Vergleich wichtiger Komponententechnologien bezüglich ihrer Eignung in SOA

Harry Trautmann
Hochschule für Technik, Stuttgart

1. Januar 2008

Zusammenfassung

Immer mehr Unternehmen setzen service-orientierte Architekturen (SOA) zur Integration ihrer Geschäftsprozesse ein. Um eine möglichst hohe Eignung von neu entwickelter Software für SOA zu erreichen, empfiehlt sich die Verwendung von Komponententechnologie. Im Folgenden werden die drei momentan wichtigsten Komponententechnologien CCM, .NET und EJB hinsichtlich ihrer Eignung für SOA verglichen. Dabei wird berücksichtigt, inwieweit die jeweilige Komponententechnologie den Charakteristiken von SOA entspricht, wie flexibel die Technologie gegenüber späteren Architekturänderungen reagiert, sowie welche weitere günstigen Eigenschaften es gibt.

Inhaltsverzeichnis

1 Einleitung

Von Beginn der Ära an, in der man computergestützte Datenverarbeitung in Unternehmen einsetzte, schrieb man eigene Programme, die einzelnen Unternehmenszweigen einen Mehrwert boten. So entstanden beispielsweise Module, die ausschließlich dem Finanzwesen dienten, Module, die man nur in der Produktionsplanung nutzte, sowie Module, die externe Kunden zur Kommunikation über Datenleitungen nutzen konnten. Doch im Laufe der Zeit zeigte sich, dass es zwar einerseits in hohem Maße erstrebenswert war, alle diese Insellösungen in einer standardisierten Anwendung zu integrieren, aber andererseits auch einen erheblichen Aufwand darstellte.

Beispielsweise wäre es nötig gewesen, weite Teile eines monolithischen Altsystems umzuschreiben, um sie nach außen hin mit den notwendigen Schnittstellen für die Integration auszustatten. Dies stellt jedoch einen erheblichen Aufwand dar, denn meist fehlt Altsystemen die Dokumentation des Quellcodes, während die ursprünglichen Programmierer bereits im Ruhestand oder zumindest nicht mehr im Unternehmen sind. Dadurch wird es extrem schwierig, sicher zu stellen, dass die umgeschriebene Software noch die gleiche Funktionalität bietet.

Um die Altsysteme also nicht verändern zu müssen und ihre Verfügbarkeit dennoch effektiv auf das ganze Unternehmen zu erweitern, werden sie über eine spezielle Schnittstellenschicht zu einer so genannten service-orientierten Architektur (SOA) ergänzt. Gerade diese Fähigkeit, Altsysteme fast unverändert zu integrieren, macht SOA zu einem wichtigen Thema für die meisten großen Unternehmen.

Damit man bei neu entwickelter Software nicht dieselben Fehler begeht wie bei Altsystemen, und gleichzeitig die Eignung für eine Integration in eine SOA erhöht, wird oft die Komponententechnologie eingesetzt. Mittels Komponententechnologie können Architekturen besser modularisiert und die einzelnen Geschäftsprozesse besser gekapselt werden. Außerdem erhöht sich die Wiederverwendbarkeit der Module, und die Versionierung, sowie das Deployment werden begünstigt.

Da sich in den letzten zehn Jahren mehrere Komponententechnologien etablierten, sollen im Folgenden die drei wichtigsten daraufhin verglichen werden, welche davon sich für die Implementierung einer SOA am besten eignet. Zunächst werden Aufbau und Funktionsweise der drei ausgewählten Technologien CCM, .NET und EJB beleuchtet. Darauf folgt die Darstellung der Charakteristiken von SOA. Die anschließende Einschätzung der Eignung dieser Komponententechnologien für SOA ist aufgeteilt in die Untersuchung, wie sehr die einzelnen Komponententechnologien den Charakteristiken von SOA entsprechen, eine Betrachtung der Flexibilität gegenüber Architekturänderungen, sowie einer Bewertung weiterer günstiger Eigenschaften. Ein Fazit mit einem Ausblick auf die zu erwartende Zukunft von SOA schließt die Untersuchung ab.

