Seminar der AG DBIS im Wintersemester 2001/2002
DB-Aspekte des E-Commerce, Schwerpunkt: Anwendungen

e-Business-Engineering

Philipp Dopichaj

27. Januar 2002

Inhaltsverzeichnis

1 Einleitung

Diese Seminararbeit beschäftigt sich mit der Erstellung von Software für das e-Business. Dabei wird keine Methodologie vorgestellt, sondern es werden Komponenten und verwandte Technologien als Lösungsansatz für typische Probleme vorgestellt und gezeigt, wie sie für die Lösung dieser Probleme nützlich sein können.

Doch zunächst einmal stellt sich die Frage, was genau e-Business ist; da es keine allgemein anerkannte Definition dieses Begriffes gibt, ist es wichtig, ihn zu definieren. In diesem Text beinhaltet e-Business jegliche Art von elektronischem Geschäftsverkehr, insbesondere Geschäftsverkehr, der über das Internet abgewickelt wird. Es ist nicht auf Beziehungen zwischen Verbrauchern und Händlern (B2C, *business to consumer*) eingeschränkt, sondern umfasst auch Beziehungen von Unternehmen untereinander (B2B, *business to business*). Es ist wichtig, diesen zweiten Bereich nicht zu vernachlässigen, denn gerade hier gibt es ein großes Potential zur Automatisierung, da diese Beziehungen meist langfristig sind. Zwar gibt es schon länger Ansätze, zumindest den Datenaustausch in diesem Bereich zu standardisieren (zum Beispiel Electronic Data Interchange (EDI)), aber diese decken zum einen nur einen sehr kleinen Bereich ab und sind zum anderen nicht flexibel genug, sich diesem schnelllebigen Markt in angemessener Zeit anzupassen.

Zunächst werden die besonderen Probleme im Bereich e-Business vorgestellt und anschließend Komponenten als flexibler Lösungsansatz für viele dieser Probleme präsentiert. Der folgende Abschnitt beschäftigt sich dann mit der praktischen Benutzung von Komponenten, indem einige Technologien kurz vorgestellt werden; danach wird ein ausführlicher Überblick über Web Services als zukunftsträchtige Ergänzung der älteren Technologien gegeben.

2 Besonderheiten im e-Business

Zunächst einmal stellt sich die Frage, wieso man e-Business-Engineering überhaupt gesondert betrachten sollte, wo sich doch schon das Software Engineering mit der Erstellung von Software im Allgemeinen beschäftigt. Dazu lässt sich sagen, dass es im e-Business-Bereich genug Besonderheiten gibt, die ihn von „normaler“ Softwareentwicklung abgrenzen – was natürlich nicht heißt, dass man auf die Methoden des Software Engineering verzichten kann.

Üblicherweise wird man im e-Business auf eine Drei-Tier-Architektur zurückgreifen, wobei die Tier folgende Bedeutungen haben:

- **Data Tier** – hier werden die Geschäftsdaten verwaltet, wobei neben DBVS auch andere Systeme zum Einsatz kommen können
- **Business Tier** (auch **Middle Tier**) – hier wird die Geschäftslogik verwirklicht und somit der Hauptteil der Arbeit erledigt; diese Seminararbeit bezieht sich hauptsächlich auf diese Schicht
- **Presentation Tier** – hier wird die Ausgabe des Business Tier für den Endnutzer aufbereitet; üblicherweise wird zur Präsentation ein Web-Browser verwendet

Im Folgenden werden die spezifischen Besonderheiten und Probleme besprochen, wobei als Beispiel ein Versandhändler herangezogen wird, der seine Waren auch in einem Online-Shop anbieten will.

2.1 Kundenkontakt (B2C) und firmeninterne Kommunikation

In fast allen Fällen ist es nötig, auf interne Firmendaten zuzugreifen. Bei einem Online-Shop ist es beispielsweise sinnvoll, für die Web-Darstellung automatisch überprüfen zu können, ob eine Ware noch im Lager vorhanden ist oder nicht. In anderen Branchen ist es zudem wichtig, dass Vertreter oder Servicepersonal von überall her über mobile Geräte wie Mobiltelefone oder PDAs auf interne Funktionen zugreifen können. Wenn für diese Funktionen aktuelle Software verwendet wird, stellt der Zugriff von außen kein großes Problem dar; in vielen Fällen werden jedoch alte Systeme *(legacy systems)* verwendet, die darauf nicht vorbereitet sind.

Weiterhin ist es meist so, dass das Unternehmen langsam und schrittweise wächst. Ein Online-Shop wird kurz nach seiner Eröffnung noch nicht viele Kunden anziehen, aber wenn er später erfolgreich ist, muss er auch mit einem großen Anfragendurchsatz fertig werden können. Dabei ist es natürlich nicht sinnvoll, gleich zu Anfang ein sehr leistungsfähiges System zu benutzen, das für eine lange Zeit nicht annähernd ausgenutzt wird. Vielmehr sollte es möglich sein, mit einem kleinen, an die Anfangsbedürfnisse angepassten System anzufangen und dieses dann schrittweise – je nach Bedarf – aufzurüsten; anders gesagt, die Skalierbarkeit des Systems muss gewährleistet sein.

Viele Problemstellungen treten im e-Business immer wieder auf; beispielsweise braucht jeder Online-Shop eine Warenkorbfunktion, und die Warenkörbe verschiedener Shops unterscheiden sich wenig. Es ist wirtschaftlich wenig sinnvoll, dieses Teilprogramm für jeden Händler von Grund auf neu zu entwickeln; daher ist es besonders sinnvoll, Wiederverwendung zu fördern.

Im e-Business ist es häufig so, dass nicht IT-Experten die Softwareerstellung durchführen, sondern Leute, die nur über Grundkenntnisse in diesem Bereich verfügen. Dafür kennen sie sich im Geschäftsbereich ihres Unternehmens sehr gut aus, was den Vorteil hat, dass Missverständnisse zwischen Entwickler und Benutzer minimiert werden. Durch die fehlenden Informatikkentnisse muss auf „klassische“ Programmiermethoden verzichtet werden und eine andere Lösung gefunden werden.

2.2 Firmenübergreifende Kommunikation (B2B)

Ein besonders wichtiger Aspekt des e-Business ist die Möglichkeit zur automatischen Kommunikation und Kooperation mit anderen Unternehmen. Beispielsweise ist es für Händler sinnvoll, mit Anbietern von Online-Bezahldiensten zusammenzuarbeiten oder Nachbestellungen halbautomatisch abzuwickeln.

