2

2 Grundlagen

In diesem Kapitel sind grundlegende Integrationskonzepte beschrieben. Es soll als Einführung in die Integrationstechnologie und -terminologie dienen.

Kapitel 2.1 beschreibt die im Zusammenhang mit Integrationsarchitekturen oft verwendeten grundlegenden Begriffe.

In Kapitel 2.2 werden Architekturvarianten wie Point to Point, Hub and Spoke, Pipeline und SOA skizziert.

Kapitel 2.3, Service-Oriented Integration, zeigt sowohl die Prozess- als auch die Workflow-Integration-Pattern.

In Kapitel 2.4 erläutert die verschiedenen Arten der Datenintegration und deren Pattern.

Kapitel 2.5 EAI / EII zeigt auf, wie mit Direct Connections-, Broker- und Router-Pattern umzugehen ist.

Kapitel 2.6, Event Driven Architecture, zeigt die Weiterentwicklung von SOA durch die Einführung unternehmensweiter Events.

In Kapitel 2.7, GRID/XTP, werden die Integrationstechnologien der Zukunft erläutert.

2.1 Einleitung

Der Begriff „Integration" hat viele Bedeutungen. Zum Handwerkszeug eines Integrationsarchitekten gehört nun mal eine grundlegende Kenntnis der Begriffe und Konzepte. Alleine die Klassifizierung verschiedener Integrationsarten kennt viele Ausprägungen. Aus Sicht des Gesamtunternehmens wird zwischen Application to Application (A2A), Business to Business (B2B) und Business to Consumer (B2C) Integration unterschieden. Portal-, Funktions- und Daten-Integration erlauben eine Klassierung aufgrund des Tiers. Eine weitere mögliche Klassifizierung ist die Integration aufgrund der Semantik.

Grundlegende Begriffe der Integration sind EAI, ESB, Middleware und Messaging. Sie bestimmten das Thema vor der Etablierung einer SOA und sind auch heute noch die Basis vieler Realisierungen. Enterprise Application Integration (EAI) ist eigentlich ein Synonym für Integration. In der ursprünglichen Definition von Linthicum bedeutet EAI, dass Daten und Geschäftsprozesse zwischen den durch EIA verbundenen Anwendungen uneingeschränkt geteilt werden können. Die technologische Realisierung von EAI-Systemen erfolgte in dem meisten Fällen durch so genannte Middleware. Prominente Basistechnologie von EAI ist das Messaging mit der Möglichkeit, über asynchrone Kommunikation mittels Meldungen, die über eine verteilte Infrastruktur und einen zentralen Message Broker ausgetauscht werden, eine Integrations-architektur zu realisieren.

Die grundlegenden Integrations-Architekturvarianten sind die Point-to-Point-Architektur, die Hub-and-Spoke-Architektur, die Pipeline-Architektur und die Service-Orientierte Architektur. Eine Point-to-Point-Architektur stellt einen Verbund unabhängiger Systeme dar, die durch ein Netzwerk miteinander verbunden sind. Die Hub-and-Spoke-Architekturform ist ein weiterer Schritt in der Evolution der Anwendungs- resp. Systemintegration. Ein zentraler Hub übernimmt die Rolle der Kommunikationsdrehscheibe. In der Pipeline-Architektur werden unabhängige Systeme entlang der Wertschöpfungskette mithilfe eines Nachrichten-Busses integriert. Die Bus-Fähigkeit resultiert daraus, dass die Schnittstellen zum zentralen Bus über das Kommunikationsnetzwerk verteilt installiert werden können und die Applikationen somit einen lokalen Zugang zur Bus-Schnittstelle erhalten. Die Integration verschiedener Anwendungen wird zu einem funktionierenden Ganzen als Integration verteilter und unabhängiger Serviceaufrufe, die über einen ESB und ggf. eine Process Engine orchestriert werden.

Zur grundlegenden Technik der Integration gehört der Einsatz von Patterns. Ob das nun die Process Integration und Workflow Integration-Pattern einer Service-Oriented Integration, die Federation-, Population- und Synchronisation-Pattern einer Datenintegration oder die Direct Connections-, Broker- und Router-Pattern von EAI und EII sind, ist unerheblich. Sie alle müssen bekannt sein, damit sie richtig eingesetzt werden können.

Die neuesten Integrationsarchitekturen basieren auf Konzepten wie der Event Driven Architecture, GRID oder XTP. Diese Technologien harren noch der Bewährungsprobe durch die Praxis. Ihre Ansätze sind jedoch vielversprechend und für eine Vielzahl von Anwendungen vor allem für Konzerne und große Organisationen sehr interessant.

2.2 Begriffe

Der Trivadis Integration Blueprint verwendet eine klare und einfache Namensgebung der einzelnen Layer. Im Kontext des Begriffs „Integration“ trifft man jedoch

auf eine Vielzahl unterschiedlicher Definitionen und Begriffe, die wir in diesem Kapitel erklären.

- *A2A, B2B und B2C:* Application to Application (A2A) ist die Integration von Anwendungen und Systemen untereinander, während Business to Business (B2B) die externe Integration von Geschäftspartner-, Kunden- und Lieferanten-Prozessen resp. Anwendungen bedeutet und Business to Consumer (B2C) die direkte Integration von Endkunden zum Beispiel über Internettechnologien in den internen Unternehmensprozess umfasst.
- *Integrationstypen:* Die Typisierung von Integrationen in Integrations-Portale, Shared Data Integration, Shared Function Integration ist üblich. Portale integrieren Anwendungen auf User Interface-Ebene, während Shared Data Integration auf Datenebene und Shared Function Integration auf Funktionsebene Integrationsarchitekturen realisieren.
- *Semantische Integration:* Ein semantischer Integrationsansatz ist zum Beispiel die Verwendung von modellbasierten semantischen Repositories zur Integration von Daten aus verschiedenen Kontextinformationen.
- *EAI:* Enterprise Application Integration als Möglichkeit, Daten und Geschäftsprozesse zwischen den durch EIA verbundenen Anwendungen uneingeschränkt zu teilen.
- *Messaging, Publish-Subscribe, Message Broker und Messaging-Infrastruktur:* Die Mechanismen zur Integration über asynchrone Kommunikation über Meldungen, die über eine verteilte Infrastruktur und einen zentralen Message Broker verteilt werden.
- *ESB:* Ein Enterprise Service Bus als Integrations-Infrastruktur, die für die Umsetzung einer EAI eingesetzt wird. Die Rolle des Enterprise Service Bus (ESB) liegt in der Entkopplung der Client-Anwendungen von den Services.
- *Middleware:* Die technologische Realisierung von EAI-Systemen erfolgt in den meisten Fällen durch so genannte Middleware. Middleware wird auch als Kommunikations-Infrastruktur bezeichnet.
- *Routing Schemas:* Informationen können in einem Netz unterschiedlich weitergeleitet werden. Ja nach Typus der Weiterleitung lassen sich Routing Schemas in Unicast (1:1 Beziehung), Broadcast (Alle Ziele), Multicast (1:N) und Anycast (1:N – best reachable) unterteilen.

2.2.1 A2A, B2B und B2C

Die betrieblichen Informationssysteme der meisten Unternehmen bestehen heute aus einer historisch gewachsenen Anwendungs- und Systemlandschaft. Der zunehmende Einsatz von Standardsoftware (packaged applications) bedeutet auch, dass so genannte „Silos“ weiterhin bestehen werden. Die IT soll jedoch die betrieblichen Prozesse durchgehend unterstützen. Diese Unterstützung kann und darf

nicht vor den Systemgrenzen bestehender oder neuer Applikationen haltmachen. Aus diesem Grunde sind Integrationsmechanismen notwendig. Sie kombinieren einzelne Insellösungen zu einem funktionierenden Ganzen. Und dies nicht nur auf der Ebene einer einzelnen Unternehmung oder einzelnen Organisation, sondern auch zwischen verschiedenen Unternehmen und zwischen dem Unternehmen und seinen Kunden. Auf Organisationsebene wird zwischen A2A-, B2B- und B2C-Integration unterschieden (Pape 2006). Diese Unterscheidung ist in Abbildung 2.1 dargestellt. Jede Integrationsart stellt spezielle Anforderungen an die Methoden, Techniken, Produkte und Tools, die zur Lösung von Integrationsaufgaben eingesetzt werden sollen. So sind beispielsweise die Sicherheitsanforderungen von B2B und B2C-Integration andere als diejenigen einer A2A-Integration.

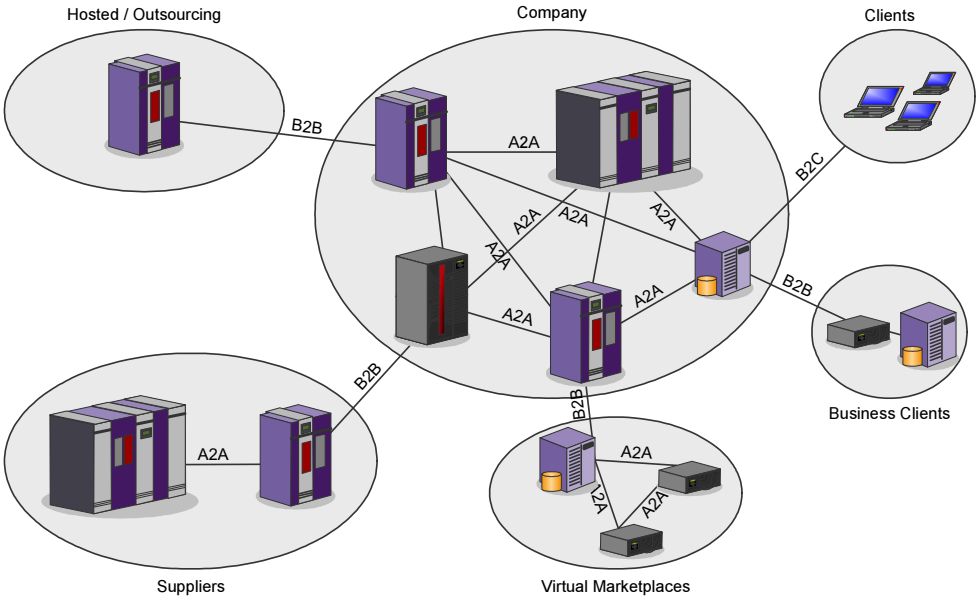

Abbildung 2.1 Übersicht der Integrationsarten (nach (Sailer 2001))

- *A2A (Application to Application):* Die Integration von Anwendungen und Systemen untereinander
- *B2B (Business to Business):* Die externe Integration von Geschäftspartner-, Kunden- und Lieferanten-Prozessen resp. -Anwendungen
- *B2C (Business to Consumer):* Die direkte Integration von Endkunden z.B. über Internettechnologien in den internen Unternehmensprozess.

Durch eine Kombination der verschiedenen Integrationsbegriffe lassen moderne Konzepte wie beispielsweise „Extended Enterprise" – Zusammenschluss über Unternehmensgrenzen hinweg (Konsynski 1993) – oder auch „Virtual Enterprise" beschreiben (Hardwick, Bolton 1997).

2.2.2 Integrationstypen

Die Typisierung von Integrationen in Integrationsportale, Shared Data Integration, Shared Function Integration ist üblich. Portale integrieren Anwendungen auf User-Interface-Ebene, während Shared Data Integration auf Datenebene und Shared Function Integration auf Funktionsebene Integrationsarchitekturen realisieren.

2.2.2.1 Informationsportale (Information Portals)

Viele geschäftliche Nutzer müssen Zugriff auf diverse Systeme haben, um ihre Geschäftsprozesse durchführen zu können. Sei es, um bestimmte Fragen beantworten zu können (man denke an ein Call-Center, das Kundenanrufe entgegennimmt und dazu aktuelle Kundendaten abfragen können muss) oder um bestimmte Geschäftsfunktionen anstoßen oder durchführen zu können (z.B. die Kundendaten zu aktualisieren). Oftmals muss ein Mitarbeiter hierbei auf mehrere Geschäftssysteme gleichzeitig zugreifen können. So kann man sich vorstellen, dass ein Mitarbeiter, um den Status einer Kundenbestellung verifizieren zu können, Zugriff auf das Bestellsystem (auf einem Host) und gleichzeitig auf ein Web-basiertes Bestellsystem mit den Eingaben eines Kunden haben muss. Informationsportale aggregieren Informationen aus multiplen Quellen in einem einzelnen Display, um dem Benutzer mehrfache Systemzugriffe (mit evtl. mehrfachen Authentisierungen) zu ersparen und seine Tätigkeit zu optimieren (Kirchhof et al. 2003). Einfache Informationsportale unterteilen den Benutzerbildschirm in einzelne Bereiche, wobei in jedem die Daten eines der angesprochenen Backendsysteme unabhängig und ohne Interaktionsmöglichkeit zu den anderen angezeigt werden. Anspruchsvollere Systeme ermöglichen limitierte Interaktionen zwischen den einzelnen Bereichen. So können Bereiche synchronisiert sein, d.h. wenn der Benutzer in einem Bereich einen Datensatz anwählt, werden die anderen Bereiche aktualisiert. Wiederum andere Portale sind integrationstechnologisch so weit fortgeschritten, dass dort die Grenze zwischen Portal-Applikation und integrierter Applikation verschwimmt (Nussdorfer, Martin 2006).

2.2.2.2 Gemeinsame Daten (Shared Data)

Shared Database, File Replication und Data Transfer fallen unter die Kategorie der Integration über gemeinsame Daten (Gorton 2006).

- *Gemeinsame Datenbank (Shared Database):* Viele unterschiedliche Geschäftssysteme erfordern den Zugriff auf gemeinsame Daten. So kann eine Kundenadresse in einem Bestellsystem, einem CRM-System und einem Vertriebs-System notwendig sein. Solcherlei Daten können in einer gemeinsamen Datenbank vorgehalten werden, um Redundanzen und Synchronisationsprobleme zu vermeiden.

- *Datenreplikation (File Replication):* Oftmals verfügen dabei die Systeme über jeweils ihre eigenen, lokalen Datenspeicher. Dann müssen die evtl. zentral verwalteten Daten (in einem führenden System) in die jeweiligen Zieldatenbanken repliziert und erforderliche Updates und Synchronisationen regelmäßig nachgezogen werden.
- *Datei-Transfer (Data Transfer):* Data Transfer ist der Spezialfall der Datenreplikation, bei dem der Datentransfer dateibasiert erfolgt.

2.2.2.3 Gemeinsame Funktionen (Shared Business Functions)

In der gleichen Weise, wie unterschiedliche Geschäftssysteme redundant Daten vorhalten, tendieren sie zur Implementierung redundanter Geschäftslogik, was die Wartung und Anpassung an neue Gegebenheiten erschwert und nicht unerhebliche Kosten verursachen kann. Beispielsweise müssen verschiedene Systeme Daten nach vordefinierten, zentral verwalteten Geschäftsregeln validieren können. Es ist sinnvoll, solche Logik zentral zu pflegen und als gemeinsame Logik zur Realisierung der Geschäftsfunktionalität als Service für andere Systeme exportieren zu können.

- *EAI:* mit EAI bezeichnet man im Allgemeinen alle Techniken, die versuchen, eine Kopplung zwischen den unterschiedlichen beteiligten Systemen zu vereinfachen, um so eine Spagetti-Architektur durch unkontrollierten Einsatz von proprietären Point-to-Point-Verbindungen zu verhindern. Eine Kopplung zwischen den Systemen erfolgt mit EAI-Lösungen statt über eine einzelnen, Hersteller-abhängige, proprietäre Schnittstellen-API.
- *SOA:* Mit „Service-Orientierter Architektur" beschreibt man den Realisierungsstil einer Unternehmensarchitektur. SOA soll ein Geschäft zur Identifizierung von Geschäftsfeldern und Geschäftsprozessen fachlich analysieren und technisch unterstützend strukturieren. Daraus werden Dienste definiert, die die Funktionalität einzelner Geschäftsfunktionalitäten implementieren. Technische Dienste entsprechen in einer SOA den fachlichen Geschäftsfeldern oder Geschäftsfunktionalitäten in Geschäftsprozessen. Dies ist ein großer konzeptioneller Unterschied zu klassischen EAI-Lösungen, die diesen Fokus nicht haben. Deren Sichtweise dient dem reinen Datenaustausch zwischen Systemen, unabhängig von der fachlichen Semantik und unabhängig jedweder fachlicher Prozessbetrachtung.

2.2.2.4 Unterschiede zwischen EAI und SOA

EAI-Lösungen könnten vielfach die in sie gesetzten Erwartungen nur bedingt erfüllen bzw. nur unbefriedigend lösen, was unter anderem auf folgenden Beobachtungen beruht (Rotem-Gal-Oz 2007):

- EAI-Lösungen sind üblicherweise Daten-fokussiert und nicht Prozess-fokussiert.

- EAI-Lösungen adressieren keinen Geschäftsprozess-unterstützenden Prozess, sondern werden unabhängig definiert.
- EAI-Lösungen sind sehr komplex und verhindern durch proprietäre Technologien einen längerfristigen Investitionsschutz, wie er unter Verwendung offener Standards möglich ist.
- EAI-Lösungen benötigen produktspezifisches Wissen, das nur im EAI-Kontext von Belang ist und sonst in Projekten nicht weiter verwendbar ist.
- EAI-Lösungen werden auf Dauer kaum günstiger zu betreiben sein als die schon angesprochenen, hausgemachten Spaghetti-Architekturen.

Setzt man EAI-Lösungen in Kombination mit Web-Services zur Kopplung von Systemen ein, so ist dies noch nicht mit einer SOA vergleichbar. Zwar reduziert sich hierdurch die proprietäre Verbindungskomponente zwischen den zu koppelnden Systemen durch die Verwendung offener WS*-Standards, doch handelt es sich bei einer „richtigen" SOA um einen weitergehenden Architekturansatz, basierend auf einer (Geschäfts-)Prozess-fokussierten Betrachtungsweise der Integrationsproblematik.

Während EAI datengetrieben ist und den Schwerpunkt auf die Applikationsschnittstellen-Integration legt, handelt es sich bei SOA um ein Geschäftsprozess-getriebenes Konzept, das seinen Schwerpunkt auf die Integration von Serviceschnittstellen unter Einhaltung offener Standards legt und die Unterschiede in den einzelnen Integrationsansätzen kapselt und so die Barriere zwischen Daten- und Applikationsintegrations-Ansätzen aufhebt. SOA hinterlässt zurzeit aber noch eine signifikante Problematik, nämlich die der semantischen Integration. Heutige Web-Services geben hierzu keine befriedigende Antwort. Allerdings erlauben sie es uns wenigstens, die richtigen Fragen zu formulieren, um künftige Lösungen zu finden.