2 Umfeld

2.1 Komponententechnologien

Eine Komponententechnologie spezifiziert nach [1, S. 261] die Syntax und Semantik von Komponenten und stellt eine zugehörige Laufzeitumgebung bereit. Diese Laufzeitumgebung verfügt über grundlegende Dienste, wie z. B. zur Erzeugung, Aktivierung und Deaktivierung von Komponenten, sowie weitere Dienste für Persistenzmanagement, Transaktionen, Sicherheit und Verzeichnisdienste. Die jeweilige Technologie kann auf spezifischen Application Servern, als auch auf spezifischer Middleware basieren.

In den letzten zehn 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. Um Unterschiede in Konzepten und Aufbau darzustellen werden sie im Folgenden beschrieben.

2.1.1 CORBA Component Model

CORBA ist ein plattform- und sprachunabhängiger Standard für Middleware und wurde von der Object Management Group (OMG) zusammengestellt, 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 [1, S. 270].

Grundlegend für CORBA ist der ORB (Object Request Broker). Dies ist eine Schicht, in der notwendige Basis-Services implementiert sind. Der ORB stellt den Komponenten und Komponenten-Containern Transaktions-, Sicherheits-, Benachrichtigungs- und Persistierungsdienste zur Verfügung [1, S. 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, das Instanzieren und die 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

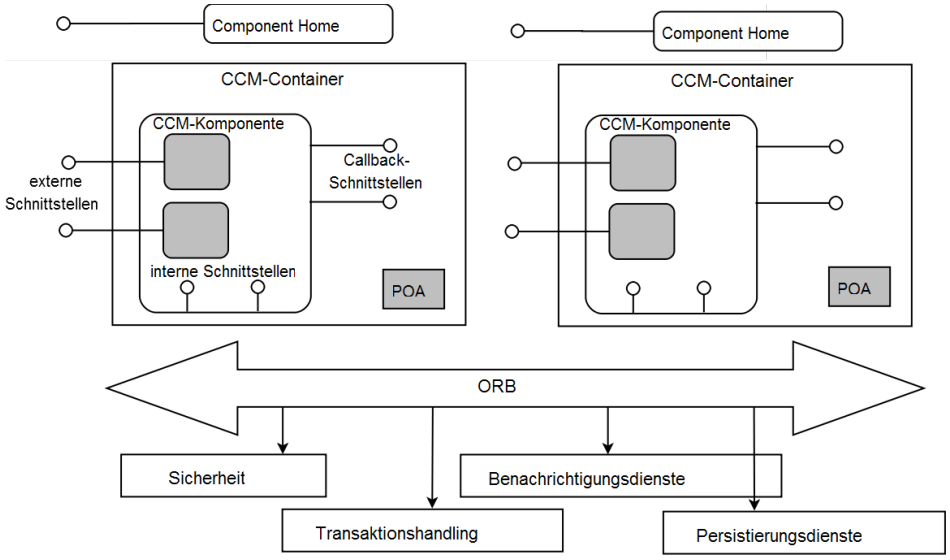

Abbildung 1: Einbettung von Komponenten in einer CCM-Architektur [1, S. 275]

portabel zu halten [9, S. 146].

In Abbildung 1 wird der Aufbau dargestellt, wie CCM-Komponenten, -Container, die jeweiligen Schnittstellen, sowie der ORB zusammenarbeiten.

CCM-Komponenten stellen sich nach außen über vier Arten von Schnittstellen dar.

1. Facets: sie sind die zur Komponente externe Schnittstelle für die Geschäftslogik 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 zur Komponente extern erzeugten Ereignissen.

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. Die Implementierung des .NET-Frameworks von Microsoft läuft nur auf den Windows-Betriebssystemen. Allerdings erweitert das „Mono Project“ mit einer eigenen Implementierung die Lauffähigkeit auch auf die Betriebssysteme Linux, Solaris, Mac OS X und Unix [6].

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 Anwendung für die Oberfläche von Windows (Windows Forms).

Abbildung 2 stellt den groben Aufbau des .NET-Frameworks dar.

.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 [9, S. 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 so genannten „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, Abläufen von Threads und Systemschutz geprüft wurden. Die CLR-Ausführungseinheit führt den Managed Native Code aus.