Auch hier ergibt sich wieder – ähnlich wie bei den firmeninternen Legacy Systems – das Problem, dass heterogene Systeme miteinander Daten austauschen müssen. Hierzu ist es nötig, passende Austauschformate und -protokolle zu definieren, wobei es wünschenswert ist, dass man mit Anbietern gleichartiger Dienste auch auf die gleiche Art kommuniziert werden kann.

Da hier teilweise sehr sensible Daten übertragen werden müssen, ist es sehr wichtig, zu verhindern, dass diese von Dritten problemlos gelesen werden können. Das lässt sich über

Verschlüsselungs- und Zertifizierungstechniken erreichen; allerdings ist es auch hierbei wieder nötig, dass beide Seiten kompatible Technologien benutzen. Schwieriger zu lösen ist das Problem des Vertrauens: Wie kann ich sicher sein, dass mein Handelspartner kein Betrüger ist? Dieses Problem lässt sich nicht allein durch Technologie lösen, es ist auch nötig, eine Möglichkeit zu finden, die Vertrauenswürdigkeit eines Anbieters zu bewerten.

3 Komponenten

Zur Lösung der im vorigen Abschnitt angedeuteten Probleme bietet sich die Verwendung von Komponenten an. In diesem Abschnitt wird geklärt, was Komponenten sind und welche Vorteile man sich erhoffen kann, wenn man sie verwendet, während der folgende Abschnitt sich mit der praktischen Anwendung beschäftigt.

3.1 Grundlagen

Komponenten

Zunächst einmal muss man klären, was man überhaupt unter einer Komponente versteht. In der Literatur findet man leicht abweichende Definitionen; in diesem Artikel orientieren wir uns an der Definition aus Ciupke und Schmidt (1997), die den Kern der Sache gut trifft:

> *Components are software units that are context independent both in the conceptual and in the technical domain.*

Das bedeutet, dass Komponenten nicht nur von der Problemstellung so weit abstrahieren, dass sie auch leicht zur Lösung ähnlicher Probleme verwendet werden können, sondern dass sie zusätzlich noch vom technischen Umfeld wie Programmiersprache oder Plattform unabhängig sind. Dadurch ergibt sich eine ganz neue Dimension der Wiederverwendung, denn frühere Lösungen waren meist sehr stark an ihr technisches Umfeld gekoppelt.

Eine weitere Eigenschaft, die sich daraus ergibt, ist die weitgehende Unabhängigkeit der Komponenten untereinander: Wenn eine Komponente auf die Dienste anderer Komponenten angewiesen ist, sollte keine bestimmte Komponente verlangt werden. Vielmehr sollte der Benutzer die Möglichkeit haben, eine Komponente seiner Wahl „anzustöpseln“, sofern diese eine passende Schnittstelle bereitstellt.

Komponentenmodelle

Neben dem Begriff der Komponente muss man auch noch Komponentenmodelle betrachten. Laut Segev und Bichler (1999) ist ein Komponentenmodell *(component model)* eine Menge von Richtlinien, die festlegen, wie die Schnittstellen der Komponenten aussehen sollen. Ferner sollte das Modell folgende Möglichkeiten bieten:

- **Properties** – abstrakte Sicht auf den nach außen sichtbaren Zustand der Komponente
- **Introspection** – Möglichkeit, die Fähigkeiten einer Komponente zur Laufzeit zu bestimmen

- **Customization** – Möglichkeit für Entwickler, über bestimmte Werkzeuge das Verhalten und Aussehen einer Komponente zu ändern
- **Persistence** – Möglichkeit, die über Customization durchgeführten Änderungen dauerhaft abzuspeichern
- **Packaging** – Beschreibung, wie Komponenten verbreitet und installiert *(deployed)* werden können

Diese Punkte zielen insgesamt darauf ab, dass Komponenten wirklich austauschbar sind, ohne dass manuelle Änderungen vorgenommen werden müssen (Properties, Introspection, Customization, Persistence) und auf einheitliche Weise installiert werden können; dadurch wird die Wiederverwendung noch weiter gefördert.

3.2 Verhältnis zur Objektorientierung

Da auch die objektorientierte Programmierung das Ziel hat, die Wiederverwendung von Software zu fördern, liegt es nahe, Unterschiede und Gemeinsamkeiten dieser Philosophien herauszuarbeiten.

Dazu betrachten wir zunächst die Einheiten, die wiederverwendet werden; in der Objektorientierung sind das Objekte und deren Klassen, denen Komponenten und Schnittstellendefinitionen gegenüber stehen. In dieser Hinsicht besteht eine deutliche Ähnlichkeit der beiden Ansätze, denn in beiden Fällen geht es darum, eine gewisse Funktionalität durch eine feste Schnittstelle zur Verfügung zu stellen, ohne die Details der Implementierung offen zu legen.

Ein wesentlicher Unterschied liegt in der Art, wie die Wiederverwendung vonstatten geht: Bei der objektorientierten Programmierung wird deutlich die Wiederverwendung durch Vererbung, also *is-a*-Beziehungen, betont, stützen sich Komponenten auf Wiederverwendung durch Zusammensetzung, also *has-a*- und *uses-a*-Beziehungen.

Bei der Vererbung wird neben der Schnittstelle auch die Implementierung und in gewisser Hinsicht der Typ der Elternklasse wiederverwendet, was zu einer sehr starken Verbindung führt – es ist nicht ohne weiteres möglich, die Elternklasse später zu ändern, ohne den Programmkode, der die abgeleitete Klasse verwendet, entsprechend anzupassen. Bei einer *has-a*-Beziehung hingegen ist es für den Aufrufenden praktisch unmöglich, festzustellen, welche Komponente intern verwendet wird, da es sich auf die öffentliche Schnittstelle nicht auswirkt. Außerdem spricht nichts dagegen, erst zur Laufzeit festzulegen, welche Komponente intern verwendet wird. Diese Flexibilität ist gerade im e-Business wünschenswert, wie in Abschnitt 3.4 noch genauer erklärt wird.

Insgesamt lässt sich sagen, dass beim Komponentenkonzept versucht wird, die Wiederverwendung noch besser zu unterstützen, als das bei der Objektorientierung ohnehin schon der Fall ist. Dies wird durch einen höheren Abstraktionsgrad und die Unabhängigkeit der Komponenten untereinander erreicht; bei Klassenverbänden ist in der Praxis häufig eine genaue Kenntnis der Interna und des Zusammenspiels der einzelnen Klassen nötig.

3.3 Verteilung

Auch wenn dies von unserer Definition nicht direkt gefordert wird, ermöglichen fast alle erfolgreichen Komponentensysteme (mit Ausnahme von COM) die Verteilung der Komponenten auf geographisch entfernte Rechnersystemen, die nur über Netzwerke – typischerweise TCP/IP – verbunden sein müssen.

