Vergleich wichtiger Komponententechnologien bezüglich ihrer Eignung in SOA

Harry Trautmann
Hochschule für Technik, Stuttgart

20. Oktober 2007

Inhaltsverzeichnis

1 Einleitung

In den vergangenen Jahrzehnten stieg sowohl Rechenleistung, als auch Speicherkapazität von Computersystemen. Ebenso wuchs mit der Nachfrage nach geeigneter Informationstechnologie die Abhängigkeit der Gesellschaft davon. Um dieser Nachfrage gerecht zu werden, mussten Systemarchitekten immer gröber granulierte Automatisierungs- und Kapselungstechnologien einführen. Gäbe es derartige Standards heute nicht, könnten beispielsweise moderne Betriebssysteme wie Windows XP nicht annähernd in der heute vorliegenden Qualität realisiert werden. Daher ist leicht einsichtig, dass Modularisierung eher zum Erfolg führt, wenn einigermaßen groe Computersysteme erstellt werden müssen. Diese Modularisierung begann auf prozeduraler Ebene (Funktionen und Funktionsbibliotheken) und führte in Form der Objektorientierung zu modularisierten Daten, um schließlich das Paradigma von semantisch in Komponenten gekapselter Software zu begründen. Heute existieren mehrere Standards für so genannte Komponententechnologien nebenher.

Moderne Bestrebungen, welche darauf abzielen, Geschäftsabläufe und die dabei benötigten Daten wiederverwendbar und zueinander kompatibel zu machen, führen die Modularisierung auf einer abstrakteren Stufe sogar noch weiter. Diese Service-orientierten Architekturen (SOA) stellen momentan die gröbste Granularitt an Modularisierung dar. Analog zu den Komponententechnologien, die in den 90er Jahren ausreiften, befindet sich die SOA-Technologie derzeit in einer Konsolidierungsphase. Abseits der Webservices, welche derzeit die bekannteste SOA-Anwendung darstellt, stehen Umsetzungen noch weitgehend aus.

Im Folgenden soll die Verbindung zwischen den wichtigsten der nunmehr etablierten Komponententechnologien und der noch neuen SOA-Technologie untersucht werden. Aufgeworfene Fragestellungen beziehen sich vor allem darauf, inwieweit die einer Architektur jeweils unterliegende Komponententechnologie die Charakteristika von SOA von sich aus untersttzt und wie flexibel die Technologie auf Entwicklungen in der noch jungen SOA-Historie reagieren kann. Insbesondere soll die Enterprise Java Beans Technologie betrachtet werden, um nachzuweisen, dass sie aktuell am geeignetsten ist, um mit ihr SOA-Systeme zu realisieren.

2 Umfeld

2.1 Komponententechnologien

Eine Komponententechnologie spezifiziert nach [Andresen2004, 261] die Syntax und Semantik der Komponenten und stellt eine Laufzeitumgebung bereit, die über gewisse Basisdienstleistungen (z. B. Erzeugung, Aktivierung, Deaktivierung von Komponenten), sowie erweiterte Dienste, wie Persistenzmanagement, Transaktionsverhalten, Sicherheit, Verzeichnisdienste, verfügt. Die Technologie kann auf spezifischen Applikations-Server und -Container, als auch auf spezifischer Middleware basieren.

In den letzten 10 Jahren wurden diverse Standards für Komponententechnologien entworfen. Doch haben sich nur wenige davon in breitem Maße durchgesetzt. Die drei wichtigsten sind CCM, EJB und .NET.

2.1.1 CORBA Component Model

CORBA ist ein plattform- und sprachunabhängiger Middleware-Standard und wurde von der Object Management Group (OMG) erschaffen, welche damals von ca. 800 Firmen gebildet wurde. CCM (CORBA Component Model) ist ein auf CORBA aufgesetzter Komponentenstandard. Während CORBA die semantischen Zusammenhänge von Programmmodulen im Großen beschreibt, ist CCM ein konkretes Modell für die Programmierung. CCM regelt die Definitionen der Komponentenschnittstellen, -lebenszyklus, -zustände und -verträge. Desweiteren bietet CCM Funktionalitäten zur Verpackung und Verteilung von Komponenten. [Andresen2004, 270]