2.2.3 Semantische Integration und die Rolle der Daten

Die Herausforderung der semantischen Integration basiert auf folgendem Problem:

- Die Repräsentation der Daten und die Information der Daten selbst sind oftmals eng miteinander verknüpft und nicht getrennt in Nutzdaten und Metadaten.
- Die Information leidet darunter, dass der Datenkontext – die Metainformation, in welchem Bezugsrahmen die Daten zu interpretieren sind, fehlt.

D.h., dass die Datenstruktur und Dateninformation (ihre Bedeutung) oftmals nicht das Gleiche sind und somit interpretiert werden müssen (Inmon, Nesavich 2008).

Folgendes Beispiel soll dies verdeutlichen:

> Ein Datum wie z.B. „7. August 1973" liegt vor. Hieraus ist nicht ersichtlich, ob diese Information im System als Textstring oder aber in einem Datums-

format vorliegt. Evtl. liegt es in einem anderen Format vor und muss zur Laufzeit auf Basis von Referenzdaten kalkuliert werden. Diese Information ist für den Benutzer auch unerheblich.

Allerdings kann es wichtig sein, um was es sich bei diesem Datum handelt, also seine semantische Bedeutung im Bezugskontext. Handelt es sich um den Geburtstag eines Kunden oder aber um den Zeitpunkt der Erstellung des Datensatzes? Das Beispiel lässt sich durchaus weiter verkomplizieren. Ein anderes Beispiel, das in unterschiedlichem Kontext unterschiedlich zu interpretieren ist, stellt z.B. der Begriff „Caesar" dar.

Je nach Kontext könnte es sich um den Namen eines Haustieres oder eines Tierfutters handeln, evtl. aber auch um einen bekannten Salat oder gar um ein Spielcasino oder den Namen eines römischen Kaisers.

Es wird deutlich, dass Daten ohne jeweiligen Bezugsrahmen, jedwede semantische Information verlieren. Ontologisch orientierte oder auch adaptive Schnittstellen können helfen, semantische Bezüge herzustellen, und werden künftig im Umfeld von autonomen Business-to-Business- oder Business-to-Consumer-Marktplätzen an Bedeutung gewinnen.

Ein semantischer Integrationsansatz kann zum Beispiel die Verwendung von modellbasierten semantischen Repositories sein (Casanave 2007). Hierin werden Implementierungs- und Integrationsentwürfe für Applikationen/Prozesse vorgehalten und gepflegt (Yuan et al. 2006). Dabei wird auf bestehende Vokabularien und Referenzmodelle zurückgegriffen, die eine standardisierte Modellierung erlauben. Vokabularien sorgen für eine semantische Kopplung zwischen Daten und spezifischen fachlichen Prozessen, wodurch beteiligte Dienste und Applikationen durch das Vokabular mit semantischen Informationen im umgebenden fachlichen Kontext versorgt werden. Die Pflege von Glossars und Vokabularien zur Schaffung einer gemeinsamen Sprache und somit eines gemeinsamen Verständnisses aller beteiligten Systeme und Partner muss als primäres Ziel künftiger Architekturentwürfe angesehen werden. Semantische Brüche sind so weit wie irgend möglich zu vermeiden bzw. zu überbrücken (z.B. durch geeignete Transformationen von Eingabe- und Ausgabedaten und die Verwendung kanonischer Modelle und einheitlicher Geschäftsdokumenten-Formate. Die beiden letzteren können für unterschiedliche Branchen als Referenzmodelle vordefiniert werden (EDI (FIPS 1993), RosettaNet (Damodaran 2004), etc.). Transformationsregeln können auf Basis der Referenzmodelle in Form von Datenkarten und Transformations-Karten generiert und vorgehalten werden. Die Zukunft wird gerade im Rahmen von Integrationsarchitekturen einen verstärkten Aufwand in der deklarativen Beschreibung (Was?) und weniger in der Beschreibung der konkreten Softwarelogik (Wie? Programmcode) erleben, d.h. der Aufwand in Integrationsprojekten verlagert sich weg von einer Implementierung hin zu einer konzeptionellen Beschreibung im Sinne eines generativen Ansatzes, in dem der notwendige Laufzeitcode automatisch erzeugt wird.

2.2.4 Enterprise Application Integration (EAI)

Mit der Zunahme der Bedeutung der Integration und damit verbunden auch verbreiterter Integrationsvorhaben kam der Begriff Enterprise Application Integration (EAI) auf. EAI stellt kein Produkt oder spezifisches Integrationsframework dar, sondern lässt sich als Kombination von Prozessen, Software, Standards und Hardware, welche eine durchgängige Integration von mehreren Enterprise-Systemen ermöglicht und diese als System erscheinen lassen, definieren (Lam, Shankararaman 2007).

EAI – Definition
EAI bedeutet, Daten und Geschäftsprozesse zwischen den durch EIA verbundenen Anwendungen uneingeschränkt zu teilen (Linthicum 2000).

Aus der Geschäftsperspektive betrachtet, wird EAI als Wettbewerbsvorteil bezeichnet, den ein Unternehmen erhält, wenn alle Anwendungen zu einem einheitlichen Informationssystem integriert werden. Aus der technischen Perspektive betrachtet, stellt EAI einen Prozess dar, um heterogene Anwendungen, Funktionen und Daten zu integrieren, um die gemeinsame Nutzung von Daten und die Integration von Geschäftsprozessen über alle Anwendungen hinweg zu ermöglichen. Diese Integration soll hierbei ohne große Änderungen der existierenden Anwendungen und Datenbanken durch effiziente Methoden in Bezug auf Kosten und Zeit erreicht werden.

Der Schwerpunkt von EAI liegt in erster Linie auf der technischen Integration einer Anwendungs- und Systemlandschaft. Als Werkzeuge zur Integration kommen entsprechende Middleware-Produkte zum Einsatz. Dabei werden die Anwendungen möglichst so belassen, wie sie konzipiert und realisiert wurden. Mittels Adapter können Informationen und Daten über die technologisch heterogenen Strukturen und Grenzen hinweg bewegt werden. Der Service-Gedanke fehlt hier ebenso wie der Anspruch, durch offene Standards die Komplexität abzubauen und Redundanzen zu vermeiden. Dies erfolgte erst später mit dem Aufkommen von Service-Orientierten Architekturen (SOA), wo die Fokussierung auf die fachliche Ebene in einem Unternehmen und dessen Geschäftsprozesse wichtig wurde.

EAI-unterstützende Softwareprodukte sind heute oftmals in der Lage, innerhalb einer SOA die technische Basis für Infrastrukturkomponenten bereitzustellen. Da sie auch die relevanten Schnittstellen einer SOA unterstützen und deren Sprache verstehen, können sie als steuernde Instanz für eine Orchestrierung eingesetzt werden und helfen, heterogene Teilsysteme zu einem Ganzen zusammenzufügen. EAI ist somit je nach strategischer Definition eine Vorstufe von SOA oder auch ein mit SOA konkurrierendes Konzept.

Heute treibt SOA den Integrationsgedanken in eine neue Dimension. Neben der auch mit SOA wichtigen, klassischen „horizontalen" Integration, das heißt der Integration von Anwendungen und Systemen im Sinne einer EAI, wird eine „vertika-

le" Integration der Abbildung der Geschäftsprozesse auf IT-Ebene sowie, umgekehrt, durch SOA verstärkt propagiert (Fröschle, Reinheimer 2007).

Service-Orientierte Architekturen prägen schon heute die Applikationslandschaft. Wenn heutzutage neue Lösungen realisiert werden, ist man gut beraten, diese SOA-tauglich zu konzipieren, selbst wenn die Einführung einer Integrationsarchitektur oder einer Orchestrierungsschicht nicht unmittelbar bevorsteht. Der Übergang zu einer Service-Orientierten Architektur kann auf diese Weise in kleinen, kontrollierbaren Schritten parallel zur bereits im Einsatz befindlichen Architektur und auf der bestehenden Integrationsinfrastruktur aufbauend vollzogen werden.

2.2.4.1 Integrationsebenen

Integrationsarchitekturen gehen von mindestens drei oder vier Integrationsebenen aus: der Integration auf der Kommunikationsebene, der Integration auf der Datenebene, der Integration auf der Objektebene und der Integration auf der Prozessebene (nach (Puschmann, Alt 2004) und (Ring, Ward-Dutton 1999)).

- *Integration auf Datenebene:* Zwischen verschiedenen Systemen werden die Daten ausgetauscht. Die sicherlich am häufigsten in Unternehmen eingesetzte Technologie zur Integration auf Datenebene ist FTP (File Transfer Protocol). Eine andere verbreitete Form ist die direkte Verbindung zweier Datenbanken, die im Falle von Oracle Daten z.B. über Database Links oder aber External Tables austauschen.
- *Integration auf Objektebene:* Die Integration auf Objektebene baut auf der Datenebene auf. Sie erlaubt eine Kommunikation zwischen Systemen über den Aufruf von Objekten der beteiligten Anwendungen.
- *Integration auf Prozessebene:* Die Integration auf Prozessebene erfolgt mittels Workflow Management-Systemen. Die Kommunikation zwischen verschiedenen Anwendungen erfolgt in dieser Ebene über die Workflows, die einen Geschäftsprozess abbilden.

2.2.5 Messaging

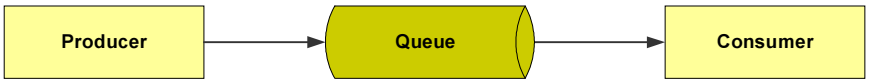

Abbildung 2.2 Messaging

Message Queues sind in den 70er-Jahren des letzten Jahrhunderts als Mechanismus zur Synchronisation zwischen Prozessen eingeführt worden (BrichHanson 1970). Message Queues erlauben persistente Meldungen und damit die Umsetzung der asynchronen Kommunikation und der garantierten Lieferung von Meldungen (**Abbildung** 2.2 Messaging

). Producer und Consumer sind durch Messaging entkoppelt. Der einzige gemeinsame Nenner ist die Queue. Die wichtigsten Eigenschaften des Messaging stellt Tabelle 2.1 dar.

Tabelle 2.1 Qualitätsattribute von Messaging

Attribut	Bemerkung
Availability – Verfügbarkeit	Physikalische Queues mit dem gleichen logischen Namen können über mehrere Serverinstanzen hinweg repliziert werden. Bei Ausfall eines Servers können die Clients die Nachricht an einen anderen senden.
Failure Handling – Fehlerbehandlung	Sollte die Kommunikation eines Clients mit einem Server fehlschlagen, so kann der Client die Nachricht via Failover-Mechanismen an eine andere Serverinstanz leiten.
Modifiability – Veränderbarkeit	Clients und Server sind durch das Nachrichtenkonzept lose miteinander gekoppelt, sprich: kennen einander nicht. Dadurch wird es möglich, dass beide Komponenten hochgradig modifizierbar bleiben, ohne das Gesamtsystem zu beeinflussen. Eine weiterhin bestehende Abhängigkeit von Producer und Consumer ist das Nachrichtenformat. Diese Abhängigkeit kann durch Einführung eines selbst beschreibenden, allgemeingültigen Nachrichtenformates (kanonisches Nachrichtenformat) ebenfalls reduziert oder sogar vermieden werden.
Performance	Mehrere Tausend Nachrichten pro Sekunde sind keine Seltenheit, abhängig auch von Nachrichtengröße und -Komplexität der notwendigen Transformationen. Großen Einfluss auf die Gesamtperformance hat auch die QoS. Non-Reliable Messaging ist durch die fehlende Zwischensicherung der Nachrichten performanter als Reliable Messaging, bei dem die Nachrichten im Dateisystem oder in Datenbanken (lokal oder remote) abgelegt werden müssen, um im Falle eines Serverausfalls nicht verloren zu gehen.
Scalability – Skalierbarkeit	Durch Replikation und Clustering kann Messaging als hochgradig skalierbare Lösung angesehen werden.

2.2.6 Publish-Subscribe

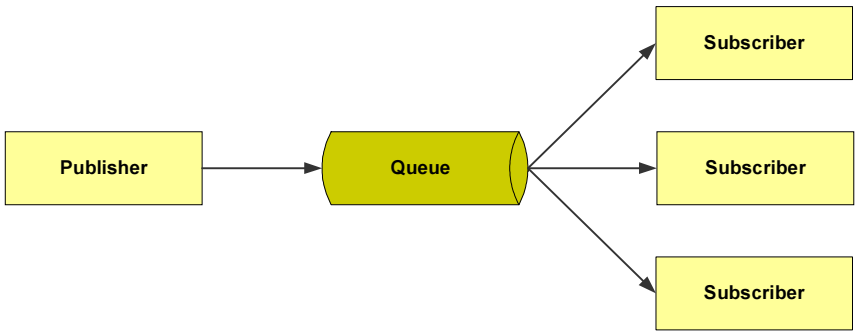

Abbildung 2.3 Publish-Subscribe

Publish-Subscribe stellt eine Evolution des Messaging dar (Quema et al. 2002). Ein Subscriber meldet in geeigneter Form sein Interesse an einer bestimmten Meldung oder einem bestimmten Meldungstyp an. Die persistente Queue sorgt für garantierte und sichere Lieferung. Der Publisher stellt seine Meldung lediglich in die Message Queue. Die Verteilung der Meldung wird von der Queue selbst übernommen. Damit wird ein so genanntes Many-to-Many-Messaging ermöglicht (**Abbildung 2.3**). Die wichtigsten Eigenschaften von Publish-Subscribe sind in Tabelle 2.2 dargestellt.

Tabelle 2.2 Qualitätsattribute von Publish-Subscribe

Attribut	Bemerkung
Availability – Verfügbarkeit	Physikalische Topics mit dem gleichen logischen Namen können über mehrere Serverinstanzen hinweg repliziert werden. Bei Ausfall eines Servers können die Clients die Nachricht an einen anderen senden.
Failure Handling – Fehlerbehandlung	Bei Ausfall eines Servers können die Clients die Nachricht an einen anderen replizierten Server senden.
Modifiability – Veränderbarkeit	Publisher und Subscriber sind durch das Nachrichtenkonzept lose miteinander gekoppelt, sprich: sie kennen einander nicht. Dadurch wird es möglich, dass beide Komponenten hochgradig modifizierbar bleiben, ohne das Gesamtsystem zu beeinflussen. Eine weiterhin bestehende Abhängigkeit besteht durch das Nachrichtenformat. Diese Abhängigkeit kann durch Einführung eines selbst beschreibenden, allgemeingültigen Nachrichtenformates (kanonisches Nachrichtenformat) ebenfalls reduziert oder sogar vermieden werden.
Performance	Publish-Subscribe kann Tausende von Nachrichten pro Sekunde verarbeiten, wobei die nicht-zuverlässige (non-reliable) Nachrichtenübertragung schneller ist als die zuverlässige (reliable), da bei Letzterem Nachrichten lokal zwischengespeichert

	werden müssen. Unterstützt ein Publish-Subscribe Broker Multicast/Broadcast-Protokolle, so können mehrere Nachrichten in zeitgleichen Zeiteinheiten an die Subscriber übermittelt werden, jedoch nicht seriell nacheinander.
Scalability – Skalierbarkeit	Topics können über Server-Cluster hinweg repliziert werden. Dies skaliert auch für sehr große Durchsätze an Nachrichten. Auch hier gilt, dass Mutlicast/Broadcast-Protokolle besser skalieren als Point-to-Point-Protokolle.

2.2.7 Message Broker

Ein Message Broker ist eine zentrale Komponente, die für die sichere Lieferung von Meldungen zuständig ist (Linthicum 1999). Der Broker verfügt über logische Ports für den Empfang respektive das Versenden von Meldungen. Er transportiert und transformiert ggf. Meldungen zwischen Sender und Subscriber.

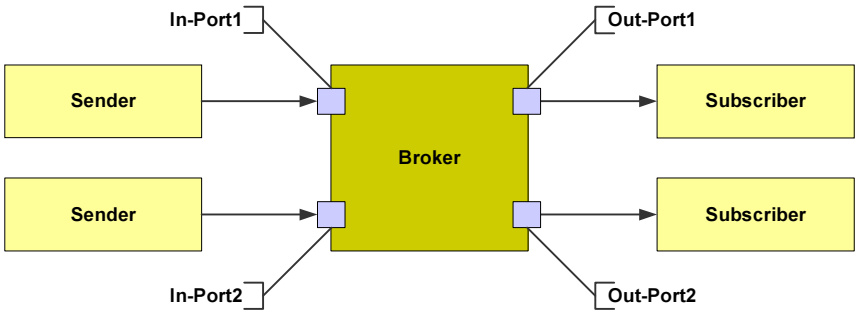

Abbildung 2.4 Message Broker

Die wichtigsten Aufgaben eines Message Broker, wie in **Abbildung 2.4** dargestellt, sind die Umsetzung einer Hub-And-Spoke-Architektur, das Routing und die Transformation der Messages.

- *Hub-And-Spoke-Architektur:* Der Broker agiert als zentrale Nachrichtenaustauschachse (Hub), bei der Sender und Empfänger quasi als Speichen radial angemeldet sind. Verbindungen zum Broker werden über Adapter-Ports realisiert, die das jeweilige Nachrichtenformat unterstützen.
- *Message Routing:* Der Broker umfasst Prozessierungslogik, um die Nachrichten weiterzuleiten. Die Weiterleitung (Routing) kann hardcodiert erfolgen oder aber deklarativ und basiert dann häufig auf dem Inhalt der Nachricht selbst (content-based routing) oder auf spezifischen Werten/Attributen im Nachrichtenkopf (attribute based routing).
- *Message Transformation:* Die Brokerlogik transformiert das Nachrichteneingangsformat in das notwendige Nachrichtenausgangsformat.

Die wichtigsten Eigenschaften eines Message Broker sind in Tabelle 2.3 dargestellt.