Üblicherweise gibt es zentrale Verzeichnisse, in denen die Adressen der Komponenten und die von ihnen angebotenen Dienste verzeichnet werden. Wenn eine Komponente erst einmal gefunden und eingebunden worden ist, macht es von der Programmierung her keinen Unterschied mehr, ob sie auf demselben Rechner abläuft wie das Programm, das sie benutzt. Somit muss die Verteilung der Komponenten nicht von vornherein feststehen, sondern kann prinzipiell zu beliebigen Zeitpunkten noch geändert werden, ohne dass große Änderungen durchgeführt werden müssen.

Das kann als Erweiterung der Unabhängigkeit von technologischen Aspekten betrachtet werden, denn damit wird auch erreicht, dass verschiedene Programmteile auf heterogenen Hard- und Softwareplattformen ausgeführt werden können.

3.4 Vorteile von Komponenten im e-Business

Die Eigenschaften von Komponenten und Komponentenmodellen machen sie zu geeigneten Werkzeugen für die Softwareentwicklung für e-Business, denn sie können zur Lösung vieler der in Abschnitt 2 besprochenen Probleme benutzt werden:

Da der Abstraktionsgrad von Komponenten sehr hoch ist (oder zumindest sein sollte), werden nicht mehr typische Software-Engineering-Konzepte wie Stacks oder Queues, sondern Objekte aus dem Problembereich, also Geschäftsobjekte, modelliert. Durch die zusätzlich noch vorhandenen Plug-and-Play-Eigenschaften wird es somit möglich, dass Experten aus dem entsprechenden Gebiet, die aber Informatiklaien sind, vorgefertigte Komponenten durch Parametrisierung an die eigenen Bedürfnisse anpassen und zu komplexeren Softwareeinheiten zusammenfügen.

Außerdem bieten Komponenten auch Lösungsansätze für folgende Probleme:

- Durch die Betonung der Wiederverwendung wird verhindert, dass das Rad immer wieder neu erfunden werden muss.
- Durch die Möglichkeit der Verteilung ist es möglich, einzelne Teile des Geschäftsprozesses auszulagern.
- Durch die Änderbarkeit der Verbindungen unter den Komponenten zur Laufzeit wird die Dynamik des e-Business unterstützt.
- Durch die Technologieunabhängigkeit wird es einfacher, Systeme zu skalieren.
- Durch die Austauschbarkeit von Komponenten wird man weniger abhängig von einzelnen Herstellern, da man prinzipiell problemlos einzelne Komponenten austauschen kann.

Auch die erweiterten Möglichkeiten von Komponentenmodellen sind im e-Business sehr nützlich. Durch den hohen Grad an Wiederverwendung ist es wichtig, dass die Installation von

Komponenten standardisiert ist, und weil man häufig auch mit entfernten Komponenten zusammenarbeiten muss, deren genaue Fähigkeiten man nicht genau kennt, ist Introspection fast unerlässlich.

Außerdem bieten alle erfolgreichen Komponententechnologien natürlich weit mehr Möglichkeiten als nur die hier gegebene Minimaldefinition. Dadurch werden noch Ansätze zur Lösung weiterer Probleme gegeben; dies wird im Abschnitt 4.1 noch genauer erläutert.

4 Komponenten in der Praxis

Dieser Abschnitt beschäftigt sich mit der Frage, wie die im vorigen Abschnitt vorgestellten Konzepte sich im e-Business anwenden lassen. Dazu werden zunächst kurz einige Technologien vorgestellt und anschließend die Vorteile der Verwendung von Skriptsprachen zum Zusammenfügen der Komponenten angesprochen.

4.1 Technologien

Alle hier aufgeführten Technologien erfüllen die einfache Komponentendefinition mehr oder weniger gut; insbesondere werden jeweils verteilte Komponenten sowie deren Auffinden über eine Art zentrales Verzeichnis unterstützt. Unterschiede gibt es hauptsächlich im Grad der Technologieunabhängigkeit und eventuellen zusätzlichen Eigenschaften. Letztere sind allerdings nicht ganz einfach zu bewerten, da oft große Teile der Spezifikation optional sind und man sich somit nicht auf deren Implementierung verlassen kann; deshalb verzichten wir hier weitgehend darauf, auf diese einzugehen.

Zur besseren Vergleichbarkeit werden jeweils folgende Punkte betrachtet:

- Enthält die Technologie ein **Komponentenmodell**?

  Wie in Abschnitt 3.1 erläutert, stellen Komponentenmodelle zusätzliche Richtlinien dar, die die Verwendung der Komponenten erleichtern.

- Ist die Technologie **unabhängig von der Programmiersprache**?

  Bei der Fixierung auf *eine* Programmiersprache hat man den Nachteil, dass Entwickler eventuell erst umgeschult werden müssen, bevor sie mit dieser Technologie arbeiten können; außerdem ist keine Programmiersprache für alle Anwendungen gleich gut geeignet.

- Ist die Technologie **Plattformunabhängig**?

  Wenn das der Fall ist, so wirkt sich das positiv auf die Skalierbarkeit aus; ist man zum Beispiel auf ein Betriebssystem beschränkt, so wird man auch in der Hardwareauswahl deutlich eingeschränkt.

- Handelt es sich um eine **e-Business-Plattform**?

  Eine e-Business-Plattform bietet zusätzlich noch Funktionalität, die speziell auf die Anforderungen des e-Business zugeschnitten ist.

- Laufen die Komponenten in einer eigenen **Laufzeitumgebung** ab?

  Eine eigene Laufzeitumgebung wie beispielsweise die Java Virtual Machine hat den Vorteil, dass die Komponenten vom Rest des Systems abgeschirmt sind und nur mit expliziter Erlaubnis darauf zugreifen können.

OMG

CORBA (Common Object Request Broker Architecture) ist eine reine Familie von Spezifikationen, es gibt keine Referenzimplementierung. Spezifiziert wird es von der Object Management Group (OMG), einem Konsortium, dem viele bekannte Firmen aus der IT-Branche angehören; eine Übersicht über die Spezifikationen ist bei Object Management Group (2002) zu finden. Durch das dadurch vorgegebene weite Spektrum an Interessen wird Unabhängigkeit von Herstellern, Plattformen und Programmiersprachen erreicht, was allerdings auch zur Folge hat, dass bei keinem dieser Punkte spezielle Vorteile ausgereizt werden können. In den älteren Versionen wurden lediglich verteilte Komponenten unterstützt, erst in der neuen Version 3.0 wurde CORBA um ein Komponentenmodell, nämlich das CORBA Component Model (CCM), erweitert. In CCM ist auch schon die Zusammenarbeit mit EJBs, die gleich noch besprochen werden, vorgesehen, denn es ist möglich, CORBA-Komponenten als EJBs zu „verkleiden“.