Grundlegend für CORBA ist der ORB (Object Request Broker). Dies ist eine Schicht, welche die notwendigen Basis-Services abstrahiert. Der ORB stellt den Komponenten und Komponenten- Containern Transaktions-, Sicherheits-, Benachrichtigungs- und Persistierungsdienste zur Verfügung. [Andresen2004, 274]
CCM-Komponenten sind stets in eine Container-Struktur eingebettet. Diese Struktur bildet für die Komponente die Laufzeitumgebung und verwaltet ihren Lebenszyklus. Das heißt, dass der Container die konkrete Erstellung und Freigabe der Komponente veranlasst, sowie Anfragen an die API der Komponente und Anfragen der Komponente an externe Komponenten-APIs ausführt. Dabei verwaltet der POA (Portable Object Adapter) die Referenzen auf externe (evtl. entfernte) Instanzen von CORBA-Komponenten. Das „Component Home“-Interface ist ein vom ORB verwaltetes Objekt, welches dazu dient, die Instanziierungen und Einbettungen von Komponenten zu kontrollieren. Da mehrere unterschiedliche Implementierungen des CORBA-Standards existieren, wurde der POA ab CORBA 2.0 notwendig, um instanzierte Komponenten zwischen den unterschiedlichen Implementierungen portabel zu halten. [Qian2005, 146]
CCM-Komponenten stellen sich nach aussen über 4 Arten von Schnittstellen dar.

1. Facets: sie sind die komponentenexterne Schnittstelle für die Business Logic der Komponente.

2. Receptables: sie sind nach aussen veröffentlichte Referenzen auf externe Komponenten/Systeme.

3. Event Sources: sie dienen der Veröffentlichung von in den Komponenten erzeugten Ereignissen.

4. Event Sinks: sie sind Empfänger von komponentenextern erzeugten Ereignissen.

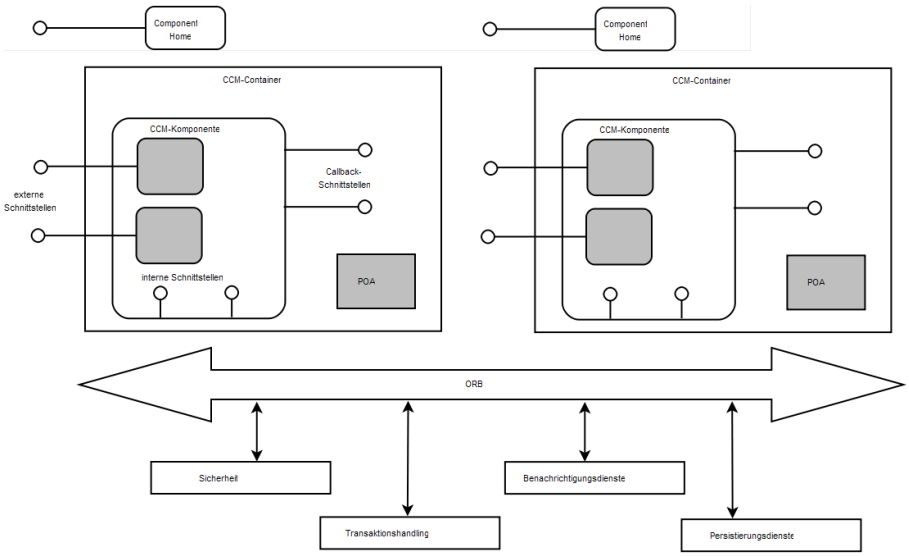

Abbildung 1: Einbettung von Komponenten in einer CCM-Architektur [Andresen2004, 275]

2.1.2 .NET

Das .NET Framework wurde von der Microsoft Corporation entwickelt und 2000 veröffentlicht. Es dient der schnellen Erstellung und Bereitstellung von Softwarekomponenten. Momentan gibt es nur für die Microsoft Windows Betriebssysteme Implementierungen des .NET Frameworks.