Tabelle 2.3 Qualitätsattribute eines Message Brokers

Attribut	Bemerkung
Availability – Verfügbarkeit	Für Hochverfügbarkeit müssen Broker repliziert und als Cluster betrieben werden.
Failure Handling – Fehlerbehandlung	Broker verfügen über typisierte Eingangs-Ports, welche ankommende Nachrichten auf ihr korrektes Format hin validieren und falsche abweisen. Bei Ausfall eines Brokers können die Clients die Nachricht an einen anderen der replizierten Broker senden.
Modifiability – Veränderbarkeit	Broker separieren Transformationslogik (transformation logic) und Weiterleitungslogik (routing logic) voneinander und von Sender und Empfänger. Hierdurch wird die Modifizierbarkeit verbessert, da diese Logik ohne Einfluss auf Sender und Empfänger bleibt.
Performance	Broker können aufgrund des Hub-and-Spoke-Ansatzes zu einem potenziellen Engpass werden. Dies gilt vor allem bei hohem Nachrichtenvolumen, großen Nachrichten und komplexen Transformationen. Der Durchsatz ist typischerweise geringer als der von einfachem, zuverlässigem Messaging (reliable delivery messaging).
Scalability – Skalierbarkeit	Broker-Cluster erlauben eine hohe Skalierbarkeit.0

2.2.8 Messaging-Infrastruktur

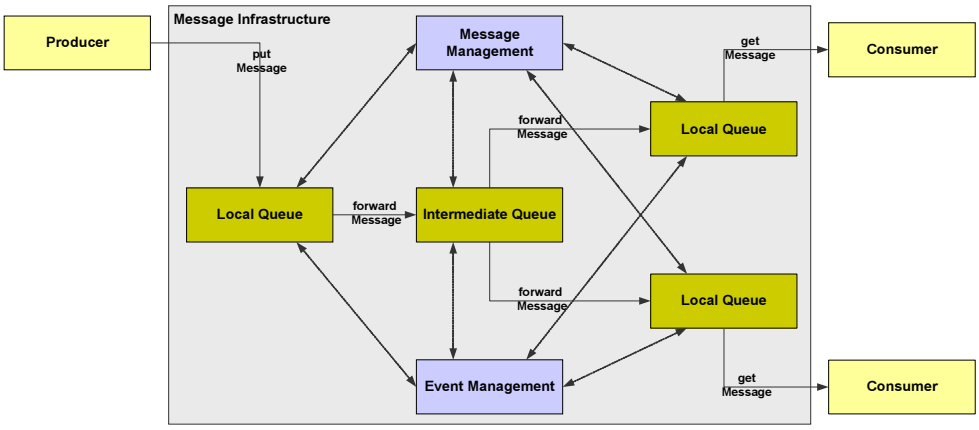

Abbildung 2.5 Aufbau einer Messaging-Infrastruktur

Eine Messaging-Infrastruktur stellt Mechanismen zum Versenden, für das Routing und die Umwandlung von Daten zwischen unterschiedlichen Anwendungen zur

Verfügung, die auf unterschiedlichen Betriebssystemen mit unterschiedlicher Technologie laufen, wie in **Abbildung 2.5** dargestellt (Eugster et al. 2003).

- *Sender:* Anwendung, die Meldungen an eine lokale Queue schickt.
- *Receiver:* Anwendung, die sich für bestimmte Meldungen interessiert.
- *Local Queue:* Lokale Schnittstelle der Messaging-Infrastruktur. Jede an eine Local Queue gesendete Meldung wird von der Infrastruktur entgegengenommen und an einen oder mehrere Receiver weitergeleitet.
- *Intermediate Queue:* Um die Lieferung von Meldungen sicherzustellen, nutzt die Infrastruktur Intermediate Queues, falls eine Meldung nicht geliefert werden kann oder für mehrere Empfänger kopiert werden muss.
- *Message Management:* Versenden, Routing und Umwandlung von Daten sowie spezielle Funktionen wie beispielsweise die garantierte Lieferung, die Kontrolle der Meldungen, das Tracing einzelner Meldungen sowie das Error Management.
- *Event-Management:* Die Steuerung des Subscribe-Mechanismus erfolgt über spezielle Ereignisse.

2.2.9 Enterprise Service Bus (ESB)

Ein Enterprise Service Bus (ESB) ist eine Infrastruktur, die für die Umsetzung einer EAI eingesetzt wird. Die Rolle des Enterprise Service Bus (ESB) liegt primär in der Entkopplung der Client-Anwendungen von den Services, wie in **Abbildung 2.6** dargestellt (Chappell 2004).

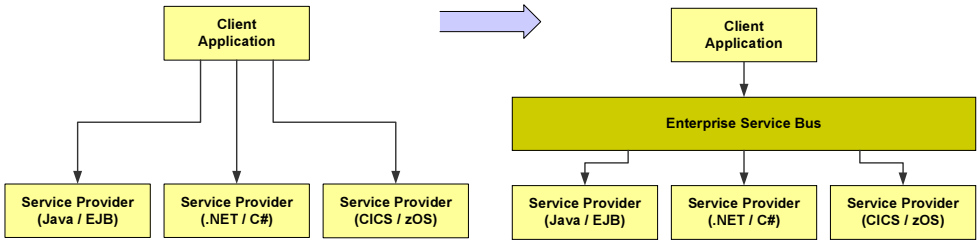

Abbildung 2.6 Entkopplung durch den ESB

Mit der Kapselung durch den ESB braucht sich die Client-Anwendung nicht über die Lokalisierung des Services sowie die verschiedenen Kommunikationsprotokolle für den Aufruf des Services zu kümmern. Der ESB ermöglicht die gemeinsame und unternehmensweite sowie unternehmensübergreifende Nutzung von Services und separiert Geschäftsprozesse von den jeweiligen Service-Implementierungen (Lee et al. 2003).

2.2.9.1 Die Kernfunktionen eines ESB

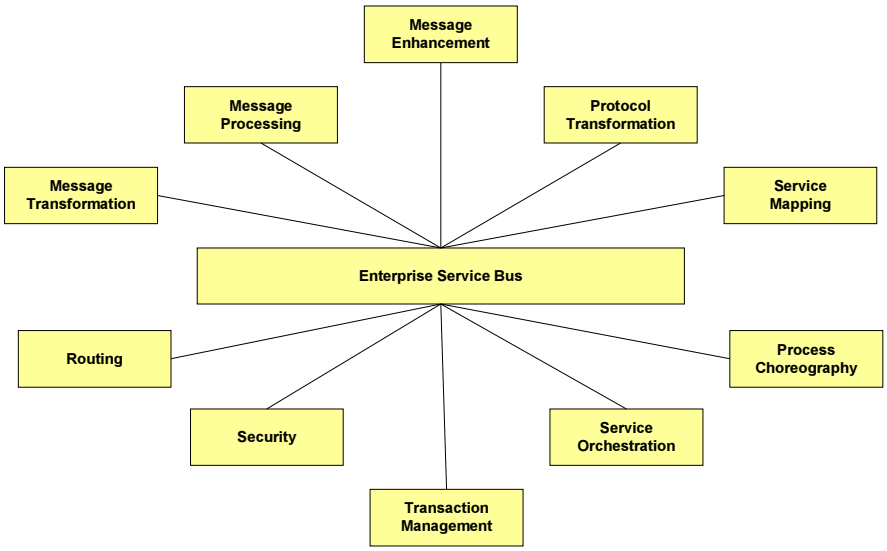

Abbildung 2.7 Kernfunktionen eines ESB

Die wichtigsten Hersteller im SOA-Umfeld bieten heute spezifische Enterprise-Service-Bus-Produkte an, welche eine Reihe von Kernfunktionen wie in **Abbildung 2.7** dargestellt in der einen oder anderen Ausprägung anbieten.

2.2.9.2 Der Aufbau eines ESB

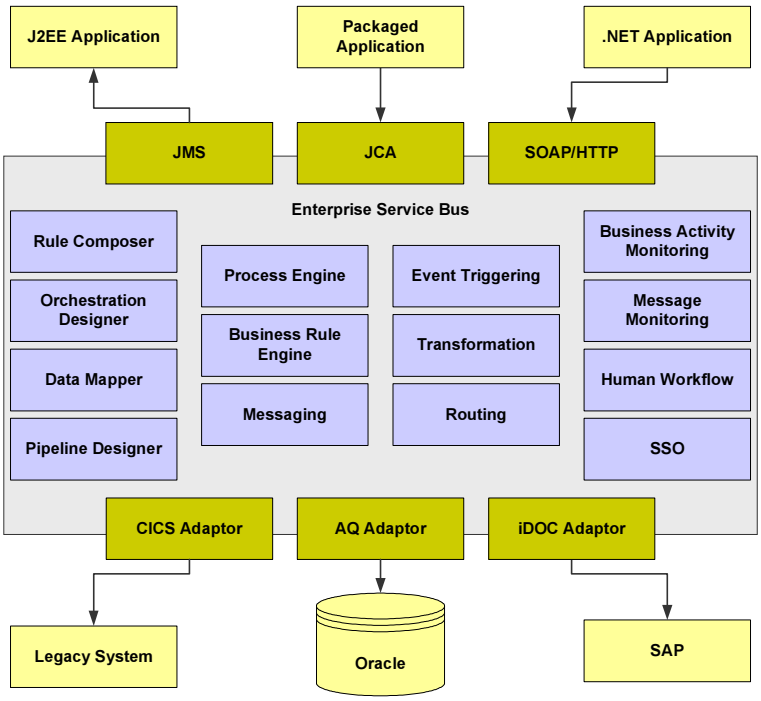

Abbildung 2.8 Beispiel für den Aufbau eines fiktiven ESB

Die Produkte diverser Hersteller variieren zwar bezüglich der in **Abbildung 2.8** dargestellten Namensgebung einzelner Komponenten, stellen jedoch mindestens folgende Funktionen zur Verfügung (Craggs 2003):

- Routing und Messaging als Basisdienste
- Kommunikationsbus, der die Integration verschiedenster Systeme über vorgefertigte Adaptoren erlaubt
- Transformation und Mapping, um verschiedenste Konvertierungen und Transformationen durchzuführen
- Mechanismen zur Ausführung von Prozessen und Regeln
- Überwachungsfunktionen für verschiedenste Komponenten
- Entwicklungstools für die Modellierung von Prozessen, Mappingregeln und Meldungs-Transfers
- Eine Reihe von standardisierten Schnittstellen, wie beispielsweise JMS (Java Messaging Specification (Hapner et al. 2002)), JCA (Java Connector Architecture (JCASpec 2003)) und SOAP/HTTP.

2.2.10 Middleware

Die technologische Realisierung von EAI-Systemen erfolgt in dem meisten Fällen durch so genannte Middleware. Middleware wird auch als Kommunikationsinfrastruktur bezeichnet. Sie erlaubt Kommunikation zwischen Softwarekomponenten, unabhängig von der Programmiersprache, in der die Komponenten realisiert wurden, unabhängig von den zur Kommunikation eingesetzten Protokollen und unabhängig von der Plattform, auf der die beteiligten Komponenten ablaufen (Thomson 1997). Middleware kann entweder aufgrund der Kommunikationsarten oder aufgrund der eingesetzten Basistechnologie unterschieden werden.

2.2.10.1 Middleware Kommunikationsarten

Kommunikationsarten für Middleware werden in 5 Kategorien eingeteilt: Conversational Communication Model (Dialogorientiertes Kommunikationsmodell), dem Request/Reply Communication Model, Message Passing Communication Model, Message Queuing Communication Model und Publish/Subscribe Communication Model.

- *Conversational:* Die beteiligten Komponenten interagieren synchron miteinander. Es wird immer sofort auf die ausgetauschten Informationen reagiert. Diese Art von Kommunikation wird meist in Real-Time-Systemen verwendet.
- *Request/Reply:* wird verwendet, wenn eine Anwendung Funktionen einer anderen Anwendung aufrufen soll; entspricht dem Aufruf einer Subroutine, nur dass die Kommunikation über ein Netzwerk erfolgen kann.
- *Message Passing*: erlaubt einen einfachen und gezielten Austausch von Informationen in Form von Meldungen. Die Kommunikation erfolgt nur in einer Richtung. Will eine Anwendung auf eine eingehende Meldung reagieren, so ist diese Reaktion wiederum in eine neue Meldung zu packen.
- *Message Queuing:* Informationen werden in Form von Meldungen ausgetauscht, die über eine Queue – also indirekt – versendet werden. Queuing erlaubt die sichere, planbare und priorisierbare Lieferung von Meldungen. Es wird oft für den „near-realtime"-Informationsaustausch von lose gekoppelten Systemen eingesetzt.
- *Publish/Subscribe:* Die nicht-gerichtete Kommunikation wird durch zwei Rollen realisiert – der Publisher einer Meldung sendet die Meldung lediglich an die Middleware. Der Subscriber abonniert all jene Meldungstypen, die ihn interessieren. Die Middleware sorgt dafür, dass alle Subscriber die entsprechenden Meldungen eines Publishers erhalten.

Tabelle 2.4 Eigenschaften von Middleware-Typen

Middleware-Typ	Kommunikation	Beteiligung	synchron / asynchron	Interaktion

Peer To Peer, API's	conversational	1:1	synchron	blockierend
Database Gateways	Request/Reply	1:1	synchron	blockierend
Database Replication	Request/Reply / Message Queue	1:n 1:n	synchron asynchron	blockierend nicht-blockierend
Remote Procedure Calls	Request/Reply	1:1	meist synchron	meist blockierend
Object Request Brokers	Request/Reply	1:1	meist synchron	meist blockierend
Direct Messaging	Message Passing	1:1	asynchron	nicht-blockierend
Message Queue Systems	Message Queue	m:n	asynchron	nicht-blockierend
Message Infrastructure	Publish / Subscribe	m:n	asynchron	nicht blockierend

2.2.10.2 Middleware-Basistechnologien

Die Unterscheidung von Middleware anhand der eingesetzten Basistechnologie sieht eine Unterscheidung zwischen Data-Oriented Middleware, Remote Procedure Call, Transaction-Oriented Middleware, Message-Oriented Middleware und Component-Oriented Middleware vor (Jung 2005).

- *Data-Oriented Middleware:* die Integration von Daten respektive die Verteilung von Daten auf verschiedene RDBMS mit den entsprechenden Synchronisationsmechanismen.
- *Remote Procedure Call:* die Umsetzung des klassischen Client-Server-Ansatzes.
- *Transaction-Oriented Middleware:* Das Transaktionskonzept (ACID – Atomicity, Consistency, Isolation, Durability) wird mit dieser Art von Middleware realisiert. Eine Transaktion ist eine endliche Folge atomarer Operationen, die entweder lesend oder schreibend auf einen Datenbestand zugreifen.
- *Message-Oriented Middleware:* Der Informationsaustausch erfolgt über Meldungen, die von der Middleware von einer Anwendung zur nächsten transportiert wird. In den meisten Realisierungen werden Message Queues eingesetzt.
- *Component-Oriented Middleware:* stellt verschiedene Anwendungen respektive deren Komponenten als ein Gesamtsystem dar.

2.2.11 Routing Schemas

Informationen können in einem Netz unterschiedlich weitergeleitet werden. Ja nach Typus der Weiterleitung lassen sich Routing Schemas in Unicast (1:1 Bezie-

hung), Broadcast (Alle Ziele), Multicast (1:N) und Anycast (1:N – best reachable) unterteilen.

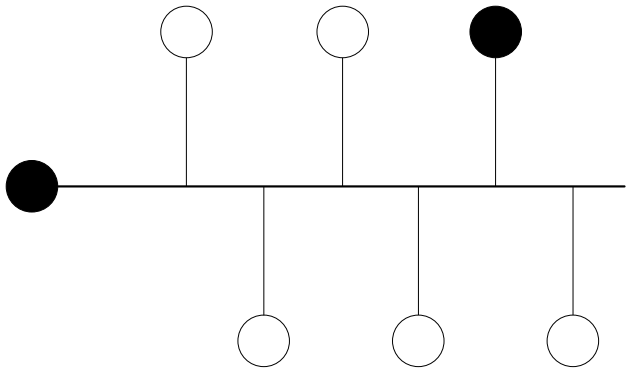

Abbildung 2.9 Uncast

Unicast: Das Unicast Routing Schema sendet Datenpakete zu einem einzelnen Ziel. Es besteht eine 1:1-Beziehung zwischen Netzwerkadresse und Netzwerk-Endpunkt (**Abbildung 2.9**).

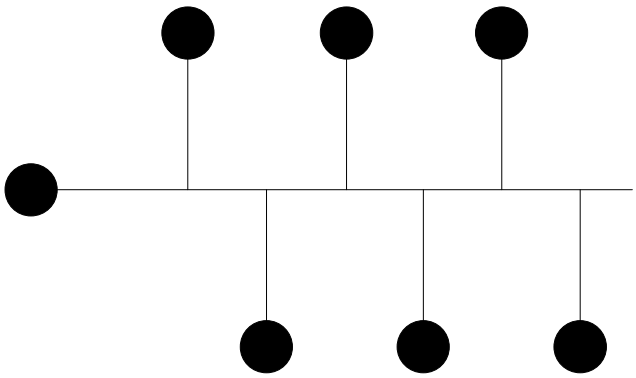

Abbildung 2.10 Broadcast

Broadcast: Das Broadcast Routing Schema sendet Datenpakete parallel an alle möglichen Ziele im Netz. Falls keine Unterstützung für dieses Verfahren vorliegt, so kann durch ein serielles Versenden der Datenpakete an alle möglichen Ziele das gleiche Ergebnis mit verminderter Performance erreicht werden. Es besteht eine 1:N-Beziehung zwischen Netzwerkaddresse und Netzwerk-Endpunkt (**Abbildung 2.10**).

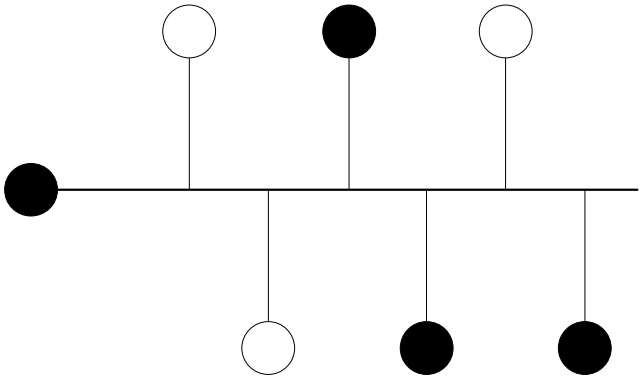

Abbildung 2.11 Multicast

Multicast: Das Multicasting Routing Schema sendet Datenpackete an eine definierte Auswahl von Zielen. Die Zielmenge ist ein Untermenge aller möglichen Ziele. Es besteht eine 1:N-Beziehung zwischen Netzwerkadresse und Netzwerk-Endpunkt (**Abbildung 2.11**).

