Reihe Informatik
TR-2006-014

Implementierung von skalierbaren, hochperformanten Web Services durch deklarative Nachrichtenverarbeitung

A. Böhm C-C. Kanne G. Moerkotte

Universität Mannheim
B6, 29
68131 Mannheim
alex|cc|moer@db.informatik.uni-mannheim.de

Implementierung von skalierbaren, hochperformanten Web Services durch deklarative Nachrichtenverarbeitung

Alexander Böhm Carl-Christian Kanne Guido Moerkotte
Universität Mannheim, Germany
alex|*cc*|*moer*@db.informatik.uni-mannheim.de

11. Juli 2006

Zusammenfassung

Der zunehmende Einsatz von Web Services für erfolgsentscheidende Unternehmensanwendungen erfordert schnelle und flexible Anwendungsentwicklung, sowie hochperformante und zuverlässige Ausführungsumgebungen. Existierende Applikationsserver erfüllen diese Anforderungen nur partiell. Wir stellen eine deklarative Regelsprache zur Definition von Geschäftsprozessen und ein zugehöriges Systemdesign auf Basis von Nachrichtenwarteschlangen und einem nativen XML-Datenbanksystem vor.

1 Einleitung

Durch fortschreitende Standardisierung und Akzeptanz gewinnen Web Services zunehmende Bedeutung für die Realisierung von verteilten, unternehmensübergreifenden Geschäftsprozessen. Ein Beispiel hierfür ist der Informationsaustausch zwischen verschiedenen Partnern in einer Supply Chain. Der Einsatz dieser flexiblen, service-orientierten Technologie zur Implementierung von erfolgsentscheidenden Anwendungen erfordert eine zuverlässige, performante und skalierbare Infrastruktur. Darüber hinaus ist die schnelle Erweiterung und Anpassung von Applikationen unverzichtbar, um dynamisch auf sich ändernde Marktanforderungen reagieren zu können.

Die heute verfügbaren Web Service Lösungen ([2] gibt einen Überblick) basieren größtenteils auf existierenden Middleware-Plattformen, die um entsprechende Web Service Adapter erweitert werden. Dieser Ansatz erlaubt die schnelle Integration der benötigten Schnittstellen in bestehende Infrastruktur, erschwert jedoch die schnelle Implementierung von effizienten Web Services:

- Die Verteilung der unterschiedlichen Teilaufgaben wie Applikationslogik, Nachrichtenaustausch, Transaktionskontrolle und Persistenz auf verschiedene Anwendungssysteme mit unterschiedlicher Datenrepräsentation (beispielsweise Geschäftsobjekte, XML-Nachrichten und Relationen) erfordert aufwändige Konvertierungsoperationen während der Datenverarbeitung. Dieser sog. *Impedance Mismatch* verringert die Performanz des angebotenen Dienstes. Darüber hinaus hat die starke Segmentierung negativen Einfluß auf die Wartbarkeit, Stabilität und Flexibilität des Gesamtsystems.
- Die Implementierung der Geschäftslogik im Rahmen existierender Applikationsserver erfolgt weitgehend durch objektorientierte Programmiersprachen, wie beispielsweise Java oder C#. Im Gegensatz zu deklarativen Sprachen erlauben diese durch ihren imperativen Charakter nur begrenzt automatische Optimierungen und erschweren dadurch die schnelle Entwicklung von performanten Anwendungen mit hohem Durchsatz.

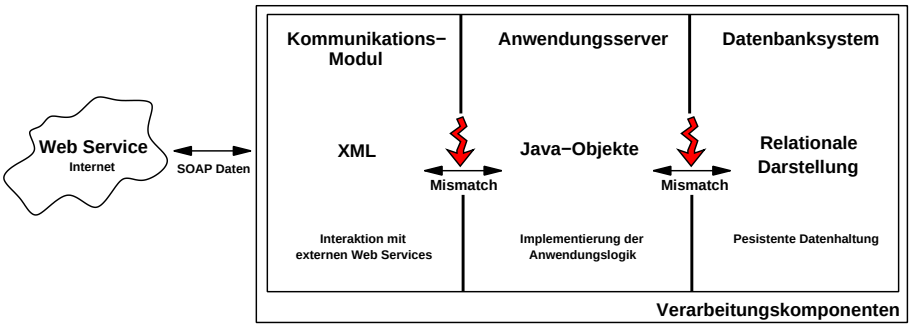

Abbildung 1: Segmentierung und Impedance Mismatch in den heute verfügbaren Lösungen

Ziel unserer Arbeit ist die Entwicklung eines skalierbaren, hochperformanten Web Service Verarbeitungssystems, das auf Basis einer deklarativen Spezifikation der Geschäftslogik eine möglichst effiziente Auswertung ermöglicht.