Microsoft

Im Gegensatz zu CORBA stellt Microsoft zu DCOM nicht nur die Spezifikation, sondern auch die Implementierung, die allerdings nur unter Windows läuft und auch so eng mit den Interna dieses Betriebssystems verzahnt ist, dass es sehr schwer ist, DCOM auf anderen Systemen umzusetzen. Ein weiteres Problem ist, dass DCOM zwar theoretisch sprachunabhängig ist, aber auf einem Binärformat basiert, dass von Microsoft-Visual-C^{++} ausgegeben wird; eine Umsetzung für andere Programmiersprachen ist daher nicht ganz einfach.

Mittlerweile ist DCOM allerdings auf dem Rückzug, da Microsoft mit .NET (Spezifikation bei Microsoft Corporation (2001)) ein Komponentensystem, das neben der Sprachunabhängigkeit – allerdings wird C♯ bevorzugt, das extra von Microsoft für diese Plattform entwickelt wurde – auch Betriebssystemunabhängigkeit bieten soll. Microsoft .NET basiert auf Web-Service-Technologien, die in Abschnitt 5 noch näher erläutert werden, und bietet eine komplette Laufzeitumgebung, was den Vorteil hat, dass Komponenten ohne explizite Erlaubnis nicht auf Daten außerhalb des „Sandkastens“ zugreifen können. Bei .NET handelt es sich um eine e-Business-Plattform, wobei Microsoft noch wesentlich mehr Funktionalität integriert als Sun mit J2EE.

Laut Microsoft soll .NET plattformunabhängig sein, allerdings existiert bis jetzt nur eine Version für Windows, und Microsofts übliche Geschäftspraxis lässt es zumindest fraglich erscheinen, ob sich das in absehbarer Zeit ändern wird: Es gibt zwar zumindest ein Open-Source-Projekt – Ximians Mono Project, siehe Wagner (2001) – das jedoch bis heute noch weit von einer vollständigen Implementierung entfernt zu sein scheint.

Technologie	spezifiziert durch	Komponentenmodell	Unabhängigkeit		e-Business-Plattform	Laufzeitumgebung
			Sprache	Plattform		
CORBA	OMG		•	•		
CCM		•	•	•		
Java/RMI	Sun			•		•
J2EE		•		•	•	•
DCOM	Microsoft		(•)			
.NET		•	(•)	?	•	•

• trifft zu
(•) trifft mit Einschränkungen zu
? noch nicht sicher

Tabelle 1: Vergleich der behandelten Komponententechnologien

Sun

Die Lösung von Sun ist – wie man es erwarten konnte – voll auf Java zugeschnitten. Zum einen gibt es Java Remote Method Invocation (RMI) als eine Technologie für verteilte Objekte, die sich etwa mit CORBA bis einschließlich Version 2 und DCOM vergleichen lässt.

Zum anderen gibt es die Java 2 Enterprise Edition (J2EE), die wie .NET eine Spezifikation für eine komplette e-Business-Plattform darstellt, mit Enterprise JavaBeans (EJBs) als Komponentenmodell. Als Kommunikationstechnologie kann dabei auch auf CORBA zurückgegriffen werden, so dass (wie eben schon erwähnt) ein gewisser Grad an Interoperabilität erreicht wird. Eine Übersicht über J2EE sowie die Spezifikationen findet man bei Sun Microsystems (2002).

Da alles auf Java ausgerichtet ist, passt es auch gut zu Java und wirkt nicht wie ein Fremdkörper, wie das teilweise bei sprachunabhängigen Technologien der Fall sein kann, allerdings wird es so natürlich auch praktisch unmöglich gemacht, andere Programmiersprachen zu verwenden. Bekanntermaßen bietet auch Java eine eigene Laufzeitumgebung an.

Ein deutlicher Vorteil von J2EE gegenüber dem direkten Konkurrenten .NET besteht darin, dass es schon relativ lange Implementierungen von diversen Herstellern gibt und diese somit auch schon praxiserprobt und stabil sind.

Tabelle 1 gibt noch einmal einen Überblick über die Gemeinsamkeiten und Unterschiede der betrachteten Technologien. Einen detaillierten Vergleich der am weitesten fortgeschrittenen Technologien J2EE und .NET findet man bei Vawter und Roman (2001).

4.2 Skriptsprachen als Kleber

Da zum einen auch Informatik-Laien Komponenten zusammenzusetzen können sollten, zum anderen aber der Einarbeitungsaufwand für „normale“ Programmiersprachen wie Java oder C^{++} zu hoch ist, braucht man eine andere Lösung. Skriptsprachen wie Perl, Python, Tcl oder VisualBasic sind für diesen Zweck besonders geeignet, denn laut Ousterhout (1998) – dem Schöpfer der Skriptsprache Tcl – ist es gerade eine besondere Eigenschaft von Skriptsprachen, dass sie nicht dafür gedacht sind, Programme von Grund auf zu schreiben. Vielmehr sind sie dafür ge-

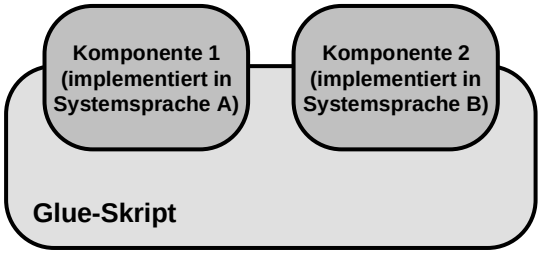

Abbildung 1: Skriptsprachen als Komponentenkleber

dacht, Programme aus Bestandteilen, die in anderen Programmiersprachen geschrieben worden sind, zu kombinieren: Die einzelnen Komponenten werden in einer systemnäheren Sprache implementiert und dann durch ein Skript zusammengesetzt (vergleiche Abbildung 1).

Eine wesentliche Eigenschaft von Skriptsprachen ist der Verzicht auf eine strenge Typprüfung zur Übersetzungszeit. Das mag auf den ersten Blick wie ein Nachteil erscheinen, aber zum einen wird nicht komplett auf Typprüfung verzichtet, sie wird nur zeitlich verschoben, was auch zu mehr Flexibilität führt. Zum anderen führt Strong Typing in vielen Fällen dazu, dass mehr Code geschrieben werden muss, nur um temporäre Objekte passender Typen zu erstellen.

Weiterhin sind Skriptsprachen interpretiert, was dazu führt, dass der gerade für Anfänger lästige Edit-Compile-Link-Zyklus wegfällt. Der Geschwindigkeitsnachteil, der dabei zwangsläufig entsteht, fällt bei den vorgesehenen Anwendungen – insbesondere auch bei der Verwendung als Komponentenkleber – nicht ins Gewicht.