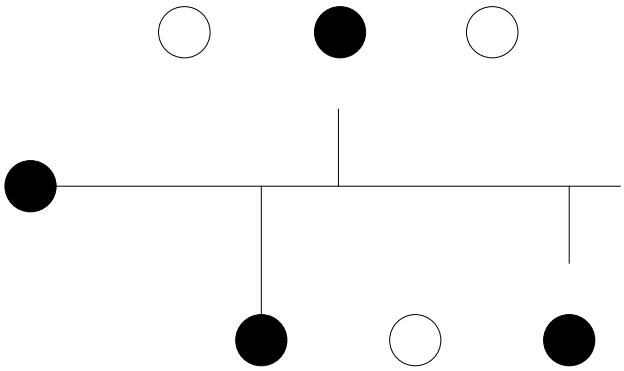

Abbildung 2.12 Anycast

Anycast: Das Anycast Routing Schema verteilt Informationen zu dem Zielrechner, der am „nächsten" oder am „besten erreichbar" ist. Es besteht eine 1:N-Beziehung zwischen Netzwerkadresse und Netzwerk-Endpunkt, wobei zu einem bestimmten Zeitpunkt jedoch nur ein einzelner Endpunkt zur Weiterleitung der Information adressiert wird (**Abbildung 2.12**).

2.3 Integrations-Architekturvarianten

Die grundlegenden Integrations-Architekturvarianten sind die Point-to-Point-Architektur, die Hub-and-Spoke-Architektur, die Pipeline-Architektur und die Service-Orientierte Architektur (Knappe et al. 2003).

- *Point-to-Point-Architektur:* Verbund unabhängiger Systeme, die durch ein Netzwerk miteinander verbunden sind.
- *Hub-and-Spoke-Architektur:* ein weiterer Schritt in der Evolution der Anwendungs- resp. Systemintegration. Ein zentraler Hub übernimmt die Rolle der Kommunikationsdrehscheibe.
- *Pipeline-Architektur:* Hier werden unabhängige Systeme entlang der Wertschöpfungskette mithilfe eines Nachrichten-Busses integriert. Die Bus-Fähigkeit resultiert daraus, dass die Schnittstellen zum zentralen Bus über das Kommunikationsnetzwerk verteilt installiert werden können und die Applikationen damit meist auch einen lokalen Zugang zur Bus-Schnittstelle erhalten.
- *Service-Orientierte Architektur:* Die Integration verschiedener Anwendungen zu einem funktionierenden Ganzen als Integration verteilter und unabhängiger Serviceaufrufe, die über einen ESB und ggf. eine Process Engine orchestriert werden.

2.3.1 Point-to-Point-Architektur

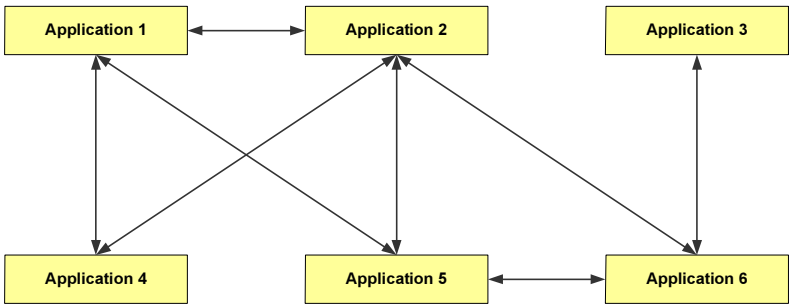

Abbildung 2.13 Point-to-Point-Architektur

Eine Point-to-Point-Architektur stellt einen Verbund unabhängiger Systeme dar, die durch ein Netzwerk miteinander verbunden sind (**Abbildung 2.13**). Alle Systeme sind gleichberechtigt und können sowohl Dienste in Anspruch nehmen als auch Dienste zur Verfügung stellen (Lublinsky 2002).

Diese Architektur findet sich in vielen Unternehmen, wo historisch gewachsene Applikationsinseln über die Zeit direkt miteinander verbunden wurden.

Hier gibt es keine zentrale Datenhaltung, jedes System verfügt über seinen eigenen Datenbestand.

Neue Systeme werden direkt mit den bestehenden verbunden, was eine mit der Zeit sehr hohe Schnittstellenkomplexität nach sich zieht. Eine Point-to-Point-Architektur mit n Applikationen besitzt theoretisch n*(n-1)/2 Schnittstellen.

Man kann sich gut vorstellen, wie komplex, fehleranfällig und schwer wartbar eine solche IT-Architektur werden kann, je mehr Applikationen hinzukommen. Erweite-

rungen sind kostenintensiv, und der Betrieb gestaltet sich mit zunehmenden Schnittstellen auch immer aufwendiger. Die Beurteilung als SWOT-Analyse ist in Tabelle 2.5 dargestellt.

Tabelle 2.5 Beurteilung Point-to-Point-Architektur

Stärken	**Schwächen**
Geringe Start-/Infrastrukturkosten Autonome Systeme	Nur bei wenigen Systemen und wenigen Verbindungen praktikabel Einzelne Systeme nur mit hohem Aufwand austauschbar Sehr unflexibel, keine Grundlage für SOA und daher schwierige Abbildung der Geschäftsprozesse Gesamtüberblick über Daten nicht vorhanden Wiederverwendbarkeit von Komponenten ist beschränkt Aufwändiger Betrieb
Chancen	**Gefahren**
Funktionserweiterung innerhalb der Systeme können schnell an neue Anforderungen angepasst werden.	hohe Folgekosten fehlende Standardisierung

2.3.2 Hub-and-Spoke-Architektur

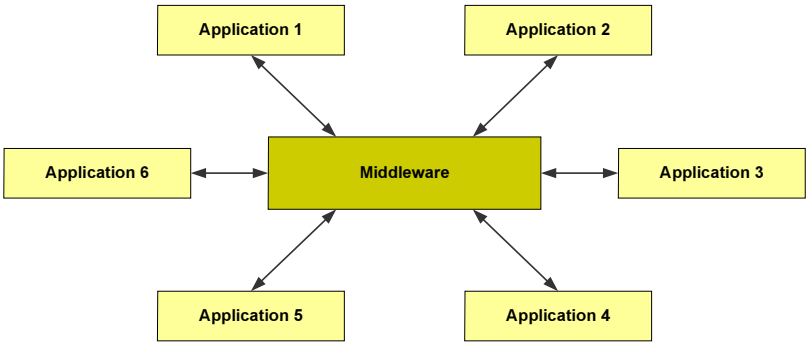

Abbildung 2.14 Hub-and-Spoke-Architektur

Die Hub-and-Spoke-Architekturform stellt einen weiteren Schritt in der Evolution der Anwendungs- resp. Systemintegration dar, wie **Abbildung 2.14** zeigt (Gilfix 2003). Sie verfolgt das Ziel, die mit der Dauer immer komplexer werdende Schnitt-

stellenproblematik zu minimieren, indem eine zentrale Integrationsplattform für den Nachrichtenaustausch zwischen den Systemen benutzt wird. Die zentrale Integrationsplattform kann die Nachrichten transformieren, von einer Anwendung an die nächste weiterleiten (Routing) oder auch die Nachrichteninhalte verändern. Diese Architektur wird oft auch für komplexe Datenverteilungsmechanismen eingesetzt. Die Beurteilung als SWOT-Analyse stellt Tabelle 2.6 dar.

Tabelle 2.6 Beurteilung Hub-and-Spoke-Architektur

Stärken	**Schwächen**
Reduzierung Schnittstellenproblematik Geringe Folgekosten Standards vorhanden Autonome Systeme Vereinfachte Überwachung	hohe Start-/Infrastruktur-Kosten
Chancen	**Gefahren**
einzelne Systeme mit geringem Aufwand einbind-/austauschbar	zentraler Hub könnte bei hohen Transfervolumina zum Performance-Bottleneck werden Single Point of Failure

2.3.3 Pipeline-Architektur

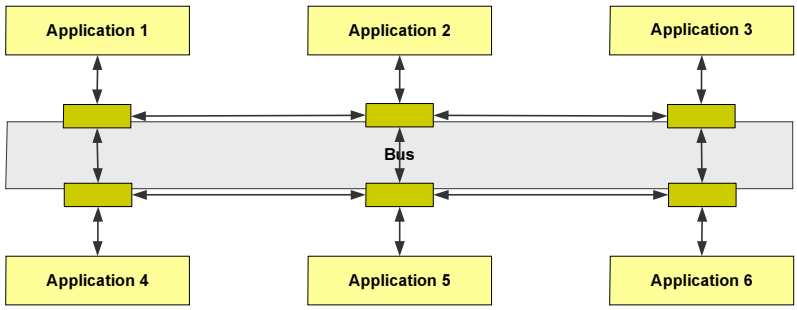

Abbildung 2.15 Pipeline-Architektur

In der Pipeline-Architektur werden unabhängige Systeme entlang der Wertschöpfungskette mithilfe eines Nachrichten-Busses integriert (**Abbildung 2.15**). Diese Architektur entspricht in der Implementierung der Hub-and-Spoke-Architektur, da die entsprechenden Middleware-Produkte im Normalfall auch hier auf zentralen Servern installiert und betrieben werden. Die Bus-Fähigkeit resultiert daraus, dass die Schnittstellen zum zentralen Bus über das Kommunikationsnetzwerk verteilt in-

stalliert werden können und die Applikationen damit meist auch einen lokalen Zugang zur Bus-Schnittstelle erhalten (Ambriola, Tortora 1993).

Ähnlich wie bei der Hub-and-Spoke-Architektur wird in dieser Architektur die Schnittstellenproblematik minimiert. Mit Einsatz entsprechender Middleware-Komponenten kann eine Standardisierung resp. Vereinheitlichung der Kommunikation der Systeme untereinander sichergestellt werden. Die Nachrichtenverteilung erfolgt über das Bussystem. Ein zentrales Repository enthält die Transformations- und Routingregeln; es können je nach Middleware-Produkt auch Business-Funktionen oder -Regeln abgebildet werden. Die Beurteilung als SWOT-Analyse stellt Tabelle 2.7 dar.

Diese Architekturform ist besonders geeignet für:

- sehr hohe Performance-Ansprüche (Event Driven Architecture)
- 1:n-Datenverteilung (z.B. Broadcasting)
- n:1-Datensammlung (z.B. DWH)

Tabelle 2.7 Beurteilung Pipeline-Architektur

Stärken	**Schwächen**
geringe Folgekosten sehr flexible Architekturform Standards vorhanden autonome Systeme	hohe Start-/Infrastruktur-Kosten
Chancen	**Gefahren**
einzelne Systeme mit geringem Aufwand einbind-/austauschbar	bei hohen Transfervolumina Gefahr eines Performance-Bottlenecks, wenn nicht vom normalen Verkehr getrennt (z.B. eigener Bulk Load Kanal)

2.3.4 Service-Orientierte Architektur

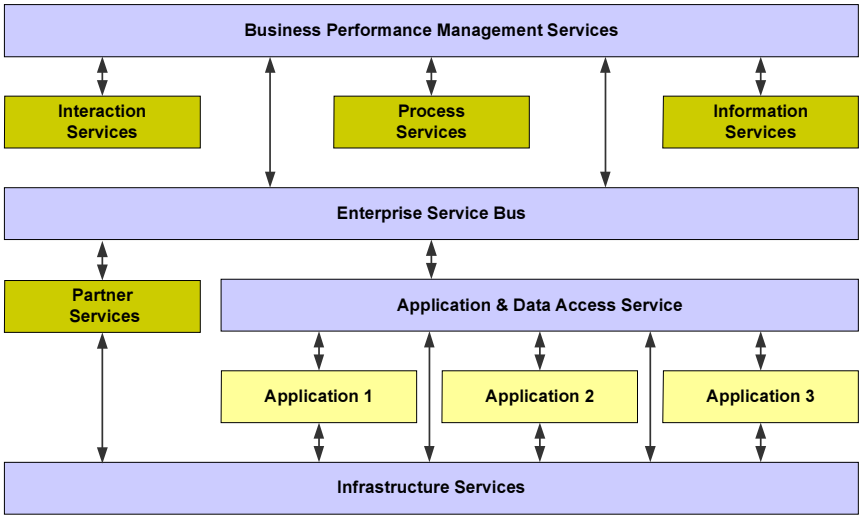

Abbildung 2.16 Service-Orientierte Architektur

Der Kern einer Service-Orientierten Architektur und damit auch der Hauptunterschied zu den vorgängig beschriebenen Integrations-Architekturformen besteht darin, dass Geschäftsprozesse resp. -Anwendungen nicht mehr als komplexe Programmstrukturen codiert, sondern als verteilte und unabhängige Serviceaufrufe orchestriert werden (**Abbildung 2.16**).

Als zentrale Integrationskomponente für die übergreifenden Serviceaufrufe kommt der Enterprise Service Bus zum Einsatz. Er beinhaltet ähnliche Eigenschaften wie die Integrationsplattform in der Hub-and-Spoke-Architektur oder wie der Bus in der Pipeline-Architektur (aus denen sich der ESB auch in einigen Produktfällen entwickelt hat). Die Beurteilung als SWOT-Analyse stellt Tabelle 2.8 dar.

Tabelle 2.8 Beurteilung Service-Orientierte Architektur

Stärken	**Schwächen**
Geringe Folgekosten Sehr flexible Architekturform Standards vorhanden Unterstützt von allen großen Herstellern	hohe Start-/Infrastrukturkosten bedingt eine umfassende SOA-Strategie und -Governance
Chancen	**Gefahren**
einzelne Services mit geringem Aufwand implementier- und orchestrierbar	Fehlende Fokussierung oder Ausrichtung auf die relevanten Geschäftsprozesse

2.4 Service-Oriented Integration

Service-Oriented Integration wird mittels der zwei grundlegenden Muster Process Integration und Workflow Integration realisiert (Adams 2001).

- *Process Integration:* Das Process Integration-Pattern erweitert die 1:N-Topologie des Broker-Patterns. Es erleichtert die serielle Ausführung von Geschäftsservices, die von den Ziel-Applikationen zur Verfügung gestellt werden.
- *Workflow Integration:* im Grunde eine Variante des Serial Process-Patterns; erweitert die Fähigkeit der einfachen seriellen Prozessorchestrierung um die Unterstützung von Benutzerinteraktion bei der Ausführung einzelner Prozessschritte.

2.4.1 Process Integration

Das Process Integration-Pattern erweitert die 1:N-Topologie des Broker-Pattern. Es erleichtert die serielle Ausführung von Geschäftsservices, die von den Ziel-Applikationen zur Verfügung gestellt werden, und erlaubt damit die Orchestrierung serieller Geschäftsprozesse, ausgehend von einer Interaktion der Quellapplikation. Die serielle Reihenfolge wird über die Process Rules definiert. Damit kann eine Entkopplung von Prozesslogik (Flow Logic) und Domänenlogik der Einzelapplikationen erreicht werden. Die Regeln definieren nicht nur den Kontroll- und Datenfluss, sondern auch die erlaubten Aufrufregeln für jede Zielapplikation. Zwischenresultate (Prozessdaten) werden in eigenen Speichern (Results) vorgehalten.

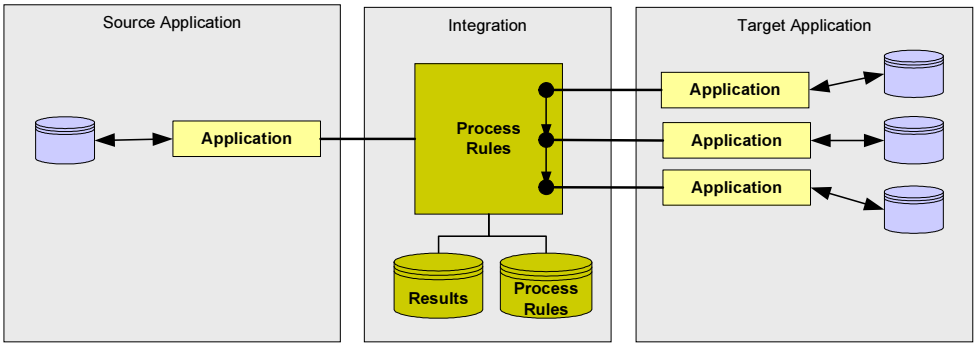

Abbildung 2.17 Process Integration

Das Prozess Integration-Pattern (**Abbildung 2.17**) kann in 3 Teile gegliedert werden:

- Die Quellapplikationen sind eine oder mehrere Applikationen, die mit den Zielapplikationen interagieren wollen.
- Die Komponente „Serial Process Rules" unterstützt die gleichen Dienste wie der Broker beim Broker-Pattern, inklusive Routing von Anfragen, Protokoll-Konvertierung, Message Broadcasting und Message Decomposition sowie

Recomposition. Zusätzlich wird die Auslagerung der Prozessfluss-Logik aus den individuellen Applikationen unterstützt. Die Prozesslogik wird durch serielle Prozessregeln bestimmt, die zusammen mit den Kontroll- und Datenflussregeln die Ausführungsregeln für die einzelnen Zielapplikationen definiert. Diese Regeln werden in einer Process Rules-Datenbank abgespeichert.

- Die Zielapplikationen repräsentieren neue, aber auch bestehende (angepasste oder nicht angepasste) Applikationen. Diese Applikationen sind verantwortlich für die Implementierung der notwendigen Geschäftsservices.

Die Vor- und Nachteile des Prozess Integration-Pattern sind in Tabelle 2.10 aufgelistet.

Tabelle 2.9 Vorteile und Nachteile des Process Integration-Pattern

Vorteile	Nachteile
Verbessert die Flexibilität und Reaktionszeit eines Unternehmens durch die Implementierung von End-to-End-Prozessflüssen und durch die Auslagerung der Prozesslogik aus den einzelnen Applikationen. Stellt die Basis für das Business Process Management dar, welches die Überwachung und Bewertung der Effektivität der Geschäftsprozesse erlaubt.	Nur direkte und automatische Abwicklung unterstützt, keine Benutzerinteraktion möglich (Workflow Variante beachten). Keine parallele Abwicklung möglich.

2.4.1.1 Einsatzgebiete

- Die Unterstützung von End-to-End-Prozessflüssen, die von den Zielapplikationen bereitgestellte Dienste nutzen.
- Die Verbesserung der Flexibilität und Reaktionszeit der IT über die Auslagerung der Prozesslogik aus den einzelnen Applikationen.