Die Basis des Frameworks bildet der CLR-Dienst (Common Language Runtime). Es ist eine virtuelle Maschine, die sogenannte Assemblies ausführen kann. Dies wiederum sind vorkompilierte, in dem Bytecode-Format MSIL (Microsoft Intermediate Language) vorliegende Binärprogramme.

Ausserdem enthält das .NET Framework eine Klassenbibliothek mit grundlegenden Klassen, sowie grundlegende Dienste zur Veröffentlichung von Funktionalität über das Web (ADO.NET) oder als Windows Desktop Applikation (Windows Forms).

.NET Komponenten sind selbstbeschreibende Assembly Dateien. Dabei kann eine Komponente wiederum aus mehreren weiteren in MSIL vorliegenden Modulen bestehen und andere Komponenten, sowie Klassen der .NET Klassenbibliothek referenzieren. Eine .NET Komponente besteht aus den folgenden vier Teilen [Qian2005, 200]:

1. einem Manifest (enthält u. a. Versionsnummer, einen eindeutigen Namen und Definitionen der verwendeten Typen),

2. Metadaten der beteiligten Module,

3. IL Code der Module und

4. Ressourcen (wie beispielsweise Bilddateien).

Wenn .NET Komponenten ausgeführt werden sollen, lädt zunächst der CLR-Class Loader das entsprechende Assembly und die evtl. referenzierten Klassen aus der .NET Klassenbibliothek. Danach erzeugt der JIT Compiler sogenannten „Managed Native Code“. Das sind in Maschinensprache des jeweiligen Betriebssystem, auf dem die Komponente ausgeführt werden soll, vorliegende binäre Anweisungen, die vorher auf Sicherheit hinsichtlich Speicherzugriffen, Threadabläufen und Systemschutz geprüft wurden. Die CLR-Ausführungseinheit führt den Managed Native Code aus.

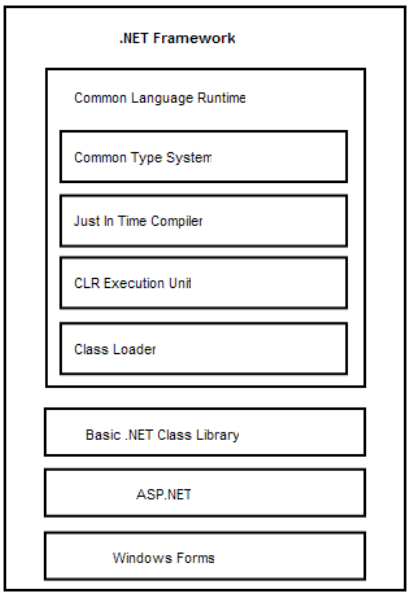

Abbildung 2: Aufbau des .NET Frameworks [Qian2005, 196]

2.1.3 Enterprise Java Beans

Die EJB-Komponententechnologie ist eng an die J2EE-Plattform gebunden. Die J2EE-Plattform wurde primär für die Entwicklung von Unternehmensdiensten (Enterprise Services) entworfen und bietet den EJB per Design essentielle Dienste, wie beispielsweise Transaktionsmanagment, Persistenzmanagement und Lifecycle Management an. Somit können bei EJB-Implementierungen diese Aspekte ausser Acht gelassen werden, so dass sich der Code voll auf die Umsetzung der Geschftslogik konzentriert. Eine J2EE-Architektur wird im Allgemeinen in die folgenden semantischen Bereiche (die sogenannten Tiers) aufgegliedert.

1. Client-Tier: diese Ebene enthält die Funktionalität zur Interaktion des Benutzers mit dem System. Dabei kann der Benutzer beispielsweise ein Browser oder eine Java Applikation sein.

2. Web-Tier: hierin wird die Information aufbereitet, die entweder dem Business-Tier entnommen wird oder vom Client erhalten zum Business-Tier weitergeleitet werden soll.

3. Business-Tier: die Geschäftslogik des Systems befindet sich im Business-Tier. EJBs gehören in diesen Tier.

4. EIS-Tier: dieser Tier dient der Anbindung des J2EE-Systems an „Enterprise Information Systeme“. Das können beispielsweise externe Datenbanksysteme oder Host-Anwendungen sein.

