SEMINARARBEIT

in der Fachrichtung

Wirtschaftsinformatik

Thema:

Service-orientierte Software-Architekturen

-

Strategien für die Integration verschiedener Anwendungssysteme

Eingereicht von: Vanessa Postel
Rathausplatz 1a
22880 Wedel
Tel. (0 4103) 7019271

Erarbeitet im: 7. Semester

Abgegeben am: 14. Dezember 2005

Referent (FH Wedel): Prof. Dr. Sebastian Iwanowski
Fachhochschule Wedel
Feldstraße 143
22880 Wedel

INHALTSVERZEICHNIS

1. Einführung

1.1 Problemstellung

Ein kurzfristiges Ziel, das möglicherweise durch den Einsatz von Web Services erreicht werden soll, ist die Veröffentlichung der Funktionalität von Anwendungen, um auch endlich mit der Zeit zu gehen und Web Services anzubieten. Langfristig gesehen ist es aber vorteilhafter nichts zu überstürzen, sondern es langsam angehen zu lassen. Denn ein derartiger Schritt will wohl überlegt sein, da er ansonsten eine Reihe von Problemen nach sich ziehen kann. Diese wiederum zu beheben, ist dann unter Umständen nicht so einfach und vor allem in den meisten Fällen nicht auf die Schnelle möglich. Daher kann es sein, dass potentielle Kunden, die den Service gerade zu dem Zeitpunkt nutzen wollten, als dieser noch mit Problemen behaftet war, sofort wieder abspringen, da sie natürlich nicht mit dem Service zufrieden sind. Für den Serviceanbieter bedeutet dies, dass er zunächst einige Rückschläge hinnehmen muss oder gegebenenfalls in diesem Bereich gar nicht mehr Fuß fassen kann, bevor er überhaupt in den Genuss der Vorteile gekommen ist. Neben dem richtigen Zeitpunkt gibt es aber natürlich noch andere wichtige Aspekte, die vor der Veröffentlichung eines Web Services bedacht werden müssen. Dazu gehört beispielsweise auch, sich darüber zu informieren, welche Möglichkeiten einem überhaupt mit einem bereits vorhandenen System zur Verfügung stehen. Kann das System mit einem Web Service erweitert werden oder muss ein komplett anderes System gewählt werden? Sollte dies der Fall sein, wie können die Daten vom alten auf das neue System übertragen werden? Welche Funktionalitäten sollen im Einzelnen zur Verfügung gestellt werden? Welche Anforderungen stellt das eigene Unternehmen an den Web Service und welche ein potentieller Kunde?

1.2 Gang der Untersuchung

Im Rahmen dieser Seminararbeit soll anhand des Buches „Service-Oriented Architecture“ von Thomas Erl sowie weiterführender Literatur ein Überblick über verschiedene Strategien zur Erstellung eines Netzwerkes gegeben werden. Jede dieser Strategien besteht wiederum aus einer Reihe von unterschiedlichen Vorschlägen, wie verfahren werden kann, um einen Web Service zu integrieren. Diese können sowohl einzeln als auch in Kombination eingesetzt werden.
Dazu wird im zweiten Kapitel zunächst einmal auf den Aufbau von Schnittstellen eingegangen, denn diese sind ein sehr wichtiger Teil einer Servicearchitektur, da sie die

Kontaktpunkte für externe Benutzer repräsentieren. Ihr Design ist entscheidend für die Effizienz und Nutzbarkeit eines Web Services.
Im dritten Kapitel wird die interne Funktionalität von Web Services und wie diese gegenseitig aufeinander einwirken angesprochen, bevor dann in Kapitel vier Möglichkeiten dargelegt werden, wie vorhandene Architekturen erweitert beziehungsweise verbessert werden können.
Kapitel fünf befasst sich mit Problemen, die durch den Einsatz von Enterprise Application Integration-Lösungen(EAI) auftreten können und zum Abschluss wird in Kapitel sechs das Thema Web Service-Sicherheit behandelt. Denn je mehr Funktionalität Web Services bieten, desto wichtiger ist natürlich auch der Aspekt der Sicherheit.
Abschließend wird noch einmal ein Fazit gezogen.