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 Verwaltung ihres Lebenszyklus an. Somit können bei EJB-Implementierungen diese Aspekte außer Acht gelassen werden, so dass sich der Code voll auf die Umsetzung der Geschäftslogik konzentriert. Eine J2EE-Architektur wird im Allgemeinen in die folgenden semantischen Bereiche (die sogenannten Tiers) aufgegliedert.

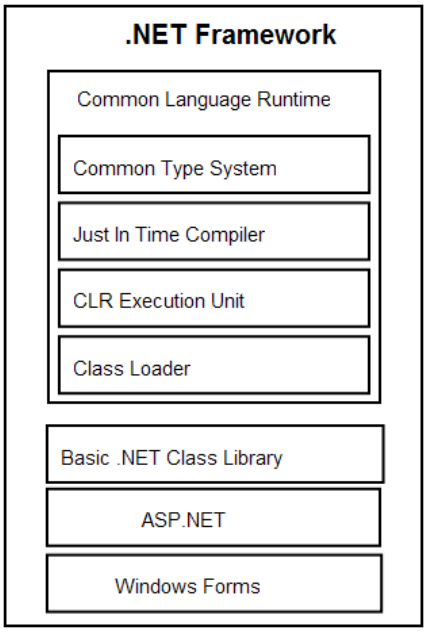

Abbildung 2: Aufbau des .NET-Frameworks [9, S. 196]

1. Client-Tier: diese Ebene enthält die Funktionalität zur Interaktion des Benutzers mit dem System. Oft werden Java-Anwendungen wie Browser diesem Tier zugerechnet.

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. EJB 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.

EJB können erst zusammen mit einer Laufzeitumgebung genutzt werden. Der EJB-Server stellt eine solche zur Verfügung. Der EJB-Server bietet den EJB grundlegende Dienste wie Verwaltung von Prozessen und Threads an, aber auch komplexe, wie beispielsweise einen Dienst für das Zusammenlegen gleichartiger Instanzen von Netzwerk- und Datenbankenressourcen, so dass mehrere EJB sie gemeinsam nutzen können.

Auch EJB 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 [9, S. 269].

Abbildung 3 visualisiert den Zusammenhang zwischen EJB-Komponente, -Container, -Server, sowie externen Systemen.

Auf den Systemen liegen EJB in Form von JAR-Dateien, wodurch alle Teile der Komponente innerhalb einer einzelnen Datei gekapselt (und optional gepackt) vorliegen.

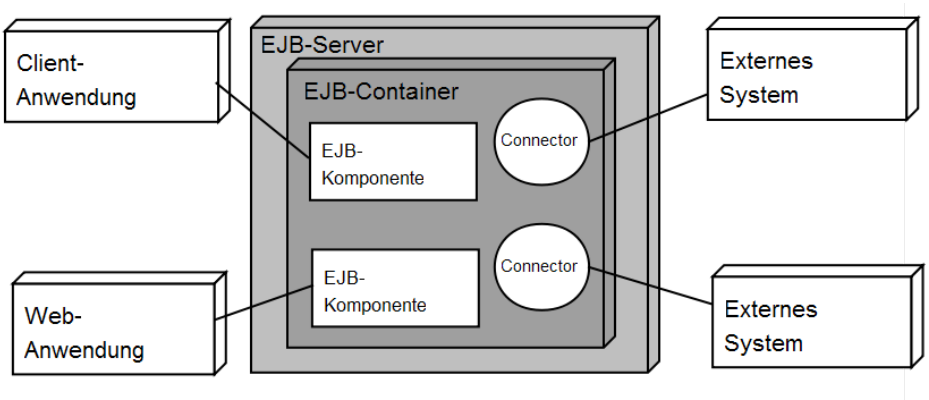

Abbildung 3: Einbettung von Komponenten in eine EJB-Architektur [1, S. 269]

2.2 Service-oriented Architecture

Nachdem die relevanten Eigenschaften der zu betrachtenden Komponententechnologien aufgeführt wurden, müssen auch die Kriterien eingehender ausgeführt werden, mittels denen sich ihre Eignung für SOA bewerten lässt. Zunächst soll der Begriff SOA geklärt werden, um dann auf die wichtigsten Charakteristiken einzugehen.

