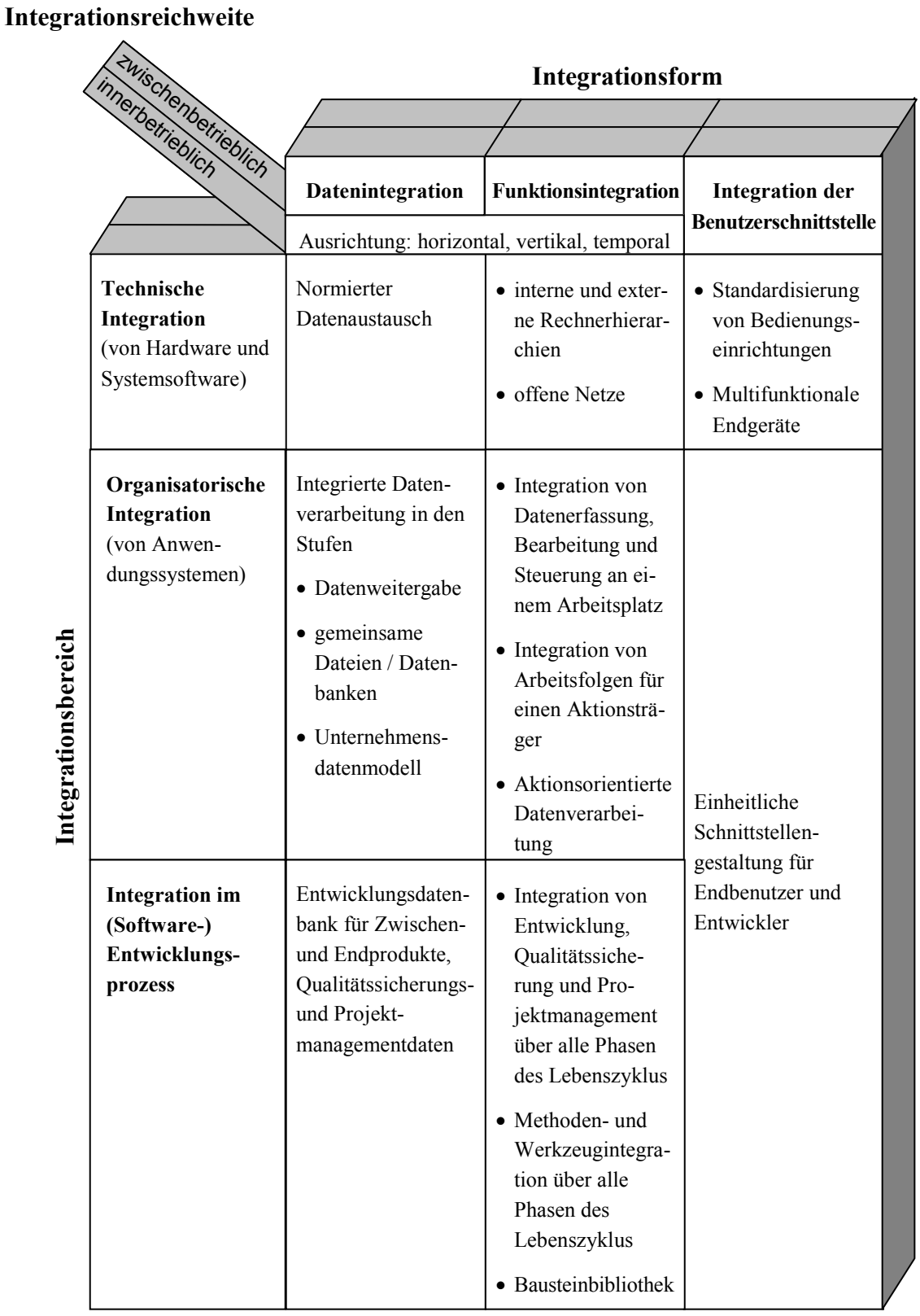

Abbildung 1: Der Integrationsbegriff nach Heilmann (nach [Linß95])

2.2 Integrationsmethoden nach Keller

Die Integrationsmethoden nach Keller [Kel02] basieren auf der Darstellung eines betrieblichen Informationssystems in Form von drei Schichten (vgl. Abbildung 2): Benutzungs-Schnittstelle, Anwendungskern und Datenhaltung. Dementsprechend unterscheidet Keller die im Folgenden vorgestellten Ansatzpunke für die Integration von Systemen.

Abbildung 2: Die Einteilung eines betrieblichen Informationssystems in drei Schichten nach Keller [Kel02]

2.2.1 Integration über die Benutzungs-Schnittstelle

Das Prinzip ist hierbei die Erstellung einer neuen Benutzungs-Schnittstelle über den bereits existierenden, sei es aufgrund einer besseren Benutzbarkeit oder einer integrationsbedingt erweiterten Funktionalität (vgl. Abbildung 3).

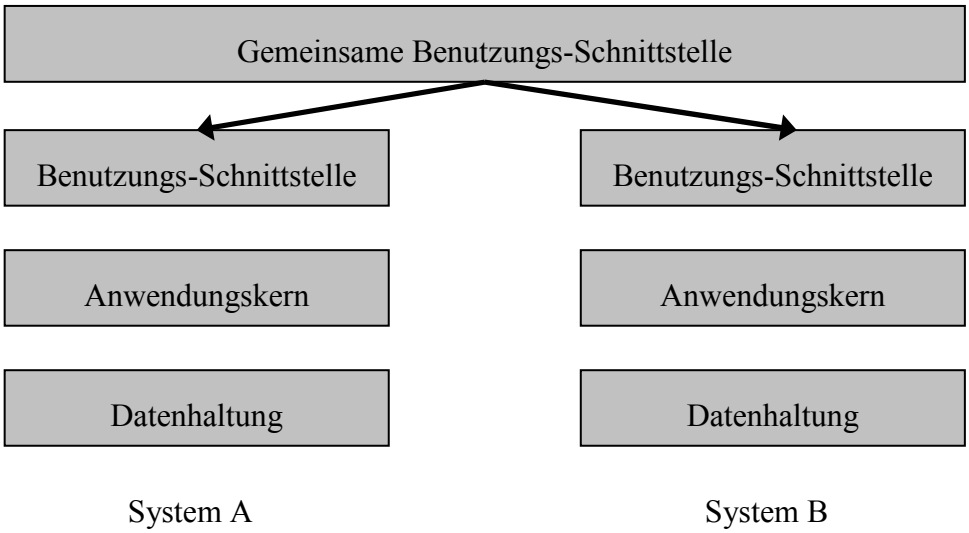

Abbildung 3: Integration über die Benutzungs-Schnittstelle nach Keller [Kel02]

Eine Integration über die Benutzungs-Schnittstelle kann z.B. über Portal-Server realisiert werden, aber auch durch das klassische „Screen Scraping“ (also das „Abkratzen“ der Benutzungs-Schnittstelle, wie es im Wesentlichen bei Host-Systemen zum Teil notwendig ist; vgl. z.B. [Kai02]).

2.2.2 Integration über Funktionsaufrufe

Hierunter wird der Aufruf einer Funktionalität eines Anwendungskerns von einem anderen Anwendungskern aus verstanden (siehe Abbildung 4).

Dies ist natürlich ein sehr weites Feld, und es gibt folglich sehr viele Faktoren und Designfragen, die beeinflussen, auf welche Art diese Integration im Detail realisiert wird. Beispielsweise spielt hier eine Rolle, ob die Anwendungen eine sauber definierte Schnittstelle haben oder ob ein Modul bzw. Unterprogramm einer anderen Anwendung direkt genutzt wird.

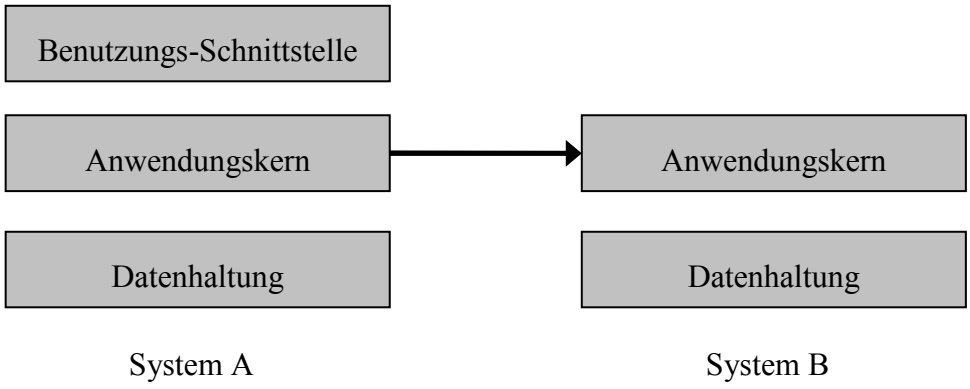

Abbildung 4: Integration über Funktionsaufrufe nach Keller [Kel02]

2.2.3 Integration über Datenbanken

Keller definiert die Integration über Datenbanken als die Kommunikation mehrerer Anwendungen über eine gemeinsame Datenbasis. Er unterscheidet dabei zwei Szenarien, nämlich die Integration über föderierte Datenbanken (d.h. die Nutzung mehrerer Datenbanken, die über eine Middleware so zusammengefasst werden, dass sie für den Programmierer wie eine einzige aussehen; siehe Abbildung 5) und die Integration über gemeinsame Datenbanken (also die Benutzung einer Datenbank durch mehrere Anwendungen; siehe Abbildung 6).

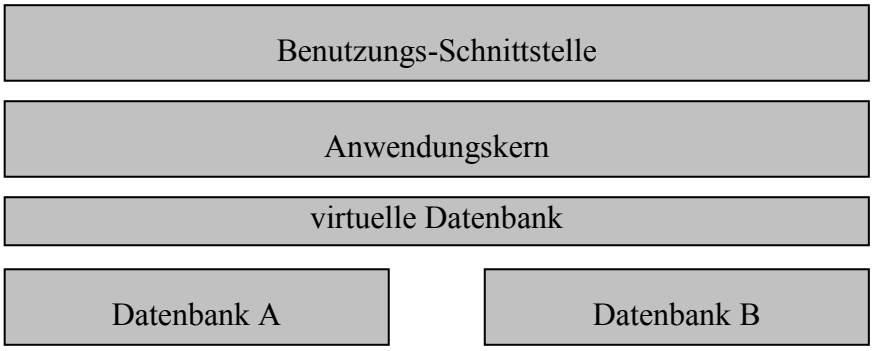

Abbildung 5: Integration über föderierte Datenbanken nach Keller [Kel02]

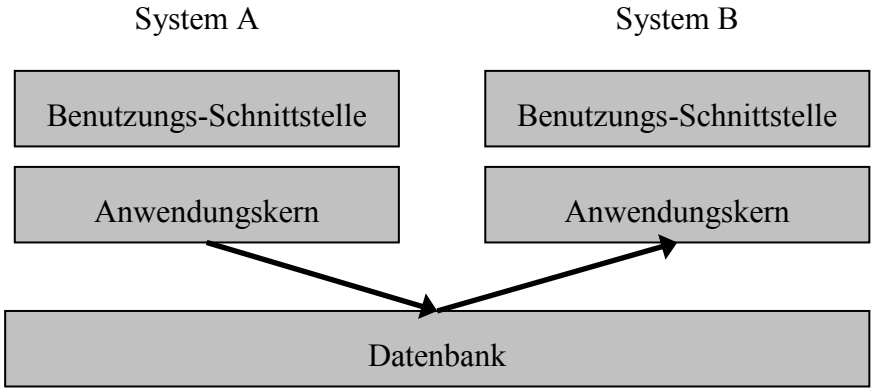

Abbildung 6: Integration über gemeinsame Datenbanken nach Keller [Kel02]

2.2.4 Integration über Komponenten

Neben diesen drei Varianten existiert nach Keller die Möglichkeit der Integration über Komponenten. Hierunter fallen Plugins (d.h. die Integration an dafür vorgesehenen Stellen) sowie Komponenten in so genannten Containern (z.B. Enterprise Java Beans).

2.3 Application Integration Approaches nach Linthicum

Linthicum [Lint04] unterteilt die Anwendungs-Integration in vier Kategorien, die im Folgenden kurz erläutert und anschließend mit der Unterteilung von Keller (vgl. 2.2) verglichen werden.

2.3.1 Information-Oriented Application Integration (IOAI)

Der Grundgedanke dieser Betrachtungsweise ist, dass Datenbanken und datenorientierte Schnittstellen als die primären Punkte der Integration angesehen werden. Folglich werden hierbei keine Anwendungsdienste oder Prozesse ausgetauscht, sondern ausschließlich Daten.

Linthicum unterteilt dabei drei Kategorien:

- **Daten-Replikation:**

  Der Grundgedanke hierbei ist das simple Kopieren von Daten zwischen zwei oder mehr Datenbanken. Diese Datenbanken können von einem oder von mehreren Herstellern stammen und sogar auf unterschiedlichen Datenmodellen basieren.

- **Daten-Föderation:**

  Mit Hilfe der Daten-Föderation können mehrere Datenbanken nach außen als eine einzige sichtbar gemacht werden. Das Ergebnis ist eine virtuelle Datenbank, deren Aufgabe darin besteht, die Anfragen korrekt an die dahinter liegenden physischen Datenbanken weiter zu leiten.

- **Datenorientierte Schnittstellen („interface processing“):**

  Hierbei werden wohldefinierte Schnittstellen genutzt, um sowohl eigene als auch Standard-Anwendungen in die vorhandene Systemlandschaft zu integrieren.

Auf eine detailliertere, insbesondere grafische Beschreibung wird an dieser Stelle verzichtet, da in Kapitel 4 noch ausführlich auf das Konzept der Datenintegration eingegangen wird.

2.3.2 Business Process Integration-Oriented Application Integration (BPIOAI)

Das Prinzip dieser Methode, die wohl am besten mit “Geschäftsprozess-orientierte Anwendungs-Integration” übersetzt werden kann, ist die Definition von einfachen, zentral verwalteten Prozessen über denjenigen, die in den einzelnen Anwendungen enthalten sind.

Diese neu geschaffenen, zentralen Prozesse stellen die Steuereinheiten dar und bieten Zugriff sowohl auf Informationen als auch auf (Sub-) Prozesse der einzelnen Anwendungen. Im Vergleich zur normalen Anwendungs-Integration sieht Linthicum diese Methode als mehr strategisch ausgerichtet an: Normalerweise entstehe die Motivation aus dem Bedarf zur Kommuni-

kation zwischen verschiedenen Anwendungen; hier aber stünden Geschäftsmodelle und -regeln, um die bestehenden Systeme in optimaler Weise zu nutzen, im Vordergrund.

Der Nutzen dieser Methode ist neben der Visualisierung der in den verteilten Systemen enthaltenen Prozesse auch die Möglichkeit einer Echtzeit-Messung ihrer Leistung.

2.3.3 Service-Oriented Application Integration (SOAI)

Die Service-orientierte Anwendungs-Integration nach Linthicum erlaubt Applikationen, nicht nur Informationen (wie bei der informationsorientierten Anwendungs-Integration; vgl. 2.3.1), sondern auch Anwendungsdienste (Methoden) zu teilen. Dies kann auf zweierlei Weise geschehen:

- durch Übertragung auf einen zentralen Server
- durch direkten Zugriff auf die Anwendungen (z.B. mit Hilfe von verteilten Objekten oder Web Services)

Auch wenn der Name dies suggeriert, so ist das, was Linthicum unter „Service-Oriented Application Integration“ versteht, keinesfalls identisch mit einer Service-orientierten Architektur (vgl. A.2 oder Kapitel 4). Linthicum fokussiert auf die technische Bottom-Up-Integration, d.h. die Identifikation von Services oder besser Schnittstellen in den verteilten Anwendungen und die Zusammenfassung dieser Funktionalitäten in so genannten Composite Applications. Der Nutzen einer SOA kommt aber erst dann vollständig zum Tragen, wenn die fachliche Seite, d.h. die Top-Down-Definition von Services, mindestens gleichwertig betrachtet wird.

2.3.4 Portal-Oriented Application Integration (POAI)

Portal-orientierte Anwendungs-Integration beschreibt Linthicum als die Darstellung einer Vielzahl von Systemen in einer gemeinsamen Benutzungs-Schnittstelle.

Die Folge ist eine Integration aller beteiligten Systeme, obwohl die Anwendungen selbst (also in technischer Hinsicht) nicht integriert sind.

2.3.5 Vergleich der Gliederungen von Keller und Linthicum

Obwohl sie prinzipiell die gleiche Problematik bearbeitet haben, sind Keller und Linthicum – wie in 2.2 und 2.3 ersichtlich war – zu sehr unterschiedlichen Ergebnissen gekommen.

Die größte Übereinstimmung ist noch im Bereich der Integration über Datenbanken (vgl. 2.2.3) bzw. Information-Oriented Application Integration (2.3.1) zu finden. Der Aspekt der föderierten Datenbanken wird von beiden Autoren beleuchtet, wenn auch Linthicum nicht auf die Variante der physischen Vereinigung eingeht. Dafür fehlt bei Keller zumindest die Daten-Replikation.

Service-Oriented Application Integration (2.3.3) passt noch am ehesten zur Integration über Funktionsaufrufe (2.2.2). Allerdings zielt Linthicum eher auf den Aspekt der Erzeugung von Composite Applications, der bei Keller überhaupt nicht auftaucht.

Ebenso stellt die Portal-Oriented Application Integration (2.3.4) nur einen Teilbereich der Integration über die Benutzungs-Schnittstelle (2.2.1) dar.

Diese signifikanten Unterschiede lassen sicht damit begründen, dass die beiden Autoren den Problembereich mit unterschiedlichen Blickwinkeln analysiert haben. Während Linthicum allgemeine Methoden oder Paradigmen herausgearbeitet hat, liegt der Fokus von Keller eher bei der Frage, auf welche Art und Weise der Zugriff auf die vorhandenen Anwendungen realisiert werden kann.

Der einzige dem Verfasser bekannte Autor, der eine derartige Unterscheidung vornimmt, ist Kaib [Kai02], dessen Gliederung im folgenden Abschnitt präsentiert wird.

2.4 Integrationskonzepte und –ansätze nach Kaib

Kaib [Kai02] unterscheidet unter der Überschrift „Wege zur Anwendungsintegration“ zwischen Integrationskonzepten und Integrationsansätzen, die im Folgenden jeweils näher erläutert werden.

2.4.1 Integrationskonzepte

Die Integrationskonzepte von Kaib decken sich weitgehend mit der Einteilung von Keller (vgl. 2.2). Auch hier wird eine dreischichtige Anwendungs-Architektur zugrunde gelegt, die sinngemäß mit Abbildung 2 identisch ist.

Kaib unterscheidet folgende Integrationskonzepte:

- **Präsentationsintegration:**

  Diese Möglichkeit, die in Abbildung 7 skizziert ist, weist eine Verwandtschaft zu der von Keller aufgeführten Integration über die Benutzungs-Schnittstelle (vgl. 2.2.1) auf, hat allerdings einen wichtigen Unterschied: Bei Keller wird eine neue Benutzungs-Schnittstelle erstellt, über die eine Person die Altanwendungen ausführen kann. Kaibs Konzept sieht dagegen vor, dass eine existierende Präsentationsschicht von der Applikationsfunktionalitäts-Schicht einer anderen Anwendung (also nicht von einem Menschen) über Screen-Scraping-Mechanismen aufgerufen wird.

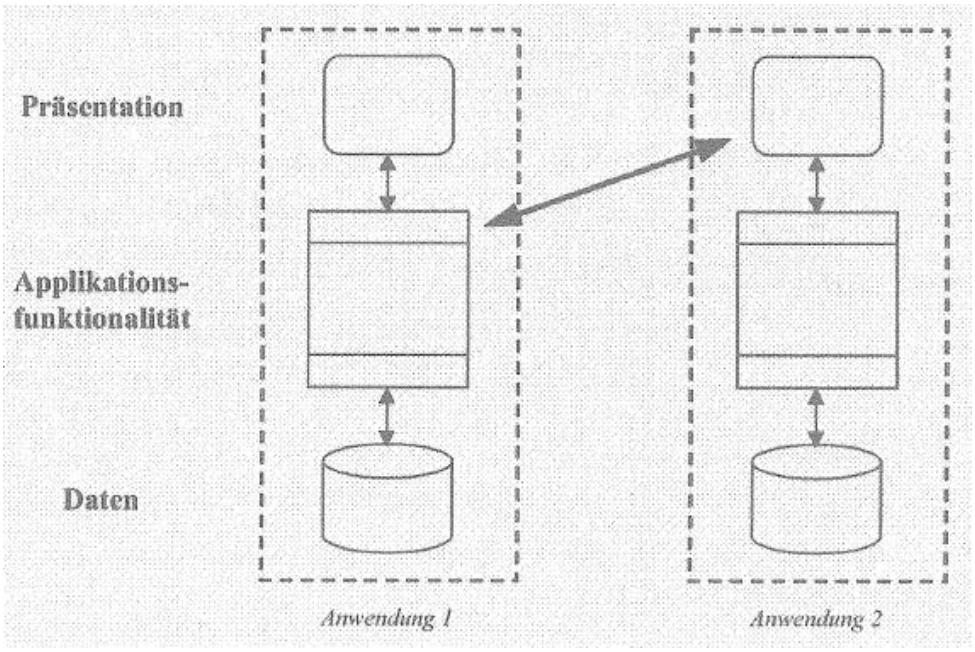

Abbildung 7: Präsentationsintegration nach Kaib [Kai02]

- **Datenintegration:**

  Die Besonderheit der Datenintegration nach Kaib (siehe Abbildung 8) ist, dass der Zugriff auf die Anwendungen ausschließlich auf der Datenebene, d.h. unter Umgehung der Präsentations- und Logikebene, erfolgt.

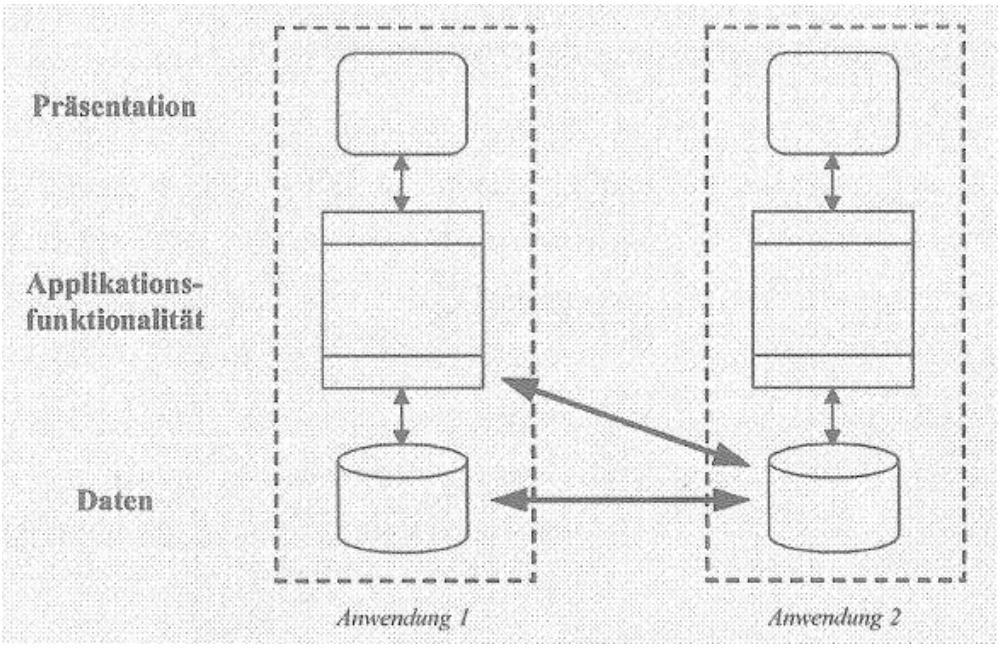

Abbildung 8: Datenintegration nach Kaib [Kai02]

Kaib unterscheidet dabei zwei Möglichkeiten der Verwirklichung (wie an den beiden Pfeilen in Abbildung 8 erkenntlich ist): Zum einen bietet sich der direkte Austausch von Daten zwischen verschiedenen Datenbanken an, zum anderen kann über virtuelle Datenbanken der Zugriff von der Applikationsfunktionalitäts-Schicht aus erreicht werden. Die-

se zweite Möglichkeit entspricht genau dem, was Keller als „Integration über föderierte Datenbanken“ (vgl. Abbildung 5) bezeichnet. Der direkte Austausch hat Ähnlichkeit zur „Integration über gemeinsame Datenbanken“ nach Keller (vgl. Abbildung 6), allerdings setzt Kaib keine gemeinsame Datenbank, sondern lediglich einen Abgleich der verteilten Datenbanken voraus.

- **Funktionsintegration:**

  Dieses Konzept, das in Abbildung 9 dargestellt ist und das Kaib als das weitestgehende bezeichnet, ist deckungsgleich mit der Integration über Funktionsaufrufe nach Keller (vgl. Abbildung 4).

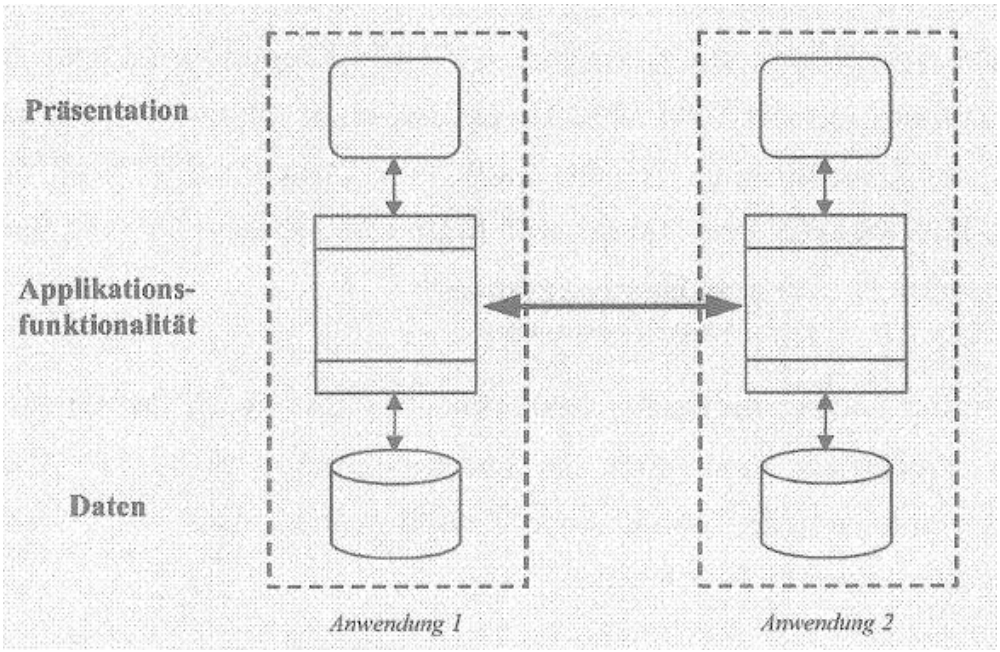

Abbildung 9: Funktionsintegration nach Kaib [Kai02]

2.4.2 Integrationsansätze

Mit den Integrationsansätzen beschreibt Kaib alternative Wege, auf denen die Integrationskonzepte verwirklicht werden können. Die von ihm gewählte Untergliederung bezeichnet er als „traditionelle Ansätze zur Anwendungsintegration“, die als Grundtypen möglicher Vorgehensweisen zu verstehen seien. Dies sind im Einzelnen:

- **Punkt-zu-Punkt-Verbindungen:**

  Unter einer Punkt-zu-Punkt-Verbindung versteht Kaib „die dedizierte Verbindung zwischen zwei gleichberechtigten Systemen bzw. Anwendungen“ [Kai02]. Allerdings sei diese Art von Verbindungen häufig das Resultat einer unzureichenden Planung. Auch wenn man auf diese Weise schnell an die Quell- und Zielsysteme angepasste Integrati-

onslösungen entwickeln könne, so würden doch auf lange Sicht die Nachteile überwiegen (detailliertere Informationen hierzu folgen in Kapitel 4).

- **ERP-basierte Integration:**

  Durch die Einführung von ERP-Paketen (das bekannteste ist sicherlich SAP R/3) sinkt der Integrationsbedarf, da deren funktionale Module vom jeweiligen Hersteller bereits in vorintegrierter Form angeboten werden. Allerdings gibt Kaib zu bedenken, dass davon auszugehen sei, dass auch in Zukunft selten mehr als 40% der Anwendungsunterstützung durch ERP-Pakete geleistet werden könne. Trotzdem hätten ERP-Pakete in vielen Unternehmen die Rolle der zentralen Kernanwendung übernommen; hierauf baue der ERP-basierte Integrationsansatz auf.

- **Middleware-basierter Integrationsansatz:**

  Hierbei wird zur Lösung des Integrationsproblems Middleware, d.h. eine vermittelnde Softwareschicht, zwischen zwei oder mehrere Systeme geschaltet. Dadurch können die angebundenen Anwendungen hersteller- und ggf. plattformunabhängig Daten austauschen. Eine weitere Unterteilung des Begriffs „Middleware“ nimmt Kaib nicht vor.

2.4.3 Kritik an den Integrationsansätzen

Auch wenn die Abgrenzung von den Integrationskonzepten ein sinnvoller Schritt war, so ist die Auswahl der Integrationsansätze in mehrfacher Hinsicht nicht zielführend.

Sicherlich kann durch den Einsatz von ERP-Systemen eine wesentlich homogenere und integriertere Anwendungslandschaft geschaffen werden als durch viele heterogene, individuell verbundene Systeme. Dennoch hat man diese freie Auswahl nur bei einer Konzeption „auf der grünen Wiese“, d.h. beim kompletten Neuaufbau der Systeme. Die Integrationsprobleme in der Praxis basieren aber im Allgemeinen auf einer vorhandenen System-Infrastruktur, deren Werte es nach Möglichkeit zu erhalten gilt (vgl. das Ziel „Investitionsschutz“ in 1.1.4).

Auch nach der Blütezeit der ERP-Systeme (wie z.B. SAP R/3) in den achtziger und neunziger Jahren werden diese nach wie vor in vielen Bereichen, wie z.B. Controlling oder Personalwesen, erfolgreich eingesetzt. Die ursprüngliche Intention bei der Entwicklung von ERP-Systemen war aber, *alle* abzubildenden Funktionalitäten mit *einem* ERP-System umzusetzten [Kay03]. Dieses Ziel hat sich jedoch als zu komplex herausgestellt; stattdessen beschränkt man sich heute darauf, zumindest eine Vielzahl fachlich verwandter Funktionalitäten in einem ERP-System zusammenzufassen.

Aus Sicht dieses Fachbereichs ist die ERP-Einführung zweifellos sinnvoll. Als Konzept für eine unternehmensweite Anwendungs-Integration ist dieser Ansatz dennoch nicht anzusehen. Schließlich ist – wie soeben erläutert – bei einer ERP-basierten Integrations-Strategie zu erwarten, dass die integrierte Landschaft nicht aus einem einzigen, sondern aus mehreren großen Systemen bestehen wird. Damit wäre aber genau die Ausgangssituation erreicht, die man mit EAI-Mitteln (und ebenso mit den Nachfolgern dieses Ansatzes) beseitigen wollte, näm-

lich ein Nebeneinander von mehreren zueinander inkompatiblen „Silos“, die oft sogar größer und verschiedener sind als zuvor [Kay03]. Da zudem Kaib selbst – wie unter 2.4.2 erwähnt – davon ausgeht, dass die ERP-basierte Integration weniger als 40% der Anwendungen betrifft, ist deren Einstufung als strategisches Integrationskonzept zumindest fragwürdig.

Der Aspekt der Middleware-basierten Integration ist zwar zielführend, aber viel zu undifferenziert. Insbesondere unter dem Aspekt der losen Kopplung (vgl. A.6 sowie die detaillierten Ausführungen hierzu in Kapitel 5) hat die Wahl der Middleware einen erheblichen Einfluss auf die Robustheit der Kommunikation und die Austauschbarkeit der beteiligten Systeme.

Darüber hinaus werden auch nach der Unterteilung zwischen Integrationskonzepten und -ansätzen immer noch verschiedene Dimensionen vermischt: Punkt-zu-Punkt-Verbindungen stellen eine abstrakte Methode oder ein Konzept der Integration dar; vergleichbar wäre der klassische EAI-Ansatz oder eine Service-orientierte Architektur[4]. Middleware dagegen steht für konkrete Technologien, die geeignet sein können, solche Methoden umzusetzen.

2.5 Ein neuer Ansatz: Die drei Dimensionen der Anwendungs-Integration

Die im bisherigen Verlauf dieses Kapitels – insbesondere in 2.4.3 – aufgeführten Kritikpunkte an verschiedenen Versuchen einer Gliederung sind die Basis für die Ausgestaltung der drei Dimensionen der Anwendungs-Integration (siehe Abbildung 10) [Tie06], die das Grundraster des weiteren Verlaufs dieser Arbeit darstellen.

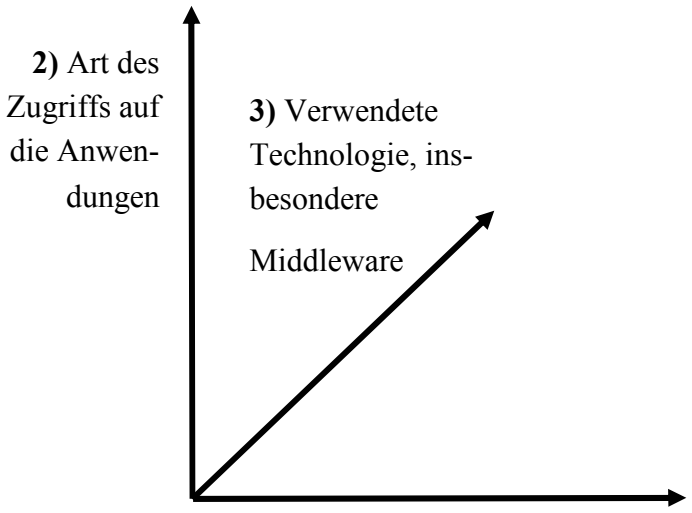

Abbildung 10: Die drei Dimensionen der Anwendungs-Integration [Tie06]

[4] Diese Konzepte werden ausführlich in Kapitel 4 beschrieben.

Die Grafik in Abbildung 10 ist nicht als Koordinatensystem im engeren (mathematischen) Sinne zu verstehen, da zum einen keine Metrik zugrunde liegt, sondern lediglich verwandte Begriffe, und zum anderen kein Nullpunkt vorhanden ist. Dennoch kann jede Integrationslösung als Punkt im Kontext dieser drei Achsen dargestellt und der Unterschied zwischen verschiedenen Realisierungsmöglichkeiten anhand der Position der zugehörigen Punkte visualisiert werden.

Diese Einteilung wird der Erkenntnis gerecht, dass sich die Lösung eines Integrationsproblems stets in drei Schritte gliedert, die im Folgenden beschrieben werden.

2.5.1 Die erste Dimension: Das zugrunde liegende Konzept

Die erste Dimension beschreibt, mit welchem grundsätzlichen Vorgehen das Integrationsproblem gelöst werden soll bzw. gelöst worden ist. Hierbei spielen verschiedene Evolutionsstufen der Anwendungs-Integration in den letzten Jahrzehnten eine wichtige Rolle (siehe hierzu im Detail Kapitel 4).

Die Ausprägungen dieser Dimension, die das Grundraster von Kapitel 4 bilden und folglich dort detailliert erläutert werden, sind die folgenden:

1. Integration auf Datenebene (Datenintegration)
2. Punkt-zu-Punkt-Verbindung
3. EAI (d.h. der klassische EAI-Ansatz mit Hub-and-Spokes-Architektur)
4. SOA

Die Subsummierung der Datenintegration unter dem Oberbegriff der Anwendungs-Integration ist in der Literatur keineswegs einheitlich und unumstritten.

So wird beispielsweise in [JMP02] der allgemeine Begriff der Integration unterteilt in Portal-, Geschäftsprozess-, Anwendungs- und Informations-Integration. Anwendungs-Integration wird mit einem Fokus auf Datentransformation und Message Queuing verstanden, Informations-Integration hat dagegen eher das Ziel, dass Anwendungen auf alle relevanten Unternehmensdaten zugreifen können [Herg03] und entspricht somit ansatzweise dem, was in dieser Arbeit unter Datenintegration verstanden wird.

Da aber auch eine solche Datenintegration beispielsweise als Alternative zu Punkt-zu-Punkt-Verbindungen zur Informations-Beschaffung eingesetzt werden kann, wird in dieser Arbeit die vorgestellte Unterteilung verwendet.

2.5.2 Die zweite Dimension: Die Art des Zugriffs auf die Anwendungen

Während die erste Dimension die Integration als Ganzes im Blick hat, richtet sich der Fokus dieser zweiten Dimension auf deren Endpunkte, d.h. auf die zu integrierenden Anwendungen. Hierbei werden – ähnlich zu den Integrations-Konzepten[5] von Kaib (vgl. 2.4.1) – die folgenden drei Möglichkeiten unterschieden:

1. über die Benutzungs-Schnittstelle
2. über Funktions-Aufrufe
3. über Datenbank-Zugriffe

Der „normale“ Weg sollte die Nutzung von Funktions-Aufrufen sein. Hierunter ist jeder Aufruf von Anwendungslogik zu verstehen, sei es beispielsweise in Form von Web Services, Enterprise Java Beans (EJBs), RPCs oder CICS-Transaktionen (die genaue Wahl der Technologie ist Gegenstand der dritten Dimension).

Auch wenn man in vielen Fällen ältere Schnittstellen-Technologien z.B. als Web Service kapseln kann, ist die Möglichkeit des Zugriffs auf Anwendungen über Funktions-Aufrufe grundsätzlich davon abhängig, ob bei der Implementierung dieser Anwendungen die Möglichkeit eines direkten Aufrufs von Funktionalität vorgesehen wurde. Insbesondere bei älteren Host-Anwendungen ist aber oftmals die Anwendungslogik so monolithisch gestaltet, dass man nur eine Möglichkeit des Zugangs hat, nämlich über die seit jeher dafür vorgesehene Benutzungs-Schnittstelle. Für deren automatisierte Nutzung ist es also unumgänglich, einen Mensch-Maschine-Dialog zu simulieren. Die Ausgaben des Programms können dann z.B. durch „Screen Scraping“ eingelesen werden. Das funktioniert so, dass sich der Wert eines Parameters aus einer Zeichenfolge an einer bestimmten Stelle des Bildschirminhalts ergibt.

Natürlich ist diese Art des Zugriffs unter dem Gesichtspunkt einer losen Kopplung extrem ungünstig, da es zum Auftreten eines Fehlers und damit zum Scheitern der Kommunikation bereits ausreicht, dass die Rückgabewerte an einer anderen Stelle einer Bildschirm-Maske auftauchen. Dennoch ist dies – wie gesagt – oftmals die einzige Möglichkeit, überhaupt automatisiert auf eine Anwendung zugreifen zu können.

[5] An dieser Stelle sei ausdrücklich darauf hingewiesen, dass in dieser Arbeit der Begriff „Integrations-Konzept“ grundlegend anders gebraucht wird als bei Kaib. Eine Analogie besteht zwischen der zweiten Dimension, also der Art des Zugriffs auf die Anwendungen, und Kaibs Integrations-Konzepten. Die erste Dimension, die den Titel „Integrations-Konzepte“ trägt, deckt dagegen einen Teil dessen ab, was Kaib mit „Integrations-Ansätze“ überschrieben hat.

Eine echte Alternative zu Funktions-Aufrufen – nicht nur im Notfall – stellen Datenbank-Zugriffe dar. Dies bedeutet, dass von anderen Anwendungen benötigte Daten nicht über Rückgabewerte von Funktionen geliefert, sondern direkt aus dem von der Zielanwendung gepflegten Datenbestand ausgelesen werden. Dies hat z.B. den Vorteil, dass keine zusätzliche Prozessorlast für die Zielanwendung erzeugt wird. Nähere Einzelheiten hierzu können unter der Überschrift „Datenintegration“ (4.1) nachgelesen werden.

Ein Grenzfall bezüglich der Zuordnung innerhalb dieser zweiten Dimension sind datenorientierte Schnittstellen. Technologisch betrachtet stellt deren Nutzung einen Funktions-Aufruf dar. Wenn sich hinter dieser Funktion aber nichts weiter verbirgt als ein Datenbank-Zugriff, dessen Resultate ohne weitere Bearbeitung zurückgegeben werden, wäre das Resultat eines direkten Datenbank-Aufrufs unter Umgehung der Anwendungslogik dasselbe. In inhaltlicher Hinsicht wäre also eine Einordnung unter der Überschrift „Datenbank-Zugriff“ durchaus gerechtfertigt. So hat beispielsweise Linthicum [Lint04] den Aspekt „interface processing“ zusammen mit der Daten-Replikation und -Föderation unter der Überschrift „information-oriented application integration“ eingeordnet (vgl. 2.3.1).

Die Gefahr an dieser Klassifikation ist aber, dass man für die Einordnung nicht nur die Kommunikation selbst, sondern auch deren Inhalte bzw. die hinter der aufgerufenen Schnittstelle liegende Anwendungslogik betrachten muss. Dies bedeutet in der Praxis einen erheblichen Mehraufwand und wird oftmals vermutlich auch einfach übersehen, so dass ein inkonsistentes Bild die Folge wäre. Darüber hinaus ist es auf diese Weise schwierig, eine klare Trennung zwischen Funktions-Aufruf und Datenbank-Zugriff zu ziehen (d.h. ob beispielsweise eine Funktion, die zwar zusätzliche, über den reinen Datenbankzugriff hinausgehende Logik durchführt – das aber nur zur Sicherung der Integrität – auch als Datenbank-Zugriff angesehen werden kann).

Aus diesen Gründen werden datenorientierte Schnittstellen in dieser Arbeit bezüglich der Zuordnung innerhalb dieser zweiten Dimension nicht von anderen Funktions-Aufrufen unterschieden.

Abschließend sei noch ausdrücklich erwähnt, dass auch im Kontext einer SOA, wo – wie in 4.4 noch genauer erläutert wird – die Anwendungen gerade nicht mehr im Mittelpunkt der Betrachtung stehen, sondern hinter Services verborgen sein sollen, an dieser Stelle der Begriff „Anwendung“ korrekt ist. Das Bereitstellen oder Kapseln eines Service ist bereits die Ausgestaltung dieser zweiten Dimension, d.h. die Lösung der Problemstellung des Zugriffs auf bestehende Anwendungen. Die Kommunikation zwischen Services spielt sich auf einer anderen Ebene ab.

Im Gegensatz zur ersten und dritten Dimension der Anwendungs-Integration (vgl. 2.5.1 und 2.5.3) wird die zweite Dimension im weiteren Verlauf dieser Arbeit nicht nochmals im Detail aufgegriffen, da die Beschreibung an dieser Stelle bereits ausreichend ist und kein weiterer Erklärungsbedarf existiert.

2.5.3 Die dritte Dimension: Die verwendete Technologie, insbesondere Middleware

Die Abgrenzung dieser dritten Dimension ist hilfreich, um nicht bestimmte Konzepte von vornherein mit Technologien zu verknüpfen. So sind z.B. Web Services die für die Umsetzung einer SOA am besten geeignete Technologie, was aber weder bedeutet, dass eine SOA nur mit Web Services realisiert werden kann, noch dass man durch die Nutzung von Web Services alleine schon eine SOA geschaffen hat (mehr hierzu in Kapitel 4).

Genauso wie bei der ersten Dimension (vgl. 2.5.1) stellt diese Beschreibung der dritten Dimension nur die Spitze des Eisbergs dar und wird in einem eigenen Kapitel im weiteren Verlauf dieser Arbeit noch ausführlich behandelt.

Die Position dieses Technologie-Kapitels (Kapitel 3) ist bewusst vor der Detaillierung der Integrations-Konzepte in Kapitel 4 gewählt. Der Grund für diese Reihenfolge (3. Dimension vor 1. Dimension) ist, dass die Technologien nicht mehr als ein Handwerkszeug darstellen. Daher ist es zweckmäßig, diese zunächst weitgehend unabhängig von möglichen Anwendungen zu präsentieren, um anschließend die Möglichkeit zu haben, bei der Darstellung der Integrations-Konzepte gleich passende Technologien zu diskutieren. Letztendlich ist ja die Nummerierung der Dimensionen auch willkürlich gewählt.

2.5.4 Kombinationsmöglichkeiten von Konzepten und Zugriffsmöglichkeiten

Auch wenn die Trennung der Dimensionen eine Entkopplung bei der Planung der jeweiligen Inhalte erreichen soll, können diese nicht beliebig kombiniert werden. Denn zum einen ist nicht jede Kombination strategisch sinnvoll, zum anderen sind manche sogar überhaupt nicht möglich.

Zugriffsmöglichkeit / Konzept	Benutzungs-Schnittstelle	Funktions-Aufrufe	Datenbank-Zugriffe
Integration auf Datenebene			
Punkt zu Punkt			
EAI			
SOA			

Abbildung 11: Kombinationsmöglichkeiten von Konzepten und Zugriffsmöglichkeiten

Eine Übersicht über die Kombinationsmöglichkeiten der ersten beiden Dimensionen ist in Abbildung 11 ersichtlich. Ein weißes Feld bedeutet eine strategisch sinnvolle Kombination, ein dunkelgrau unterlegtes eine grundsätzlich nicht mögliche. Ein hellgrau unterlegtes Feld besagt, dass diese Kombination zwar denkbar, aber nicht strategisch wünschenswert ist und somit nur im Notfall gebraucht werden sollte.

Im Folgenden werden die Gründe für diese Festlegungen kurz erläutert, um die Nachvollziehbarkeit bereits an dieser Stelle zu ermöglichen und Querverweise zu vermeiden. Ausführlich werden die Integrations-Konzepte erst in Kapitel 4 behandelt.

Eine Integration auf Datenebene bedeutet per definitionem, dass nicht auf die Anwendungen selbst (sei es über die Benutzungs-Schnittstelle oder über Funktions-Aufrufe), sondern nur auf die unter ihnen liegenden Datenbanken zugegriffen wird.

Eine Punkt-zu-Punkt-Verbindung ist stets ein individuelles Konstrukt, das an beliebiger Stelle auf die Zielanwendung zugreifen kann. Einzig der direkte Zugriff auf Datenbanken ist unter dem Gesichtspunkt der Wahrung der Integrität zu vermeiden, technisch möglich ist aber auch diese Variante.

Der klassische EAI-Ansatz wie auch die SOA sehen normalerweise einen Aufruf von Anwendungslogik vor. Falls – wie oben beschrieben – keine andere Möglichkeit existiert, ist auch der Weg über die Benutzungs-Schnittstelle denkbar, wenn er auch – insbesondere im SOA-Fall – hinter einer standardisierten Schnittstelle verborgen werden dürfte. Der direkte Zugriff auf Datenbanken ist bei EAI noch zumindest theoretisch denkbar, bei SOA dagegen gar nicht, denn sobald ein Datenbank-Zugriff als Service gekapselt wird, erfolgt ein Aufruf von Anwendungslogik, die aus den unter 2.5.2 genannten Gründen nicht mehr als Datenbank-Zugriff eingestuft wird.

2.5.5 Vergleich der ersten Dimension mit anderen Gliederungen

Die aus wissenschaftlicher Sicht interessanteste Dimension ist zweifellos die erste, d.h. das zugrunde liegende Konzept (vgl. 2.5.1). Zum Abschluss der Vorstellung der drei Dimensionen der Anwendungs-Integration soll an dieser Stelle ein Vergleich dieser ersten Dimension mit anderen Gliederungen aus der Literatur in tabellarischer Form (siehe Tabelle 1) erfolgen.

In der linken Spalte sind zum einen die soeben erläuterten möglichen Konzepte zu finden. Zum anderen wird in Stichworten begründet, warum die in der Literatur ergänzend enthaltenen Punkte hier keine Entsprechung finden.

Dies kann zwei Ursachen haben: Wie im bisherigen Verlauf dieses Kapitels mehrfach begründet, werden an mehreren Stellen in der Literatur verschiedene Dimensionen miteinander vermischt. Dadurch enthalten die hier zitierten Gliederungen einige Punkte, deren Einordnung in die erste Dimension der Anwendungs-Integration nicht sinnvoll ist.

Auf der anderen Seite werden einige Aspekte aus diesen Gliederungen im Rahmen dieser Arbeit bewusst nicht betrachtet, wie beispielsweise das Themenfeld der Portale. Natürlich haben

Portale etwas mit Integration zu tun, aber in der Praxis stellen sie doch zumeist nur den Einstiegspunkt für einen menschlichen Benutzer zu Systemen dar, die untereinander mit anderen Methoden und Technologien integriert sind. Die letzteren sind es aber, auf die der Fokus dieser Arbeit gerichtet ist.

In den weiteren Spalten sind die Gliederungen von Kaib [Kai02], Keller [Kel02], Hohpe/Woolf [HW04] und Erl [Erl04] sowie zwei Ansätze von Linthicum [Lint00][Lint04] zu sehen.

Die Reihenfolge ist dabei teilweise gegenüber der originalen so abgeändert worden, dass sie zur linken Spalte passt, also inhaltlich identische Konzepte jeweils in einer Zeile angeordnet sind.

Die Klassifikation von Heilmann wurde in der Tabelle nicht berücksichtigt, da sie inhaltlich zu stark von den anderen abweicht, so dass der tabellarische Vergleich nicht sinnvoll ist. Auf eine nähere Erläuterung der Ansätze von Hohpe/Woolf und Erl sowie der älteren Gliederung von Linthicum wurde verzichtet, da hierdurch kein zusätzlicher Erkenntnisgewinn erreicht werden kann.

Im Vergleich zur Übersicht in 2.5.1 sind in dieser Tabelle EAI und SOA in der Reihenfolge vertauscht worden. Dies geschah alleine aufgrund der Übersicht, da in der (zweiten) Einteilung von Linthicum SOA und Punkt-zu-Punkt-Verbindungen nicht wesentlich voneinander getrennt werden und somit untereinander stehen sollten.

Zusammengefasst sind die in diesem Kapitel vorgestellten drei Dimensionen der Anwendungs-Integration aus mehreren Gründen besser zur Gliederung der dieser Arbeit zugrunde liegenden Thematik geeignet.

Die Trennung der ersten und dritten Dimension hilft dabei sicherzustellen, dass auf der einen Seite nicht eine Integrations-Methode von vornherein mit einer bestimmten Technologie verknüpft wird und auf der anderen Seite die Technologie nicht alleine als Lösung für ein Integrations-Problem angesehen wird.

Die konkrete Ausgestaltung der zweiten Dimension hat eher vervollständigenden Charakter; viel wichtiger ist ihre getrennte Betrachtung, insbesondere von der ersten Dimension. Auf diese Weise wird verhindert, dass die Ebene des Zugriffs auf die Anwendungen bereits als Integrations-Konzept angesehen wird. So ist beispielsweise die Aussage „wir binden diese Systeme nicht in die SOA ein, sondern sprechen sie stattdessen auf der Datenhaltungs-Schicht an“ wenig sinnvoll, da zum einen der besagte Daten-Zugriff auch als Service gekapselt und in eine SOA eingebunden werden kann und zum anderen die Ansprache auf der Datenhaltungs-Schicht noch nichts darüber aussagt, mit welchem Konzept dies erfolgen soll.

Zur Erhöhung der Übersichtlichkeit wurden außerdem innerhalb der drei Dimensionen einige Aspekte nicht berücksichtigt, die zwar nicht verkehrt sind, aber in der industriellen Praxis zumindest in strategischer oder architektureller Sicht nur eine untergeordnete Rolle spielen (wie z.B. die Komponenten-basierte Integration).

in dieser Arbeit gewählte Kategorie	Linthicum (2000)	Kaib (2002)	Keller (2002)	Linthicum (2003)	Hohpe / Woolf (2004)	Erl (2004)
Datenintegration	Data level integration	Datenintegration	Integration über Datenbanken	Information-oriented integration	File transfer / shared database	Data-level integration
Punkt-zu-Punkt-Verbindung	Applic. interface level / method level Integration	Punkt-zu-Punkt-Verbindungen		Service-oriented integration	Remote procedure invocation	Application-level integration
SOA						Service-oriented integration
EAI	*(Spezialfall der unternehmens-internen Integr.)*	*(Oberbegriff für alle Ansätze)*	*(Oberbegriff für alle Ansätze)*	*(Spezialfall der unternehmens-internen Integr.)*	*(nur ein Konzept, aber keine Umsetzung)*	*(Oberbegriff für fortschrittlichere Middleware)*
(nicht berück-sichtigt)	Business-process-integration-oriented integr.					Process-level integration
(nicht berück-sichtigt)				Portal-oriented integration		
(Art des Zugriffs, also 2. Dimension)	User interface level integration	Präsentations-integration	Integration über die Benutzungs-Schnittstelle			
(nicht berück-sichtigt)			Integration über Komponenten			
(Art des Zugriffs, also 2. Dimension)		Funktions-integration	Integration über Funktions-Aufrufe			
(verwendete Technologie, also 3. Dimension)		ERP-basierte Integration				
(verwendete Technologie, also 3. Dimension)		Middleware-basierte Integration			Messaging	

Tabelle 1: Vergleich der ersten Dimension der Anwendungs-Integration mit anderen Gliederungen

Kapitel 3: Basis-Technologien

In diesem Kapitel werden verschiedene Technologien beschrieben, die man zur Integration von Anwendungen verwenden kann. Dies entspricht folglich der dritten Dimension (verwendete Technologie) der in 2.5 vorgestellten drei Dimensionen der Anwendungs-Integration.

Nicht zuletzt aufgrund der Tatsache, dass in jedem größeren Unternehmen eigene, proprietäre Technologien zum Einsatz kommen, kann und soll dieses Kapitel keinen Anspruch auf Vollständigkeit erheben. Die im Folgenden beschriebenen Technologien sind vielmehr als Beispiele oder als Repräsentanten verschiedener Technologie-Familien zu sehen, die als Orientierungshilfe dienen, um ggf. andere Technologien gemäß dem in Kapitel 5 beschriebenen Vorgehen ebenso zu evaluieren.

Es sei ausdrücklich darauf hingewiesen, dass in diesem Kapitel – wie schon die Überschrift aussagt – lediglich Basis-Technologien und keine Integrations-Produkte oder gar Frameworks behandelt werden. Der IBM WebSphere Message Broker [IBM06b] beispielsweise ist natürlich eine Alternative für die Anwendungs-Integration, baut aber intern auf WebSphere MQ auf. Alle weiteren Bestandteile gehen über die unmittelbare Basis-Kommunikation hinaus und sind daher nicht Gegenstand dieses Kapitels und auch nur ausschnittsweise Gegenstand dieser Arbeit.

In 3.1 bis 3.5 werden RVS, Java RMI, CORBA, IBM WebSphere MQ und Web Services vorgestellt. Anschließend wird in 3.6 ein Vergleich dieser Technologien vorgenommen.

Die Struktur dieser Unterkapitel gliedert sich jeweils wie folgt: Nach einer Einleitung mit einer kurzen Darstellung der Geschichte der jeweiligen Technologie folgen, sofern entsprechende Daten ermittelt werden konnten, einige Zahlen zu ihrer Verbreitung. Anschließend werden der Protokoll-Stapel (d.h. die einzelnen Bestandteile) und die Kommunikation (d.h. das Verhalten oder die Interaktion dieser Bestandteile) erläutert.

Diese Übersicht orientiert sich an Technologien zur Kommunikation. Da aber – wie in Kapitel 1 erläutert – auch der Aspekt der Wiederverwendbarkeit zum Themenbereich der Integration gehört, könnten hier durchaus auch Technologien wie CICS (Customer Information Control System) oder EJBs (Enterprise Java Beans) auftauchen.

3.1 Datei-Transfer am Beispiel RVS

RVS[6] (Rechner-Verbund-System) ist ein von der gedas AG entwickeltes Programm für den Austausch von Dateien [Ged06a]. Die ersten Grundzüge entstanden bereits Anfang der 70er Jahre; folglich ist RVS die älteste der in diesem Kapitel behandelten Technologien.

Ergänzend zu der am Anfang dieses Kapitels erläuterten Struktur werden in 3.1.2 die verfügbaren Varianten von RVS vorgestellt.

3.1.1 Einsatz

Ursprünglich wurde RVS ausschließlich für die Kommunikation innerhalb des Volkswagen-Konzerns eingesetzt; mittlerweile hat es sich jedoch zu einem Standardprodukt entwickelt, das in anderen großen Automobilkonzernen ebenso benutzt wird wie in Banken und Speditionen (von den 20 größten Industrieunternehmen der Bundesrepublik haben 10 RVS installiert).

Nicht zuletzt wird über RVS auch die logistische Anbindung von mehr als 1300 Zulieferfirmen der Volkswagen AG abgewickelt. Die Gesamtanzahl der an RVS angebundenen Partner beträgt derzeit (d.h. 2006) fast 4000. Allein über die RVS-Installation im Stammwerk Wolfsburg laufen täglich ca. 60.000 Dateien mit einem Volumen von ca. 100 Gigabyte[7].

3.1.2 Varianten (Produktportfolio)

RVS ist in unterschiedlichen Varianten erhältlich, die sich zum Teil nur hinsichtlich des unterstützten Betriebssystems, zum Teil aber auch bezüglich ihrer Architektur unterscheiden. Diese Varianten, also das Produktportfolio (so die offizielle Bezeichnung der gedas AG), werden im Folgenden kurz beschrieben. Detailliertere technische Informationen können der gedas-Homepage [Ged06b] entnommen werden.

[6] RVS ist ein eingetragener Markenname der gedas AG. Die offizielle Bezeichnung ist „rvs®“. Im Rahmen dieser Arbeit wird vereinfachend die Bezeichnung „RVS“ verwendet.

[7] Quelle: Volkswagen-Intranet

3.1.2.1 RVS Portabel und RVS MVS

RVS Portabel[8] ist der Sammelbegriff für mehrere RVS-Varianten, die für die Verwendung auf Clients mit unterschiedlichen Betriebssystemen konzipiert sind. Dies sind im Einzelnen:

- rvsX (für Unix)
- rvs400 (für AS/400 und OS/400)
- rvsNT (für Windows NT)
- rvsXP (für ür Windows XP/2000/Server2003)

RVS MVS ist die Version für den Einsatz unter z/OS und OS/390, d.h. auf Großrechnern. Da die unter dem Namen RVS Portabel zusammengefassten Varianten für Desktop-Rechner konzipiert sind, ist die Abgrenzung von RVS MVS durchaus sinnvoll, allerdings wäre auch eine Zusammenfassung von RVS Portabel und RVS MVS in einer übergeordneten Kategorie denkbar, da sie sich architektonisch nicht wesentlich voneinander unterscheiden.

3.1.2.2 RVS Data Center

Mit dem Namen RVS Data Center [Ged05][9] wird eine spezielle Architektur bezeichnet, die auf RVS Portabel bzw. RVS MVS aufbaut (siehe Abbildung 12).

In einem RVS Data Center sind mehrere RVS-Server zusammengefasst. Damit sind mehrere Vorteile verbunden:

- hohe Verfügbarkeit
- Lastverteilung
- Skalierbarkeit (neue Server können im laufenden Betrieb des Data Center hinzugefügt werden)
- Ausfallsicherheit (beim Ausfall eines Servers werden dessen Aufgaben von einem anderen Server übernommen)

Nach außen, d.h. gegenüber Kommunikationspartnern oder bei der Bedienung, verhält sich das RVS Data Center dagegen wie eine einzelne RVS-Instanz.

---

[8] Die deutsche und englische Bezeichnung, d.h. RVS Portabel und RVS portable, wechseln sich auch innerhalb der deutschsprachigen Beschreibungen der gedas AG ohne erkennbaren Grund ab; in dieser Arbeit wird konsistent die Bezeichnung „RVS Portabel“ benutzt.