2. Integration von Endpunktschnittstellen

Die Funktionalität von Schnittstellen sollte soweit wie möglich vereinfacht werden, damit der Web Service möglichst effektiv eingesetzt werden kann. Dies kann entweder dadurch erreicht werden, dass mehrere Methoden eines Systems oder Programms zu einer Serviceoperation zusammen gefasst werden, wenn diese beispielsweise eine ähnliche Logik haben, oder indem eine größere Methode in mehrere kleine Operationen aufgeteilt wird, was unter Umständen dann Sinn machen würde, wenn ein bestimmter Teil dieser Methode häufiger ausgeführt wird als andere. Die Schnittstellen können jederzeit an sich neu ergebenden Anforderungen angepasst werden. Derartiges sollte aber spätestens dann gesehen, wenn sich Gegebenheiten im Unternehmen ändern. Sollte also ein Web Service später in einem anderen Umfang eingesetzt werden, als es ursprünglich gedacht war, lassen sich möglicherweise sogar mehrere Schnittstellen zu einer sinnvolleren zusammenfassen und damit die Performance steigern. In mehrschichtigen Anwendungen kommen häufig Proxy-Schnittstellen zum Einsatz. Diese sind Stellvertreter der Anwendungen, für die Serviceschnittstellen benötigt werden und befinden sich zwischen Anwendung und Client. Wird vom Client eine Anfrage gestellt, leitet der Proxy diese entweder an die Anwendung weiter oder bearbeitet sie selbstständig anhand von abgespeicherten Daten, falls die gleiche Anfrage schon einmal gestellt wurde. Um eine Interaktion mit anderen Serviceumgebungen zu ermöglichen, sollten Proxy-Services, die dies von Natur aus nicht unterstützen, durch einen Businessservice ersetzt werden. Businessaufgaben werden ab dem Zeitpunkt ausschließlich über ihn abgewickelt. Diese Lösung hat noch einmal eine wesentliche Verbesserung der Performance zur Folge.

Eine ähnliche Funktionalität wie sie Proxy-Schnittstellen bieten, kann durch eine Ergänzung der bestehenden Logik um eine externe Logik erreicht werden. Dabei erhält der Web Service zusätzlich zu seiner eigentlichen Logik eine weitere, die es ihm ermöglicht die Daten, die er von Anwendungen erhält, eigenständig zu bearbeiten und in der Form zusammenzustellen, wie er sie benötigt. Beispielsweise hat in homogenen Umgebungen häufig ein einzelner Service Zugriff auf eine ganze Reihe von Anwendungen. Auf diese Weise kann er die Teile der Funktionalität nutzen, die er gerade benötigt, um eine Serviceoperation auszuführen.

Neben Möglichkeiten wie Schnittstellen vereinfacht werden können, sollten auch Features in Betracht gezogen werden, die ihre Effizienz verbessern, zum Beispiel die Option verschiedene Datenausgabeformate zu unterstützen. Auch wenn die Umsetzung vielleicht nicht ganz einfach ist, wird dadurch die Kompatibilität enorm verbessert und die anfänglichen Mühen machen sich auf jeden Fall in Zukunft bezahlt. Des Weiteren ist es nützlich, alternative Schnittstellen für verschiedene SOAP-Clients bereitzustellen, da unterschiedliche Nachrichtenformate natürlich auch anders verarbeitet werden müssen. Zum Beispiel

unterstützen viele Serviceschichten SOAP-Nachrichten, die einen RPC beinhalten. Ändern sich die Anforderungen an die Anwendung, kann es sinnvoll sein, dieses Format durch ein dokumentenbasiertes SOAP-Nachrichtenformat auszutauschen. Ein weiterer Grund ist aber auch, dass nicht ständig bei jeder neuen Anfrage ein RPC durchgeführt werden muss, um die gewünschten Daten zu erhalten, sondern die Daten aus dem Speicher geladen werden können. Um den Übergang vom RPC- zum dokumentenbasierten Format zu erleichtern, können zunächst beide Formate mit separaten Integrationsschichten unterstützt werden.