EJBs können erst zusammen mit einer Laufzeitumgebung genutzt werden. Der EJB-Server stellt eine solche zur Verfügung. Der EJB-Server bietet den EJBs grundlegende Dienste wie Prozess- und Thread-Verwaltung an, aber auch komplexe, wie beispielsweise einen Dienst für das Pooling der von mehreren EJBs gemeinsam genutzten Netzwerk- und Datenbankenressourcen.

Auch EJBs werden innerhalb Containern ausgeführt. Sie abstrahieren die EJB-Schnittstellen nach aussen, so dass externe Zugriffe auf die EJB-Schnittstelle vom Container kontrolliert werden. Ausserdem bietet der Container einer EJB-Komponente die Implementierung allgemeiner Funktionalitäten wie Sicherheits-, Konstruktions- und Destruktionsmechanismen an. Dadurch kann der Code der EJB in noch höherem Maß auf die eigentliche Geschäftslogik konzentriert werden. Die Container werden über die J2EE-Plattform oder von J2EE- konformen Tools bereitgestellt.

EJB-Container verfügen über sogenannte Connector-Objekte, die dazu dienen, J2EE-externe Systeme des EIS-Tiers an die J2EE-Architektur anzubinden. [Qian2005, 269]

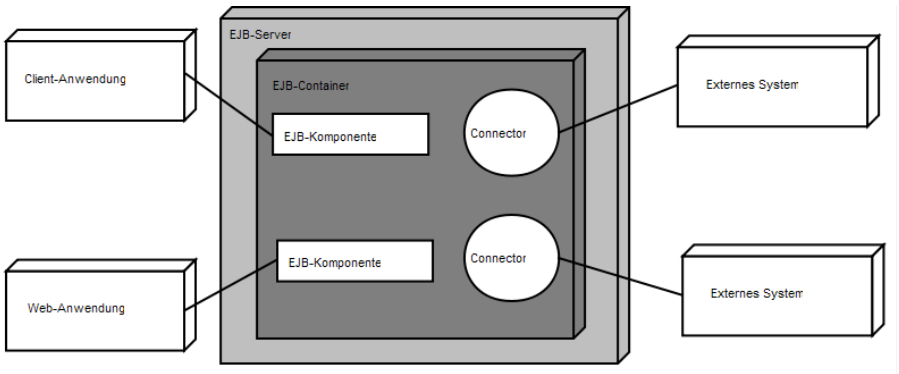

Abbildung 3: Einbettung von Komponenten in eine EJB-Architektur [Andresen2004, 269]

2.2 Service-oriented Architecture

Mit der Etablierung verteilter Computersysteme ergaben sich zwei Hauptprobleme: Heterogene Standards (Proprietät) und enge Kopplung existierender Systemmodule aneinander. Dies erzeugte die Problematik inkompatibler und schwer zu wartender Systeme. Moderne Anforderungen, wie z. B. global benötigte Verfügbarkeit von Programmfunktionalität und notwendige Kosteneinsparung bei der Systemwartung, führten dazu, vor allem die vorhandenen Systemdienste mit einer offenen Schnittstellenschicht auszustatten, um ihre Funktionalität in anderen Systemmodulen wiederverwenden zu können. Dies führte zu Architekturen, deren Aufbau sich an den zur Verfügung stehenden Diensten ausrichtet.

2.2.1 Definition

Derzeit existiert noch keine einheitliche Definition des Begriffs „Service-orientierte Architektur“. Entlang [Gartner2007, 32] wird er im Folgenden verstanden als Stil einer Anwendungsarchitektur, bei der die von der Anwendung über Metadaten bereitgestellten und von der Implementierung getrennten Schnittstellen der unterliegenden Geschäftsprozesse wiederholt von unterschiedlichen Konsumenten aufrufbar sind.

2.2.2 Charakteristika

Um die Eignung einer Komponententechnologie für denm Einsatz in SOA beurteilen zu können, muss zuvor eine Charakterisierung der Eigenschaften von SOA vorgenommen werden. Dann können die Eigenschaften der jeweiligen Technologie dagegen verglichen werden.