[9] Der Titel von [Ged05] („RVS portable“) mag zu Verwirrung Anlass geben. Tatsächlich behandelt dieses Dokument im Wesentlichen RVS Portabel, aber auf Seite 22 ff. wird auch das RVS Data Center beschrieben.

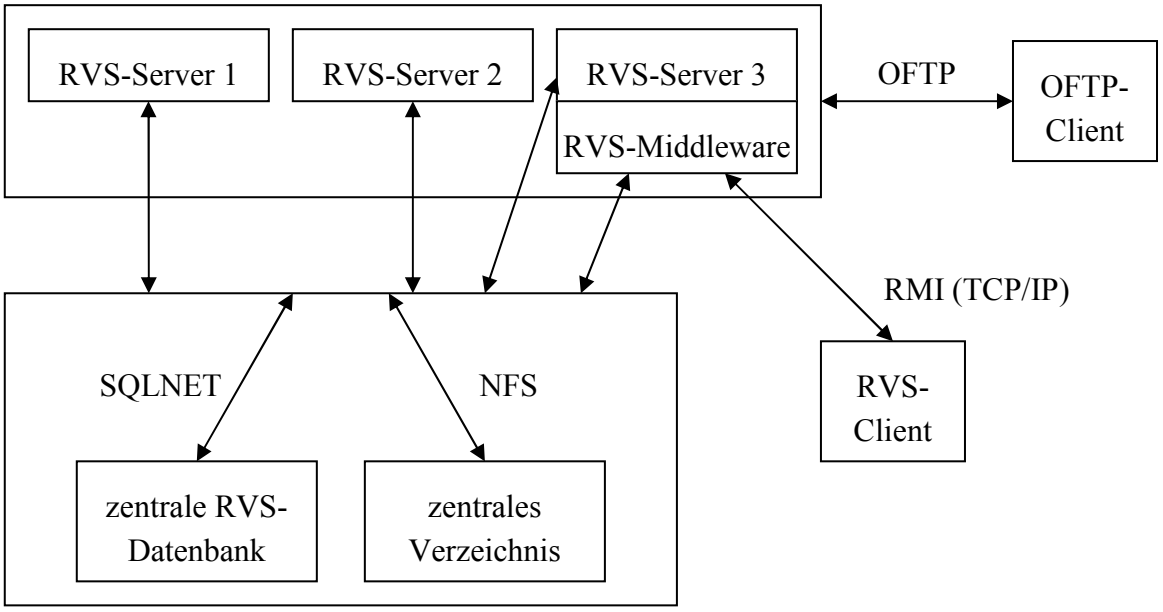

Abbildung 12: Architektur des RVS Data Center [Mül06]

Um die o.g. Vorteile realisieren zu können, sind zwei zentrale Komponenten innerhalb der Architektur des RVS Data Center vorhanden:

- eine zentrale RVS-Datenbank, in der alle Sende- und Empfangsaufträge registriert sind und in der auch Reaktionen auf empfangene Dateien definiert werden können
- das zentrale Verzeichnis, in dem die wichtigsten vom RVS Data Center verwendeten Verzeichnisse enthalten sind (diese Verzeichnisse dienen z.B. zur Speicherung von temporären Dateien, Log-Dateien und Schlüsseln zur Ver- und Entschlüsselung der übertragenen Dateien)

RVS Client/Server ist ein Zusatzmodul, das die entfernte Konfiguration und Wartung des RVS Data Center erlaubt. Hierfür muss auf einem beliebigen Knoten die RVS-Middleware (ebenfalls ein Zusatzmodul) installiert sein, die sich vom Client aus über Java RMI[10] ansprechen lässt.

[10] Eine Beschreibung von Java RMI ist im weiteren Verlauf dieser Arbeit (3.2) zu finden.

3.1.2.3 RVS Tiny

RVS Tiny ist vollständig Java-basiert und somit Betriebssystem-unabhängig. Zur Ausführung wird lediglich ein Java Runtime Environment (JRE) benötigt.

Allerdings ist der Funktionsumfang von RVS Tiny stark abgeschwächt; so ist diese Variante nur in der Lage, über TCP/IP zu kommunizieren. Darüber hinaus stellt RVS Tiny keine Routing-Funktionalität zur Verfügung und benötigt eine zentrale RVS-Instanz zur Kommunikation. Aufgrund dieser Eigenschaften ist eine Verwendung von RVS Tiny lediglich als Endknoten vorgesehen.

3.1.3 Protokoll-Stapel

Die Protokoll-Schichten in einer RVS-Umgebung, wie sie in Abbildung 13 dargestellt sind, können als Schichten unterhalb der Anwendungsschicht, also Schicht 7, des ISO-OSI-Referenzmodells betrachtet werden [Ged05].

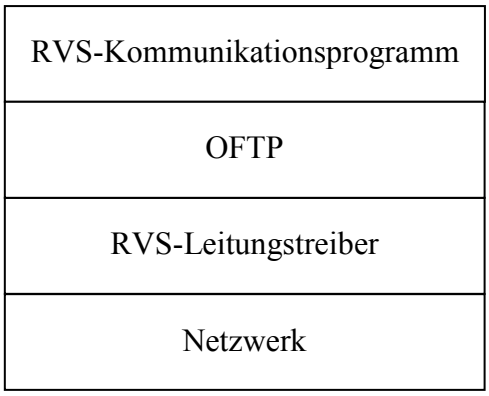

Abbildung 13: Der Protokoll-Stapel von RVS

Die Netzwerk-Schicht regelt die Steuerung und Definition des physikalischen Netzwerks. Diese Schicht ist zwar richtigerweise mit im Protokoll-Stapel aufgeführt, gehört aber nicht direkt zu RVS.

Die Aufgabe des Leitungstreibers ist, eine Ende-zu-Ende-Verbindung mit dem Ziel einer sicheren und nachvollziehbaren Übertragung herzustellen. Jeder Leitungstreiber gehört zu einer spezifischen RVS-Station und kommuniziert mit dem Leitungstreiber der Gegenseite über ein spezielles Leitungstreiber-Protokoll.

Der Leitungstreiber erhält Anfragen von der übergeordneten OFTP-Schicht, eine Verbindung mit der Netzwerk-Schicht herzustellen oder abzubrechen. War dies erfolgreich, leitet der Leitungstreiber der sendenden Station die zu übertragenden Dateien an den Leitungstreiber der empfangenden Station weiter, der sie der lokalen OFTP-Schicht übergibt.

Das ODETTE File Transfer Protocol (OFTP) wurde 1986 im Umfeld der europäischen Automobilindustrie für den elektronischen Datei-Austausch konzipiert. Sein Zweck ist dessen Realisierung ohne Abhängigkeit von der Netzwerk-, Hardware- und Software-Umgebung. Dabei wurden Faktoren berücksichtigt wie z.B. Größenunterschiede in der Datei-Speicherung sowie Systeme unterschiedlichen Alters und unterschiedlicher Hersteller.

Die Aufgabe des OFTP innerhalb des Protokoll-Stapels ist die Sicherstellung einer zuverlässigen Übermittlung der Dateien. Dazu gehören Mechanismen des Quittierens der Übertragung und deren Wiederaufnahme nach eventuellen Leitungsunterbrechungen.

Das OFTP einer RVS-Station eröffnet eine Protokoll-Sitzung mit dem OFTP der entfernten Station, die logisch über der Leitungstreiber-Verbindung läuft. In diesem Zusammenhang realisiert das OFTP auch eine Ver- und Entschlüsselung der übertragenen Dateien.

Detaillierte Informationen hierüber können dem RFC 2204 [Nas97] entnommen sowie im Internet unter www.odette.org nachgelesen werden.

Das Kommunikationsprogramm schließlich kümmert sich um inhaltliche Aspekte der übertragenen Dateien, wie z.B. die Konvertierung zwischen unterschiedlichen Zeichensätzen. Eingehende Informationen werden vom Kommunikationsprogramm in einer temporären Datei zwischengespeichert, und auch Datenformate, die vom lokalen System nicht verstanden werden, können gespeichert und weiter verschickt werden.

3.1.4 Kommunikation

Neben dem in 3.1.3 analysierten Kommunikationsprogramm umfasst RVS intern noch weitere Elemente [Ged05]; die wesentlichen sind folgende:

- Datenbank
- Monitor
- Mastertransmitter

In der RVS-Datenbank sind alle für die Kommunikation benötigten Informationen abgelegt. Diese können sowohl statisch (wie z.B. die sog. Stationstabelle) als auch dynamisch sein, d.h. aktuelle Prozesse, wie z.B. Sendeaufträge, betreffen. Diese Prozesse kommunizieren nicht direkt miteinander, sondern nutzen die Datenbank als Kommunikations-Medium.

Um einen Sendeauftrag anzustoßen, muss ihn die betroffene RVS-Station (bzw. der Benutzer, der sie bedient) in die Datenbank schreiben. Diese wird vom Monitor periodisch auf neue Aufträge durchsucht. Hat dieser einen neuen Sendeauftrag gefunden, aktiviert er den Mastertransmitter. Dessen Hauptaufgabe ist die Verwaltung und Kontrolle paralleler Übertragungsprozesse.

Nach der Aktivierung des Mastertransmitters wird der Sendeauftrag in einem Task an das Kommunikationsprogramm weitergereicht, das den Sendeauftrag letztendlich ausführt. Die einzelnen Bestandteile des Kommunikationsprogrammns wurden in 3.1.3 beschrieben.

3.2 Java Remote Method Invocation (RMI)

RMI [Sun03] ist ein vollständig auf Java basierendes, verteiltes Objektsystem. Es ermöglicht dem Benutzer, aus einem Java-Programm heraus nicht nur eigene, lokale Objekte anzusprechen, sondern auch die Methoden entfernter Objekte zu nutzen. RMI ist der Nachfolger des Anfang der 1980er Jahre entwickelten SUN-RPC [Emm03], um der Objektorientierung gerecht werden zu können.

Die Ursprünge von Java gehen auf das Jahr 1991 zurück [KaroJ]. Die damals unter dem Namen „Green Project“ entstandene objektorientierte Programmiersprache wurde zunächst OAK (Object Application Kernel) genannt. Nach einigen Änderungen wurde OAK im Jahre 1994 in die heutige Bezeichnung Java umbenannt.

3.2.1 Protokoll-Stapel

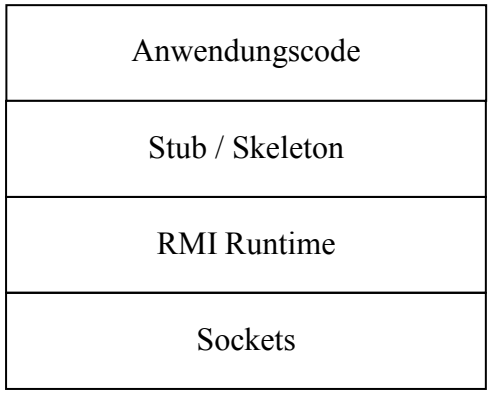

Abbildung 14: Der Protokoll-Stapel von RMI

Der Protokoll-Stapel von RMI, der in Abbildung 14 dargestellt ist, besteht aus den folgenden Teilen:

- **Sockets** zur Realisierung des betriebssystemabhängigen Zugriffs auf das Netzwerk über TCP/IP
- die **RMI Runtime** für die eigentliche Kommunikation (sie sorgt für die Transparenz des entfernten Objektaufrufs)
- **Stub und Skeleton** zum Marshalling bzw. Demarshalling (unter Marshalling versteht man den Vorgang der Serialisierung eines Objekts oder eines entfernten Methoden-Aufrufs zum Verschicken über ein Netzwerk [TS03])

- Aus dem **Anwendungscode** heraus wird nur mit dem Stub bzw. Skeleton kommuniziert. Kenntnisse in Socket-Programmierung sind für einen RMI-Entwickler folglich nicht notwendig.

Das genaue Zusammenspiel dieser Protokoll-Schichten wird im folgenden Kapitel (3.2.2) beschrieben.

3.2.2 Kommunikation

In RMI wird sehr deutlich zwischen einer Client- und einer Serverkomponente unterschieden. Bei RVS (vgl. 3.1) beispielsweise kann eine Kommunikation zwischen zwei identischen Dateisystemen erfolgen; in RMI dagegen können die Rollen des Anbieters und des Aufrufers nicht ohne weiteres getauscht werden.

Zunächst einmal muss ein Server, der ein Objekt für den entfernten Zugriff zur Verfügung stellen möchte, dieses registrieren. Hierzu dient die RMI-Registry; dieses zentrale Verzeichnis ermöglicht das Auffinden von verfügbaren Objekten.

Nach dieser Registrierung kann ein Client das entfernte Objekt auf dem in Abbildung 15 gezeigten Weg aufrufen:

Nachdem der Client eine Suchanfrage für ein Server-Objekt an die RMI-Registry geschickt hat (und dieses Objekt dort auch gefunden wurde), gibt diese ein Stub-Objekt an den Client zurück.

Auf diesem Stub-Objekt führt der Client den entfernten Methoden-Aufruf aus. Der Stub serialisiert ihn und sendet ihn unter Verwendung der RMI Runtime (vgl. 3.2.1) an den Skeleton, der die Deserialisierung und Weitergabe an die Server-Anwendung übernimmt. Stub und Skeleton unterscheiden sich hinsichtlich der beteiligten Klassen nicht wesentlich; entscheidend ist die Zugehörigkeit zum Client (Stub) bzw. Server (Skeleton).

Der Rücktransport des Ergebnisses des entfernten Methodenaufrufs erfolgt auf demselben Weg in umgekehrter Reihenfolge: Der Skeleton serialisiert das Objekt zum Versand an den Client bzw. zunächst an dessen Stub, der die Deserialisierung durchführt.

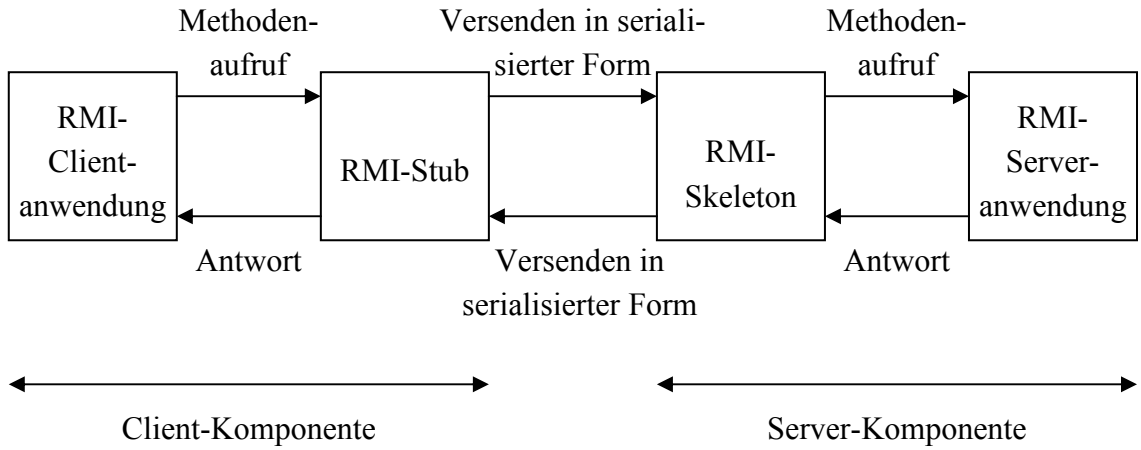

Abbildung 15: Die Kommunikation in einer RMI-Anwendung [Mül06]

3.3 CORBA

CORBA [OMG04][Linn98] ist die Abkürzung für Common Object Request Broker Architecture. Es handelt sich hierbei um kein Produkt, sondern eine Spezifikation für die Integration verteilter Objektsysteme von der Object Management Group (OMG).

Die OMG wurde 1989 als Konsortium von 8 Firmen mit der Zielsetzung gegründet, unter Verwendung von Objekttechnologie die Interoperabilität von Anwendungen in heterogenen, verteilten Umgebungen zu realisieren [Linn98]. Schon bald nach ihrer Gründung veröffentlichte die OMG die erste Version des Referenzmodells Object Management Architecture (OMA), das in den folgenden Jahren immer wieder erweitert und verbessert wurde und aus dem auch CORBA hervorging. Im Dezember 1990 wurde CORBA 1.0 eingeführt; Anfang 1991 folgte die Version 1.1 [Vat04]. Mit CORBA 2.0 folgte 1996 ein Standardprotokoll, das die Interoperabilität von CORBA-Implementierungen verschiedener Hersteller gewährleisten sollte. Die wesentliche Neuerung in CORBA 3.0 (2002) ist das Object-Packaging-Schema für Enterprise Java Beans (EJB), das es ermöglicht, die Virtual Machine zur Laufzeit zu laden und den EJB-Code auszuführen.

Im Folgenden wird die aus den vergangenen Unterkapiteln bekannte Struktur wieder aufgegriffen. Darüber hinaus wird in 3.3.3 kurz auf die CORBA-Dienste eingegangen.

3.3.1 Protokoll-Stapel

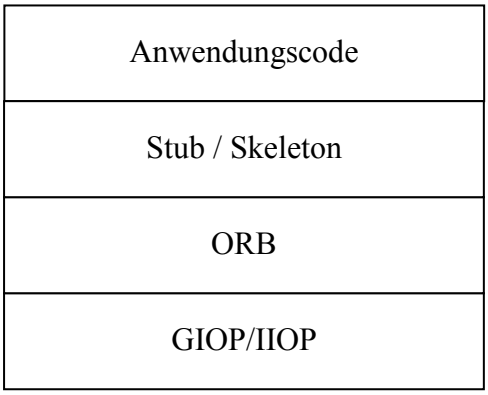

Abbildung 16: Der Protokoll-Stapel von CORBA

Der Protokoll-Stapel von CORBA, der in Abbildung 16 dargestellt ist, ist dem von RMI (vgl. Abbildung 14 in 3.2.1) sehr ähnlich. Tatsächlich ist ja auch das Grundprinzip dieser beiden Technologien, nämlich die Kommunikation zwischen verteilten Objekten, identisch.

Das zentrale Element von CORBA ist der ORB (Object Request Broker). Dessen Aufgabe ist es – analog zur Runtime bei RMI –, die Transparenz des entfernten Methoden-Aufrufs herzustellen, d.h. ihn genauso wie einen lokalen aussehen zu lassen. Im Gegensatz zu RMI müssen die an den ORB angeschlossenen Objekte nicht zwingend in Java implementiert sein.

Die eigentliche Interaktion zwischen Client und Server erfolgt über das GIOP (General Inter-ORB Protocol). Dieses Protokoll adressiert die Bereiche oberhalb des Transportprotokolls und kann folglich mit verschiedenen strombasierten Transportprotokollen zusammenarbeiten. Die Realisierung des GIOP über TCP ist die häufigste Variante und wird mit IIOP (Internet Inter-ORB Protocol) bezeichnet.

3.3.2 Kommunikation

Wie schon der Protokoll-Stapel, so ist auch die Kommunikation in CORBA sehr ähnlich zu der von RMI.

Als erstes muss ein Objekt registriert werden, damit es entfernt aufgerufen und unter einem bestimmten Namen wiedergefunden werden kann. Hierfür ist der CORBA Naming Service zuständig.

Um ein allgemeines Verständnis der Schnittstellen-Syntax zu erreichen, wurde mit der IDL (Interface Definition Language) eine sprachunabhängige Darstellung der Objektschnittstelle

eingeführt. Bei RMI ist dieser Schritt nicht notwendig, da alle Objekte Java-basiert implementiert sind und folglich die Java-Syntax als kleinsten gemeinsamen Nenner verwenden können.

Anschließend kann das Objekt aufgerufen werden, indem – wie in Abbildung 17 gezeigt – ein Client eine entsprechende Anfrage an den ORB richtet und dieser sie an das richtige Server-Objekt weiterleitet.

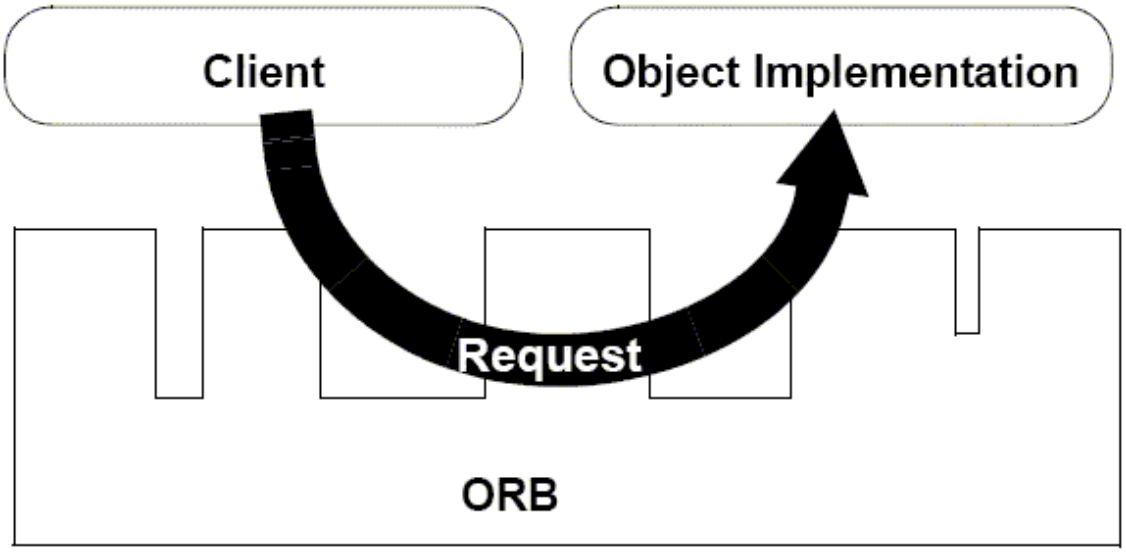

Abbildung 17: Die Kommunikation bei CORBA [OMG04]

3.3.3 CORBA-Dienste

Neben der soeben beschriebenen Kernfunktionalität bietet CORBA zahlreiche ergänzende Dienste an. Der wichtigste ist der bereits erwähnte Naming Service. Zu den weiteren Diensten gehören folgende [Linn98]:

- Concurrency Service (zur Realisierung von nebenläufigen Zugriffen auf ein Objekt, so dass sich dieses immer in einem konsistenten Zustand befindet)
- Event Service (zur Ermöglichung einer asynchronen Kommunikation, wenn der Empfänger nicht verfügbar ist)
- Security Service
- Transaction Service (zur Transaktions-Verwaltung nach dem ACID-Prinzip[11])

Diese Dienste stellen einen großen Vorteil von CORBA dar, da sie in vielen anderen Technologien eigenverantwortlich durch den Benutzer hinzugefügt oder gar selbst implementiert werden müssen.

[11] Atomarität, Konsistenz, Isolation, Dauerhaftigkeit; vgl. z.B. [HS00b]

3.4 Messaging-Middleware am Beispiel IBM WebSphere MQ

IBM WebSphere MQ [IBM06a][Wac99] wurde kommerziell im Jahre 1994 eingeführt. Während es früher „MQSeries“ genannt wurde (und auch heute in der Praxis noch oftmals mit dem Namen verknüpft wird), ist es mittlerweile (seit der Version 5.3) in die WebSphere-Produktfamilie integriert worden. Im weiteren Verlauf dieser Arbeit wird vereinfachend die Bezeichnung „MQ“ verwendet.

MQ ist eine Messaging-Middleware, d.h. das Grundprinzip ist, dass Nachrichten von einer Anwendung zu einer anderen geschickt werden. Genauer gesagt wird nicht direkt mit der Anwendung kommuniziert, sondern die Nachrichten werden in eine Message Queue („Warteschlange“) gelegt, aus der sie dann von der Zielanwendung ausgelesen werden können.

Dieses Prinzip der indirekten physikalischen Kopplung liegt auch bei RVS (vgl. 3.1) vor; hier funktioniert der Informationsaustausch zwar nicht über Nachrichten, sondern über Dateien, aber auch diese werden in einem Verzeichnis abgelegt, und es wird nicht direkt mit der Zielanwendung selbst kommuniziert. Auf die Vorteile dieser Architektur wird noch detailliert in Kapitel 5 eingegangen.

Auch wenn MQ in Reinform als eher ältere Technologie (verglichen z.B. mit Web Services) angesehen werden kann, ist es die Basis für nahezu alle aktuellen Produkte aus dem Hause IBM im Themenbereich des „Enterprise Service Bus“ und damit hilfreich bei der Umsetzung einer Service-orientierten Architektur (mehr hierzu in Kapitel 4).

3.4.1 Protokoll-Stapel

Der Protokoll-Stapel von MQ ist in Abbildung 18 dargestellt und wird im Folgenden erläutert.

Wie eingangs erwähnt, erfolgt die Kommunikation zweier Anwendungen über MQ durch das Schreiben von Nachrichten in eine Message Queue und deren Auslesen durch die Zielanwendung. Für den Zugriff auf die einzelnen Queues ist ein sog. Message Queue Manager (MQM) notwendig. Dieser ist das zentrale Element von WebSphere MQ. Seine wesentlichen Aufgaben sind folgende [Wac99]:

- Verwalten von Queues (d.h. Erstellen oder Löschen) und Nachrichten für Anwendungen
- Nutzung von existierenden Netzwerken für den Transport von Nachrichten zu anderen MQMs (zumeist über TCP/IP, aber es werden auch andere Protokolle unterstützt)
- Segmentierung und Reassemblierung von Nachrichten, die zu groß sind, um als Ganzes transferiert werden zu können

- Zusammenfassung von mehreren kleineren Nachrichten zu einer physikalischen Nachricht, die weniger Netzwerk-Kapazität verbraucht
- Verteilen einer Nachricht an mehrere Empfänger auf Basis eines Publish-and-Subscribe-Verfahrens
- Bereitstellung einer grafischen Benutzungs-Schnittstelle zur Verwaltung des Queues durch Administratoren

Das Message Queue Interface (MQI) realisiert den Zugang zu den einzelnen MQMs für die Anwendungen.

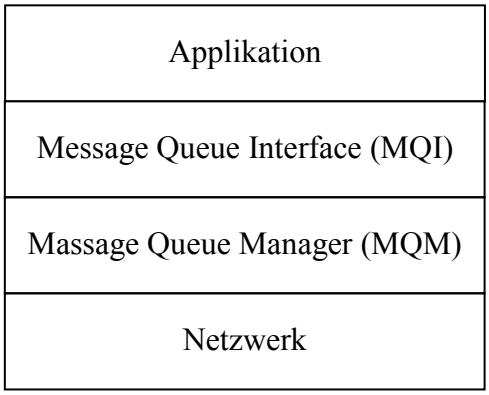

Abbildung 18: Der Protokoll-Stapel von IBM WebSphere MQ

3.4.2 Kommunikation

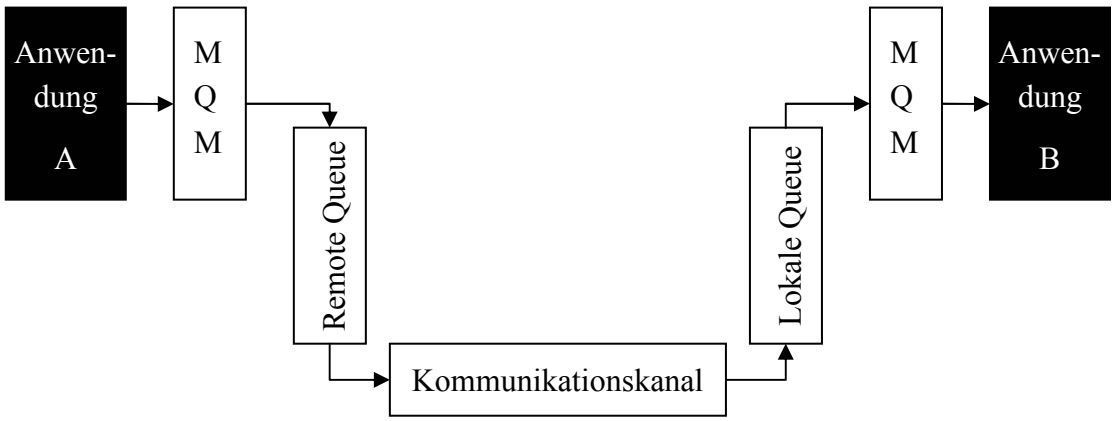

Abbildung 19: Die Kommunikation in IBM WebSphere MQ (in Anlehnung an [Mül06])

Der Verlauf der Kommunikation in MQ wird in Abbildung 19 veranschaulicht. Die sendende Anwendung (im Bild Anwendung A) legt die zu übertragende Nachricht mit Hilfe des zuständigen MQM in die Ausgangs-Queue. Anschließend erfolgt der Transfer über einen Kommunikationskanal zur Eingangs-Queue der Zielanwendung (Anwendung B). In der Grafik

sind die beiden Queues mit „Remote Queue“ und „Lokale Queue“ bezeichnet; dies ist aus der Perspektive der empfangenden Anwendung zu sehen.

Eine MQ-Nachricht besteht aus zwei Teilen, nämlich den eigentlichen Daten und einem Metadaten-Teil, der Header oder Distributor genannt wird [Wac99]. Mit dessen Hilfe können die Nachrichten über eine MessageID identifiziert werden; außerdem enthält er Steuerungs-Informationen (auch Attribute genannt), wie z.B. ein Ablaufdatum, die Priorität, den Namen der Queue, zu der die Antwort geschickt werden soll, und den Nachrichten-Typ.

In MQ werden vier Nachrichten-Typen unterschieden:

- Datagram (eine Nachricht, für die keine Antwort benötigt wird)
- Request (eine Nachricht, für die eine Antwort erwartet wird)
- Reply (eine Antwort auf eine Request-Nachricht)
- Report (z.B. eine Bestätigung für den Erhalt einer Nachricht)

Durch die Kombination von Request- und Reply-Nachrichten kann nach außen eine synchrone Kommunikation realisiert werden, auch wenn diese natürlich MQ-intern weiterhin asynchron erfolgt und im Gegensatz beispielsweise zu RMI (vgl. 3.2) die Antwort über eine andere Verbindung und einen anderen Kommunikationskanal gesendet werden kann als die Anfrage.

3.5 Web Services

Web Services [EF03] stellen derzeit das Ende der Evolution der Integrations-Technologien dar. Das W3C (World Wide Web Consortium) definiert einen Web Service folgendermaßen:

> *„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“ [W3C04]*

Das Grundprinzip ist also eine Zusammenarbeit von Software auf verschiedenen Rechnern über ein Netzwerk. Das ist allerdings bei den anderen im Verlauf dieses Kapitels vorgestellten Technologien nicht grundlegend anders. Das Besondere an Web Services sind jedoch u.a. folgende Tatsachen [EF03]:

- Web Services genießen eine breite Unterstützung aus der Industrie, und ihre Entwicklung wird auch z.B. von Microsoft, IBM oder Sun aktiv vorangetrieben.
- Web Services basieren auf offenen und allgegenwärtigen Internet-Protokollen, so dass sie zum einen ohne Lizenzkosten eingesetzt werden können und zum anderen auf sehr

vielen Systemen ohne größere Umstellung umgesetzt werden können, da keine neue Infrastruktur aufgebaut werden muss.

Die restlichen, in der Definition vorkommenden Begriffe werden im Rahmen der Vorstellung des Protokoll-Stapels (3.5.1) noch näher erläutert.

3.5.1 Protokoll-Stapel

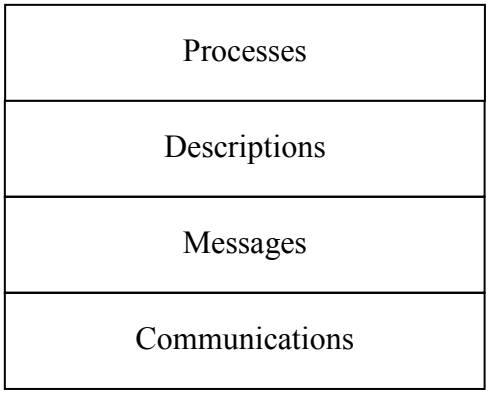

Abbildung 20: Der Protokoll-Stapel von Web Services

Der Protokoll-Stapel von Web Services, der in Abbildung 20 zu sehen ist, enthält die folgenden Schichten (vgl. auch [Boo+04]):

- **Communications**: Da Web Services unabhängig von der Netzwerk- und Transportschicht sind, kann diese Ebene allgemein nicht präziser festgelegt werden. Zur Umsetzung sind beispielsweise HTTP, SMTP oder FTP, aber auch JMS (Java Messaging Service) oder IIOP (Internet Inter-ORB-Protocol; vgl. 3.3.1) einsetzbar.
- **Messages**: Zur Übermittlung von Nachrichten zwischen Web Services wird das Simple Object Access Protocol (SOAP) eingesetzt. Dieses ist vollständig XML-basiert. Eine SOAP-Nachricht besteht aus einem (optionalen) Header, der Metadaten (wie z.B. Transaktions-Nummern) enthält, und dem Body, in dem die eigentliche Nachricht codiert ist. Header und Body sind im Envelope eingeschlossen, der signalisiert, dass es sich bei dem XML-Dokument um eine SOAP-Nachricht handelt. Darüber hinaus kann der Envelope einen Verweis auf eine XML-Schema-Datei enthalten, gegen die die Nachricht validiert werden kann.

  SOAP-Nachrichten können auf zwei unterschiedliche Arten verfasst werden: Entweder wird eine Schnittstelle samt den zugehörigen Parametern codiert (RPC-Stil), oder im Body wird ein Geschäftsobjekt übergeben (Dokument-Stil). Eine ausführliche Beschreibung dieser Stile – auch unabhängig von Web Services – folgt in Kapitel 5.

  Neben dem „reinen“ SOAP existieren noch zahlreiche Erweiterungen, die sogenannten SOAP-Extensions. Diese ergänzen SOAP um Aspekte wie Zuverlässigkeit, Transaktionsverwaltung oder Sicherheit.

- **Descriptions**: Analog zur IDL bei CORBA (vgl. 3.3.2) gibt es auch bei Web Services eine Sprache zur einheitlichen und unabhängigen Beschreibung der Schnittstelle, die „Web Service Description Language“ (WSDL). Die Erstellung einer passenden WSDL-Datei ist jedoch für die erfolgreiche SOAP-basierte Kommunikation nicht zwingend erforderlich, sondern stellt nur eine Hilfe dar, um Schnittstellen zu beschreiben.
- **Processes**: In dieser Schicht sind alle Mechanismen zu finden, die beispielsweise mit dem Zusammenfügen (Orchestrieren) von verschiedenen Web Services zu einem Prozess zu tun haben. Ein wichtiger Bestandteil auf dieser Ebene ist auch UDDI (Universal Description, Discovery and Integration), ein Verzeichnis-Dienst, mit dessen Hilfe Web Services prinzipiell weltweit gefunden werden können. Ähnlich wie WSDL ist aber auch UDDI keine zwingende Voraussetzung für eine Kommunikation zwischen Web Services.

3.5.2 Kommunikation

Wie in 3.5.1 vorgestellt, können für den Transport von SOAP-Nachrichten verschiedene Protokolle eingesetzt werden. Die Zusammenarbeit zwischen SOAP und einem konkreten Transportprotokoll wird als Bindung bezeichnet.

Das am häufigsten für den Transport verwendete Protokoll ist HTTP[12]. Die Kommunikation zwischen Client und Server (d.h. Service) kann dabei auf zwei verschiedenen Wegen erfolgen [EF03]:

Eine Möglichkeit ist, eine SOAP-Nachricht mit Hilfe der HTTP-POST-Methode an den Service zu senden, worauf dieser mit einer HTTP-Response antwortet, in der die SOAP-Antwort abgelegt ist.

Die in der Praxis insbesondere bei Verwendung des RPC-Stils weitaus häufiger vorkommende Variante ist jedoch die, dass auf dem Hinweg gar kein SOAP verwendet, sondern eine HTTP-GET-Anfrage verschickt wird. Der Web Server erkennt, dass es sich um den Aufruf eines Web Service handelt, und leitet die Anfrage entsprechend weiter. Als Antwort erhält der Client wiederum eine HTTP-Response mit der Antwort in SOAP.

Wie in 3.5.1 beschrieben, ist es für die Nutzung eines Web Service prinzipiell ausreichend, eine passende SOAP-Nachricht zu übermitteln. Das setzt jedoch voraus, dass zum einen der Web Service als socher, d.h. seine Existenz und Funktionalität, und zum anderen die erwartete Syntax der Anfrage bekannt sind. Zum Ermitteln dieser Informationen können UDDI und WSDL eingesetzt werden.

[12] Hypertext Transfer Protocol

3.6 Vergleich der vorgestellten Technologien

Zum Abschluss dieses Technologie-Kapitels werden nun die vorgestellten Technologien bzw. Technologie-Familien miteinander verglichen. Dabei wird in 3.6.1 zunächst auf den allgemeinen Aufbau und die Bestandteile eingegangen, bevor in 3.6.2 und 3.6.3 die Performance und Sicherheit analysiert werden. Diese Erkenntnisse beruhen auf einer hierzu durchgeführten Diplomarbeit [Mül06].

3.6.1 Aufbau und Bestandteile

Im Folgenden werden die in diesem Kapitel behandelten Technologien bezüglich mehrerer Kriterien in tabellarischer Form miteinander verglichen.

	Technologietyp	**Kommunikations-form**	**Serialisierungs-protokoll**
RVS	Produkt	asynchron	OFTP
Java RMI	Produkt	synchron	RMI Wire Protocol
CORBA	Spezifikation	synchron und asynchron	GIOP / IIOP
WebSphere MQ	Produkt	asynchron	proprietär
Web Services	Spezifikation	synchron und asynchron	SOAP, darunter z.B. HTTP
	Schnittstellen-beschreibung	**Namensdienst**	**Verteiltes Objektsystem**
RVS	–	–	nein
Java RMI	Java	RMI Registry	ja
CORBA	IDL	CORBA Naming Service	ja
WebSphere MQ	–	MQM	nein
Web Services	WSDL	UDDI	nein

Tabelle 2: Vergleich verschiedener Integrations-Technologien (nach [Mül06])

3.6.2 Performance

Ein wesentliches Merkmal einer jeden Integrations-Technologie ist ihre Performance, d.h. die Übertragungs-Geschwindigkeit oder die Zeit, die benötigt wird, um eine bestimmte Datenmenge von einem Sender zu einem Empfänger zu transportieren.

3.6.2.1 Metriken und Möglichkeiten zur Messung

Um eine aussagekräftige Messung zu ermöglichen, ist es zunächst erforderlich, eine geeignete Metrik zu definieren, die bei allen betrachteten Technologien gleichermaßen anwendbar ist.

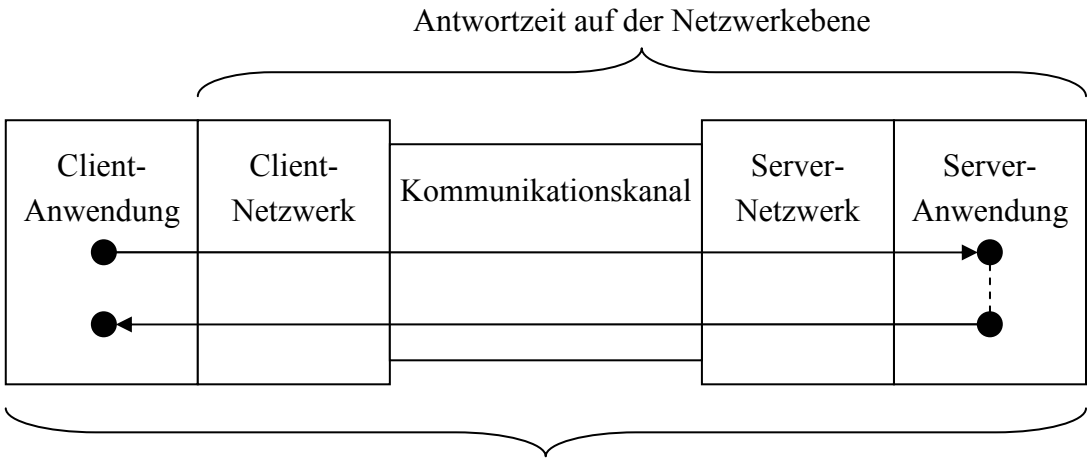

Abbildung 21: Antwortzeit auf der Netzwerk- und Anwendungsebene (in Anlehnung an [Mül06])

Den Kern eines solchen Performance-Vergleichs stellt zweifellos die Antwortzeit dar. Es zeigt sich jedoch, dass dieser Begriff noch viel zu undifferenziert ist. Denkbar wäre eine Messung der Antwortzeit auf der Anwendungsebene (d.h. beispielsweise die Zeit vom Aufruf einer Methode bis zu deren vollständiger Abarbeitung). Während dies bei Java RMI oder CORBA natürlich problemlos möglich wäre, lässt sich bei RVS keine entsprechende Messung durchführen, da die Antwort (in Form einer Datei) nicht wieder auf die Anwendungsebene zurückkehrt.

Abbildung 22: Antwortzeit und ihre Bestandteile (vgl. [MA02])

Daher ist es sinnvoller, die Messung der Antwortzeit auf der Netzwerkebene durchzuführen (siehe Abbildung 21). Dies ist nicht nur ein geeigneter „kleinster gemeinsamer Nenner“ für alle Technologien, sondern es ergeben sich auch weitere interessante Möglichkeiten der Untergliederung. So kann analysiert werden, wie viel Zeit die Übertragung (Netzwerkzeit) und wie viel die Abarbeitung (Residenzzeit) benötigt. Diese beiden Teilzeiten lassen sich – wie Abbildung 22 zu entnehmen ist – nochmals unterteilen in die Verzögerungs- und Übertragungszeit bzw. Service- und Wartezeit (siehe hierzu im Detail [MA02]).

Durch Analyse des Netzwerkverkehrs kann ebenfalls untersucht werden, wie viel Zeit für die eigentliche Übertragung verbraucht wird und wie viel für den Verbindungsauf- und -abbau.

Wie in 3.6.1 vorgestellt, ist mit einigen der Technologien nur eine synchrone bzw. asynchrone Kommunikation möglich. Da mit einer synchronen Technologie keine asynchrone Kommunikation simuliert werden kann, umgekehrt aber problemlos (durch zwei asynchrone Übertragungen hintereinander), dient eine synchrone Anfrage als Basis für die Messungen.

3.6.2.2 Ergebnisse des Performance-Vergleichs

Im Folgenden werden die Ergebnisse des im Rahmen der o.g. Diplomarbeit durchgeführten Performance-Vergleichs präsentiert.

Neben den in diesem Kapitel vorgestellten und in 3.6.1 grundlegend verglichenen Technologien wurden auch die Möglichkeiten der Datenintegration, d.h. Daten-Replikation und -Föderation, auf Basis eines mySQL-Datenbanksystems mit berücksichtigt. Nähere Informationen zur Datenintegration folgen in Kapitel 4.

Zudem wurden bei den Web Services die beiden in 3.5 skizzierten Stile (RPC- und Dokument-Stil) getrennt betrachtet, um zu untersuchen, ob deren Wahl einen Einfluss auf die Performance hat.

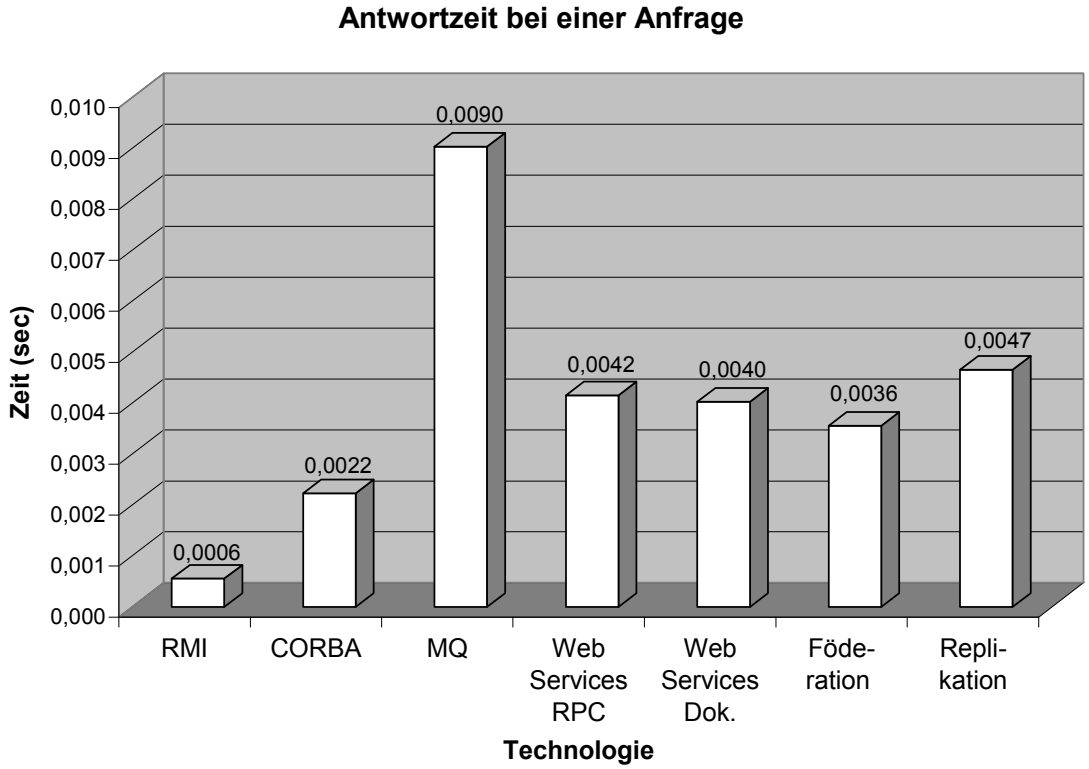

Abbildung 23: Performance-Vergleich (nach [Mül06])

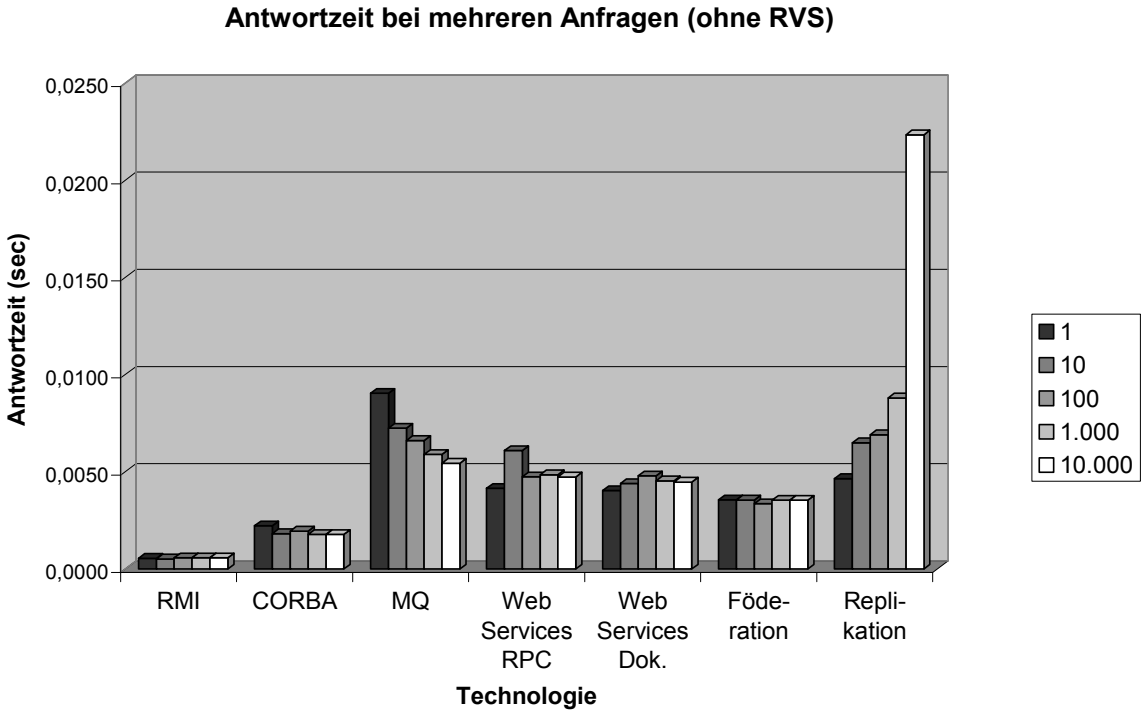

Abbildung 24: Vergleich der durchschnittlichen Antwortzeiten für mehrere Anfragen (nach [Mül06])

Die erste Grafik (Abbildung 23) zeigt die Antwortzeiten bei einer einzigen Anfrage (inklusive Antwort). Es zeigt sich, das RMI deutliche Vorteile gegenüber allen anderen Technologien besitzt. Bei den beiden Web-Service-Stilen erwies sich der Dokumenten-Stil als minimal schneller. Die beiden Varianten der Datenintegration liegen ebenfalls in etwa auf diesem Niveau; die Antwortzeit von MQ ist etwa doppelt so hoch. Nicht enthalten in dieser Grafik ist RVS, denn diese Technologie zeigte sich mit 6,15 Sekunden als die bei weitem langsamste und hätte den grafischen Vergleich der anderen Technologien untereinander nicht mehr ermöglicht.

Neben dieser Messung einer Anfrage wurde ebenfalls untersucht, wie sich die betrachteten Technologien bei mehreren, zeitlich aufeinander folgenden Anfragen verhalten. Abbildung 24 zeigt die Ergebnisse, d.h. die durchschnittlichen Antwortzeiten, für 10, 100, 1000 und 10000 Anfragen.

Es wird deutlich, dass sich bei RMI, CORBA, Daten-Föderation und beiden Web-Service-Stilen praktisch keine Veränderungen der durchschnittlichen Antwortzeit ergeben und dass folglich die Niveau-Unterschiede von Abbildung 23 hier weiterhin Gültigkeit haben. MQ dagegen zeigt eine sinkende Kurve, d.h. mit steigender Anzahl von Anfragen wird die durchschnittliche Antwortzeit geringer. Bei der Daten-Replikation wiederum zeigt sich ein deutlicher umgekehrter Trend.

Zusätzlich zur eigentlichen Performance-Messung wurde auch der benötigte Netzwerkverkehr aufgezeichnet. Wie in Abbildung 25 ersichtlich ist, zeigen bei der Messung einer einzigen Anfrage MQ und vor allem die Daten-Replikation einen deutlich überdurchschnittlichen Wert. Letzterer ist damit zu erklären, dass auch bei der Abfrage von sehr geringen Datenmengen zunächst die komplette Datenbank-Tabelle repliziert wird.

In Abbildung 26 wird deutlich, dass bereits bei 10 Anfragen der Wert der Daten-Replikation wieder zumindest auf dem Niveau von MQ, bei einer noch höheren Anfragen-Anzahl sogar unter dem von RVS, Daten-Föderation und Web Services liegt. Der Unterschied ist in Wirklichkeit noch höher, als er in dieser Grafik zum Ausdruck kommt, da aus Skalierungsgründen die erste Säule bei 11.000 Bytes abgeschnitten wurde (tatsächlich sind es über 26.000). RMI, CORBA und RVS zeigen ebenfalls einen deutlichen Rückgang des durchschnittlichen Netzwerkverkehrs bei steigender Anzahl von Anfragen, während die Werte von Web Services nahezu konstant bleiben. Der Grund hierfür ist, dass für jeden Aufruf eine Initialisierung und ein Reset der genutzten Netzwerkverbindungen durchgeführt werden [Mül06].

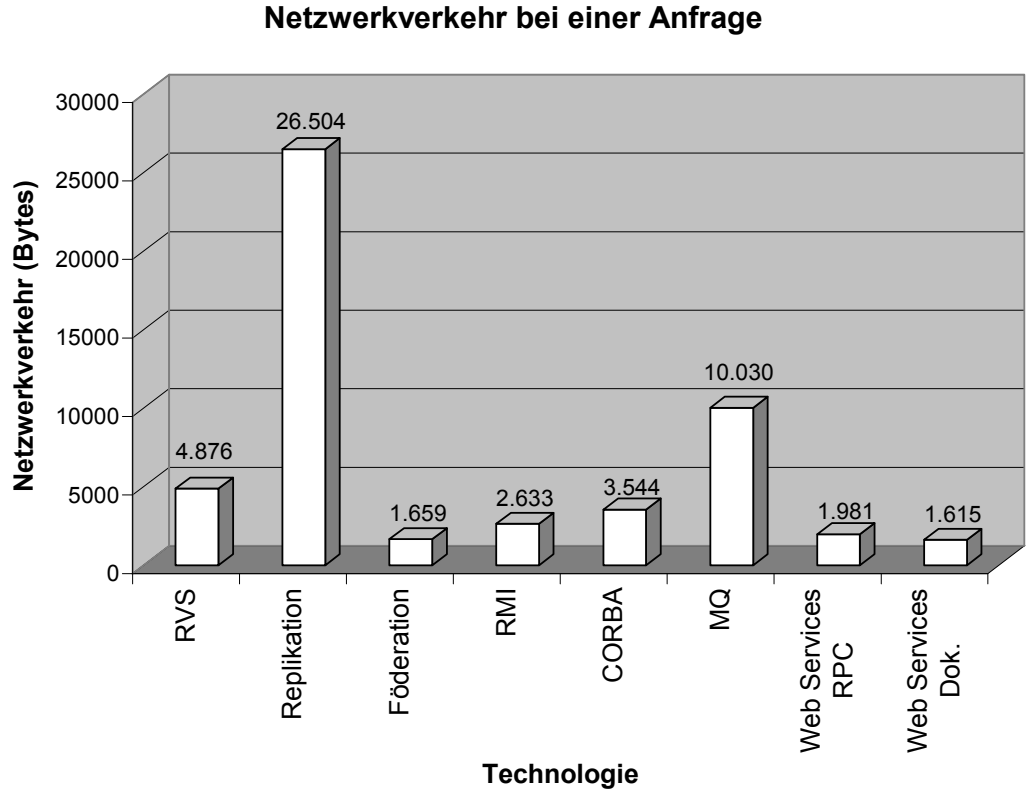

Abbildung 25: Netzwerkverkehr bei einer Anfrage (nach [Mül06])

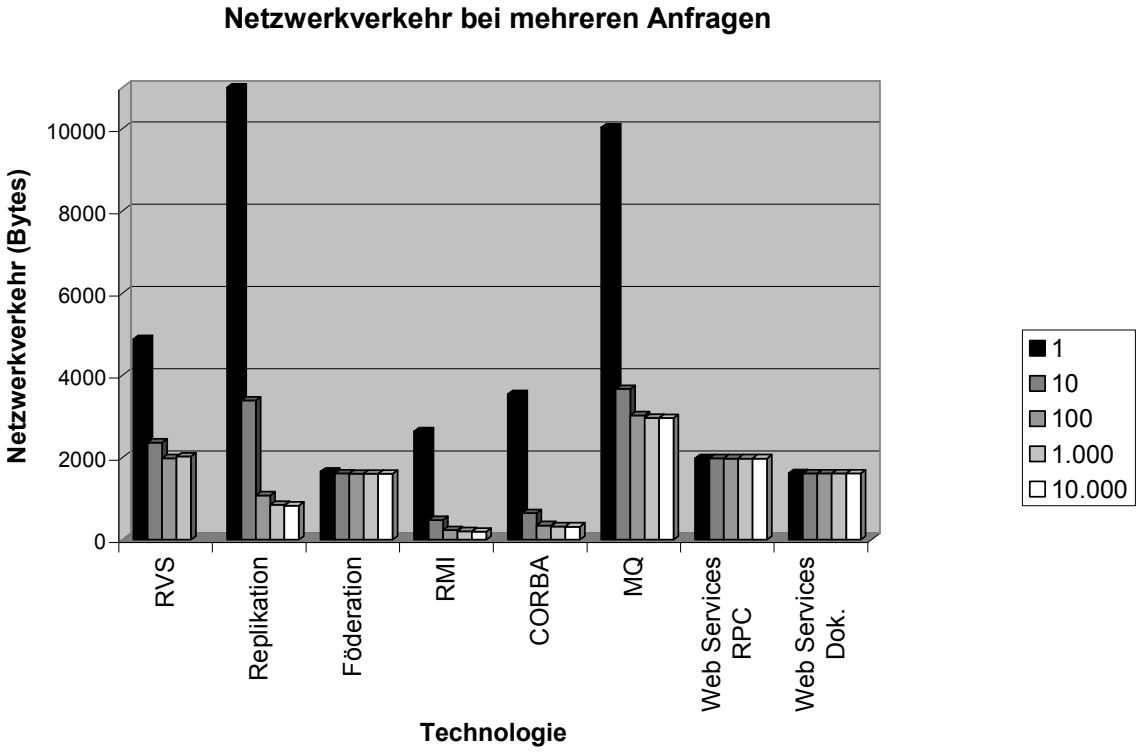

Abbildung 26: Netzwerkverkehr bei mehreren Anfragen (nach [Mül06])

3.6.3 Sicherheit

Der Begriff der Sicherheit unterteilt sich im Kontext der Informatik in die folgenden Bestandteile [EF03]:

- **Vertraulichkeit:**

  Vertraulichkeit bedeutet, dass eine Nachricht, die zwischen einem Sender und einem Empfänger ausgetauscht wird, durch keine Dritten gelesen werden kann. Dieses Lesen bezieht sich auf den Inhalt; die Nachricht an sich (d.h. die Übertragung) muss dabei nicht zwingend geheim gehalten werden.

- **Authentizität:**

  Im Bereich der digitalen Kommunikation ist es keineswegs selbstverständlich, dass der Kommunikationspartner tatsächlich derjenige ist, für den er sich ausgibt. Durch Authentifizierung (d.h. das Überprüfen der Authentizität) soll ein solches Vortäuschen einer falschen Identität und damit der Empfang von Nachrichten, die eigentlich für einen anderen Empfänger bestimmt sind, verhindert werden.

  Hierdurch können insbesondere dann Probleme entstehen, wenn es um die **Autorisierung**, d.h. die Zuteilung von Rechten, geht. Gelingt es einem Angreifer, eine falsche Identität vorzutäuschen, so kann er über die Rollenzugehörigkeit Rechte erhalten, die ihm nicht zustehen.

- **Integrität:**

  Die Sicherstellung der Integrität einer Nachricht bedeutet, dass sie genauso, wie sie beim Empfänger ankommt, auch vom Sender abgeschickt und folglich nicht durch Dritte verändert worden ist.

- **Nicht-Anfechtbarkeit:**

  Im realen Leben wird beispielsweise durch die Unterschrift unter einem Kaufvertrag zweifelsfrei nachgewiesen, dass der Käufer ein Produkt bestellt hat und folglich zu dessen Abnahme und Bezahlung verpflichtet ist. Im digitalen Bereich ist dies natürlich nicht so einfach möglich; hier müssen andere Mechanismen implementiert werden, um zu beweisen, dass eine bestimmte Aktion durchgeführt wurde.

- **Verfügbarkeit:**

  Dieser Punkt adressiert die Gefahr, dass ein Dienst durch Angriffe von außen nicht mehr verfügbar ist. Das ist natürlich physisch durch die Zerstörung des Rechners möglich, aber auch virtuell durch sog. Denial-of-Service-Attacken.

In Tabelle 3 wird die Umsetzung der soeben vorgestellten Sicherheitsbestandteile durch die in diesem Kapitel behandelten Technologien sowie der Daten-Föderation und

-Replikation vorgestellt. Die Autorisierung ist ergänzend zur Authentizität als eigener Punkt berücksichtigt.

Im Folgenden sind hierzu einige kurze Erläuterungen zu finden. Detailliertere Informationen können in [Mül06] nachgelesen werden. Es sei an dieser Stelle ausdrücklich erwähnt, dass in der Tabelle nur diejenigen Technologien bei einem Sicherheitsbestandteil markiert worden sind, die diesen nativ unterstützen. Durch Drittanbieter-Tools oder Eigenentwicklungen sind Erweiterungen der Sicherheit einzelner Technologien möglich.