3. Optimierung der Integration von Endpunktdiensten

Unter dem Begriff „Endpunktdienst“ ist eine Umgebung zu verstehen, die eigene Anwendungen mittels standardisierten Serviceschnittstellen anderen Leuten zur Verfügung stellt. Werden zum ersten Mal Web Services eingesetzt, sollte die Architektur nicht unnötig verkompliziert, sondern so einfach wie nur möglich gehalten werden, da ansonsten Performanceprobleme auftreten können. Deshalb sollte zum Beispiel der Einsatz von Serviceintermediaries auf ein Minimum beschränkt werden. Diese sind Vermittler, deren Aufgabe es ist, Serviceanfragen an einen entsprechenden Service weiterzuleiten. Sie können sich jedoch in beliebiger Menge zwischen dem Client und dem Service befinden und machen die Architektur aufgrund dessen möglicherweise sehr komplex. Daher sollten sie lediglich für hub-artige Services außerhalb der Architektur verwendet werden.
Des Weiteren gibt es die Möglichkeit so genannte Serviceinterceptors einzusetzen. Diese sollen Abhängigkeiten zwischen der Servicelogik und einer proprietären Plattform reduzieren, indem sie entweder selbst die Bearbeitung von Routineprozessen, wie zum Beispiel Verwaltungsaufgaben, übernehmen oder sie an andere Services weiterleiten. Sind sie jedoch erst einmal implementiert, werden sie in der Regel auch sehr häufig verwendet und haben daher natürlich einen hohen Bedarf an Speicherplatz. Um dies zu verhindern, sollten sie möglichst klein gehalten werden. Ebenso wie Serviceinterceptors werden natürlich auch Web Services oft angesprochen. Um dann potentielle Engpässe zu vermeiden, sollte die Verarbeitung der übermittelten Daten an einem anderen Ort stattfinden als das Empfangen oder Übertragen. Dazu könnte beispielsweise der Serviceprovider benötigte Daten schon formatieren und validieren, bevor er sie versendet oder gegebenenfalls einen Validationscode in das Dokument einbetten. Der empfangende Service müsste dann nur noch diesen einen Teil der Nachricht überprüfen.
Um auch noch den Verarbeitungsoverhead zu verringern, der dann auftritt, wenn der Servicerequestor die WSDL-Definition zur Laufzeit auswertet und verarbeitet, kann das WSDL-Dokument des Providers gespeichert werden. Bei dieser Variante wird ein einziges

Mal eine Anfrage nach dem WSDL-Dokument an den Service gestellt. Nachdem es übermittelt wurde, wird es ausgewertet und gespeichert. Danach kann der Service jederzeit verwendet werden, ohne das Dokument erneut anfordern zu müssen. Es sei denn, das Dokument wird irgendwann vom Provider geändert. Dann kann der Service mit der alten WSDL-Definition nicht mehr benutzt werden und eine neue Anfrage ist erforderlich.