Ein Beispiel, bei dem deutlich wird, wie gut Komponenten im weiteren Sinn und Skriptsprachen zusammenarbeiten können, ist die UNIX-Philosophie. Fast alle UNIX-Programme, die Texte bearbeiten, sind als Filter implementiert, lesen also von der Standardeingabe und schreiben auf die Standardausgabe. Diese Programme kann man als eine Art Komponenten ansehen, die beispielsweise über Pipes, also die Verknüpfung der Ausgabe eines Programms mit der Eingabe eines anderen Programms, verbunden werden können.

Folgende Befehlskette ergibt eine nach steigender Anzahl sortierte Liste der in der Datei `text` vorkommenden Wörter:

```
tr -sc 'a-zA-ZäöüÄÖÜß' "\n" < text | sort
  | uniq -c | sort -n
```

Da UNIX-Shells typischerweise auch über Programmkonstrukte wie Schleifen und Unterprogramme verfügen, kann man sie bedenkenlos als Skriptsprachen betrachten, mit denen man die Komponenten zusammenfügen kann.

Auch die Hersteller der großen Komponententechnologien haben die Bedeutung von Skriptsprachen erkannt. Während Microsoft hauptsächlich auf das eigene VisualBasic beziehungsweise dessen Weiterentwicklung VisualBasic.NET setzt, gibt sich die OMG bei CORBA wieder unabhängiger: Zum einen wird eine CORBA-eigene Skriptsprache unterstützt und ein Language Binding für Python mitgeliefert, zum anderen ist es auch in diesem Bereich möglich und erwünscht, Bindings für andere Sprachen zu erstellen.

4.3 Probleme

Theoretisch sollten sich also viele Probleme des e-Business mit Komponenten lösen lassen. Leider funktioniert das in der Praxis nicht immer so gut, wie man sich es wünschen würde; wie wir gleich sehen werden liegt das aber weniger an den grundlegenden Konzepten oder den technischen Aspekten der Komponentensysteme, sondern vielmehr daran, dass dieser Bereich noch relativ jung ist.

Leider ist der Grad der Wiederverwendung momentan nicht so hoch, wie es technisch möglich wäre; das liegt zumindest teilweise daran, dass es noch keinen ausreichend großen Marktplatz für Komponenten gibt. Es ist somit nicht ganz einfach, festzustellen, ob es für eine benötigte Funktionalität schon vorgefertigte Komponenten gibt, und wenn man gar verschiedene Anbieter vergleichen will, muss man den Suchaufwand mehrmals betreiben.

Ein weiteres Problem besteht darin, dass die Austauschbarkeit von Komponenten momentan kaum möglich ist: Verschiedene Hersteller verwenden in der Regel auch verschiedene Schnittstellen, auch wenn eigentlich die gleiche Funktionalität implementiert wird. Um dieses Problem zu lösen ist es nötig, dass (Industrie-)Standards entstehen, in denen für gewisse Dienste festgelegt wird, wie die entsprechende Schnittstelle aussehen soll.

Momentan ist es auch so, dass sich noch keine Technologie wirklich durchsetzen konnte. Der ohnehin relativ kleine Markt ist somit noch zersplittert, was dadurch zum Problem wird, dass die Zusammenarbeit verschiedener Modelle nicht sehr ausgereift ist, wobei allerdings zumindest die Zusammenarbeit von EJBs und CORBA-Komponenten machbar sein sollte. Man wird also gezwungen, sich von vorneherein auf eine Technologie festzulegen und muss sich dann praktisch auf Komponenten beschränken, die diese Technologie ebenfalls unterstützen.

Ein Ansatz zur Lösung einiger dieser Probleme ist durch die Web Services gegeben, die im folgenden Abschnitt besprochen werden.

5 Web Services

Als Lösungsansatz für die in Abschnitt 4.3 vorgestellten Probleme zeichnen sich die von IBM spezifizierten Web Services ab, deren Grundidee es ist, Dienste über weit verbreitete Internet-Protokolle – insbesondere aus dem Bereich des WWW – anzubieten.

5.1 Eigenschaften

Eine klare Definition für Web Services zu finden ist nicht einfach. Die folgende, von IBM stammende Definition (zitiert nach Vasudevan (2001)) ist weitgehend nichtssagend:

> *Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes [...] Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.*

Nach dieser Beschreibung klingt es zunächst einmal so, als ob Web Services einfach Programme (oder auch Komponenten) wären, die über das Web aufrufbar sind. Normalerweise

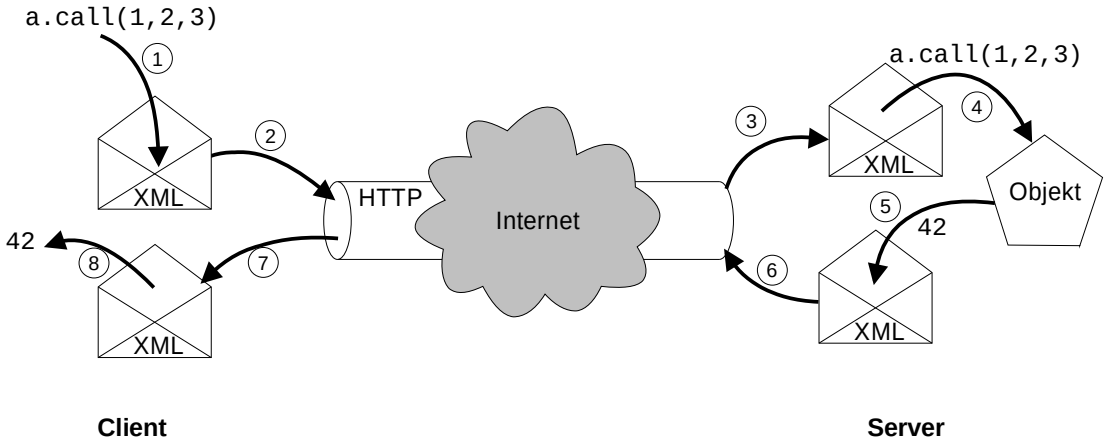

Abbildung 2: Funktionsweise von SOAP

werden aber mit diesem Begriff auch eine Menge an Technologien und Standards verbunden, die später noch vorgestellt werden. Die Grundidee ist der der Komponenten recht ähnlich, allerdings wird hier zum einen die Objektorientierung stärker betont und zum anderen das WWW als Medium festgeschrieben.