2.4.1.2 Varianten

Dieses Pattern kennt zwei Varianten: das Parallel Process Integration-Pattern und das External Business Rules-Pattern.

Das Parallel Process-Pattern erweitert die einfache serielle Prozessorchestrierung des Serial Process-Patterns durch die Unterstützung konkurrierender (concurrent) Ausführung und Orchestrierung von Geschäftsservice-Aufrufen. Die konkurrierende Ausführung der Subprozesse setzt ein geeignetes Aufsplitten und Zusammenführen der Teilschritte voraus, damit diese parallel ausgeführt werden können. Hierzu existieren unterschiedliche Muster auf Implementierungsebene (z.B. sog. Patterns for Parallel Computing) und verschiedene Architekturstile (z.B. Pipes-and-Filter-Architekturen). Die Zwischenergebnisse eines Teilschrittes können Einfluss

auf das Gesamtergebnis haben oder auch nicht. Es ist ebenso möglich, dass das Zwischenergebnis eines Teilschrittes die Ausführung anderer Teilschritte beeinflusst.

Die External Business Rules-Variante erweitert das Process Integration-Pattern um die Möglichkeit, Geschäftsregeln aus dem seriellen Prozess in eine Business Rule Engine auszulagern und von dieser auswerten zu lassen. Der Prozess reagiert dabei nur noch auf die Antworten der Rule Engine, die komplexen Regelauswertungen werden von der spezialisierten Rule Engine vorgenommen. Die Auslagerung der Regeln verbessert die Flexibilität und Reaktionszeit, da die Geschäftsregeln damit viel einfacher und schneller anpassbar werden.

2.4.2 Workflow Integration

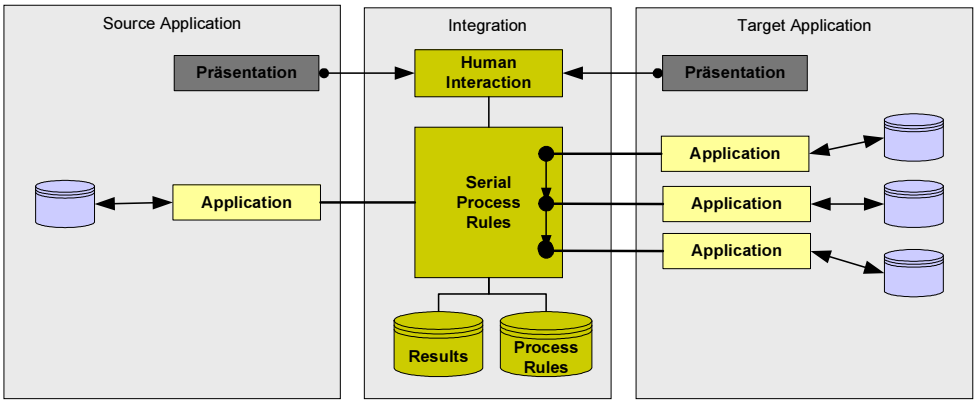

Abbildung 2.18 Workflow Integration

Das Workflow Integration-Pattern (**Abbildung 2.18**) ist eigentlich eine Erweiterung vom Process Integration-Pattern. Es erweitert die Fähigkeit der einfachen seriellen Prozessorchestrierung um die Unterstützung von Benutzerinteraktion bei der Ausführung einzelner Prozessschritte. Damit wird ein „klassischer" Workflow unterstützt.

2.4.2.1 Variante

Das Parallel Workflow Integration-Pattern ist eine Variante des Workflow Integration-Pattern und entspricht der Parallel Process Integration-Pattern des Process Integration-Patterns. Es erweitert die Fähigkeit der parallelen Prozessorchestrierung um die Unterstützung von Benutzerinteraktion bei der Ausführung einzelner Prozessschritte. Damit wird ein „paralleler" Workflow unterstützt.

2.5 Data Integration

Data Integration wird mittels dreier grundlegender Pattern realisiert:

- *Federation:* erlaubt den Zugriff auf verschiedene Datenquellen und erweckt dabei für die aufrufende Applikation den Anschein, es handle sich um eine einzelne, logische Datenquelle.
- *Population:* sammelt Daten von einer oder mehreren Datenquellen, verarbeitet diese Daten in geeigneter Form und appliziert sie gegen eine Zieldatenbank.
- *Synchronisation:* erlaubt bidirektionale Update-Flüsse von Daten in Multi-Copy-Datenbankumgebungen.

2.5.1 Federation

Das Federation-Pattern ist ein einfaches Data Integration-Pattern, das den Zugriff auf verschiedene Datenquellen ermöglicht und dabei für die aufrufende Applikation den Anschein erweckt, es handle sich um eine einzelne, logische Datenquelle. Dies erreicht man folgendermaßen:

1. Eine einzige konsistente Schnittstelle wird gegen außen angeboten.
2. Übersetzung der Schnittstelle auf die unterliegenden Daten.
3. Kompensation der funktionalen Unterschiede zwischen den unterschiedlichen Datenquellen.
4. Daten von mehreren unterschiedlichen Datenquellen können zu einem einzelnen Result-Set zusammengefasst und so dem Benutzer zur Verfügung gestellt werden.

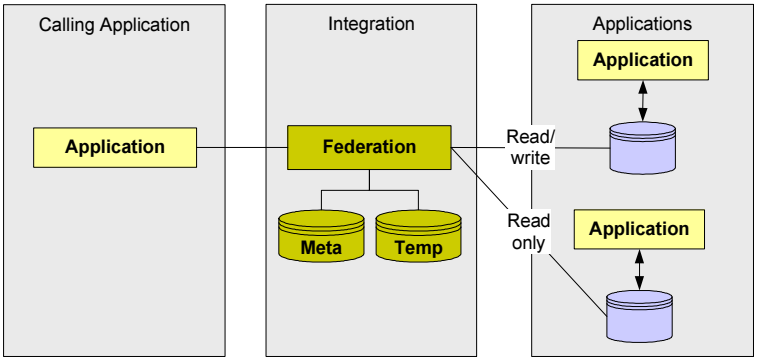

Abbildung 2.19 Federation

Das Federation-Pattern (**Abbildung 2.19**) ist in folgende logischen Komponenten aufgeteilt:

- Die aufrufenden Applikationen haben ein Informationsbedürfnis, besitzen die Informationen aber nicht.
- Die Federation-Komponente verwendet Metadaten, um bestimmen zu können, wo und in welchem Format die benötigten Daten gespeichert sind. Das Metadaten Repository ermöglicht weiter die Dekomposition eines einzelnen Queries in einzelne Anfragen an die unterschiedlichen Datenquellen. Das Informationsmodell erscheint damit dem Nutzer (aufrufenden Applikationen) als ein einheitliches virtuelles Repository. Über entsprechende Adapter für jedes Zielrepository wird auf die Daten zugegriffen. Die Federation-Komponente gibt ein einzelnes Resultat zur aufrufenden Applikation zurück und integriert dabei die mehreren unterschiedlichen Formate in ein gemeinsames Federated Schema.
- Die Quellapplikationen besitzen die Information, die für die aufrufenden Applikationen wichtig sind.

Federation unterstützt sowohl strukturierte als auch unstrukturierte Daten, sowie Read-Only- und Read/Write-Zugriff auf die unterliegenden Datenquellen. Read/Write-Zugriffe sollten wenn möglich auf eine einzelne Datenquelle beschränkt werden, da sonst ein Two-Phase Commit notwendig wird, der bei verteilten Datenbanken schwierig sein kann.

2.5.1.1 Einsatzgebiete

- Von einer Applikation benötigte Daten liegen verteilt in verschiedenen Datenbanken (aus historischen, technischen oder organisatorischen Gründen).
- Federation ist besser als andere Daten-Integrationstechnologien, wenn
  - ein Nahezu-Real-Time-Zugriff notwendig ist auf oft veränderte Daten;
  - das Anfertigen einer konsolidierte Kopie der Daten aus technischen, legalen oder anderen Gründen nicht möglich ist;
  - Read/Write-Zugriff möglich sein muss;
  - das Reduzieren bzw. Minimieren von Datenkopien ein Ziel ist.
- Bestehende Investitionen können weiter genutzt werden.

2.5.2 Population

Dem Population-Pattern liegt ein sehr einfache Modell zu Grunde. Es sammelt Daten von einer oder mehreren Datenquellen, verarbeitet diese Daten in geeigneter Form und appliziert sie gegen eine Zieldatenbank. Im einfachsten Fall repräsentiert das Population-Pattern das Modell „Datenset lesen, Daten verarbeiten – Dataset schreiben“. Dies entspricht dem klassischen ETL (Extract Transform Load)-Prozess.

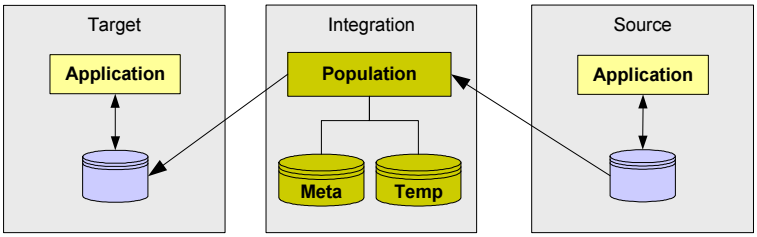

Abbildung 2.20 Population

Das Population-Pattern (**Abbildung 2.20**) ist in folgende logischen Komponenten aufgeteilt:

- Die Zielapplikationen sind die Applikationen, die ein Informationsbedürfnis haben, diese Informationen selbst aber nicht besitzen und daher eine Kopie von einer anderen Datenquelle einer Quellapplikation benötigen.
- Die Population-Komponente liest von einer oder mehren Datenquellen der Quellapplikationen und schreibt diese in eine Datenquelle der Zielapplikation. Die Regeln für die Extraktion von den Quellapplikationen kann beliebig komplex sein, von einfachen Regeln wie „alle Daten lesen“ bis zu komplexeren, wo nur spezifische Felder von spezifischen Records unter bestimmten Bedingungen gelesen werden sollen. Ebenso können die Laderegeln in die Zieldatenbank vom einfachen Überschreiben der Daten bis zum komplexeren Prozess des Einfügens neuer Records und Updates bestehender Records reichen. Die Metadaten beschreiben diese Regeln.
- Die Quellapplikationen besitzen die für die Ziel-Applikationen wichtigen Information.

2.5.2.1 Einsatzgebiete

- eine spezialisierte Kopie bereits existierender Daten (abgeleitete Daten) ist notwendig:
  - Subsets von bestehenden Datenquellen
  - Modifizierte Version einer bestehenden Datenquelle
  - Kombinationen von bestehenden Datenquellen
- Nutzung der abgeleiteten Daten in der Zielapplikation ist nur lesend (oder nur wenige schreibende Zugriffe).
- Bei einer signifikanten Anzahl von schreibenden Zugriffen ist die Verwendung des Two-Way Synchronization-Patterns angebracht.
- Der Benutzer soll einen schnellen Zugriff auf die Informationen bekommen, die er benötigt, und er sollte nicht mit zu vielen, nicht relevanten, inkorrekten und anderweitig sinnlosen Daten belästigt werden.

- Oft ist der Ansporn zur Nutzung des Population-Patterns jedoch die IT, d.h., das Anfertigen von Datenkopien erfolgt aus technischen Gründen. Diese Treiber können sein:
  - Die verbesserte Performance von Benutzerzugriffen
  - Die Lastverteilung über mehrere Systeme

2.5.3 Synchronisation

Das Synchronization (auch bekannt als Replication)-Pattern erlaubt bidirektionale Update-Flüsse von Daten in Multi-Copy-Datenbankumgebungen. Der „two-way"-Synchronisierungsaspekt dieses Patterns unterscheidet es von den „one-way"-Fähigkeiten, die der Population-Pattern zur Verfügung stellt.

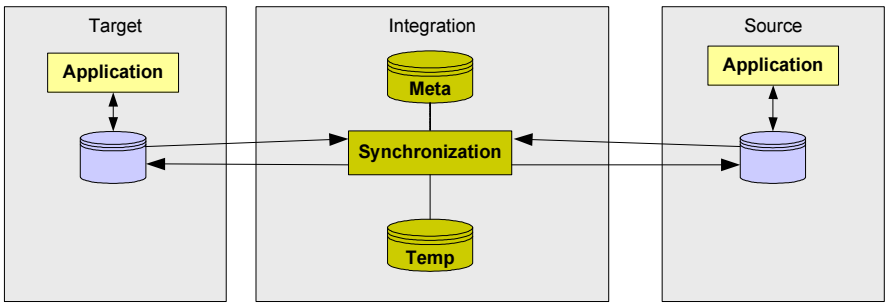

Abbildung 2.21 Synchronsation

Das Synchronization-Pattern (**Abbildung 2.21**) ist in folgende logischen Komponenten aufgeteilt:

- Die Zielapplikationen sind die Applikationen, die ein Informationsbedürfnis haben, diese Informationen selbst aber nicht besitzen und daher eine Kopie von einer anderen Datenquelle einer Quellapplikation benötigen.
- Die Synchronization-Komponente kann im einfachsten Fall mit dem Population-Pattern verglichen werden, mit dem einzigen Unterschied, dass Daten in beide Richtungen fließen. Wenn die Datenelemente, die in beide Richtungen fließen, völlig unabhängig voneinander sind, besteht die Two-Way-Synchronisation aus nicht mehr als zwei separaten Instanzen des Population-Patterns. Allerdings kommt häufiger eine gewisse Überschneidung der Datensets vor, die in die eine oder andere Richtung fließen. In diesem Fall sind Konflikterkennung und -lösung wichtig.
- Die Quellapplikationen besitzen die für die Zielapplikationen relevanten Informationen.

2.5.3.1 Einsatzgebiete

- Eine spezialisierte Kopie bereits existierender Daten (abgeleitete Daten) ist notwendig. Diese Kopie kann verschiedene Ausprägungen haben:
  - Subsets von bestehenden Datenquellen
  - Modifizierte Version einer bestehenden Datenquelle
  - Kombinationen von bestehenden Datenquellen

2.5.3.2 Multi-Step-Synchronisation

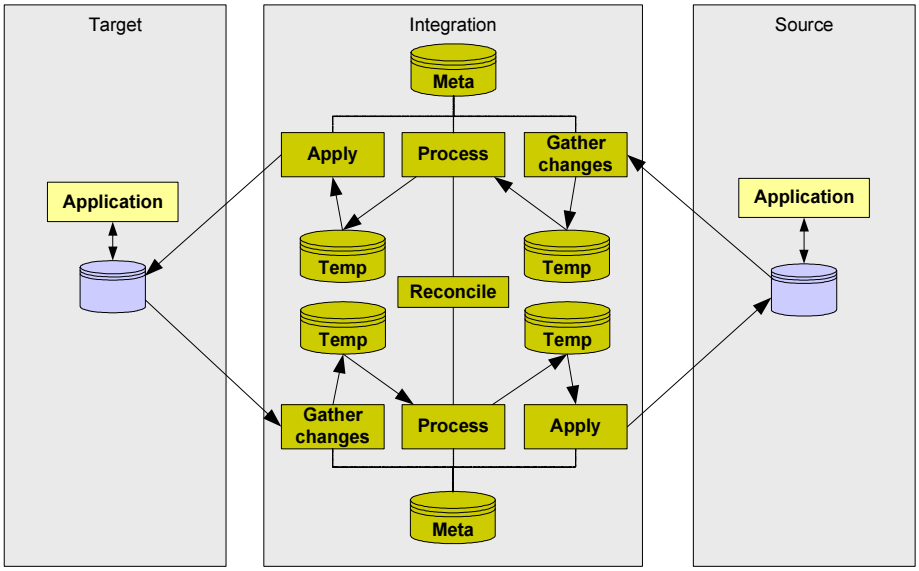

Abbildung 2.22 Multi-Step-Synchronisation

Das Synchronization-Pattern kennt eine Variante, die Multi-Step-Variante (**Abbildung 2.22**). Die Multi-Step-Variante des Two-Way-Synchronization-Patterns nutzt für die beiden Richtungen der Synchronisation jeweils einmal das Population-Pattern mit den Gather-, Process- und Apply-Schritten. Ein zusätzlicher Schritt „Reconcile" ist zwischen den beiden Datenflüssen positioniert und stellt sicher, dass es bei den Updates keine Konflikte geben kann. Wenn die Möglichkeit für Konflikte gering ist, kann dieses Pattern komplett auf existierenden Population-Komponenten aufgebaut werden. Für komplexere Situationen ist jedoch eine spezialisierte Lösung anzustreben.

2.6 EAI / EII

Zur Umsetzung von EAI (Enterprise Application Integration)- und EII (Enterprise Information Integration)-Plattformen werden drei grundlegende Patterns verwendet:

- *Direct Connections:* repräsentiert die einfachste Interaktionsart zweier Applikationen und basiert auf einer 1:1-Topologie, d.h. einer einzelnen Point-to-Point-Verbindung.
- *Broker*: baut auf dem Direct-Connections-Pattern auf und erweitert dieses auf eine 1:1-Topologie; ermöglicht es, eine einzelne Anfrage ausgehend von der Quellappplikation an mehrere Zielapplikationen weiterzuleiten.
- *Router*: eine Variante des Broker-Patterns, bei dem es mehrere potenzielle Zielapplikationen gibt, jedoch immer nur genau eine Zielapplikation die Nachricht weitergeleitet bekommt.

2.6.1 Direct Connections

Direct-Connection repräsentiert die einfachste Interaktionsart zweier Applikationen und basiert auf einer 1:1-Topologie, d.h. einer einzelnen Point-to-Point-Verbindung. Es erlaubt einem Paar von Applikationen, direkt innerhalb eines Unternehmens zu kommunizieren. Interaktionen zwischen den Quell- und den Zielapplikationen können beliebig komplex sein. Für komplexere Point-to-Point-Verbindungen werden zusätzlich Connection Rules definiert. Beispiele von Connection Rules: Data Mapping-Regeln, Sicherheitsregeln oder Verfügbarkeitsregeln.

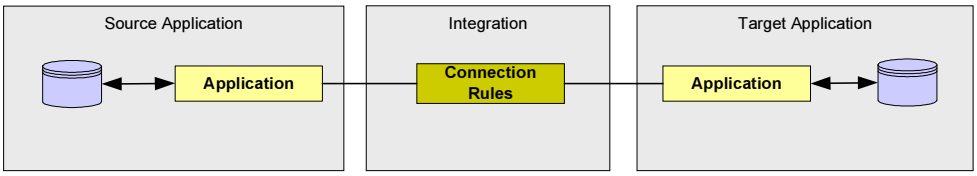

Abbildung 2.23 Direct Connections

Das Direct Connections-Pattern (**Abbildung 2.23**) ist in folgende logische Komponenten aufgeteilt.