Das folgende Kapitel 2 gibt einen Überblick über die Anforderungen an ein solches Verarbeitungssystem. Kapitel 3 beschreibt die einzelnen Teilkomponenten und die ihnen zugeordneten Aufgaben. Syntax und Semantik der von uns zur Abbildung der Anwendungslogik vorgeschlagenen, deklarativen Regelsprache sowie zugehörige Beispiele finden sich in Kapitel 4. Kapitel 5 gibt einen Überblick über verwandte Arbeiten, Kapitel 6 fasst die Arbeit zusammen und gibt einen Ausblick auf künftige Forschungsvorhaben.

2 Anforderungen

Das Einsatzgebiet eines Web Service Verarbeitungssystems liegt im Bereich der unternehmensübergreifenden Geschäftsprozesse, die teilweise erfolgsentscheidende Anwendungen darstellen. Ein wichtiges Kriterium für den erfolgreichen Einsatz eines solchen Systems ist die effiziente Implementierung neuer Geschäftsanwendungen sowie die Anpassung existierender Applikationen an sich ändernde Marktanforderungen. Weitere Schlüsselkriterien eines Web Service Verarbeitungssystems sind eine möglichst hohe Laufzeiteffizienz sowie die verlässliche und persistente Datenverarbeitung, auch im Fehlerfall. Das nachfolgende Kapitel gibt einen Überblick über die detaillierten Anforderungen, die sich für die eingesetzte Nachrichtenverarbeitungssprache und das zugehörige Verarbeitungssystem ergeben.

2.1 Verarbeitungssystem

2.1.1 Direkte Verarbeitung und Speicherung von XML

Eines der Kernprobleme der heute zur Implementierung von Web Service Anwendungen verwendeten Systeme ist deren evolutionäre Struktur. Die Web Service Funktionalität wird hierbei in der Regel durch die Erweiterung von bereits eingesetzten Systemen mit Hilfe eines entsprechenden Adapters realisiert. Dieser stellt die benötigten Protokollanbindungen wie das für die Nachrichtenübertragung verwendete, XML-basierte SOAP-Protokoll [18] sowie zugehörige Dienste zur Verfügung, beispielsweise für bereits im Unternehmen existierende Middleware-Lösungen. Die Ausführung der durch den Web Service implementierten Anwendungslogik obliegt dann einem Anwendungsserver, der, basierend auf einer üblicherweise in einer objektorientierten, imperativen Hochsprache wie Java oder C# verfassten Anwendungsspezifikation, die einzelnen Geschäftsprozesse verwaltet. Die persistente Datenhaltung erfolgt in einem solchen System in der Regel durch einen relationalen Datenbankserver.

Die einzelnen Verarbeitungsschritte, die durch eine eingehende Nachricht ausgelöst werden, sind in Abbildung 1 zusammengefasst. Ausgehend vom Eingang einer Benachrichtigung von einem aufrufenden, externen Web Service im Kommunikationsmodul muss die entsprechende Applikationslogik ausgeführt werden. Hierbei verwenden das XML-basierte Kommunikationsmodul und der objektbasierte Anwendungsserver unterschiedliche Datenrepräsentationen (*Impedance Mismatch*). Vor der eigentlichen Datenverarbeitung wird deshalb eine entsprechende Konvertierungsoperation in das durch den Anwendungsserver verwendetet Format benötigt. Ist das Kommunikationsmodul nicht direkt in den Anwendungsserver integriert erhöht sich der Verarbeitungsaufwand eventuell noch weiter (beispielsweise durch zusätzliche Netzwerkkommunikation zum Anwendungsserver). Im Laufe der Datenverarbeitung tritt ein weiterer Impedance Mismatch bei der Speicherung oder dem Zugriff auf persistente Daten auf. Hierbei wird die Konvertierung zwischen den Objekten des Anwendungsservers und der relationalen Repräsentation des Datenbankssystems erforderlich. Üblicherweise verursacht die Anwendungsschnittstelle des Datenbankservers (z.B. JDBC) noch weiteren Verarbeitungsaufwand.

Um die oben genannten Probleme im Rahmen eines Web Service Verarbeitungssystems zu vermeiden, muss ein solches System die native Verarbeitung und Speicherung des zur Kommunikation verwendeten XML-Datenformats unterstützen und durchgängig auf dieser Datenrepräsentation arbeiten. Hierdurch kann ein Impedance Mismatch und der daraus resultierende, zusätzliche Verarbeitungsaufwand vermieden werden.