	Java RMI	CORBA	Web Services	Web-Sphere MQ	RVS	Föderation/Replikation
Vertraulichkeit	X	X	X	X	X	X
Authentizität	X	X	X	X	X	X
Autorisierung	X	X	-	-	-	X
Integrität	X	X	X	X	X	X
Nicht-Anfechtbarkeit	-	X	X	-	-	-
Verfügbarkeit	-	-	-	-	-	-

Tabelle 3: Sicherheitsmechanismen von Integrations-Technologien [Mül06]

Java RMI selbst hat kein eigenes Sicherheitskonzept, sondern bedient sich der durch die Programmiersprache Java zur Verfügung gestellten Mechanismen. Hier ist das sog. Sandkastenprinzip zu nennen: Der Code wird in einer sicheren Umgebung ausgeführt, wodurch z.B. kein Stapelüberlauf möglich ist. Darüber hinaus beschränken die Schlüsselwörter public, protected und private die Zugriffsmöglichkeiten von anderen Teilen des Codes. Die Authentizität kann mittels JSSE (Java Secure Socket Extension), die Autorisierung über JAAS (Java Authentication and Authorization Service) hergestellt werden.

In CORBA werden die o.g. Sicherheitsmechanismen durch den Security Service bereitgestellt. Dieser ist allerdings kein Teil der Standardspezifikation und daher nicht zwingend in allen CORBA-Implementierungen vorhanden.

Die Sicherheit von Web Services ist zum gegenwärtigen Zeitpunkt noch nicht vollständig ausgereift. Lediglich WS-Security ist als Standard verabschiedet; die anderen sicherheitsrelevanten Standards (WS-SecureConversation, WS-Federation, WS-Authorization, WS-Policy, WS-Trust, WS-Privacy) sind zwar zum Teil schon bedeutend vorangeschritten, aber eben noch nicht endgültig abgeschlossen.

Für WebSphere MQ gibt es kein umfassendes Sicherheitskonzept. Die Sicherheit umfasst hier im Wesentlichen zwei Punkte: Zum einen kann (mit einigen Einschränkungen) SSL[13] zur Sicherung der Kommunikation auf der Transportebene eingesetzt werden, zum anderen bietet der Object Authority Manager (OAM) die Möglichkeit, bestimmten Gruppen den Zugriff auf Queues zu gewähren oder zu verweigern. Allerdings wird die Identität des Zugreifenden und damit die Korrektheit der Gruppenzugehörigkeit nicht überprüft, so dass eine Autorisierung nicht vollständig gewährleistet ist.

Auch RVS bietet kein umfassendes Sicherheitskonzept. Das OFTP bietet die Möglichkeit der Nutzung von Benutzernamen und Passwörtern beim Übertragen von Dateien. Die OFTP-Spezifikation schlägt die Verschlüsselung der Dateien unterhalb des Protokolls vor, in RVS wird die Verschlüsselung aber oberhalb vorgenommen, und zwar mittels einer NxN-PKI[14]. Dabei werden die Schlüssel dezentral verwaltet, so dass jeder den öffentlichen Schlüssel des Partners besitzen muss, mit dem er kommunizieren möchte.

[13] Secure Socket Layer

[14] Public Key Infrastructure

Kapitel 4: Integrations-Konzepte

Ohne die in Kapitel 3 vorgestellten Technologien ist eine Integration von Anwendungen nicht möglich. Aber die Technologie alleine vermag allenfalls einzelne proprietäre Verbindungen, nicht aber eine ganzheitliche und durchdachte Integrations-Strategie zu erzeugen. Daher werden – unter Anwendung der verschiedenen Technologien – Integrations-Konzepte benötigt. Dies ist Gegenstand dieses Kapitels.

Es handelt sich folglich um die erste der in 2.5 vorgestellten drei Dimensionen der Anwendungs-Integration. Die dort stichwortartig aufgeführten vier Kategorien Datenintegration, Punkt-zu-Punkt-Verbindungen, Enterprise Application Integration (EAI) und Service-orientierte Architektur (SOA) werden in den Unterkapiteln 4.1 bis 4.4 nun detailliert betrachtet.

4.1 Datenintegration

Wie bereits im Rahmen der Vorstellung der drei Dimensionen der Anwendungs-Integration in 2.5.1 angeschnitten, wird Datenintegration in dieser Arbeit als ein Integrations-Konzept verstanden, das ausschließlich die Datenhaltungs-Schicht der Anwendungen anspricht.

Die organisatorische Ausgestaltung ist in mehreren Varianten denkbar. Beispielsweise könnte vereinbart werden, dass jeder Datenbank-Zugriff individuell vom Support-Team, das die zugehörige Anwendung betreut, genehmigt werden muss.

Es ist aber auch das andere Extrem möglich, nämlich dass die Quellanwendung (bzw. das betreuende Team) nicht einmal von der Datenintegration als solches, geschweige denn von den einzelnen Zugriffen weiß.

Für die technische Realisierung einer Datenintegration gibt es zwei Varianten, und zwar die Daten-Replikation und die Daten-Föderation. Diese werden in 4.1.1 und 4.1.2 vorgestellt.

Darüber hinaus existieren noch weitere Möglichkeiten, Anwendungen unter Zuhilfenahme eines Datenbank-Zugriffs zu integrieren. Welche Methoden hier denkbar sind und warum diese keine Datenintegration im o.g. Sinne darstellen, wird in 4.1.3 diskutiert.

Das Konzept des Data Warehousing (DWH) ähnelt dem der Datenintegration. An welchen Stellen tatsächlich Gemeinsamkeiten vorhanden sind und in welcher Hinsicht sich diese Konzepte dennoch unterscheiden, diese Fragen werden zumindest oberflächlich in 4.1.4 beantwortet.

4.1.1 Daten-Replikation

Nach der Vorstellung des Prinzips der Daten-Replikation sowie einer möglichen weiteren Unterteilung in 4.1.1.1 befassen sich 4.1.1.2 und 4.1.1.3 mit den Vor- und Nachteilen dieser Integrations-Methode.

Auf Basis dieser Informationen zeigt 4.1.1.4 sinnvolle Einsatz-Szenarien und Restriktionen.

4.1.1.1 Das Prinzip

Daten-Replikation bedeutet, dass bestimmte Datensätze aus einer Datenbank herauskopiert und in eine andere Datenbank eingefügt werden. Dieses Prinzip wird in Abbildung 27 veranschaulicht.

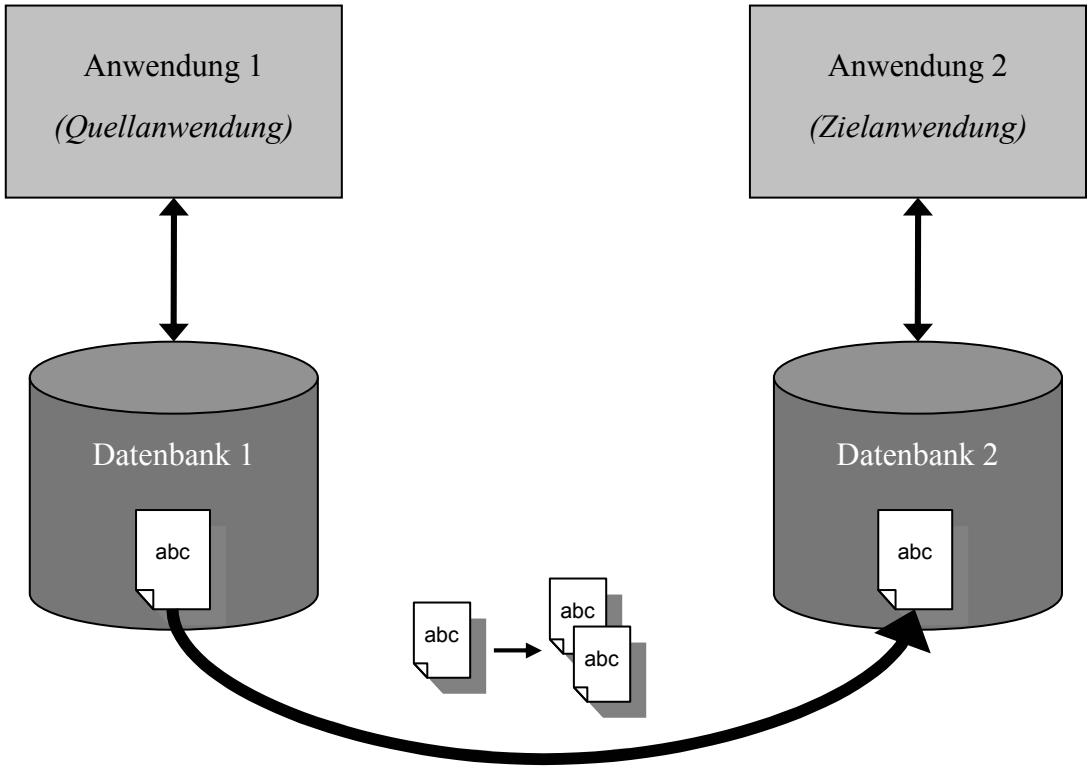

Abbildung 27: Daten-Replikation

Yuhanna, ein Analyst von Forrester Research, unterscheidet die folgenden Möglichkeiten zur Umsetzung dieses Prinzips (unter der Bezeichnung „database replication architectures“) [Yuh05]:

- Trigger-basierte Replikation

  Dies ist Yuhanna zufolge die am häufigsten benutzte Form der Daten-Replikation und wird von den meisten Datenbankmanagement-Systemen (DBMS), darunter Oracle, IBM DB2 und Microsoft SQL Server, unterstützt.

  Sowohl neue als auch geänderte Daten werden entweder sofort synchron repliziert oder in einer „staging table“ für die spätere asynchrone Verteilung zwischengespeichert.

  Der große Nachteil der Trigger-basierten Replikation ist ihr hoher Verbrauch an Rechenleistung, der zu einer geringeren Antwortzeit bei anderen Datenbank-Anfragen führen kann. Deshalb ist diese Art der Daten-Replikation primär für selektive Replikationen (also z.B. eine bestimmte Auswahl an Spalten) strukturierter Daten geeignet.

- SQL-basierte Replikation

  Diese Art der Daten-Replikation ist grundsätzlich asynchron und funktioniert durch regelmäßig ausgeführte SQL-Statements, die Daten aus bestimmten Tabellen extrahieren und entweder in unstrukturierte Dateien oder gleich direkt in eine andere Datenbank schreiben.

  Die SQL-basierte Replikation bietet eine Möglichkeit des schnellen Datenaustausches, hat aber den Nachteil, dass die Einrichtung durch das direkte Schreiben von SQL-Code aufwändig werden kann. Diese Tatsache hat auch einen erheblichen System-Overhead zur Folge. Zudem können diese SQL-Statements nur so lange korrekt arbeiten, wie das Schema der Quell-Datenbank nicht verändert wird.

- Log-basierte Replikation

  Bei der Log-basierten Replikation werden nicht die Daten selbst, sondern die eine abgeschlossene Transaktion dokumentierenden Logs zu einer entfernten Datenbank übermittelt, so dass diese Transaktion ebenfalls auf dem dortigen Datenbestand durchgeführt werden kann. Die Durchführung erfolgt zumeist asynchron.

  Log-basierte Replikation wird bevorzugt für Wiederherstellungs-Transaktionen nach Fehlern („disaster recovery“), aber auch im Bereich von Reporting und Backup eingesetzt.

- Tablespace-basierte Replikation

  Diese relativ neue Art der Daten-Replikation ist der Log-basierten Replikation sehr ähnlich, legt den Fokus allerdings nicht auf die Übertragung einer ganzen Datenbank bzw. deren Änderungen, sondern lediglich auf einen Ausschnitt in Form eines Tablespace. Daher ist diese Methode besonders bei sehr großen Datenbanken geeignet.

4.1.1.2 Vorteile

Der große Vorteil der Daten-Replikation ist die Tatsache, dass dies die einzige Variante der Anwendungs-Integration ist, bei der weder die Quell- noch die Zielanwendung verändert werden muss.

Die Quellanwendung muss hierfür nichts weiter tun als die Daten in die eigene Datenbank zu schreiben – ein Vorgang, der seit jeher unabhängig von jeglichen Integrationsgedanken durchgeführt wird und für den die Anwendung folglich in keinster Weise umprogrammiert werden muss.

Die Zielanwendung wiederum muss lediglich Daten aus ihrer eigenen Datenbank lesen – auch das war und ist stets möglich. Der durch die Integration bedingte Unterschied ist lediglich der, dass in der Datenbank nun mehr Datensätze als vorher zu finden sind und diese Datensätze nicht alle von der Zielanwendung selbst angelegt worden sind. Dem Datenbank-Schema entsprechen sie natürlich trotzdem.

Ein weiterer, nicht zu unterschätzender Vorteil speziell der Daten-Replikation ist die relativ kostengünstige Umsetzung insbesondere im Vergleich zu den klassischen EAI-Tools.

4.1.1.3 Nachteile und mögliche Probleme

Der Vorteil der Unabhängigkeit der zur Datenbank gehörenden Anwendung[15] ist aber auch mit einem Nachteil verbunden: Es sind nur solche Operationen möglich, die ein einwandfreies Weiterarbeiten dieser Anwendung(en) gewährleisten.

So könnte beispielsweise über SQL-Kommandos die Datenbank problemlos in ihrer Struktur verändert werden. Auch wenn das im Einzelfall noch so sinnvoll ist, kann diese Möglichkeit nicht zugelassen werden, da ansonsten die zugreifende Anwendung nicht mehr mit der Struktur der eigenen Datenbank arbeiten kann bzw. entsprechend umprogrammiert werden müsste.

Eine Erweiterung des Datenbestandes um zusätzliche Attribute in Form von Tabellenspalten ist demgegenüber prinzipiell möglich, da hierdurch die Arbeitsfähigkeit der Anwendung nicht beeinträchtigt würde. Wenn man sich allerdings das Ziel einer Datenintegration – das Arbeiten von Anwendungen mit Daten, die zusätzlich in die Datenbank geschrieben wurden – vor

[15] Der Einfachheit halber wird das Wort „Anwendung“ hier ausschließlich im Singular benutzt. Trotzdem können auf eine Datenbank unabhängig von Datenintegrations-Maßnahmen mehrere Anwendungen zugreifen.

Augen führt, wird deutlich, dass diese Variante wenig Sinn ergibt. Schließlich würde die Anwendung die neuen Attribute überhaupt nicht kennen und demzufolge auch nicht ohne Umprogrammierung in der Lage sein, damit zu arbeiten.

Aus diesen Gründen ist nur ein Auffüllen, kein Erweitern des Datenbestands möglich bzw. sinnvoll.

Eine die Daten verändernde Operation (d.h. INSERT, UPDATE oder DELETE) kann – auch unter fachlichen Gesichtspunkten – prinzipiell erlaubt sein; trotzdem sind möglicherweise ergänzende Transaktionen zur Sicherung bzw. Wiederherstellung der Integrität notwendig. Der Aufruf einer Methode der zur Datenbank gehörenden Anwendung könnte in einem solchen Fall diese Transaktion nach dem Änderungs-Vorgang automatisch anstoßen; durch das Auslagern in eine Stored Procedure könnte diese Logik sogar vollständig auf der Datenhaltungs-Schicht der Anwendung verbleiben.

Doch selbst wenn diese Funktionalität denkbar einfach zu implementieren ist und auch das Umprogrammieren der Anwendung keine Schwierigkeit dargestellt, bleibt ein großes Problem bestehen. Denn das Prinzip einer Daten-Replikation bzw. einer Datenintegration im Allgemeinen ist ja – wie bereits mehrfach erwähnt – ein Fremdzugriff auf den Datenbestand ohne das Involvieren der zugehörigen Anwendung. Mit anderen Worten, die Information über Änderungen des Datenbestandes ist aus Sicht dieser Anwendung – selbst dann, wenn die Integrität verletzt wurde – keine Bring-, sondern eine Holschuld. Das bedeutet, dass die Anwendung ihren eigenen Datenbestand periodisch (eigentlich ständig) auf Integritäts-Verletzungen überprüfen müsste, auch wenn eigentlich gar keine Transaktion selbst durchgeführt wurde. Somit wird der Vorteil des Datenzugriffs ohne Belastung der Rechenleistung auf Seiten der Anwendung durch zusätzliche, sonst nicht notwendige Last durch dieses Überwachen der Datenbank vermutlich egalisiert.

Ein weiteres Problem bei der Daten-Replikation ist die Gefahr von Inkonsistenzen im Datenbestand nach dem Überschreiben von Daten.

Schließlich arbeiten nach einer Replikation mindestens zwei Anwendungen mit zumindest teilweise identischen Daten, so dass eine Veränderung auf der einen sofort eine entsprechende Modifikation auf der anderen Seite zur Folge haben müsste.

Die Daten-Replikation sieht aber – wenn die andere Richtung nicht explizit programmiert wurde – lediglich einen unidirektionalen Transfer vor. „Hilfe“ von Seiten der Anwendungslogik ist in diesem Falle nicht zu erwarten, da diese ja gar nicht über den Replikations-Vorgang informiert wurde.

Eine redundante Datenhaltung per se kann durchaus gewollt sein, wie das Beispiel des Data Warehousing zeigt. Allerdings ist hierfür entweder eine Echtzeit-Synchronisation der Datenbestände oder eine eindeutige Datenhoheit notwendig, so dass nur jeweils eine Anwendung berechtigt ist, den Datenbestand zu verändern. Auch im zweiten Falle ist natürlich eine zeitnahe Verteilung der geänderten Daten zweckmäßig, allerdings ist der zugreifenden Anwendung die Tatsache bekannt, dass der kopierte Datenbestand u.U. nicht mehr aktuell ist, so dass

bei dessen weiterer Verarbeitung entsprechende Maßnahmen ergriffen werden können. Genau diese Information fehlt aber eben im Falle einer unbemerkten Daten-Replikation.

4.1.1.4 Folgen dieser Vor- und Nachteile für den Einsatz

Auch bei Standard-Software, wie z.B. SAP R/3, ist Datenintegration prinzipiell möglich. Bei Volkswagen beispielsweise greift dieses System auf eine eigenständige Oracle-Datenbank zu, bei der sich problemlos Datenintegrations-Werkzeuge zum Einsatz bringen ließen. Diese Möglichkeit ist allerdings von SAP-Seite aus den unter 4.1.1.3 genannten Gründen verboten worden.

Dieses Beispiel verdeutlicht nochmals den Stellenwert der Probleme. Daher sollte der Einsatz einer Daten-Replikation an einige Bedingungen geknüpft sein, die im Folgenden erläutert werden.

Eine entscheidende Bedingung ist, dass die empfangende Anwendung nur lesend mit den Daten arbeitet. Auf diese Weise werden die Probleme der Rückübermittlung geänderter Daten zur Beseitigung von Inkonsistenzen von vornherein vermieden.

Darüber hinaus sollten bei der Implementierung der Daten-Replikation detaillierte Kenntnisse über die beteiligten Datenbanken und deren Struktur vorliegen, um integritätssichernde Transaktionen gleich im Zusammenhang mit der Replikation ausführen zu können.

Am besten kann dies erreicht werden, indem die zuständigen Personen von vornherein in die Planung der Daten-Replikation involviert oder zumindest darüber informiert werden, so dass von ihrer Seite die Zustimmung eingeholt werden kann, dass die beabsichtigten Transaktionen die beteiligten Datenbanken in einem stabilen Zustand hinterlassen. Auch wenn die Anwendung keinen neuen Code erhält, so ist es doch von großem Vorteil, wenn die betreuenden Personen zumindest das Wissen haben, dass ein Replikations-Werkzeug auf den Datenbestand zugreift und welche Daten dies betrifft.

Bei dieser Gelegenheit kann sich herausstellen, dass es am effizientesten ist, wenn die Replikation beispielsweise in Form einer Stored Procedure auf Seiten der Quell- oder Ziel-Datenbank implementiert und nur noch periodisch aufgerufen wird. Das ist dann zwar keine Daten-Replikation im o.g. Sinne mehr, da diese nicht unbemerkt von den Anwendungen erfolgt, kann aber große Vorteile hinsichtlich der Einarbeitung und der Netzlast mit sich bringen.

Das Idealbeispiel einer Daten-Replikation ist daher das Kopieren von Daten zwischen zwei Instanzen ein und derselben Anwendung. Hierbei werden die o.g. Probleme entweder vollständig vermieden, oder man hat zumindest Kenntnis hiervon und ist in der Lage, sie abzuwägen und ggf. Gegenmaßnahmen zu ergreifen.

Allerdings muss auch bei dieser Variante dafür gesorgt werden, dass man nicht quasi von der Technik ausgetrickst wird: Das Einfügen eines abhängigen Datensatzes würde einen Datenfehler erzeugen, wenn es auf Seiten des Parent-Datensatzes kein Attribut mit dem referenzierten Schlüssel gibt. Ein Beispiel hierfür wäre das Einfügen einer Lieferung mit einer Lieferan-

ten-Nummer, die in der Lieferanten-Datenbank (als Primärschlüssel verwendet) gar nicht existiert. Die referenzielle Integrität wäre in diesem Falle nicht mehr gewahrt.

Bei einer Daten-Replikation ist es jedoch durchaus nicht unwahrscheinlich, dass der Parent-Datensatz (im Beispiel der Lieferant) erst unmittelbar vor dem abhängigen Datensatz (also der Lieferung) angelegt worden ist und mit ein und demselben Replikations-Vorgang übertragen wird. Wenn nun beim Schreiben in die Zieldatenbank diese Reihenfolge vertauscht wird, ist eine Verletzung der referenziellen Integrität die Folge.

Dieses Problem kann dadurch gelöst werden, dass die Integrität der importierten Daten als gegeben angenommen wird, sofern die Replikation vollständig durchgeführt worden ist. Mit anderen Worten, die Sicherstellung der Integrität liegt ausschließlich in der Verantwortung des Quellsystems.

4.1.2 Daten-Föderation

Die Daten-Föderation ist neben der Daten-Replikation (4.1.1) die zweite Möglichkeit zur Realisierung einer Datenintegration.

In 4.1.2.1 wird zunächst das Prinzip veranschaulicht. Anschließend beschreibt 4.1.2.2 die Unterschiede zwischen virtueller und physischer Daten-Föderation, also der beiden Varianten zur Realisierung.

4.1.2.1 Das Prinzip

Der Begriff der Föderation entstammt ursprünglich dem politischen Bereich [Con97]. Man versteht hierunter ein Bündnis, d.h. einen vertraglichen Zusammenschluss von Staaten oder Teilstaaten. Ein typisches Beispiel hierfür ist die aus verschiedenen Bundesländern zusammengesetzte Bundesrepublik Deutschland.

Unter Daten-Föderation wird die Vereinigung von existierenden Datenbanken zu einer neuen Datenbank verstanden, wie es in Abbildung 28 schematisch gezeigt wird.

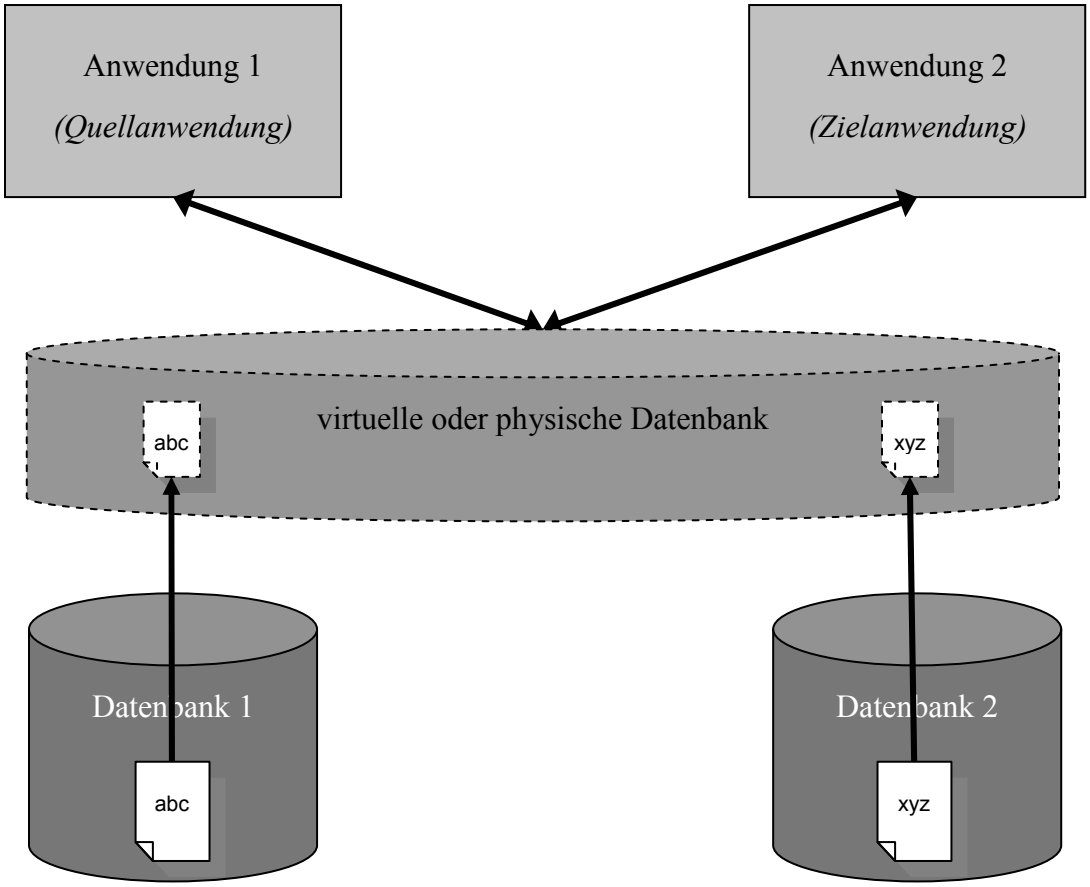

Abbildung 28: Daten-Föderation

4.1.2.2 Virtuelle vs. physische Daten-Föderation

Eine Möglichkeit zur Realisierung einer Daten-Föderation ist die Einrichtung einer neuen, physischen Datenbank, in welche die Daten aus den verteilten Datenbanken kopiert werden.

Hierzu wird typischerweise auf Werkzeuge zur Daten-Replikation (vgl. 4.1.1) zurückgegriffen. In einem solchen Falle baut also die Daten-Föderation auf der Daten-Replikation auf, ohne sie aber nach außen vollständig ablösen zu können (d.h. die Daten-Replikation bleibt weiterhin eine eigenständige Alternative).

Die zweite Variante ist eine so genannte virtuelle Datenbank. Diese sieht nach außen genauso aus wie eine normale, physische Datenbank und kann ebenso angesprochen werden. Anfragen werden von der virtuellen Datenbank in einer für den Benutzer transparenten Art und Weise aber lediglich an die jeweils korrekte physische Datenbank weiter geleitet. Es handelt sich hierbei demzufolge eigentlich um keine Datenbank, sondern lediglich um einen Routing-Mechanismus zur richtigen Datenbank.

Insbesondere dann, wenn man sich die in 4.1.1.3 aufgeführten Probleme einer Daten-Replikation vor Augen hält, erscheint es wenig zweckmäßig, eine redundante Datenhaltung in einer physischen Datenbank der Alternative einer Daten-Föderation vorzuziehen.

Doch auch eine physische Datenvereinigung hat Vorteile:

- In den operativen Systemen sind im Allgemeinen nur aktuelle Daten enthalten. Durch eine physische Datenvereinigung hat man die Möglichkeit, die vorhandenen Daten nicht durch die neuen zu überschreiben, sondern eine Historienführung aufzubauen.
- Da eine virtuelle Datenbank nur ein Routing-Werkzeug darstellt, wird jede Anfrage direkt auf der Original-Datenbank ausgeführt. Das hat zur Folge, dass der jeweilige Datenbank-Server eine höhere Last zu bewältigen hat.
- Wie in 4.1.4 noch näher erläutert wird, unterscheiden sich die Lastprofile von operativen und dispositiven Systemen deutlich: Operative Systeme haben zumeist ein relativ gleichmäßiges Aufkommen von Anfragen und damit eine ebensolche Prozessor-Belastung. Dispositive Systeme dagegen erzeugen eine stark schwankende, durch lediglich kurze, aber starke Ausschläge gekennzeichnete Prozessor-Last.

  Damit hat der Datenbank-Server nicht nur mit der bereits angesprochenen höheren Last, sondern auch mit grundlegend veränderten Lastprofilen zurechtzukommen bzw. muss ggf. mit entsprechend verstärkter Rechenleistung bestückt werden.

4.1.3 Methoden, die keine zusätzliche Alternative darstellen

Neben den in 4.1.1 und 4.1.2 beschriebenen Varianten der Daten-Replikation und Daten-Föderation existieren noch weitere Möglichkeiten, Anwendungs-Integration in Verbindung mit Daten oder Datenbanken zu betreiben.

Warum diese Möglichkeiten aber dennoch zumindest nicht als zusätzliche Alternative unter der Überschrift der Datenintegration geführt werden, soll im Folgenden verdeutlicht werden.

4.1.3.1 Datenorientierte Schnittstellen

Eine Möglichkeit aus Sicht einer Anwendung, beispielsweise die in 4.1.1.3 beschriebenen Probleme bezüglich der Integritätssicherung zu vermeiden, wäre, den direkten Zugriff auf die Datenbank nicht zu erlauben und stattdessen eine oder mehrere Methoden anzubieten, welche die gewünschten Datenbank-Operationen ausführen und darüber hinaus die Sicherung der Integrität gewährleisten.

Diese architektonische Variante, die in Abbildung 29 schematisch dargestellt ist, stellt einen Grenzfall dar: Auf der einen Seite wird hier Anwendungslogik aufgerufen und demzufolge eindeutig nicht nur die Datenhaltungs-Schicht angesprochen. Auf der anderen Seite reichert diese Anwendungslogik im einfachsten Fall weder die Anfragen an die Datenbank noch die

Ergebnisse an, sondern überwacht lediglich die Integrität, so dass das Resultat aus Sicht einer zugreifenden Anwendung dasselbe ist wie bei einem direkten Datenbank-Zugriff.

Dennoch wird dieses Vorgehen aus den bereits in 2.5.2 im Rahmen der Trennung verschiedener Zugriffs-Möglichkeiten auf Anwendungen (zweite Dimension der Anwendungs-Integration) genannten Gründen nicht als Datenintegration, sondern als Funktions-Aufruf klassifiziert. Daran ändert sich auch dann nichts, wenn der Zugriff nicht – wie in Abbildung 29 gezeigt – durch eine andere Anwendung, sondern durch ein Datenintegrations-Werkzeug erfolgt.

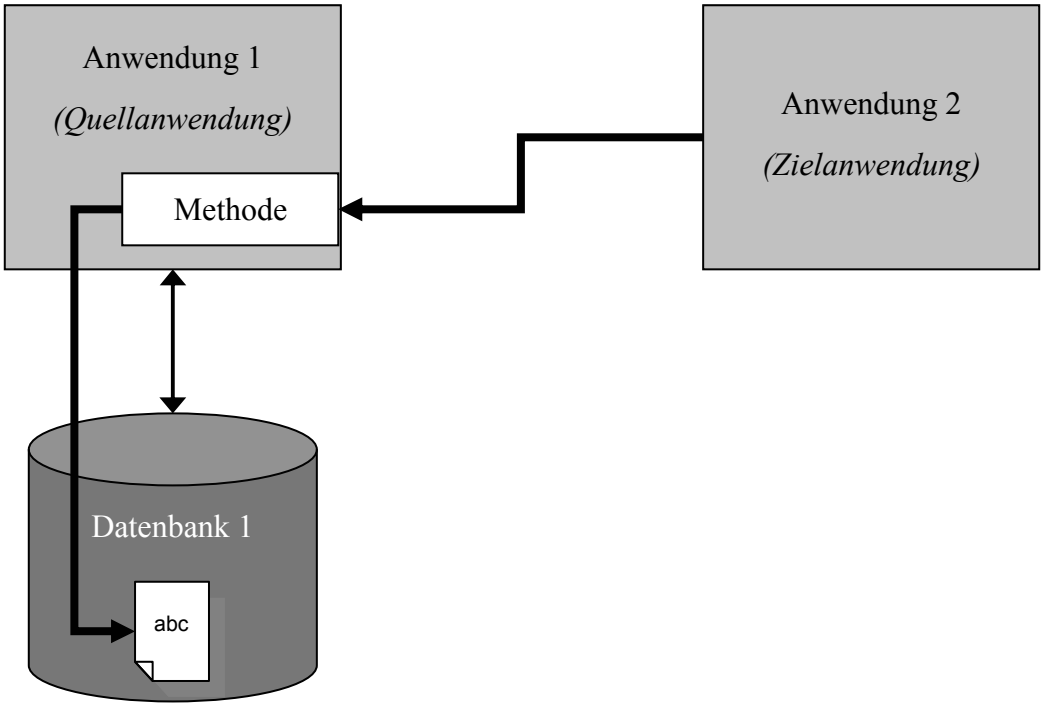

Abbildung 29: Keine Variante der Datenintegration: Datenorientierte Schnittstellen

4.1.3.2 Direkter Datenbank-Zugriff

Der offensichtlich einfachste Weg, entfernte Datenbanken anzusprechen, ist der direkte Zugriff beispielsweise über SQL-Kommandos, wie er in Abbildung 30 dargestellt ist.

Jedoch wird hierbei das Grundprinzip der Datenintegration, nämlich die Unabhängigkeit sowohl von der Quell- als auch von der Zielanwendung, verletzt. Es handelt sich vielmehr um einen in der Anwendungslogik der Zielanwendung codierten entfernten Datenbank-Zugriff, also eine Punkt-zu-Punkt-Verbindung. Je nach Kapselung der angesprochenen Methode kann es sich hierbei auch um EAI oder SOA handeln, aber auf keinen Fall um Datenintegration.

Es ist erkennbar, dass an dieser Stelle wiederum die drei Dimensionen der Anwendungs-Integration (vgl. 2.5) zur Erklärung herangezogen werden können: Die zweite Dimension (der Zugang zur Zielanwendung) erfolgt über Datenbank-Zugriffe – also genau wie bei einer Da-

tenintegration. Die erste Dimension (das zugrunde liegende Konzept) entspricht jedoch nicht dem der Datenintegration, sondern – wie oben erwähnt – entweder einer Punkt-zu-Punkt-Verbindung oder dem klassischen EAI-Ansatz oder einer SOA.

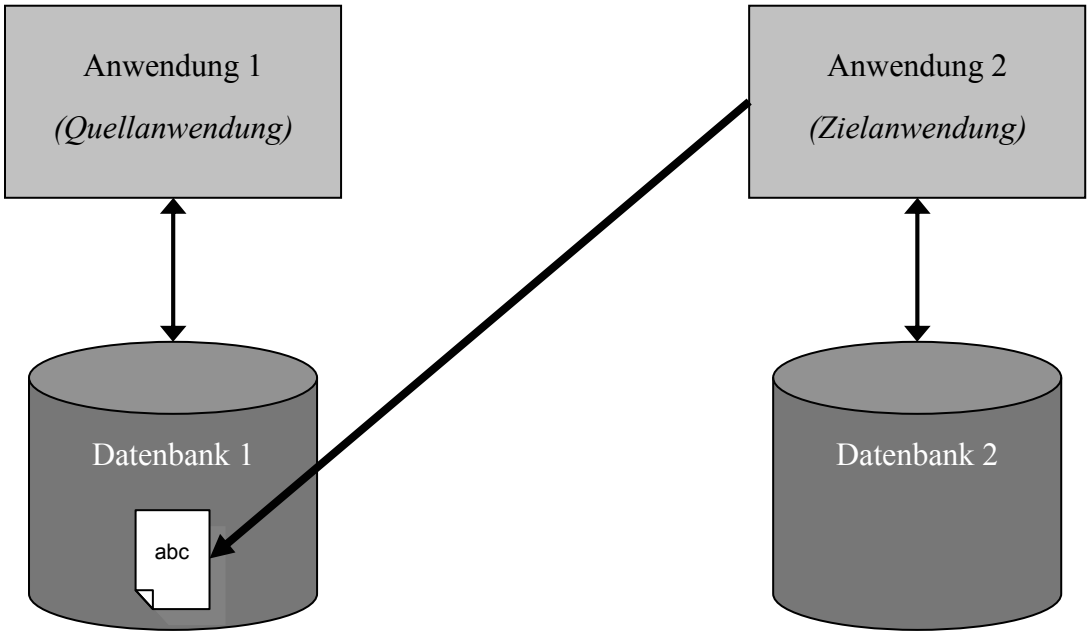

Abbildung 30: Keine Variante der Datenintegration: Direkter Datenbank-Zugriff

4.1.3.3 Datei-Transfer

Eine Alternative zur Daten-Replikation scheint aus Abbildung 31 hervorzugehen: Statt in eine Datenbank zu schreiben, auf die anschließend z.B. mit Replikations-Werkzeugen zugegriffen wird, speichert die Quellanwendung die Daten in einer Datei (im Beispiel im TXT-Format; hier sind aber auch andere Formate denkbar). Diese kann dann so wie jede Datei an einen anderen Ort kopiert und dort eingelesen werden.

Im Gegensatz zu den datenorientierten Schnittstellen und zum direkten Datenbank-Zugriff ist in diesem Falle nur die Datenhaltungs-Schicht an der Integration beteiligt. Ein Grund dafür, dass dies unbedingt eine Datenbank sein muss, ist nicht ersichtlich. Folglich kann diese Möglichkeit durchaus unter der Überschrift der Datenintegration geführt werden.

Doch beim Vergleich mit einer „regulären“ Daten-Replikation (vgl. 4.1.1) wird schnell deutlich, dass der Datei-Transfer nur die Anpassung dieses Konzepts an solche Anwendungen darstellt, die ihre Daten nicht in einer Datenbank, sondern in Dateien speichern. Deshalb sollte der Datei-Transfer nicht als zur Daten-Replikation und Daten-Föderation gleichwertiges Konzept, sondern als ein Spezialfall der Daten-Replikation angesehen werden.

Es sei noch ausdrücklich darauf hingewiesen, dass an dieser Stelle das Kopieren von bereits vorhandenen Dateien an einen anderen Ort gemeint ist. Ein explizites Anlegen von Dateien

zum ausschließlichen Zwecke der Informations-Übermittlung zu anderen Anwendungen ist selbstverständlich keine Datenintegration, sondern eine Punkt-zu-Punkt-Verbindung.

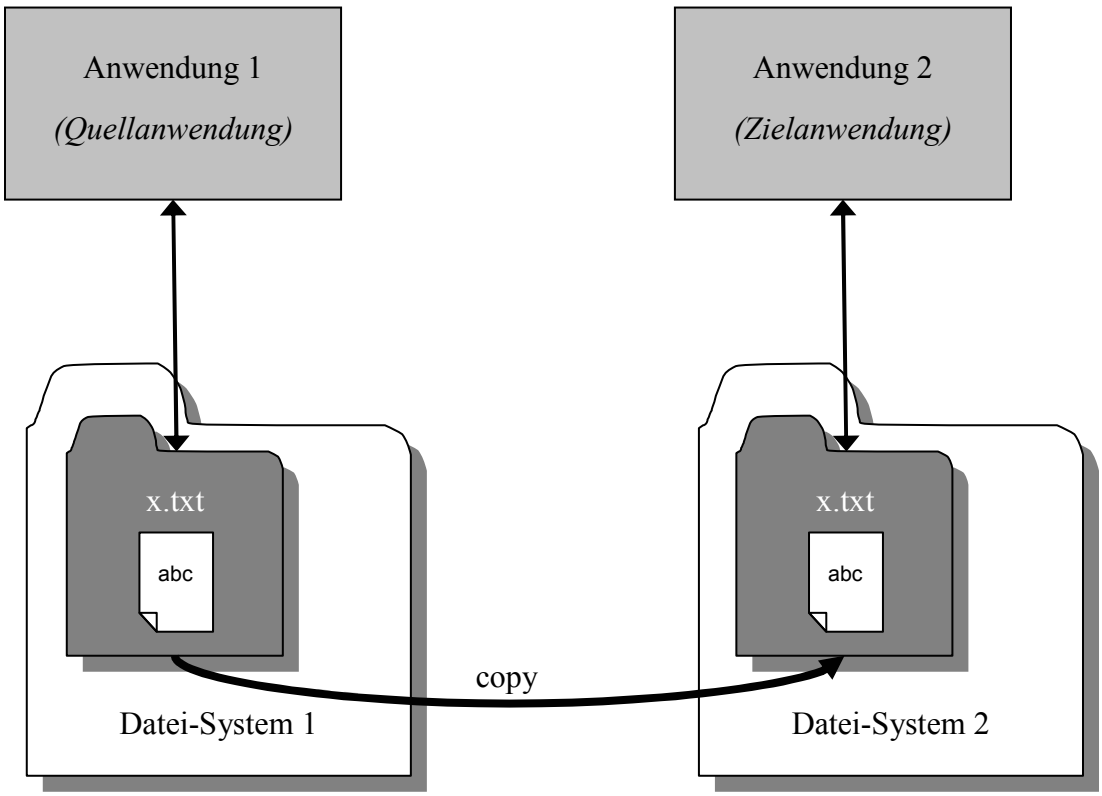

Abbildung 31: Keine (eigene) Variante der Datenintegration: Datei-Transfer

4.1.4 Datenintegration und Data Warehousing

Die Ursprünge des Data Warehousing-Konzepts reichen unter der Bezeichnung „zweckneutrale Grundrechnung“ bis ins Jahr 1948 (Schmalenbach) zurück [HBO97].

Ein Data Warehouse ist ein prinzipiell unternehmensweiter, zweckneutraler Datenpool, der aus den operativen Systemen gespeist wird. Im Gegensatz zur dortigen Datenspeicherung lässt sich ein Data Warehouse durch die folgenden Merkmale charakterisieren [Inm96]:

- Themenorientierung (im Gegensatz zur weitgehend funktions- oder prozessorientierten Form der Speicherung der Daten in den operativen Systemen)
- Vereinheitlichung (u.a. zur Vermeidung von identischen Daten unter verschiedenen Namen)
- Zeitraumorientierung (und nicht Zeitpunkt-Betrachtung)
- Dauerhaftigkeit (persistente Speicherung, Historienführung)

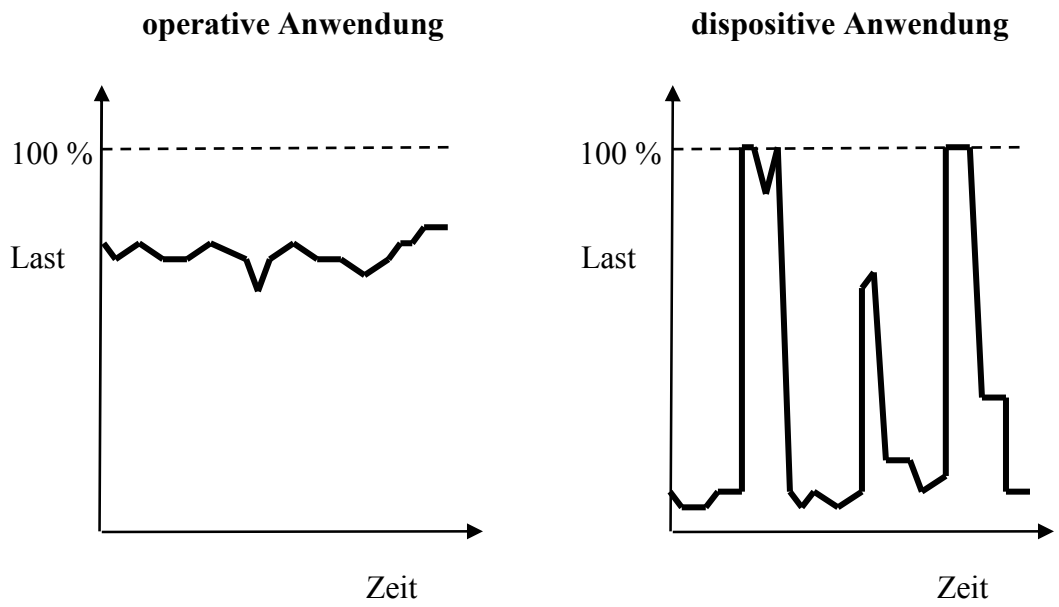

Abbildung 32: Unterschiedliche Lastprofile von operativen und dispositiven Anwendungen

Ein weiterer, technischer Vorteil ist die Trennung der für das operative Geschäft genutzten Daten von denen, die über das Data Warehouse für Controlling-Zwecke genutzt werden.

Diese Trennung ist aufgrund der unterschiedlichen Lastprofile dieser Nutzungsarten sinnvoll (siehe Abbildung 32): Bei operativen Anwendungen ist die Last zwar selten völlig konstant, ihre Schwankungen bewegen sich aber in einem prozentual geringen Bereich, da kontinuierlich Anfragen eintreffen. Eine dispositive Anwendung hingegen kennt so gut wie keinen Routine-Betrieb, d.h. zu einem großen Teil der Zeit ist die Last äußerst gering. Werden aber Anfragen ausgeführt (wie beispielsweise eine komplexe Abfrage aus einem Data Warehouse), so steigt die Last für eine mehr oder weniger kurze Zeitspanne auf einen deutlich höheren Wert.

Würden eine operative und eine dispositive Anwendung auf einen gemeinsamen Datenbestand zugreifen, so müsste entweder eine große Leistungsreserve (also z.B. mehrere Prozessoren) zur Kompensation der Lastprofil-Spitzen der dispositiven Anwendung bereitgehalten und bei Bedarf zugeschaltet werden, oder die dispositive Anwendung würde die Performance der operativen temporär deutlich senken.

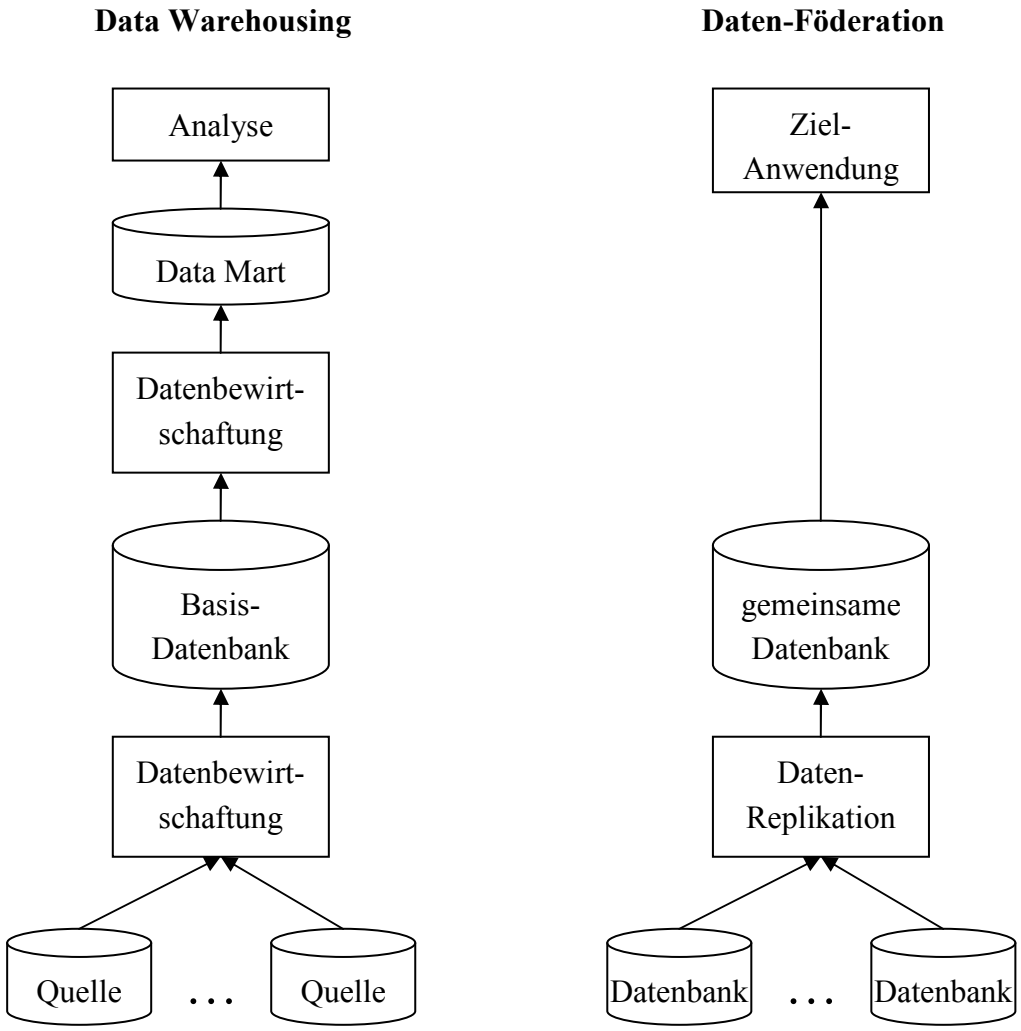

Abbildung 33: Konzeptioneller Vergleich von Data Warehousing und Daten-Föderation

Wenn man den Prozess des Data Warehousing, ohne auf die einzelnen Schritte im Detail einzugehen, grafisch darstellt, so erhält man die linke Hälfte von Abbildung 33. In der rechten Hälfte dieser Abbildung ist das Prinzip der Daten-Föderation (vgl. 4.1.2) zu sehen. Wie man unschwer erkennen kann, sind hier auf den ersten Blick zahlreiche Gemeinsamkeiten auszumachen: Beide Konzepte greifen auf bestehende Datenquellen zu und replizieren deren Inhalte (oder zumindest Teile hiervon) in eine neue Datenbank.

Also könnte der Eindruck entstehen, dass man zwei Namen für ein identisches Konzept benutzt. Dies ist aber nicht der Fall, denn obwohl in technologischer Hinsicht tatsächlich Gemeinsamkeiten bestehen, existieren in konzeptioneller Hinsicht fundamentale Unterschiede, die in Abbildung 34 gezeigt werden.

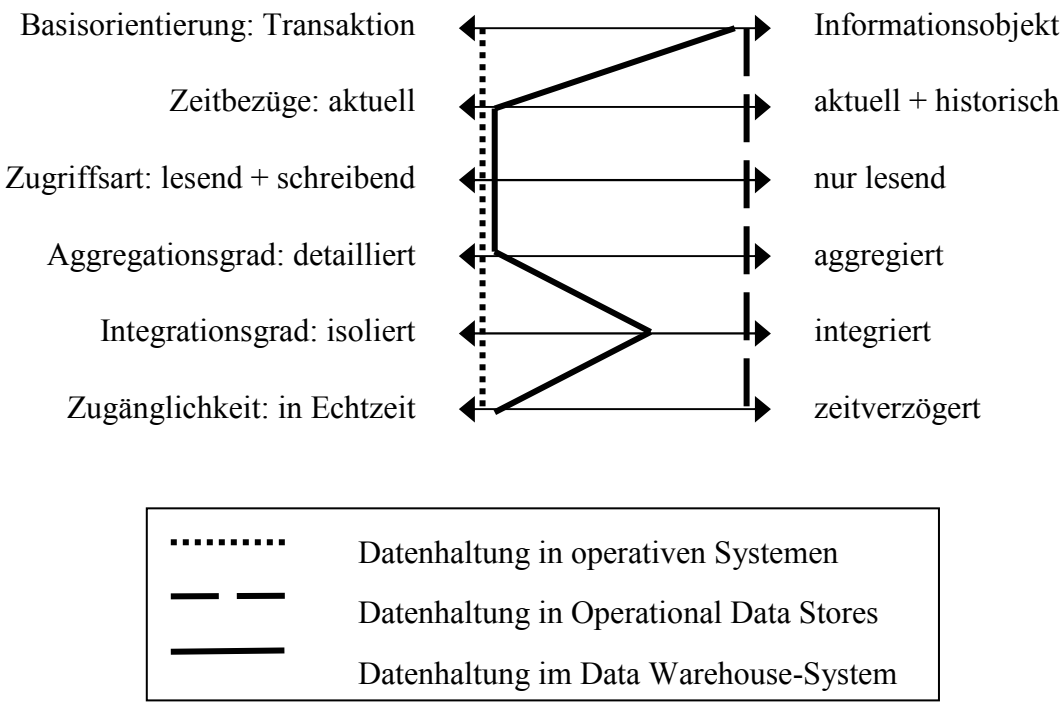

Abbildung 34: Vergleich von Datenintegration und Data Warehousing anhand von Eigenschaftsprofilen (nach [MSW03])

In dieser von der Universität St. Gallen stammenden Darstellung wird die Datenhaltung in operativen Systemen der in Operational Data Stores (hier nicht weiter betrachtet) und Data Warehouses gegenübergestellt. Die wesentlichen Unterschiede bestehen darin, dass in Data Warehouses auch historische Daten (also mehrere Instanzen von Daten) gehalten werden, und dass die Daten in einem Data Warehouse aus mehreren Quellen aggregiert werden. Der Echtzeit-Aspekt hingegen spielt mittlerweile eine eher untergeordnete Rolle, da die meisten neueren Data-Warehousing-Tools auch einen Echtzeit-Zugriff zulassen.

4.2 Punkt-zu-Punkt-Verbindungen

Eine Punkt-zu-Punkt-Verbindung verknüpft genau zwei Software-Komponenten miteinander und ist im Allgemeinen speziell für diese eine Kommunikation isoliert geplant und implementiert worden.

Im Folgenden wird zunächst die Problematik der Einordnung von Punkt-zu-Punkt-Verbindungen beleuchtet. Anschließend werden die Vor- und Nachteile erörtert, die diese Art der Integration mit sich bringt.

4.2.1 Einordnung als Integrations-Konzept

Auf den ersten Blick scheint es nicht schlüssig zu sein, dass Punkt-zu-Punkt-Verbindungen unter der Überschrift „Integrations-Konzepte“ geführt werden. Im Gegensatz zu EAI, SOA und selbst Datenintegration stellt diese Art der Integration schließlich keine geplante Strategie oder technologische Lösung dar, sondern bezeichnet vielmehr eine historisch gewachsene Situation.

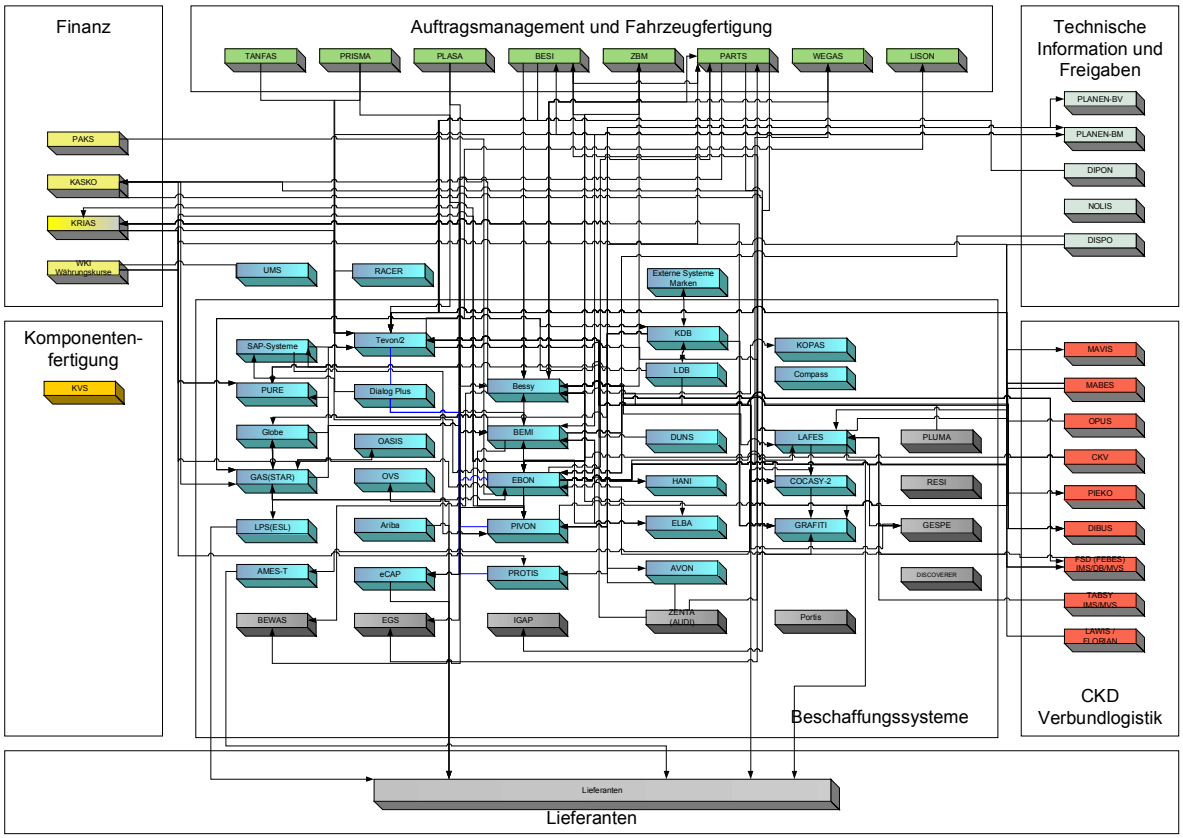

Abbildung 35: Die sog. „Spaghetti-Architektur“ von Punkt-zu-Punkt-Verbindungen (Quelle: Volkswagen AG, Konzern-Beschaffung)

Die ausschließliche Verwendung dieser Art von Integration in größeren Unternehmen – soviel kann schon einmal vorweg genommen werden – ist zweifellos nicht strategisch zielführend

und stellt genau den Zustand dar, der mit Hilfe von EAI und SOA geändert werden sollte und soll. Dieser Zustand ist gekennzeichnet durch eine Vielzahl von einzelnen Verbindungen und wird oftmals treffend als „Spaghetti-Architektur“ bezeichnet (siehe Abbildung 35). Die genauen Bezeichnungen der einzelnen Systeme, die in der Grafik kaum lesbar sind, sind an dieser Stelle nicht von Bedeutung.

4.2.2 Vorteile von Punkt-zu-Punkt-Verbindungen

Bei nur wenigen Schnittstellen bzw. wenig Kommunikationsbedarf zwischen Anwendungen sind Punkt-zu-Punkt-Verbindungen eine sehr geeignete, wenn nicht gar genau die richtige Methode. Schließlich sind alle übergreifenden Ansätze, sei es EAI oder SOA mit einem ESB, zumindest mit einem höheren Implementierungs-Aufwand, wenn nicht sogar mit Performance-Nachteilen (vgl. 3.6.2) verbunden.

4.2.3 Nachteile von Punkt-zu-Punkt-Verbindungen

Schwierigkeiten treten bei Punkt-zu-Punkt-Verbindungen dann auf, wenn die Anzahl der notwendigen Verbindungen größer wird, so wie es in wohl allen größeren Unternehmen im Laufe der letzten Jahrzehnte sukzessive der Fall war. Daraus ergeben sich die im Folgenden erläuterten Probleme.

4.2.3.1 Langwierige Integration neuer Systeme

Bei jedem System, das neu in eine solche gewachsene IT-Landschaft integriert werden soll, ist es notwendig, die Verbindungen zu allen vorhandenen Systemen, mit denen eine Kommunikation notwendig ist, individuell zu implementieren.

Als Schätzwert für die notwendige Anzahl von Schnittstellen bei n Anwendungen wird an vielen Stellen in der Literatur $\frac{n*(n-1)}{2}$ genannt. Diese Formel errechnet sich daraus, dass von jeder der n Anwendungen zu allen anderen (also n-1) Anwendungen Verbindungen aufgebaut werden. Die Ursache für die Division durch 2 ist, dass die Schnittstellen als bidirektional angenommen werden, so dass die Verbindungen von einer Anwendung zur nächsten und wieder zurück nur eine Schnittstelle darstellen.