4. Integration von Altsystemen

Entschließt sich ein Unternehmen dazu Web Services einzusetzen, stellt sich an diesem Punkt zunächst die Frage, wie dies mit dem momentanen System vereinbar ist. Obwohl das oberste Ziel natürlich die Erweiterung der bereits integrierten Schichten ist, so dass alle Anwendungen darüber kommunizieren können, ist eine schrittweise Annäherung an eine service-orientierte Architektur ratsamer. Dazu könnte zunächst eine Übergangsarchitektur erstellt werden, indem lediglich eine Teilintegrationsschicht zur bestehenden Umgebung hinzugefügt wird. Diese Möglichkeit birgt weniger Probleme als eine sofortige Ersetzung der gesamten Umgebung, hat aber den Nachteil, dass die ohnehin schon umfangreiche Umgebung um eine weitere Schicht ergänzt wird.
Wurde ein Altsystem gerade erst durch Web Services erweitert, kann dies sehr schnell zu einem erhöhten Benutzeraufkommen führen. Um dann nicht an die Grenzen des Systems zu stoßen, gibt es eine simple Möglichkeit an dieser Stelle die Skalierbarkeit zu verbessern und zwar in Form einer Warteschlange. Diese nimmt dann alle Anfragen auf, die zurzeit nicht bearbeitet werden können. Um eine Anwendung darüber hinaus bei der Bearbeitung von Anfragen zu entlasten, kann die Serviceschicht zum Beispiel um eine In-Memory Database (IMDB) erweitert werden. Diese legt Daten im Hauptspeicher ab, die häufiger verwendet werden. Die benötigten Daten sind nun sofort verfügbar und die Anwendung muss sie nicht erst bei jeder neuen Anfrage wieder aus der Datenbank heraussuchen. Auf diese Weise können entsprechende Anfragen schneller bearbeitet werden. Sollen bestimmte Daten auch zukünftig häufiger verwendet werden, kann die Architektur jederzeit um den Aspekt der physischen Speicherung ergänzt werden. Um außerdem die Abhängigkeit der IMDB von der Anwendung zu minimieren, kann die IMDB in eine andere Umgebung ausgelagert werden.
Durch das Hinzufügen einer dritten Integrationsschicht haben fremde Anwendungen einen zentralen Zugriffspunkt. Diese Lösung wirkt wie ein Mini-Hub und hat ein wesentlich besseres Design zur Folge, da die verschiedenen Web Services nicht länger mit einander sondern nur noch mit dem Hub verbunden sind. Ab diesem Zeitpunkt erfüllt jeder Web Service einen anderen Zweck.

Werden in der vorhandenen Umgebung Web Services für eine Korrelation mit Adaptertechnologien benötigt, besteht die Möglichkeit einen so genannten Utility Layer hinzuzufügen, um die Businesslogik von proprietären Systemen unabhängig zu machen. Änderungen bei den Adaptern, die zuvor ebenfalls eine Änderung bei der Businesslogik erforderlich gemacht hätten, haben nun keinen Einfluss mehr auf die Logik. Um überprüfen zu können, ob der Utility Layer die Adapter überhaupt verwenden kann, steht ihm ein virtueller Adapter zur Verfügung, den er mit den Adaptern vergleicht.
Die Funktionalität vorhandener Architekturen können in einem Web Service auf einer standardisierten Integrationsschicht zusammengeschlossen werden. Die Unterstützung verschiedener Anwendungsumgebungen bietet den Vorteil der Unabhängigkeit von bestimmten proprietären Plattformen.
Sollen Anwendungen zu einer service-orientierten Umgebung hinzugefügt werden, um anderen Leuten den Zugriff auf Logik und Daten zu ermöglichen, muss dafür nicht zwangsläufig eine neue Architektur eingesetzt werden. Dies wäre nur dann sinnvoll, wenn grundlegende Abläufe verändert werden sollen. Ansonsten ist es vollkommen ausreichend einen Web Service einzufügen, um auch weiterhin eine gute Performance zu gewährleisten.

5. Integration von EAI-Lösungen

Enterprise Application Integration ist ein Konzept, das die unterschiedlichsten Anwendungen eines Unternehmens unterstützen soll. EAI-Produkte sind ausschließlich auf die Kommunikation mit EAI-Servern ausgelegt und haben den großen Nachteil nicht herstellerneutral zu sein. Da dies auf lange Sicht eine wenig zufrieden stellende Lösung ist, wird sie von immer weniger Unternehmen eingesetzt. Soll nun eine entsprechende EAI-Lösung um einen Web Service erweitert werden, führt dies zwangsläufig zu einem inkonsistenten System. Haben sich die Anforderungen, die an das System gestellt werden, in ihrem Ausmaß verändert, so dass sie nicht mehr von der EAI-Lösung erfüllt werden, sollte das System entweder durch ein besseres EAI-Produkt oder aber eine service-orientierte Architektur, wie zum Beispiel einen Enterprise Service BUS (ESB), ersetzt werden.
Falls in einem Unternehmen sogar mehrere Plattformen verschiedener Hersteller existieren, beispielsweise nach einer Fusion, und Daten zwischen diesen Umgebungen ausgetauscht werden sollen, wird es sehr schwer dies zu ermöglichen. Für gewöhnlich stellen EAI-Produkte eine Kommunikation zu proprietären Plattformen über Adapter her. Allerdings ist die Konkurrenz zwischen EAI-Händlern so groß, dass sie keine Adapter entwickeln, die eine Kommunikation mit Konkurrenzprodukten ermöglichen würden, und zum Teil noch nicht einmal eine API implementieren. Aber Ausnahmen bestätigen ja bekanntlich die Regel und