2.2.1 Definition

Moderne Anforderungen, wie z. B. globale Verfügbarkeit von Programmfunktionalität und notwendige Kosteneinsparung bei der Wartung, führten dazu, die vorhandenen Dienste einer Software-Architektur mit einer offenen

Schnittstellenschicht auszustatten, um ihre Funktionalität in anderer Software wieder verwenden zu können. Das ermöglichte Architekturen, deren Aufbau sich an den zur Verfügung stehenden Diensten ausrichtet. Obwohl dies den SOA entspricht, existiert derzeit immer noch keine einheitliche Definition des Begriffs „Service-orientierte Architektur“. Entsprechend [4, S. 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 Charakteristiken

Um die Eignung einer Komponententechnologie für den 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. Im Zusammenhang mit den Komponententechnologien werden die folgenden Charakteristiken als bedeutend identifiziert.

- Fokussierung auf Geschäftsprozesse: jede durch eine SOA bereitgestellte Schnittstelle repräsentiert einen Geschäftsprozess in Form einer Black Box. Komponententechnologien, die eine Trennung Geschäftslogik und deren Präsentation begünstigen, sind daher im Vorteil.

- Lose Kopplung von Diensten: Dienste sind lose gekoppelt, wenn kein Konsument des Dienstes ausschließlichen Zugriff darauf erhält. Daraus resultiert eine hohe Flexibilität für Dienste bezüglich ihrer internen Kombination und Rekombination. Diese Änderungen in ihrer internen Zusammenstellung wirken sich dann weniger gravierend auf die darüberliegende Architektur aus. Somit ist eine Komponententechnologie besser für SOA geeignet, je flexibler ihre Komponenten zur Laufzeit zusammengestellt werden können.

- Skalierbarkeit: SOA werden vornehmlich im Umfeld der Web Economy eingesetzt. Dort ist es eine übliche Anforderung an eine Software, einer großen Anzahl von Anfragen an Dienste frei von Verzögerungen nachzukommen [2]. Daher ist eine Komponententechnologie umso besser für SOA geeignet, je besser sie unter Last von Anfragen skaliert, um der Bedienung vieler Konsumenten frei von Verzögerungen nachzukommen.

3 Entsprechung gegenüber SOA-Charakteristiken

3.1 Fokussierung auf Geschäftsprozesse

Um eine SOA wiederverwendbar zu halten, muss bereits in der Technologie, welche die SOA implementiert, eine möglichst weit gehende Trennung zwischen der Geschäftslogik und ihrer Präsentation vorhanden sein.

- Mit Hilfe von unterschiedlichen Container-Klassen erzwingt die J2EE-Plattform eine Trennung zwischen Geschäftslogik und deren Präsentation. Dabei sind für die Implementierung von Geschäftslogik Klassen vorgeschrieben, die auf den EJB aufbauen. Für die Präsentation müssen Klassen auf Basis von JSP oder Servlets verwendet werden.
- CCM schreibt keine derartige Trennung vor. Eine Fokussierung auf Geschäftsprozesse ist daher nicht inhärent gegeben.
- .NET schreibt keine Trennung vor, bietet aber mit der Bereitstellung von entsprechenden Klassen in der ASP.NET-Klassenbibliothek eine Hilfestellung dafür.

3.2 Lose Kopplung

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 erneute Konfiguration einer Komponenteninstanz einfach gestaltet.

- EJB werden in JAR-Dateien verpackt, die auch einen so genannten 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 [1, S. 291]. Dies stellt innerhalb SOA einen Vorteil dar.
- Wenn eine neue CCM-Komponente bereitgestellt werden soll, ist es lediglich notwendig, deren in IDL (Interface Definition Language) spezifizierte Schnittstellen um die neuen Schnittstellen zu erweitern und die bestehende Komponente mit der neu kompilierten zu ersetzen [1, S. 233]. Somit ist es auf relativ unkomplizierte Art möglich, die Konfiguration eines Systems, das auf CCM-Komponenten aufbaut, zu ändern.
- .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. [9, S. 195]

3.3 Skalierbarkeit

Die Komponententechnologien bieten diverse Techniken an, um die Systemressourcen zu schonen. Einige dieser Techniken verfolgen das Ziel, die Antwortszeit des Systems auch unter einer sich dynamisch verändernden Last von Anfragen stabil zu halten und dienen damit der Skalierbarkeit. Hier wird beleuchtet, welche dieser Techniken das sind.