Bei 100 Anwendungen wären folglich $\frac{100*99}{2} = 4.950$ Schnittstellen erforderlich.Diese Formel beinhaltet allerdings drei Annahmen, die in der Praxis nur in Ausnahmefällen vorkommen[16]:

1. Jede Anwendung muss mit jeder anderen kommunizieren

   Laut einer Statistik[17] benutzen nur 15% der Anwendungen, die miteinander kommunizieren, spezielle Integrations-Middleware. In den anderen Fällen wird eher auf ETL[18]-Werkzeuge oder Batch-gesteuerten Datei-Transfer zurückgegriffen. Dadurch reduziert sich die Anzahl der durchschnittlich zu implementierenden Schnittstellen bei n Anwendungen auf n*0,15.

2. Jede Anwendung stellt für die Kommunikation mit einer anderen Anwendung genau eine Schnittstelle zur Verfügung

   Insbesondere bei der Kommunikation zwischen großen Systemen (wie z.B. ERP-Systemen), die mehrere Funktionalitäten in sich vereinen, ist es zu erwarten, dass es mehr als nur einen fachlichen Berührungspunkt gibt und folglich mehr als eine Schnittstelle notwendig ist.

3. Jede Schnittstelle ist für genau eine zugreifende Anwendung geeignet

   Auch wenn der Gedanke der Wiederverwendung bei der Implementierung von Punkt-zu-Punkt-Verbindungen in der Vergangenheit noch nicht wie heute im Bereich SOA ausgeprägt war, sind viele Schnittstellen (bzw. Zugangspunkte zu Anwendungen) so allgemein gestaltet, dass sie von mehreren Anwendungen genutzt werden.

[16] Chappell [Cha04] weist grundsätzlich auf diese Annahmen hin. Allerdings vermischt er die zweite und dritte Annahme miteinander: Er schätzt die durchschnittliche Anzahl von Schnittstellen im RPC-Stil pro Anwendung auf vier und multipliziert diese Zahl direkt mit der Anzahl der anderen Anwendungen, ohne zu berücksichtigen, dass diese vier Schnittstellen nicht zwangsläufig für eine einzige Zielanwendung bestimmt sein müssen (nach der klassischen Formel ist schließlich auch nicht mit einer, sondern mit n-1 Schnittstellen zu rechnen). Im Gegenteil kann die Zahl von vier Schnittstellen in diesem Beispiel sogar zu einer Verringerung der notwendigen Verbindungen führen, wenn damit beispielsweise sechs Anwendungen bedient werden.

Chappells Argumentation ist allerdings dann korrekt, wenn unter dem Begriff der Schnittstelle nicht auch ein Zugangspunkt, sondern ausschließlich eine Verbindung zwischen genau zwei Anwendungen verstanden wird.

[17] ohne weitere Quellenangabe in [Cha04] zitiert

[18] Extract, Transform, Load; vgl. A.5

Doch selbst wenn man zusätzlich zu den o.g. 15% die sehr optimistisch geschätzten Zahlen von durchschnittlich zwei Schnittstellen zwischen je zwei kommunizierenden Anwendungen und zwei nutzenden Anwendungen pro Schnittstelle zugrunde legt, ergeben sich bei 100 Anwendungen immer noch

$$\frac{100 * (99 * \frac{15}{100} * 2 * \frac{1}{2})}{2} \approx 743 \text{ Schnittstellen.}$$

Diese Zahl ist zwar deutlich geringer als die nach der traditionellen Formel berechneten 4.950, aber immer noch groß genug, um den Handlungsbedarf deutlich zu machen.

4.2.3.2 Aufwändige Wartbarkeit

Nicht nur die Einführung neuer Systeme, sondern auch die laufende Wartung der bereits vorhandenen ist bei Punkt-zu-Punkt-Verbindungen sehr aufwändig.

Schließlich muss für jede Art von Schnittstelle ausgebildetes und erfahrenes Personal bereitstehen: zum einen zur technischen Wartung jeder einzelnen verwendeten Technologie, zum anderen für fachlichen Support.

4.2.3.3 Schwere Änderbarkeit

„Ich ziehe an einem System, und wenn ich fest genug ziehe, habe ich das ganze Unternehmen in der Hand."

Dieser Spruch, den man in großen Unternehmen nicht selten hören kann, macht die Problematik des Austausches eines bestehenden Systems in einer mit Punkt-zu-Punkt-Verbindungen vermaschten Landschaft deutlich.

Die in 4.2.3.1 präsentierten Zahlen sind natürlich nicht nur ein Maß für die zu implementierenden Schnittstellen bei der Integration neuer Systeme, sondern auch für die bestehenden Schnittstellen eines jeden vorhandenen Systems.

Schon die Aufgabe, alle diese Schnittstellen anzupassen, d.h. die zugreifenden Anwendungen so zu verändern, dass sie nunmehr mit dem neuen System statt dem ausgetauschten kommunizieren, ist nicht unproblematisch. Schließlich spielt hier nicht nur der Faktor Arbeitsaufwand eine Rolle, sondern auch die Frage, ob diese Veränderung der Altsysteme überhaupt ohne weiteres möglich ist (vgl. hierzu auch 2.5.2). Aber eine zumeist noch schwierigere, wenn nicht gar z.T. unlösbare Aufgabe ist es, wirklich alle Kommunikationspartner zu identifizieren. Wohlgemerkt geht es hier nicht darum, alle entfernten Prozedur-Aufrufe aus dem eigenen Quellcode herauszufinden, sondern um die andere Richtung, nämlich die Identifikation der Nutzer einer publizierten Schnittstelle. Nicht wenige bestehende Punkt-zu-Punkt-Verbindungen wird man erfahrungsgemäß erst dann erkennen, wenn nach dem Entfernen oder Austauschen einer Anwendung irgendwo anders ein Fehler auftritt. Im glücklichsten Fall muss dort nur im Quellcode ein Methodenname o.ä. geändert werden; im Katastrophenfall hat man keine andere Möglichkeit, als auch diese Anwendung auszutauschen.

Aus denselben Gründen ist nicht nur das Entfernen oder Austauschen eines Systems problematisch, sondern auch bereits eine simple Schnittstellen-Änderung. Wenn es – wie oben beschrieben – keine Möglichkeit der Änderung eines auf diese Schnittstelle zugreifenden Systems gibt, steht man vor der Wahl, entweder die Änderung der Schnittstelle rückgängig zu machen oder das zugreifende System auszutauschen – was wiederum die o.g. Folgen haben kann.

4.3 Enterprise Application Integration (EAI)

Nachdem insbesondere in Kapitel 2 schon mehrfach die Abkürzung EAI gefallen ist, soll an dieser Stelle zunächst der Hintergrund dieser begrifflichen Bedeutung aufgezeigt werden.

Anschließend werden das dahinter liegende Konzept erläutert, einige ausgewählte, darauf aufbauende Produkte vorgestellt sowie Vor- und Nachteile dieses Ansatzes beleuchtet. Letztere bilden die Basis für den darauf folgenden Themenbereich SOA.

4.3.1 Definition(en)

EAI bedeutet Enterprise Application Integration. Diese Abkürzung, die gegen Anfang der neunziger Jahre entstand und um die Jahrtausendwende in Fachkreisen in aller Munde war, steht für die Reaktion auf das Problem der zunehmenden Punkt-zu-Punkt-Verbindungen (vgl. 4.2).

Was aber genau EAI ist und mit EAI erreicht werden soll, darüber findet man sehr unterschiedliche Angaben und Definitionen, von denen im Folgenden einige Beispiele aufgeführt werden, bei denen unterschiedliche Schwerpunkte im Vordergrund stehen.

- Technische Verbindung von Anwendungen: *„**Middleware** zur Kopplung von Systemen, meistens betriebswirtschaftlicher Systeme, wie z.B. ERP, SCM, CRM und E- Commerce-Software.“* [Hor05]
- Unterstützung von Geschäftsprozessen: „*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]
- Semantische Verbesserung der Kommunikation: *„Mit EAI ist es möglich systemübergreifende Geschäftsprozesse mit Hilfe einer geeigneten Technik so abzubilden, dass die am Geschäftsprozess beteiligten unterschiedlichen Systeme in der Lage sind, prozessrelevante Daten so miteinander auszutauschen, dass alle beteiligten Anwendungen ein **gleiches Verständnis der Daten** haben.“* [Fel05]

- Software-Architektur: *„EAI bezeichnet nun eine **Software-Architektur**, die die einzelnen Anwendungen miteinander ‚verbindet' – integriert. EAI ermöglicht einen **Datenaustausch untereinander** und lässt ganz **neue Anwendungen** entstehen, die vorhandene Betriebslogik und Funktionalitäten nutzen und erweitern. Dabei geschieht dies möglichst so, dass vorhandene Anwendungen nicht verändert werden müssen."* [EAIF04]

Diese Beispiele verdeutlichen die Unterschiede der Schwerpunkte, die bei EAI-Definitionen gesetzt werden. Das EAI-Forum [EAIF04] definiert den EAI-Begriff sogar – wie Abbildung 36 veranschaulicht – in zwei verschiedenen Bedeutungen, nämlich einmal als Oberbegriff für jegliche Integration und einmal als Bezeichnung für die unternehmensinterne Integration.

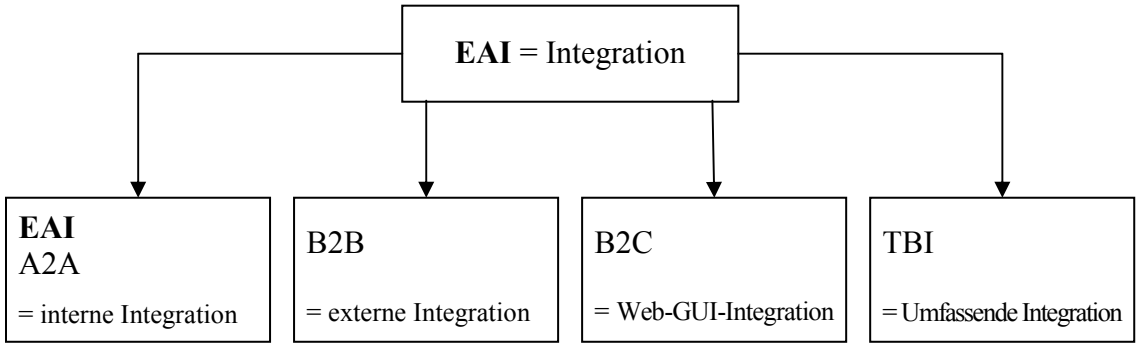

Abbildung 36: Die Unterteilung von EAI nach dem EAI-Forum [EAIF04][19]

Auch wenn die Abkürzung EAI ursprünglich als Bezeichnung für die Integration von Anwendungen insgesamt gedacht war, ist sie heute im Allgemeinen mit einer speziellen architektonischen Variante, nämlich der so genannten Hub-and-Spokes-Architektur (Nabe und Speichen; näheres hierzu in 4.3.2), verbunden. Eine entsprechende Auslegung des EAI-Begriffs findet man z.B. bei Kaye: *„EAI software typically runs on dedicated servers, often called* hubs *or* brokers *[...]. EAI packages also include adapters or* connectors *to popular ERP and CRM application suites."* [Kay03]. Allerdings ist diese Definition sehr technisch orientiert und sagt nichts über die logische Funktionsweise und die Art der angebundenen Software-Komponenten aus.

Daher wird EAI im Rahmen dieser Arbeit grundsätzlich definiert als eine Form der Anwendungs-Integration, bei der verschiedene Anwendungen über einen zentralen Broker kommunizieren und dabei in ihrer ursprünglichen Struktur sichtbar bleiben, d.h. nicht als Services gekapselt werden (siehe auch A.1).

[19] A2A = Application to Application,
B2B = Business to Business,
B2C = Business to Consumer,
TBI = Total Business Integration

4.3.2 Funktionsweise

Wie die Ausführungen in 4.3.1 schon angedeutet haben, ist das Prinzip dieser Architektur eine zentrale Instanz, an die sich jede Anwendung (also nicht nur ERP- und CRM-Systeme, wie bei der in 4.3.1 aufgeführten Definition von Kaye) mit Hilfe von Adaptern oder Konnektoren andocken kann, wie in Abbildung 37 skizziert ist. Die Kommunikation zwischen den angeschlossenen Anwendungen erfolgt also nie direkt, sondern stets über den zentralen Broker.

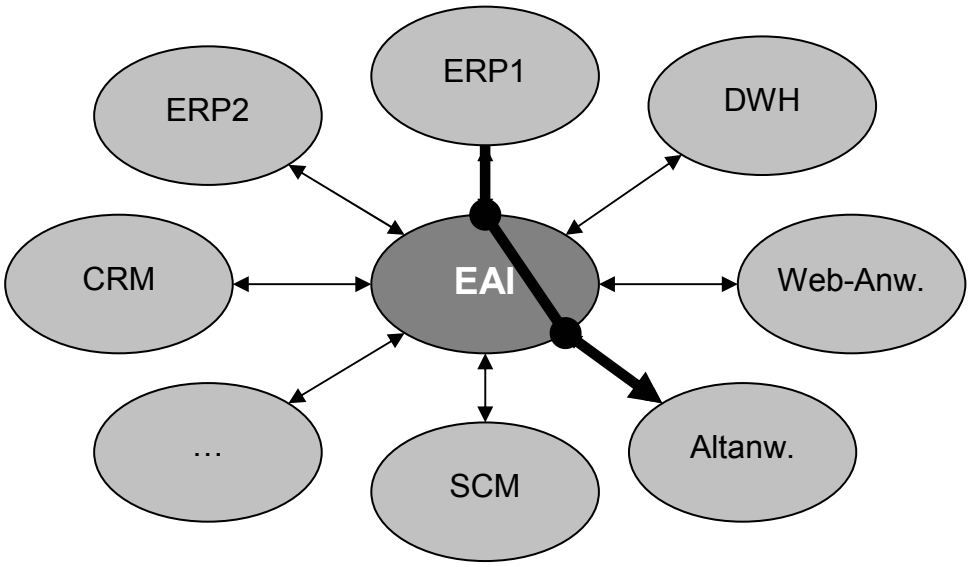

Abbildung 37: Der klassische EAI-Ansatz (Hub-and-Spokes-Architektur) [Kai02]

Damit die Kommunikation unabhängig von einzelnen Anwendungen erfolgen kann, definiert der Broker ein zentrales, anwendungsunabhängiges Datenformat. Hierdurch wird in syntaktischer (heißt z.B. das Feld für den Namen des Kunden „Kundenname“, „Kunde“, „KName", „Name_Kunde“ oder ganz anders) und semantischer Hinsicht (gibt es z.B. auch interne Kunden oder wird dieser Terminus ausschließlich auf externe Kunden bezogen) die Basis für die Interaktion zwischen den Anwendungen geschaffen.

Um an dieser zentral vermittelten Kommunikation teilhaben zu können, ist für jede Anwendung ein passender Adapter oder Konnektor notwendig. Dessen Aufgabe ist es, das anwendungsspezifische, meist proprietäre Format auf das o.g. zentrale, anwendungsunabhängige Format zu konvertieren. Häufig können solche Adapter fertig erworben werden (insbesondere für sehr verbreitete Systeme oder solche desselben Herstellers wie der EAI-Broker), oftmals müssen sie aber auch selbst implementiert werden – je nach Komplexität der Datenstrukturen und der Kommunikation kann dies einen nicht unerheblichen Aufwand darstellen.

4.3.3 Technologien zur Umsetzung

Wie am Anfang von Kapitel 3 begründet, soll diese Arbeit keine „Produktschau“ sein, sondern eine Zusammenstellung von Basis-Technologien und ein Abbilden der Konzepte auf diese enthalten.

Diese Unterscheidung tritt beim klassischen EAI-Ansatz besonders deutlich hervor. Natürlich kann man sich selbst ein EAI-Tool mit Hub-and-Spokes-Architektur entwickeln, im Allgemeinen wird man jedoch auf fertige Lösungen zurückgreifen. Dabei ist es für den Fokus dieser Arbeit nebensächlich, welche Features einzelne Lösungen bieten; entscheidend ist, welche Basis-Technologie jeweils für die Kommunikation genutzt wird.

Ein Produkt dieser Art ist der IBM WebSphere Message Broker [IBM06b]. Er kann als Enterprise Service Bus (ESB) innerhalb einer SOA[20], aber auch als klassischer Broker benutzt werden. Für die Kommunikation wird WebSphere MQ (vgl. 3.4) verwendet; das „Andocken“ von Anwendungen an den Broker kann auch über JMS geschehen.

Auch der Microsoft BizTalk Server 2006 [Mic06] kann als eine Art ESB[21], aber auch als klassischer EAI-Broker benutzt werden. Zur internen Kommunikation wird mit SOAP zumindest die Basis von Web Services (vgl. 3.5) eingesetzt. Für die Kommunikation nach außen existieren zahlreiche Adapter, beispielsweise für Web Services, Dateien, HTTP, MSMQ[22], IBM WebSphere MQ (also sogar ein Konkurrenz-Produkt) und SQL.

Grundsätzlich ist der Aufbau einer EAI-Lösung mit vielen verschiedenen Technologien möglich. Am besten geeignet sind Nachrichten-orientierte Lösungen, da auf diese Weise die Nachrichten vom EAI-Broker einfach an die richtige Anwendung weiter geleitet werden können. Aber auch RPC-orientierte Technologien wie RMI oder CORBA sind prinzipiell als Basis nutzbar, Technologien zur Datenintegration hingegen weniger.

4.3.4 Vorteile

Der klassische EAI-Ansatz behebt weitgehend alle in 4.2.3 aufgeführten Probleme von Punkt-zu-Punkt-Verbindungen. Das hat folgende Ursachen:

- Bei der Integration eines neuen Systems in eine bestehende, mit EAI verbundene Systemlandschaft muss nicht zu jedem System eine individuelle Schnittstelle implementiert werden, sondern es ist nur eine einzige (bidirektionale) Verbindung zum EAI-Broker notwendig. Es muss lediglich einmal das Mapping der lokalen, anwendungsspezifischen

[20] Mehr hierzu im nächsten Abschnitt (4.4).

[21] Microsoft benutzt ausdrücklich nicht den Begriff ESB; mehr hierzu in [Mic05] sowie im nächsten Abschnitt (4.4).

[22] Microsoft Message Queuing (eine Messaging-Middleware, die grundsätzlich mit dem in dieser Arbeit (3.4) vorgestellten IBM WebSphere MQ vergleichbar ist)

Schnittstellen-Parameter auf das Meta-Format des EAI-Brokers vorgenommen werden; danach ist mit allen Anwendungen, die auch an den Broker angeschlossen sind, eine Kommunikation prinzipiell möglich.

- Der Austausch eines vorhandenen Systems zieht nicht automatisch eine Umimplementierung der mit ihm kommunizierenden Systeme nach sich. Diese kontaktieren schließlich nicht direkt das Zielsystem, sondern den EAI-Broker. Hier ist lediglich das Mapping auf das neue Zielformat sicher zu stellen.
- Die Abschaltung eines bestehenden Systems führt auch nach der Einführung von EAI unverändert dazu, dass alle Versuche, es anzusprechen, einen Fehler erzeugen. Allerdings ist es durch den Einsatz von EAI möglich, die existierenden Verbindungen zu überwachen und somit ex ante, also vor der Entscheidung über die Abschaltung, die Konsequenzen aufzuzeigen.

4.3.5 Nachteile

Obwohl der klassische EAI-Ansatz – wie in 4.3.4 angesprochen – grundsätzlich in der Lage war und ist, die durch ein Geflecht von Punkt-zu-Punkt-Verbindungen entstandenen Probleme zu lösen, wurden mit der Zeit andere Probleme, die dieser Ansatz hervorruft, offenkundig.

EAI-Produkte basieren nicht vollständig auf Standards. Das hat zur Folge, dass das EAI-Tool eines jeden Anbieters nahezu einzigartig ist.

Auf die Funktionsweise hat dies keinen Einfluss, aber die Kommunikation mit anderen EAI-Tools ist nur in Ausnahmefällen oder verbunden mit viel Programmierarbeit möglich. Natürlich kann ein Unternehmen die Strategie verfolgen, ausschließlich ein einziges solches Produkt einzusetzen. Aber abgesehen davon, dass dies in großen, heterogenen Unternehmen schwer durchsetzbar ist, macht diese Maßnahme den Zusammenschluss mit anderen Unternehmen (oder deren Aufkauf) nur dann möglich, wenn diese zufällig dasselbe EAI-Tool im Einsatz haben oder aufwändig zu diesem migrieren.

Letztlich wird erfahrungsgemäß allen Bemühungen zum Trotz doch die Situation eintreten, dass die IT-Landschaft aus mehreren isolierten Integrations-Inseln besteht. Diese Konstellation erinnert an das Scheitern des ursprünglichen ERP-Gedankens, der zu genau solchen Silos von wenigen, sehr verschiedenen und zueinander inkompatiblen ERP-Systemen geführt hat (vgl. z.B. [Kay03]). Und ebenso, wie es seinerzeit galt, diese Silos – im Wesentlichen durch den Einsatz von EAI – zu integrieren, müssen jetzt die Integrationslösungen integriert werden. Es ist also offensichtlich nicht gelungen, das Integrationsproblem zu lösen; man hat es nur eine Ebene nach oben verlagert.

Ein weiteres Problem, das aus der Heterogenität der EAI-Tools entsteht bzw. entstanden ist, äußert sich in proprietären Protokollen, die für die Kommunikation eingesetzt werden. Das soeben diskutierte Problem der Inkompatibilität mit den Werkzeugen anderer Hersteller lässt

sich natürlich auch hierauf übertragen, zusätzlich existiert im Bereich der Protokolle aber die Schwierigkeit, dass sie oftmals von den unternehmenseigenen Firewalls abgewiesen werden.

Zwar ist es selbstverständlich möglich, die Firewalls entsprechend zu öffnen, aber durch diese Maßnahme wird eben auch das Sicherheitsrisiko erhöht.

Eine Hub-and-Spokes-Architektur ist grundsätzlich zentral ausgerichtet. Physisch ist der EAI-Hub – u.a. im Hinblick auf die Ausfallsicherheit und Performance – zumeist auf mehrere Maschinen verteilt [Lint04], aber logisch stellt er einen einzigen, unternehmensweiten Anlaufpunkt dar. Dadurch ist es aber von vornherein unmöglich, einzelne Domänen unter eine regionale Kontrolle zu stellen und nur den Datenverkehr, der eine Domäne verlässt, über die zentrale Instanz abzuwickeln (näheres zum Domänenkonzept folgt in 4.4).

Diese Situation ließe sich lösen, indem der Hub nicht nur physisch, sondern auch logisch unterteilt wird. Das würde allerdings bedeuten, dass keine Hub-übergreifenden Prozesse mehr möglich sind.

Auch wenn der EAI-Ansatz ursprünglich ausdrücklich mit dem Ziel konzipiert worden war, vom Denken in Applikationen und Datenbanken hin zur Geschäftsprozess-orientierten Perspektive zu gelangen [Kai02], so sieht die Realität doch weitgehend so aus, dass die IT-Landschaft nach wie vor aus einzelnen Applikationen besteht, die lediglich auf standardisierte Weise miteinander kommunizieren.

Der Grund ist darin zu sehen, dass der Blickwinkel beim EAI-Ansatz nach wie vor von einer Anwendung ausgeht, die mit einer anderen kommunizieren möchte. Es werden nicht abstrakte Funktionalitäten an den EAI-Broker angebunden und aufgerufen, sondern konkrete Applikationen, die es auch vor der EAI-Einführung schon gab und die jetzt lediglich mit einer neuen Schnittstelle versehen worden sind.

Auf diese Weise ist auch die Prozess-Steuerung, d.h. die Verwaltung der Informationen darüber, welche Schnittstelle wann aufzurufen ist, dezentral in den einzelnen Anwendungen gespeichert. Um einzelne Software-Komponenten – egal ob Anwendungen oder Services – zu einem Geschäftsprozess zu verknüpfen und insbesondere auch dessen schnelle Änderbarkeit zu gewährleisten, ist aber eine separate Prozess-Schicht und ein Auslagern zumindest der logischen Routing-Informationen dorthin notwendig.

4.4 Service-orientierte Architektur (SOA)

Das Konzept der Service-orientierten Architektur (SOA) stellt das vorläufige Ende der in diesem Kapitel vorgestellten Entwicklung im Bereich der Anwendungs-Integration dar.

In 4.4.1 wird zunächst der Frage nachgegangen, was genau unter einer SOA zu verstehen ist bzw. wie an verschiedenen Stellen in der Literatur dieser Begriff definiert wird.

4.4.2 beschäftigt sich mit dem Kernelement einer SOA, dem Service. Dabei wird sowohl darauf eingegangen, wie ein solcher Service aufgebaut ist und funktioniert, als auch auf die Frage, welche verschiedenen Arten von Services unterschieden werden können und wie sich ein Service von einer Komponente abgrenzt.

Anschließend werden in 4.4.3 bis 4.4.6 das Zusammenspiel von Services und die Gestaltung einer SOA beleuchtet, ehe in 4.4.7 auf die Frage der zur Umsetzung geeigneten Technologien eingegangen wird.

Der Gegenstand von 4.4.8 ist ein Vergleich zwischen der SOA und dem in 4.3 vorgestellten klassischen EAI-Ansatz.

4.4.9 schließlich widmet sich der Problematik der Einführung einer SOA in einer gewachsenen Anwendungs- und Schnittstellen-Landschaft.

4.4.1 Definitionen aus der Literatur

Ähnlich wie bei EAI (vgl. 4.3.1) gibt es auch für den Begriff der Service-orientierten Architektur keine einheitliche Definition. Im Folgenden werden einige Definitionen analysiert und verglichen.

Der Begriff der SOA tauchte zum ersten Mal gegen Mitte der neunziger Jahre auf. Gartner Research beispielsweise definierte SOA im Jahre 1996 folgendermaßen:

> *"A service-oriented architecture is a style of application partitioning and targeting (placement). It assumes multiple software tiers and usually has thin clients and fat servers (i.e., little or no business logic on the client), but it is more than that. It organizes software functions into modules in a way that maximizes sharing application code and data." [SN96a]*

Es wird also die Wiederverwendung von Anwendungcode und –daten in den Vordergrund gestellt.

Diese Sichtweise von Gartner scheint sich im Laufe der Jahre nicht grundlegend geändert zu haben, wie die im „Gartner Glossary of Information Technology Acronyms and Terms" [Gar04] enthaltene SOA-Definition zeigt:

> *"An application topology in which the business logic of the application is organized in modules (services) with clear identity, purpose and programmatic-access*

> *interfaces. Services behave as "black boxes": Their internal design is independent of the nature and purpose of the requestor. In SOA, data and business logic are encapsulated in modular business components with documented interfaces. […]" [Gar04]*

Dennoch enthält diese Definition einen zusätzlichen Aspekt, nämlich den der Granularität bzw. der fachlichen Orientierung der Services (*"modular business components"*). Dieser wird z.B. von Pallos sehr deutlich in den Vordergrund gestellt:

> *"SOA is the aggregation of components satisfying a business driver. […]" [Pal01]*

An anderer Stelle wird nicht nur die fachliche Orientierung der Services, sondern darüber hinaus auch ihre Verknüpfung zu Geschäftsprozessen als Kern einer SOA dargestellt:

> *"SOA is an architectural style that promotes business process orchestration of enterprise-level business services" [LT03]*

Das CBDI-Forum[23] berücksichtigt den Aspekt der Granularität weniger stark; sie muss lediglich passend für den Nutzer des Service sein:

> *"The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards based form of interface." (CBDI, zitiert bei [SW04])*

Dafür wird hier ein anderes Merkmal hervorgehoben, nämlich die Schnittstelle. Diese muss sowohl standardisiert als auch auffindbar sein.

Die Eigenschaft der Standardisierung steht z.B. bei Datz [Dat04] im Vordergrund:

> *"An architecture built around a collection of reusable components with well-defined interfaces" [Dat04]*

Das W3C [W3C04] betont dagegen ausschließlich die Auffind- und Ausführbarkeit:

> *"A set of components which can be invoked, and whose interface descriptions can be published and discovered" [W3C04]*

[23] CBDI ist nach eigener Darstellung ein unabhängiges Analyse- und Beratungsunternehmen für die Industrie.

Auch Erl [Erl04] charakterisiert SOA über die Standardisierung, allerdings nicht bezogen auf die Schnittstelle, sondern auf die Kommunikation:

> *"An SOA is a design model with a deeply rooted concept of encapsulating application logic within services that interact via a common communications protocol." [Erl04]*

Krafzig u.a. [KBS05] geben als Definition für SOA eine Art „Bauanleitung" an, d.h. eine Aufzählung der logischen Bestandteile, aus denen sich eine SOA ihrer Ansicht nach zusammensetzt:

> *"A SOA is a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation." [KBS05]*

Eine Definition, die sehr viele der soeben beschriebenen Aspekte enthält, ist im "SOA Blueprints Concept" der Middleware Company zu finden:

> *"A SOA can be defined as a way of designing and implementing enterprise applications that deals with the intercommunication of loosely coupled, coarse grained (business level), reusable artifacts (services). Determining how to invoke these services should be through a platform independent service interface." [WH04]*

In dieser Definition taucht zudem ein im SOA-Umfeld sehr wichtiges Stichwort auf: die lose Kopplung (*„loosely coupled"*). He [He03] beispielsweise sieht diese als oberstes Ziel einer SOA:

> *"SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents." [He03]*

In der CBDI-Definition war dieser Aspekt auch schon enthalten (*„services […] are abstracted away from the implementation"*), allerdings nicht als lose Kopplung benannt.

Wie auch in A.6 zusammengefasst, bedeutet lose Kopplung allgemein, dass [Tie06]:

- Sender und Empfänger so weit wie möglich entkoppelt werden sowie
- die Kommunikation möglichst robust, d.h. wenig fehleranfällig ist.

Wie diese Ziele im Einzelnen erreicht werden können, wird in 5.1 im Detail erläutert.

Die nun folgende Tabelle gibt nochmals einen Überblick über die soeben zitierten Definitionen.

Quelle	Was ist SOA?	Wiederverwendung	Granularität	Geschäftsprozesse
[SN96a]	a style of application partitioning and targeting	It organizes software functions into modules in a way that maximizes sharing application code and data.	–	–
[Gar04]	an application topology	the business logic of the application is organized in modules	modular business components	–
[Pal01]	an aggregation of components	–	components satisfying a business driver	–
[LT03]	an architectural style	–	enterprise level business services	promotes business process orchestration
[SW04]	policies, practices, frameworks	enable application functionality to be provided and consumed	granularity relevant to the service consumer	–
[Dat04]	an architecture	reusable components	–	–
[W3C04]	a set of components	components which can be invoked	–	–
[Erl04]	a design model	–	–	–
[KBS05]	a software architecture	–	–	–
[WH04]	a way of designing and implementing enterprise applications	reusable artifacts (services)	coarse grained (business level)	–
[He03]	an architectural style	–	–	–

Tabelle 4: Definitionen für SOA in der Literatur

Quelle	Auffindbarkeit der Schnittstelle	Standardisierung der Schnittstelle	Standardisierung der Kommunikation	lose Kopplung	Bestandteile
[SN96a]	–	–	–	–	–
[Gar04]	documented interfaces	modules with clear […] programmatic-access interfaces	–	–	–
[Pal01]	–	–	–	–	–
[LT03]	–	–	–	–	–
[SW04]	services can be invoked, published and discovered	single, standards based form of interface	–	services […] are abstracted away from the implementation	–
[Dat04]	–	well-defined interfaces	–	–	–
[W3C04]	interface descriptions can be published and discovered	–	–	–	–
[Erl04]	–	encapsulating application logic within services	services that interact via a common communications protocol	–	–
[KBS05]	–	–	–	–	application frontend, service, service repository, and service bus
[WH04]	–	platform-independent service interface		loosely coupled […] services	–
[He03]	–	–	–	loose coupling among interacting software agents	–

Tabelle 4 *(Fortsetzung)*

4.4.2 Services

Services sind die Kernkonzepte einer SOA. Was genau darunter zu verstehen ist und welche unterschiedlichen Arten man unterscheiden kann, wird im Folgenden vorgestellt.

4.4.2.1 Funktionsweise und Eigenschaften

Der Begriff „Service“ ist dafür verantwortlich, dass bei vielen Menschen, die nur am Rande mit dem Thema zu tun haben, der Eindruck entsteht, eine SOA sei gleichbedeutend mit der Verwendung von Web Services. Tatsächlich hat ein Service aber nichts mit einer konkreten Technologie zu tun, sondern lässt sich am besten durch die Übersetzung vom Englischen ins Deutsche als ein Dienst charakterisieren.

Ein Beispiel, das die Funktionsweise eines solchen Dienstes sehr gut beschreibt (und zudem gut in das Umfeld der Entstehung dieser Arbeit passt), ist die Bestellung eines neuen Autos.

Der Kunde findet eine Schnittstelle (das Autohaus oder zumindest das dort auszufüllende Bestellformular) vor, über die er den Dienst in Anspruch nehmen kann. Er muss ihm bestimmte Parameter (Motorisierung, Farbe, Innenausstattung etc.) zur Ausführung an die Hand geben und diese ggf. vorher transformieren (z.B. kann in das Bestellformular als Farbe nicht einfach „black magic perleffect“ eingetragen werden, sondern es ist ein Farbschlüssel zu verwenden).

Nach Eingabe dieser Parameter möchte der Kunde nur noch wissen, wann und wo er sein neues Auto in Empfang nehmen kann. Ob das Fahrzeug in Deutschland oder in einem ausländischen Montagewerk zusammengebaut wurde, ob erst das Lenkrad und dann das Radio oder umgekehrt eingebaut wurde, ob die Sitze in der Endmontagehalle zusammengebaut oder als Ganzes von einem Dienstleister zugekauft werden – all solche Details sind für ihn irrelevant.

Prinzipiell genauso wie in diesem Beispiel die Auto-Bestellung „funktioniert“ ein Service in einer SOA. Die Implementierung eines Service bedeutet ausdrücklich nicht, dass alle Anwendungen, welche die entsprechende Funktionalität bislang erbracht haben, abgeschaltet und neu implementiert werden müssen. Vielmehr wird mit den Services über der existierenden Anwendungslogik eine neue Abstraktionsschicht geschaffen. Jeder Service kann auf standardisierte Art und Weise gefunden und über eine Schnittstelle angesprochen (d.h. der Dienst in Anspruch genommen) werden, wobei die Implementierungs-Details verborgen werden. Diese Implementierung kann auch den Aufruf anderer Services beinhalten und somit deren Funktionalität mit einbinden (siehe zum Thema untergeordnete Service auch 4.4.9.2).

Damit die einzelnen Services die Funktionsweise der SOA in optimaler Weise unterstützen, sollten sie folgende Eigenschaften aufweisen [HJ06]:

1. **Grobgranular**: Effizienter als viele kleine Aufrufe, die erst zusammen den gewünschten Effekt erzielen, ist ein grobgranularer Aufruf.
2. **Vollständig und redundanzfrei**: Die komplette Funktionalität einer Komponente wird in Form von Services zur Verfügung gestellt, wobei niemals dieselbe Arbeit an verschiedenen Stellen gemacht wird.
3. **Idempotent**: Ein mehrmaliger Aufruf eines Service mit identischen Paramtern hat denselben Effekt wie ein einmaliger.
4. **Kontextfrei bzgl. Transaktionen**: Auch wenn ein Service-Aufruf als Transaktion verstanden, d.h. entweder ganz oder gar nicht ausgefürt werden soll, so ist es nicht zielführend, anwendungsübergreifende Transaktionen als Service zu implementieren, da ansonsten die Anwendungen zu stark gekoppelt werden. Stattdessen kann z.B. eine kompensierende Operation angeboten werden.
5. **Kontextfrei bzgl. Sessions**: Ein Service kennt grundsätzlich keine Benutzer und keine Sessions. Ist es notwendig, eine Verbindung zwischen verschiedenen Service-Aufrufen herzustellen, so liegt dies in der Verantwortung des Aufrufers. Die Prüfung von Berechtigungen ist ebenfalls nicht Sache des Service, sondern entweder des Aufrufers oder der System-Umgebung.
6. **Technikneutral**: Der Service, d.h. die Schnittstelle nach außen, beinhaltet keine Rückschlüsse auf die intern verwendete Technologie.

4.4.2.2 Bestandteile und interner Aufbau

Wie in Abbildung 38 grafisch dargestellt, besteht ein Service aus den folgenden Bestandteilen:

- Kontrakt:

  Der Service-Kontrakt beinhaltet Informationen über den angebotenen Dienst, wie z.B. Zweck, Funktionalität, Bedingungen / Beschränkungen und Benutzung [KBS05]. Ein möglicher, wenngleich nicht zwingender Bestandteil des Kontrakts ist eine formelle Schnittstellen-Beschreibung auf der Basis von Sprachen wie IDL oder WSDL (vgl. 3.3 bzw. 3.5).

- Schnittstelle:

  Über die Schnittstelle wird die Anwendungs-Logik, die der Service anbietet, nach außen zugänglich gemacht. Hiermit ist tatsächlich die technische Schnittstelle gemeint; ihre Beschreibung ist Teil des Kontrakts [KBS05].

- Implementierung:

  Die Implementierung ist die technische Realisierung dessen, was im Kontrakt „versprochen“ wird. Sie kann aus Anwendungs-Logik, aber auch nur aus Daten (bzw. Datenbank-Zugriffen) bestehen. Nach außen, d.h. aus Sicht des Service-Nutzers, ist diese Implementierung transparent; einzig die Schnittstelle ist auffind- und nutzbar.

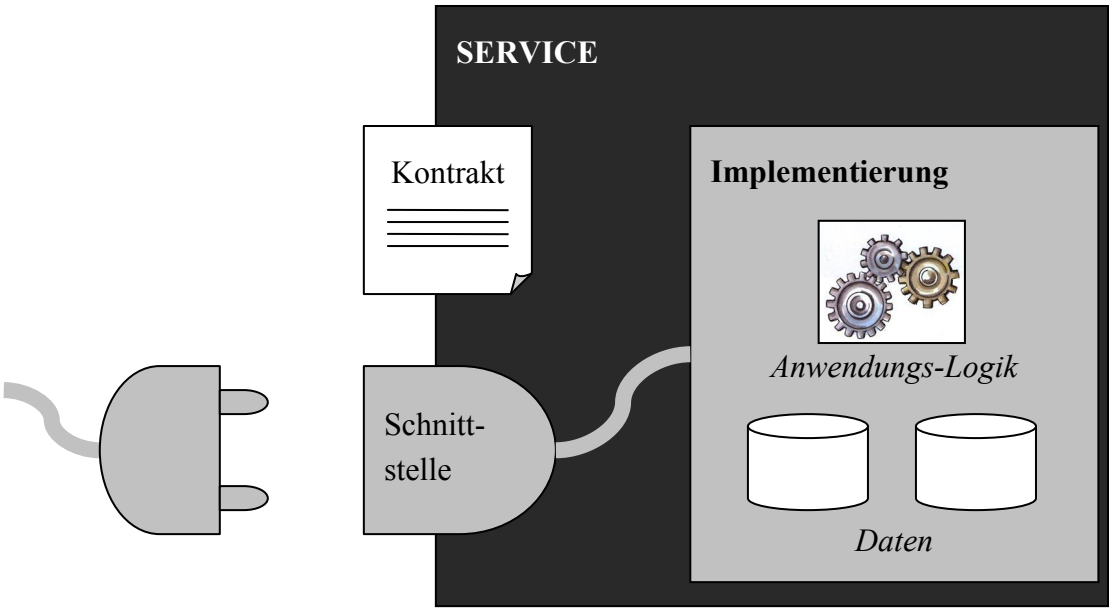

Abbildung 38: Der Aufbau eines Service

Da es in der Praxis diesbezüglich oft zu Missverständnissen kommt, sollen an dieser Stelle drei Dinge ausdrücklich hervorgehoben werden:

- Die Implementierung des Service wird im Allgemeinen nicht speziell hierfür entwickelt, sondern befindet sich in bereits vorhandenen Systemen. Es wird also über Services keine neue Anwendungs-Logik zur Verfügung gestellt, sondern lediglich eine neue, standardisierte und zumeist aggregierte Form des Zugriffs.
- Auch wenn es womöglich in Abbildung 38 anders suggeriert wird, gehört die Implementierung nicht zwingend zu einem einzigen Service. Die vorhandenen Systeme können durchaus von mehreren Services verwendet werden.
- Die Service-Schnittstelle stellt zwar die einzige Möglichkeit dar, um den Service zu nutzen, nicht aber den einzigen Zugangspunkt zu dessen Implementierung. Die vor der Einrichtung des Service vorhandenen Schnittstellen zu den Anwendungen und Datenquellen sind weiterhin vorhanden und können prinzipiell weiterhin aufgerufen werden. Ob ihre Benutzung durch Richtlinien eingeschränkt wird oder einzelne Schnittstellen ganz aus der Implementierung entfernt werden, ist natürlich jeder Organisation freigestellt; konzeptionell oder technisch unabdingbar sind solche Maßnahmen aber nicht.

4.4.2.3 Verschiedene Arten von Services

Aus den verschiedenen Definitionen, die in 4.4.1 analysiert wurden, kann u.a. die Quintessenz gezogen werden, dass eine SOA aus grobgranularen, fachlich ausgerichteten Services besteht. Gleichzeitig wird aber gefordert, dass Services so allgemein und wiederverwendbar wie möglich gestaltet werden. Dies ist aber nur dann zu erreichen, wenn sie möglichst klein und atomar sind.

Also ein Widerspruch? Nicht unbedingt, wenn man sich die in 1.1 vorgestellten verschiedenen Zielsetzungen von Anwendungs-Integration vor Augen hält: Wenn das Ziel eine Flexibilisierung von Geschäftsprozessen ist, sollten die Services in der Tat möglichst grobgranular und fachlich orientiert sein. Doch warum sollte die Nutzung der hierfür aufgebauten Infrastruktur grundsätzlich verboten werden, wenn das Ziel eher die Wiederverwendung von einzelnen atomaren Bausteinen für die Entwicklung neuer Anwendungen ist?

Eine Lösung dieses Problems kann durch die Unterscheidung verschiedener Arten von Services erreicht werden. Während diese Differenzierung an manchen Stellen nur den Einsatzbereich verdeutlicht, kann sie an anderer Stelle dazu führen, dass die Schnittstelle mit einer komplett anderen Technologie gestaltet wird. Außerdem können nichtfunktionale Eigenschaften, wie z.B. die zu leistende Performance oder die Skalierbarkeit, bei verschiedenen Service-Typen variieren.

Im Folgenden werden zunächst zwei Beispiele für solche Klassifikationen aus der Literatur vorgestellt, ehe die im Rahmen dieser Arbeit verwendete Gliederung präsentiert wird.

4.4.2.3.1 „Levels of Component Granularity" nach Herzum

Herzum [HS00a] erarbeitete bereits im Jahre 1999, also weit vor dem Beginn der SOA-Welle, unter der Überschrift *„Levels of Component Granularity"* eine Einteilung in folgende drei Kategorien:

- Distributed Component (Komponenten mit niedriger Granularität, die z.B. mit Hilfe von Enterprise Java Beans oder CORBA implementiert werden können[24])
- Business Component (besteht im Allgemeinen aus mehreren Distributed Components, die zusammen ein autonomes Geschäfts-Konzept implementieren)
- Business Component System (mehrere Business Components, die gemeinsam eine spezifische geschäftliche Anforderung unterstützen)

[24] Die Tatsache, dass Web Services hier nicht erwähnt werden, bedeutet nicht, dass sie grundsätzlich nicht für die Implementierung einer „Distributed Component" bzw. deren Schnittstelle geeignet wären, sondern ist eher damit zu begründen, dass diese Technologie zum Zeitpunkt der Entstehung der Quelle (1999) noch nicht die heutige Bedeutung und Verbreitung hatte.

Auch wenn diese Einteilung in der Tat nicht auf Services oder SOA, sondern auf Komponenten bezogen war, adressiert sie bereits den o.g. Konflikt zwischen fachlicher Prozess-Gestaltung und technischer Wiederverwendung, der im SOA-Rahmen erst die heutige Bedeutung erlangte.

4.4.2.3.2 „Service Types" nach Krafzig et al.

Krafzig et al. [KBS05] unterscheiden folgende „Service Types":

- **Application Frontends**:

  Obwohl dies selbst keine Services sind, stellen sie die aktiven Elemente einer SOA dar. Sie initiieren alle Geschäftsprozesse und empfangen schließlich deren Resultate. Typische Beispiele sind grafische Benutzeroberflächen oder Batch-Prozesse.

- **Basic Services**:

  Diese Services bilden das Fundament einer SOA und repräsentieren die wesentlichen Elemente der vertikalen, fachlichen Domäne.

  Die Autoren unterscheiden zwei Arten von Services, deren Grenzen in der Praxis aber oftmals fließend seien:

  - Data Centric Services (zur Verwaltung von persistenten Daten mit Mechanismen wie z.B. Speicherung, Locking-Funktionalität und Transaktionsmanagement)
  - Logic Centric Services (Algorithmen für komplexe Berechnungen oder Geschäftsregeln)

- **Intermediary Services**:

  Dies sind zustandslose Services, die technische oder konzeptionelle Diskrepanzen bei der Kommunikation zwischen verschiedenen Services überbrücken. Auch hier können verschiedene Unterkategorien unterschieden werden:

  - Technology gateway (zur Überbrückung von technologischen Differenzen, wie etwa von Web Services zu Terminal-Datenströmen)
  - Adapter (zur Abbildung des Nachrichten-Formats)
  - Façade (zur Gestaltung von verschiedenen, zum Teil aggregierten Sichten auf einen oder mehrere Services)
  - Functionality-adding service (zur Anreicherung eines Service um zusätzliche Funktionalitäten, ohne ihn selbst zu verändern)

- **Process Centric Services**:

  Mit dieser Art von Services können verschiedene andere Services zu einem gemeinsamen Prozess zusammengefasst werden. Damit erfüllen sie eine ähnliche Aufgabe wie die Business Component Systems nach Herzum (vgl. 4.4.2.3.1), allerdings ist dort die Prozess-Steuerung nicht explizit enthalten.

  Im Unterschied zu den Intermediary Services ist ein Process Centric Service nicht zustandslos, sondern verwaltet den Prozess-Status für den Benutzer.

  Die Verwendung von Process Centric Services ist für das Funktionieren einer SOA nicht zwingend notwendig, sondern eher eine Hilfe für den Client, da dieser über die Application Frontends die Prozess-Steuerung der einzelnen Services auch selbst in die Hand nehmen kann.

- **Public Enterprise Services**:

  Diese Services stellen Schnittstellen für die unternehmensübergreifende Integration zur Verfügung. Folglich sind sie von sehr grober Granularität. Aufgrund ihres erweiterten Einsatzbereichs gelten bei Public Enterprise Services verschärfte Anforderungen in Bezug auf Sicherheit, Abrechnungs-Verfahren oder Robustheit.

Die Autoren betonen zunächst, dass der Fokus einer SOA auf der funktionalen Infrastruktur und entsprechenden Services, nicht aber auf der Technik liege. Ein geschäftlich orientierter Service in einer SOA beschäftige sich typischerweise mit einem wesentlichen Aspekt des Geschäfts, sei es eine Einheit, eine Funktion oder ein Geschäftsprozess.

Dennoch ist dieser Einsatzbereich recht weit gefasst, denn das Fundament der SOA bilden mit den Basic Services solche, die noch zu feingranular sein können, um daraus einen Geschäftsprozess zu modellieren und die durchaus auch außerhalb einer SOA eine sinnvolle Verwendung finden könnten.

4.4.2.3.3 Zusammenfassung und eigene Kategorien

Zum Abschluss der Betrachtung verschiedener Arten von Services folgt nun in Tabelle 5 ein Überblick über die beiden soeben vorgestellten Klassifikationen sowie zwei weitere, nicht detailliert betrachtete. Ein leeres Feld in dieser Tabelle bedeutet, dass ein vergleichbarer Gliederungspunkt beim jeweiligen Autor nicht vorgesehen wurde, also auch nicht als Untermenge einer anderen Kategorie.

Die Einteilung von Bien [Bie05] bietet keine wesentlichen zusätzlichen Erkenntnisse gegenüber den beiden anderen Darstellungen, und die von Simmons [Sim05] im Rahmen der IBM WebSphere Reference Architecture publizierte Zusammenstellung soll lediglich als Beispiel für eine sehr detaillierte Unterteilung dienen, auf deren genaue Beschreibung aber hier verzichtet wird.

Es wird deutlich, dass insbesondere die Granularität der Unterteilung zwischen den einzelnen Ansätzen erheblich variiert. Idealerweise sollten die Kategorien klein genug sein, um alle grundlegend verschiedenen Anwendungsgebiete widerspiegeln zu können. Auf der anderen Seite ist die Unterteilung nur dann sinnvoll, wenn sich hieraus auch ein Unterschied in der Behandlung oder in der technologischen Umsetzung der in den Klassen enthaltenen Services ableiten lässt. Daher wird in der linken Spalte von Tabelle 5 – quasi als Quintessenz aus den vorgestellten Untergliederungen – eine eigene Einteilung präsentiert, die im weiteren Verlauf dieser Arbeit (Kapitel 5) wieder aufgegriffen wird. Diese umfasst folgende Kategorien:

- **Benutzerinteraktions-Service** (als Schnittstelle für menschliche Benutzer zu der von den anderen Services bereitgestellten Funktionalität)
- **Basis-Service** (mit beliebiger Granularität)
- **Daten-Service** (ebenfalls mit beliebiger Granularität, aber mit dem Ziel, Daten aus verschiedenen Quellen zu liefern und nur äußerst begrenzt Berechnungslogik auszuführen)
- **Geschäfts-Service** (Spezialfall eines Basis-Service: umfasst einen fachlich orientierten, abgegrenzten Prozess-Schritt und stellt folglich den eigentlichen Kern der SOA gemäß den in 4.4.2.1 formulierten Kriterien dar)
- **Transaktions-Service** (sorgt für die Wiederherstellung des Ausgangszustands nach dem Fehlschlagen einer Transaktion über mehrere Services)
- **Vermittlungs-Service** (zur syntaktischen oder semantischen Transformation zwischen Services, also ebenfalls ein Spezialfall eines Basis-Service)
- **Öffentlicher Service** (ein beliebiger Service, der außerhalb der eigenen Organisation verfügbar sein soll und an den folglich besondere Anforderungen bezüglich Sicherheit, Verfügbarkeit etc. gestellt werden müssen)

eigene Kategorie	Herzum (2000)	Krafzig et al. (2005)	Bien (2005) (Java Magazin)	Simmons (2005) (IBM)
Benutzerinteraktions-Service		Application Frontend		*Interaction Service*: Delivery Service, Experience Service, Resource Service
Basis-Service	Distributed Component	*Basic Service*: Logic Centric	Component Service	*Business application Service*: Component Service, Core Service, Interface Service; *Application and information access*: Event detect Service, On-ramp Service
Daten-Service		*Basic Service*: Data Centric	Data Service	*(fällt mit unter "Application and information access)*
Datenmanagement-Service (fällt mit unter Basis-Service)				*Information Service*: Federation Service, Replication Service, Transformation Service, Search Service
(kommt nicht vor)	Business Component	*(fällt mit unter Basic Service)*	Composite bzw. Business Service	*(Geschäfts-Services sind immer aus mehreren anderen zusammengesetzt)*
Geschäfts-Service *(immer zusammengesetzt)*	Business Component System	Process Centric Service	Workflow Service	*Process Service*: Choreography Service, Staff Service
Transaktions-Service			Compensating Service	Transaction Service
Vermittlungs-Service		*Intermediary Service*: Technology Gateway, Adapter, Façade, Functionality-Adding Service		*Connectivity Service*: Event Service, Transport Service, Mediation Service
öffentlicher Service		Public Enterprise Service		*Partner Service*: Community Service, Document Service, Protocol Service

Tabelle 5: Arten von Services in der Literatur und neue Einteilung im Rahmen dieser Arbeit

4.4.2.4 Services vs. Komponenten

Komponenten-basierte Software-Entwicklung verfolgt grundsätzlich ähnliche Ziele wie Service-Orientierung: Bestehende Anwendungslogik wird in standardisierter Form aufbereitet und kann an anderer Stelle wieder verwendet werden. Daher stellt sich die Frage, ob sich hinter SOA vielleicht nur ein neuer Name für ein existierendes Konzept verbirgt oder wo die Unterschiede zwischen Komponenten und Services liegen und wie diese beiden Begriffe im Rahmen dieser Arbeit gebraucht werden. Diesen Fragen wird im Folgenden nachgegangen.

Pallos [Pal01] beispielsweise definiert Komponenten als ausführbare Programme, die bestehende Ressourcen nutzen. Er unterscheidet hierbei folgende Komponenten-Typen:

- Technische Komponenten, die unabhängig von der geschäftlichen Anwendung sind (sie betreffen also z.B. Fehlerbehandlung, Datenkompression oder Sicherheit)
- Geschäfts-Komponenten, die bestimmte Aufgaben auf der Basis von Geschäftsregeln bearbeiten (wie z.B. Kreditkarten-Überprüfung)
- Applikations-Templates, die Pallos als Zusammenstellung mehrerer Komponenten zu einer neuen, großen Komponente definiert

Services dagegen beschreibt er als logische Gruppierungen von Komponenten, um eine geschäftliche Anforderung zu erfüllen. Sie sind im Gegensatz zu Komponenten ausschließlich für die Nutzung aus anderen Programmen heraus, d.h. nicht auch für den direkten Zugriff durch Menschen gedacht.

Folglich umfassen Services nach Pallos' Ansicht mehrere Komponenten. Dieser Meinung sind auch andere Autoren [LT03] [LF03]. In letzterem Beitrag werden darüber hinaus mehrere Unterschiede zwischen Services und Komponenten herausgearbeitet; die wichtigsten hiervon sind folgende:

- Komponenten sind in sich abgeschlossene Software-Konstrukte, die zum Aufbau einer oder mehrerer Anwendungen benutzt werden, d.h. mehrfach deployed werden können. Services hingegen existieren normalerweise innerhalb einer Organisation nur einmal; Redundanz ist allenfalls mit Lastverteilung oder Ausfallsicherheit begründet. Sie werden zwar von Anwendungen genutzt, sind aber für ihre Existenz nicht auf diese angewiesen.
- Komponenten können zustandsbehaftet oder zustandslos sein. Zudem existiert für jeden Aufrufer eine eigene Instanz, d.h. hier liegt das Prinzip der Objektorientierung zugrunde. Services dagegen sind grundsätzlich zustandslos und eher prozedural gestaltet, d.h. es gibt eine einzige Service-Instanz für alle Benutzer.
- Komponenten werden zur Laufzeit von einer Middleware („Componentware") verwaltet, die auch Aufgaben wie Transaktionsmanagement übernimmt. Bei Services hingegen ist eine solche Middleware nicht zwingend notwendig, dafür werden aber üblicherweise spezielle Adapter gebraucht, um sie anzusprechen.

Herzum [Herz01] ist ebenfalls der Ansicht, dass Services und Komponenten nicht ein und dasselbe sind. Der Unterschied liegt aber seiner Meinung nach nicht darin begründet, dass Services grundsätzlich größer als Komponenten sind, sondern in den in Tabelle 6 aufgeführten Punkten.

	Komponenten	Services
Architektur-Perspektive	die inneren Bestandteile eines Systems, die nicht notwendigerweise nach außen publiziert werden müssen	die externe Sicht auf ein System, das intern aus Komponenten bestehen kann, aber nicht muss
Deployment-Modell	Die Software wird physikalisch deployed („install and use“).	Die Software existiert irgendwo („connect and use“).
Umfang des Informations-Austauschs	meistens innerhalb von Unternehmen	meistens zwischen Unternehmen
Kopplung	ziemlich lose, basierend auf internen Standards	sehr lose, basierend auf Industrie-Standards
Kommunikation	Protokolle, die im betroffenen Unternehmen verwendet werden	Internet-basierte Protokolle

Tabelle 6: Vergleich zwischen Komponenten und Services nach Herzum [Herz01]

Auch Meinhold [Mei04] unterstützt diese Sichtweise. Im Rahmen eines Beitrags, in dem er darauf hinweist, dass die Bestandteile einer SOA erst einmal gestaltet werden müssten und dieser Aufgabe viel zu wenig Beachtung geschenkt werde, schreibt er, dass Anwendungen erst einmal intern in Komponenten zerschnitten werden müssten, die dann *„später auch anderen Anwendungen als Services zur Verfügung gestellt werden“*.

Krafzig et al. [KBS05] definieren einen Service als

> *„[…] a software component of distinctive functional meaning that typically encapsulates a high-level business concept“.*

Demnach ist ein Service ein Spezialfall einer Komponente. Noch drastischer wird dies in [HJ06] ausgedrückt:

> *„Die Schnittstellen von Komponenten im Kontext einer SOA nennt man Services.“*

Auch wenn diese Definitionen von den zuerst genannten Darstellungen hinsichtlich der unterschiedlichen Größe von Services und Komponenten abweicht, so ist dies kein Widerspruch. Vielmehr hängt die Sichtweise stark davon ab, in welcher Bedeutung der Begriff der Komponente verstanden wird.

Wenn man beispielsweise bei Wikipedia [Wik06] den Begriff „Komponente“ sucht, erhält man zunächst eine allgemeine Beschreibung:

> *„Eine Komponente (von lat. componere = zusammensetzen […]) ist Teil eines Systems oder kann Teil eines Systems sein.“*

Dies deckt sich mit der Definition als *„Bestandteil eines Ganzen“* bei wissen.de [Wis06]. Zusätzlich zu dieser Definition wird jedoch bei Wikipedia eine Vielzahl von Bedeutungen aufgezählt, darunter die Bereiche der Graphentheorie, der Mathematik allgemein, der Informatik und des Maschinenbaus. Je nachdem, welche dieser Bedeutungen zugrunde gelegt wird, kann man den Begriff der Komponente enger oder weiter fassen, und dementsprechend ergeben sich auch unterschiedliche Abgrenzungen von Services.

Im Rahmen dieser Arbeit wird der Begriff der Komponente in seiner allgemeinsten Form als Teil eines Ganzen verstanden. Die Bezeichnung „Software-Komponente“ sagt somit nur aus, dass es sich um ein abgegrenztes „Stück Software“ (Software-Artefakt, Software-Konstrukt oder wie man es auch nennen will) handelt, nicht aber, wie genau dieses entwickelt und deployed wurde oder welche Granularität es besitzt.

Ein Service ist folglich ein Spezialfall einer Komponente, nämlich eine solche, die eine oder mehrere standardisierte Schnittstellen nach außen anbietet und über diese aufrufbar ist. Dafür können Web Services verwendet werden, es sind aber auch andere Technologien möglich (mehr hierzu später in 4.4.7).

Der Hintergrund für diese Festlegung ist folgender: Im Rahmen einer SOA nicht von Services zu sprechen, wäre konträr zu allen aktuell geführten Diskussionen in der Literatur sowie zum Sprachgebrauch in der Praxis. Jedoch können für die Integration von Anwendungen – wie bereits beschrieben – verschiedene Methoden gewählt werden. Bei Punkt-zu-Punkt-Verbindungen könnte der Begriff des Service noch gehalten werden, nur dass eben die Standardisierung und die Infrastruktur einer SOA nicht vorhanden sind. Spätestens bei der Datenintegration ist die Verwendung dieses Begriffs jedoch nicht mehr möglich: Die Aussage, dass ein Service über Daten-Replikation aufgerufen wird, ist schlichtweg unsinnig. Zwei Komponenten hingegen können durchaus auf diese Art und Weise Daten austauschen.