so ist es auch hier. Zwar stellen auch diese Händler keine Adapter für Konkurrenzprodukte zur Verfügung, sie bieten jedoch erweiterte APIs und Support für Web Services an. Unter Umständen gibt es aber einen Drittanbieter, der einen entsprechenden Adapter für genau die beiden vorhandenen Plattformen entworfen hat, da ihm die Nachfrage groß genug erschien. Sollte dies nicht der Fall sein, gibt es aber eventuell EAI-to-Web Service-Adapter für beide Produkte. Diese wären dann mittels eines „zwischengeschalteten" Service-Hubs dazu in der Lage, einen Datenaustausch zwischen den beiden Plattformen zu ermöglichen. Sollten hingegen noch keine entsprechenden Adapter auf dem Markt existieren, aber es ist wenigstens eine API vorhanden, kann auf die Daten der einen Plattform zugegriffen und diese der anderen zur Verfügung gestellt werden. Gibt es jedoch auf beiden Seiten APIs, können die Plattformen direkt interagieren. Als Ergänzung zu diesen beiden Fällen, könnten Orchestrationsschnittstellen integriert werden, die dann mit den externen Ressourcen korrelieren. Die „einfachste" Lösung ist allerdings, zu einer EAI-Lösung zu greifen, bei der Web Services schon integriert sind, und die alte zu ersetzen.

Ist lediglich ein Altsystem vorhanden und es wird in Betracht gezogen eine EAI-Lösung einzusetzen, ist es am sinnvollsten zu versuchen das Altsystem anzupassen und zwar indem die EAI-Lösung um das vorhandene System herum erstellt wird. Da auch kleinste Änderungen am Altsystem schon zu einer Instabilität führen könnten.

Des Weiteren ist es sinnvoll eine private Serviceregistry zu erstellen. Dies ist ein Verzeichnis, das die WSDL-Definitionen der einzelnen Web Services des eigenen Unternehmens enthält und es ermöglicht, darin gezielt nach bestimmten Web Services zu suchen. Bei EAI-Lösungen mit sofort einsetzbaren Web Services ist eine solche in den meisten Fällen schon enthalten, zum Beispiel in Form eines UDDI-Verzeichnisses. Auf diese Weise kann die Performance zur Laufzeit entscheidend verbessert werden. Falls außerdem gewünscht, kann die Registry abgesichert und dann auch externen Benutzern zugänglich gemacht werden.

6. Integration von Web Service-Sicherheit

Ebenso wie in vielen anderen Bereichen spielt das Thema Sicherheit auch bei Web Services eine große Rolle. Es gibt eine Reihe von Sicherheitsspezifikationen, die speziell für Web Services entwickelt wurden. Diese sollten unbedingt implementiert werden, da ansonsten kaum jemand den Service benutzen wird, wenn nicht sichergestellt ist, dass seine Daten vertraulich behandelt werden. Deshalb ist es wichtig immer auf dem neusten Stand zu sein und diesen auch im Unternehmen einzusetzen. Damit einhergehend sollte ein standardisiertes Sicherheitsmodell erstellt werden, das unterschiedlichen Benutzergruppen verschiedene Zugriffsrechte einräumt. Diese wiederum sollten nach Möglichkeit konform zu

den Sicherheitsgrundsätzen im Unternehmen sein. Um die Komplexität, die durch die Implementation der Sicherheitsaspekte entsteht, zu verringern, können sie auf eine eigene Serviceschicht ausgelagert werden. Auf diese Weise wird eine zentrale Verwaltung ermöglicht und die Kontrolle vereinfacht. Allerdings muss der Aspekt der Skalierbarkeit ausreichend getestet werden bevor diese Möglichkeit zum Einsatz kommt, da es ansonsten durch die Abhängigkeit zahlreicher Serviceoperationen von dieser Schicht zu Performanceengpässen kommen kann. Die Beeinflussung der Performance ist einer der am häufigsten unterschätzten Punkte. Eine effiziente Variante ist ein Sicherheitsmodell mit single Sign-on. Dafür muss sich der Benutzer lediglich ein einziges Mal beim Server anmelden und kann dann sämtliche, ihm zur Verfügung stehenden Services nutzen. Der Vorteil liegt hier in der zentralen Verwaltung der Benutzerdaten.