- EJB-Server verfügen über Mechanismen für den Zusammenschluß gleicher Datenbankverbindungen und EJB-Instanzen, sowie deren Ablage in einem Cache . Somit ist die EJB-Technologie sehr gut skalierbar. [1, S. 288]
- Skalierbarkeit kann in CCM durch Verwendung unterschiedlicher Container erreicht werden [1, S. 289]. Dies verlangt also, dass bereits während des Kompilierens 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 ist CCM prinzipiell, aber nicht besonders flexibel skalierbar.
- Innerhalb des .NET-Framework sorgen die ASP.NET- und ADO.NET-Module für die Ablage von Datenbankverbindungen und Statusinformationen der Instanzen in einem Cache. Wie gut eine .NET Architektur skaliert, hängt aber vor allem auch von der Konfiguration des Backend ab [1, S. 289].

4 Flexibilität bei Architekturänderungen

Neben der Überdeckung der SOA-Charakteristiken ist auch entscheidend, wie flexibel sich eine auf Komponententechnologie basierende Architektur verändern lässt. Denn einmal vorliegende Architekturen tendieren dazu, erweitert zu werden, ebenso wie die Anforderungen und das Umfeld einer Software dazu tendieren, sich in unerwarteter Weise zu ändern. Im Kontext der SOA wird dieser Umstand noch verschärft, da man Dienste auch dynamisch zusammengestellen können soll. Flexibilität einer Komponententechnologie definiert sich daher aus deren Grad an Ungebundenheit von Plattform, Programmiersprache und Hersteller.

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 EJB 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 von Microsoft nur auf den Windows-Betriebssystemen implementiert. Doch durch das „Mono Project“ existieren auch für eine Reihe weiterer Betriebssysteme Umsetzungen des .NET-Frameworks. Somit sind auch .NET-Komponenten plattformunabhängig [6].

  TODO Mono einfügen

4.2 Sprachunabhängigkeit

- EJB 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 zeitnah 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.

- Aufgrund der Existenz des „Mono Projects“ besteht die Wahl, ob man lieber die Implementierung von Microsoft einsetzt oder die Open-Source-Implementierung des „Mono Projects“. Die Implementierung von Microsoft ist zwar einerseits an die Windows-Betriebssysteme gebunden. Andererseits pflegt Microsoft sie sorgfältig, beispielsweise, indem es zeitnah mit Updates auf Sicherheitslücken in der Framework-Implementierung reagiert. Die „Mono“-Implementierung befreit von der Abhängigkeit von Microsoft, gibt allerdings keine Gewähr.

5 Weitere günstige Eigenschaften

Im Folgenden sollen einige Eigenschaften der betrachteten Technologien untersucht werden, deren Vorliegen 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 der Sicherheit dienender Mechanismus 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 [1, S. 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 wie viele der vorhandenen Ressourcen in Anspruch nehmen. Insbesondere ist dies dann

wichtig, wenn extern veröffentlichte Dienste Kunden berechnet werden sollen [5].

- 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 tief greifende Kostentransparenz möglich, aber unter Umständen auch kompliziert zu implementieren ist.

- Falls die .NET-Klassenbibliotheken ASP.NET und ADO.NET verwendet werden, können die von diesen Klassen verursachten Aufwände leicht abgeschätzt werden.

6 Fazit

Die derzeit wichtigsten Komponententechnologien EJB, CCM und .NET wurden ebenso dargestellt, wie das noch relativ neue Konzept der serviceorientierten Architekturen. Die Untersuchungen dieser Technologien auf ihre Eignung im SOA-Kontext bezogen Eigenschaften ein, die den SOA-Charakteristiken entsprechen, flexible Reaktionen auf neue Entwicklungen begünstigen, sowie generell wünschenswert sind. Es wurde nachgewiesen, dass EJB vor allem aufgrund der größeren Offenheit für SOA-Systeme am besten geeignet ist, während das proprietäre .NET sehr auf Webservices abzielt, die 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 [3]. Einerseits wird im aktuellen Gartner-Bericht „Hype Cycle for Application Development 2007“ sowohl für SOA als auch für die Microsoft .NET-Plattform eine Dauer von noch zwei bis fünf Jahren geschätzt, bis diese Konzepte von der breiten Maße akzeptiert sind [4]. Andererseits sieht Gartner für die J2EE-Plattform eine Dauer von deutlich unter 2 Jahren voraus. Somit ist EJB bereits wegen der allgemein größeren Akzeptanz die derzeit am besten geeignete Technologie für SOA-Anwendungen.