4.4.3 Das Zusammenspiel von Services in einer SOA

Im Gegensatz beispielsweise zu einem entfernten Methodenaufruf, wie er u.a. mit Java RMI (vgl. 3.2) realisiert werden kann, existiert bei einer Kommunikation zwischen Services im Rahmen einer SOA keine klare Trennung zwischen Client und Server. Natürlich ist zum Zeitpunkt der Kommunikation ein Service der Aufrufer (also der Client) und ein anderer der An-

bieter (also der Server), aber diese Rollenverteilung ist temporär. Jeder Service kann sowohl Client als auch Server sein. Auch bei der Interaktion mit Software außerhalb der SOA, beispielsweise bei der Anfrage eines Client-Programms an einen Service, steht zwar jeweils die Rolle des Clients und des Servers für diese individuelle Kommunikation fest, der Service selbst kann aber bei anderen Kommunikationsbeziehungen eine andere Rolle einnehmen.

Diese Feststellung erscheint auf den ersten Blick eher theoretische Gründe zu haben. Tatsächlich ist sie aber von großer Bedeutung für die Praxis, und zwar dann, wenn es um die Fragestellung geht, ob für die Implementierung von Services auf Basis von bestehender Anwendungslogik diese verändert werden muss oder nicht.

Wenn eine Schnittstelle zu einer bestehenden Anwendung als Service verfügbar gemacht werden soll (also ein neuer „Server“ implementiert wird), so ist es in technischer Hinsicht lediglich erforderlich, eine Software-Schicht zu entwickeln, die den Service-Aufruf entgegennimmt und an die alte, proprietäre Schnittstelle weiterleitet, diese also nach außen kapselt und einschließt. Diese Art von Software wird auch als Wrapper bezeichnet. Das evtl. notwendige Mapping eines zentralen, neutralen Datenformats auf das der Schnittstelle muss nur einmal bei der Implementierung, nicht aber bei jedem Aufruf durchgeführt werden.

Wenn jedoch eine bestehende Anwendung statt der bislang benutzten, proprietären Schnittstelle einen Service nutzen soll, so ist diese Umstellung nicht einfach über Wrapper realisierbar. Schließlich erfolgt der Aufruf im Allgemeinen in hart codierter, auf die Schnittstelle ausgerichteter Form irgendwo im Quellcode. Je nachdem, wie alt die Anwendung ist und wieviel Fachpersonal noch zur Verfügung steht, kann diese Änderung mehr oder weniger aufwändig sein; notwendig ist ein „Anfassen“ des Quellcodes aber in jedem Fall.

Damit ein Service ohne direkten Kontakt mit dessen Entwicklern genutzt werden kann, muss es ein geeignetes Repository geben. Dabei ist es möglich, auf das zurückzugreifen, was die verwendete Technologie quasi mitliefert (z.B. UDDI bei Web Services), stattdessen – oder auch in Ergänzung dazu – kann aber auch ein individuelles Repository für die SOA z.B. in einem Unternehmen eingerichtet werden.

Wenn ein Service in solch einem Repository eingetragen ist, funktioniert sein Aufruf ganz ähnlich zur Vorgehensweise bei Web Services (vgl. 3.5) oder auch CORBA (vgl. 3.3):

Zunächst sucht ein potenzieller Service-Nutzer im Repository nach einem geeigneten Service. Dieser Vorgang muss nicht bei jedem Aufruf auf’s Neue erfolgen, falls ein Service von einer früheren Nutzung bekannt ist. Da aber beispielsweise Veränderungen der Adresse oder des Namens eines Service durchaus vorkommen können, ist es nicht zielführend, die einmal gewonnenen Informationen fest im Quellcode einer Anwendung zu verankern. Gleichwohl ist es in der Praxis bislang noch äußerst selten, dass ein konkreter Service oder gar ein zu einer abstrakten Funktionalität passender Service stets über ein Repository dynamisch gefunden wird.

Nach dem Auffinden erfolgt die Bindung des Service, und anschließend kann seine Funktionalität genutzt werden. Dieses Prinzip, das unter dem Namen „find, bind, execute“ bekannt ist, ist in Abbildung 39 (in Anlehnung an [WB05]) dargestellt.

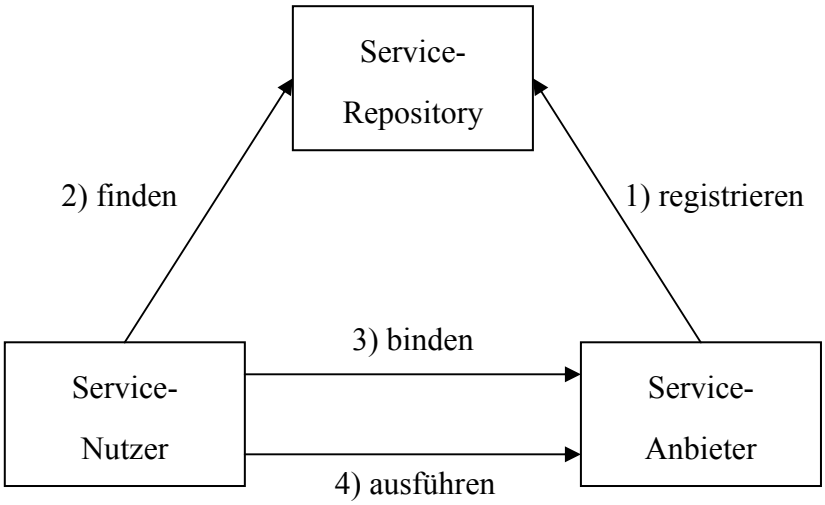

Abbildung 39: Das Auffinden und Benutzen eines Service

4.4.4 Services als Bindeglied zwischen Anwendungen und Prozessen

Die in 4.4.3 beschriebene, technisch orientierte Interaktion sagt noch nichts über die optimale Verwendung von Services aus. Wie in 4.4.2.3 dargelegt, können zwar grundsätzlich verschiedene Arten von Services unterschieden werden, der Hauptnutzen einer SOA liegt aber im Bereich der Geschäftsprozess-Modellierung oder besser gesagt der Verknüpfung der Geschäftsprozesse mit den existierenden Anwendungen.

Ohne SOA müssen die Prozesse direkt mit den IT-Systemen gekoppelt werden, wie es in der linken Hälfte von Abbildung 40 dargestellt ist. Das ist selbstverständlich möglich, aber es ist zum einen eine hohe Detail-Kenntnis der Systeme notwendig, und zum anderen ist sowohl die Änderung eines Prozesses als auch die Ablösung eines Systems nicht trivial.

Im rechten Teil von Abbildung 40 hingegen bilden die Services eine neue, bislang noch nicht existierende Schicht zwischen den Anwendungen und den Prozessen.

Aus IT-Sicht gilt es nunmehr, die im Rahmen des Service, also eines abgegrenzten Prozess-Schritts, benötigte Logik durch Aggregation der vorhandenen Anwendungen (oder auch nur von Teilen davon) bereitzustellen. Wie dieser Service nachher mit anderen Services zusammenarbeitet, spielt dabei keine Rolle und muss somit vom IT-Personal auch nicht verstanden werden.

Aus Prozess-Sicht ist es die Aufgabe, die grobgranularen Services so zu verknüpfen, dass sie die Geschäftsprozesse optimal unterstützen. Kenntnisse der Technik oder der Anwendungslandschaft, die unter diesen Services liegt, sind dabei nicht notwendig. Das gilt auch dann, wenn sich ein Prozess ändert: Vorausgesetzt, dass keine bislang noch gar nicht vorhandene Funktionalität benötigt wird, müssen die benutzten Services einfach nur in eine andere Rei-

henfolge gebracht werden; ein Verändern der IT-Systeme ist jedoch nicht notwendig. Andersherum fällt bei der Ablösung eines IT-Systems keine zusätzliche Arbeit für den Prozess-Verantwortlichen an, da der Service als Schnittstelle nach außen bestehen bleibt und lediglich der Aufruf (oder ein Teil davon) an ein anderes System weiter geroutet wird.

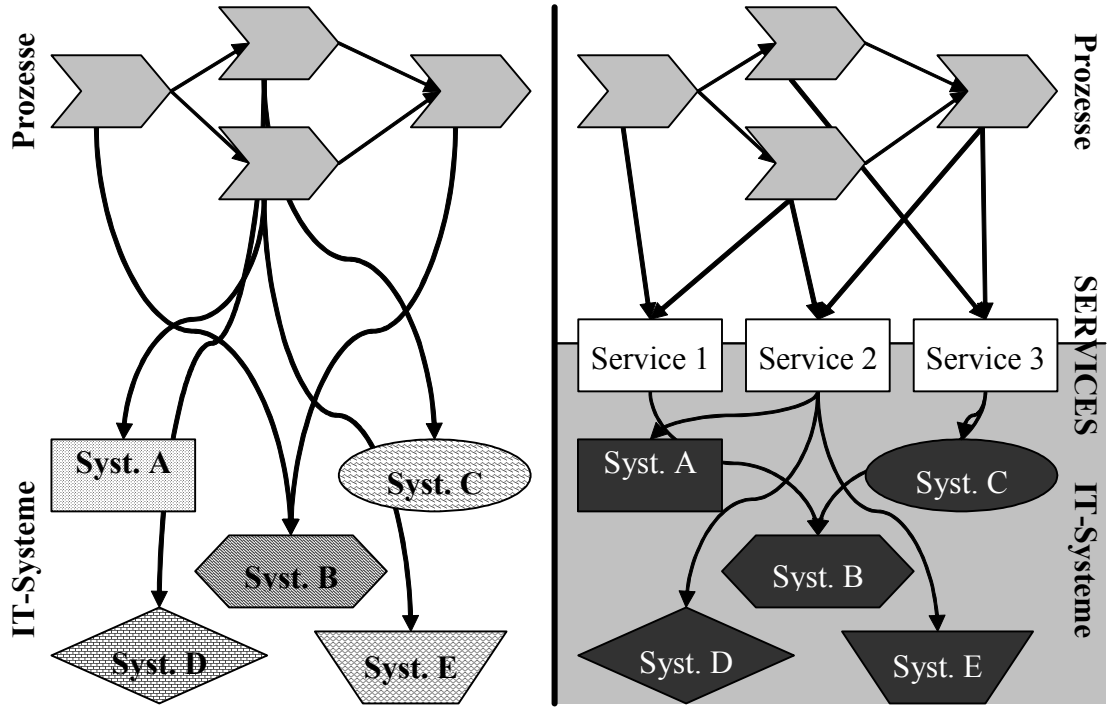

Abbildung 40: Services als Bindeglied zwischen Anwendungen und Prozessen

Es sei an dieser Stelle nochmals ausdrücklich erwähnt, dass eine SOA nicht bedeutet, dass die existierenden Anwendungen ersetzt werden müssen. Es kann sich zwar als notwendig herausstellen, dass einzelne sehr alte Anwendungen, die über keine ausreichenden Schnittstellen verfügen, im Zuge der SOA-Einführung abgelöst werden; generell baut eine SOA aber auf der bestehenden Anwendungslandschaft auf und ergänzt lediglich die erwähnte zusätzliche Schicht, wie es auch aus Abbildung 40 hervorgeht.

4.4.5 Die Zusammenfassung von Services in Domänen

Ein wesentliches Ziel einer SOA ist die Wiederverwendung von bestehender Anwendungslogik bzw. die Vermeidung von redundanter Entwicklung. Um dieses zu erreichen, ist es aber zunächst notwendig, die Masse an vorhandener Funktionalität und an zur Verfügung stehenden Services in geeigneter Art und Weise zu sortieren. Eine Möglichkeit hierzu wäre die Klassifizierung nach der organisatorischen Einheit, die einen Service implementiert hat oder betreibt. Doch dies wäre im SOA-Umfeld noch keine zufrieden stellende Lösung, da sich bestehende Aufgabenbereiche – insbesondere dann, wenn sie sehr grobgranular definiert werden – selten auf eine organisatorische Einheit beschränken.

Vor diesem Hintergrund hat sich die Unterscheidung verschiedener fachlicher Domänen weitgehend durchgesetzt und wird von namhaften Analysten gefordert (vgl. z.B. [Rei06]). Dabei gibt es keinen standardisierten „Baukasten“ mit verschiedenen, vorgefertigten Domänennamen, sondern diese müssen für jedes Unternehmen individuell definiert werden.

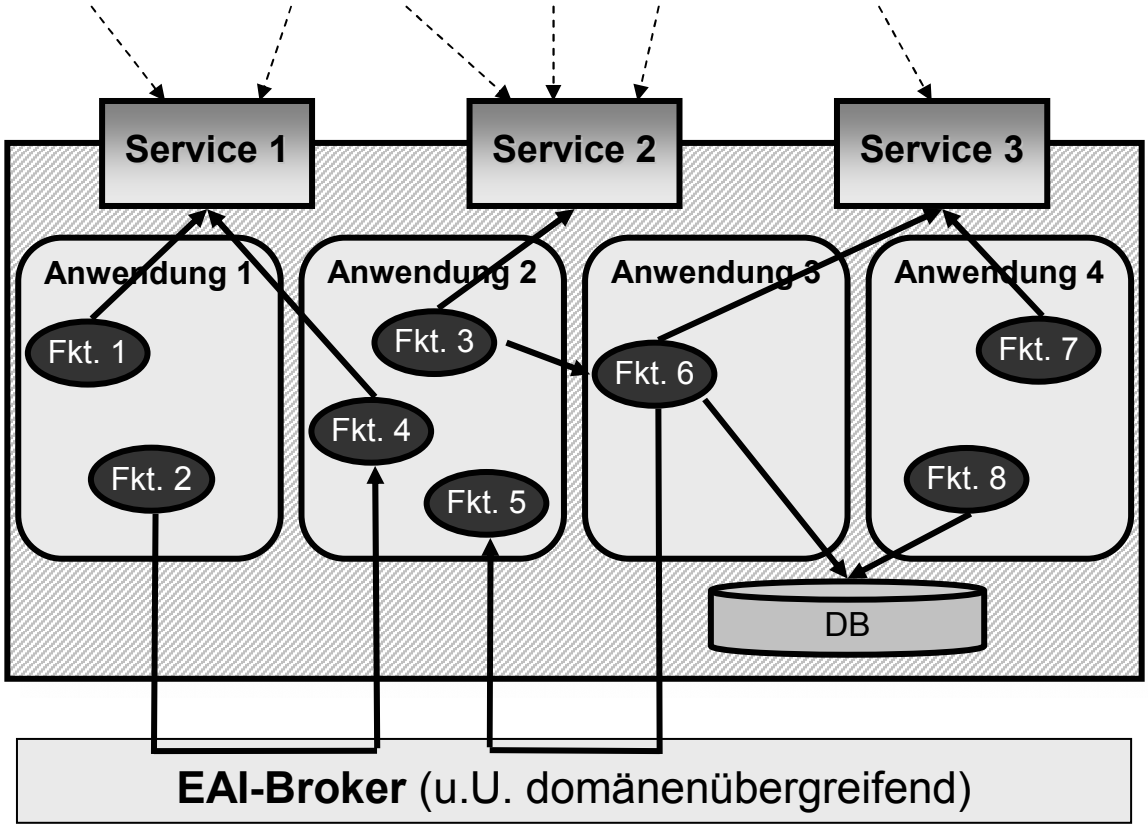

Abbildung 41: Der innere Aufbau einer Domäne

Neben dem angesprochenen Aspekt der besseren Übersicht hat die Gestaltung von Domänen aber noch einen anderen entscheidenden Hintergrund. Wie in 4.4.2 beschrieben, ist ein wesentlicher Vorteil einer SOA und damit der technischen Bereitstellung von Services, dass diese eine sehr lose Kopplung zum Aufrufer zulassen, d.h. dieser muss äußerst wenige Interna kennen und Bedingungen erfüllen, damit er die angebotene Funktionalität nutzen kann.

Aber wann ist eine solche lose Kopplung notwendig und wann nicht? Nachvollziehbarerweise sollte die Kopplung umso loser sein, je weiter das Fachgebiet des Aufrufers von dem des Anbieters entfernt ist. Auf der anderen Seite ist zu erwarten, dass Services, die demselben Fachgebiet entstammen, mehr gegenseitige Anfragen starten, d.h. hier ist beispielsweise der Aspekt der Performance-Steigerung von höherer Priorität als eine losere Kopplung. Dementsprechend hat es sich zur „Best Practice“ entwickelt, die Verbindungen innerhalb der Domänen weniger stark zu standardisieren und eine engere Kopplung zuzulassen; im Gegenzug bieten die Domänen ihre für außerhalb bestimmten Schnittstellen in Form von Services an [Gal04]. Mit anderen Worten, Services stellen die Schnittstellen der Domänen nach außen dar

und kapseln somit nicht nur ihren eigenen internen Aufbau, sondern auch den der eigenen Domäne. Dieser Zusammenhang ist grafisch in Abbildung 41 dargestellt.

Diese Abbildung macht auch noch einmal das Zusammenspiel zwischen Anwendungen und Services deutlich: Es existieren nach wie vor verschiedene Anwendungen (im Beispiel vier), d.h. es wird nicht die bestehende Anwendungslandschaft komplett abgeschafft und in Form von Services neu implementiert. Die in diesen Anwendungen enthaltene Logik wird aber – zumindest Domänen-übergreifend – anders als bisher benutzt oder angesprochen, nämlich unter Verwendung von Services. Diese Services kapseln aber nicht jeweils eine Anwendung, sondern bedienen sich der dort implementierten Funktionalität (in der Abbildung mit „Fkt." abgekürzt). Dabei können Services jeweils einen Teil einer bestehenden Anwendung kapseln (wie im Beispiel Service 2), sie können aber auch Funktionalitäten aus mehreren Anwendungen vereinen (so wie es im Beispiel die Services 1 und 3 machen).

4.4.6 Das verbindende Element: Der Enterprise Service Bus (ESB)

Wie im bisherigen Verlauf dieses Kapitels herausgearbeitet wurde, beruhte der Erfolg der SOAs zu einem großen Teil auf der Vermeidung der Nachteile des klassischen EAI-Ansatzes hinsichtlich seiner zentral ausgerichteten Architektur. Eine SOA in Reinform ist ein vollständig dezentraler Ansatz: Ein Service existiert als abgeschlossene Einheit und kann ohne „fremde" Hilfe (d.h. ausschließlich über ein geeignetes Transport-Medium) andere Services aufrufen.

Es stellte sich allerdings relativ schnell heraus, dass eine solche vollständige Dezentralität in der Praxis auch nicht das Optimum darstellt. Unterstützende Funktionalitäten wie ein Service Repository, die Möglichkeit der Erstellung von Routing-Regeln, syntaktische und semantische Transformationen sowie ein übergreifendes Sicherheitskonzept lassen sich entweder nur zentral aufbauen oder bedeuten einen erheblichen Mehraufwand bei dezentraler Implementierung. Darüber hinaus besteht die Gefahr, bei steigender Anzahl von Services wieder eine Landschaft von Punkt-zu-Punkt-Verbindungen zu haben, nur dass die Endpunkte jetzt nicht mehr Anwendungen, sondern Services sind [Deg05].

Vor diesem Hintergrund wurde mit der Verbreitung des sog. Enterprise Service Bus (ESB) [Cha04] gewissermaßen ein Teilschritt zurück zur Zentralität vorgenommen. Auf eine abschließende Festlegung, ob ein solcher ESB ein konkretes Produkt oder eher ein Architektur-Ansatz ist, konnte man sich bislang in Theorie und Praxis nicht einigen. So verfolgte beispielsweise IBM lange Zeit eine Strategie, derzufolge sich ein Kunde seinen ESB aus den bereits am Markt befindlichen Bausteinen zusammenstellen kann, hat aber mittlerweile doch einen „WebSphere ESB" auf den Markt gebracht. Überhaupt ist die Position der Software-Anbieter zum ESB sehr unterschiedlich: Einige Anbieter, wie z.B. Sonic oder Fiorano, haben sich auf den ESB-Bereich spezialisiert (siehe auch [Rei05]); andere, wie eben z.B. IBM, haben mittlerweile ein entsprechendes Produkt in ihr Portfolio aufgenommen. Microsoft dagegen verzichtet bewusst auf ein Produkt mit dem Namen ESB; in einer Stellungnahme hierzu

[Mic05] begründet das Unternehmen diese Entscheidung damit, dass es viele verschiedene ESB-Definitionen gebe und lediglich die Kernfunktionalitäten unstrittig seien; diese biete aber auch die aktuelle Version von Microsoft Biztalk.

Im Folgenden werden die wesentlichen Funktionen bzw. Eigenschaften eines ESB, über die weitgehend Einigkeit besteht, aufgeführt [Cha04][Mic05]. Dabei werden an verschiedenen Stellen Unterschiede zu den klassischen EAI-Tools herausgearbeitet.

- **Vermittlung der Kommunikation zwischen Komponenten**

  Die Grundfunktionalität eines ESB ist es, die Kommunikation zwischen einem Sender und einem Empfänger zu gewährleisten. Hierbei kann es sich um simple Datenquellen, herkömmliche Anwendungen, Dienste oder ganze Prozesse handeln.

- **Intelligentes Routing**

  Ein ESB sollte nicht nur DNS-Funktionalitäten wahrnehmen, d.h. die aktuell richtige Instanz eines Service finden können, sondern auch das automatische Auffinden des jeweils nächsten Service Ereignis-orientiert und basierend auf einem fachlichen Prozess ermöglichen.

- **Standard-basierte Integration**

  Auch wenn die Entwicklung von ESBs – wie gerade beschrieben – ein Schritt zurück zur Zentralität ist, so ist es doch entscheidend, die Probleme der klassischen EAI-Tools hinsichtlich Hersteller-Abhängigkeit zu vermeiden. Das heißt zum einen, dass für die interne Kommunikation Standards wie der Java Message Service (JMS) eingesetzt werden, und zum anderen, dass man sich nicht darauf festlegt, dass die zu integrierenden Komponenten in Java, .NET, C# o.ä. implementiert sein müssen.

- **Verteilte Daten-Transformation**

  Wenn Sender und Empfänger ein unterschiedliches Datenformat benutzen, so ist die Transformation Aufgabe des ESB. Diese Funktionalität erfüllen auch klassische EAI-Werkzeuge, allerdings auf zentraler Basis, d.h. die Transformationslogik ist Teil des Brokers. Bei einem ESB hingegen werden Services zur Transformation genutzt, die verteilt entwickelt und genauso wie die eigentlichen Kommunikationspartner an den ESB angedockt werden.

- **Endpunkt-Metadaten**

  Ein ESB verbindet nicht nur verschiedene Endpunkte, sondern stellt auch standardisierte Informationen über diese bereit, so dass es leichter ist, einen passenden Dienst zu finden.

- **Prozess-Unterstützung**

  Über einen ESB soll es möglich sein, unter Verwendung von Orchestrierungs-Sprachen wie z.B. BPEL4WS (Business Process Execution Language for Web Services) Geschäftsprozesse auszuführen, die sich der angedockten Komponenten bedienen. Es ist so-

gar machbar, einzelne Teilprozesse lokal zu verwalten, diese aber trotzdem als Teil eines übergreifenden Geschäftsprozesses wirken zu lassen.

- **Sicherheit**

  Die ESB-Kommunikation sollte so gestaltet sein, dass sie nicht von unternehmensinternen Firewalls aufgehalten werden kann. Das bedeutet aber, dass andere Mechanismen zur Gewährleistung der Sicherheit bereitgestellt werden müssen, die zweckmäßigerweise nicht jeder einzelnen Komponente überlassen, sondern durch den ESB zur Verfügung gestellt werden.

- **Dezentrale Konfiguration und Verwaltung**

  Um den Nachteil der klassischen EAI-Tools bezüglich Zuständigkeitsverteilung und Domänenbildung zu vermeiden, sollte ein ESB so aufgebaut sein, dass er sich in verschiedene logische Einheiten unterteilen lässt, die separat verwaltet werden, sich aber nach außen trotzdem wie ein einziger ESB verhalten.

4.4.7 Technologien zur Umsetzung

Wie im bisherigen Verlauf von 4.4 ausführlich beschrieben, ist eine SOA lediglich ein Architektur-Konzept. Zur Realisierung ist natürlich eine geeignete Technologie erforderlich, aber hierfür existieren prinzipiell mehrere Möglichkeiten.

In einem bezogen auf die SOA-Ausbreitung sehr frühen Beitrag von Gartner Research [SN96b] wurden mehrere Typen von Middleware unterschieden, die grundsätzlich geeignet sind, eine SOA zu implementieren:

- TP-Monitore, wie z.B. CICS
- Datenbankmanagement-Systeme, mit deren Hilfe Stored Procedures implementiert werden können
- RPC-basierte Systeme: Hier wird explizit DCE (Distributed Computing Environment) erwähnt; aus heutiger Sicht sind einige Varianten mehr zu nennen, wie z.B. RMI oder auch Web Services
- Middleware, die auf Komponenten oder verteilten Objekten basiert, wie z.B. CORBA

Konkrete Nachrichten-orientierte Middleware (wie z.B. MQ) wird zwar nicht explizit erwähnt, aber dafür ist die Empfehlung zu lesen, dass verbindungslose, nachrichtenorientierte Middleware einer verbindungsorientierten vorzuziehen sei – und zwar aufgrund von Argumenten, die heutzutage unter den Begriff „lose Kopplung“ fallen würden, d.h. im Wesentlichen die Entkopplung von Sender und Empfänger.

Ebenso wie Kapitel 3 nicht jede mögliche Basis-Technologie abdecken konnte, so würde auch an dieser Stelle eine Betrachtung **aller** Technologien, mit denen eine SOA prinzipiell umgesetzt werden kann, zu weit führen.

Stattdessen wird zunächst in 4.4.7.1 erläutert, worin genau der Unterschied zwischen einer SOA und der bloßen Verwendung von Web Services liegt. Anschließend werden in 4.4.7.2 mit Web Services (vgl. 3.5), CORBA (vgl. 3.3) und MQ (vgl. 3.4) drei Beispiele für verschiedene Umsetzungen erläutert. Es handelt sich dabei lediglich um einen Überblick, der besonders signifikante oder bemerkenswerte Punkte hervorhebt. Detaillierte Ausführungen zu den technologischen Umsetzungsmöglichkeiten einer SOA sind beispielsweise in [NL05] oder [Erl04] zu finden.

Es sei an dieser Stelle nochmals auf die drei Dimensionen der Anwendungs-Integration (vgl. 2.5) verwiesen: Eine mögliche Ausgestaltung dieser drei Dimensionen wäre eine SOA (1. Dimension) mit einem Zugriff auf die bestehenden Anwendungen auf der Ebene der Anwendungslogik (2. Dimension) unter Verwendung von Web Services (3. Dimension). Sollte man sich entscheiden, statt Web Services lieber CORBA zu verwenden, so hat dies lediglich einen Wechsel bei der 3. Dimension zur Folge. Dies Beispiel zeigt die guten Strukturierungsmöglichkeiten, die sich durch diese dreidimensionale Einteilung ergeben.

4.4.7.1 Web Services vs. SOA

Wie bereits mehrfach im Verlauf dieser Arbeit erwähnt, werden in der Praxis oftmals SOA und Web Services in unzulässiger Art und Weise miteinander vermischt. Grundsätzlich sind Web Services eine eigenständige Technologie. Doch sie bieten viele Eigenschaften, aufgrund derer sie sich für die Umsetzung einer SOA anbieten. Hierzu zählen im Wesentlichen die folgenden [SW04]:

- Technologie-Neutralität
- Standardisierte Protokolle
- Mechanismen zum automatisierten Finden und Benutzen

Doch eine funktionierende SOA beinhaltet – wie im bisherigen Verlauf von 4.4 herausgearbeitet wurde – noch mehr Anforderungen. Die folgenden Eigenschaften können zwar auch unter Verwendung von Web Services umgesetzt werden, sind aber nicht durch deren Einsatz an sich gewährleistet [SW04]:

- Wiederverwendung auf der Service- und nicht auf der Code-Ebene
- Formale Kontrakte zwischen Service-Anbieter und -Nutzer
- Angemessene Granularität

4.4.7.2 Web Services, CORBA und WebSphere MQ im SOA-Umfeld

Grundsätzlich ist eine SOA-Implementierung sowohl mit Web Services als auch mit CORBA oder MQ möglich. Allerdings werden einzelne Eigenschaften oder Funktionalitäten nicht von allen diesen Technologien unterstützt bzw. ermöglicht. Im Folgenden werden einige solcher Beispiele aufgeführt [NL05].

Für die Definition der Service-Kontrakte stehen bei CORBA (IDL) und Web Services (WSDL, XML Schema, WS Policy) standardisierte Mittel zur Verfügung, bei MQ hingegen muss man sich mit Text-Dateien oder allenfalls Word oder Excel begnügen.

Auch geeignete Konzepte für ein Daten-Management auf Service-Ebene existieren in MQ nicht. CORBA bietet mit der IDL zumindest Datentypen und ferner die Möglichkeit, die Methoden-Parameter zu validieren. Daten-Suche oder –Transformation, wie sie bei der Verwendung von Web Services mit Xpath und Xquery bzw. XSLT möglich ist, kann CORBA dagegen auch nicht liefern.

Für die Registrierung und Suche von Services stehen bei Web Services (mit UDDI) und bei CORBA (mit dem CORBA Interface Repository bzw. CORBA Naming Service) standardisierte Hilfsmittel zur Verfügung, bei MQ hingegen nicht.

Bezüglich der Sicherheit ist bei MQ lediglich eine Authentifizierung über User-ID und Passwort möglich. Web Services bieten neben der Authentifizierung (über HTTPS oder WS-Security) zusätzlich die Möglichkeit der Autorisierung (über die Security Assertion Markup Language (SAML) und die Extensible Access Control Markup Language (XACML)) sowie der Überprüfung der Vertraulichkeit und Integrität der Daten (ebenfalls über HTTPS oder WS-Security). Auch CORBA bietet alle diese Funktionen über TLS (Transport Layer Security).

Verschiedene Interaktionsmuster sowie Aspekte der Dienstgüte (wie Session-Management oder Lastverteilung) sind mit allen drei Technologien möglich. Einzig beim Service-Level-Management muss bei MQ und CORBA auf Drittanbieter zurückgegriffen werden, während bei Web Services WSDM (Web Services Distributed Management) und WS-Management genutzt werden können.

4.4.8 Vergleich von EAI und SOA

Die Ziele von EAI (vgl. 4.3) und SOA sind trotz unterschiedlicher Konzeptionen relativ gleich: Es soll eine Integration von Anwendungen auf der Ebene der Anwendungslogik erreicht werden. Daher stellt sich die Frage, ob EAI und SOA alternative Lösungsansätze für denselben Einsatzbereich sind oder ob es sich um komplementäre Konzepte handelt. Dieser Frage wird im Folgenden nachgegangen.

Zunächst einmal unterscheiden sich EAI und SOA bezüglich ihrer Architektur: EAI ist ein zentraler Ansatz, d.h. jeder Integrationsweg zwischen zwei Anwendungen führt über einen

zentralen EAI-Broker. Eine SOA dagegen ist in Reinform vollständig dezentral. Mit Hilfe eines ESB (vgl. 4.4.6) kann zwar eine zentrale Komponente installiert werden, trotzdem bleiben die beteiligten Anwendungen bzw. Services wesentlich eigenständiger und loser verbunden als bei EAI.

Diese Unterscheidung sagt aber natürlich noch nichts darüber aus, ob die beiden Ansätze komplementär oder konkurrierend sind. Die Beantwortung dieser Frage ist am besten grafisch möglich, wie im Folgenden veranschaulicht wird.

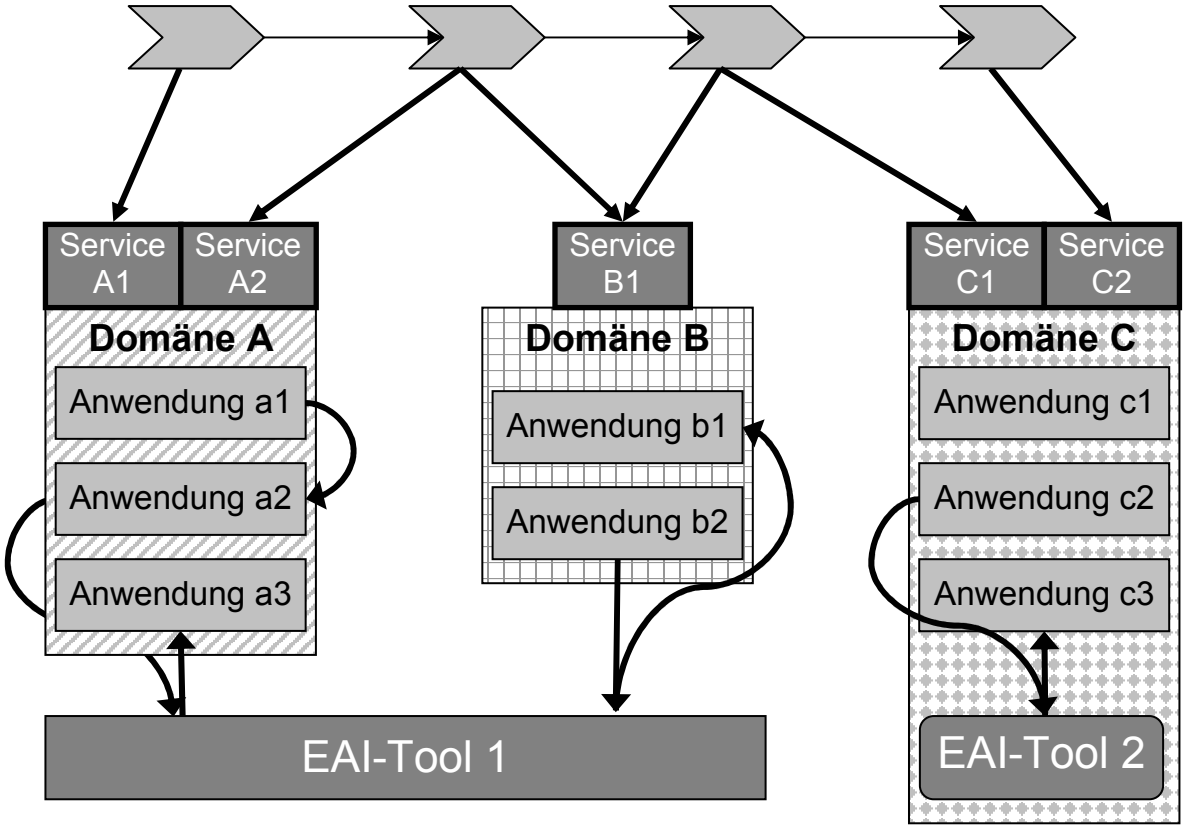

Abbildung 42: Das Zusammenspiel von EAI und SOA

Abbildung 42 zeigt ein mögliches Zusammenspiel von EAI und SOA gemäß ihrer wesentlichen individuellen Stärken: EAI wird für die direkte Kommunikation zwischen Anwendungen innerhalb einer Domäne eingesetzt, für die Verbindungen zwischen diesen Domänen wird auf SOA zurückgegriffen. Im Beispiel existieren drei Domänen, die jeweils eine unterschiedliche Zahl von Anwendungen beinhalten und verschiedene Services bereitstellen. Darüber hinaus sind zwei EAI-Tools im Einsatz; das eine benutzen die Domänen A und B gemeinsam, das zweite ist auf die Domäne C beschränkt.

Für die Innerdomänen-Kommunikation muss natürlich nicht zwingend der klassische EAI-Ansatz gebraucht werden, sondern es ist prinzipiell jede Methode und Technologie denkbar. Dies soll in der Abbildung durch die direkte Verbindung zwischen den Anwendungen a1 und a2 ausgedrückt werden.

In Abbildung 43 ist die gleiche Anwendungs- und Domänenlandschaft dargestellt, und auch der Prozess, den es abzubilden gilt, ist identisch. Allerdings wurde jetzt auf SOA verzichtet und stattdessen vollständig auf EAI gesetzt.

Das Resultat ist, dass die in 4.4.4 beschriebene Schicht zwischen dem Prozess und den Anwendungen, nämlich die Services, nicht mehr da ist. Also müssen die Prozess-Schritte direkt mit den Anwendungen verknüpft werden. Die Domänen-Einteilung ist nun bedeutungslos geworden, da die einzelnen Domänen nicht mehr ihre Funktionalität über Services nach außen kapseln.

Zwar besitzen die meisten EAI-Tools Werkzeuge zur Prozess-Modellierung, jedoch wäre dies im Beispiel nur innerhalb der Domänen A und B möglich, da die Domäne C ja ihr eigenes EAI-Tool einsetzt.

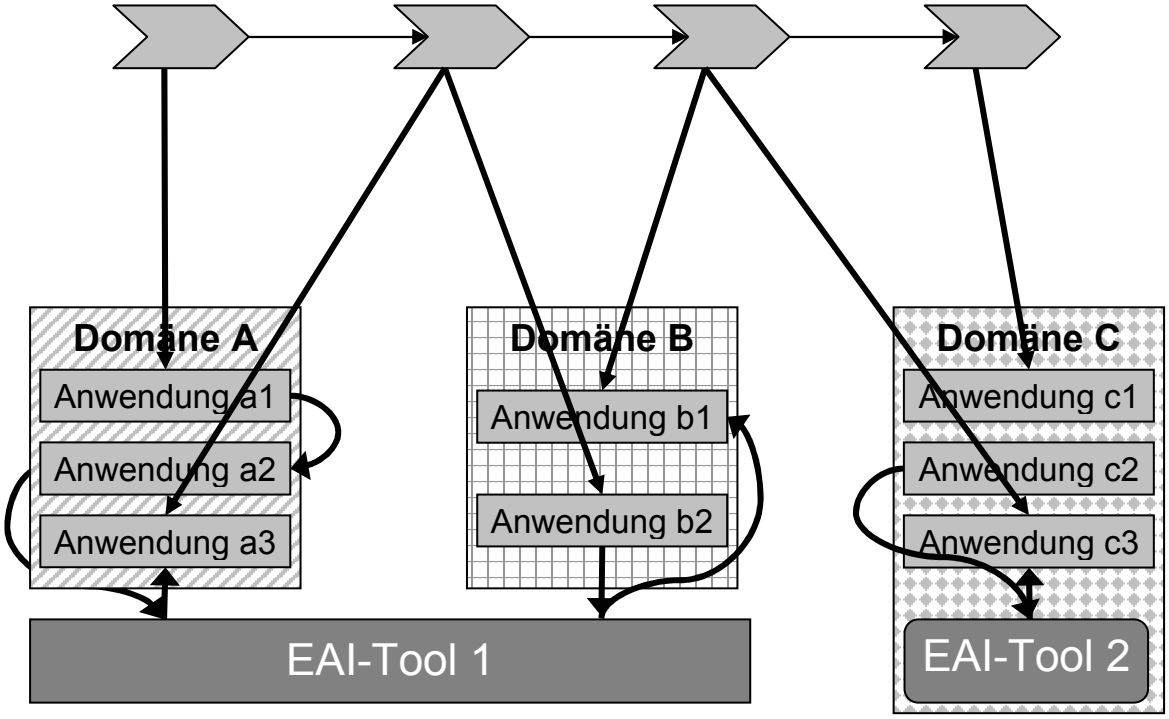

Abbildung 43: Integration ohne SOA

Die Liste der Nachteile, die nicht direkt aus der Grafik ersichtlich sind, ließe sich noch weiter führen und würde schließlich in der Zusammenstellung enden, die in 4.3.5 im Rahmen der generellen Nachteile von EAI vorgestellt wurde.

Also bleibt festzustellen, dass eine Integration zwar auch ohne SOA möglich ist, deren Nutzen aber nicht vollständig durch EAI ersetzt werden kann.

Diese Erkenntnis ist zugegebenermaßen nach den bisherigen Ausführungen in diesem Kapitel keine Überraschung mehr. Schließlich ist SOA in der zeitlichen Reihenfolge nach EAI entwickelt worden und in wesentlichen Teilen eine Reaktion auf die Nachteile dieses Ansatzes. Al-

so wäre es – nicht zuletzt angesichts des derzeitigen „SOA-Hypes“ – höchst verwunderlich, wenn SOA vom „Vorgänger“ EAI vollständig ersetzt werden könnte.

Es bleibt aber die Frage nach dem umgekehrten Fall: Kann SOA die von EAI gebotenen Funktionalitäten vollständig ersetzen oder hat der klassiche EAI-Ansatz nach wie vor seine „Daseinsberechtigung“?

Zur Beantwortung dieser Frage wurde die Grafik aus Abbildung 42 wiederum verändert und alle Nicht-SOA-Bestandteile entfernt. Das Resultat ist in Abbildung 44 zu sehen: Aus Prozess-Sicht ergibt sich kein Unterschied, da die bereits vorhandenen Services ja alle unverändert sind.

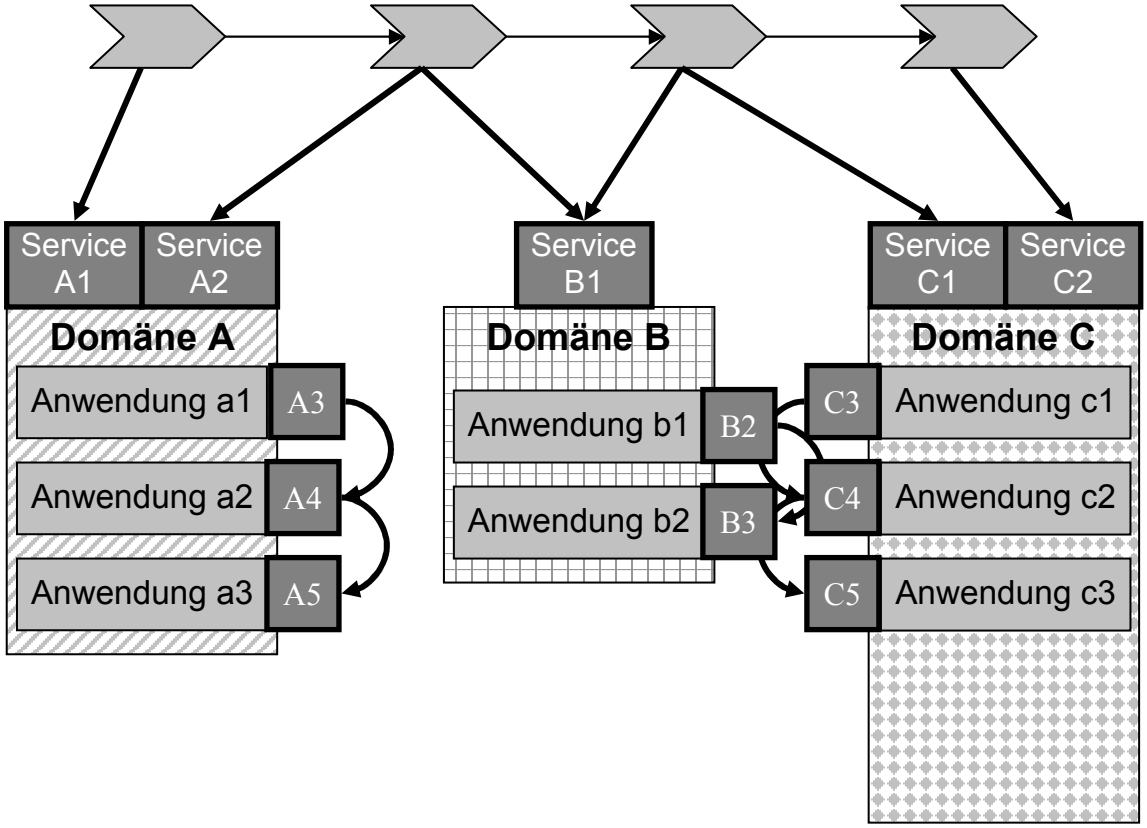

Abbildung 44: Integration ohne EAI

Die Kommunikation innerhalb der Domänen kann nun jedoch nicht mehr über EAI oder andere Wege wie Datenintegration oder Punkt-zu-Punkt-Verbindungen erfolgen. Stattdessen muss jede Anwendung mit einer neuen Schnittstelle (zumindest einem Wrapper zu einer neuen Technologie) ausgestattet und als Service angeboten werden.

Die Folge ist, dass nunmehr eine fast unüberschaubare Menge an Services zur Verfügung steht, von denen der Großteil eigentlich gar nicht für eine SOA geeignet ist, da die Granularität viel zu fein ist.

Man könnte dieses Problem durch eine Einteilung unterschiedlicher Service-Klassen, wie sie in 4.4.2.3 vorgestellt wurde, zumindest ansatzweise lösen. Das macht aber nur dann Sinn,

wenn bei einzelnen Schnittstellen die Vorteile, die eine Service-Orientierung bietet, zum Tragen kommen. Im Beispiel wäre das jedoch nur in Ausnahmen der Fall; die Mehrheit der Anwendungen verwendet die SOA nur, weil keine andere Möglichkeit existiert, und handelt sich dadurch Nachteile in Bezug auf den Implementierungsaufwand und ggf. auch die Performance ein (vgl. 3.6.2).

Noch deutlicher wird diese Tatsache dann, wenn man bedenkt, dass Abbildung 44 eine vereinfachte Darstellung ist: In der Grafik wird angenommen, dass jede Anwendung genau eine Schnittstelle besitzt, die Service-orientiert nach außen zur Verfügung gestellt wird. Dies ist in der verwendeten Darstellung sinnvoll, um die Komplexität nicht unnötig anwachsen zu lassen. Tatsächlich aber widerspricht dies sowohl der Ist-Situation einer gewachsenen IT-Landschaft als auch dem Prinzip der Service-Orientierung, denn hierbei sollen ja gerade keine Anwendungen, sondern Funktionalitäten unabhängig von deren Herkunftsort angeboten werden.

Wenn man den Ausschnitt auf eine Domäne einschränkt, wie es ja in Abbildung 41 der Fall ist, und diese Grafik analog zum Schritt von Abbildung 42 zu Abbildung 44 verändert, also nur noch die SOA als Schnittstellen-Prinzip zulässt, so kommt man zu der Darstellung in Abbildung 45. Hier werden die o.g. Konsequenzen nochmals sehr deutlich.

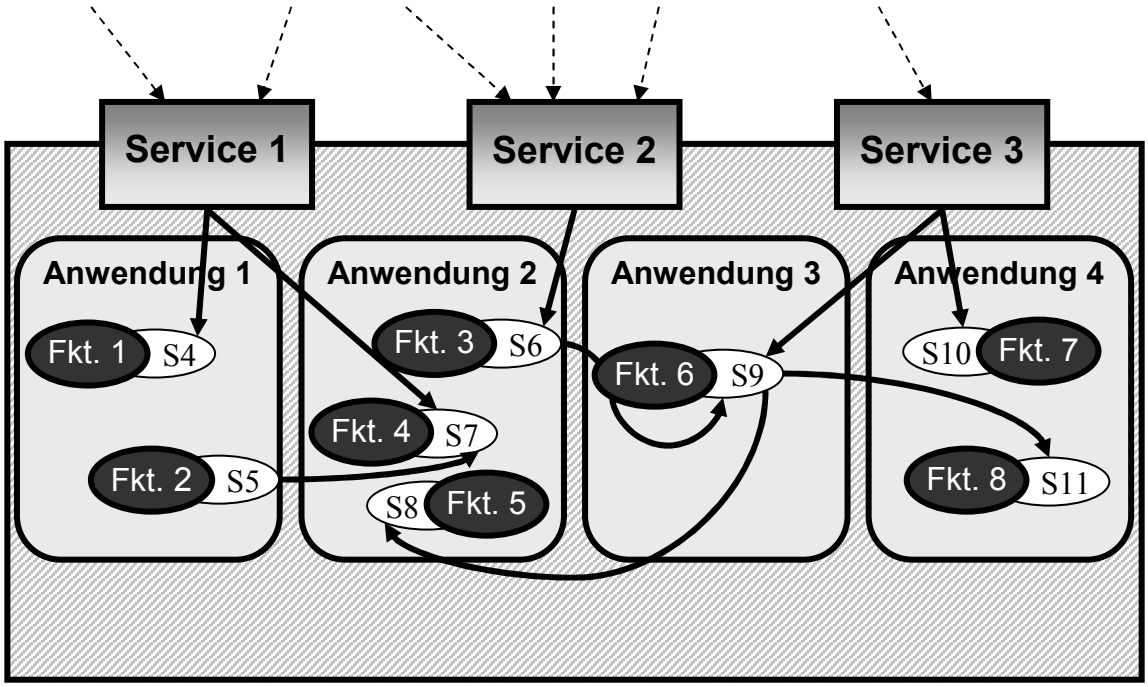

Abbildung 45: Integration ohne EAI aus der Sicht einer Domäne

4.4.9 SOA-Einführung in einer vorhandenen Schnittstellen-Landschaft

Wie in 4.4.8 erläutert, sind also selbst EAI und SOA grundsätzlich komplementäre Konzepte. Damit ergibt sich aber ein neues Problem: Bei der Einführung einer SOA hat man es ja im Allgemeinen nicht mit einer „grünen Wiese“ zu tun, sondern es existieren bereits Systeme und vor allem auch Schnittstellen zwischen diesen. Und wenn nun die Strategie, alles Vorhandene abzuschaffen und auf SOA umzustellen, offensichtlich nicht zielführend ist, ergeben sich plötzlich verschiedene Möglichkeiten des Zugangs zu ein und derselben Funktionalität.

Eine Erläuterung dieser Problematik sowie mögliche Lösungsansätze werden im Folgenden präsentiert. Diese Ausführungen basieren auf [Tie05].

4.4.9.1 Alternative Zugangsmöglichkeiten

In Abbildung 46 ist eine Situation skizziert, wie man sie in der Praxis häufig antreffen wird: Aus einem Teil einer Anwendung (im Beispiel Funktion 3 in Anwendung 2) heraus wird auf eine andere Anwendung bzw. eine dort implementierte Funktionalität (hier Funktion 2 aus Anwendung 1) zugegriffen. Dies wurde im Beispiel in der Vergangenheit über einen EAI-Broker erreicht (Pfeil 1); prinzipiell wäre hier aber genauso gut eine Punkt-zu-Punkt-Verbindung oder eine Datenintegration möglich.

Nun soll im gesamten Unternehmen eine SOA aufgebaut werden. Unter der Annahme, dass die in der Grafik betrachteten vier Funktionen in fachlicher Hinsicht als Service geeignet sind, würden diese mit einem entsprechenden Wrapper ausgestattet (im Beispiel werden Web Services (WS) verwendet), auf den durch einen beliebigen Dienstaufrufer – entsprechende Rechte vorausgesetzt – zugegriffen werden könnte (Pfeil 2).

Im Hinblick auf die eigentlichen Ziele einer Anwendungs-Integration (vgl. 1.1) hat man damit bis auf eine eventuelle Standardisierung aber noch keine Vorteile geschaffen. Im Gegenteil: Die Anzahl der Schnittstellen ist nicht kleiner, sondern sogar größer geworden, denn zum Aufruf von Funktion 2 stehen der EAI-Broker und die Web-Service-Schnittstelle nun als prinzipiell gleichberechtigte Alternativen zur Verfügung.

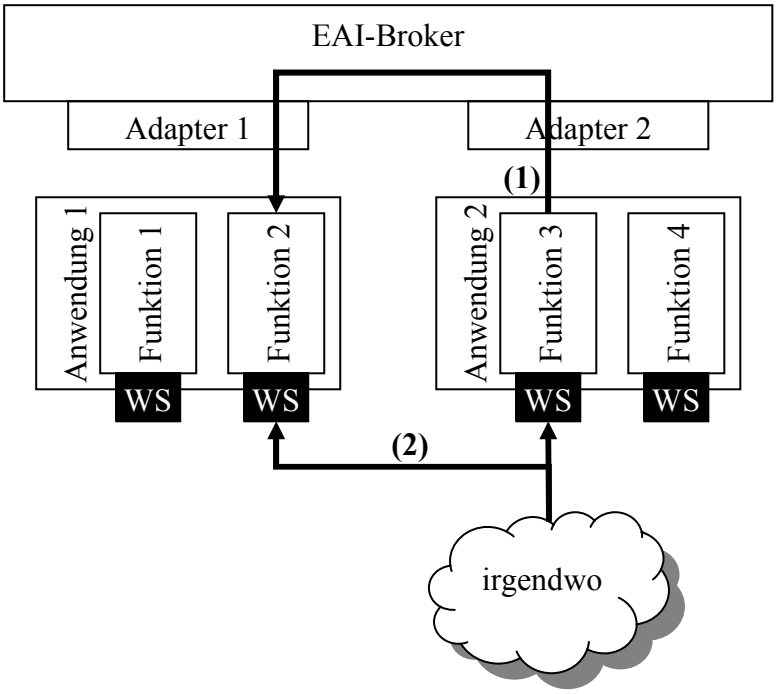

Abbildung 46: Unterschiedlicher Zugriff auf vorhandene Funktionalität (nach [Tie05])

Folglich kann man zu dem Entschluss gelangen, die Entscheidung zu treffen, den EAI-Broker abzuschaffen oder zumindest diese beiden Anwendungen nicht mehr anzudocken. Voraussetzung hierfür ist natürlich, dass der Code von Funktion 3 bzw. Anwendung 2 überhaupt (noch) änderbar ist, denn schließlich muss diese Anwendung so umprogrammiert werden, dass sie sich die Ergebnisse von Funktion 2 nicht mehr aktiv besorgt, sondern als Eingabe erwartet.

Damit hätte man zwar die Erhöhung der Anzahl der Schnittstellen vermieden, dafür aber ein anderes Problem geschaffen: Die Sicherstellung der richtigen Reihenfolge der Aufrufe von Funktion 2 und 3 muss nun durch den Aufrufer umgesetzt werden. Schließlich sind die beiden Web-Service-Schnittstellen absolut gleichberechtigt; ihr Aufruf ist aber zumindest im gegebenen Anwendungsfall nur dann sinnvoll, wenn zunächst Funktion 2 und anschließend – unter Nutzung der Ergebnisse – Funktion 3 aufgerufen wird.

Eine solche Konstellation ist zwar grundsätzlich unter dem Aspekt der Kontextfreiheit bezüglich Sessions (vgl. 4.4.2.1) erlaubt, entspricht aber nicht der Vorstellung von in sich abgeschlossenen Services und dürfte zudem in der Praxis nur schwer auf Akzeptanz stoßen: Schließlich bedeutet die neue Architektur mehr Arbeit für den Aufrufer.

Eine mögliche Lösung dieses Problems ergibt sich dadurch, dass Services mehrschichtig (kaskadiert) gestaltet werden können. So könnte ein neuer Service eingerichtet werden, der nichts weiter erledigt als die beiden vorhandenen Schnittstellen von Funktion 2 und 3 in der richtigen Reihenfolge aufzurufen. Nach außen würde aber dieser neue Service die aggregierte Funktionalität repräsentieren. Diese Konstellation ist in Abbildung 47 zu sehen.

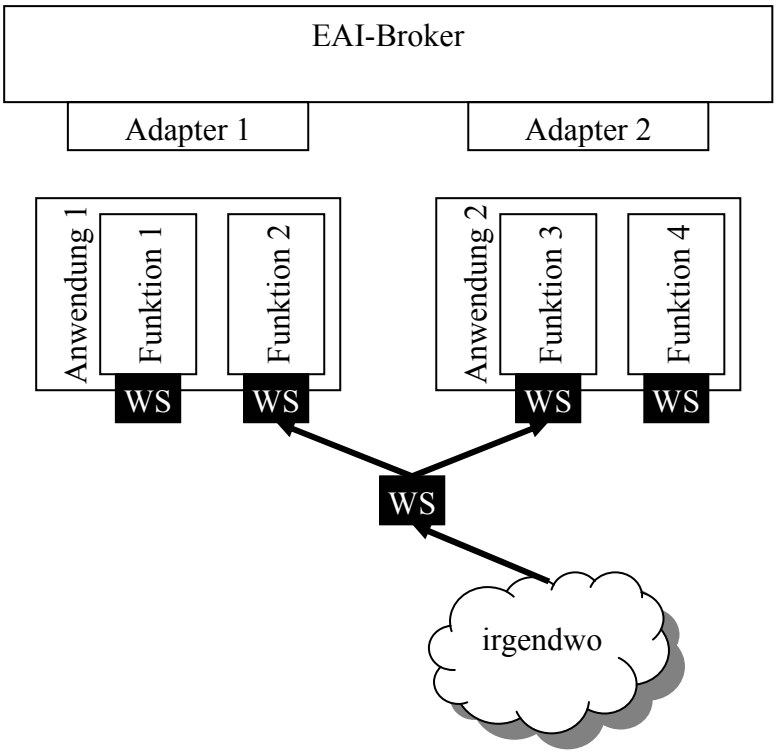

Abbildung 47: Kaskadierte Services (nach [Tie05])

So kann tatsächlich die angesprochene Problematik der Reihenfolge-Erhaltung gelöst werden. Allerdings ergibt sich nun eine bemerkenswerte Situation: Die Funktion des neuen Service als „Fassade“ für die beiden anderen wurde zwar textuell beschrieben, drückt sich aber in keinster Weise in einer möglichen Implementierung aus. Stattdessen gibt es – wie auch anhand der Grafik deutlich wird – drei technologisch gesehen gleichwertige Services, von denen einer über- und zwei untergeordnet zu gebrauchen sind. Und selbst diese beiden untergeordneten sind nicht gleichwertig: Während Funktion 2 in sich abgeschlossen ist und ein Einsatz auch in anderen Szenarien denkbar wäre, ist der Aufruf des direkt mit Funktion 3 verbundenen Service nur in dem hier diskutierten Zusammenhang oder zumindest nicht alleine zweckmäßig. Schließlich macht er nur dann Sinn, wenn vorher Funktion 2 benutzt wurde, und für diese Reihenfolge ist ja gerade der neue Service geschaffen worden.

Ausgehend von diesem Beispiel lassen sich allgemeine Regeln und Vorschläge für die Behandlung von solchen untergeordneten Services definieren. Dies ist Gegenstand des folgenden Abschnitts.

4.4.9.2 Untergeordnete Services

Im vorigen Abschnitt (4.4.9.1) wurde an einem Beispiel bereits die Problematik von über- und untergeordneten Services verdeutlicht. Die grundsätzliche Frage für jede Anwendung bzw. Funktionalität ist, ob es notwendig ist, eine neue, SOA-konforme Schnittstelle zu implementieren, oder ob die bestehende ausreichend ist.

Der erste Schritt besteht hierbei in der Definition eines untergeordneten Service. Dies ist relativ einfach: Ein untergeordneter Service ist ein solcher, der nie direkt von „außen“, sondern nur durch andere Services aufgerufen wird bzw. werden sollte. Wichtig ist das Wort „nie“, denn ein Service, der nur teilweise untergeordnet, sonst aber auf oberster Ebene benutzt wird, kann nicht anders als ein reiner „Top-Level-Service“ behandelt werden.

Trotzdem scheint es angesichts der Erkenntnisse, die das Beispiel im vorigen Abschnitt (Abbildung 47) geliefert hat, offensichtlich unterschiedliche Arten von untergeordneten Services zu geben. Schließlich wurde hier das Ergebnis postuliert, dass der direkt mit Funktion 3 verbundene Service im betrachteten Zusammenhang vollständig durch den neu geschaffenen, übergeordneten Service gekapselt werden sollte. Bei Funktion 2 hingegen sind noch weitere Aufruf-Möglichkeiten denkbar, weshalb hier trotz des untergeordneten Status die Gestaltung einer neuen, mit SOA-Mitteln auffindbaren Schnittstelle zweckmäßig wäre.

Das ist in diesem Beispiel zweifellos korrekt; daher könnte man versuchen, dieses Kriterium zu verallgemeinern und die Anzahl der zugreifenden Komponenten oder die Zahl der Anwendungsfälle, an denen der betrachtete Service beteiligt ist, als Maßstab dafür zu erklären, ob die Implementierung einer neuen Schnittstelle sinnvoll ist. Mit anderen Worten, wenn nur ein Anwendungsfall existiert, kann die vorhandene Schnittstelle weiter benutzt werden.