Für diesen Artikel betrachten wir Web Services als Objekte im Sinne der objektorientierten Programmierung, deren Methoden über das Web aufgerufen werden können – also eine Art von objektorientiertem Remote Procedure Call (RPC). Diese voneinander unabhängigen Objekte stellen jeweils bestimmte Dienste zur Verfügung, die von anderen automatisch aufgefunden und benutzt werden können.

5.2 Technologien

Da die Standardisierung von Web-Service-Technologien noch nicht abgeschlossen ist, kann es noch deutliche Änderungen geben; die in diesem Abschnitt gegebenen Beschreibungen basieren auf Vasudevan (2001). Es ist außerdem zu beachten, dass das nur eine kleine Auswahl ist, da ständig neue Standards entwickelt werden.

SOAP

Die grundlegendste Technologie – und die, die im Allgemeinen untrennbar mit dem Begriff Web Services verbunden wird – ist SOAP, das Simple Object Access Protocol (Spezifikation zu finden bei Box u. a. (2000)). Dieses Protokoll legt fest, wie RPCs in XML „eingepackt“ und dann über Internet-Basisprotokolle wie HTTP verschickt werden können (siehe Abbildung 2).

Ein entscheidender Vorteil hierbei ist der geringe Aufwand, der hierfür benötigt wird: Es ist nicht schwer, mit Hilfe einer XML- und einer Netzwerkbibliothek einen einfachen SOAP-Client oder -Server zu implementieren, und der Resourcenverbrauch in Hinblick auf Programmcode dafür ist vergleichsweise klein, so dass man Web Services auch von PDAs aus benutzen kann.

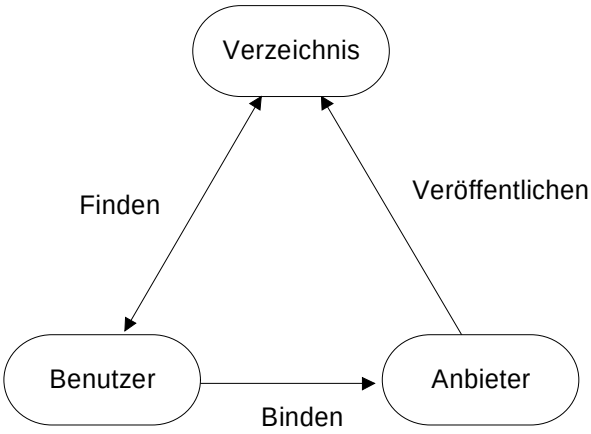

Abbildung 3: Veröffentlichen, Finden und Binden von Web Services

Demgegenüber steht jedoch leider eine deutliche Aufblähung der Daten durch das Einpacken in XML, was die Übertragungszeit deutlich erhöht; das fällt insbesondere bei mobilen Geräten ins Gewicht, auch wenn sich das vermutlich in naher Zukunft durch UMTS etwas relativiert.

Verzeichnisdienste

Bevor ein Web Service benutzt werden kann, muss er zunächst einmal bekannt gemacht werden. Dazu muss er zunächst von seinem Bereitsteller beschrieben werden. Das kann beispielsweise mit WSDL (Web Services Definition Language) geschehen. WSDL ist eine XML-basierte Sprache, mit der beschrieben werden kann, was der Web Service für einen Dienst anbietet, wo er zu finden ist und wie man ihn ansprechen kann. Die Spezifikation zu WSDL ist bei Christensen u. a. (2001) zu finden.

Anschliessend wird diese Beschreibung über einen Universal Description, Discovery and Integration Service (UDDI) veröffentlicht, der wiederum auf SOAP basiert. Über diesen Dienst können dann die Anwender den Dienst finden, wobei allerdings momentan nur relativ einfache Anfragen unterstützt werden; beispielsweise ist es momentan nicht möglich, zur Suche auch die geographische Lage des Dienstes zu benutzen.

Diese Prozedur wird in Abbildung 3 verdeutlicht; die UDDI-Spezifikationen findet man bei UDDI.org (2001).

Transaktionen

Gerade im e-Business ist es nötig, Transaktionen zu verwenden, die dem ACID-Paradigma entsprechen, da es in diesem Bereich um viel Geld gehen kann. Allerdings werden die Aktionen in den seltensten Fällen aus nur einer Teiloperation bestehen, vielmehr werden meist die Dienste mehrerer Anbieter benutzt werden, und erst wenn feststeht, dass alle Operationen gut gehen werden, dürfen die Ergebnisse wirklich festgeschrieben werden; dadurch ist eine verteilte Transaktionsverwaltung unbedingt nötig. Es gibt verschiedene Möglichkeiten, an dieses Problem heranzugehen:

Man kann optimistisch davon ausgehen, dass alle Teiloperationen erfolgreich sein werden und für den Nichterfolgsfall Kompensationsoperationen vorsehen; dies wird für Web Services durch XLANG (siehe Thatte (2002)) spezifiziert. Dadurch wird allerdings ACID nicht komplett garantiert, da zumindest die Atomität der Aktionen im Fehlerfall nicht gegeben ist. So kann man zwar eine fälschliche Kontoabbuchung nachträglich durch eine Gutschrift kompensieren, aber idealerweise sollte gar nicht erst abgebucht werden, wenn die Gesamttransaktion fehlschlägt.

Eine mögliche Lösung ist die Verwendung eines verteilten Zwei-Phasen-Commit-Protokolls (2PC). Dieses Protokoll wird (neben anderen) durch die Transaction Authority Markup Language TAML spezifiziert.

Welche dieser Lösungen verwendet werden sollte, hängt stark von den Umständen ab. Obwohl 2PC in den meisten Fällen wünschenswert ist, ist es teilweise sinnvoll, Kompensationsoperationen zu verwenden, da diese deutlich billiger zu implementieren und durchzuführen sind – allerdings gibt es gerade im Geschäftsbereich nur wenige Operationen, die sich im nachhinein kompensieren lassen.

Sicherheit

Da Sicherheit im e-Business, wie schon erwähnt, eine große Rolle spielt, gibt es auch für Web Services Ansätze, um diese zu behandeln. In diesem Fall zeigt sich deutlich ein Vorteil der Verwendung des WWW als Basistechnologie: Es lassen sich auch Standards verwenden, die nicht im Hinblick auf Web Services entworfen wurden, sondern allgemein auf das WWW zugeschnitten sind. Beispielsweise kann man zur Datenübertragung statt des HTTP-Protokolls auch das verschlüsselte HTTPS-Protokoll verwenden, um das Abhören sensibler Geschäftsdaten zu erschweren.

Microsoft bietet mit dem Passport-Service für .NET (http://www.passport.com) einen kostenlosen Dienst, der es Endnutzern ermöglicht, sich gegenüber Web Services, die mit .NET implementiert sind, zu authentifizieren. Hierzu werden neben dem Namen und der Anschrift auch Kreditkartennummer gespeichert, so dass der Benutzer – genügend Vertrauen vorausgesetzt – problemlos und ohne die immer wiederkehrende Eingabe persönlicher Details einkaufen kann.