- Fokus auf Geschäftsprozesse: jede durch eine SOA bereitgestellte Schnittstelle repräsentiert einen Geschäftsprozess in Form einer Black Box. Komponententechnologien, welche eine Trennung von Layout und Geschäftslogik begünstigen, sind daher von Vorteil.

- Lose Kopplung von Diensten: Dienste sind lose gekoppelt, wenn ihre Kommunikation auf nicht-proprietären offenen Standards basiert. Änderungen in der Zusammenstellung der Dienste wirken sich dann weniger gravierend auf die Architektur aus. Diese standardisierte Interoperabilität [Taylor2006, 52] ermöglicht die sogenannte Service-Orchestrierung - die Erstellung eines neuen Dienstes, der ausschließlich bestehende Dienste verwendet. Somit ist eine Komponententechnologie besser für SOA geeignet, je mehr sie die Verwendung von offenen Standards fördert.

- Gleichzeitige Bedienung vieler Konsumenten: SOA werden vornehmlich im Umfeld der Web Economy eingesetzt, in dem es für ein System eine grundlegend zu erfüllende Anforderung ist, viele gleichzeitig gestellte

Konsumentenanfragen zu befriedigen [Becking2007]. Daher ist eine Komponententechnologie besser für SOA geeignet, wenn sie über gute Skalierungseigenschaften verfügt.

2.2.3 Webservices als SOA-Killerapplikation

Web Services sind eine Spezialisierung von SOA. [Samaschke2006] beschreibt einen Web Service als eine über eine URI (Uniform Resource Locator) adressierbare Software-Komponente, deren Schnittstelle mittels den standardisierten Web-Protokollen HTTP, HTTPS, SMTP und/oder FTP angesprochen werden kann. Für den Austausch von Daten wird XML in Form von WSDL und SOAP verwendet. Ein UDDI-Komponentenverzeichnis (Universal Description, Discovery and Integration) ist optional.

Das .NET Framework bietet Entwicklern viele Hilfen an für die Entwicklung von Web Services, beispielsweise indem es mit ASP.NET entsprechende für die Verwendung in Web Services vorgefertigte Klassen bereit stellt (siehe Abbildung 2).

Dennoch sollten Dienste anderen Applikationen allerdings nicht nur mittels WSDL (Web Service Definiton Language) als Webservice-Schnittstelle zur Verfügung stehen, sondern auch möglichst breit andere Standardschnittstellen abdecken, wie z. B. JCA (J2EE Connector Architecture), XML, EDI (Electronic Data Interchange) und SOAP (Simple Object Access Protocol), sowie Kombinationen davon. Sonst besteht die Gefahr, dass die erstellte Architektur nur einseitig nutzbar ist, statt von beliebigen Konsumenten genutzt werden zu können.

3 Entsprechung gegenüber SOA-Charakteristiken

3.1 Trennung zwischen Geschäftsprozess und dessen Präsentation

Um eine SOA wiederverwendbar zu halten, muss eine möglichst weitgehende Trennung zwischen der Geschäftslogik und deren Präsentation seitens der Technologie unterstützt werden.

- Die J2EE-Plattform trennt mittels unterschiedlicher Container-Klassen zwischen den EJB, welche für die Implementierung der Geschäftslogik gedacht sind, und dem Layout, das durch JSPs oder Servlets realisiert werden kann.
- CCM schreibt keine derartige Trennung vor.
- .NET schreibt keine Trennung vor, unterstützt dies aber mit der Bereitstellung der ASP.NET-Klassenbibliothek.

3.2 Skalierbarkeit

Von besonderer Bedeutung ist es für eine SOA, dass sie eine große Anzahl von Konsumentenanfragen bedienen kann. Um die Systemressourcen zu schonen, ist es wichtig, dass eine Komponententechnologie, aus der sich die SOA zusammensetzt, möglichst gut skaliert.