- Die Quellapplikationen sind eine oder mehrere Applikationen, die eine Interaktion mit den Zielapplikationen initiieren wollen.
- Die Verbindung ist die Linie zwischen der Quell- und der Zielapplikation und repräsentiert eine Point-to-Point-Verbindung zwischen den beiden Applikationen.
- Connection Rules repräsentiert die zur Verbindung gehörenden Geschäftsregeln, wie Data Mapping und Security Rules.

- Die Zielapplikation ist eine neue oder bestehende (angepasste oder nicht angepasste) Applikation, die die notwendigen Geschäftsservices zur Verfügung stellt.

Die Vor- und Nachteile des Direct Connections-Patterns sind in Tabelle 2.10 aufgelistet.

Tabelle 2.10 Vor- und Nachteile des Direct Connections-Patterns

Vorteile	Nachteile
Funktioniert gut für Applikationen mit einfachen Integrationsanforderungen und nur einigen Backend-Applikationen. Lose Koppelung. Empfänger müssen nicht online sein.	Führt zu vielen Point-to-Point-Verbindungen zwischen jedem Paar von Applikationen und zu Spaghetti-Konfigurationen. Unterstützt kein intelligentes Routing von Anfragen. Unterstützt keine Decomposition / Recomposition von Anfragen.

2.6.1.1 Einsatzgebiete

Direct Connection kommt in folgenden Einsatzgebieten zur Anwendung:

- Reduktion der Latenzzeit von Business Events
- Unterstützung für strukturierten Informationsaustausch innerhalb eines Unternehmens
- Unterstützung für Real-Time-Ein-Weg-Meldungsflüsse
- Unterstützung für Real-Time Request/Reply-Meldungsflüsse
- Weiternutzung von Legacy-Investitionen
- Die Integration von Backend-Applikationen ermöglicht

eine Minimierung der Applikationskomplexität.

2.6.2 Broker

Das Broker-Pattern baut auf dem Direct-Connections-Pattern auf und erweitert dieses auf eine 1:1 Topologie. Es erlaubt, eine einzelne Anfrage ausgehend von der Quellappplikation an mehrere Zielapplikationen weiterzuleiten. Dadurch wird die Anzahl notwendiger 1:1-Verbindungen reduziert. Die Verbindungsregeln werden in den Broker Rules vorgehalten. Auf diese Weise trennt man die Regeln für die Verteilung von der Applikationslogik (Separation of Concerns). Die Kompositionen und Dekompositionen der Interaktionen nimmt ebenfalls der Broker vor. Das Broker-Pattern nutzt das Direct-Connections-Pattern für die Verbindung zwischen den Applikationen. Es Dabei sind beide Varianten – Message Connection und Call

Connection – denkbar. Das Broker-Pattern stellt die Basis für einen Publish/Subscribe-Meldungsfluss zur Verfügung.

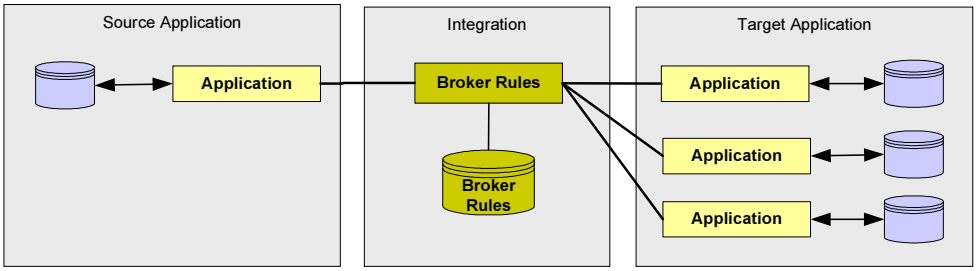

Abbildung 2.24 Broker

Das Broker-Pattern (**Abbildung 2.24**) ist in folgende logische Komponenten aufgeteilt:

- Die Quellapplikationen sind eine oder mehrere Applikationen, die mit den Zielapplikationen interagieren wollen.
- Die Broker-Komponente reduziert das rasche Anwachsen von direkten Verbindungen. Zudem unterstützt sie Message Routing, Message Enhancement, Transformation, Decomposition und Recomposition von Messages.
- Die Zielapplikationen repräsentieren neue, aber auch bestehende (angepasste oder nicht angepasste) Applikationen. Diese Applikationen sind verantwortlich für die Implementierung der notwendigen Geschäftsservices.

Die Vor- und Nachteile des Broker-Patterns listet Tabelle 2.11 auf.

Tabelle 2.11 Vor- und Nachteile des Broker-Patterns

Vorteile	Nachteile
Ermöglicht die Interaktion mehrerer, unterschiedlicher Applikationen. Minimiert die Auswirkungen auf den bestehenden Applikationen. Stellt Routing Services zur Verfügung, so dass die Quellapplikation nicht mehr länger die Zielapplikationen kennen muss. Stellt Transformation Services zur Verfügung, die es der Quell- und Zielapplikation erlauben, unterschiedliche Kommunikationsprotokolle zu nutzen. Es können Decomposition/Recomposition - Dienste zur Verfügung gestellt werden, so dass sich eine einzelne Anfrage von einer Quelle an mehrere unterschiedliche	Für die Aufgaben Routing und Decomposition/Recomposition muss Logik auf dem Broker implementiert werden.

Zielapplikationen senden lässt. Die Verwendung des Routers minimiert die Anzahl notwendiger Anpassungen, wenn die Location der Zielapplikation geändert wird.	

2.6.2.1 Einsatzgebiete

- Eine einzelne Applikation soll mit einer oder mehreren Zielapplikationen interagieren.
- Eine Hub-and-Spoke-Architektur anstelle einer Point-to-Point-Architektur minimiert die Komplexität.
- Die Auslagerung der Routing-, Decomposition- und Recomposition-Regeln erhöht die Wartbarkeit und Flexibilität.
- Wichtig, wenn die Verarbeitung einer Anfrage von einer Quellapplikation zu mehreren Interaktionen mit den Zielsystemen führt.
- Das Quellsystem soll nichts über die Zielapplikationen wissen müssen.

2.6.3 Router

Das Router-Pattern ist eine Variante des Broker-Patterns, bei dem es mehrere potenzielle Zielapplikationen gibt, jedoch immer nur genau eine Zielapplikation die Nachricht weitergeleitet bekommt. Der Router entscheidet, an welche Zielapplikation die Interaktion weitergeleitet wird. Während das Broker-Patterns 1:N-Verbindungen unterstützt, erlaubt das Router-Pattern nur 1:1-Verbindungen, indem die Router Rules die entsprechende Zielapplikation bestimmen.

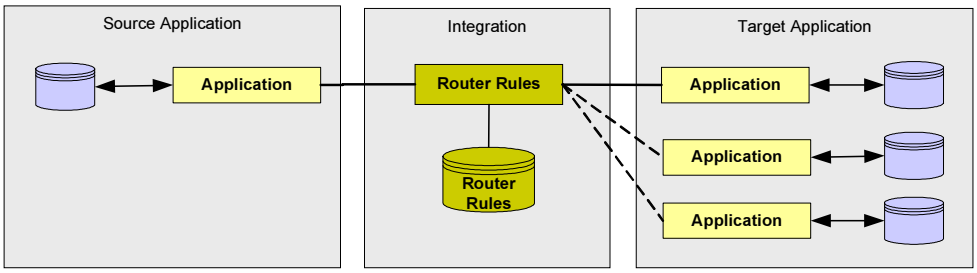

Abbildung 2.25 Router

Das Router-Pattern (**Abbildung 2.25**) ist in folgende logische Komponenten aufgeteilt:

- Die Quellapplikationen sind eine oder mehrere Applikationen, die mit den Zielapplikationen interagieren wollen.

- Die Router-Komponente repräsentiert sämtliche für die Behandlung der Meldung benötigten Geschäftsregeln, wie z.B. Routing und Transformation. Sie empfängt Anfragen von mehreren Quellapplikationen und leitet diese intelligent zur richtigen Zielapplikation. Bei der resultierenden Integration handelt es sich in Wirklichkeit um eine Point-to-Point-Verbindung zwischen Quelle und Ziel.
- Die Zielapplikationen repräsentieren neue, aber auch bestehende Applikationen (angepasste oder nicht angepasste), die für die Implementierung der erforderlichen Geschäftsservices verantwortlich sind.

Die Vor- und Nachteile des Router-Patterns listet Tabelle 2.12 auf.

Tabelle 2.12 Vor- und Nachteile des Router-Patterns

Vorteile	Nachteile
Ermöglicht die Interaktion mehrerer, unterschiedlicher Applikationen. Minimiert die Auswirkungen auf bestehende Applikationen. Stellt Routing Services zur Verfügung, so dass die Quellapplikation nicht mehr länger die Zielapplikationen kennen muss. Stellt Transformation Services zur Verfügung, die es der Quell- und Zielapplikation erlauben, unterschiedliche Kommunikationsprotokolle zu nutzen. Die Verwendung des Routers minimiert die Anzahl notwendiger Anpassungen, wenn die Location der Zielapplikation geändert wird.	Keine Dekomposition und Recomposition der Meldungen. Keine Möglichkeit, basierend auf der einkommenden Anfrage mehrere gleichzeitige Anfragen an Zielapplikationen zu senden.

2.6.3.1 Einsatzgebiete

- Eine einzelne Applikation soll mit einer von mehreren Zielapplikationen interagieren.
- Eine Hub-and-Spoke-Architektur anstelle einer Point-to-Point-Architektur minimiert die Komplexität.
- Die Auslagerung der Routing-, Decomposition- und Recomposition-Regeln erhöht die Wartbarkeit und Flexibilität.
- Wichtig, wenn die Verarbeitung einer Anfrage von einer Quellapplikation zu einer Interaktion mit nur einem Zielsystem aus mehreren potenziellen Zielsystemen führt.
- Das Quellsystem soll nichts über die Zielapplikationen wissen müssen.

2.7 Event Driven Architecture

Event Driven Architectures (EDA) sind gegenwärtig in aller Munde. Oft werden sie fälschlicherweise als Nachfolger von Service-Oriented Architectures (SOA) bezeichnet (Mühl et al. 2006). Tatsache ist, dass die Konzepte rund um EDA schon so alt sind wie die IT selbst. Tatsache ist ferner, dass EDA's im Zusammenhang mit den Integrationsarchitekturen von SOA im Moment einen enormen Aufschwung erleben, beide Architekturstile jedoch völlig unabhängig voneinander eingesetzt und orthogonal kombiniert werden können. Unter dem Blickwinkel der Integration sind zwei Aspekte der EDA von besonderem Interesse:

- Die bereits angedeutete Symbiose von EDA und SOA, um SOA-Domänen ereignisbasiert miteinander zu verbinden, d.h. zu integrieren.
- Die Technologie, welche EDA anbietet, um auf der Ebene der Datenintegration Ereignisse aus einem oder mehreren Ereignisströmen zu Informationen zu verdichten.

2.7.1 Einführung EDA

Glaubt man einer Studie von Gartner (Gartner 2006), so sind die Erfolge von Firmen wie Dell oder Google auf die Tatsache zurückzuführen, dass beide Unternehmen die Marktimpulse – Marktereignisse – auf dem globalen Handelsplatz früh erkennen und diese folgerichtig und schnell umsetzen. Beide Beispiele kommen dem ebenfalls von Gartner gezeichneten Bild eines Idealunternehmens sehr nahe: der so genannten Real Time Enterprise (RTE). Eine RTE zeichnet sich durch hoch automatisierte Geschäftsprozesse und kleinstmögliche Prozesslaufzeiten aus (Nussdorfer, Martin 2003).

Während mit SOA-Konzepten innerhalb der IT-Strukturen die Basis für die Automatisierung von Geschäftsprozessen geschaffen wird, geht es, dem Idealbild RTE folgend, in einem zweiten Schritt um die Verarbeitung immer feinkörnigerer Informationen über die Zustandsänderungen dieser Geschäftsprozesse. Die Komplexität der Zustandsänderungen wird zusehends größer, genauso wie die Zahl elektronischer Reaktionsschnittstellen in den Geschäftsprozessen. Die Bedeutung von EDA liegt genau hier, denn die beobachtbaren Änderungen im Zustand der Geschäftsprozesse können als Ereignisse modelliert werden. Klassische Integrationsarchitekturen à la OLTP oder OLAP sind den Anforderungen nach raschen und folgerichtigen Aktionen aufgrund von Ereignis-Analysen kaum mehr gewachsen (Zeidler 2007).

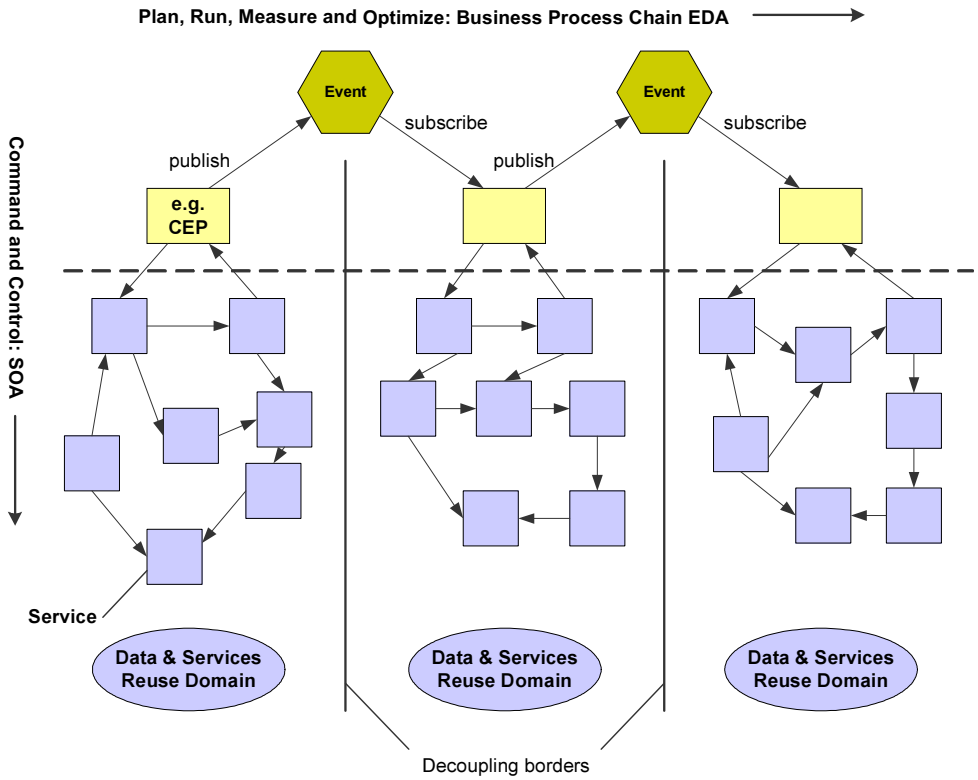

Abbildung 2.26 Event Driven Architecture

Abbildung 2.26 zeigt die Idee der Symbiose zwischen EDA und SOA, die oftmals auch als SOA 2.0 (Carter 2007) oder Next Generation SOA bezeichnet wird (Luckham 2002). Innerhalb einer SOA bzw. einer SOA-Domäne werden fachliche, kombinierbare (d.h. orchestrierbare) und von Konsumenten unabhängige Services, also die Bausteine der Geschäftsprozesse, geliefert. Löst nun ein solcher Service z.B. durch seine Zustandsänderung ein Ereignis aus, kann durch eine orthogonale EDA-Erweiterung eine neue SOA-Domäne aktiviert werden. Ein Service wird dadurch zum ereignis-produzierenden EDA-Baustein. Im Unterschied zum typischen Produce-Consume-Pattern einer SOA arbeitet die EDA meist nach einem Publish-Subscribe-Verfahren. Ein sog. Event Processor verarbeitet die ankommenden Ereignisse und publiziert seinerseits die verarbeiteten Ereignisse bzw. deren Resultate an einen sog. Event Channel, über den Services anderer SOA-Domänen angestoßen werden. Die Ausprägung solcher Event-Prozessoren kann je nach Art der Ereignisverarbeitung sehr unterschiedlich ausfallen. Es kann sich beispielsweise um einen Complex Event-Prozessor handeln, wie er im Anschluss an diesen Abschnitt dargestellt wird.

Die (zu integrierenden) SOA-Domänen sind idealerweise so definiert, dass sie wieder verwendbare Serviceleistungen darstellen und in beliebig langen Geschäfts-

prozess-Ketten mehrfach genutzt werden können. Je fein-granularer die SOA-Domänen und ihre EDA-Erweiterung, desto flexibler wird das System. Wie bei der SOA ist auch bei der EDA das Prinzip der losen Kopplung (loosely coupled) von entscheidender Bedeutung für die Bildung solch flexibler Geschäftsprozessketten.

2.7.2 Event Processing

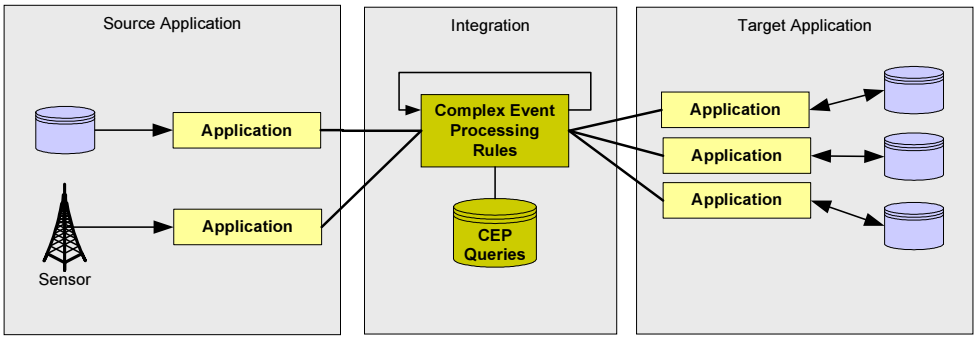

Abbildung 2.27 Event Processing

Der zweite Aspekt, den wir im Zusammenhang mit dem Thema „Integration" beleuchten wollen, ist eher technischer Natur: es geht um die Möglichkeiten der Ereignisverarbeitung innerhalb des EDA-Konzepts, wie in **Abbildung 2.27** dargestellt. Ereignisverarbeitende Technologien gehören in manchen Branchen seit Jahren zum Alltag. Denken wir nur an das Algorithmic Trading in Börsengeschäften oder an die RFID-Technologie in Mautsystemen.