Ein Standard zur Zertifizierung und Signaturen, der wie Web Services auf XML, SOAP und WSDL basiert, ist die XML Key Management Specification (XKMS, Spezifikation zu finden bei Ford u. a. (2001)), die von Microsoft und Verisign stammt. XKMS verlagert Signaturprozesse generell auf einen Trust Server, so dass auch leistungsschwache Endgeräte wie PDAs von diesen Diensten profitieren können.

5.3 Einfaches Beispiel für SOAP in Perl

Um zu verdeutlichen, dass die Handhabung von SOAP wirklich sehr einfach ist, werden wir in diesem Abschnitt einen einfachen Server und Client vorstellen, implementiert in Perl unter Verwendung von `SOAP::Lite` in der Version 0.52 (siehe Kulchenko (2002)).

Server

Um einen Server zu implementieren, muss man sich zunächst überlegen, wie dieser Server angesprochen werden soll. In diesem Beispiel (siehe Listing 1) verwenden wir einen eigenständigen

```
1 #!/usr/local/ActivePerl-5.6/bin/perl -w
2 use strict;
3 use SOAP::Transport::HTTP;
4
5 SOAP::Transport::HTTP::Daemon->new(LocalPort => 9001)
6   ->dispatch_to('Test')->handle;
7
8 package Test;
9 sub dividiere {
10  my ($class, $dividend, $divisor) = @_;
11
12  die "Man kann nicht durch 0 teilen!" if $divisor == 0;
13  return $dividend / $divisor;
14 }
```

Listing 1: Ein einfacher SOAP-Server

Server, der direkt über HTTP angesprochen werden kann, aber mit minimalen Änderungen wäre beispielsweise auch ein CGI-Server denkbar.

In Zeile 3 wird das passende Perl-Package geladen, und in den Zeilen 5 und 6 wird dann ein Daemon installiert, der auf SOAP-Anfragen über HTTP auf Port 9001 wartet. Der Methodenaufruf `dispatch_to('Test')` gibt an, dass SOAP-Funktionsaufrufe direkt in Aufrufe der Funktionen im Package `Test` weitergeleitet werden. Eine explizite Deklaration der Funktionen ist nicht nötig; wenn der Client versucht, eine nicht vorhandene Funktion aufzurufen, bekommt er eine entsprechende Fehlermeldung zurück.

Das war auch schon alles, was nötig ist, um den Server zu installieren, und somit fehlt nur noch die Definition der eigentlichen Funktionalität. Die im Package `Test` (Zeilen 8–14) enthaltene Funktion bekommt als Parameter zum einen die Klasse, die für objektorientierte Programmierung in Perl von Bedeutung ist, hier aber ignoriert wird, und die vom Aufrufer übergebenen Parameter. Diese Parameter sehen für die Funktion nicht anders aus als bei einem normalen Funktionsaufruf, und auch die Rückgabe von Werten erfolgt auf die normale Weise über `return`. Die Perl-Variante von Exception Handling über `die` wird abgefangen und dem Client als Fehlermeldung zurückgegeben.

Client

Auch die Programmierung eines Clients ist nicht schwierig. Zunächst muss wieder das passende Modul eingebunden werden und eine Verbindung zum Server aufgebaut werden, was in Listing 2 in den Zeilen 3–6 geschieht.

Anschließend können die Methodenaufrufe fast wie bei „normalen“ Funktionen durchgeführt werden; der einzige wesentliche Unterschied ist, dass nicht direkt der Rückgabewert der Funktion zurückgegeben wird, sondern ein Objekt, das neben dem Rückgabewert auch noch eventuelle Fehlermeldungen enthält. Bei laufendem Server ergibt der Aufruf des Clients folgende Ausgabe:

```
#!/usr/local/ActivePerl-5.6/bin/perl -w
use strict;
use SOAP::Lite;

my $server = SOAP::Lite->uri('http://localhost:9001/Test')
  ->proxy('http://localhost:9001/');

my $result = $server->dividiere(10, 5);
if (!$result->fault) {
  print "Ergebnis: ", $result->result, "\n";
} else {
  print "Fehler: ", $result->faultstring, "\n";
}

$result = $server->dividiere(10, 0);
if (!$result->fault) {
  print "Ergebnis: ", $result->result, "\n";
} else {
  print "Fehler: ", $result->faultstring, "\n";
}
```

Listing 2: Ein einfacher SOAP-Client

```
Ergebnis: 2
Fehler: Man kann nicht durch 0 teilen! at ./server.pl line 12.
```

Insgesamt kann man sehen, dass es keine große Einarbeitungszeit braucht, um SOAP-Clients und -Server zu implementieren.

5.4 Beziehung zu anderen Komponententechnologien

Auch wenn Web Services von der Grundidee den Komponententechnologien stark ähnelt, sind sie nicht als Ersatz, sondern vielmehr als Ergänzung dazu gedacht. Da die verschiedenen Komponentenstandards weitgehend inkompatibel untereinander sind, bietet es sich an, Komponenten mit einer dieser „herkömmlichen“ Technologien zu implementieren und dann in ein Web-Service-Interface einzubetten. Dadurch werden die Interoperabilitätsprobleme umgangen, da es unerheblich ist, wie genau ein Web Service intern aufgebaut ist.

Ein weiterer Vorteil sind die relativ geringen technologischen Barrieren: XML und das Web sind mittlerweile allgegenwärtig, und die Nutzung von HTTP als Transportprotokoll ermöglicht es, Web Services hinter Firewalls zu installieren, da diese üblicherweise so konfiguriert sind, den entsprechenden Port durchzulassen.

Bei der Zusammenarbeit mit Web-Service-Technologien hebt sich Microsofts .NET besonders hervor, da es auf diesen Technologien aufsetzt und es somit sehr einfach ist, .NET-Komponenten als Web Services zur Verfügung zu stellen. Im Dezember 2001 wurde die Unterstützung von Web Services auch für J2EE angekündigt, und es ist absehbar, dass die OMG auch

bald nachziehen wird. Außerdem ist es auch ohne direkte Unterstützung möglich, die entsprechende Anbindung über XML-Bibliotheken mit relativ wenig Aufwand selbst zu realisieren, so dass auch das kein großer Hinderungsgrund wäre; außerdem existieren diverse inoffizielle Unterstützungsbibliotheken für Web Services. Damit ist die Verwendung von Web Services als Verbindung zwischen den verschiedenen Komponententechnologien in greifbare Nähe gerückt.