Glossar

.NET eine von Microsoft spezifizierte Komponententechnologie, 1

Altsystem (engl. Legacy System) eine i. d. R. über lange Zeit verwendete und historisch gewachsene Anwendung im Bereich der Unternehmenssoftware, deren Wartung ausgesetzt wurde, da entweder die Dokumentation unvollständig oder überhaupt nicht vorliegt und / oder die ursprunglichen Programmierer nicht mehr im Unternehmen angestellt sind (zumeist, da sie den beruflichen Ruhestand antraten), 1

API (Application Programming Interface) von einem Programm bereitgestellte Schnittstelle, die dazu dient, von außerhalb des Programms Einfluss auf den Ablauf des Programms zu nehmen oder einzelne Teile davon aufzurufen, 2

Application Server Software, die als Umgebung für den Ablauf von Anwendungen dient und ihnen besondere Dienste zur Verfügung stellt, 1

Assembly Bezeichnung für .NET-Komponenten [9, S. 26], 3

Backend (dt. hinteres Ende) in einer Client-Server-Architektur die Ebene der Server, 9

Binärprogramm Programm, das als binär repräsentierte Anweisungen vorliegt, 3

Bytecode i. d. R. maschinenunabhängige Sammlung von Befehlen, die von einer virtuellen Maschine interpretiert werden, 3

Cache ein Speicherbereich mit geringer Zugriffszeit, der Kopien von Inhalten anderer (langsamerer) Speicherbereiche enthält mit dem Ziel, den zukünftigen Zugriff auf diese Inhalte zu beschleunigen, 9

CCM (CORBA Component Model) eine auf CORBA 3.0 basierende Komponententechnologie, 1

CLR (Common Language Runtime) Laufzeitumgebung im .NET-Framework; dient der Ausführung von Assemblies, 3

Container (Komponenten-) ein Programm, das eine oder mehrere Komponenten enthält, 2

CORBA (Component Object Request Broker Architecture) von der Object Management Group zusammengestellter Standard für Middleware, 2

Deployment Verteilung, Konfiguration und Installation von Software auf einem Zielsystem, 1

Destruktion Vorgang der Freigabe von durch ein Objekt gebundenen Ressourcen, 5

Dienst wohldefiniertes, in sich abgeschlossenes Modul, das standardisierte, für einen Geschäftsprozess charakteristische Funktionalität bereitstellt und unabhängig von Zustand oder Kontext anderer Dienste ist [8, S. 1], 1

EJB (Enterprise Java Bean(s)) eine auf Java basierende Komponententechnologie; im Singular eine einzelne Komponente, 1

Framework (dt. Rahmengerüst) der beschleunigten Programmierung dienende implementierte Klassen und Hilfsprogramme, 3

Geschäftslogik Ablaufbeschreibung der Interaktion von Geschäftsprozessen miteinander, 4

Geschäftsprozess Abfolge von Schritten innerhalb einer Organisation, um ein bestimmtes Resultat zu erhalten, 1

Herstellerbindung Freiheitsgrad in der Beziehung zwischen einer Software und ihrem Ersteller, 10

IDL (Interface Definition Language) eine deklarative formale Sprache, die der Beschreibung von Schnittstellen von Software-Komponenten dient, 8

IL (Intermediate Language) eine Sprache, die der Definition von Assemblies dient, 3

Insellösung technisches System, das nur innerhalb der eigenen Grenzen wirksam und zu ähnlichen oder verwandten Systemen inkompatibel ist, 1

Instanzieren ein individuelles Abbild einer Klasse, das über einen eigenen Speicherbereich verfügt, (Instanz) erzeugen, 2

Interface (dt. Schnittstelle) eine von einem Protokoll festgelegte Austauschweise für Informationen; im Rahmen der Komponententechnologien bezieht sich der Begriff auf die semantische Bedeutung sowie deren technische Implementierung von Methoden und Objekten des Komponentenmodells, 2