Drei Typen der Ereignisverarbeitung können grundsätzlich unterschieden werden: Simple Event Processing (SEP), Event Stream Processing, (ESP) und Complex Event Processing (CEP).

2.7.2.1 Simple Event Processing (SEP)

Ereignisse treten grundsätzlich einzeln oder stromartig auf. Einzelereignisse können als wichtige Zustandsänderung in einer Nachrichtenquelle – speziell auch eines Geschäftsereignisses – angesehen werden. Solche Ereignisse lösen durch ihr Auftreten typischerweise Prozesse in jenen Systemen aus, die die Nachricht auch empfangen werden. Diese Form der Ereignisverarbeitung entspricht exakt der Spezifikation zu Java Messaging Service, kurz JMS. Typisches Beispiel einer SEP ist also JMS.

2.7.2.2 Event Stream Processing (ESP)

Die flussartige Bearbeitung eingehender Nachrichten bzw. Ereignisse bezeichnet der Begriff „ESP". Typische ESP-Systeme besitzen Sensoren, die eine große Anzahl

von Ereignissen durchschleusen und mittels Filtern und anderen Verarbeitungsschritten den Nachrichten- bzw. Ereignisfluss beeinflussen. Das Einzelereignis verliert an Bedeutung, während der Ereignisfluss an Stellenwert gewinnt. Bekannte Beispiele sind Systeme, die Börsenkurse verfolgen: eine einzelne Kursschwankung ist meist nicht sehr aussagekräftig, während sich der gehaltvollere Trend nur aus mehreren Ereignissen herleiten lässt.

2.7.2.3 Complex Event Processing (CEP)

Als dritte Verarbeitungsform wird oftmals das Complex Event Processing genannt. CEP ist ein Zweig der ESP-Technologie. Der Fokus bei CEP liegt jedoch besonders stark im Aufspüren von Mustern unter einer Vielzahl von Ereignissen sowie deren (Nachrichten-)Inhalten, die auch über verschiedene Datenströme verteilt sein können.

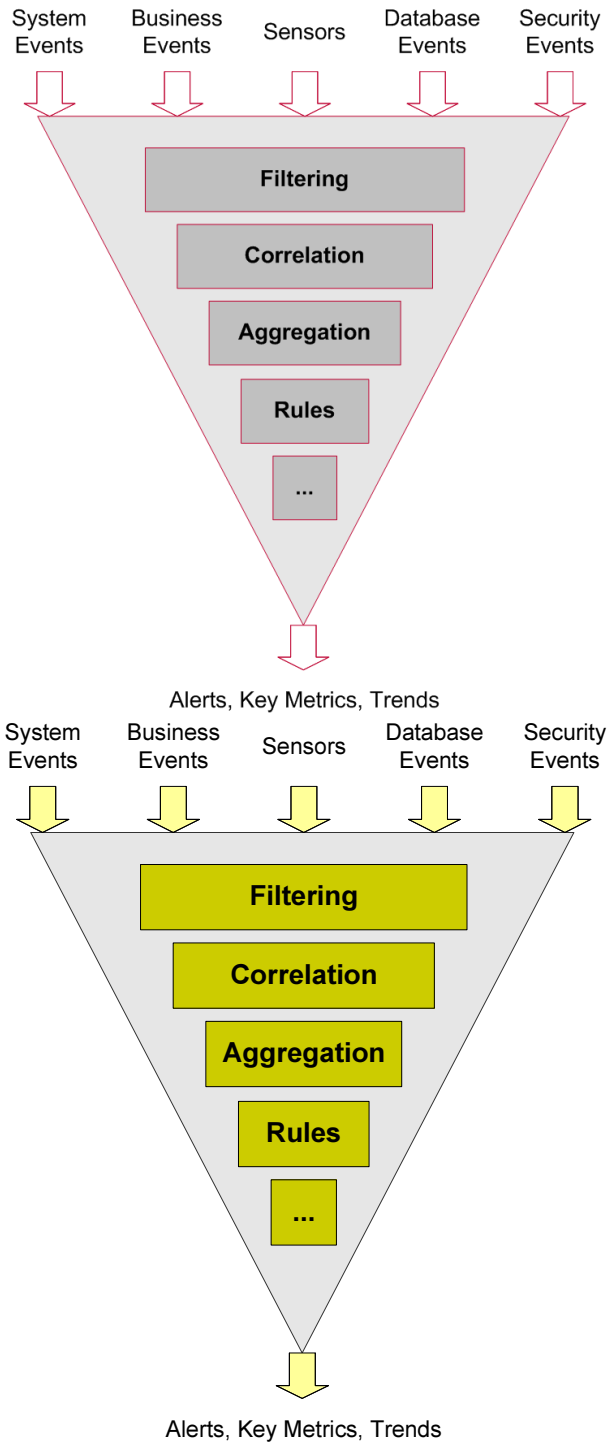

Abbildung 2.28 Das „Trichter-Modell" von CEP

Das „Trichter-Modell" von CEP zeigt die Verdichtung von (großen) Ereignismengen zu verdichteter Information (**Error! Reference source not found.**). Als Ereignisquelle kommen auch Geschäftsereignisse in Frage (Viehmann 2008). Der klassische Anwendungsfall einer CEP-Implementierung ist das Aufspüren von Kreditkartenbetrug: Man stelle sich zwei Transaktionen von jeweils ein und derselben Kreditkarte vor, die in zeitlich naher Abfolge, doch an weit voneinander entfernten Orten stattfinden. Ein solches räumliches und zeitliches Muster, das auf einen Betrugsfall schließen lässt, kann auf das Modell einfach angewendet werden. Solche Systeme, die mit Tausenden von Transaktionsereignissen korrelieren und die wenigen Betrugsfälle herausfiltern, sind heute real im Einsatz.

2.8 GRID / XTP

GRID und XTP sind neue Integrationstechnologien, deren Umsetzung in den nächsten Jahren zu erwarten ist.

- *GRID:* eine Infrastruktur zur integrierten, kollaborativen Nutzung von Ressourcen. Grids lassen sich hinsichtlich ihrer primären Funktionalität in Data Grid, In-Memory Data-Grid, Domain Entity Grid und Domain Object Grid differenzieren und kennen eine Vielzahl von Anwendungen.
- *XTP:* Extreme Transaction Processing ist eine Architektur für verteilte Speicherarchitekturen mit parallelen Applikationszugriffmöglichkeiten. Sie ist geeignet für den verteilten Zugriff auf große und sehr große Datenmengen.

2.8.1 GRID

Grid Computing (engl.; Gitterberechnung) bezeichnet alle Methoden, die Rechenleistung vieler Computer innerhalb eines Netzwerks so zusammenzufassen, dass über den reinen Datenaustausch hinaus die (parallele) Lösung von rechenintensiven Problemen ermöglicht wird (verteiltes Rechnen). Jeder Computer in dem „Gitter" ist eine den anderen Computern gleichgestellte Einheit. Damit übertrifft man zu deutlich geringeren Kosten sowohl die Kapazität als auch die Rechenleistung heutiger Supercomputer. Grid-Systeme skalieren sehr gut: durch Hinzufügen von Rechnern zum Netz oder Zusammenfassen von Grids zu Meta-Grids erhöht sich die Rechenleistung in entsprechendem Maße.

Definition Grid

Grid ist Infrastruktur zur integrierten, kollaborativen Nutzung von Ressourcen, die verschiedenen Organisationen gehören und von diesen verwaltet werden (Foster, Kesselmann 1999).

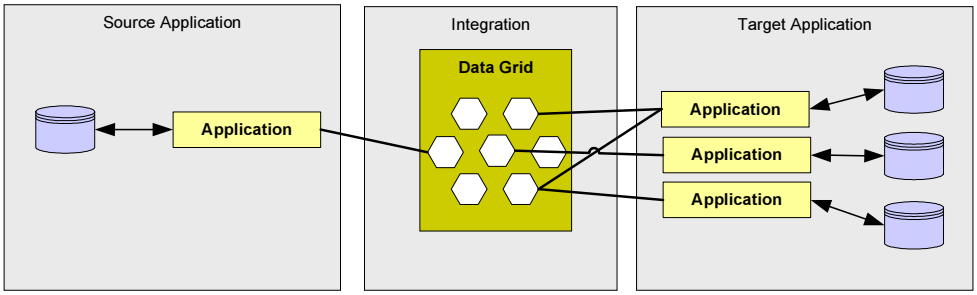

Abbildung 2.29 GRID

Die wesentliche Aufgaben von Grids sind das „Distributed Caching und -Processing" und das „Event Driven Processing".

- *Distributed Caching und -Processing*: Daten werden über alle Gridknoten verteilt. Hierfür sind unterschiedliche Verteilungstopologien und -Strategien möglich. Ein Data-Grid kann in unterschiedliche, voneinander getrennte Sub-Caches unterteilt werden. Dies erlaubt effektivere Zugriffsmechanismen durch Vorfilterung. Die Verteilung der Daten über verschiedene physikalische Knoten hinweg garantiert eine dauerhafte Verfügbarkeit und Datenintegrität der Daten auch im Fehlerfall einzelner Knoten. Das automatische Failover-Verhalten und die Loadbalancing-Funktionalität sind Teil der Grid-Infrastrukturkomponente. Die Transaktionssicherheit auch über den kompletten Grid hinweg wird gewährleistet.
- *Event Driven Processing:* Funktionalität von Computational Grids. Rechenoperationen und Transaktionen können parallel über alle Gridknoten hinweg durchgeführt werden. Eine einfache Ereignisverarbeitung, ähnlich dem Triggermechanismus von Datenbanken, sorgt dafür, dass auf Datenänderungen reagiert werden kann. Einzeldaten im Grid können mit dem Konzept von „In-Memory-Views" und „In-Memory-Materialized Views" zu komplexeren Datenkonstrukten gejoint werden.

Grids ermöglichen folgende Aspekte zur Etablierung auch anspruchsvoller SLA's:

- Voraussagbare Skalierbarkeit (predictable scalability)
- Kontinuierliche Verfügbarkeit und Ausfallsicherheit (continuous availability)
- Bereitstellen einer Ersatzverbindung im Falle eines Serverausfalls (failover)
- Zuverlässigkeit (reliability)

Grids lassen sich hinsichtlich ihrer primären Funktionalität in Data Grid, In-Memory Data-Grid, Domain Entity Grid und Domain Object Grid differenzieren.

2.8.1.1 Data Grid

Data Grid ist ein System, das sich aus einer Komposition von mehreren, verteilten Servern ergibt, die als Einheit zusammenarbeiten, um auf gemeinsame Informatio-

nen zuzugreifen und gemeinsame, aber verteilte Operationen auf den Daten auszuführen.

2.8.1.2 In-Memory Data Grid

In-Memory Data Grid stellt eine Variante des Data Grid dar, bei dem die gemeinsame Information in einem verteilten (oftmals transaktionalen) Cache lokalisiert im Memory vorgehalten wird. Als verteilter Cache versteht man eine Datensammlung – oder besser: eine Objektsammlung, die über eine beliebige Anzahl von Clusterknoten verteilt (oder partitioniert) vorliegt, sodass genau ein Knoten im Cluster für eine spezifischen Datenanteil verantwortlich ist und diese Information der Verantwortlichkeit ebenfalls im Cluster verteilt vorliegt.

Konkurrierende Datenzugriffe werden von der Grid-Infrastruktur im Falle eines gewünschten transaktionalen Verhaltens, Cluster-weit behandelt. Der Vorteil liegt in der möglichen hohen Performance durch geringe Latenzen beim Speicherzugriff. Bei heutigen 64-Bit-Architekturen und den gefallenen Speichermodelpreisen können so auch größere Datenmengen im Memory vorgehalten und einem Low-latency-Zugriff zugänglich gemacht werden.

Übersteigt der Memory-Bedarf jedoch den verfügbaren Speicher, so können „Überlauf"-Strategien eine Auslagerung auf Platte (z.B. in das lokale Dateisystem oder in lokale Datenbanken) veranlassen, was dann allerdings Performance-Einbußen durch systembedingte höhere Latenzzeiten verursacht. Neuere Entwicklungen wie die Solid-State-Disk werden hier zukünftig einen sinnvollen und kostengünstigen Kompromiss ermöglichen und finden in solchen Szenarien hervorragende Einsatzgebiete.

Einem Datenverlust durch Ausfall eines Servers und damit dessen Memory-Anteil wird durch redundantes Verteilen der Informationen entgegengewirkt. Hierbei können produktabhängig sowohl unterschiedliche Verteilungstopologien als auch unterschiedliche Verteilungsstrategien gewählt oder ergänzt werden.

Im einfachsten Fall werden die Informationen gleichmäßig auf alle verfügbaren Server verteilt.

Ein In-Memory Data Grid verhilft der Applikation zu niedrigen Antwortzeiten, indem die Nutzdaten im Speicher in für die Applikation direkt nutzbaren Formaten vorgehalten werden, sodass Storage-Zugriffe mit niedriger Latenzzeit und aufwendige – somit zeitaufwendige – Transformationen und Aggregationen zum Zeitpunkt des Konsumentenzugriffs auf die Daten entfallen. Weil die Daten im Grid repliziert vorgehalten werden, können somit auch Datenbankausfälle gepuffert werden, und die Verfügbarkeit des Systems wird erhöht. Bei Ausfall eines Clusterknotens des Data-Grids sind die Daten noch auf mindestens einem weiteren Knoten verfügbar, sodass auch hierdurch die Verfügbarkeit erhöht wird. Nur ein Totalausfall führt zum Datenverlust, dem durch regelmäßige Zwischenspeicherung in persistenten Speichern (Festplatte, Solid-State Disk, ...) begegnet werden kann.

2.8.1.3 Domain Entity Grid

Domain Entity Grid verteilt die Domain-Daten des Systems (der Applikationen) über mehrere Server. Da es sich hierbei oftmals um grobgranulare Bausteine handelt, die hierarchisch aufgebaut sind, kann es vorkommen, dass ihre Daten aus mehreren unterschiedlichen Datenquellen extrahiert werden müssen, bevor die Komponenten im Grid clusterweit verfügbar gemacht werden. Das Data Grid übernimmt hierbei die Rolle eines Aggregators/Assemblers, der durch Pre-Population den Konsumenten einen clusterweiten, performanten Zugriff auf die aggregierten Entitäten liefert.

2.8.1.4 Domain Object Grid

Ein Domain Object Grid verteilt die Laufzeitkomponenten des Systems (der Applikationen) und deren Status (Prozessdaten) über eine Vielzahl von Servern. Dies kann sowohl aus Gründen der Ausfallsicherheit erforderlich sein, als auch wegen der parallelen Ausführung von Programmlogik. Durch Hinzufügen zusätzlicher Server lassen sich Applikationen somit horizontal skalieren. Die erforderlichen Informationen (Daten) für die parallelisierten Funktionen können aus einem gemeinsamen Datenspeicher (shared data storage) bezogen werden (wodurch dieser zentrale Zugriff möglicherweise zu einem Engpass wird, was die Skalierbarkeit des Gesamtsystems reduziert) oder aber direkt aus dem gleichen oder einem anderen Grid. Hierbei muss man auch die Möglichkeiten einzelner Produkte berücksichtigen oder z.B. mehrere Produkte (Data Grid und Computing Grid) miteinander kombinieren.

2.8.1.5 Verteilungs-Topologien

Es gibt unterschiedliche Verteilungs-Topologien und -Strategien (Replicated Cache und Partitioned Cache) (Misek, Purdy 2006).

Replicated Cache

Daten bzw. Objekte werden auf allen verfügbaren Knoten im Cluster gleichermaßen verteilt. Die bedeutet jedoch, dass das verfügbare Memory des kleinsten Servers limitierend wirkt. Ein solcher Knoten bestimmt dann, wie groß das verfügbare Datenvolumen sein kann.

Vorteile:

- Maximale Performance, auf allen Knoten gleiche Zugriffsperformance, da auf allen Knoten auf das lokale Memory zugegriffen wird, was man auch als „zero latency access“ bezeichnet.

Nachteile:

- Datenverteilung über alle Knoten bedeutet einen hohen Netzwerkverkehr und kostet Zeit. Gleiches gilt für Daten-Updates, die auf alle Knoten propagiert werden müssen.

- Das verfügbare Memory des kleinsten Servers bestimmt das Limit der Kapazität. Ein solcher Knoten bestimmt, wie groß das verfügbare Datenvolumen sein kann.
- Im Falle der Transaktionalität muss beim Setzen eines Locks auf einem Knoten jeder Knoten damit einverstanden sein.
- Im Falle eines Cluster-Fehlers können sämtliche vorgehaltene Informationen (Daten und Locks) verloren gehen.

Die Nachteile sind durch die Grid-Infrastruktur möglichst auszugleichen bzw. durch geeignete Maßnahmen zu umgehen. Dies sollte durch ein möglichst einfaches API transparent für den Programmierer erfolgen. Eine mögliche Implementierung kann so erfolgen, dass lesende Zugriffe (read-only) lokal erfolgen und eine Benachrichtigung der anderen Cluster-Knoten entfällt. Operationen mit zu überwachendem konkurrierendem Zugriff benötigen mindestens die Kommunikation mit einem weiteren Knoten. Update-Operationen erfordern eine Benachrichtigung aller Cluster-Knoten. Eine solche Implementierung resultiert in sehr hoher Performance und Skalierbarkeit mit gleichzeitigem transparenten Failover und Failback.

Es muss jedoch beachtet werden, dass replizierte Caches mit einer hohen Anzahl an notwendigen Daten-Updates nicht mit einem potenziellen Clusterwachstum (durch Hinzufügen zusätzlicher Knoten) linear skalieren, was mit einem zusätzlichen Kommunikationsaufwand pro Knoten einhergeht.

Partitioned Cache

Ein replizierter Cache adressiert die genannten Nachteile eines replizierten Caches hinsichtlich Memory- und auch Kommunikations-Aspekten.