- EJB-Server verfügen über Mechanismen für des Pooling von Datenbankverbindungen und EJB-Instanzen, sowie Caching derselben. Somit ist die EJB-Technologie hochgradig skalierbar. [Andresen2004, 288]
- Skalierung kann in CCM durch Verwendung unterschiedlicher Container erreicht werden [Andresen2004, 289]. Dies verlangt also, dass bereits zur Compile-Zeit bekannt sein muss, wie das System skalieren muss. Soll zu einem späteren Zeitpunkt auf einen flexibleren Container gewechselt werden, muss ein neues Deployment stattfinden. Insofern verfügt CCM prinzipiell über Skalierungseigenschaften, die aber nicht flexibel verwendbar sind, wenn sich die Systemarchitektur ändert.
- Innerhalb des .NET Framework sorgen die ASP.NET- und ADO.NET-Module für Caching von Datenbankverbindungen und Statusinformationen der Instanzen. Wie gut eine .NET Architektur skaliert, hängt aber vor allem auch von der Konfiguration der dahinterliegenden Back-End-Server ab [Andresen289].

3.3 Verteilung

Eine grundlegende Eigenschaft von SOA ist ihre Flexibilität hinsichtlich der Komposition miteinander. Dabei kommt es zwangsläufig dazu, dass Komponenten unter geänderten Parametern verwendet werden müssen. Daher ist es von Vorteil, wenn ein Komponentenstandard die einfache Neukonfiguration einer Komponente unterstützt.

- EJBs werden in JAR-Dateien verpackt, die auch einen sogenannten Deployment-Descriptor mit umfangreichen Einstellmöglichkeiten enthält. Der Deployment-Deskriptor enthält u. a. die Versionierungsinformationen und kann auch nach Veröffentlichung noch geändert werden, ohne dass die Komponente neu erstellt werden muss [Andresen2004, 291]. Dies stellt innerhalb SOA einen Vorteil dar.
- Wenn eine neu CCM-Komponente bereitgestellt werden muss, reicht es, deren in IDL (Interface Definition Language) spezifizierte Schnittstellen um die neuen Schnittstellen zu erweitern und die neu kompilierte Komponente in das System einzubauen [Andresen2004, 233]. Somit kann auch ein System, das auf CCM-Komponenten aufbaut, einfach umkonfiguriert werden.
- .NET unterstützt die Berücksichtigung der Verteilung von Komponenten während ihrer Zusammenstellung (Component Assembly Deployment).

Dadurch können mehrere Versionen von Komponenten desselben Namens nebenher ohne Konflikt koexistieren. [Qian2005, 195]

4 Offenheit gegenüber zukünftigen Entwicklungen

Die folgenden Kriterien sind für die Eignung der jeweiligen Komponententechnologien hinsichtlich SOA nicht entscheidend. Sie stellen aber generell wünschenswerte Eigenschaften dar, denn einmal vorliegende Architekturen tendieren dazu, erweitert zu werden, sowie die Anforderungen und das Umfeld dazu tendieren, sich in unerwarteter Weise zu ändern.

4.1 Plattformunabhängigkeit

Plattformunabhängigkeit meint hier die Loslösung vom unterliegenden Betriebssystem, nicht die Loslösung vom unterliegenden Middleware-Standard, der ja ebenfalls als Plattform bezeichnet wird.

- Die EJBs sind durch Verwendung von Java als unterliegende Sprache prinzipiell plattformunabhängig, da die Implementierung einer JVM für moderne Betriebsysteme mandatorisch ist.
- Auch CCM ist prinzipiell plattformunabhängig, denn eines der wesentlichen Ziele beim Entwurf des CCM unterliegenden Middleware-Standards CORBA war es, für CORBA-Architekturen Abhängigkeit vom Betriebssystem zu eliminieren.
- Das .NET Framework ist derzeit nur auf den Microsoft Windows Betriebssystemen implementiert. Somit sind .NET Komponenten plattformabhängig.

4.2 Sprachunabhängigkeit

- EJBs können nur in Java programmiert werden und setzen die Verwendung von Klassen aus der J2EE voraus. Damit ist die EJB-Technologie in hohem Maß sprachabhängig.
- CCM-Komponenten setzen keine bestimmte Programmiersprache voraus.
- .NET-Komponenten können in einer Vielzahl von Sprachen erstellt werden. Derzeit werden die meisten Komponenten in VB 6.0, VB 7.0, VC.NET und C geschrieben, da dies die ersten Sprachen waren, für die es IL Compiler für das .NET Framework gab.