J2EE (Java 2 Enterprise Edition) Spezifikation einer Softwarearchitektur für transaktionsbasierte Ausführung von Java-Anwendungen; insbesondere stellt J2EE den Rahmen für die Verwendung von EJB zur Verfügung, 4

JSP Java Server Page, 8

JVM (Java Virtual Machine) die Laufzeitumgebung für Java-Programme, die zu Bytecode kompiliert wurden, 9

Klasse (in der Objektorientierung) die Beschreibung der gemeinsamen Struktur und des gemeinsamen Verhaltens von Objekten, 4

Klassenbibliothek Sammlung von fertig gestellten Klassen, 4

Kompilieren Vorgang des Umwandelns des Quelltextes eines Programms in ein Binärprogramm, 3

Komponententechnologie Spezifikation für Syntax und Semantik von (Software-)Komponenten; i. d. R. werden auch grundlegende Dienste zur Erstellung, Aktivierung und Deaktivierung von Komponenten bereitgestellt, 1

Konfiguration eine bestimmte Einstellung von Software oder Hardware, 9

Konstruktion Vorgang der Instanzierung eines Objekts einer Klasse, 5

Kostentransparenz Offenlegung der Verbrauchs an Ressourcen (Kosten), 11

Laufzeitumgebung (engl. Runtime Environment) ein meist vom Betriebssystem abhängiges Programm, das Dienste zur Interpretation und Ausführung von Programmen dient, die einem speziellen Protokoll folgen und / oder in einen Bytecode vorkompiliert wurden, 1

lose Kopplung Beziehung zwischen einem Dienst und einem Konsumenten, wobei der Konsument bei Anforderung des Dienstes keinen ausschlieslichen Zugriff darauf erhält, 7

Metadaten Daten, welche Informationen über andere Daten enthalten, 4

Middleware Programme, die zwischen anderen Anwendungen vermitteln mit dem Ziel, die Komplexität dieser Kommunikation zu verbergen, 1

monolitisch (in der Informatik) eine Software, deren Architektur technisch und / oder semantisch nicht in Module aufgeteilt ist, 1

proprietär nicht allgemein anerkannten Standards entsprechend, 13

Servlet spezielle Java-Klassen, die innerhalb eines J2EE-Applikationsservers dazu dienen, Anfragen von Clients zu beantworten, 8

Skalierbarkeit Verhalten des Wachstums an Ressourcenbedarf von Programmen bei veranderter Eingabemenge, 7

SOA Paradigma zur Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet wird[7], 1

Systemressource die auf einem Computer verfügbaren Betriebsmittel, i. d. R. Rechenzeit und Speicherplatz, 8

Thread ein einzelner Ausführungsstrang bei der Abarbeitung von Computerprogrammen, 4

Tier eine einzelne Schicht in einem mehrschichtigen Architektur einer Anwendung, 4

Versionierung archivisches Verwaltungssystem von alternativen Programmversionen; im Rahmen der Komponententechnologien dient die Versionierung dazu, die Kompatibilität von Programmen sicher zu stellen, die auf unterschiedliche Versionen von Komponenten zugreifen, 1

virtuelle Maschine Software, die eine virtuelle Laufzeitumgebung bereitstellt, so dass Bytecode ausgeführt werden kann, 3

Web Economy eine vom Kommerz mit den (neuen) Kommunikationsmedien des Internet geprägte Wirtschaftsform; der auf das Web ausgerichtete Zweig der New Economy, 7

Webservice Softwareanwendung, die mittels einem Uniform Resource Identifier identifiziert wird und deren Schnittstellen über XML-Artefakte beschrieben werden, 13

Literatur

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

[2] 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

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

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

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

[6] Mono Project *The Mono Project*, 2007, Mono Project, http://www.mono-project.com/Main_Page, Abruf: 31.12.2007

[7] OASIS Consortium *Reference Model for Service Oriented Architecture 1.0*, 2006, OASIS Consortium, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm, Abruf: 30.12.2007

[8] Mike P. Papazoglou, Willem-Jan van den Heuvel *Service oriented architectures: approaches, technologies and research issues*, 2007, Springer Verlag, Berlin Heidelberg

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

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