6 Fazit

Die Grundidee von Komponentenmodellen scheint gut geeignet zu sein, viele der e-Business-spezifischen Probleme auf elegante Weise zu lösen. Leider ist es in der Praxis noch ein Problem, dass sich (noch?) kein klarer Gewinner unter den Komponententechnologien abzeichnet und dass spezielle Standards für e-Business-Komponenten noch fehlen.

Web Services können zumindest teilweise die Interoperabilitätsprobleme der Technologien lösen, und wegen der Unterstützung durch Microsoft kann man annehmen, dass sie auf lange Sicht erfolgreich sein werden. Im Moment scheint es allerdings wenig ratsam, zu stark auf die Verwendung von Web Services zu setzen, da die entsprechenden Spezifikationen zum Teil noch nicht endgültig sind und sich somit noch deutlich ändern können; so findet sich beispielsweise sowohl in der SOAP- als auch in der XKMS-Spezifikation (Ford u. a. (2001) und Box u. a. (2000)) folgender Hinweis: *This document is a work in progress and may be updated, replaced, or rendered obsolete by other documents at any time.*

Trotz der noch vorhandenen Kinderkrankheiten können Komponenten auch heute schon sinnvoll eingesetzt werden. So findet sich beispielsweise bei Iona Technologies eine Studie, in der beschrieben wird, wie Nike Dienste für den Kundenservice neu mit CORBA implementieren ließ, und bei Microsoft Corporation (2002) finden sich diverse Fallstudien zur erfolgreichen Verwendung von .NET und Web Services. Bei all diesen Beispielen ist natürlich zu beachten, dass die Studien von den jeweiligen Herstellern der Komponententechnologien veröffentlicht wurden und somit sicher nicht objektiv sind, aber alleine die Tatsache, dass multinationale Konzerne wie Nike, DaimlerChrysler und Continental Airlines sich für diese Technologien interessieren, zeigt, dass Komponenten im e-Business Potential haben.

Literatur

Box u. a. 2000 BOX, Don ; EHNEBUSKE, David ; KAKIVAYA, Gopal ; LAYMAN, Andrew ; MENDELSOHN, Noah ; NIELSEN, Henrik F. ; THATTE, Satish ; WINER, Dave: *Simple Object Access Protocol (SOAP) 1.1*. Mai 2000. – URL http://www.w3.org/TR/SOAP/. – Zugriffsdatum: 15. 01. 2002

Christensen u. a. 2001 CHRISTENSEN, Erik ; MEREDITH, Greg ; CURBERA, Francisco ; WEERAWARANA, Sanjiva: *Web Services Description Language (WSDL) 1.1*. Januar 2001. – URL http://msdn.microsoft.com/library/?url=/library/en-us/dnwebsrv/html/wsdl.asp. – Zugriffsdatum: 24. 01. 2002

Ciupke und Schmidt 1997 CIUPKE, Oliver ; SCHMIDT, Rainer: Components as Context-Independent Units of Software. In: MÜHLHÄUSER, Max (Hrsg.): *Special Issues in Object-Oriented Programming*. Heidelberg : dpunkt, 1997, S. 139–143

Ford u. a. 2001 FORD, Warwick ; HALLAM-BAKER, Phillip ; FOX, Barbara ; DILLAWAY, Blair ; LAMACCHIA, Brian ; EPSTEIN, Jeremy ; LAPP, Joe: *XML Key Management Specification (XKMS)*. March 2001. – URL http://www.w3.org/TR/xkms/. – Zugriffsdatum: 24. 01. 2002

Iona Technologies IONA TECHNOLOGIES: *NIKE – doing IT*. – URL http://www.iona.co.jp/aboutus/customers/pdf/nike.pdf. – Zugriffsdatum: 12. 12. 2001

Kulchenko 2002 KULCHENKO, Paul: *SOAP::Lite, XMLRPC::Lite and UDDI::Lite for Perl*. 2002. – URL http://www.soaplite.com/. – Zugriffsdatum: 18. 01. 2002

Microsoft Corporation 2001 MICROSOFT CORPORATION: *XML Web Service Specifications*. 2001. – URL http://www.gotdotnet.com/team/xml_wsspecs/default.aspx. – Zugriffsdatum: 24. 01. 2002

Microsoft Corporation 2002 MICROSOFT CORPORATION: *XML Web Services Case Studies*. Januar 2002. – URL http://www.microsoft.com/net/use/casestudies.asp. – Zugriffsdatum: 12. 01. 2002

Object Management Group 2002 OBJECT MANAGEMENT GROUP: *Catalog of OMG CORBA/IIOP Specifications*. 2002. – URL http://www.omg.org/technology/documents/corba_spec_catalog.htm. – Zugriffsdatum: 24. 01. 2002

Ousterhout 1998 OUSTERHOUT, John K.: *Scripting: Higher Level Programming for the 21st Century*. März 1998. – URL http://home.pacbell.net/ouster/scripting.html. – Zugriffsdatum: 25. 11. 2001

Segev und Bichler 1999 SEGEV, A. ; BICHLER, M.: Component-based Electronic Commerce. In: SHAW, M. (Hrsg.) ; BLANNING, R. (Hrsg.) ; STRADER, T. (Hrsg.) ; WHINSTON, A. (Hrsg.): *Handbook on Electronic Commerce*. Heidelberg : Springer Verlag, 1999, S. 313–337

Sun Microsystems 2002 SUN MICROSYSTEMS: *Java 2 Platform, Enterprise Edition – Downloads & Specifications*. 2002. – URL http://java.sun.com/j2ee/download.html. – Zugriffsdatum: 24. 01. 2002

Thatte 2002 THATTE, Satish: *XLANG – Web Services for Business Process Design*. 2002. – URL http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm. – Zugriffsdatum: 24. 01. 2002

UDDI.org 2001 UDDI.ORG: *UDDI: Specifications*. 2001. – URL http://www.uddi.org/specification.html. – Zugriffsdatum: 15. 01. 2002

Vasudevan 2001 VASUDEVAN, Venu: *A Web Services Primer*. April 2001. – URL http://xml.com/pub/a/2001/04/04/webservices/index.html. – Zugriffsdatum: 19. 11. 2001

Vawter und Roman 2001 VAWTER, Chad ; ROMAN, Ed: *J2EE vs. Microsoft.NET*. Juni 2001. – URL http://www.theserverside.com/resources/pdf/J2EE-vs-DotNET.pdf. – Zugriffsdatum: 11. 01. 2002

Wagner 2001 WAGNER, Mitch: *.Net To Be Open Source*. Juli 2001. – URL http://www.internetwk.com/story/INW20010710S0006. – Zugriffsdatum: 15. 01. 2002