Wird eine Auftragsdatenverarbeitung in Form der Nutzung des Services eines Drittanbieters in Betracht gezogen, ist es erforderlich diesen mit Hilfe eines Prototyps auf die Unterstützung der Sicherheitsanforderungen, die von ihm erwartet werden, zu testen. Denn wenn sensible Daten erst einmal gesendet wurden, hat das Unternehmen keine Kontrolle mehr über diese Daten und ist darauf angewiesen, dass sich der Service des Providers nicht zu einer Gefährdung der Sicherheit entwickelt. Um auch eine derartige potentielle Gefahr auszuschließen, sollte mit dem Provider ein Vertrag geschlossen werden, der den Datenschutz gewährleistet. Doch natürlich ist nicht nur bei Services anderer Anbieter Vorsicht geboten. Beispielsweise sollten die Beschreibungen der eigenen Services, wie etwa WSDL-Beschreibungen oder UDDI-Registries, nicht zu ausführlich sein, da anderenfalls Rückschlüsse auf interne Prozesse und mögliche Schwachstellen gezogen werden können.

Sollen Anwendungen von Altsystemen neuen Benutzern zur Verfügung gestellt werden, muss auch an dieser Stelle zunächst überprüft werden, ob das System einem entsprechenden Aufkommen stand hält oder möglicherweise Sicherheitsmängel aufweist. Dies kann bei Altsystemen häufiger der Fall sein, da nicht immer soviel Wert auf ein ausgefeiltes Exception-Handling gelegt wurde wie heutzutage und das System demzufolge unter Umständen nicht für derartig viele Benutzer ausgelegt ist. Können Sicherheitsmängel des Altsystems nicht behoben werden, ist es sinnvoller dieses zugunsten eines neuen besseren Systems aufzugeben.

Ein weiterer wichtiger Aspekt der Sicherheit ist die granulare Sicherheit von Nachrichteninhalten. Dabei handelt es sich um eine Möglichkeit, bei der nur die wesentlichen Teile einer Nachricht verschlüsselt werden, zum Beispiel mittels XML-Encryption. Ein positiver Nebeneffekt dieser Lösung ist die Reduzierung des Performanceoverheads. Ergänzend dazu kann mit Hilfe von XML-Signature eine Signatur hinzugefügt werden, die das entsprechende Dokument verifiziert. Mittlerweile können aber auch SOAP-Nachrichten ein Sicherheitsrisiko darstellen. Denn auch bei diesen ist es möglich, Viren an die Datei

anzuhängen. Eine denkbare Lösung dieses Problems wäre, dass der empfangende Web Service oder sogar schon der Vermittler die Nachricht zunächst entschlüsselt und erst danach einen Virusscan ausführt. Gänzlich verhindern lässt sich aber eine eventuelle Bedrohung natürlich auch dadurch nicht.

7. Abschließende Betrachtung

Es gibt zahlreiche verschiedene Anwendungssysteme und Möglichkeiten diese einzusetzen. Jedes einzelne hat natürlich seine ganz eigenen Vor- und Nachteile. Doch welche Lösung nun tatsächlich am besten für das jeweilige Unternehmen geeignet ist, hängt letztendlich von den Gegebenheiten im Unternehmen selbst ab. Durch ein eventuell schon vorhandenes Altsystem oder eine EAI-Lösung kann die Auswahl der Möglichkeiten bereits eingeschränkt werden, da nicht jedes System kann beliebig mit jedem anderen kombiniert beziehungsweise dadurch ersetzt werden. Deshalb ist es wichtig, sich zuerst einen Überblick über die Möglichkeiten zu verschaffen, die für das Unternehmen aufgrund der Gegebenheiten überhaupt in Frage kommen. Des Weiteren sollten die Vorstellungen, in welcher Form beziehungsweise welchem Ausmaß der Web Service eingesetzt werden soll, klar spezifiziert werden. Denn davon hängt schlussendlich ab, welche der verbliebenen Möglichkeiten den Vorstellungen und Anforderungen am besten entspricht und schließlich integriert werden sollte.