Bei dieser Verteilungsstrategie müssen mehrere Aspekte in Erwägung gezogen werden:

- *Partitioned:* Die Daten werden so im Cluster verteilt, dass keine Verantwortungsüberschneidung hinsichtlich Datenbesitz entsteht. Genau ein Knoten ist für einen bestimmten Datenteil verantwortlich und hält diesen als Master-Datensatz. Dies hat unter anderem den Vorteil, dass die Größe des verfügbaren Speichers und die verfügbare Rechenleistung linear mit der Cluster-Größe wachsen kann. Des Weiteren ergibt dies, verglichen mit dem Replicated Cache, den Vorteil, dass alle Operationen, die auf den vorgehaltenen Objekten durchgeführt werden sollen, nur einen einzelnen „Netzwerk-Hop" erforderlich machen, d.h., zusätzlich zum Server, der die Masterdaten verwaltet, nur ein einzelner weiterer Server involviert werden muss (der die zugehörigen Backup-Daten für den Fall eines Failovers vorhält). Diese Art des Datenzugriffs auf Master und Backup ist extrem skalierbar, weil sie Point-to-Point-Verbindungen in einem „Switched-Network" optimal nutzen kann.
- *Load Balanced:* Verteilungsalgorithmen sorgen dafür, dass die Informationen im Cache optimal über die zur Verfügung stehenden Ressourcen im Cluster verteilt werden und somit über ein (für den Entwickler) transparentes Load-Balancing

verfügen. Bei manchen Produkten lassen sich die Algorithmen konfigurieren bzw. durch eigene Strategiemodule ersetzen. Je nach Verteilungs- und Optimierungsstrategie ergeben sich jedoch auch Nachteile dieses technischen Aspekts. Die dynamische Natur der Datenverteilung bedingt, dass es zu Datenumverteilungen durch Aktivierung der Optimierungsstrategie bei Zuschalten eines weiteren Cluster-Mitglieds kommen kann. Gerade in Umgebungen, wo eine hohe Volatilität temporärer Cluster-Mitglieder vorherrscht, sollten häufige Neuberechnungen der optimalen Verteilungscharakteristik und physikalische Datenumverteilungen mit resultierendem Netzverkehr vermieden werden. Dies lässt sich vermeiden, indem man volatile Clusterknoten als solche der Grid-Infrastruktur kenntlich macht und diese nicht in die Verteilungsstrategien integriert werden.

- *Location Transparency:* Obwohl die Informationen über die Knoten im Cluster verteilt vorliegen, wird für deren Zugriff die gleiche API verwendet, d.h. ein Programmierer greift für ihn transparent auf die Informationen zu. Anhand der API ist nicht erkennbar, und es ist auch nicht nötig, zu entscheiden, wo die Informationen im Cluster physikalisch lokalisiert sind. Es ist Aufgabe der Grid-Infrastruktur, die Datenverteilung dem Zugriffsverhalten optimal anzupassen. Heuristiken und Konfigurationen sowie austauschbare Strategien tragen hierzu bei. Sofern keine eigene Verteilungsstrategie erstellt werden muss, ist es unerheblich, wie diese Strategie im Hintergrund funktioniert.

2.8.1.6 Agenten

Unter Agenten versteht man autonome Programmteile, die, angestoßen durch eine Applikation, unter Kontrolle der Grid-Infrastruktur auf den im Grid vorgehaltenen Informationen ausgeführt werden. Je nach Produkt müssen hierfür bestimmte Klassen des Programmier-APIs erweitert bzw. implementiert werden oder deklarative Möglichkeiten erlauben die Erstellung solcher Agentenfunktionalitäten (z.B. mittels Aspekt-orientierter Techniken oder mithilfe von Präkompilierungsschritten). Oftmals sind schon vordefinierte Agenten bei den jeweiligen Produkten verfügbar.

2.8.1.7 Ausführungsmuster

- *Targeted Execution:* Agenten können gegen einen einzelnen Informationssatz im Data-Grid ausgeführt werden. Der Informationssatz wird hierbei durch einen eindeutigen Schlüssel identifiziert. Es ist Aufgabe der Grid-Infrastruktur, auf Basis der ihr verfügbaren Laufzeitdaten (z.B. Lastverhältnisse, Knotenauslastung, Netzwerkbelastung) den optimalen Ort der Ausführung im Cluster zu ermitteln.
- *Parallel Execution:* Agenten können gegen eine bestimmte, durch eine Anzahl von eindeutigen Schlüsseln identifizierbare Menge an Informationssätzen ausgeführt werden. Es ist Aufgabe der Grid-Infrastruktur, auf Basis der ihr verfüg-

baren Laufzeitdaten (z.B. Lastverhältnisse, Knotenauslastung, Netzwerkbelastung) den optimalen Ort der Ausführung im Cluster zu ermitteln.

- *Query-Based Execution:* stellt eine Erweiterung des Parallel Execution-Musters dar. Die Menge an betroffenen Informationssätzen wird nicht durch Angabe ihrer eindeutigen Schlüssel definiert, sondern durch Formulierung einer oder mehrerer Filterfunktion(en) in Form eines Query-Objektes.
- *Data-Grid Wide Execution:* Agenten werden parallel auf allen verfügbaren Informationssätzen im Grid ausgeführt. Dies entspricht einer Spezialisierung des Query-Based Execution-Musters, bei dem ein NULL-Query-Objekt übergeben wird, also eine nichtausschließende Filterbedingung.
- *Data-Grid Aggregation:* Zusätzlich zu den skalaren Agenten, können auch auf den Zielmengen clusterweite Aggregationen zur Durchführung von Berechnungen in (Nahe-)Echtzeit durchgeführt werden. Produkte liefern hierzu oftmals schon vordefinierte Funktionalität mit (Count, Average, Max, Min, etc.).
- *Node-Based Execution:* Agenten können auf bestimmten Knoten im Grid ausgeführt werden. Hierbei kann es sich um einen definierten einzelnen Knoten handeln. Agenten lassen sich aber auch parallel auf einem definierten Subset der verfügbaren Knoten oder auf allen Knoten des Grids ausführen.

2.8.2 Einsatzgebiete

Grid-Technologie kann in vielfältiger Weise in Architekturen zum Einsatz gelangen.

- *Verteilter, transaktionaler Daten-Cache (Domänen Entitäten):* Applikationsdaten können in einem verteilten Cache, linear skalierbar und mit transaktionalem Zugriff versehen vorgehalten werden.
- *Verteilter, transaktionaler Objekt-Cache (Domänen Objekte):* Applikationsobjekte (Geschäftsobjekte) können in einem verteilten Cache, linear skalierbar und transaktionsgesichert vorgehalten werden.
- *Verteilter, transaktionaler Prozess-Cache(Prozess-Status):* Prozessobjekte bzw. ihr Status können in einem verteilten Cache, linear skalierbar und transaktionsgesichert vorgehalten werden.
- *SOA-Grid:* Eine Spezialisierung des vorigen Szenarios. Hier werden BPEL-Prozesse in serialisierter Form (Hydration) clusterweit verteilt und können durch Deserialisierung (Dehydration) auf einem anderen Server weiter prozessiert werden. Dies resultiert in hoch-skalierbaren BPEL-Prozessen.
- *Datenzugriffs-Virtualisierung:* Ein Grid erlaubt den virtualisierten Zugriff auf verteilte Informationen in einem Cluster. Wie bereits erwähnt, besteht eine Ortstransparenz (Location Transparency) beim Datenzugriff unabhängig von der Größe des Clusters, die sich auch dynamisch ändern kann.

- *Datenspeicherzugriff-Virtualisierung:* Informationen werden im verteilten Cache, in dem für die Applikation passenden Format vorgehalten, unabhängig von der Art der Quellsysteme und ihrer Zugriffsprotokolle oder Zugriffs-APIs. Dies ist vor allem dort von Vorteil, wo die Informationen aus verteilten, heterogenen Quellsystemen bezogen werden müssen.
- *Datenformat-Virtualisierung:* Informationen werden im verteilten Cache, in dem für die Applikation passenden Format vorgehalten, unabhängig von den Formaten in den Quellsystemen. Dies ist vor allem dort von Vorteil, wo die Informationen aus verteilten, heterogenen Quellsystemen bezogen werden müssen.
- *Datenzugriffspuffer:* Der Zugriff auf Datenspeichersysteme (z.B. RDBMS) wird transparent für die Applikation gekapselt und gepuffert. Auf diese Weise können auch eventuelle Failover-Aktionen des Zielsystems (z.B. Oracle RAC) und notwendige Reaktionen von der Applikation entkoppelt werden. Eine Applikation muss somit nicht mehr auf Failover-Events diverser Zielsysteme reagieren können, da dies auf Ebene des Grids erfolgt.
- *Wartungsfenster-Virtualisierung:* Ein Data Grid unterstützt, wie bereits beschrieben, eine dynamische Cluster-Größe. Server können zur Laufzeit zum Cluster hinzugefügt und von ihm entfernt werden. Dadurch wird es möglich, verteilte Applikationen schrittweise zu migrieren, ohne größere Down-Zeiten der Applikation oder gar des gesamten Grids. Ein Server wird aus dem Cluster entfernt, auf diesem die Applikation migriert und der Server wieder zum Cluster hinzugeschaltet. Anschließend kann man mit jedem weiteren Server in gleicher Weise verfahren. Zukünftige Anwendungsentwicklungen, basierend auf OSGI, werden diese Problematik entschärfen.
- *Verteiltes Master-Data Management:* In hochlastigen Umgebungen kann es bei zentralen Master-Data-Applikationen zu nicht tolerierbaren Engpässen kommen. Hier lässt sich zwar Abhilfe durch klassische Datenreplikation schaffen, was jedoch mit gewissen Aufwänden verbunden ist und vor allem in einem (Nahe-)Echtzeit-Umfeld nicht tolerabel ist. Eine Verteilung der Master-Daten in einem Data-Grid bringt Abhilfe, sofern der Speicher ausreicht.
- *High Performance Backup & Recovery:* Es ist vorstellbar, dass langläufige Backup-Läufe zur Performance-Steigerung mehrschrittig konzipiert werden und schrittweise zuerst in einen In-Memory-Cache schreiben, welcher zeitverzögert in einen persistenten Speicher schreibt.
- *Notification-Service in einem ESB:* Grid-Technologie ersetzt ein Nachrichtenbasiertes System zur Notifikation in einem Service Bus.
- *Complex Realtime Intelligence:* kombiniert die Funktionalitäten von CEP (complex event processing) und Data-Grids und bietet so die Voraussetzung, hochskalierbare Analyse-Applikationen für komplexe Mustererkennungen in Echtzeitszenarien dem Business zur Verfügung zu stellen. Im Grunde genommen handelt es sich hierbei im einfachsten Fall um eine Event-driven Architecture

mit CEP-Engines als Consumern, wobei der Nachrichtentransport sowie eine Voranalyse und Vorfilterung feingranularer Einzelereignisse mit Gridtechnologie erfolgen. Die Infrastrukturkomponenten des Grids sorgen auch für ein Loadbalancing, Ausfallsicherheit und Verfügbarkeit von historischen Daten aus Data Marts im In-Memory-Cache. Die Kombination aus Grid und CEP erlaubt es, mit einfachen Mitteln höchstskalierbare, aber dennoch einfach wartbare Auswertungsarchitekturen für (Nahe-)Echtzeit-BI zu schaffen.

2.8.3 XTP (Extreme Transaction Processing)

Im Zuge der Anforderung zur – vor allem auch komplexen – Prozessierung großer und sehr großer Datenmengen (z.B. im XML-Umfeld, Import von großen Dateien mit Formattransformationen, etc.) etablierten sich in den letzten Jahren neue Architekturen für verteilte Speicherarchitekturen mit parallelen Applikationszugriffsmöglichkeiten.

Hierbei handelt es sich um eine Reihe unterschiedlicher, Plattform-übergreifender Produkte und Lösungen, die unter dem Begriff „eXtreme Transaction Processing" – (XTP) zusammengefasst werden. Der Begriff wurde durch die Gartner Group geprägt und steht für einen Architekturstil, der das Ziel verfolgt, sichere, hochskalierende und hochperformante Transaktionen über verteilte Umgebungen hinweg auf Commodity-Hardware und -Software zu ermöglichen.

Es ist davon auszugehen, dass solche Lösungen eine zunehmende Rolle in serviceorientierten und Ereignis-gesteuerten Architekturen einnehmen werden. Interoperabilität ist dabei eine treibende Komponente für XTP.

Verteilte Caching-Mechanismen und GRID-Technologien mit einfachen Zugriffs-APIs bilden dabei (im Gegensatz zu den in der Vergangenheit im wissenschaftlichen Umfeld verbreiteten, komplexen Produkten) die Basis für einen problemlosen und erfolgreicheren Einsatz. Während verteilte Cache-Produkte bereits eine große Rolle im high-end transaction processing (=Forrester Research-Ausdruck) spielten, ist eine zunehmende Positionierung im aufkommenden Information-as-a-Service (IaaS)-Markt zu erwarten.

Im Umfeld von Finanzdienstleistern lassen sich seit einiger Zeit neue Strategien der Geschäftspriorität erkennen. Finanzinstitute versuchen, die Grenzen ihrer vorhandenen Hardware-Ressourcen zu überschreiten, um immer leistungsfähigere Applikationen zu entwickeln, ohne in einen exponentiellen Anstieg ihrer Hardware- und Energiekosten investieren zu müssen.

Das Wachstum von XTP in Unternehmensfeldern wie fraud detection, risk computation und stock track resolution lässt bestehende Systeme an ihre Leistungsgrenzen stoßen. Neue Systeme, die diese anspruchsvollen Funktionalitäten umsetzen können, verlangen nach neuen Architekturparadigmen.

Man kann verfolgen, dass ein Verbund von SOA, EDA und XTP die Zukunft der Finanzdienst-Infrastrukturen darstellt, um die Ziele komplexer Berechnungen mit sehr großen Datenmengen unter Echtzeitbedingungen zu erreichen. XTP gehört zu einer speziellen Klasse von XTP-Applikationen (extrem transaction processing platform), die große Datenvolumina hoch performant und mit hohem Durchsatz verarbeiten, aggregieren und korrelieren oder in anderer Art und Weise bearbeiten. Typischerweise entstehen bei diesen Prozessen eine große Anzahl an einzelnen Ereignissen, die als hochvolatile Daten verarbeitet werden müssen. Applikationen im XTP-Stil gestatten, dass Transaktionen und Berechnungen üblicherweise direkt im Speicher der Applikation (In-Memory) durchgeführt werden und nicht auf komplizierte Remote-Zugriffe auf Backend-Services angewiesen sind, um so Kommunikations-Latenzen zu vermeiden (low latency computation). Dies ermöglicht extrem schnelle Antwortzeiten bei gleichzeitiger gesicherter transaktionaler Integrität der Daten.

SOA Grid („next generation Grid enabled SOA“) stellt konzeptionell eine Variante der XTPP („eXtreme Transaction Processing Platform“) dar. SOA-Grid ermöglicht eine „state-aware“, kontinuierliche Verfügbrkeit für eine Dienste-Infrastruktur, Applikationsdaten und Prozesslogik. Sie basiert auf einer Architektur, die horizontal skalierbare, Datenbank-unabhängige Middle-Tier-Caching-Lösungen mit intelligenter Parallelisierung kombiniert und so Prozesslogik und gecachte Daten für geringe Latenzzeiten aneinander bindet („data and process affinity“). Dies ermöglicht neue, einfachere und effizientere Modelle für hochskalierbare, SOA-basierte Applikationen, die jedoch in vollem Umfang von den Möglichkeiten Ereignis-gesteuerter Architekturen profitieren können.

2.8.4 XTP und CEP

XTP und CEP sind vergleichbar, indem sie eine große Anzahl an Ereignisdaten konsumieren und korrelieren müssen, um daraus aussagekräftige Antworten zu generieren.

Häufig übersteigt hierbei die zu konsumierende Ereignismenge bei weitem die Verarbeitungsmöglichkeit konventioneller Storage-Mechanismen („ there isn’t just a disk that can spin fast enough“). In solchen Fällen können die Daten in einem Grid gespeichert werden. CEP-Engines lassen sich auf diese Daten verteilen und parallel zugreifen, Auswertungen sind möglich, und man ist in der Lage, Geschäftsereignis-relevante Muster in Echtzeit zu finden und zu analysieren. Letztere können anschließend mittels BAM (business activity monitoring) weiterverarbeitet und ausgewertet werden.

2.8.5 Solid State Disks und Grids

Die Solid-State-Disks (SDD)-Technologie entwickelt sich rasant. Die Datenkapazitäten steigen rapide, und die I/O-Raten sind verglichen mit konventionellen Laufwerken phänomenal. Das Preis-/Leistungsverhältnis von Kosten pro GB Speicher stellte bislang das größte Hindernis für einen flächendeckenden Einsatz dar. Aktuell beträgt er ca. Faktor 12x der Kosten einer normalen Serverdisk pro GB Speicher. Der größte Vorteil im Datacenter liegt im enorm niedrigen Energieverbrauch, welcher signifikant unter dem konventioneller Disklösungen liegt.

SDDs sind daher aufgrund ihrer niedrigen Energieanforderungen und ihrer hohen Performance durch niedrige Latenzzeiten und bei zu erwartenden fallenden Kosten attraktiv in Blades oder dichten Racks. Es stellt sich die Frage, welchen Einfluss SDDs auf Data-Grid-Technologie haben können und evtl. werden.

Diskbasierte XTP-Systeme können von einem SDD-Laufwerk profitieren. Allerdings speichern heutzutage SDDs wesentlich weniger Daten (128GB vers. 1TB) als gängige Disks. Dies ist allerdings mehr als die Kapazität von Standard-Hauptspeicherelementen, und sie sind auch günstiger pro GB als Hauptspeicher. Die Kapazität von SDDs liegt ungefähr um Faktor 10 unter einer konventionellen Disk, aber um Faktor 8 höher als die Kapazität von möglichem Hauptspeicher.

SDDs füllen somit die Kluft zwischen strikten Hauptspeicher-basierten und strikt Disk-basierten XTP-Architekturen. SDDs-basierte Architekturen sind zwar etwas langsamer als Memory-basierte, allerdings um Faktoren schneller als die schnellsten Disk-basierten Systeme. Es liegt daher nahe, in XTP-Systemen eine hierarchische Storage-Architektur vorzusehen, in welcher die volatilsten Daten im Memory vorgehalten werden, und weniger oft gebrauchte Daten in SDDs und Langzeitpersistenz in klassischen Disk-basierten Storage-Technologien. Außerdem erscheint es sinnvoll, Memory-Überläufe für Memory-basiertes Caching auf SDDs zu speichern.