4.3 Herstellerbindung

Die Abhängigkeit einer Technologie vom Hersteller hat zwei Seiten. Zum einen stellt dies die Anhängigkeit von Plänen dar, die der Hersteller in Zukunft mit der Technologie verfolgt (z. B. Unterstützung, Erweiterungen). Zum anderen bedeutet das aber auch, dass diese Pläne zweitnah umgesetzt werden können, statt dass sich mehrere Parteien zuerst über ein Vorgehen einig werden müssen. Insofern muss vor Verwendung der jeweiligen Technologie abgewogen werden, ob eher die Unabhängigkeit von einem Hersteller oder ein beschleunigter Reifeprozess der Technologie als wichtiger zu bewerten ist.

- EJB benötigt die J2EE-Plattform. Diese wird aber von mehreren großen Unternehmen (z. B. Sun Microsystems, Oracle, JBoss Inc.) angeboten, wodurch die EJB-Technologie herstellerunabhängig ist.
- CORBA, wie auch CCM, wurden von einem Zusammenschluss aus mehreren hundert Firmen entworfen. Somit ist CCM nicht spezifisch von einem Hersteller abhängig.
- .NET-Komponenten sind mitsamt dem .NET Framework an Microsoft Betriebssysteme gebunden. Allerdings pflegt Microsoft den Standard sorgfältig, beispielsweise, indem es mit Updates auf Sicherheitslücken in der Framework-Implementierung reagiert.

4.4 Eignung für SOA-Middleware

Momentan gibt es noch keine vorliegende SOA-Middleware, lediglich einige Ankündigungen. Beispielsweise hat Sun CAPS (Composite Application Platform Suite) angekündigt, eine integrierte Software Suite für SOA-Applikationen. Aufgrund der gegebenen Affinität von Sun Microsystems zur Java-Programmiersprache ist insbesondere die EJB-Technologie für einen Einsatz in CAPS prädestiniert [Sun2007].

Weiterhin arbeitet die Firma iWay mit mehreren großen Unternehmen, wie z. B. BEA, Microsoft und Sun, zusammen um einen SOA Middleware Standard zu realisieren, der über eine möglichst breit gefächerte Unterstützung von Technologien verfügt, darunter auch das .NET Framework [iWay2007].

5 Weitere günstige Eigenschaften

Im Folgenden sollen einige Eigenschaften der betrachteten Technologien untersucht werden, die zwar nicht notwendig sind, deren Vorliegen aber die Eignung für SOA begünstigt. Zumeist lassen sich aufgrund dieser Eigenschaften die üblichen SOA-Einsatzzwecke leichter umsetzen.

5.1 Sicherheitsmechanismen

SOA schreibt keine besonderen Sicherheitsmaßnahmen vor. Sowohl Konsumenten, als auch Dienstleister sind selbst in der Pflicht, für geeigneten Schutz zu

sorgen. Insofern wird jeglicher von einer Komponententechnologie bereitgestellter Sicherheitsmechanismus als positiv bewertet.

- Die EJB-Technologie nutzt die J2EE-Sicherheitsmechanismen, welche auf denjenigen des darunterliegenden Betriebssystems aufbauen. Über Rollen kann der Zugriff bis auf Methodenebene eingeschränkt werden [Andresen2004, 289].
- CCM verfügt über ein ähnliches Rollen-System wie die EJB-Technologie basierend auf CORBA.
- Beim .NET Framework wird zwischen Authentifizierung, Autorisierung und Personifizierung unterschieden. Um eine Zugriff auf das System zu erlangen, muss ein Dienstkonsument erst jede dieser Stufen durchlaufen.

5.2 Kostentransparenz

Oft soll mittels einer SOA die Kostentransparenz eines Systems erhöht werden. Man soll nachvollziehen können, welche Teile des Systems wieviel der vorhandenen Ressourcen in Anspruch nehmen. Insbesondere ist dies dann wichtig, wenn extern veröffentlichte Dienste Kunden berechnet werden sollen [Koll2007].