Dabei bleibt festzuhalten, dass es zurzeit noch keine ultimative Lösung gibt, die zu jedem bereits vorhandenem System kompatibel ist und zudem ohne großen Aufwand implementiert werden kann. Bis eine derartige Lösung gefunden wird, die für alle Unternehmen gleichermaßen von Vorteil ist, wird es wahrscheinlich auch noch eine ganze Weile dauern, da dies eine sehr komplexe Aufgabe ist. Das System, was es dabei zu entwickeln gilt, muss nicht nur mit sämtlichen anderen existierenden Systemen kompatibel sein, damit die Daten vom alten auf das neue System übertragen werden können, sondern auch mit den unterschiedlichen Anwendungen, die auf dem alten System laufen. Darin liegt aller Wahrscheinlichkeit nach auch das größere Problem, denn während die verschiedenen Arten von Systemen bekannt sind, können die implementierten Anwendungen sowohl proprietärer Natur sein als auch im Unternehmen selbst für den internen Gebrauch entwickelt worden sein. Aufgrund dieser Tatsache bleibt abzuwarten, ob je eine Möglichkeit gefunden wird, die dazu in der Lage ist, diese ganzen „Vorgaben“ zu unterstützen.

QUELLENVERZEICHNIS

Albers, Sönke; Clement, Michel; Peters, Kay; Skiera, Bernd: eCommerce – Einstieg, Strategie und Umsetzung im Unternehmen., 3. Auflage, Frankfurt am Main, 2001, S.151-164, S.179-190

Alonso, Gustavo; Casati, Fabio; Kuno, Harumi; Hachiraju, Vijay: Web Services – Concepts, Architectures and Applications, 1. Auflage, Berlin, 2004, S.151-185

Berres, Anita; Bullinger, Hans-Jörg: E-Business – Handbuch für Entscheider - Praxiserfahrungen, Strategien – Handlungsempfehlungen, 2. Auflage, Heidelberg, 2002, S. 51-100, S.277-295

Biethahn, Prof. Dr. Jörg; Nomikos, Dipl.-Kff. Marina: Ganzheitliches E-Business, München, 2002, S.60-68

Birman, Kenneth P.: Reliable Distributed Systems – Technologies, Web Services, and Applications, New York, 2005, S.193-203

Daum, Berthold; Scheller, Markus: Electronic Business – Methoden, Werkzeuge, Techniken und Systeme für den Unternehmenserfolg im Internet, München, 2000, S.163-177

Erl, Thomas: Service-Oriented Architecture – A Field Guide to Integrating XML and Web Services, 5. Auflage, Upper Saddle River, 2004

Hammerschall, Ulrike: Verteilte Systeme und Anwendungen – Architekturkonzepte, Standards und Middleware-Technologien, München, 2005, S.16-30, S.109-125

Hein, Manfred; Zeller, Henner: Java Web Services – Entwicklung plattformübergreifender Dienste mit J2EE, XML und SOAP, München, 2003, S.164-171, S.220-227

Jablonski, Stefan; Petrov, Ilia; Meiler, Christian; Mayer, Udo: Guide to Web Application and Platform Architectures, Berlin, 2004, S.121-147, S.171-179

Linthicum, David S.: Next Generation Application Integration – From Simple Information to Web Services, 2. Auflage, Boston, 2003, S.369-392

Schubert, Petra; Selz, Dorian; Haertsch, Patrick: Digital erfolgreich – Fallstudien zu strategischen E-Business-Konzepten, 2. Auflage, Berlin, 2003, S.13-21, S. 270f.