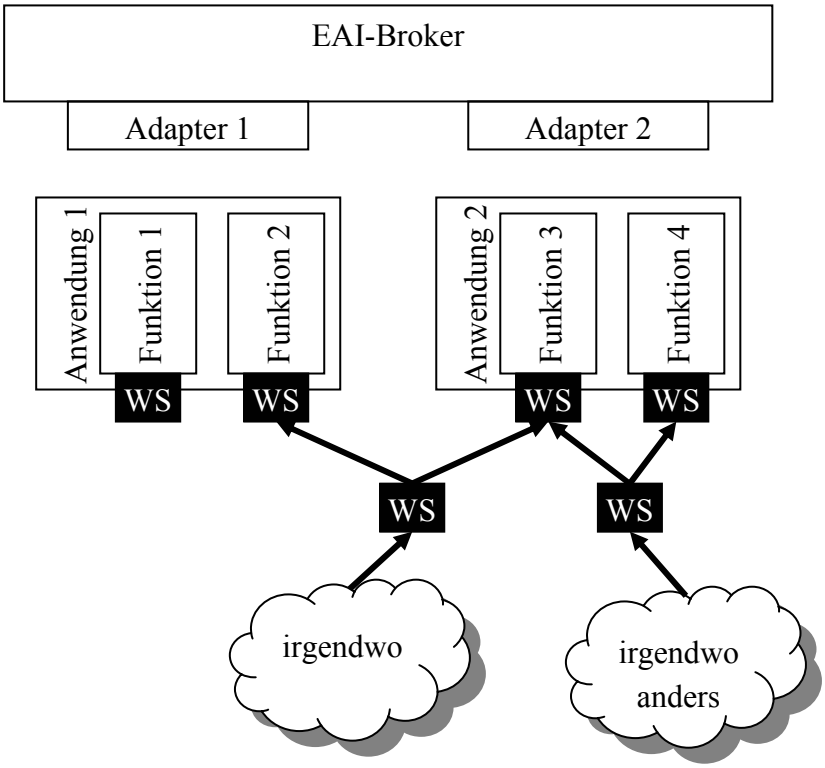

Abbildung 48: Ein Service mit zwei untergeordneten Aufrufen

Dies würde jedoch bedeuten, dass in dem Moment, in dem ein zweiter übergeordneter Service – so, wie in Abbildung 48 skizziert – Funktion 3 benutzt, diese zwingend mit einer neuen Schnittstelle ausgestattet werden müsste. Wie aber anhand der Grafik deutlich wird, ist dieser Schluss nicht richtig, da Funktion 3 auch im neu geschaffenen Anwendungsfall untergeordnet

ist und lediglich die vorgelagerte Funktion 2 durch Funktion 4 unter Benutzung vorhandener Verbindungen ersetzt wurde.

Also besteht der Unterschied nicht darin, in wie vielen Anwendungsfällen ein Service genutzt wird. Vielmehr ist das entscheidende Kriterium die Frage, ob alle diese Anwendungsfälle, d.h. die Aufrufer, vorhersehbar sind. Dabei reicht es nicht aus, neue Aufrufer unmittelbar vor der Nutzung des Service zu identifizieren, sondern deren Kenntnis ist bereits zu dem Zeitpunkt notwendig, an dem die Entscheidung über die Form der Implementierung und Bereitstellung der Schnittstelle getroffen wird. Eine solche Vorhersehbarkeit wird vermutlich in der Praxis zumeist dann der Fall sein, wenn die Funktionalität derart spezifisch ist, dass man mit Sicherheit sagen kann, dass keine weitere Nutzungsmöglichkeit außer den bereits existierenden besteht.

Im Falle, dass tatsächlich alle zugreifenden Services vorhersehbar und die Verbindungen zu ihnen vorhanden sind, ist es in technischer Hinsicht wenig sinnvoll, die alte Schnittstelle abzuschaffen und eine neue zu implementieren, die am Ende exakt die vorher bereits funktionierende Funktion übernimmt. Der einzige Gesichtspunkt, unter dem ein solcher Schritt doch sinnvoll sein könnte, ist eine Standardisierung der Schnittstellen-Technologie innerhalb einer Organisation.

Auf Basis der bisherigen Erläuterungen zeigt Abbildung 49 ein mögliches Vorgehen beim Umgang mit untergeordneten Services.

Wie soeben begründet, ist eine Beibehaltung der bisherigen Technologie unter technischen Gesichtspunkten dann sinnvoll, wenn der Service ausschließlich untergeordnet gebraucht wird, alle zugreifenden Services bekannt bzw. vorhersehbar sind und die Verbindungen dorthin bereits existieren.

Ist es zum Zeitpunkt der Implementierung nicht möglich, eine Vorhersage über die zugreifenden Services zu treffen, so ist es selbst bei existierenden Verbindungen sinnvoll, die Schnittstelle in eine SOA einzugliedern, um nicht durch immer wieder neu hinzukommende Aufrufer ein Gewirr von Punkt-zu-Punkt-Verbindungen zu erzeugen.

Der dritte in der Grafik betrachtete Fall (in der Mitte), nämlich die Situation, dass zwar die zugreifenden Services bekannt sind, aber keine Verbindungen dorthin existieren, ist nicht so einfach auf theoretischer Ebene lösbar. Auf der einen Seite ist es aufgrund der Vorhersehbarkeit der Aufrufer nicht notwendig, die Schnittstelle SOA-fähig zu machen, auf der anderen Seite ist es fraglich, ob die Strategie verfolgt werden sollte, eine neue Schnittstelle bewusst mit proprietären Technologien zu implementieren. Die endgültige Entscheidung wird hier von mehreren Faktoren abhängen, unter denen die Kosten sicherlich ein bedeutender sind.

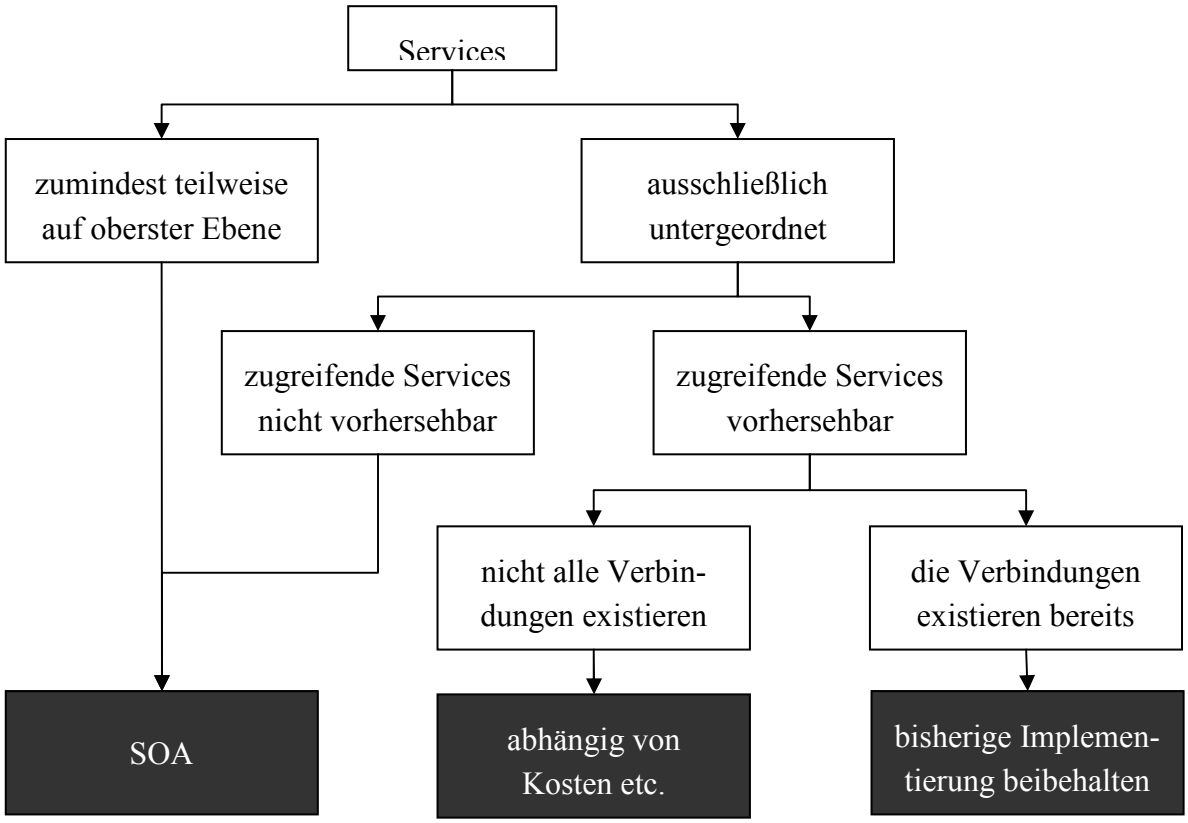

Abbildung 49: Mögliche Strategien bei untergeordneten Services (nach [Tie05])

Auch wenn – wie soeben begründet – im Falle einer fehlenden Vorhersagbarkeit der zugreifenden Services auch für derzeit vollständig untergeordnet gebrauchte Funktionalität die Implementierung einer neuen Schnittstelle sinnvoll ist, so sollte in diesem Fall doch eine von übergeordneten Services abweichende Maßnahme getroffen werden: Für die Auffindbarkeit von Services ist ein zumindest unternehmensweiter Katalog (Repository) hilfreich, der weit über den von UDDI bereitgestellten Funktionsumfang hinaus reicht. Dieser sollte aber zum Zwecke der Übersichtlichkeit ausschließlich solche Services enthalten, die tatsächlich auf oberster Ebene benutzbar sind, nicht aber solche, bei denen nur aus strategischen Gründen eine identische Schnittstellen-Technologie gewählt wurde. Folglich sollten untergeordnete Services mit nicht vorhersehbaren Aufrufern zwar technologisch identisch mit übergeordneten gestaltet, aber in einem eigenen, getrennten Repository verwaltet werden.

Kapitel 5: Die strukturierte Auswahl

Wie in der Einleitung (Kapitel 1) angesprochen, wurden in den letzten beiden Kapiteln lediglich die „Werkzeuge“ vorgestellt, die zur Realisierung einer Anwendngs-Integration zur Verfügung stehen. Der wesentliche Teil dieser Arbeit besteht jedoch in der Entwicklung einer strukturierten Vorgehensweise für die Auswahl einer passenden Methode und einer Technologie für ein konkretes Integrations-Szenario, mit anderen Worten des richtigen Gebrauchs der „Werkzeuge“. Dies ist Gegenstand dieses Kapitels.

Der Kern dieser Vorgehensweise basiert auf dem Begriff der losen Kopplung, der zunächst in 5.1 detailliert erläutert wird.

Danach wird das Vorgehen im Überblick vorgestellt (5.2) und in 5.3 sowie 5.4 im Detail präsentiert.

In 5.5 werden anschließend die Anwendungsmöglichkeiten und Grenzen dieses Modells erläutert, bevor in 5.6 die Anwendung an einem konkreten Beispiel demonstriert wird.

Zum Abschluss wird in 5.7 auf ähnliche Ansätze in der Literatur eingegangen.

5.1 Der Begriff der losen Kopplung

Der Begriff der losen Kopplung tauchte bereits vor der SOA-Welle in der Literatur im Zusammenhang mit föderierten Datenbanken (vgl. hierzu 4.1.2) auf.

Conrad [Con97] beispielsweise versteht unter loser Kopplung eine Daten-Föderation, die jeder Benutzer selbst erstellt. Dementsprechend ist er selber dafür verantwortlich, welche verteilten Datenquellen an dieser Föderation beteiligt sind und in welcher Art und Weise diese Daten – insbesondere in semantischer Hinsicht – benutzt werden.

Enge Kopplung hingegen bedeutet in diesem Zusammenhang die einheitliche Verwaltung der Föderation durch einen Administrator, der die unterschiedlichen Wünsche der Nutzer an die Föderation berücksichtigen und ggf. Kompromisse finden muss. Dadurch nimmt er den Benutzern viel Arbeit ab, allerdings zu Lasten der Flexibilität.

Das für diese Arbeit passende Verständnis des Begriffs der losen Kopplung ist hingegen eher im Bereich der Interaktion verteilter Software-Komponenten zu sehen.

Die Unterscheidung zwischen loser und enger Kopplung und die Definition dieser Begriffe sind von entscheidender Bedeutung im Umfeld der Anwendungs-Integration und der zentrale Punkt innerhalb des in diesem Kapitel beschriebenen Vorgehens. Daher werden in 5.1.1 zunächst die allgemeinen Ziele einer losen Kopplung beschrieben, bevor in 5.1.2 detailliert die Kriterien vorgestellt werden, mit denen diese Ziele erreicht werden können.

5.1.1 Generelle Ziele

Bei der Untersuchung der verschiedenen SOA-Definitionen (vgl. 4.4.1) tauchte der Begriff der losen Kopplung bereits als ein zentrales Merkmal einer SOA auf. Eine Definition dieses Begriffs in einem Satz und eine Abgrenzung zwischen loser und enger Kopplung sucht man in der Literatur jedoch vergebens[25].

Allenfalls findet man eine grobe Beschreibung, was denn mit loser Kopplung erreicht werden soll. Hierbei werden zwei Aspekte bzw. Ziele hervorgehoben [Tie06]:

- Die Verbindung der beteiligten Komponenten soll so robust, d.h wenig fehleranfällig, wie möglich sein.
- Der Aufrufer einer Komponente soll so wenig wie möglich über deren Anbieter wissen müssen, um die Funktionalität nutzen zu können.

5.1.2 Kriterien einer losen Kopplung

Für das Fehlen einer allgemeinen Definition hat Kaye, Autor des Buchs „Loose Coupling" [Kay03], folgende Begründung: „*This is because loose coupling is a methodology or style, rather than a set of established rules and specifications*".

Diese Aussage ist prinzipiell nicht falsch; dennoch ist es möglich, die beiden o.g. Ziele einer losen Kopplung dahingehend weiter zu untersuchen, dass man konkrete Technologie- oder Architektur-Bestandteile aufführt und abgrenzt, durch welche diese Ziele erreicht werden können. Solche Merkmale einer losen Kopplung sind beispielsweise in [Kay03] oder [KBS05] nachzulesen. Eine auf Basis dieser beiden Quellen zusammengestellte Übersicht ist in Tabelle 7 zu sehen; eine detaillierte Erläuterung folgt im weiteren Verlauf dieses Kapitels.

Spätestens beim Betrachten dieser Tabelle wird deutlich, dass der Begriff „lose Kopplung" an sich eigentlich nur für die sehr allgemeine Betrachtung dieses Problembereichs angebracht ist. Eine spezielle Technologie pauschal als lose oder eng gekoppelt zu bezeichnen, ist hingegen viel zu undifferenziert, da dies keine dedizierten Zustände, sondern vielmehr Endpunkte eines Kontinuums sind.

Daher ist die Bezeichnung „Grad der losen Kopplung" [Tie06] angemessener. Jedes der in Tabelle 7 aufgeführten Kriterien erhöht individuell diesen Grad der losen Kopplung. Dabei ist dessen Maximum – also der Fall, dass für alle 8 Kriterien die lose gekoppelte Variante realisiert wird – nur in Ausnahmefällen die optimale Gestaltung einer Schnittstelle. Schließlich hat eine lose(re) Kopplung zwar grundsätzlich Vorteile, ist aber auch mit Nachteilen insbesondere

[25] Zum Teil findet man zwar Ein-Satz-Definitionen von loser Kopplung, die aber nicht ansatzweise alle Aspekte berücksichtigen. Beispielsweise wird lose Kopplung oftmals einfach mit asynchroner Kommunikation gleichgesetzt. Dieser Zusammenhang ist zwar nicht verkehrt, aber es ist nur einer von mehreren Aspekten, wie im weiteren Verlauf dieses Kapitels noch dargelegt wird.

bezüglich Performance oder Implementierungs-Aufwand verbunden (vgl. 3.6.2). Wenn die Vorteile im individuellen Anwendungsbereich gar nicht gebraucht werden, ist folglich die eng gekoppelte Ausprägung dieses Kriteriums die bessere Wahl.

Kriterium	bei enger Kopplung	bei loser Kopplung
1) Kommunikation	synchron	asynchron
2) Nachrichten-Stil	RPC (d.h. Beschreibung der Schnittstelle)	Dokument (d.h. Beschreibung des Inhalts)
3) Bindung	statisch, spätestens bei der Konfiguration	dynamisch, zur Laufzeit (also später)
4) Nachrichten-Wege	bekannt, hart codiert	unbekannt, geroutet
5) Datentypen	abhängig von den Anwendungen	unabhängig
6) Syntax-Definition	direkte Abstimmung	über Schemata
7) Transformation	durch Codierung in den Anwendungen	durch externe Transformatoren
8) Physikalische Kopplung	direkt	unter Verwendung von Middleware (z.B. Message Queues)

Tabelle 7: Kriterien einer losen Kopplung

Ergänzend zu den in Tabelle 7 enthaltenen Kriterien führt Kaye [Kay03] noch drei weitere auf:

- Heterogene Technologien
- Fokus auf breite Anwendbarkeit, nicht auf Wiederverwendbarkeit
- Unerwartete Anfragen

Diese Punkte haben zweifellos mit dem Begriff der losen Kopplung zu tun. Dennoch unterscheiden sie sich von den in Tabelle 7 gezeigten dadurch, dass sie keine direkt beeinflussbaren Kriterien darstellen, sondern Folgen, die durch die Wahl einer lose(re)n Kopplung im Nachhinein entstehen.

Da die weitere Verwendung der Kriterien im Rahmen dieser Arbeit darauf ausgerichtet ist, die Planung einer Schnittstelle zu unterstützen, wurden diese zusätzlichen Punkte nicht mit in die Tabelle aufgenommen.

Im Folgenden werden die in Tabelle 7 im Überblick gezeigten 8 Kriterien näher untersucht und erläutert.

5.1.2.1 Kommunikation

Unter diesem Kriterium ist die Unterscheidung zwischen synchroner und asynchroner Kommunikation zu verstehen.

Synchrone Kommunikation bedeutet, dass der Sender eine Anfrage an einen Empfänger sendet und so lange wartet, bis er von dort eine Antwort zurück erhalten hat.

Bei asynchroner Kommunikation hingegen schickt der Sender eine Anfrage ab und arbeitet anschließend ganz normal weiter („fire and forget“). Die Antwort auf die Anfrage (falls dies überhaupt notwendig ist) kann z.B. als Ereignis gemeldet werden, woraufhin sie bearbeitet werden kann. Der wesentliche Unterschied zur synchronen Kommunikation liegt also darin, dass der Sender nicht blockiert ist, während er auf eine Antwort wartet.

5.1.2.2 Nachrichten-Stil

Bei der Kommunikation zwischen Anwendungen können grundsätzlich zwei Stile unterschieden werden:

- **RPC-Stil**, d.h. es wird eine entfernte Methode mit exakt definierten Parametern aufgerufen, die genau spezifizierte Daten zurück gibt (es wird also eine Schnittstelle beschrieben)
- **Dokument-Stil**, d.h. es wird lediglich ein (vorzugsweise XML-basiertes) Dokument übermittelt, das einen kompletten Vorgang bzw. ein Geschäftsobjekt umfasst und aus dem sich die Zielanwendung die benötigten Daten heraussucht (es wird also keine Schnittstelle, sondern der erwartete Inhalt beschrieben)

Die Unterscheidung zwischen RPC- und Dokument-Stil findet man explizit unter diesem Namen bei Web Services (vgl. 3.5.1); das Prinzip des Verschickens eines Geschäftsobjekts lässt sich aber auch mit anderen Technologien umsetzen.

Auch wenn dieser Unterschied auf den ersten Blick spitzfindig erscheinen mag, ist die Unterscheidung dieser beiden Stile ein entscheidendes Element innerhalb der Anwendungs-Integration und der Wechsel vom RPC- zum Dokument-Stil ein wichtiger Schritt zu einer funktionierenden SOA.

Der Unterschied zwischen diesen beiden Stilen soll anhand eines kleinen Beispiels verdeutlicht werden [Til03][Tie06]: Ein Kunde hat zwei Waschmaschinen bestellt, welche die Bestellnummern 4711 und 0815 haben. Außerdem sind natürlich auch die Rechnungs- und die Lieferadresse zur korrekten Abwicklung der Bestellung notwendig. Im RPC-Stil würde diese Bestellung prinzipiell so aussehen:

```
productFactory.findProductsByCategory
("Washing Machine");

/* Darstellung des Ergebnisses, Auswahl von 2 Produkten aus einer
Liste im Client */

order = orderFactory.create();

order.addProduct(4711);

order.addProduct(0815);

order.setBillingAddress(...);

order.setShippingAddress(...);

order.submit();
```

Dieser Stil gleicht einer Konversation: Der Client sendet dem Server nach und nach die einzelnen Parameter, die dieser braucht, um die Transaktion durchführen zu können; der Server könnte jeweils z.B. mit einer Bestätigung antworten, dass er die Daten erhalten hat (er kann aber auch durchaus auf eine Antwort verzichten).

Allerdings sind nur die ersten beiden Befehle wirklich unabhängig von der individuellen Bestellung; alle anderen bauen auf einem Kontext auf, der serverseitig vorgehalten werden muss. Das führt zu mehreren Nachteilen hinsichtlich des Grades der losen Kopplung:

Die Semantik der einzelnen Aufrufe ist nur dann eindeutig, wenn die „Vorgeschichte“ bekannt ist; dementsprechend wird der Server (bzw. der aufgerufene Service in einer SOA) anschließend kontextabhängig unterschiedlich weiter agieren. Das Prinzip einer SOA aber ist, dass die einzelnen Services zustandslos sind und bei gleichen Parametern auch das gleiche Ergebnis liefern (vgl. 4.4.2.1).

Darüber hinaus ist es – bedingt durch die Struktur der Anfrage in Form von mehreren Befehlen – nur in Ausnahmefällen möglich, die Bearbeitung nachträglich (d.h. ohne explizite Anforderung und sogar ohne das Wissen des Clients) aufzuteilen.

Bei Verwendung des Dokument-Stils würde hingegen ein vorzugsweise XML-basiertes Dokument, das ein vollständiges „Geschäftsobjekt“ repräsentiert, an die empfangende Komponente übermittelt. Dies könnte im o.g. Beispiel die folgende Struktur haben:

```
<order>
  <products>
    <product id="4711" />
    <product id="0815" />
  </products>
  <shipping-address>
     <...>
  </shipping-address>
  <billing-address>
     <...>
  </billing-address>
</order>
```

Durch diese Form des Aufrufs werden alle aufgeführten Nachteile vermieden. So ist es nun vollkommen unproblematisch, z.B. die Rechnungserstellung an einen anderen Service weiter zu leiten, da dieser alle notwendigen Daten des jeweiligen Geschäftsvorgangs zur Verfügung hat, wenn man ihm einfach das Dokument mitschickt. Auch ein Kontext auf Serverseite ist nun nicht mehr notwendig, so dass die Zustandslosigkeit einer SOA umgesetzt werden kann.

Auch wenn solche Schnittstellen durchaus denkbar sind, ist dies natürlich ein Extrembeispiel. Gerade vor dem Hintergrund der Forderung nach grobgranularen, geschäftlich orientierten Komponenten bzw. Services könnte auch im RPC-Stil auf Client-Seite im Beispiel ein Bestellungs-Objekt erzeugt und als Ganzes der Schnittstelle übergeben werden. Dies ist natürlich eine Annäherung zum Dokument-Stil, hat aber zu dessen Idealform den entscheidenden Unterschied, dass das Bestellungs-Objekt nur die Parameter enthält, welche die betrachtete Komponente direkt benötigt, und nicht das gesamte Geschäftsobjekt. Darüber hinaus ist von einer Standardisierung dieses Bestellungs-Objekts über die betrachtete Komponente hinaus nicht auszugehen.

5.1.2.3 Bindung

Die Auswirkungen einer Bindung zur Laufzeit kann sich auch ein IT-Laie am Beispiel des Internets verdeutlichen: Durch das Vorhandensein eines Domain Name Service (DNS) ist es möglich, dass man durch die Eingabe von z.B. „http://www.google.de“ zur Google-Homepage kommt, ohne deren genaue IP-Adresse kennen zu müssen.

Was bis hierhin nur der Benutzerfreundlichkeit und Merkbarkeit der Adressen dient, hat dann einen Einfluss auf den Grad der losen Kopplung, wenn der Fall der Änderung einer IP-

Adresse betrachtet wird. In dem Moment, in dem der DNS entsprechend umkonfiguriert wird, kann die Google-Homepage auf eine andere IP-Adresse umziehen, ohne dass dies vom „normalen“ Internet-Nutzer gemerkt wird; dieser kommt weiterhin durch die Eingabe von „http://www.google.de“ zum Ziel.

Auch wenn dieses Beispiel selbst einfach nur mit Benutzerfreundlichkeit zu tun hat, wird die strategische Bedeutung dann offensichtlich, wenn man es auf den Aufruf entfernter Anwendungslogik aus einem Quellcode heraus überträgt: Die genaue Adresse eines jeden Aufrufs kann sich auch dann prinzipiell jederzeit ändern, wenn die Anwendungslogik an sich unberührt bleibt, wie im Falle des Umzugs auf einen neuen Server. Wie unter 4.2.3.3 erläutert, ist es bei vielen Altsystemen problematisch, auch nur kleinste Änderungen einzuarbeiten. Aber selbst im Falle eines neuen Systems mit noch zahlreich vorhandener Entwickler-Mannschaft ist die Änderung dann in keinem Fall möglich, wenn sie innerhalb von Sekunden nach der Bekanntgabe umgesetzt werden soll.

Spätestens unter diesen Rahmenbedingungen kann nur eine Bindung zur Laufzeit die Lösung sein. Krafzig et al. [KBS05] verdeutlichen deren Einfluss anhand von zwei Grafiken, die ursprünglich mit „traditional application architecture“ und „service-based application architecture“ überschrieben sind, aber nicht nur für SOAs alleine Gültigkeit besitzen. Diese Grafiken sind in übersetzter und leicht veränderter Form in Abbildung 50 und Abbildung 51 zu sehen. Es handelt sich hierbei um das Beispiel einer eCommerce-Anwendung, die Umsatzsteuer- und Versandkosten-Kalkulation durchführt.

In diesem Beispiel – und zwar in beiden Abbildungen – stellt ein externer Anbieter sowohl die Anwendungslogik (d.h. Umsatzsteuer- und Versand-Algorithmen) als auch die Daten für die Umsatzsteuer- und Versandkosten-Kalkulation zur Verfügung. Diese „fremden“ Bestandteile sind in den Abbildungen grau unterlegt.

Im ersten Fall (Abbildung 50) wird die Anwendungslogik zur Entwicklungszeit in die Anwendung integriert. Das kann z.B. mit Hilfe von Bibliotheken geschehen, aber auch durch den Aufruf entfernter Methoden unter Verwendung von Middleware. Der genaue Ort dieser Anwendungslogik spielt folglich in diesem Zusammenhang keine entscheidende Rolle. Die Umsatzsteuer-Daten werden auf CD-ROM geliefert und von dort aus in eine lokale Datenquelle eingefügt; die Versand-Daten werden über das Internet bereitgestellt und periodisch aktuell heruntergeladen.

Diese Aktualisierung erfolgt vielleicht wöchentlich oder täglich. Insofern stellt diese Variante der Verlinkung zur Konfigurationszeit zweifellos eine wesentlich flexiblere Lösung gegenüber der Alternative dar, alle Daten zusammen mit der Anwendungslogik zur Entwicklungszeit fest mit der Applikation zu verlinken. Dennoch ist es – wenn die Applikation einmal produktiv ist – nicht ohne Änderung des Quellcodes (und selbst dann nicht innerhalb von Sekunden) möglich, beispielsweise den Bezugspunkt der Versanddaten zu wechseln.

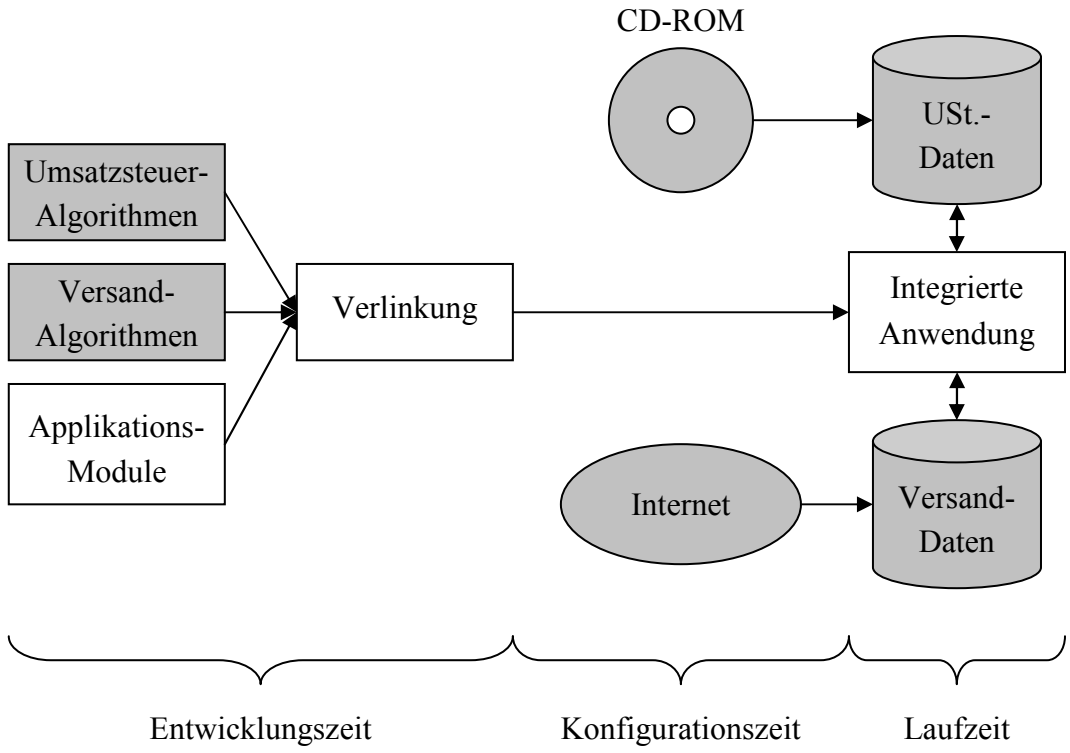

Abbildung 50: Bindung zur Entwicklungs- oder Konfigurationszeit

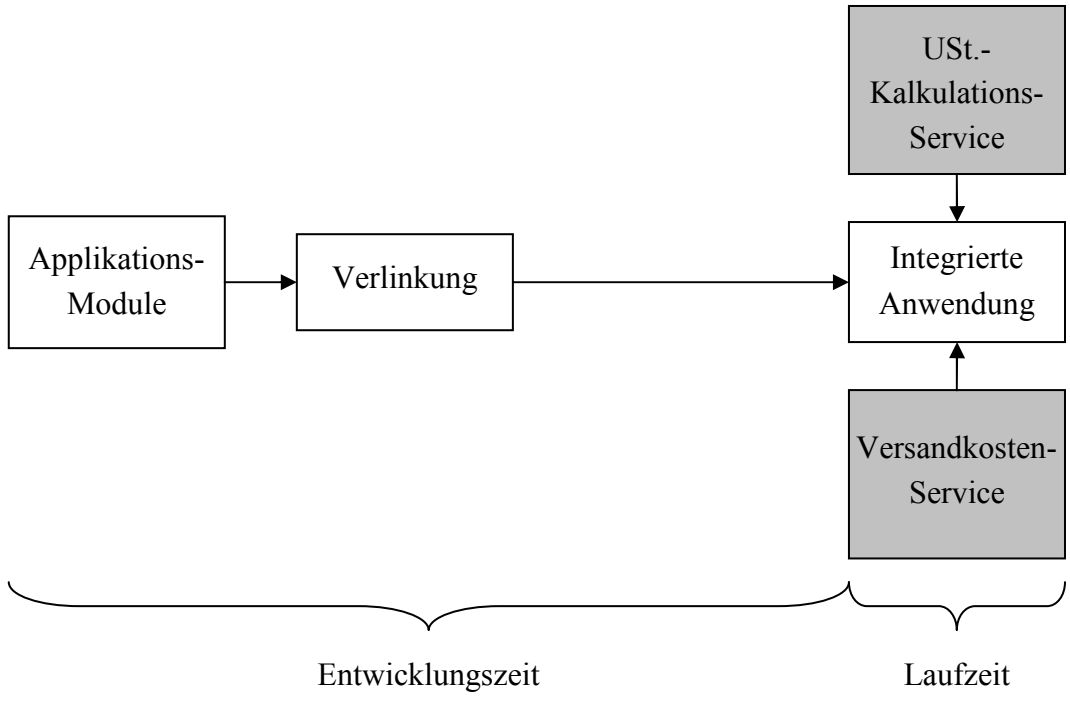

Abbildung 51: Bindung zur Laufzeit

Im zweiten Fall (Abbildung 51) dagegen werden statt der Anwendungslogik und der Daten komplette Services zur Umsatzsteuer- und Versandkosten-Kalkulation fremdbezogen. Dadurch ist nicht nur die Beschaffung der aktuellen Daten aus Applikationssicht transparent,

auch die Einbindung der zur Verarbeitung notwendigen Anwendungslogik erfolgt erst zur Laufzeit, also zum spätestmöglichen Zeitpunkt. Falls eine Änderung des Codes oder der Daten notwendig ist, muss einfach nur die Verlinkung aus der Applikation heraus auf einen anderen Service gelenkt werden.

Die Bezeichnung „Service“ impliziert nicht zwingend die Einrichtung einer SOA. Es ist einfach eine Software-Komponente notwendig, die sich zur Laufzeit einbinden lässt und die die Anwendungslogik mitsamt den aktuellen Daten bereitstellt.

5.1.2.4 Nachrichten-Wege

Die Entscheidung zwischen hart codierten und gerouteten Nachrichten-Wegen hat eine enge Verwandtschaft mit der soeben (5.1.2.3) erörterten Frage der Bindung. Auch hier geht es darum, ob Informationen über eine aufzurufende Komponente zur Entwicklungszeit oder spätestens zur Konfigurationszeit fest in der Anwendungslogik verankert oder zur Laufzeit dynamisch ermittelt werden.

Der Inhalt oder die Reichweite dieser Informationen ist jedoch eine andere: Bei der Bindung steht die Frage im Mittelpunkt, welche Instanz einer Komponente aufgerufen werden soll bzw. wo sich diese befindet. Welchen fachlichen Inhalt sie hat, steht hingegen fest. Bei einer dynamischen Wahl der Nachrichten-Wege ist dies nicht der Fall: Hier wird dynamisch ermittelt, welche fachliche Funktion als nächstes aufzurufen ist.

Der große Vorteil von gerouteten Nachrichten-Wegen ist jedoch die Tatsache, dass bei einer Änderung von Aufruf-Ketten und damit von Geschäftsprozessen die sendende Komponente nicht verändert werden muss (die empfangende sowieso nicht). Es wird grundsätzlich bei einer zentralen Instanz (die z.B. mit einer Process Engine verknüpft ist) nach der nächsten Komponente gefragt; wenn sich diese ändert, wird eine andere Information als zuvor übermittelt und dementsprechend eine andere Komponente aufgerufen.

Somit sind geroutete Nachrichten-Wege von entscheidender Bedeutung, wenn das Ziel der Anwendungs-Integration in der Flexibilisierung von Prozessen liegt (vgl. 1.1).

5.1.2.5 Datentypen

Spätestens im Zuge der Verbreitung des klassischen EAI-Ansatzes mit der Verwendung eines zentralen Brokers (vgl. 4.3) wurde die Notwendigkeit eines einheitlichen Datenmanagements und der Definition von anwendungsunabhängigen Metadaten aktuell. Schließlich ist mit der gelungenen technischen Verbindung der Anwendungen noch nichts darüber gesagt, dass die publizierten Schnittstellen auch richtig verstanden werden und damit die richtigen Ergebnisse liefern bzw. überhaupt arbeiten können.

Wenn die Verbindung zu einer fremden Komponente automatisch oder zumindest halbautomatisch zustande kommen soll, ist neben dem semantischen Verständnis der Bezeichner auch deren syntaktische Vereinheitlichung vonnöten: Dass Kunde, Kundenname, Customer und

KName dasselbe bezeichnen können, ist einem menschlichen Betrachter unmittelbar klar; für eine Maschine sind dies jedoch völlig getrennte Begriffe. Noch komplizierter wird es, wenn auch noch Groß- und Kleinschreiben unterschieden werden, d.h. wenn kundenname etwas anderes ist als KundenName.

Die Notwendigkeit einer solchen individuellen Definition von unabhängigen Datentypen könnte zukünftig entfallen, wenn semantische Web Services eine angemessene Reife und Verbreitung gefunden haben. Hierdurch soll es ermöglicht werden, dass Computer ohne Intervention eines Menschen in der Lage sind, Dienste zu verstehen und semantisch zu verknüpfen.

5.1.2.6 Syntax-Definition

Selbst wenn die Semantik eines Parameters, die im Rahmen der unabhängigen Datentypen (5.1.2.5) eine Rolle gespielt hat, geklärt ist, bleibt die Frage des Formats: Wie muss z.B. der 4. Februar 2003 als Parameter übergeben werden? Die möglichen Varianten sind nahezu unbegrenzt: 04.02.2003, 4.2.03, 04/02/03, 03/02/04, 03/04/02, 04 Feb 03 usw. Bei der dritten bis fünften Variante ist ohne Zusatz (wie z.B. die geografische Herkunft der Datumsangabe) nicht einmal für einen menschlichen Benutzer erkennbar, um welches Datum es sich genau handelt.

Die klassische Variante ist hierbei, dass man sich bemüht, schriftliche Dokumentationen über die Schnittstelle, die man nutzen möchte, zu beschaffen. Wenn diese nicht vorhanden sind oder die gewünschten Informationen nicht enthalten, bleibt nur der Griff zum Telefon, zum Fax, zur Mail etc., d.h. die direkte Kommunikation mit den Personen, die für die Schnittstelle verantwortlich sind.

Auch wenn in diesem Fall eine frei ausformulierte Dokumentation ausreichen würde, bietet die standardisierte Publikation von Schnittstellen-Syntax beispielsweise im XML-Format den großen Vorteil, dass sie in syntaktischer Hinsicht automatisiert ausles- und auswertbar ist und somit auch ein Computer ohne explizites menschliches Auswählen eine geeignete Komponente aussuchen und die notwendigen Parameter ggf. entsprechend umformatieren kann.

5.1.2.7 Transformation

Eine Transformation ist immer dann notwendig, wenn für die Nutzung einer Komponente Informationen benötigt werden, die zwar inhaltlich vorliegen, aber nicht dem geforderten Format entsprechen. Transformationen können sowohl syntaktischer als auch semantischer Art sein[26]. Ein Beispiel für eine syntaktische Transformation wäre die Konvertierung eines beliebigen XML-Dokuments in einen Standard wie EDIFACT[27] (ohne dabei die Inhalte der ein-

[26] Bei [Kay03] heißt dieser Punkt ausdrücklich „Semantic Adaption“, die Erläuterung bezieht aber die syntaktische Seite gleichermaßen mit ein.

[27] EDIFACT („Electronic Data Interchange For Administration, Commerce and Transport“), durch die UN-Schirmherrschaft auch oft als UN/EDIFACT bezeichnet, ist ein branchenübergreifender, internationaler Standard

zelnen Parameter zu ändern), unter semantischer Transformation kann z.B. eine Währungsumrechnung verstanden werden.

Die Bedeutung der Unterscheidung zwischen interner und externer Transformation wird in der Praxis oftmals unterschätzt. Sie soll an einem kleinen Beispiel verdeutlicht werden:

Angenommen, eine Komponente, die intern in Euro rechnet, soll mit einer anderen kommunizieren, die intern in Dollar rechnet und deren Schnittstellen folglich Dollar-Werte erwarten. Die offensichtliche Lösung dieses Problems wäre, entweder der Sender- oder der Empfänger-Komponente eine Methode zur Währungsumrechnung hinzuzufügen. Die Anwendungslogik wäre natürlich denkbar simpel, schließlich ist einfach nur eine Multiplikation von zwei Zahlen durchzuführen. Zuvor muss jedoch der aktuelle Wechselkurs ermittelt werden. Und damit beginnen die Probleme, die selbst mit diesem trivialen Beispiel verbunden sind.

Selbst wenn es grundsätzlich kein Problem ist, eine der beiden kommunizierenden Komponenten um zusätzliche Anwendungslogik zu erweitern, so bedeutet es dennoch das Einfügen vollkommen fachfremder Logik. Und dieser Vorgang ist nicht durch die Funktionalität der Komponente selbst, sondern alleine durch die Notwendigkeit der Kommunikation mit einer anderen begründet. Neben dieser fremden Logik ist es im Allgemeinen zusätzlich erforderlich, aktuelle Daten zu beschaffen; auch die Identifikation von Datenquellen und die Sicherstellung der Aktualität gehören nicht zur Kernaufgabe, die diese Komponente zu erfüllen hat. Darüber hinaus ist es bei einer internen semantischen Transformation nicht ohne weiteres möglich, die implementierte Logik in anderen Komponenten, deren Entwickler vor ähnlichen Problemen stehen, wieder zu verwenden.

Die Lösung dieser Probleme ist das Auslagern der Transformations-Funktionalität in eine eigens dafür entwickelte Komponente. Diese könnte im Beispiel ein Währungstransformations-Service sein. Ein solcher Service muss lediglich beim Aufruf der in Dollar rechnenden Komponente zwischengeschaltet werden; fortan muss weder für die Implementierung noch für die Aktualität und Einheitlichkeit der Daten Sorge getragen werden.

5.1.2.8 Physikalische Kopplung

Wie bereits bei der Vorstellung der Basis-Technologien in Kapitel 3 deutlich wurde, kann eine entfernte Komponente auf zwei unterschiedliche Arten kontaktiert werden: Entweder spricht man sie direkt an (z.B. über einen entfernten Methodenaufruf mit RMI; vgl. 3.2) oder man nutzt ein Zwischenmedium. Dieses kann beispielsweise ein Dateisystem (wie bei RVS; vgl. 3.1) oder eine Message Queue (wie bei WebSphere MQ; vgl. 3.4) sein. Im Falle der Verwendung eines solchen Zwischenmediums würde man von einer indirekten, im anderen Fall von einer direkten physikalischen Kopplung der Anwendungen sprechen.

Auf den ersten Blick ähnelt diese Unterscheidung der zwischen synchroner und asynchroner Kommunikation (vgl. 5.1.2.1) aus. Dennoch sind dies zwei getrennte Kriterien, die unabhän-

für das Format elektronischer Daten im Geschäftsverkehr. Mehr Informationen hierzu sind bei [ISOoJ] zu fin-

gig voneinander erfüllt sein können. So stellt ein unidirektionaler Methodenaufruf (also der Aufruf einer Methode, die keinen Rückgabewert besitzt) eine asynchrone Kommunikation dar, weil die anfragende Komponente nicht auf eine Antwort wartet, sondern nach dem Methoden-Aufruf ganz normal weiter arbeitet. Trotzdem ist dies eine direkte physikalische Kopplung, da nicht z.B. eine Message Queue kontaktiert wird, sondern die Zielkomponente selbst. Trotz der Vorteile, die die Asynchronität mitbringt, bleibt beispielsweise die Gefahr einer temporären Nichterreichbarkeit der angesprochenen Komponente und damit des Fehlschlagens der Anfrage durch Timeout.

den.

5.2 Das Vorgehensmodell

Da die soeben präsentierten acht Kriterien – wie erwähnt – unabhängig voneinander erfüllt oder nicht erfüllt sein können, taugen sie nicht nur zur Definition des Begriffs der losen Kopplung, sondern es ergibt sich die viel interessantere Möglichkeit der Beschreibung und Kategorisierung bestehender Schnittstellen. Darüber hinaus können auf Basis dieser Kriterien sogar detaillierte Anforderungen an noch nicht existierende, zukünftig zu gestaltende Schnittstellen definiert werden.

Aspekte wie Kosten oder Bandbreite spielen natürlich auch eine entscheidende Rolle bei der Auswahl insbesondere von Technologien, aber der Aspekt der losen Kopplung ist für die strukturierte Ausarbeitung von Anforderungen an eine Schnittstelle sogar noch wichtiger, weil er in der Praxis zumeist unterschätzt oder sogar überhaupt nicht betrachtet wird. Dabei berühren die meisten Aspekte, die vor allem im Bereich der Technologien neuere Lösungen im Gegensatz zu älteren Alternativen bieten, genau diesen Grad der losen Kopplung. Und erst wenn so detailliert wie möglich erarbeitet worden ist, welche der hierfür maßgeblichen Kriterien tatsächlich notwendig sind, kann auch entschieden werden, welche Umsetzungsmöglichkeiten infrage kommen.

Aus diesem Grund bilden die acht Kriterien, die Merkmale einer losen Kopplung, den Kern des im Folgenden präsentierten Vorgehensmodells, dessen Grundlagen auch in [FT06] beschrieben werden. Dabei ist es nicht zwingend erforderlich, in jedem Anwendungsfall alle Kriterien zu berücksichtigen. Wenn z.B. kein unabhängiges Datenmodell verfügbar ist, kann auch keine Entscheidung über dessen Verwendung getroffen werden, allenfalls über eine zusätzliche Implementierung. Daher ist es nicht nur geduldet, sondern sogar ausdrücklich vorgesehen, dass nur diejenigen Kriterien betrachtet werden, die im Hinblick auf die individuelle Ausgangssituation sinnvoll sind. Allerdings sollte man sich beim Streichen von Kriterien darüber im Klaren sein, dass dies im Allgemeinen eine Folge auf die optimale Technologie-Auswahl hat.

Auf dieser Basis werden zwei verschiedene Perspektiven unterschieden, die in 5.3 und 5.4 näher erläutert werden:

- **1. Perspektive: Orientierung an den Anforderungen (Soll)**

  Hier werden Mindestanforderungen definiert, welche bei der Kommunikation zwischen zwei oder mehr Software-Komponenten[28] erfüllt sein sollten. Die Grundlage für deren Definition ist neben den zentralen 8 Kriterien die Ausgangssituation (d.h. die vorhandenen Anwendungen bzw. Komponenten); die zur Verfügung stehenden Technologien spielen hier aber bewusst noch keine Rolle.

[28] Zur Begründung des Gebrauchs des Begriffs „Komponente“ (und nicht z.B. „Service“) an dieser Stelle wird auf 4.4.2.4 verwiesen.

Auf der Basis dieser Mindestanforderungen ist es im Anschluss erreichbar, geeignete Möglichkeiten zur Umsetzung zu finden.

- **2. Perspektive: Orientierung an den Umsetzungsmöglichkeiten (Ist)**

  Mit genau diesen Möglichkeiten beschäftigt sich die zweite Perspektive. Hierbei ist zu untersuchen, welche hiervon grundsätzlich im gegebenen Szenario Erfolg versprechend anwendbar wären und welche schließlich die Optimallösung darstellt bzw. darstellen.

  Folglich ist die Reihenfolge dergestalt vorgesehen, dass die erste Perspektive die Basis für die zweite bildet. Aber auch die andere Richtung ist vorstellbar: Die zweite Perspektive kann darüber Aufschluss geben, wie die Mindestanforderungen (und damit ggf. die gesamte Ausgangssituation) verändert werden müssten, damit beispielsweise eine bestimmte Technologie eingesetzt werden kann. Diese Richtung ist allerdings mit äußerster Vorsicht und nach Möglichkeit nur zu Informationszwecken anzuwenden, denn wenn der Anwender keine falschen Angaben vorausgesetzt hat, ist z.B. die schlechtere Eignung einer bestimmten Technologie durch deren konkrete Eigenschaften begründet. Es besteht somit die Gefahr, die Anforderungen stets so zu manipulieren, dass die „Lieblings-Technologie“ doch die Optimallösung darstellt. In diesem Falle bräuchte man das ganze Vorgehensmodell nicht, denn die Mindestanforderungen haben ja keinen Selbstzweck.

Das gesamte in diesem Kapitel präsentierte Vorgehen wird grafisch in Abbildung 52 und Abbildung 53 gezeigt. Diese Grafiken zeigen grundsätzlich dasselbe Modell, allerdings liegen verschiedene Blickwinkel zugrunde:

Abbildung 52 ist am Inhalt der einzelnen Schritte und an deren Abhängigkeiten orientiert. Es wird gezeigt, welche Ergebnisse in die Spalten und Zeilen der einzelnen Folgetabellen einfließen.

Das Ziel von Abbildung 53 ist dagegen die Verdeutlichung des Ablaufs. Diese Grafik ist in drei Kategorien unterteilt, die einen unterschiedlich starken Input des Anwenders benötigen. Durch die Färbung wird verdeutlicht, welche Schritte zur Soll- (weiß) und welche zur Ist-Perspektive (grau) gehören. Die Reihenfolge der Schritte drückt sich durch ihre vertikale Anordnung, ihre Nummerierung und die Pfeile aus.

Auch wenn diese beiden Grafiken zur Vermeidung von Redundanz nicht zwischendurch erneut abgedruckt sind, können und sollen sie zur Orientierung in den einzelnen Abschnitten immer wieder herangezogen werden. Die einzelnen Schritte sind jeweils mit der zugehörigen Kapitelnummer versehen, um dies zu erleichtern.

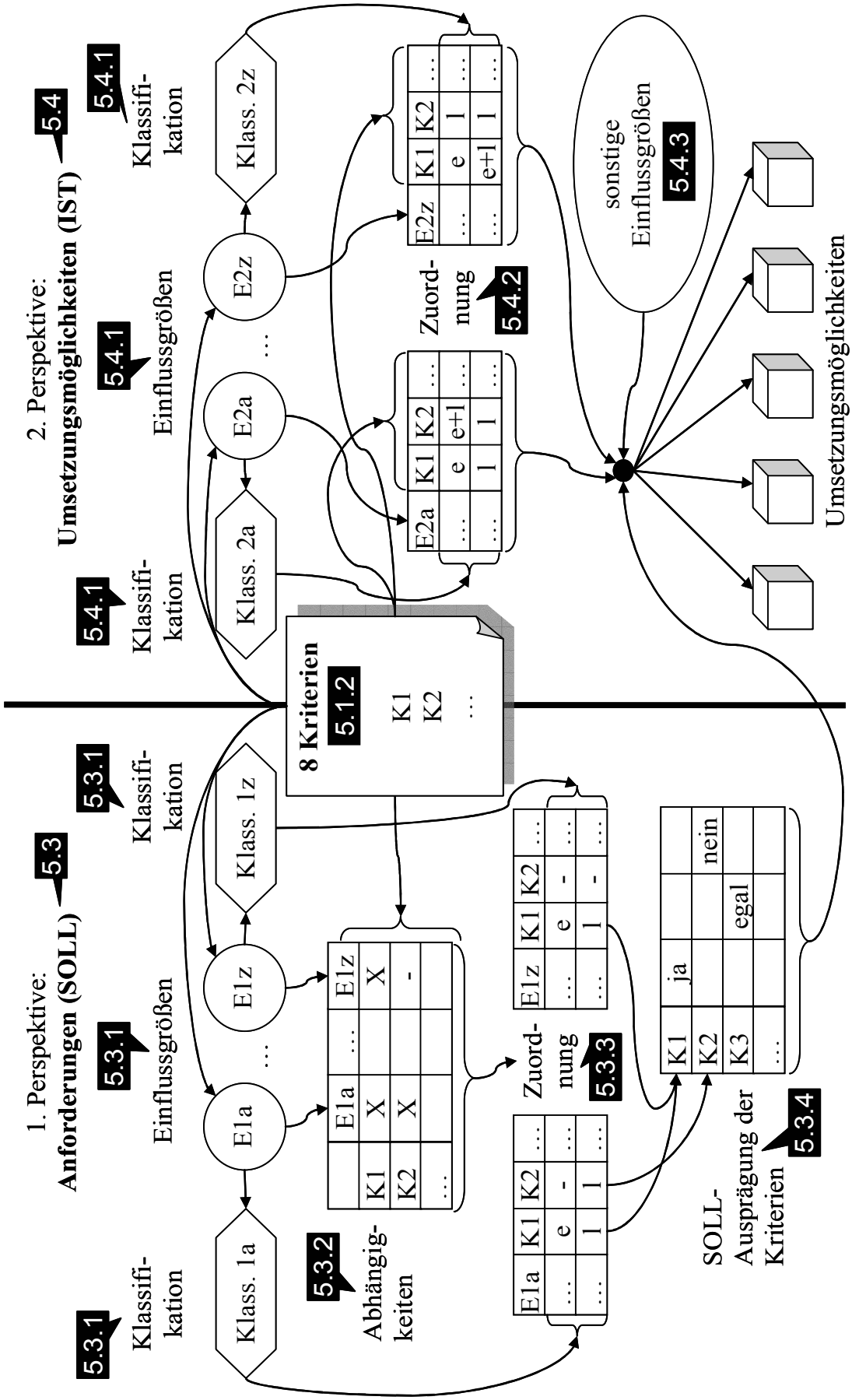

Abbildung 52: Das Vorgehensmodell (in Anlehnung an [FT06])

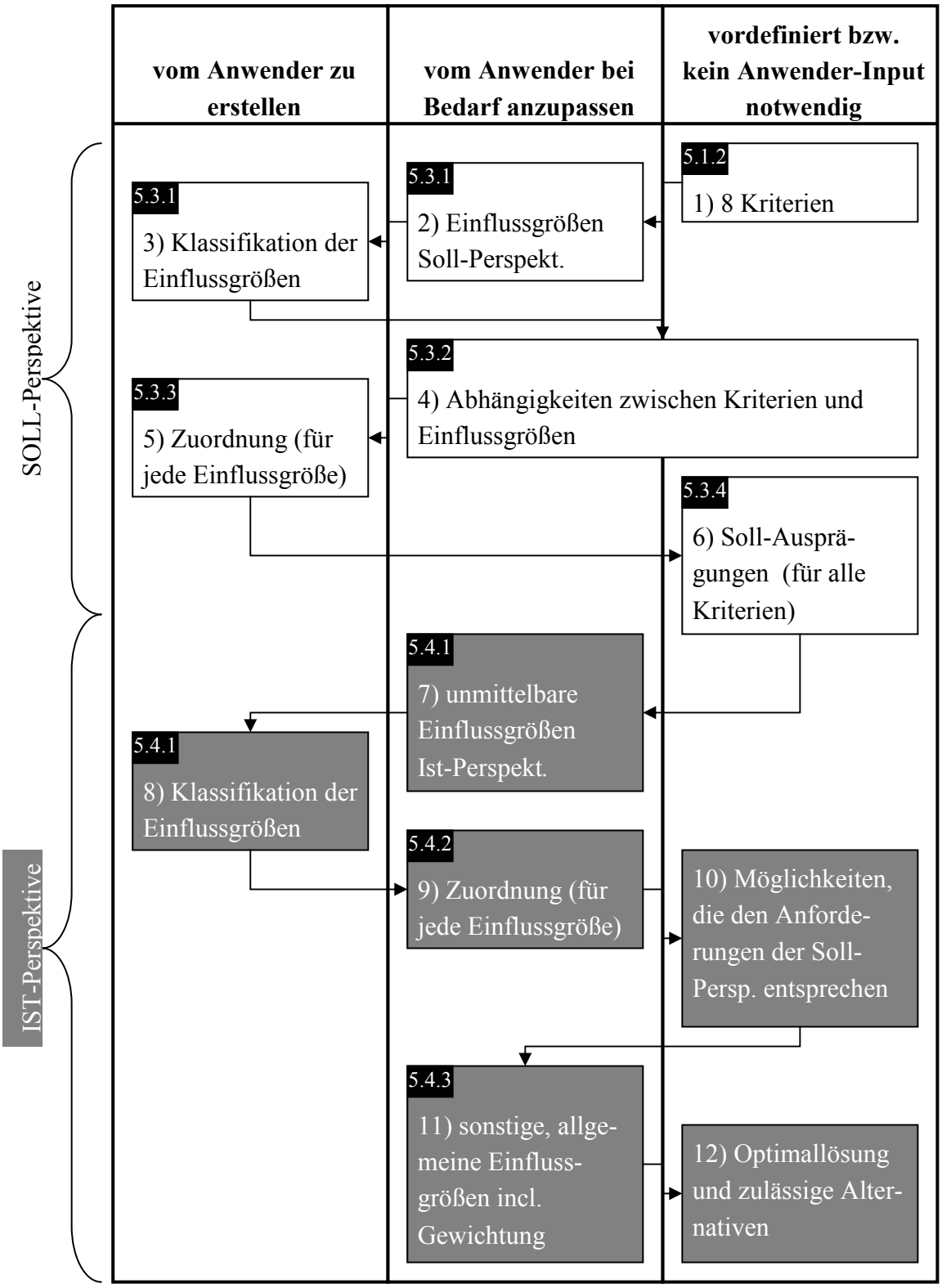

Abbildung 53: Das Vorgehensmodell ablauforientiert

5.3 Die erste Perspektive: orientiert an den Anforderungen (Soll)

Wie soeben (5.2) beschrieben, ist es das Ziel dieser Perspektive, Technologie-unabhängige Mindestanforderungen für die Kommunikation zwischen verschiedenen Software-Komponenten in Form von bestimmten Ausprägungen der 8 Kriterien zu definieren.

Der erste Schritt dorthin ist, die Einflussgrößen zu identifizieren, die grundsätzlich diese Kriterien beeinflussen. Zudem muss jeweils eine geeignete Möglichkeit der Klassifikation gefunden werden. Die ist Gegenstand von 5.3.1.

Anschließend ist zu untersuchen, von welcher oder welchen dieser Einflussgrößen die 8 Kriterien jeweils abhängig sind (5.3.2), bevor eine Soll-Zuordnung für jede Einflussgröße festgelegt wird (5.3.3).

Schließlich werden diese einzelnen Zuordnungen zu einer aggregierten Tabelle zusammengefasst, die das Ergebnis dieser Perspektive bildet. Dies wird in 5.3.4 demonstriert.

Als Ergänzung und Hilfestellung wird im Anschluss in 5.3.5 die Verwendung verschiedener Komponenten-Klassen diskutiert.

5.3.1 Die Einflussgrößen und ihre Klassifikation

Im Folgenden werden die bislang identifizierten Einflussgrößen erläutert, die grundsätzlich die optimale Ausprägung einzelner Kriterien beeinflussen können. Darüber hinaus werden Vorschläge für eine Klassifikation präsentiert. Eine detaillierte Betrachtung hinsichtlich der Vor- und Nachteile der verschiedenen Ausprägungen dieser Kriterien und der sich daraus ergebenden Konsequenzen folgt im weiteren Verlauf dieses Kapitels.

Die Formulierung „bislang“ soll verdeutlichen, dass diese Einflussgrößen bezüglich der betrachteten acht Kriterien imstande sind, alle diese Vor- und Nachteile abzubilden. Falls weitere Kriterien hinzugefügt oder weitere Vor- oder Nachteile offenkundig werden, die mit Maßnahmen erreicht bzw. vermieden werden können, die sich über keine der betrachteten Einflussgrößen steuern lassen, so ist es für den Anwender nicht nur möglich, sondern sogar zwingend notwendig, eine oder mehrere zusätzliche Einflussgrößen zu ergänzen.

Bei der Entwicklung dieses Vorgehensmodells war die Reihenfolge tatsächlich so, dass zunächst die Vor- und Nachteile eines jeden Kriteriums betrachtet und bei Bedarf eine neue Einflussgröße definiert wurde. In der fertigen Arbeit ist es jedoch sinnvoller, zunächst alle Einflussgrößen auf einmal zu benennen und vor allem zu beschreiben, um dem Leser ein Gefühl dafür zu geben, welche Größen hierbei betrachtet werden, ehe diese konkreten Kriterien zugeordnet werden.