- Das J2EE-Konzept unterstützt weitgehende Trennung zwischen dem eigentlichen Geschäftsprozess und dessen Präsentation. Somit fällt es bei Verwendung der EJB leichter, die Kosten zu berechnen.
- Das CCM ist ein sehr fein granulierter Komponentenstandard. Die Verwendung von externen Klassenbibliotheken ist nicht vorgeschrieben. Dadurch wird ein tiefer Einblick in die Abläufe des Systems möglich, was bedeutet, dass tiefgreifende Kostentransparenz möglich, aber unter Umständen auch kompliziert zu implementieren ist.
- Die Verwendung der .NET Klassenbibliotheken ASP.NET und ADO.NET machen einen Übersicht über die tatsächlichen Aufwände schwierig.

6 Fazit

Die derzeit wichtigsten Komponententechnologien EJB, CCM und .NET wurden ebenso dargestellt, wie die das noch junge Konzept der service-orientierten Architekturen. Die Untersuchungen dieser Technologien auf ihre Eignung im SOA-Kontext bezogen vor allem Eigenschaften ein, die den SOA-Charakteristika entsprechen, flexible Reaktionen auf neue Entwicklungen begünstigen, sowie generell wünschenswert sind. Es wurde nachgewiesen, dass EJB sich vor allem aufgrund der größeren Offenheit als geeignetster Standard zur Realisierung von SOA-Systemen darstellt, während das Microsoft-proprietäre .NET sehr auf Webservices abzielt, die ja nur eine Spezialisierung von SOA sind.

Gartner sah bereits 2003 für das Jahr 2008 voraus, dass 60% der Unternehmen SOA als leitendes Prinzip einsetzen werden, wenn sie geschäftskritischen Anwendungen, bzw. Prozesse erstellen [Gartner2003]. Im aktuellen Gartner-Bericht „Hype Cycle for Application Development 2007“ wird für sowohl fr SOA als auch für die Microsoft .NET-Plattform eine Dauer von noch 2 bis 5 Jahren geschätzt, bis sich diese Konzepte von der breiten Masse akzeptiert ist [Gartner2007]. Andererseits sieht Gartner für die J2EE Plattform dafür eine Dauer von unter deutlich 2 Jahren voraus. Wie es aussieht, ist EJB somit auch schon wegen der allgemein größeren Akzeptanz die geeignetere Technologie für SOA-Anwendungen.

Literatur

[Andresen2004] Andreas Andresen, *Komponentenbasierte Software-Entwicklung mit MDA, UML 2 und XML*, 2004, Carl Hanser Verlag München Wien

[Quian2005] Andy Ju An Wang, Kai Quian *Component-Oriented Programming*, 2005, John Wiley and Sons, Inc. Hoboken, New Jersey

[Taylor2006] Eric Pulier, Hugh Taylor *Understanding Enterprise SOA*, 2006, Manning Publications Co.

[Gartner2003] Gartner Research *Introduction to Service-Oriented Architecture*, 14.04.2003, Gartner Inc. and/or its Affiliates

[Gartner2007] Gartner Research *Hype Cycle for Application Development 2007*, 29.06.2007, Gartner Inc. and/or its Affiliates

[Koll2007] Sabine Koll *Softwarepreis wird individuell*, 10.09.2007 Computer Zeitung Nr. 37, Konradin IT-Verlag GmbH

[iWay2007] iWay Software *iWay Software - business integration, EAI, middleware and infrastructure solutions*, 2007, iWay Software, http://www.iwaysoftware.com/, Abruf: 19.10.2007

[Sun2007] Sun Microsystems *Sun Java Composite Application Platform Suite at a Glance*, 2007, Sun Microsystems, http://developers.sun.com/javacaps/index.jsp, Abruf: 19.10.2007

[Becking2007] Rolf Becking *Wohnmodil statt Lego-Haus*, 2007, e-commerce Magazin Special: Integration, serviceorientierte Architektur für Legacy-Applikationen, http://www.e-commerce-magazin.de/index.php3?page=05-04/special.html, Abruf: 20.10.2007

[Samaschke2006] Karsten Samaschke *XML.NET - XML und Web Services mit dem .NET Framework*, 2006, entwickler.press