Bei der Anwendung dieser Einflussgrößen in der Praxis sollte folgendes unbedingt im Hinterkopf behalten werden: Genauso wie nicht *alle* Kriterien – wie bereits erläutert – zur Beschrei-

bung der Anforderungen herangezogen werden müssen, so ist es auch nicht zwingend, *alle* Einflussgrößen zu berücksichtigen. Aber auch hier gilt der Grundsatz, dass mit steigender Lockerung der Restriktionen die Menge der geeigneten Lösungen größer wird und man sich am Ende unter Umständen wundert, dass die Optimallösung z.B. mit einer Technologie verbunden ist, die gegenwärtig nicht in aller Munde ist.

5.3.1.1 Fachliche Zusammengehörigkeit (Domänen)

In 4.4.5 wurde beschrieben, welchen Zweck die Definition von fachlichen Domänen und die Einordnung von Services im Rahmen einer SOA in diese verfolgt. Folglich ist – wenn ein solches Domänenmodell verwendet wird – die Unterscheidung zwischen einer Domänen-internen und einer Domänen-übergreifenden Kommunikation von entscheidender Bedeutung für die Wahl sowohl der Methode als auch der Technologie der Integration. Denn auch wenn das Domänenkonzept erst im SOA-Kontext an Bedeutung gewonnen hat, so hat die Kommunikation innerhalb einer Domäne ausdrücklich nicht zwingend Service-orientiert zu sein.

Die Klassifikation gestaltet sich bei dieser Einflussgröße sehr einfach: Die kommunizierenden Komponenten befinden sich entweder in derselben Domäne oder in verschiedenen Domänen.

5.3.1.2 Einsatzbereich (organisatorische Zusammengehörigkeit)

Während bei der Frage der Domänen-Einteilung (5.3.1.1) eine fachliche Verwandtschaft verschiedener Komponenten im Vordergrund steht, ist beim Einsatzbereich der Fokus auf die Organisationsstruktur und die Frage gerichtet, wie weit der Kreis der möglichen Nutzer einer Komponente ausgeweitet werden sollte.

Es können grundsätzlich zwei Arten des Einsatzbereichs unterschieden werden [Tie06]:

- der fachlich mögliche Einsatzbereich und
- der tatsächliche Einsatzbereich

Ein Unterschied dieser beiden Mengen würde bedeuten, dass eine Komponente in fachlich sinnvoller Art und Weise auf die von einer anderen Komponente bereitgestellte Funktionalität zugreifen könnte, dies aber aufgrund von Restriktionen nicht darf. Mögliche Gründe für eine solche Einschränkung des Einsatzbereichs sind folgende:

- Sicherheit / Geheimhaltung (Dies spielt insbesondere dann eine Rolle, wenn die Verwendung außerhalb der Grenzen der eigenen Organisation möglich wäre, im Extremfall weltweit.)
- Performance (wenn die Überschreitung einer bestimmten Nutzer-Anzahl aufgrund von begrenzten Hardware- und/oder Basis-Software-Kapazitäten eine nicht mehr akzeptable Leistung für alle Teilnehmer zur Folge hätte)
- Sicherung eines angemessenen Supports (wenn z.B. die Aufrechterhaltung eines Supports rund um die Uhr bei einer Erweiterung des Nutzerkreises aus technischen, organisatorischen oder finanziellen Gründen nicht mehr möglich wäre)

Theoretisch kann der tatsächliche Einsatzbereich niemals größer sein als der fachlich mögliche. Sollte sich beispielsweise bei einer Anwendung, die eigentlich nur für den Gebrauch innerhalb eines Unternehmens entwickelt worden ist, nachträglich eine externe Verwendungsmöglichkeit herausstellen, so würde hierdurch zunächst nur der fachlich mögliche Einsatzbereich ausgedehnt. Im Falle einer Entscheidung, diese externe Verwendung zuzulassen, wären der tatsächliche und der fachlich mögliche Einsatzbereich wieder gleich.

In der Praxis kann es dennoch eine Möglichkeit geben, bei der der tatsächliche Einsatzbereich größer ist als der fachlich mögliche, nämlich die nicht sinnvolle Verwendung einer Komponente. Mit anderen Worten, diese ist nicht nur ursprünglich für einen anderen Einsatzbereich entwickelt worden, sondern sie kann die vom betrachteten Aufrufer geforderte Funktionalität – zumindest in semantischer Hinsicht – gar nicht liefern. Eine derartige Nutzung kann z.B. mit Hilfe von Zugriffsrechten verhindert werden.

Die Vorgabe einer Klassifikation des Einsatzbereichs ist auf theoretischer Ebene nur äußerst begrenzt möglich, denn schließlich müssen hier die individuellen Gegebenheiten der anwendenden Organisation berücksichtigt werden.

Für den Vertriebsbereich eines Automobilkonzerns könnte beispielsweise folgende Unterteilung gewählt werden:

- OEM (Hersteller)
- Großhändler / Importeur
- Autohaus

Für die Strukturierung der Kommunikation innerhalb desselben Unternehmens ist diese Aufteilung hingegen wenig zielführend: Autohäuser und Großhändler müssen nicht betrachtet werden, die Kategorie „OEM“ ist dagegen viel zu undifferenziert. Hier wäre das sogenannte Zonen- und Schichten-Modell [TWR04], das in Abbildung 54 zu sehen ist, eine Lösung.

Dieses Modell ist ursprünglich für die Konzernbeschaffung der Volkswagen AG entwickelt worden, lässt sich aber problemlos auf andere Einsatzgebiete übertragen. Es stellt ein Raster aus Zonen und Schichten zur Verfügung. Die Zonen beschreiben dabei eine logische Gruppierung bestimmter Aufgabengebiete zur Informationsverarbeitung, die Schichten definieren zusätzlich eine logische Reihenfolge der Abhängigkeiten. Jede Komponente kann so dem Knotenpunkt einer Zone und einer Schicht zugeordnet werden.

Die Schichten dieses Modells (in der Grafik horizontal) sind eine Möglichkeit zur Klassifikation des Einsatzbereichs: Von der Applikations-Schicht bis zur Basis-Services-Schicht wird der Einsatzbereich der dort eingeordneten Komponenten sukzessive von der Verwendung ausschließlich in einer einzigen Applikation bis zum potenziell weltweiten Zugriff erweitert.

Für detailliertere Informationen bezüglich des Zonen- und Schichtenmodells sowie Beispielen zu seiner Anwendung wird auf [TWR04] verwiesen.

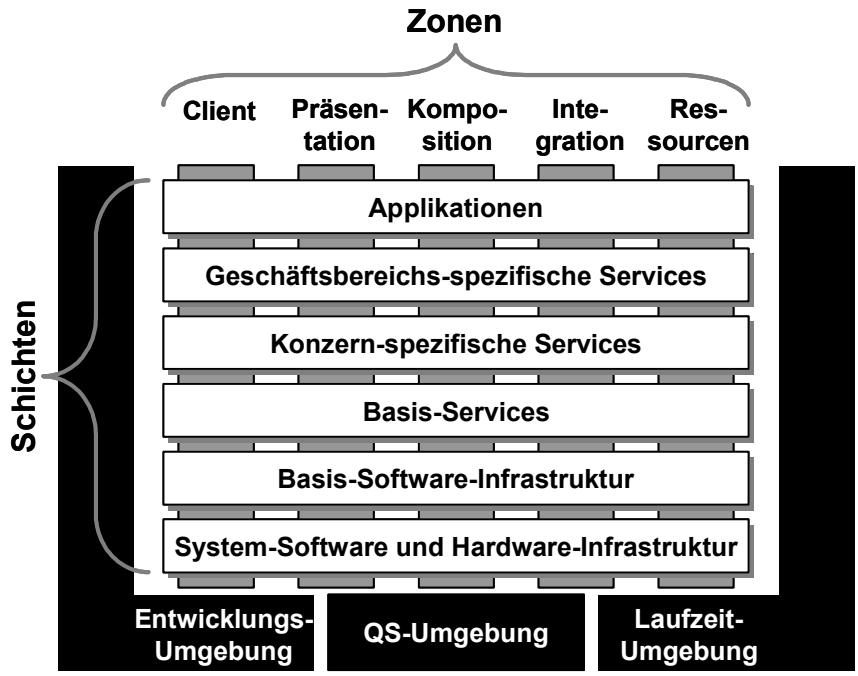

Abbildung 54: Das Zonen- und Schichten-Modell [TWR04]

5.3.1.3 Bearbeitungsdauer

Die Bearbeitungsdauer dient als Maß dafür, wie viel Zeit eine Komponente benötigt, um eine Anfrage zu bearbeiten. Welche Schritte dazu notwendig sind und welche anderen Komponenten hinzugezogen werden, spielt hierbei keine Rolle.

Eine Klassifikation kann hierbei nur auf Basis der Zeit vorgenommen werden, d.h. in Gestalt von einem oder mehreren Grenzwerten, so dass verschiedene Zeitbereiche unterschieden werden können.

5.3.1.4 Komplexität

Bei der Komplexität gelten im Vergleich zur Bearbeitungsdauer die umgekehrten Bedingungen: Entscheidend ist die Frage, wie viele Komponenten an der Bearbeitung einer Anfrage beteiligt sind, d.h. wie weit sich die Anfrage verzweigt. Die Zeit, die hierfür benötigt wird, ist zweitrangig.

Eine Klassifikation kann sich nur an der absoluten Anzahl der beteiligten Komponenten orientieren und hierbei – ähnlich wie bei der Bearbeitungsdauer - verschiedene Bereiche definieren. Eine Grenze ist dabei sinnvollerweise zwischen einer und zwei Komponenten zu setzen.

5.3.1.5 Verfügbarkeit (Erreichbarkeit)

Diese Einflussgröße dient als Maß dafür, ob eine Komponente stets „ansprechbar“ ist oder ob bzw. wie häufig es vorkommen kann, dass keine Verbindung dorthin besteht. Die Ursachen für eine eingeschränkte Erreichbarkeit können vielfältiger Natur sein: Denkbar wären Netzwerk-Ausfälle (wie möglicherweise bei mobiler Kommunikation), geplante Stillstandszeiten (wie z.B. nachts) oder auch temporäre Überlastung (bei Lastspitzen).

Je nachdem, welche dieser Größen entscheidend ist, kann sich eine Klassifikation an der prozentualen Verfügbarkeit, an der Anzahl der Verbindungsunterbrechungen pro Zeiteinheit oder am Lastprofil orientieren.

Es ist an dieser Stelle ausdrücklich die Abgrenzung der Verfügbarkeit von der Bearbeitungsdauer zu betonen: Die Folge eines Anstiegs der Bearbeitungsdauer wie auch einer geringeren Verfügbarkeit ist eine Verlängerung der durchschnittlichen Antwortzeit einer Anfrage. Jedoch ist die Ursache im letzteren Falle darin zu sehen, dass die Bearbeitung erst verspätet beginnen kann, sei es aufgrund einer Nichterreichbarkeit oder Überlastung. Im Falle einer langen Bearbeitungsdauer ist die Zielkomponente dennoch bereit, Anfragen anzunehmen; ist sie nicht verfügbar, so bedeutet dies, dass die aufrufende Komponente entweder auf die Wiederereichbarkeit warten oder die Anfrage in einem Zwischenmedium ablegen muss. Letzteres ist genau die Entscheidung, die im Rahmen des Kriteriums „indirekte physikalische Kopplung“ getroffen werden muss.

5.3.1.6 Änderbarkeit

Die Änderbarkeit gibt zunächst einmal Auskunft darüber, ob es überhaupt möglich ist, Änderungen am Quellcode einer Komponente vorzunehmen oder ob beispielsweise eine so alte Programmiersprache verwendet wurde, dass kein qualifiziertes Personal mehr vorhanden ist. Hierbei ist eine Klassifikation recht einfach, da eine Ja/Nein-Entscheidung vorliegt.

Des Weiteren muss mit dieser Einflussgröße aber auch ausgedrückt werden, wie schnell mögliche Änderungen ausgeführt werden können. Dieser Aspekt ist schon weitaus schwieriger zu klassifizieren. Denkbar wäre eine Maßzahl wie der durchschnittliche oder der maximale Zeitaufwand für Änderungen, allerdings müsste hierfür zuvor noch ein Raster für Änderungen mit verschiedenen Aufwänden erstellt werden.

5.3.1.7 Anzahl der Parameter

Diese Einflussgröße kann als eine Art Granularität der Schnittstelle verstanden werden. Unabhängig davon, wie umfangreich oder weit verzweigt eine Komponente ist, wird hierdurch die Anzahl der Variablen gemessen, die sie zur Erfüllung ihrer Funktionalität benötigt.

Im Rahmen der Klassifikation muss lediglich entschieden werden, in wie viele und welche Bereiche die Zahlen-Skala sinnvollerweise unterteilt werde sollte.

5.3.1.8 Syntaktische Änderungshäufigkeit

Die syntaktische Änderungshäufigkeit bezieht sich direkt auf die Parameter einer Schnittstelle. Eine hohe syntaktische Änderungshäufigkeit bedeutet, dass zum Abruf der Funktionalität der betrachteten Komponente häufig andere Parameter als zuvor notwendig sind. Die Ermittlung oder Berechnung einzelner Parameter kann aber trotzdem unverändert bleiben.

Die Klassifikation erfolgt sinnvollerweise über die Anzahl der Änderungen pro Zeiteinheit. Wie viele Kategorien dabei unterschieden und wie groß die Zeiteinheiten gewählt werden, liegt im Ermessen des Anwenders.

5.3.1.9 Semantische Änderungshäufigkeit

Die semantische Änderungshäufigkeit hat nur indirekt mit den Parametern einer Schnittstelle zu tun. Sie ist vielmehr ein Ausdruck dafür, wie häufig sich die Berechnungslogik der Parameter ändert. Eine hohe semantische Änderungshäufigkeit ist folglich ein Ausdruck dafür, dass beispielsweise aufgrund von sich ändernden Gesetzen oder unternehmensinternen Vorschriften einzelne Parameter anders als zuvor berechnet werden müssen. Die Schnittstelle selbst, d.h. die zum Aufruf notwendigen Parameter an sich, muss sich dadurch nicht verändern.

Die Klassifikation ist analog zur syntaktischen Änderungshäufigkeit zu gestalten.

5.3.2 Abhängigkeiten der Einflussgrößen von den Kriterien

Nachdem in 5.3.1 die Einflussgrößen präsentiert wurden, die grundsätzlich die Ausprägung einzelner Kriterien beeinflussen können, wird nun deren Zuordnung zu den Kriterien vorgenommen. Das bedeutet, dass für jedes Kriterium bestimmt werden muss, von welcher oder welchen der Einflussgrößen es abhängig ist.

Zu diesem Zweck werden in den nun folgenden Tabellen zunächst die Vorteile der eng gekoppelten sowie der lose gekoppelten Variante gesammelt. Diese werden anschließend verdichtet, indem aus den Vorteilen potenzielle Einsatzgebiete abgeleitet werden. Diese Informationen werden anschließend auf die in 5.3.1 vorgestellten Einflussgrößen übertragen.

Die Formulierung „potenziell“ soll verdeutlichen, dass hierdurch (noch) kein Zwang formuliert wird, schon gar nicht bei der engen Kopplung. Das Ziel ist an dieser Stelle lediglich, jeweils diejenigen Einflussgrößen zu identifizieren, bei denen eine Abhängigkeit zum betrachteten Kriterium vorliegt. Die Festlegung von konkreten Restriktionen hinsichtlich der Verwendung der losen oder engen Kopplung erfolgt erst im nächsten Schritt (5.3.3) auf Basis der zu den einzelnen Einflussgrößen definierten Klassifikationsvorgaben (vgl. 5.3.1).

	KOMMUNIKATION	
	synchron	**asynchron**
Vorteile	• leichtere Zuordnung der Antwort bei der aufrufenden Komponente (weniger Kontext-Informationen müssen mitgeschickt werden) • keine Gefahr einer unbemerkten Zeitverzögerung • Geschäftslogik ist oftmals nicht auf Asynchronität ausgelegt	• bessere Eignung zum Initiieren lang laufender Prozesse (kein Warten auf Rückgabewerte) • zielgerichtetere Fehlerbehandlung möglich (nicht nur Timeouts auf oberster Ebene) • kein Timeout bei Verzögerung der Antwort einer Komponente aufgrund von temporärer Nichterreichbarkeit
potenzielle Einsatzgebiete	Aufruf von Komponenten mit • kurzer Ausführungsdauer	Aufruf von Komponenten mit • langer oder nicht vorhersehbarer Ausführungsdauer • eingeschränkter Verfügbarkeit
Also: Abhängigkeit von folgenden Einflussgrößen: • Bearbeitungsdauer • Verfügbarkeit		

Tabelle 8: Zuordnung der Einflussgrößen zu den Kriterien: Kommunikation

Die Abhängigkeit von der Einflussgröße „Verfügbarkeit“ ist nur in dem Sinne zu verstehen, der unter den Vorteilen aufgeführt ist, nämlich einer Verzögerung der *Antwort* einer fremden Komponente. Mit anderen Worten, die Anfrage an sich wurde erfolgreich übermittelt, aber während deren Bearbeitung ist die Verbindung unterbrochen worden. Falls aufgrund einer fehlenden Verfügbarkeit die Anfrage selbst nicht übermittelt werden kann, so bringt eine asynchrone Kommunikation alleine noch keine Abhilfe, sondern es ist die Kombination mit indirekter physikalischer Kopplung notwendig, um die Anfrage bis zur Wiedererreichbarkeit der Zielkomponente in einem Zwischenmedium ablegen zu können.

	NACHRICHTEN-STIL	
	RPC-Stil	**Dokument-Stil**
Vorteile	• weniger Netzwerk-Belastung, o da jeweils nur die unbedingt benötigten Daten übertragen werden (und nicht ein ganzes Geschäftsobjekt) o da evtl. fehlende Daten zum vollständigen Füllen des von der aufzurufenden Komponente geforderten Dokuments erst besorgt werden müssen (auch wenn diese Daten für die benutzte Funktionalität gar nicht benötigt werden) • Performance-Vorteil, da nicht auf das Eintreffen von fehlenden Daten zum vollständigen Füllen des Dokuments gewartet werden muss	• eine statt mehrerer Anfragen, d.h. kein Kontext bei der aufgerufenen Komponente notwendig • Robustheit gegen kleinere Schnittstellen-Änderungen (d.h. wenn die notwendigen Parameter auf Seiten der aufrufenden Komponente sowieso vorliegen und mit dem Geschäftsobjekt übermittelt werden); Lebenszyklus-Management auf Dokument- und nicht auf Schnittstellen-Ebene • Da stets ein vollständiges Geschäftsobjekt vorhanden ist, wird die Möglichkeit der späteren Aufteilung der Arbeit erleichtert. • weniger Netzwerk-Verkehr durch Vermeidung von unnötigem Request/ Reply, o da die nächste Komponente einer Aufruf-Kette direkt kontaktiert werden kann (ohne den Weg über die „Zentrale“) o da Informationen z.B. über geroutete Nachrichten-Wege oder Bindung im Dokument mitgeschickt werden können und somit der Aufruf dafür zuständiger Komponenten nicht notwendig ist • Möglichkeit der Orientierung an Industrie-Standards • Möglichkeit der Validierung im Falle der Nutzung von XML

potenzielle Einsatzgebiete	• Nutzung von Funktionalität, die nur wenige Parameter benötigt • weitgehend monolithische, d.h. nicht weiter verzweigte Komponenten-Strukturen • bekannte Komponenten • Komponenten, deren Parameter sich selten ändern	• Nutzung von Funktionalität, die viele Parameter benötigt • stark verzweigte (kaskadierte) Komponenten-Strukturen • unbekannte Komponenten • Komponenten, deren Parameter sich häufig ändern
Also: Abhängigkeit von folgenden Einflussgrößen: • Komplexität • Einsatzbereich • Anzahl der Parameter • Syntaktische Änderungshäufigkeit		

Tabelle 9 Zuordnung der Einflussgrößen zu den Kriterien: Nachrichten-Stil

An dieser Stelle wird bereits deutlich, dass manchmal verschiedene Argumente in Abhängigkeit des Anwendungsfalls gegeneinander abzuwägen sind: Eine unbekannte Komponente mit wenigen Parametern vereinigt die Vorteile von RPC- und Dokument-Stil. Hier ist im Einzelfall zu entscheiden, welcher Aspekt der wichtigere ist, d.h. ob die Erreichung einer höheren Robustheit gegen Schnittstellen-Änderungen die Beschaffung aller Parameter eines Geschäftsobjekts rechtfertigt.

Ebenso kann nicht pauschal festgelegt werden, welcher Stil weniger Netzwerk-Verkehr hervorruft. Wie bei den Vorteilen ersichtlich ist, kann die Netzwerk-Belastung durch Vermeidung der Übertragung von gar nicht benötigten Daten und deren eventueller Beschaffung verringert werden, der Dokument-Stil hingegen erzeugt dadurch einen geringeren Netzwerk-Verkehr, dass die Anzahl der ausgetauschten Nachrichten verringert werden kann.

	BINDUNG	
	zur Entwicklungs- oder Konfigurationszeit	**zur Laufzeit**
Vorteile	• einfachere und schnellere Implementierung • weniger Netzwerk-Verkehr durch Einsparen der Anfragen beim Naming Service • Performance-Gewinn, da nicht auf die Antwort des Naming Service gewartet werden muss • keine Abhängigkeit von der Erreichbarkeit des Naming Service	• keine hart codierte Bindung an Daten(quellen) oder Implementierungen • Möglichkeit der „Umleitung“ einer Anfrage bei Nichterreichbarkeit einer Komponente • leichtere Austauschbarkeit von Komponenten • Sicherstellung einer aktuellen Datenbasis • Möglichkeit der Auswahl zwischen fachlich gleichwertigen Komponenten, die unterschiedliche Leistung bieten, zur Laufzeit
potenzielle Einsatzgebiete	• sich selten ändernde Datenstrukturen oder Implementierungen • leichte Änderbarkeit der aufrufenden Komponente • ausreichende Verfügbarkeit der aufgerufenen Komponente • Komponenten, auf die organisatorisch unmittelbarer Einfluss besteht (d.h. Änderungen können mit angemessenem Aufwand umgesetzt werden)	• sich häufig ändernde Datenstrukturen oder Implementierungen • eingeschränkte Verfügbarkeit der aufgerufenen Komponente • schwere Änderbarkeit der aufrufenden Komponente • Komponenten, auf die organisatorisch kein unmittelbarer Einfluss besteht • bei Komponenten, deren Funktion an vielen Stellen benötigt wird und von denen mehrere Varianten (im Wesentlichen außerhalb der eigenen Organisation) existieren
Also: Abhängigkeit von folgenden Einflussgrößen: • Verfügbarkeit • Änderbarkeit • Einsatzbereich		

Tabelle 10: Zuordnung der Einflussgrößen zu den Kriterien: Bindung

Die genannten Nachteile einer späten Bindung bzgl. Performance und Netzwerk-Verkehr können dann zumindest teilweise vermieden werden, wenn der Dokument-Stil (vgl. 5.1.2.2) verwendet wird und die Information über die aufzurufenden Komponente(n) nicht von einem Naming Service über das Netzwerk bezogen werden müssen, sondern im Dokument mitgeschickt werden.

Die Einsetzbarkeit dieser Variante ist davon abhängig, wie lange ein auf diesem Dokument basierender Prozess dauert (d.h. wie lange das Dokument bereits unterwegs ist, bevor es die betrachtete Komponente erreicht) und wie schnell die Änderungen bzgl. der Bindung umgesetzt werden müssen.

	NACHRICHTEN-WEGE	
	bekannt, hart codiert	**geroutet**
Vorteile	• einfache und schnelle Implementierung • weniger Netzwerkbelastung (Informationen über die nächste Komponente liegen bereits in hart codierter Form vor.) • Performancegewinn (Auf die Information über die nächste aufzurufende Komponente muss nicht gewartet werden.) • keine Abhängigkeit von der Erreichbarkeit des Naming Service	Bei einer Änderung des Nachrichtenwegs muss die sendende Komponente nicht verändert werden (die empfangende sowieso nicht). Dies ist insbesondere bei längeren Aufrufketten (wie z.B. Geschäftsprozessen) von großem Vorteil.
potenzielle Einsatzgebiete	• sich selten ändernde Kommunikationswege • leicht änderbare Komponenten • Komponenten, auf die organisatorisch unmittelbarer Einfluss besteht	• sich häufig ändernde Kommunikationswege • schwer änderbare Komponenten (z.B. aufgrund veralteter Programmiersprachen) • Komponenten, auf die organisatorisch kein unmittelbarer Einfluss besteht
Also: Abhängigkeit von folgenden Einflussgrößen: • Änderbarkeit • Einsatzbereich		

Tabelle 11: Zuordnung der Einflussgrößen zu den Kriterien: Nachrichten-Wege

Wenn die Information der nächsten Komponente nicht jedes Mal direkt von einer zentralen Instanz erfragt werden muss, sondern bei Verwendung des Dokument-Stils (vgl. 5.1.2.2) im Dokument mitgeschickt wird, können die o.g. Nachteile bzgl. Netzwerkbelastung und Performance – analog zum Fall der dynamischen Bindung – vermieden werden.

	DATENTYPEN	
	abhängig von den Anwendungen	**unabhängig**
Vorteile	• kein Arbeitsaufwand für die Definition eines Meta-Datenformats • teilweise Performance-Vorteile, da weniger Daten-Konvertierungen notwendig sind	• gemeinsames Verständnis für die Semantik bzw. den geforderten Inhalt von Parametern • Möglichkeit des automatischen Mappings der eigenen Variablen auf die Parameter einer bislang unbekannten Schnittstelle • leichtere Anpassungen an Änderungen von Schnittstellen und Prozessen, da keine Datenformat-bedingten Konflikte auftreten können
potenzielle Einsatzgebiete	• bekannte Komponenten	• unbekannte Komponenten • schwer änderbare Komponenten
Also: Abhängigkeit von folgenden Einflussgrößen: • Einsatzbereich • Änderbarkeit		

Tabelle 12: Zuordnung der Einflussgrößen zu den Kriterien: Datentypen

	SYNTAX-DEFINITION	
	durch direkte Abstimmung	**über Schemata**
Vorteile	• weniger Arbeit, da keine Standardisierung und Abstimmung der Syntax notwendig ist	• Die Syntax einer Schnittstelle kann automatisiert eingelesen werden. • Syntax-Änderungen müssen nicht explizit an jeden Nutzer kommuniziert werden.
potenzielle Einsatzgebiete	• bekannte Komponenten • Komponenten mit wenigen Nutzern	• unbekannte Komponenten • Komponenten mit vielen Nutzern
Also: Abhängigkeit von folgenden Einflussgrößen: • Einsatzbereich		

Tabelle 13: Zuordnung der Einflussgrößen zu den Kriterien: Syntax-Definition

	TRANSFORMATION	
	durch Codierung in den Anwendungen	**durch externe Transformatoren**
Vorteile	• einfachere und schnellere Implementierung bei unkomplizierter Berechnungslogik • Performance-Gewinn, da nicht auf das Ergebnis der Transformation gewartet werden muss • weniger Netzwerk-Verkehr, da kein Aufrufen eines externen Transformators notwendig ist • keine Abhängigkeit von der Erreichbarkeit oder Leistung anderer Komponenten	• Delegation: Weder die Quell- noch die Zielkomponente muss sich kümmern um o die Implementierung fachfremder Logik sowie o die Sicherstellung einer aktuellen Datenbasis hierfür. • Wiederverwendung der Transformationslogik • Bei einer Änderung der Transformation muss nur der Transformations-Dienst, nicht aber die Quell- oder Zielanwendung verändert werden.
potenzielle Einsatzgebiete	• Aufruf von Komponenten mit Parametern, o deren Berechnungslogik sich selten ändert o die relativ einfach zu berechnen sind o die im Verhältnis zur aufrufenden Komponente nicht fachfremd sind • leicht änderbare aufrufende Komponenten	• Aufruf von Komponenten mit Parametern, o deren Berechnungslogik sich häufig ändert o die sehr kompliziert zu berechnen sind o die im Verhältnis zur aufrufenden Komponente sehr fachfremd sind • schwer änderbare aufrufende Komponenten
Also: Abhängigkeit von folgenden Einflussgrößen: • Semantische Änderungshäufigkeit • Einsatzbereich • Änderbarkeit		

Tabelle 14: Zuordnung der Einflussgrößen zu den Kriterien: Transformation

Eine Bemerkung hierzu: Die Aktualität der Daten ist kein Argument für eine externe Transformation. Auch bei einer internen Transformation können externe (und aktuelle) Datenquellen genutzt werden; nur die Berechnungslogik befindet sich per definitionem innerhalb der aufrufenden Komponente.

	PHYSIKALISCHE KOPPLUNG	
	direkt	**unter Verwendung von Middleware**
Vorteile	• einfachere und günstigere Implementierung	• Es muss keine aufgerufene Komponente im Detail bekannt sein, sondern nur z.B. eine Message Queue. • Ermöglichung einer zeitlichen Entkopplung von Sender und Empfänger (der Sender muss nicht „online“ sein) • bessere Lastverteilung durch Ablegen von eingehenden Nachrichten in einer „Warteschlange“ • weniger Fehlerbehandlung im Falle einer temporären Nichterreichbarkeit des Empfängers notwendig • Verlagerung von Aufgaben wie Sicherheit, Transformation, Logging, Monitoring in die Middleware
potenzielle Einsatzgebiete	• bekannte Komponenten • Komponenten mit ausreichender Verfügbarkeit • Komponenten mit relativ homogener Auslastung	• unbekannte Komponenten • Komponenten mit eingeschränkter Verfügbarkeit • Komponenten mit hohen Lastspitzen
Also: Abhängigkeit von folgenden Einflussgrößen: • Einsatzbereich • Verfügbarkeit		

Tabelle 15: Zuordnung der Einflussgrößen zu den Kriterien: Physikalische Kopplung

Eine gesonderte Rolle spielt die erste Einflussgröße (die fachliche Zusammengehörigkeit; vgl. 5.3.1.1), denn die hiervon abhängigen Einflussgrößen lassen sich nicht direkt auf Basis der Vor- und Nachteile der Kriterien bestimmen. Daher muss die Zuordnung auf umgekehrtem Wege erfolgen, d.h. es ist für jedes Kriterium die Frage zu beantworten, ob das Verlassen der eigenen Domäne den Wechsel von enger zu loser Kopplung zur Folge haben muss[29]. Es ist

[29] Es wird an dieser Stelle vorausgesetzt, dass innerhalb einer Domäne die eng gekoppelte Variante zumindest erlaubt ist, denn bei der einzigen Alternative, der Domänen-übergreifenden Kommunikation, kann die Restrikti-

dabei von entscheidender Bedeutung, dass die Formulierung „muss“ ernst genommen wird: Es ist nicht entscheidend, ob eine losere Kopplung strategisch brauchbar klingt, sondern ob die engere Kopplung mit gravierenden Nachteilen verbunden ist.

Auf dieser Basis wurden zwei Kriterien identifiziert, von denen die fachliche Zusammengehörigkeit *nicht* abhängig ist: die Bindung und die physikalische Kopplung. Im ersteren Falle lässt sich dies damit begründen, dass das Kriterium der Nachrichten-Wege (also die Frage nach der in fachlicher Hinsicht nächsten Komponente) gegebenenfalls domänenübergreifend erfolgen muss, während sich verschiedene Instanzen dieser Komponente grundsätzlich in derselben Domäne befinden und die Bindung folglich innerhalb von deren Grenzen entschieden werden kann. Die physikalische Kopplung ist zu technisch orientiert, als dass die Domänenzugehörigkeit darauf einen Einfluss haben könnte.

Abschließend werden in Tabelle 16 die soeben ermittelten Abhängigkeiten nochmals zusammengefasst, wodurch auch eine Betrachtung aus der Perspektive der Einflussgrößen möglich wird.

Kriterium / Einflussgröße	Kommunikation	Nachrichten-Stil	Bindung	Nachrichten-Wege	Datentypen	Syntax-Definition	Transformation	Physikalische Kopplung
fachliche Zusammengehörigkeit	X	X		X	X	X	X	
Einsatzbereich		X	X	X	X	X	X	X
Bearbeitungsdauer	X							
Komplexität		X						
Verfügbarkeit	X		X					X
Änderbarkeit			X	X	X		X	
Anzahl der Parameter		X						
Syntaktische Änderungshäufigkeit		X						
Semantische Änderungshäufigkeit							X	

Tabelle 16: Übersicht der Abhängigkeiten der Kriterien von den Einflussgrößen

Die Ermittlung der Abhängigkeiten der Einflussgrößen von den Kriterien erstreckt sich in Abbildung 53 (also im ablauforientierten Vorgehensmodell von Seite 128) über zwei Spalten

on nur in Richtung einer loseren Kopplung verschärft werden. Wenn diese schon Domänen-intern vorgeschrieben ist, ergibt die Unterscheidung an sich keinen Sinn mehr.

(„vom Anwender bei Bedarf anzupassen“ und „vordefiniert bzw. kein Anwender-Input notwendig“). Der Grund hierfür ist, dass die Einordnung von der Ausgestaltung des vorigen Schrittes abhängt: Falls die vorgegebenen Einflussgrößen ohne Ergänzungen übernommen werden, so ist für die Ermittlung der Abhängigkeiten kein Input des Anwenders notwendig. Wenn jedoch zusätzliche Einflussgrößen identifiziert wurden, die im Verlauf der weiteren Anwendung des Vorgehensmodells berücksichtigt werden sollen, so müssen für diese natürlich zunächst die abhängigen Einflussgrößen durch den Anwender ermittelt werden.

Es kann sich oftmals als sinnvoll erweisen, die Abhängigkeiten der einzelnen Einflussgrößen vor der Festlegung der Klassifikation zu betrachten bzw. sie für eigene Einflussgrößen zu ermitteln. Insbesondere bei solchen Einflussgrößen, von denen nur ein einziges Kriterium abhängt, ist es nicht sinnvoll, mehr als zwei verschiedene Alternativen im Rahmen der Klassifikation zu definieren. Schließlich ist eine Unterscheidung – ggf. verbunden mit Aufwand zur richtigen Einordnung konkreter Schnittstellen – nur dann sinnvoll, wenn sich alle identifizierten Gruppen bezüglich der Ausprägung zumindest eines Kriteriums unterscheiden.

5.3.3 Die Soll-Zuordnung der Kriterien zu den Einflussgrößen

Nachdem nun die Abhängigkeiten zwischen Kriterien und Einflussgrößen ermittelt worden sind (Übersicht in Tabelle 16), können die konkreten Zuordnungen vorgenommen werden.

Das bedeutet, dass für alle Kategorien, die im Rahmen der Klassifikation einer Einflussgröße abgegrenzt wurden (vgl. 5.3.1), jeweils eine Vorschrift für jedes Kriterium formuliert werden muss. Diese Vorschrift kann entweder die Verwendung der lose oder der eng gekoppelten Variante bestimmen, oder sie kann besagen, dass die Auswahl beliebig ist. Letzteres würde natürlich nur bedeuten, dass die lose oder enge Kopplung für die betrachtete Klassifikations-Kategorie egal ist, nicht aber für die gesamte Einflussgröße oder gar im gesamten Einsatz-Szenario.

Wenn im Rahmen der in 5.3.2 durchgeführten Untersuchungen keine Abhängigkeit zwischen einer Einflussgröße und einem Kriterium identifiziert wurde, so ist die Folge, dass zwingend bei allen Klassifikations-Kategorien „egal“ zu stehen hat. In den beiden folgenden Grafiken sind diese Spalten grau unterlegt.

Die Möglichkeiten der Vorarbeit auf theoretischer Ebene sind bereits an dieser Stelle äußerst begrenzt. Die Abhängigkeiten zwischen den Einflussgrößen und den Kriterien sind zwar in allen Situationen gleich, aber schon die Wahl der Klassifikation der einzelnen Einflussgrößen muss – wie in 5.3.1 erläutert – in den meisten Fällen in Abhängigkeit der Ausgangssituation getroffen werden. Und dies ist erst recht der Fall bei der Bestimmung der Soll-Ausprägung eines jeden Kriteriums, d.h. der Grenze zwischen enger und loser Kopplung.

Aus diesem Grund werden im Folgenden nur zwei Beispiele zur Veranschaulichung der Vorgehensweise präsentiert, d.h. weder die Klassifikation noch die festgelegten Restriktionen sind unumgänglich.

Das erste Beispiel behandelt eine relativ simple Einflussgröße, nämlich die fachliche Zusammengehörigkeit. Simpel ist sie nicht deshalb, weil die Abhängigkeiten leichter zu bestimmen sind als bei anderen, sondern weil hier nur zwei alternative Klassifikations-Kategorien vorliegen: die Kommunikation zwischen Komponenten aus derselben Domäne oder aus unterschiedlichen Domänen.

Tabelle 17 zeigt eine mögliche Zuordnung. Bei der Kommunikation in derselben Domäne steht überall die Restriktion „egal“. Es hätte dort auch die enge Kopplung zwingend vorgeschrieben werden können, nicht jedoch die lose. Denn die Inter-Domänen-Kommunikation kann ja per definitionem nicht enger gekoppelt sein als die Inner-Domänen-Kommunikation, und in der Kategorie mit den schwächsten Restriktionen muss die enge Kopplung zumindest erlaubt sein, da sie ja sonst in jedem Fall zwingend wäre und das ganze Vorgehensmodell bezüglich dieses Kriteriums ad absurdum geführt würde.

Domänen-Zugehörigkeit	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
selbe Domäne	egal	egal	egal	egal	egal	egal	egal	egal
verschiedene Domänen	asynchron	Dokument	geroutet	unabh.	über Schemata	extern	egal	egal

Tabelle 17: Die Zuordnung der Einflussgrößen zu den Kriterien am Beispiel der fachlichen Zusammengehörigkeit

Als zweites Beispiel soll der Einsatzbereich dienen. Wie in 5.3.1.2 erläutert, existieren hier sehr viele verschiedene Möglichkeiten der Klassifikation. Im Folgenden wird exemplarisch die Einteilung gemäß des Zonen- und Schichten-Modells gewählt. Eine mögliche Zuordnung auf dieser Basis ist in Tabelle 18 zu sehen.

Die „Füllung“ dieser Tabellen sollte spaltenweise erfolgen, d.h. man legt z.B. beim Einsatzbereich nacheinander für jedes Kriterium fest, wann es welche Ausprägung haben soll bzw. wo die Grenze zwischen enger und loser Kopplung angesetzt wird.

Die Anwendung hingegen ist zeilenweise vorgesehen: Man ordnet die betrachtete Schnittstelle in die Klassifikation ein und kann anschließend die geltenden Restriktionen ablesen. Wenn also beispielsweise zwei Komponenten miteinander kommunizieren sollen, die zum selben Geschäftsbereich, aber zu unterschiedlichen Applikationen gehören, so ist die Schnittstelle im Rahmen der in Tabelle 18 gewählten Klassifikation in die Kategorie „innerhalb eines Geschäftsbereichs“ einzuordnen. Das hat bei den Restriktionen in diesem Beispiel zur Folge, dass der Dokument-Stil erlaubt, jedoch nicht zwingend ist (Spalte „Nachrichten-Stil“), die Syntax-Definition aber über Schemata zu erfolgen hat usw.

Einsatzbereich	Kommuni-kation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transforma-tion	Bindung	physikalische Kopplung
innerhalb einer Applikation	egal	egal	egal	egal	egal	egal	egal	egal
innerhalb eines Geschäftsbereichs	egal	egal	egal	unabh.	Sche-mata	egal	zur Laufzeit	indirekt
im gesamten Unternehmen	egal	Doku-ment	geroutet	unabh.	Sche-mata	extern	zur Laufzeit	indirekt
auch außerhalb des Unternehmens	egal	Doku-ment	geroutet	unabh.	Sche-mata	extern	zur Laufzeit	indirekt

Tabelle 18: Die Zuordnung der Einflussgrößen zu den Kriterien am Beispiel des Einsatzbereichs (vgl. [FT06])

5.3.4 Das Ergebnis: Die Soll-Ausprägung der Kriterien

Die in 5.3.3 vorgestellten Tabellen beziehen sich jeweils auf genau eine Einflussgröße. Um nun die gesamten, für das gegebene Szenario geltenden Mindestanforderungen formulieren zu können (und damit die Soll-Seite abzuschließen), ist es notwendig, diese Tabellen zu einer einzigen zu konsolidieren.

Diese Tabelle enthält für jedes Kriterium die maximale Restriktion aus den Einzeltabellen. Mit anderen Worten, wenn z.B. die Auswahl zwischen enger und loser Kopplung fünfmal egal ist, aber bezüglich einer Einflussgröße die lose Kopplung vorgeschrieben ist, so muss die Gesamt-Tabelle beim betrachteten Kriterium als Restriktion die lose Kopplung enthalten.

Bei konkurrierenden Restriktionen, d.h. wenn für dasselbe Kriterium für eine Einflussgröße die enge, für eine andere aber die lose Kopplung vorgeschrieben ist, so kann die endgültige Auswahl der Restriktion wiederum nur in Abhängigkeit des Anwendungsfalls erfolgen. Dazu müssen die Herleitungen der konkurrierenden Restriktionen nochmals betrachtet und gewichtet werden, so dass die wichtigere Restriktion am Ende in der Gesamt-Tabelle erscheint.

Tabelle 19 zeigt eine mögliche Gesamt-Tabelle. In diesem Beispiel wird die asynchrone Kommunikation, externe Transformation und Bindung zur Laufzeit (also die lose Kopplung bezüglich dieser Kriterien) vorgeschrieben; beim Nachrichten-Stil hingegen ist mit dem RPC-Stil die enge Kopplung Pflicht. Bei allen anderen Kriterien bestehen keine Restriktionen („egal“). Dass dies bei mehreren Kriterien der Fall ist, ist nicht zwingend ein Indiz für eine nicht ausreichend strenge Anwendung des bisherigen Vorgehens. Wenn beispielsweise kein

zentrales, anwendungsunabhängiges Datenformat zur Verfügung steht, kann auch dessen Verwendung nicht zwingend vorgeschrieben werden. Auf der anderen Seite würde hier eine Restriktion zur engen Kopplung, d.h. zur Verwendung eines anwendungsabhängigen Datenformats, ein falsches Signal setzen, da hiermit die Nutzung eines eventuell in der Zukunft entwickelten Formats verboten würde.

Kriterium	**enge Kopplung**	**egal**	**lose Kopplung**
Kommunikation			asynchron
Nachrichten-Stil	RPC-Stil		
Nachrichten-Wege		egal	
Datentypen		egal	
Syntax-Def.		egal	
Transformation			extern
Bindung			zur Laufzeit
physik. Kopplung		egal	

Tabelle 19: Eine mögliche Soll-Ausprägung der Kriterien

5.3.5 Komponenten-Klassen als Hilfe

Das in den vergangenen Abschnitten präsentierte Vorgehen ist natürlich – in Abhängigkeit der Größe und Komplexität des betrachteten Szenarios – zum einen mit einem nicht unerheblichen Aufwand verbunden und birgt zum anderen die Gefahr, dass verschiedene Personen identische Szenarien unterschiedlich beurteilen.

Wie mehrfach begründet, ist eine automatisierte Anwendung dieses Vorgehensmodells nicht möglich; dennoch gibt es eine Möglichkeit der Vereinfachung über die Definition von Komponenten-Klassen. Eine solche Klasse beschreibt Komponenten, die über ähnliche Eigenschaften oder Rahmenbedingungen verfügen und deren Integration folglich auf eine ähnliche Art und Weise realisiert werden kann.

Eine Möglichkeit für eine solche Einteilung von Komponenten-Klassen sind die in 4.4.2.3 herausgearbeiteten Arten von Services (siehe Tabelle 20). Die Abkürzung „E“ bedeutet, dass die enge Kopplung vorgeschrieben ist, bei „L“ ist die lose Kopplung zwingend. Ein Strich signalisiert, dass sowohl die enge als auch die lose Kopplung bezüglich des betroffenen Kriteriums verwendet werden kann.

Auch bei dieser Tabelle ist wiederum zu betonen, dass es sich lediglich um ein Beispiel handelt. Denn auch wenn sich diese Begrifflichkeiten auch auf andere Konzepte als die SOA

übertragen lassen, ist weder die Wahl der Unterteilung noch deren Detaillierungsgrad zwingendermaßen für jedes Szenario bzw. jede Organisation passend.

Kriterium / Komponenten-Klasse	Kommunikation	Nachrichten-Stil	Bindung	Nachrichten-Wege	Datentypen	Syntax-Definition	Transformation	Physikalische Kopplung
Benutzerinteraktions-Service	E	E	–	–	–	–	–	–
Basis-Service	–	–	–	–	–	–	–	–
Daten-Service	–	–	L	–	–	–	–	–
Geschäfts-Service	L	L	L	L	L	L	–	–
Transaktions-Service	–	L	–	–	–	–	–	–
Vermittlungs-Service	E	–	–	–	–	–	–	–
öffentlicher Service	–	–	L	E	L	L	–	L

Tabelle 20: Die Übertragung der Service-Typen auf die Kriterien

5.4 Die zweite Perspektive: orientiert an den Umsetzungsmöglichkeiten (Ist)

Nachdem mit Hilfe des in 5.3 präsentierten Vorgehens die Soll-Situation, d.h. die optimale Ausprägung der in 5.1.2 vorgestellten acht Kriterien, erarbeitet worden ist, gilt es nun, geeignete Möglichkeiten zu deren Umsetzung zu finden.

Wie auch auf der Soll-Seite, so ist auch hier zunächst die Definition von Einflussgrößen und deren Klassifikation notwendig. Allerdings müssen auf der Ist-Seite zwei Arten von Einflussgrößen unterschieden werden, nämlich diejenigen, die unmittelbaren Einfluss auf die Kriterien haben, und andere, die zwar auch die Auswahl der Umsetzungsmöglichkeiten beeinflussen, aber mit dem Gedanken der losen Kopplung nichts zu tun haben. Letztere werden im Folgenden mit „sonstige, allgemeine Einflussgrößen“ bezeichnet.

Ein Beispiel hierfür sind die Kosten: Natürlich ist es nicht unerheblich für die Auswahl, wie teuer z.B. eine bestimmte Middleware ist, aber keines der acht Kriterien lässt sich dadurch verändern, dass eine teurere oder günstigere Lösung gewählt wird.

5.4.1 Die unmittelbaren Einflussgrößen und ihre Klassifikation

Die unmittelbaren Einflussgrößen sind diejenigen, durch deren Anpassung eine Veränderung der acht Kriterien erreicht werden kann, so dass deren Ausprägung der Soll-Situation entspricht.

Da es sich hierbei um nichts anderes als die Umsetzung einer Anwendungs-Integration handelt, können die in Kapitel 2 herausgearbeiteten drei Dimensionen der Anwendungs-Integration (vgl. 2.5) als die ersten drei Einflussgrößen übernommen werden. Zu ergänzen ist lediglich eine Größe, die mit „logische Kommunikations-Infrastruktur“ überschrieben wird. Denn beispielsweise die Verwendung eines unabhängigen Datenmodells ist nur dann möglich, wenn ein solches erarbeitet wurde, dann aber mit jeder Integrations-Methode und -Technologie.

Dementsprechend können auf der Ist-Seite folgende unmittelbare Einflussgrößen unterschieden werden:

1. **das zugrunde liegende Konzept**
   - Datenintegration
   - Punkt-zu-Punkt-Verbindung
   - EAI
   - SOA
2. **die Art des Zugriffs auf die Anwendungen**
   - über die Benutzungs-Schnittstelle
   - über Funktions-Aufrufe
   - über Datenbank-Zugriffe
3. **die verwendete Technologie**
4. **die logische Kommunikations-Infrastruktur**

Natürlich ist diese Aufzählung nicht so zu verstehen, dass der Anwender eine bestimmte Technologie oder ein bestimmtes Integrations-Konzept als Input einbringt. Schließlich sind diese Größen das Ergebnis des gesamten Vorgehensmodells. Die Vorgehensweise ist vielmehr dergestalt gemeint, dass bei den ersten drei Einflussgrößen zunächst alle grundsätzlich infrage kommenden Alternativen aufgelistet werden und jeweils individuell geprüft wird, ob es mit diesen möglich ist, den Soll-Anforderungen gerecht zu werden. Somit stellen die Einflussgrößen auf der Ist-Seite eine Art immer feiner werdendes Sieb dar. Wenn also beispielsweise eine asynchrone Kommunikation vorgeschrieben ist, so können alle Technologien nicht

weiter berücksichtigt werden, mit denen lediglich synchron kommuniziert werden kann. Bei der logischen Kommunikations-Infrastruktur ist zunächst zu untersuchen, welche Ausprägung der acht Kriterien mit dem Status Quo erreicht werden kann und welche Veränderungen ggf. notwendig sind, um einzelne Kriterien beeinflussen zu können. Im Anwendungsbeispiel (5.6) wird diese Vorgehensweise mit konkreten Werten demonstriert.

Während bei den ersten beiden Einflussgrößen (d.h. das zugrunde liegende Konzept und die Art des Zugriffs auf die Anwendungen) die in Kapitel 2 eingeführten Unterteilungen als Klassifikation herangezogen werden können, so ist eine solche abgeschlossene Liste bei der Technologie nur in Einzelfällen möglich. Man könnte statt einer bloßen Aufzählung verschiedener Technologien eine Gruppierung z.B. in verschiedene Middleware-Typen (d.h. objekt-, nachrichtenorientiert etc.) vornehmen, aber für jeden dieser Typen wird man am Ende eine Alternativen-Liste von undefinierter Länge haben. Diese auf theoretischer Ebene vorzudefinieren, ist nicht sinnvoll, und zwar aus denselben Gründen, warum Kapitel 3 nicht den Anspruch erheben kann, *alle* Technologien zu behandeln.

Im Gegensatz zu Kapitel 3, in dem ausdrücklich nur Basis-Technologien vorgestellt wurden, macht es hier Sinn, auch konkrete Produkte in die Auswahl einzubeziehen. Schließlich reicht es zur Beurteilung von z.B. Kosten oder IT-Standards nicht aus, nur die Technologie zu betrachten; ebenso spielt der Anbieter von Produkten, die möglicherweise auf derselben Technologie basieren, eine wichtige Rolle bei der Auswahl.

Eine detaillierte Klassifikation der logischen Kommunikations-Infrastruktur ist noch weit unrealistischer als bei der Technologie. Natürlich können hier einige Beispiele formuliert werden, um dem Anwender einen Anhaltspunkt über den Inhalt und die Intention dieser Einflussgröße an die Hand zu geben, aber die endgültige Ausgestaltung kann nur in Abhängigkeit des Anwendungsgebiets vorgenommen werden.

5.4.2 Die Zuordnung

Da auf der Ist-Seite die Anzahl und Heterogenität der Einflussgrößen wesentlich geringer ist als auf der Soll-Seite, kann hier auf eine getrennte Betrachtung der Abhängigkeiten (wie in 5.3.2 und 5.3.3) verzichtet werden. Stattdessen wird direkt mit der Zuordnung der Einflussgrößen zu den Kriterien fortgefahren. Das Ergebnis dieser Untersuchungen ist in Tabelle 21 zu sehen.

Die Notation („E“ für enge, „L“ für lose Kopplung und „E+L“, wenn beides möglich ist) entspricht der von Tabelle 20. Sie ist hier so zu interpretieren, dass beispielsweise das „L“ im Feld Kommunikation / Datenintegration aussagt, dass mit einer Datenintegration bezüglich des Kriteriums der Kommunikation nur die lose gekoppelte Variante (d.h. die asynchrone Kommunikation) realisierbar ist. Die graue Färbung der Zeile der Datentypen verdeutlich, dass hier überall „E+L“ steht; ein Indiz dafür, dass die Beeinflussung dieses Kriteriums offensichtlich nicht über die Wahl des Integrations-Konzepts erfolgen kann.

Konzept / Kriterium	Punkt zu Punkt	Daten-integration	EAI	SOA
Kommunikation	E+L	L	E+L	E+L
Nachrichten-Stil	E+L	E+L	E+L	E+L
Nachrichten-Wege	E	E	E	E+L
Datentypen	E+L	E+L	E+L	E+L
Syntax-Definition	E+L	E+L	E+L	L
Bindung	E+L	E+L	E+L	L
Transformation	E	E	E	E+L
Physikalische Kopplung	E+L	L	E+L	E+L

Tabelle 21: Die Zuordnung der Einflussgröße „Integrations-Konzept“ zu den Kriterien auf der Ist-Seite (nach [FT06])

Neben ihrer Funktion für das Vorgehensmodell verdeutlicht diese Tabelle einige interessante Tatsachen. So ist beispielsweise bei einem relativ alten Konzept wie der Datenintegration bei der Kommunikation und der physikalischen Kopplung die lose Kopplung nicht nur machbar, sondern sogar die einzige Möglichkeit. Beim viel neueren Konzept der SOA hingegen ist in beiden Fällen auch die enge Kopplung realisierbar, verbunden mit der Gefahr, durch eine falsche Implementierung eine insgesamt zu enge Kopplung zu erzeugen.

Offensichtlich hat der Gedanke der losen Kopplung, auch wenn dieser Begriff erst seit einigen Jahren zum „Hype“ geworden ist, nichts mit dem Alter der Konzepte und Technologien zu tun und war auch schon zuvor in vielen Punkten realisierbar.

5.4.3 Sonstige, allgemeine Einflussgrößen

Neben denjenigen Einflussgrößen, welche die acht Kriterien unmittelbar beeinflussen, existieren – wie erwähnt – noch weitere Parameter, die zwar nichts mit loser Kopplung zu tun haben, aber trotzdem einen zum Teil erheblichen Einfluss auf die Wahl der Umsetzungsmöglichkeit haben.

Zu diesen sonstigen, allgemeinen Einflussgrößen zählen folgende:

- benötigte Bandbreite
- Sicherheit
- Unternehmens-interne IT-Standards
- Unternehmens-interne IT-Strategie
- Kosten
- Anbieter

Um eine möglichst objektive Berücksichtigung dieser verschiedenen Größen zu gewährleisten, bietet sich folgendes Vorgehen an: Zunächst ist eine gemeinsame Metrik zu definieren, mit der sich alle Einflussgrößen vergleichen lassen. Die beste Methode hierzu ist eine Punktbewertung, beispielsweise von 1 (sehr schlecht) bis 10 (sehr gut). Dann müsste man nur noch für jede Größe definieren, wie die individuellen Ausprägungen auf dieses Punktesystem übertragen werden sollen.

Natürlich könnte auch für jedes Kriterium eine aussagekräftigere Metrik als die Punktbewertung definiert werden. Dies hätte jedoch zwei gravierende Nachteile:

- Zum einen ist bei einer individuellen Bewertung i.A. nicht ohne weitere Informationen erkenntlich, wo das absolute Minimum bzw. Maximum liegt. Ein Preis von 11.000 Euro innerhalb eines Kosten-Spektrums von 10.000 bis 1.000.000 Euro beispielsweise suggeriert einen relativ günstigen Preis. Wenn jedoch alles über 10.000 Euro als zu teuer eingestuft wird, ist dies ein Trugschluss.
- Eine Abwägung verschiedener Alternativen wird ohne einheitliche Metrik schwieriger: Wenn z.B. für einen Zusatzpreis von 1.000 Euro die Sicherheit von „sicher“ auf „sehr sicher“ erhöht werden kann, ist nicht ohne weiteres erkennbar, ob dies ein lohnender Schritt ist. Wenn aber durch die besagten 1.000 Euro z.B. die Bewertung der Sicherheit von 6 auf 8 von 10 Punkten erhöht wird, im Gegenzug aber die Kosten statt mit 9 nur noch mit 8 Punkten bewertet werden, sind die Konsequenzen direkt erkenn- und vergleichbar.

Als nächstes ist auf Basis dieser Metrik eine Zulässigkeitsgrenze zu definieren, die zum sofortigen Ausschluss führt, wenn sie auch nur bei einer Einflussgröße unterschritten wird. Der Grund für dieses Vorgehen ist, dass es für den Anwender die Möglichkeit geben sollte, Ausschlusskriterien zu definieren. Wenn also beispielsweise die Sicherheit als so niedrig angesehen wird, dass ein Einsatz nicht verantwortet werden kann, dann sollte dies nicht z.B. durch sehr niedrige Kosten bei der Bewertung egalisiert werden können.

Bei allen Alternativen, welche die Zulässigkeitsgrenze überschreiten, müssen die einzelnen Einflussgrößen gegeneinander abgewägt werden. Dafür bietet es sich an, diese als Zielkriterien in eine Nutzwertanalyse (vgl. z.B. [BL95]) einzubringen. Gemäß dem Grundprinzip dieser Analyse müssten die Zielkriterien zunächst gewichtet werden, um unterschiedliche Präfe-

renzen des Entscheidungsträgers berücksichtigen zu können. Anschließend sind die einzelnen Teilnutzwerte, also der Nutzen einer Alternative bezüglich eines Zielkriteriums, festzulegen. Aus der Summe der Produkte dieser Teilnutzwerte mit der zugehörigen Gewichtung lässt sich schließlich der Nutzwert der kompletten Handlungsalternative errechnen und mit den Nutzwerten der übrigen Alternativen auf quantitativer Basis vergleichen.

Diejenige Alternative, die insgesamt als zulässig befunden wurde und bei der Nutzwertanalyse die meisten Punkte erhalten hat, ist die Optimallösung bei den gegebenen Voraussetzungen.

5.5 Anwendungsmöglichkeiten und Grenzen dieses Vorgehensmodells

So ausführlich, wie es im Verlauf dieses Kapitels behandelt wurde, muss das Vorgehensmodell beim praktischen Einsatz zweifellos nicht durchleuchtet werden (siehe auch das Anwendungsbeispiel im nächsten Abschnitt). Insbesondere die Überlegungen im Rahmen der Soll-Zuordnungen der Kriterien zu den Einflussgrößen (vgl. 5.3.3) sind nur einmal notwendig, denn die Abhängigkeiten zwischen den Kriterien und den Einflussgrößen sind unabhängig vom konkreten Einsatzgebiet. Ebenso müssen die Zuordnungen auf der Soll-Seite (vgl. 5.4.2) allenfalls vor der ersten Anwendung überprüft werden und können anschließend als gegeben behandelt werden.

Dennoch kann dieses Vorgehensmodell nie zu einer Art Blackbox werden, die nach Eingabe von ein paar Zahlenwerten ohne weitere Analyse die optimale Integrations-Lösung liefert. Die individuelle Ausgangssituation muss an vielen Stellen einfließen, sei es in Form von einfachen Festlegungen der Art „eng“ / „lose“ / „egal“ oder aber in Form von Klassifikationen, welche die Bedeutung der einzelnen Einflussgrößen auf die jeweilige Organisation widerspiegeln. Inwieweit solche Klassifikationen zumindest innerhalb einer Organisation vorgegeben werden und inwieweit z.B. jeder Projektleiter diese selbst festzulegen hat, kann dabei im Einzelfall variieren.

Die Bezeichnung „Optimallösung“ ist mit Vorsicht zu betrachten. Wenn alle Einflussgrößen (ggf. auch zusätzliche) berücksichtigt und alle Eingaben hundertprozentig der Realität entsprechen, so sollte das Resultat auch tatsächlich das Optimum sein. Fehlerhafte Voraussetzungen oder Bewertungen führen aber natürlich im Allgemeinen auch zu fehlerhaften Ergebnissen. Wenn also beispielsweise die Sicherheit falsch beurteilt und dementsprechend einzelnen Alternativen falsche Punktzahlen zugeordnet wurden, ist es nicht zu erwarten, dass die errechnete Lösung tatsächlich das Optimum darstellt. Also bleibt festzuhalten, dass die Ergebnisse dieses Vorgehensmodells auf jeden Fall noch mit einem „gesunden Menschenverstand“ betrachtet werden sollten.

Die Erweiterung und Anpassung dieses Modells ist ausdrücklich erwünscht. Damit sind nicht nur Entscheidungen gemeint, eines der in dieser Arbeit vorgestellten acht Kriterien nicht zu

berücksichtigen. Auch das Hinzufügen von weiteren Kriterien und Einflussgrößen ist bei Bedarf durchaus erlaubt. Das Vorgehensmodell dient lediglich als Hilfestellung für die strukturierte Auswahl von Integrationskonzepten und -technologien auf der Basis von Kriterien einer losen Kopplung und unter der Zuhilfenahme zahlreicher Einflussgrößen.

5.6 Ein Anwendungsbeispiel

Im Folgenden wird die Anwendung dieses Vorgehensmodells an einem Beispiel demonstriert. Es handelt sich um keine reelle Situation, denn sowohl das betrachtete Unternehmen als auch die Anwendungen und Rahmenbedingungen sind frei erfunden (evtl. Ähnlichkeiten mit bestimmten Unternehmen wären rein zufälliger Natur). Dennoch sind alle Voraussetzungen und Entscheidungen so gewählt, dass eine solche Konstellation in einem tatsächlichen Unternehmen sehr gut denkbar ist.

An mehreren Stellen wurde die Errechnung von Zahlen (wie z.B. verschiedener Punktbewertungen) nicht weiter beleuchtet, um den Fokus auf dem eigentlichen Vorgehensmodell zu belassen. Die Relation dieser Zahlen zueinander ist aber so gewählt, wie sie auch in der Praxis vorkommen könnte.

In 5.6.1 wird zunächst die Ausgangssituation, d.h. das Unternehmen und die Anwendungslandschaft, geschildert, ehe in den darauf folgenden Abschnitten die Anwendung des Vorgehensmodells demonstriert wird.

5.6.1 Die Ausgangssituation

Die HWL AG (Hardware-Logistik) ist ein Versandunternehmen für PC-Hardware. Das Kerngeschäft besteht darin, Hardware-Teile in großen Stückzahlen von den Herstellern einzukaufen und an geschäftliche wie private Endkunden weiter zu veräußern. Zu diesem Zweck betreibt das Unternehmen ein großes Warenlager, aus dem von einem Kunden bestellte Teile bei Bedarf entnommen werden.

Die Schnittstelle zum Kunden ist bereits erfolgreich automatisiert worden: Die Bestellungen können über das Internet in Auftrag gegeben werden, und der Kunde kann sofort sehen, wie viele Exemplare seines ausgesuchten Artikels sich noch im Lager befinden. Die Basis hierfür ist ein EDV-System, das Kundenverwaltungs-System (KVS), das bereits seit mehreren Jahren im Einsatz ist. Mit seiner Hilfe können die Mitarbeiter in der Verwaltung in Echtzeit erkennen, bei welchen Teilen die Vorräte knapp werden und Nachschub vom Hersteller bestellen.

Im Lager, das sich einige Kilometer vom HWL-Verwaltungsgebäude entfernt befindet, ist ein anderes EDV-System, das Lagerverwaltungs-System (LVS) im Einsatz, mit dessen Hilfe die genaue Position eines jeden Artikels im Lager verwaltet wird. Außerdem ist es für die Verwaltung der Lieferanten zuständig. Wenn neue Ware ankommt, wird diese zwar im LVS erfasst, muss aber anschließend in gedruckter Form per Hauspost an die Verwaltung geschickt

werden, damit dort die Daten im KVS entsprechend aktualisiert werden können. In der Zwischenzeit kann es vorkommen, dass ein Kunde trotz gerade wieder aufgefüllten Lagers die Meldung erhält, dass das gewünschte Teil leider nicht mehr vorrätig ist, worauf er womöglich bei der Konkurrenz bestellt.

Um diesen Zustand zu verbessern, ist bei HWL ein Projekt gestartet worden, um eine neue Schnittstelle zwischen LVS und KVS zu implementieren, die den Ausdruck und Papierversand vollständig ablösen soll. Stattdessen soll neu im Lager angekommene Ware nach dem Erfassen im LVS auch sofort ins KVS übernommen werden.

Die Fachlichkeit ist somit zwar vorgegeben, mit der Entscheidung über die technische Umsetzung dieser Schnittstelle wurde jedoch der Projektleiter beauftragt. Derzeit sind bei HWL folgende Integrations-Technologien an anderen Stellen im Einsatz:

- Dateitransfer auf FTP-Basis,
- ein im Unternehmen selber entwickeltes Werkzeug zur Daten-Replikation,
- RMI,
- Web Services (über HTTP) und
- IBM WebSphere MQ (im Folgenden als „MQ“ bezeichnet).

Darüber hinaus wird bei HWL schon seit längerer Zeit überlegt, ob mit dem IBM WebSphere Business Integration Message Broker (WBIMB) eine professionelle Integrations-Software angeschafft werden soll. Folglich entschließt sich der Projektleiter, auch diese Möglichkeit mit zu berücksichtigen, um ggf. das entscheidende Argument für die Beschaffung entsprechender Lizenzen zu liefern.

Ein Dienst zur Realisierung einer Bindung zur Laufzeit (DNS) ist bei HWL vorhanden; für die dynamische Bestimmung der Nachrichten-Wege müsste ein solcher Dienst hingegen erst implementiert werden. Unabhängige Datentypen wurden sowohl innerhalb des KVS als auch des LVS definiert, ein unternehmensweites, einheitliches Datenmodell existiert jedoch nicht.

Auf Basis dieses Szenarios wird im Folgenden die Anwendung des Vorgehensmodells demonstriert. Soll- und Ist-Seite werden in jeweils einem eigenen Abschnitt (5.6.2 und 5.6.3) behandelt; die Anwendung der einzelnen Schritte ist dabei an einigen Stellen etwas komprimierter dargestellt als bei der theoretischen Darstellung in 5.3 und 5.4.

5.6.2 Die Soll-Seite

Der Projektleiter entscheidet sich dazu, keines der acht Kriterien generell aus der Betrachtung zu entfernen und auch kein weiteres zu definieren.

Auch die in 5.3.1 vorgestellten Einflussgrößen werden übernommen, allerdings mit einer Ausnahme: Da HWL eine relativ überschaubare Anwendungslandschaft besitzt, ist die Ab-

grenzung von verschiedenen fachlichen Domänen nicht notwendig. Daher wird die Einflussgröße „fachliche Zusammengehörigkeit“ nicht betrachtet.

Für die restlichen Größen hat der Projektleiter zunächst jeweils eine geeignete Klassifikation zu entwerfen (vgl. 5.3.1), zu deren Kategorien er im Anschluss die korrekten Soll-Ausprägungen der Kriterien zuordnen muss (vgl. 5.3.3). Dies wird in einem Schritt in tabellarischer Form in 5.6.2.1 erläutert. Da keine zusätzlichen Einflussgrößen definiert wurden, können die in 5.3.2 ermittelten Abhängigkeiten übernommen werden, so dass eine entsprechende Analyse nicht nochmals notwendig ist.

5.6.2.1 Klassifikation und Zuordnung der Einflussgrößen auf der Soll-Seite

In den folgenden Tabellen ist jeweils in der linken Spalte die vom Projektleiter gewählte Klassifikation erkennbar, die auch zuvor kurz erläutert wird. Die betrachtete Schnittstelle ordnet er dabei derjenigen Ausprägung zu, deren Zeile mit einem dicken Rahmen versehen worden ist.

Für die Klassifikation des Einsatzbereichs (Tabelle 22) wurde eine am Zonen- und Schichten-Modell (vgl. 5.3.3) orientierte Variante gewählt. Die Kategorie „innerhalb eines Geschäftsbereichs“ wurde aufgrund der geringen Unternehmensgröße weggelassen. Also wird unterschieden, ob die Schnittstelle zwei Komponenten innerhalb des LVS bzw. des KVS verbindet, ob eine Kommunikation über diese Anwendungsgrenzen hinaus stattfindet oder gar Komponenten außerhalb der Unternehmensgrenzen von HWL mit einbezogen werden. Unabhängig davon, wie die Daten über die neu geschaffene Schnittstelle vom LVS zum KVS gelangen (d.h. welche Anwendung die Schnittstelle bereitstellt), ist ein Zugriff von außerhalb der eigenen Applikationsgrenzen, nicht aber außerhalb der Unternehmensgrenzen vorgesehen. Daher wurde die Kategorie „innerhalb des gesamten Unternehmens“ ausgewählt.

Erklärungsbedürftig ist die Wahl der Zuordnung bei den Datentypen. Normalerweise ist es zu erwarten, dass die geforderte Kopplung umso loser wird, je weiter sich der Einsatzbereich der betrachteten Komponente ausdehnt. Hier jedoch ist genau das Gegenteil der Fall: Bei der Kommunikation innerhalb der eigenen Applikation wurde keine Restriktion formuliert, bei einer Ausdehnung des Einsatzbereichs sind jedoch abhängige Datentypen, also die enge Kopplung, vorgeschrieben. Der Grund für diese Festlegung ist, dass ein unabhängiges Datenmodell nur innerhalb der jeweiligen Applikationen existiert und der Projektleiter den Entwurf eines weiter reichenden unabhängigen Datenmodells grundsätzlich für zu aufwändig hält.

Ansonsten ergibt die Auswahl des Projektleiters keine bemerkenswerten Tatsachen. Für die Kommunikation innerhalb einer Applikation (falls also im Rahmen des Projekts zusätzliche Schnittstellen innerhalb des LVS oder des KVS notwendig sein sollten) werden keine Restriktionen hinsichtlich des Grades losen Kopplung definiert. Erweitert sich der Einsatzbereich auf das gesamte Unternehmen, so sollen Dokument-Stil, Bindung der Laufzeit und indirekte physikalische Kopplung verwendet werden. Bei einer Kommunikation mit außerhalb des Unter-

nehmens befindlichen Komponenten sind auch geroutete Nachrichten-Wege, eine Syntax-Definition über Schemata und externe Transformatoren notwendig.

Einsatzbereich	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
innerhalb einer Applikation	egal	egal	egal	egal	egal	egal	egal	egal
im gesamten Unternehmen	egal	Dokument	egal	abhängig	egal	egal	zur Laufzeit	indirekt
auch außerhalb des Unternehmens	egal	Dokument	geroutet	abhängig	Schemata	extern	zur Laufzeit	indirekt

Tabelle 22: Beispiel für die Klassifikation und Zuordnung des Einsatzbereichs

Die Bearbeitungsdauer (Tabelle 23) ist die erste betrachtete Einflussgröße, von der nur ein Kriterium abhängt. Insofern wird nur eine zweigeteilte Klassifikation definiert und festgelegt, dass bei einer Bearbeitungsdauer von mehr als 24 Stunden asynchrone Kommunikation zu wählen ist. Natürlich ist eine weitere Abstufung unterhalb dieses Wertes (z.B. im Sekunden-Bereich) sinnvoll, da sie aber keine Auswirkungen auf eines der Kriterien hat, hat der Projektleiter darauf verzichtet.

Gemäß der fachlichen Vorgabe sollen bei der vorliegenden Schnittstelle die Informationen über neu im Lager angekommene Waren möglichst innerhalb von Sekunden im KVS eintreffen und damit über das Internet beim Kunden zugreifbar sein. Daher ist auf jeden Fall eine Bearbeitungsdauer von weniger als 24 Stunden notwendig. Da keine nächtlichen Batch-Läufe notwendig sind und keine Genehmigungen eingeholt werden müssen, stellt dies technisch auch kein Problem dar.

Bearbeitungsdauer	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
max. 24 Stunden	egal	egal	egal	egal	egal	egal	egal	egal
mehr als 24 Stunden	asynchron	egal	egal	egal	egal	egal	egal	egal

Tabelle 23: Beispiel für die Klassifikation und Zuordnung der Bearbeitungsdauer

Auch die Komplexität (Tabelle 24) beeinflusst nur ein Kriterium. Hier wird die Grenze, bis zu der noch der RPC-Stil erlaubt ist, bei 5 beteiligten Komponenten gesetzt.

Da sowohl LVS als auch KVS keine externen Komponenten aufrufen, ist die Kategorie „maximal 5 beteiligte Komponenten“ zutreffend.

Komplexität	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
max. 5 beteiligte Komponenten	egal	egal	egal	egal	egal	egal	egal	egal
mehr als 5 beteiligte Komponenten	egal	Dokument	egal	egal	egal	egal	egal	egal

Tabelle 24: Beispiel für die Klassifikation und Zuordnung der Komplexität

Wenn eine Verfügbarkeit von weniger als 99% vorliegt, werden eine indirekte physikalische Kopplung sowie eine asynchrone Kommunikation vorgeschrieben (Tabelle 25). Sinkt die durchschnittliche Verfügbarkeit unter 80%, so ist zusätzlich eine Bindung zur Laufzeit notwendig, um Anfragen ggf. dynamisch zu alternativen Komponenten umleiten zu können.

Der Projektleiter entscheidet sich dafür, die Mindest-Verfügbarkeit auf 80 % zu setzen. 99 % oder mehr sind seiner Ansicht nach nicht notwendig, da der technische Aufwand den Zusatznutzen deutlich übersteigt.

Verfügbarkeit (Erreichbarkeit)	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
mindestens 99 %	egal	egal	egal	egal	egal	egal	egal	egal
weniger als 99 %, aber mind. 80 %	asynchron	egal	egal	egal	egal	egal	egal	indirekt
weniger als 80 %	asynchron	egal	egal	egal	egal	egal	zur Laufzeit	indirekt

Tabelle 25: Beispiel für die Klassifikation und Zuordnung der Verfügbarkeit

Wenn die verwendete Programmiersprache zwar bekannt ist, die fachlichen Zusammenhänge jedoch nicht (sei es, weil es sich um ein sehr altes System oder eine nicht selbst entwickelte Software handelt), sollen unabhängige Datentypen und externe Transformation verwendet werden. Existiert auch kein Personal mehr, das die verwendete Programmiersprache beherrscht, sind darüber hinaus geroutete Nachrichten-Wege und eine Bindung zur Laufzeit vorgeschrieben (Tabelle 26).

Da sowohl beim LVS als auch beim KVS eingearbeitete Projektteams verfügbar sind, wird hinsichtlich dieser Einflussgröße keine Restriktion formuliert.

Änderbarkeit	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
eingearbeitetes Projektteam verf.	egal	egal	egal	egal	egal	egal	egal	egal
Programmiersprache (noch) bekannt, aber Fachlichkeit nicht	egal	egal	egal	unabh.	egal	extern	egal	egal
gar nicht (mehr) änderbar	egal	egal	geroutet	unabh.	egal	extern	zur Laufzeit	egal

Tabelle 26: Beispiel für die Klassifikation und Zuordnung der Änderbarkeit

Die Grenze, ab der eine Verwendung des Dokument-Stils dem RPC-Stil vorzuziehen ist, wird vom Projektleiter auf 6 Parameter festgelegt (Tabelle 27).

Da neben dem Warennamen selbst einige weitere Informationen ins KVS eingespeist werden müssen, wird dieser Wert übertroffen, so dass aufgrund dieser Einflussgröße die Verwendung des Dokument-Stils vorgeschrieben ist.

Anzahl der Parameter	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
max. 5 Parameter	egal	egal	egal	egal	egal	egal	egal	egal
6 oder mehr Parameter	egal	Dokument	egal	egal	egal	egal	egal	egal

Tabelle 27: Beispiel für die Klassifikation und Zuordnung der Anzahl der Parameter

Bei der Änderungshäufigkeit wurde als Grenzwert ein durchschnittliches Intervall von sechs Monaten zwischen zwei Änderungen gewählt. Dessen Unterschreitung hat auf der syntaktischen Seite (Tabelle 28) den Wechsel vom RPC- zum Dokument-Stil und auf der semantischen Seite (Tabelle 29) externe Transformationen zur Folge.

Zwar wird eine bislang noch gar nicht vorhandene Schnittstelle bei HWL implementiert, jedoch ist deren Syntaktik wie auch Semantik nicht so kompliziert, dass häufig Änderungen erwartet werden.

Syntaktische Änderungshäufigkeit	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
Abstand 6 Monate oder mehr	egal	egal	egal	egal	egal	egal	egal	egal
Abstand geringer als 6 Monate	egal	Dokument	egal	egal	über Schemata	egal	egal	egal

Tabelle 28: Beispiel für die Klassifikation und Zuordnung der syntaktischen Änderungshäufigkeit

Semantische Änderungshäufigkeit	Kommunikation	Nachr.-Stil	Nachr.-Wege	Datentypen	Syntax-Def.	Transformation	Bindung	physikalische Kopplung
Abstand 6 Monate oder mehr	egal	egal	egal	egal	egal	egal	egal	egal
Abstand geringer als 6 Monate	egal	egal	egal	egal	egal	extern	egal	egal

Tabelle 29: Beispiel für die Klassifikation und Zuordnung der semantischen Änderungshäufigkeit

5.6.2.2 Das Ergebnis der Soll-Seite

Auf Basis der in 5.6.2 präsentierten, individuellen Zuordnungen der einzelnen Einflussgrößen zu den Kriterien kann nun die Soll-Seite vervollständigt werden, indem diese Einzeltabellen zu einer Gesamttabelle zusammengefasst werden. Dies wird in Tabelle 30 gezeigt.

Kriterium	**enge Kopplung**	**egal**	**lose Kopplung**
Kommunikation			asynchron
Nachrichten-Stil			Dokument-Stil
Nachrichten-Wege		egal	
Datentypen	abhängig		
Syntax-Definition		egal	
Transformation		egal	
Bindung			zur Laufzeit
physikalische Kopplung			indirekt

Tabelle 30: Das Ergebnis der Soll-Seite im Beispiel

Bei den Nachrichten-Wegen, der Syntax-Definition und der Transformation existierte bei keiner Einflussgröße bei der gewählten Klassifikation und Zuordnung eine Restriktion, so dass auch in der Gesamttabelle ein „egal“ stehen muss.

Restriktionen zur losen Kopplung ergeben sich bei der Kommunikation (aufgrund der Verfügbarkeit von weniger als 99 %), dem Nachrichten-Stil (weil mehr als 5 Parameter vorliegen), der Bindung (weil der Einsatzbereich über die eigene Applikation hinaus reicht) und der physikalischen Kopplung (aus demselben Grund sowie der Verfügbarkeit von unter 99 %).

Ebenfalls der Einsatzbereich ist dafür verantwortlich, dass bei den Datentypen eine Restriktion zur engen Kopplung vorliegt. Glücklicherweise (für den Projektleiter) schreibt keine andere Einflussgröße einen Zwang zur losen Kopplung vor; in diesem Falle hätte er abwägen müssen, welche Restriktion wichtiger ist oder ob beide gleichwertig sind und insofern in der Gesamttabelle der Wert auf „egal“ gesetzt werden kann.

5.6.3 Die Ist-Seite

Gemäß dem Vorgehensmodell hat der Projektleiter nun zu prüfen, auf welche Art und Weise der Grad der losen Kopplung zu erreichen ist, wie er in Tabelle 30 (dem Ergebnis der Soll-Seite) gefordert wird. Die Größen, die hierauf Einfluss haben, sind (wie in 5.4.1 erläutert)

1. das zugrunde liegende Konzept,
2. die Art des Zugriffs auf die Anwendungen,
3. die verwendete Technologie sowie
4. die logische Kommunikations-Infrastruktur.

Der erste Punkt (das Konzept) wurde bereits in 5.4.2 (Tabelle 21) behandelt; der Projektleiter sieht keine Veranlassung, daran etwas zu ändern. Die von ihm gewählte Zuordnung der drei übrigen Einflussgrößen (auf Basis der eingangs erwähnten, zur Verfügung stehenden Alternativen) ist im Folgenden ersichtlich.

5.6.3.1 Klassifikation und Zuordnung der unmittelbaren Einflussgrößen auf der Ist-Seite

Tabelle 31 und Tabelle 32 zeigen die Zuordnungen der Einflussgrößen „Art des Zugriffs“ und „Technologie“. Eine graue Hinterlegung einer Zeile signalisiert (wie schon in 5.4.2), dass es offensichtlich keine Beziehung zwischen dem betroffenen Kriterium und der Einflussgröße, welche in der Tabelle behandelt wird, existiert.

Alleine aufgrund der Zuordnung des Integrations-Konzepts ergeben sich noch keine Einschränkungen. Allerdings scheidet nach der Analyse der Art des Zugriffs nachträglich die Datenintegration aus, da hierfür ein Zugriff auf der Datenbank-Ebene notwendig gewesen wäre. Hiermit ist jedoch der geforderte Dokument-Stil nicht realisierbar.

Darüber hinaus kann der Zugang über die Benutzungs-Schnittstelle nicht verwendet werden, da hiermit gar keines der in Richtung einer losen Kopplung zu gestaltenden Kriterien geeignet beeinflusst werden kann).

Kriterium \ Zugriff über	Datenbank-Zugriffe	Funktions-Aufrufe	Benutzungs-Schnittstelle
Kommunikation	E+L	E+L	E
Nachrichten-Stil	E	E+L	E
Nachrichten-Wege	E	E+L	E
Datentypen	E+L	E+L	E+L
Syntax-Definition	E+L	E+L	E
Bindung	E+L	E+L	E+L
Transformation	E	E+L	E
Physikalische Kopplung	L	E+L	E

Tabelle 31: Die Zuordnung der Einflussgröße „Art des Zugriffs“ zu den Kriterien auf der Ist-Seite

Kriterium \ Technologie	Datei-transfer	Daten-replika-tion	RMI	Web Services	MQ	WBIMB
Kommunikation	L	L	E+L	E+L	L	L
Nachrichten-Stil	L	E	E	E+L	E+L	E+L
Nachrichten-Wege	E+L	E+L	E+L	E+L	E+L	E+L
Datentypen	E+L	E+L	E+L	E+L	E+L	E+L
Syntax-Definition	E+L	E+L	E+L	E+L	E+L	E+L
Bindung	E+L	E+L	E+L	E+L	E+L	E+L
Transformation	E+L	E+L	E	E+L	E+L	E+L
Physikalische Kop-plung	L	L	E	E	L	L

Tabelle 32: Die Zuordnung der Einflussgröße „Technologie“ zu den Kriterien auf der Ist-Seite

Nach dieser Zuordnung scheiden folgende Technologien aus der weiteren Betrachtung aus:

- das Datenreplikations-Werkzeug (wegen der fehlenden Unterstützung des Dokument-Stils)
- RMI (weil kein Dokument-Stil und keine indirekte physikalische Kopplung realisiert werden kann)
- Web Services über HTTP (aufgrund der fehlenden Möglichkeit der indirekten physikalischen Kopplung)

Bei der Kommunikations-Infrastruktur (Tabelle 33) ergeben sich neben der Beibehaltung des Status Quo zwei Varianten, den ermöglichten Grad der losen Kopplung zur ändern, nämlich durch die Implementierung eines Dienstes zur dynamischen Bestimmung der Nachrichten-Wege oder durch die Ausarbeitung eines unabhängigen Datenmodells.

Man kann an diesem Beispiel gut erkennen, dass bei der logischen Kommunikations-Infrastruktur im Gegensatz zu den übrigen Einflussgrößen i.A. keine Klassifikation und erst recht keine Metrik möglich ist. Stattdessen werden hier verschiedene, individuelle Maßnahmen aufgelistet und bewertet, die generell voneinander unabhängig sind.

Maßnahme / **Kriterium**	Beibehaltung des Status Quo	Dienst zur dynamischen Bestimmung der Nachr.-Wege	unabhängiges Datenmodell
Kommunikation	E+L	E+L	E+L
Nachrichten-Stil	E+L	E+L	E+L
Nachrichten-Wege	E	E+L	E
Datentypen	E	E	E+L
Syntax-Definition	E+L	E+L	E+L
Bindung	E+L	E+L	E+L
Transformation	E+L	E+L	E+L
Physikalische Kopplung	E+L	E+L	E+L

Tabelle 33: Die Zuordnung der Einflussgröße „Kommunikations-Infrastruktur“ zu den Kriterien auf der Ist-Seite

Die Kommunikations-Infrastruktur muss nicht verändert werden, da die Soll-Anforderungen bereits mit dem Status Quo erreicht werden können.

Die Kombination einer Punkt-zu-Punkt-Verbindung und des WBIMB sieht der Projektleiter als nicht erstrebenswert an und schließt sie von vornherein aus. Ein solcher Schritt ist zwar nicht ausdrücklich im Vorgehensmodell vorgesehen, doch es wäre nicht zielführend, die

Möglichkeit zu verbieten, bestimmte potenzielle Alternativen auf Basis des „gesunden Menschenverstandes“ oder anderer, nicht mit dem Vorgehensmodell erfassten Gründen auszuschließen. Dieses erhebt schließlich den Anspruch, eine Entscheidungshilfe für die beteiligten Personen zu sein und nicht ein starrer, theoretischer Rahmen, der unnötige Arbeitsschritte erzeugt.

Folglich bleiben nach der Berücksichtigung der unmittelbaren Einflussgrößen folgende Realisierungsmöglichkeiten übrig (Zugriff jeweils über Funktions-Aufrufe):

1. Punkt zu Punkt mit FTP
2. Punkt zu Punkt mit MQ
3. EAI mit FTP
4. EAI mit MQ
5. EAI mit WBIMB
6. SOA mit FTP
7. SOA mit MQ
8. SOA mit WBIMB

Mit diesen Kombinationen können die auf der Soll-Seite definierten Anforderungen umgesetzt werden. Damit ist ist allerdings noch nichts darüber gesagt, ob beispielsweise die Sicherheit angemessen ist oder unternehmensinterne Richtlinien eine Alternative verbieten. Diese sonstigen Einflussgrößen werden nun im nächsten Abschnitt berücksichtigt.

5.6.3.2 Berücksichtigung der sonstigen, allgemeinen Einflussgrößen und Errechnung der Optimallösung

Der letzte Schritt des Vorgehensmodells besteht in der Berücksichtigung der sonstigen, allgemeinen Einflussgrößen und in der Bestimmung der Optimallösung mit Hilfe einer Nutzwertanalyse. Dies ist in Tabelle 34 zu sehen. Aufgrund der in 5.4.3 erläuterten Probleme bei nicht vergleichbaren Metriken entscheidet sich der Projektleiter für eine einheitliche Punktbewertung von 1 (sehr schlecht) bis 10 (sehr gut). Die erste Zeile der Tabelle enthält die Gewichtungen, d.h. die Faktoren, welche die Bedeutung berücksichtigen, die der Projektleiter den einzelnen Größen beimisst. In der zweiten Zeile wird festgelegt, welche Punktzahlen grundsätzlich als zulässig angesehen werden können. Aufgrund von diesen Beschränkungen können die Alternativen der Punkt-zu-Punkt-Verbindung mit FTP (durch die zu geringe Sicherheit) und EAI mit WBIMB (durch die zu hohe beanspruchte Bandbreite) nicht genutzt werden.

Alle anderen Alternativen sind grundsätzlich zulässig und werden auf Basis der Nutzwertanalyse miteinander verglichen. Das Ergebnis errechnet sich nach der Formel (Bewertung der Bandbreite) * (Gewichtung der Bandbreite) + (Bewertung der Sicherheit) * (Gewichtung der Sicherheit) usw. geteilt durch die Summe der Gewichtungen (hier 15). Bei den Punkt-zu-Punkt-Verbindungen über MQ sieht das also z.B. so aus: (7*1 + 5*2 + 10*3 + 3*2 + 9*4 + 10*3) / 15 = 7,93.

Die Optimallösung gemäß aller Eingaben und Voraussetzungen, die der Projektleiter festgelegt hat, ist also die Punkt-zu-Punkt-Verbindung mit MQ.

	benötigte Bandbreite	Sicherheit	Unternehmens-interne IT-Standards	Unternehmens-interne IT-Strategie	Kosten	Anbieter	zulässig?	Ergebnis (gewichteter Durchschnitt)
Gewichtung	1	2	3	2	4	3		
zulässig bei Punktzahl	>2	>4	>4	>3	>1	>5		
Punkt zu Punkt mit FTP	10	**2**	6	3	10	3	nein	*5,80*
Punkt zu Punkt mit MQ	7	5	10	3	9	10	ja	**7,93**
EAI mit FTP	9	6	6	6	5	3	ja	5,33
EAI mit MQ	4	8	10	6	4	10	ja	7,20
EAI mit WBIMB	**2**	10	9	8	2	10	nein	*6,87*
SOA mit FTP	7	5	6	9	6	3	ja	5,73
SOA mit MQ	5	7	10	9	5	10	ja	7,80
SOA mit WBIMB	5	9	9	10	2	10	ja	7,20

Tabelle 34: Die Anwendung der sonstigen Einflussgrößen und die Nutzwertanalyse im Beispiel

5.6.4 Bemerkungen zu diesem Anwendungsbeispiel

Die vielfältigen Anwendungsmöglichkeiten des Vorgehensmodells können nicht mit einem einzigen Beispiel abgedeckt werden. Es wurde zur Demonstration ein bewusst einfaches Szenario gewählt: Das Unternehmen HWL ist relativ klein, die Anwendungslandschaft folglich äußerst überschaubar, und das Ziel des Projektleiters, der das Vorgehensmodell anwendet, ist die Gestaltung einer einzigen Schnittstelle. Die Ausgangslage würde natürlich wesentlich komplizierter, wenn es um die Entwicklung einer neuen, sehr komplexen Anwendung (mit vielen internen Schnittstellen) und deren Integration in die bestehende, heterogene Systemlandschaft geht. Dann würde auch die Klassifikation der verschiedenen Einflussgrößen auf

der Soll-Seite eine viel entscheidendere Bedeutung erlangen, da das Ziel tatsächlich die Einordnung mehrerer Schnittstellen wäre. Dennoch hat das vorliegende Anwendungsbeispiel gezeigt, dass die Anwendung des Vorgehensmodells auch unter den relativen einfachen Rahmenbedingungen durchaus sinnvoll ist, denn die gewonnenen Erkenntnisse sind auch hier keinesfalls offensichtlich.

Die in diesem Beispiel gewählten Rahmendaten hätten an manchen Stellen noch konkretisiert werden können (z.B. detaillierte Angaben dazu, wie der Projektleiter die einzelnen Punktbewertungen im Rahmen der Nutzwertanalyse errechnet bzw. bestimmt hat). Doch da diese Informationen nicht zum Vorgehensmodell an sich gehören, sondern nur Input darstellen, wurde aus Gründen der Übersichtlichkeit bewusst darauf verzichtet. Dass auf der Ist-Seite an relativ wenigen Stellen auf die Anwendungssituation Bezug genommen wurde, begründet sich durch die Struktur des Vorgehensmodells, denn die Rahmenbedingungen haben ja im Wesentlichen auf die Soll-Seite Einfluss, während das Ziel der Ist-Seite das Finden geeigneter Umsetzungsmöglichkeiten hierfür ist.

5.7 Ähnliche Ansätze in der Literatur

Es ist durchaus wahrscheinlich, dass insbesondere Beratungsunternehmen über interne Richtlinien und Erfahrungsberichte verfügen, deren Zielsetzung mit dem in diesem Kapitel vorgestellten Vorgehensmodell übereinstimmt. In der öffentlich zugänglichen Literatur scheint es etwas Derartiges hingegen nicht zu geben. Beschränkt man die Suche auf Teilbereiche, wird man hingegen fündig, beispielsweise bei Untersuchungen verschiedener Technologien auf ihre Eignung zur Unterstützung einer SOA (siehe hierzu auch 4.4.7).

Ein Vergleich verschiedener Integrations-Architekturen, der im Folgenden kurz vorgestellt wird, stammt von Aier und Schönherr [AS06]. Unter diesem Stichwort subsummieren sie individuelle Schnittstellen, die Hub-and-Spokes-Architektur und die SOA. Mit Ausnahme der Datenintegration sind dies also genau die Integrations-Konzepte, die im Rahmen des in dieser Arbeit präsntierten Vorgehensmodells Berücksichtigung finden.

Für den Vergleich wurden folgende Kriterien herausgearbeitet (übersetzt nach [AS06]):

- **Anfänglicher Planungs-Aufwand** (insbesondere bezogen auf die Weiterbildung des vorhandenen Personals oder die Beauftragung externer Mitarbeiter)
- **Anfänglicher Entwicklungs-Aufwand**
- **Konsistente Modellierung** (d.h. Methoden, Notationen und Werkzeuge zur konsistenten Modellierung sowohl von Geschäftsprozessen als auch von Workflows)
- **Technische Adaption** (d.h. Aufwände, um die Integrations-Architektur an neue Bedürfnisse anzupassen)
- **Legacy- / Host-Integration** (bezogen auf ihre eingeschränkte oder gar nicht vorhandene Änderbarkeit)

- **Sicherheit**
- **Wartbarkeit**
- **Anpassbarkeit** (auf nicht-technischer Ebene; vgl. „Technische Adaption“)
- **Stabilität** (technische Verfügbarkeit)
- **Transaktions-Unterstützung** (d.h. Sicherstellung der Atomarität einer Transaktion und Bereitstellung von Rollback-Mechanismen)
- **Kosten**

Im Anschluss an die Formulierung dieser Kriterien wird untersucht, inwieweit sie von den drei betrachteten Intrations-Architekturen unterstützt werden bzw. wie gut sie die Anforderungen erfüllen. Das Ergebnis ist in Tabelle 35 zu sehen. Die Skala reicht hierbei von „sehr schlecht“ (– –) über „schlecht“ (–), „neutral“ (0) und „gut“ (+) bis „sehr gut“ (+ +).

Kriterium	**individuelle Schnittstellen**	**Hub and Spokes**	**SOA**
Anfänglicher Planungs-Aufwand	+	– –	–
Anfänglicher Entwicklungs-Aufwand	–	+	–
Konsistente Modellierung	–	+ +	+
Technische Adaption	+	0	+
Legacy- / Host-Integration	– –	+	0
Sicherheit	+ +	+	–
Wartbarkeit	– –	+ +	–
Anpassbarkeit	– –	+ +	–
Stabilität	+ +	+	0
Transaktions-Unterstützung	+ +	0	–
Kosten	– –	+	0

Tabelle 35: Evaluation von Integrations-Architekturen nach [AS06]

Es ist ersichtlich, dass sich einige dieser Kriterien – zum Teil wörtlich, zum Teil sinngemäß – bei den Einflussgrößen auf der Ist-Seite (5.4) des in dieser Arbeit präsentierten Vorgehensmodells wiederfinden lassen. Alle anderen würden sich problemlos dort einfügen lassen – je nach den Präferenzen des Anwenders.

Über die Ist-Seite hinaus geht der Ansatz von Aier und Schönherr jedoch nicht. Es werden zwar die verschiedenen Integrations-Architekturen hinsichtlich der Erfüllung der sehr detail-

lierten und aufwändig erarbeiteten Kriterien untersucht, jedoch ist die Übertragung dieser Kriterien auf die individuelle Ausgangssituation (d.h. wann welche Ausprägung sinnvoll ist) dem Anwender vorbehalten. Es ist allerdings zu betonen, dass dies in der erwähnten Studie auch nicht beabsichtigt war.

Kapitel 6: Ausblick

Die in Kapitel 4 vorgestellten Integrations-Konzepte sind nicht mehr als eine Momentaufnahme des Fortschritts, die Basis-Technologien in Kapitel 3 sogar lediglich ein kleiner Ausschnitt des technologischen Lösungsportfolios. Mit Sicherheit wird die Entwicklung der Integrations-Technologien bei den Web Services nicht stehen bleiben; ebenso wenig ist zu erwarten, dass die SOA über die nächsten Jahrzehnte oder vielleicht auch nur Jahre den Grad an Aufmerksamkeit behalten wird, den sie heute hat. So hat beispielsweise Gartner Research mit der Web Oriented Architecture (WOA), die als SOA auf Web-Basis definiert wird [Smi06], bereits ein neues Schlagwort geschaffen, das möglicherweise demnächst die Aufmerksamkeit der „breiten Öffentlichkeit“ erlangen wird.

Die Bedeutung dieser beiden Kapitel ist auch vielmehr darin zu sehen, dass sie dem Anwender einen Eindruck der vielfältigen Möglichkeiten zur Anwendungs-Integration gibt. Es soll ihm vor Augen führt werden, dass auch Datenintegration oder Punkt-zu-Punkt-Verbindungen auf der Konzept- sowie Dateitransfer auf der Technologie-Seite keine geduldeten Altlasten sind, sondern heute wie in Zukunft ernsthafte Alternativen darstellen, deren Vor- und Nachteile bei jedem Integrations-Szenario aufs Neue denen vermeintlich modernerer Lösungen gegenüber gestellt werden sollten.

In Kapitel 5 wurde an vielen Stellen deutlich, dass das dort präsentierte Vorgehensmodell nur bis zu gewissen Grenzen auf theoretischer Ebene, d.h. ohne Kenntnis des individuellen Anwendungs-Szenarios, vorformuliert werden kann. Schließlich variieren die Kriterien und Einflussgrößen, die sinnvollerweise betrachtet oder nicht betrachtet werden müssen, ebenso wie Maßstäbe, an denen sich ihre Klassifikation orientiert, von Branche zu Branche, von Unternehmen zu Unternehmen oder vielleicht sogar von Abteilung zu Abteilung.

Was mit diesem Vorgehensmodell aber in jedem Falle erreicht werden kann, ist, ein Gespür dafür zu schaffen, von welch vielfältigen Faktoren die Wahl der optimalen Integrationslösung abhängen kann. Wenn sich der Anwender darüber bewusst wird, dass Integrationsmethoden und -technologien voneinander zu trennen sind, dass eine lose Kopplung mehr bedeutet als nur asynchronen Nachrichten-Austausch, dass die „loseste“ Kopplung nur in Einzelfällen das Optimum darstellt und mit welchen Maßnahmen sich der „Grad der losen Kopplung“ beeinflussen lässt, dann kann man bereits mit Recht von einem Erfolg dieses Vorgehensmodells und dieser Arbeit insgesamt sprechen.

Anhang A: Begriffsdefinitionen und -abgrenzungen

In diesem Anhang werden einige Begriffe und Abkürzungen Themenbereich der Anwendungs-Integration erläutert. Einige dieser Abkürzungen, wie z.B. EAI oder SOA, sind zentrale Bestandteile dieser Arbeit und wurden dementsprechend bereits ausführlich beschrieben, so dass dieser Anhang als Glossar dienen kann. Andere Abkürzungen, wie z.B. EII, wurden allenfalls beiläufig erwähnt und sollen hier der Vollständigkeit halber erläutert und ggf. von anderen Termini abgegrenzt werden.

A.1 EAI

Die Abkürzung EAI steht für Enterprise Application Integration.

Entgegen einigen Quellen (z.B. [Kai02]) wird EAI in dieser Arbeit nicht als Oberbegriff über alle Integrations-Konzepte benutzt (hierfür wird die Bezeichnung „Anwendungs-Integration“ gebraucht), sondern repräsentiert den Architektur-Ansatz, der unter diesem Namen bekannt geworden ist.

Dieser so genannte Hub-and-Spokes-Ansatz („Nabe und Speichen“) ist gekennzeichnet durch eine in logischer und physischer Hinsicht zentrale Komponente (Hub oder Broker genannt), an welche die Applikationen mit Hilfe von Adaptern (auch Konnektoren genannt) sternförmig angebunden werden.

Die Kommunikation erfolgt von einer Anwendung zunächst zum zentralen Broker und wird mit dem Adapter der Quellanwendung in ein neutrales, anwendungsunabhängiges Zwischen- oder Metaformat umgewandelt. Auf dieser Basis ist die Umwandlung in das Format der Zielanwendung (mit Hilfe des zugehörigen Adapters) möglich.

Ein weiteres, wichtiges Merkmal dieses Ansatzes ist, dass im Gegensatz zur Service-orientierten Architektur (SOA; vgl. A.2 und 4.4) die Anwendungen weiterhin als solche erkennbar und nicht als Services gekapselt sind.

Dieser klassische EAI-Ansatz wird im Detail in 4.3 beleuchtet.

A.2 SOA

SOA bedeutet Service-orientierte Architektur (siehe hierzu im Detail 4.4). Dahinter verbirgt sich kein Produkt (wie z.B. Middleware), sondern – wie der Name schon andeutet – ein abstrakter Software-Architektur-Ansatz.

Der Kerngedanke ist, dass die Funktionalität aus den Anwendungen in Form von Services (Diensten) gekapselt wird. Diese können dann durch ihre sehr lose Kopplung (vgl. A.6 oder 5.1) ohne detaillierte Kenntnis z.B. ihrer Programmiersprache, ihres Betriebssystems oder ihrer Datenbank-Struktur von Applikationen oder anderen Services aufgerufen werden.

Bestehende Services können beispielsweise zu neuen Anwendungen oder kaskadierten Services (also solchen, die selbst wiederum aus mehreren untergeordneten Services bestehen) zusammengesetzt werden. Den größten Gewinn aber bringt eine SOA für die Abbildung von fachlichen Geschäftsprozessen auf die vorhandenen IT-Systeme. Wenn die Services grobgranular genug entworfen worden sind, können sie einfach als „black boxes“ zu einem Geschäftsprozess verknüpft werden, der auch dementsprechend einfach geändert werden kann.

Web Services (vgl. 3.5) sind eine Technologie, die aus mehreren Gründen besonders gut für die technische Umsetzung einer SOA geeignet ist. Trotzdem ist weder eine SOA ausschließlich mit Web Services realisierbar, noch bedeutet der Einsatz von Web Services automatisch die Implementierung einer SOA.

A.3 Datenintegration

Datenintegration bezeichnet eine spezielle Form der Anwendungs-Integration, bei der nur die Datenhaltungs-Schicht der Anwendungen, nicht aber die Anwendungslogik selbst involviert ist [Lint04] [Kel02]. Auf dieses Konzept wird detailliert in 4.1 eingegangen.

Datenintegration kann in zweierlei Weise realisiert werden:

1. Daten-Replikation (d.h. Kopieren von Datensätzen von einer Datenbank in eine andere unter bewusster Inkaufnahme von Redundanzen)
2. Daten-Föderation (Vereinigung)
    a. logische Vereinigung (virtuelle Datenbanken)
    b. physische Vereinigung (gemeinsame, physische Datenbank)

Ein SQL-Zugriff aus einer Anwendung heraus auf eine fremde Datenbank ist somit keine Datenintegration, da die Logik der zugreifenden Anwendung an der Integration beteiligt ist. Es handelt sich hierbei stattdessen z.B. um eine Punkt-zu-Punkt-Verbindung (je nach Ausgestaltung) mit einem Zugriff auf die Zielanwendung auf der Datenhaltungs-Schicht (vgl. hierzu auch 4.1.3 sowie die drei Dimensionen der Anwendungs-Integration in 2.5).

A.4 EII

Enterprise Information Integration (EII) ist nach [BLM05] eine Art der Datenintegration, die es Anwendungen ermöglicht, auf Daten aus verschiedenen Datenquellen zuzugreifen, ohne dass diese Daten repliziert werden müssen.

Das Prinzip ist somit identisch mit dem der Daten-Föderation (vgl. 4.1.2); folglich handelt es sich hierbei um einen Teilbereich der Datenintegration (vgl. A.3 und 4.1).

EII kann EAI und vergleichbare Konzepte unterstützen, da mit der Integration der Daten auch eine Vereinheitlichung und Abstraktion verbunden ist.

Dennoch ist EII nicht dasselbe, da hierunter ein „Pulling"-Mechanismus zu verstehen ist, der eigenständig die Daten aus den verschiedenen Systemen extrahiert, während bei EAI die Initiative von den einzelnen Anwendungen ausgeht und somit ein „Pushing"-Prinzip zugrunde liegt.

Ferner ist EII komplementär zum ETL-Prinzip (vgl. A.5), indem eine Datenbasis mit Echtzeit-Synchronisation bereitgestellt wird, auf deren Basis mit Hilfe von ETL komplexe Transformationen durchgeführt werden können.

Im Rahmen dieser Arbeit wird nicht die Abkürzung EII, sondern der Begriff der Daten-Föderation verwendet.

A.5 ETL

Der ETL-Prozess (die Abkürzung steht für „extract, transform, load") ist ein wesentlicher Prozess beim Betrieb eines Data Warehouse (vgl. 4.1.4). Er besteht – wie die Abkürzung schon andeutet – aus den folgenden drei Teilen:

1. **Extraktion** der benötigten Daten aus den operativen Systemen
2. **Transformation** dieser Daten, d.h. beispielsweise Anpassung der Datenstruktur
3. **Laden** der Daten in das Data Warehouse

Die Funktionsweise von ETL ist ähnlich zu der der Datenintegration (und damit auch zu EII als einer Untermenge), allerdings werden unterschiedliche Integrations-Probleme adressiert: Während Datenintegration bedarfsgesteuert i.A. auf kleinere Datenmengen zugreift, wobei Echtzeit-Anforderungen gestellt werden, liegt der Fokus bei ETL auf dem Laden eines Data Warehouse oder eines anderen persistenten Datenspeichers, das üblicherweise periodisch durchgeführt wird [BLM05] (siehe hierzu im Detail 4.1.4).

A.6 Lose Kopplung

Generell werden unter dem Begriff der losen Kopplung zwei Ziele verfolgt [Tie06]:

1. die Entkopplung von Sender und Empfänger (in dem Sinne, dass der Aufrufer einer Software-Komponente so wenig wie möglich über deren Anbieter wissen muss, um die Funktionalität nutzen zu können)
2. eine robuste, d.h. wenig fehleranfällige Kommunikation

Lose und enge Kopplung sind jedoch keine klar abgegrenzten Zustände oder Eigenschaften. Passender wäre die Bezeichnung „Grad der losen Kopplung“ [Tie06], da sich die o.g. Ziele aus verschiedenen Kriterien ergeben, die individuell erfüllt oder nicht erfüllt sein können und somit diesen Grad der losen Kopplung erhöhen oder verringern.

Diese Kriterien sind die folgenden [Kay03] [KBS05]:

1. Asynchrone Nachrichtenübermittlung
2. Dokument-Stil statt RPC-Stil
3. Geroutete Nachrichtenwege
4. Unabhängige Datentypen
5. Syntax-Definition über Schemata
6. Externe semantische Transformation
7. Bindung zur Laufzeit
8. Keine direkte physikalische Kopplung

Eine nähere Erläuterung dieser Kriterien, die die Basis der in Kapitel 5 vorgestellten Konzepte bilden, ist in 5.1 zu finden.

Literatur

[AS06] Stephan Aier, Marten Schönherr: Evaluating Integration Architectures – A Scenario-Based Evaluation of Integration Technologies, in: D. Draheim, G. Weber (Hrsg.): TEAA 2005, LNCS 3888, S. 2-14, Berlin/Heidelberg 2006

[Bie05] Adam Bien: Von Objekt-Killern und SOA-Autobahnen, in: Javamagazin 11.2005, S. 52-60

[BL95] Hans Blohm, Klaus Lüder: Investition, 8. Auflage, Verlag Vahlen, München 1995

[BLM05] Stephen Brobst, Evan Levy, Craig Muzilla: BI Expert's Perspective, in: Business Intelligence Journal, Volume 10, Number 2, Spring 2005

[Boo+04] David Booth u.a.: Web Services Architecture. W3C Working Group Note 11 February 2004, http://www.w3.org/TR/ws-arch/wsa.pdf

[Cha04] David A. Chappell: Enterprise Service Bus, O'Reilly 2004

[Con97] Stefan Conrad: Föderierte Datenbanksysteme, Springer-Verlag, Berlin / Heidelberg 1997

[CT04] Brent Carlson, Dmitry Tyomkin: Service-Oriented Architecture: Elements of Good Design, in: Business Integration Journal, Jan. 2004

[Dat04] Todd Datz: What You Need to Know About Service Oriented Architecture, in: CIO Magazine, Januar 2004, http://www.cio.com/archive/ 011504/soa.html

[Deg05] Arne Degenring: Enterprise Service Bus, in: Javaspektrum 4/2005, S. 16-18

[EAIF04] EAI-Forum, http://www.eaiforum.de, Zugriff am 06.08.2004

[EF03] Andreas Eberhart, Stefan Fischer: Web Services – Grundlagen und praktische Umsetzung mit J2EE und .NET, Hanser-Verlag, München/Wien 2003

[Emm03] Wolfgang Emmerich: Konstruktion von verteilten Objekten, dpunkt, Heidelberg 2003

[Erl04] Thomas Erl: Service-Oriented Architecture – A Field Guide to Integrating XML and Web Services, Prentice Hall 2004

[Fel05] Carsten Felden: Vorlesung „Grundlagen des Electronic Commerce", Duisburg 2005, http://www.uni-duisburg.de/FB5/BWL/WI/download/gec/gec_05_teil4.pdf

[FT06] Stefan Fischer, Marcus Tiedemann: Auswahl von Methoden und Technologien für die Integration von Anwendungen, in: WISU – Das Wirtschaftsstudium 6/2006, S. 812-817

[Gal04] Björn Eric Gallas: Der Aufbau eines Service Life Cycle Managements für eine Service Orientierte Architektur als Brücke zwischen Geschäftsprozess und IT Integration, in: Stephan Aier, Marten Schönherr (Hrsg.): Enterprise Application Integration – Serviceorientierung und nachhaltige Architekturen, Berlin 2004

[Gar04] o.V.: The Gartner Glossary of Information Technology Acronyms and Terms, Gartner Research 2004

[Ged05] gedas AG: RVS portable, Version 4.00, Referenzhandbuch, Berlin 2005, http://rvs.info.gedas.de/docu/german/version_4.00/Referenzhandbuch_4.00.pdf

[Ged06a] gedas AG: rvs – Rechner-Verbund-System, http://rvs.info.gedas.de, Zugriff am 24.02.2006

[Ged06b] gedas AG: RVS Technik, https://servicenet.gedas.de/gedasag/international/servicenet/de/datenaustausch/rvs/rvs_technik.htx, Zugriff am 27.02.2006

[Has00] Wilhelm Hasselbring: Information System Integration, in: Communications of the ACM, Vol. 43, No. 6 (Juni 2000), S. 33-37

[HBO97] Burkhard Huch, Wolfgang Behme, Thomas Ohlendorf: Rechnungswesen-orientiertes Controlling, Physica-Verlag, Heidelberg 1997

[He03] Hao He: What is Service-Oriented Architecture?, 2003, http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html, Zugriff am 16.01.2006

[Hei89] Heidi Heilmann: Integration: Ein zentraler Begriff der Wirtschaftsinformatik im Wandel der Zeit, in: Handbuch der modernen Datenverarbeitung (HMD) 26 (1989), 150, S. 46-58

[Herg03] Klaudia Hergula: Daten- und Funktionsintegration durch föderierte Datenbanken, Kaiserslautern 2003

[Herz01] Peter Herzum: Web Services and Service-Oriented Architectures, in: Cutter Distributed Enterprise Architecture Advisory Service, Executive Report, Vol. 4, No. 10, 2002

[HJ06] Bernhard Humm, Oliver Juwig: Eine Normalform für Service, in: Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Software Engineering 2006: Fachtagung des GI-Fachbereichs Softwaretechnik, Leipzig/Bonn 2006

[Hor05] Thorsten Horn: Internet Glossar, http://www.torsten-horn.de/glossar/GlossarE.htm, Zugriff am 20.10.2005

[HS00a] Peter Herzum, Oliver Sims: Business Component Factory, John Wiley and Sons 2000

[HS00b] Andreas Heuer, Gunther Saake: Datenbanken: Konzepte und Sprachen, MITP, Bonn 2000

[HW04] Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns, Addison Wesley 2004

[IBM06a] IBM: WebSphere MQ, http://www-306.ibm.com/software/integration/wmq/, Zugriff am 28.02.2006

[IBM06b] IBM: WebSphere Message Broker 6.0 Introduction, März 2006, ftp://ftp.software.ibm.com/software/integration/wbibrokers/docs/V6.0/messagebroker_Introduction.pdf, Zugriff am 01.06.2006

[Inm96] William H. Inmon: Building the Data Warehouse, New York u.a. 1996

[ISOoJ] International Organization for Standardization (ISO): Documents EDIFACT Syntax Version 4, o.J., http://www.gefeg.com/jswg/v4/data/v4_docs.htm, Zugriff am 11.07.2006

[JMP02] A. D. Jhingran, N. Mattos, H. Pirahesh: Information Integration: A research agenda, in: IBM Systems Journal, Vol. 41, No. 4, Dez. 2002

[Kai02] Michael Kaib: Enterprise Application Integration: Grundlagen, Integrationsprodukte, Anwendungsbeispiele, Wiesbaden 2002

[KaroJ] Chris Karakas: Java – Geschichte und Ausblick, o.J., http://www.karakas-online.de/teia/JAVA/java_1_1_1.htm, Zugriff am 22.03.2006

[Kay03] Doug Kaye: Loosely Coupled: The Missing Pieces of Web Services, RDS Press 2003

[Kel02] Wolfgang Keller: Enterprise Application Integration, Heidelberg 2002

[KBS05] Dirk Krafzig, Karl Banke, Dirk Slama: Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall 2005

[Linn98] Claudia Linnhoff-Popien: CORBA – Kommunikation und Management, Springer, Berlin/Heidelberg 1998

[Linß95] Heinz Linß: Integrationsabhängige Nutzeffekte der Informationsverarbeitung, Wiesbaden 2005

[Lint00] David S. Linthicum: Enterprise Application Integration, Addison Wesley 2000

[Lint04] David S. Linthicum: Next Generation Application Integration: From Simple Information to Web Services, Addison-Wesley 2004

[LF03] Boris Lublinsky, Michael Farrell Jr.: 10 Misconceptions About Web Services, in: EAI Journal, Feb. 2003

[LT03] Boris Lublinsky, Dmitry Tyomkin: Dissecting Service-Oriented Architectures, in: Business Integration Journal, Okt. 2003

[MA02] Daniel A. Menasce, Virgilio A. F. Almeida: Capacity Planning for Web Services, Prentice Hall 2002

[Mei04] Günther Meinhold: EAI und SOA: Die Komponenten fallen nicht vom Himmel, in: Objektspektrum 2/2004, S. 33-36

[Mic05] Microsoft Corporation: Microsoft on the Enterprise Service Bus (ESB), August 2005, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/47850cbd-63ed-4370-a467-6bd320636902.asp

[Mic06] Microsoft Corporation: Understanding BizTalk Server 2006, http://www.microsoft.com/biztalk/techinfo/whitepapers/understanding.mspx, Zugriff am 13.07.2006

[MSW03] Eitel von Maur, Joachim Schelp, Robert Winter: Integrierte Informationslogistik – Stand und Entwicklungstendenzen, 2003, http://web.iwi.unisg.ch/org/iwi/iwi_pub.nsf/wwwPublAuthorGer/4761764BF67EC4B3C125713000561FFB/$file/Integrierte%20Informationslogistik.pdf

[Mül06] Jan Müller: Vergleich der Performance und Sicherheit verschiedener Integrations-Technologien, Diplomarbeit, Braunschweig 2006

[Nas97] D. Nash: ODETTE File Transfer Protocol (RFC 2204), http://www.faqs.org/ftp/rfc/pdf/rfc2204.txt.pdf

[NL05] Eric Newcomer, Greg Lomow: Understanding SOA with Web Services, Addison Wesley 2005

[OMG04] Object Management Group (OMG): Common Object Request Broker Architecture: Core Specification, März 2004, http://www.omg.org/docs/formal/04-03-12.pdf

[Pal01] Michael S. Pallos: Service-Oriented Architecture: A Primer, in: EAI Journal, Dez. 2001

[Rei05] Michael Reiter: Hersteller streiten um Enterprise Service Bus, in: Computer Zeitung Nr. 15, 11.04.2005

[Rei06] Michael Reiter: Weg zur Serviceorientierung erfordert Geschäftsverständnis, in: Computer Zeitung Nr. 16, 18.04.2006, S. 13

[RHS05] Jan-Peter Richter, Harald Haller, Peter Schrey: Serviceorientierte Architektur, in: Informatik-Spektrum, Band 28, Heft 5, Oktober 2005, S. 413-416

[Sim05] Scott Simmons: Introducing the WebSphere Integration Reference Architecture, in: IBM WebSphere Developer Technical Journal, 17.08.2005, http://www-128.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html

[Smi06] David Mitchell Smith: Check Your Destination: Advances Web Services Lead to Enterprise 2.0, Gartner Research, 14.04.2006

[SN96a] Roy W. Schulte, Yefim V. Natis: “Service-Oriented” Architectures, Part 1, Gartner Research, 12.04.1996

[SN96b] Roy W. Schulte, Yefim V. Natis: Middleware for Service-Oriented Architectures, Gartner Research, 12.04.1996

[Sun03] Sun Microsystems: Java Remote Method Invocation, 2003, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html

[SW04] David Sprott, Lawrence Wilkes: Understanding Service Oriented Architecture, in: Journal1 – Microsoft Architects Journal, Jan. 2004, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/aj1soa.asp; Zugriff am 12.01.2006

[TS03] Andrew Tanenbaum, Marten van Steen: Verteilte Systeme, Prentice Hall 2003

[Tie05] Marcus Tiedemann: Handling subordinate services in a service-oriented architecture, in: Althoff u.a. (Hrsg.): WM2005: Professional Knowledge Management: Experiences and Visions, Kaiserslautern 2005

[Tie06] Marcus Tiedemann: Die Identifikation geeigneter Technologien für Integrations-Szenarien auf Basis des Grades der losen Kopplung, in: Lehner, Nösekabel, Kleinschmidt (Hrsg.): Multikonferenz Wirtschaftsinformatik 2006, GITO-Verlag (Berlin) 2006

[Til03] Stefan Tilkow: Web Services-Stil: Dokument- vs. RPC-orientiert, 01.03.2003, http://www.innoq.com/blog/st/2003/03/web_servicesstile_dokument_vs_rpcorientiert.html, Zugriff am 02.03.2006

[TWR04] Marcus Tiedemann, Volker Wittig, Stefan Rougier: Die Gestaltung einer unternehmensweiten, flexiblen Software-Architektur auf Basis eines selbst entwickelten Zonen- und Schichten-Modells, in: Schelp, Winter (Hrsg.): Auf dem Weg zur Integration Factory: Proceedings der DW2004 – Data Warehousing und EAI, Heidelberg 2005

[Vat04] Radu Vatav: CORBA (Überblick, IDL), Erlangen 2004, http://www4.informatik.uni-erlangen.de/Lehre/WS04/PS_KVBK/talks/CORBA-ausarbeitung.pdf, Zugriff am 09.03.2006

[W3C04] W3C: Web Services Glossary, W3C Working Group Note 11 February 2004, http://www.w3.org/TR/ws-gloss/, Zugriff am 12.01.2006

[Wac99] Dieter Wackerow: MQSeries Primer, 1999, http://www.redbooks.ibm.com/redpapers/pdfs/redp0021.pdf

[WB05] Markus Weyerhäuser, Stephan Becker: Enterprise SOA, in: Javaspektrum 04/2005, S. 11-15

[WH04] Steve Wilkes, John Harby (The Middleware Company): SOA Blueprints Concept, Draft v0.5, Jun. 2004, www.oasis-open.org/committees/download.php/14566/SOA_Blueprints_Concepts_v0_5.doc, Zugriff am 13.01.2006

[Wik06] Wikipedia: Komponente, http://de.wikipedia.org/wiki/Komponente, Zugriff am 15.03.2006

[Wis06] Wissen Media Verlag: Komponente, http://www.wissen.de/wde/generator/wissen/ressorts/technik/index,page=1167780.html, Zugriff am 16.03.2006

[Yuh05] Noel Yuhanna: Database Replication: A Flexible Technology for Data Integration Tasks, in: Business Integration Journal, Sept. 2005