2.1.2 Persistenz und zuverlässige Speicherung

Die im Rahmen von verteilten Geschäftsprozessen ausgetauschten Daten stellen wertvolle, oft erfolgskritische Informationen beispielsweise über Bestellungen, Aufträge und Verträge dar. Die persistente und zuverlässige Speicherung dieser Daten ist eine der entscheidenden Anforderungen an ein entsprechendes Verarbeitungssystem. Die persistente Datenhaltung ist hierbei sowohl für den eigentlichen Geschäftsprozess als auch für spätere Analysen von Bedeutung und wird auch benötigt, um gegebenenfalls rechtlich bedingten Aufbewahrungsvorschriften Rechnung zu tragen. Insbesondere muss sichergestellt werden, dass die vorliegende Daten auch im Fall eines Anwendungs- oder Systemfehlers wieder hergestellt werden können (*recovery*). Die genannten Anforderungen überschneiden sich hierbei deutlich mit denen, die üblicherweise an ein Datenbanksystem gestellt werden.

2.1.3 Skalierbarkeit und hohe Performanz

Die Kommunikation zwischen den Teilnehmern eines Web Service-basierten Geschäftsprozesses erfolgt üblicherweise durch den asynchronen Austausch von XML-Nachrichten, wie beispielsweise einer Bestellung und der zugehörigen Versandbestätigung. Ein Geschäftsvorfall besteht hierbei üblicherweise aus einer Reihe von potentiell zeitaufwändigen Aktionen wie die Beschaffung einer benötigten Komponente von einem Zulieferer, die Auftragsgenehmigung durch einen Verantwortlichen oder die Auslieferung der Bestellung durch einen Logistikdienstleister. Als direkte Konsequenz sind solche Geschäftstransaktionen von Natur aus langlebig und können durchaus mehrere Tage oder noch länger dauern.

Das exklusive Sperren von Systemressourcen über einen derart langen Zeitraum schränkt den realisierbaren Grad an Nebenläufigkeit stark ein und ist im Rahmen eines skalierbaren und performanten Verarbeitungssystems kein gangbarer Weg. Statt dessen muss die nebenläufige Ausführung einer Vielzahl von unterschiedlichen Anwendungen und Anwendungsinstanzen möglich sein. Ein wichtiger Bestandteil hierfür ist die effiziente Unterstützung des asynchronen, nachrichten-basierten Verarbeitungsmusters durch entsprechende, native Datenstrukturen, die den nebenläufigen und synchronisierten Zugriff auf die im System vorliegenden Nachrichten gestatten.

2.2 Nachrichtensprache

2.2.1 Direkte XML-Unterstützung

Wie bereits in Abschnitt 2.1.1 beschrieben ist eines der Hauptprobleme der meisten, aktuell eingesetzten Web Service Lösungen die unterschiedliche Datenrepräsentation zwischen den einzelnen Verarbeitungskomponenten und der daraus resultierende Konvertierungsaufwand (vgl. Abbildung 1). Um diesen Zusatzaufwand vermeiden zu können muss nicht nur das Web Service Verarbeitungssystem selbst direkt auf der XML-Repräsentation der ausgetauschten Daten arbeiten, auch die zugehörige Anwendungssprache muss XML als primären Datentyp unterstützen. Dies schließt insbesondere entsprechende Anfrage- und Verarbeitungsmöglichkeiten mit ein, mit denen sowohl der Inhalt als auch die Struktur eingehender XML-Fragmente analysiert werden kann und die auch das Erzeugen neuer Dokumente gestatten. Die im Rahmen heutiger Applikationsserver eingesetzten Hochsprachen verfügen in der Regel nicht oder nur partiell über derartige XML-Anbindungen.

2.2.2 Deklarativität

Neben der nur eingeschränkten Unterstützung des XML-Datenformats ist eines der Hauptprobleme beim Einsatz von imperativen Hochsprachen zur Programmierung von Web Services deren eingeschränktes Optimierungspotential. Bedingt durch die imperative Struktur sind die Abfolge und die im Rahmen der Anwendung auszuführenden Operationen explizit vom Programmierer vorgegeben und schränken hierdurch den dem Ausführungssystem verbleibenden Freiheitsgrad für Optimierungen stark ein. Als direkte Konsequenz müssen die für eine hohe Laufzeiteffizienz erforderlichen Optimierungen oftmals manuell beim Schreiben des Programmcodes berücksichtigt werden und erhöhen dadurch den Entwicklungsaufwand.

Unserer Ansicht nach stellt eine deklarative Anwendungssprache im Rahmen eines Web Service Verarbeitungssystems hier eine vielversprechende Alternative dar. Bei diesem Ansatz wird bei der Implementierung der Anwendungslogik festgelegt wie sich das System beim Eintreten eines Ereignisses zu verhalten hat (beispielsweise durch das Senden einer entsprechenden Antwort auf eine eingehende Nachricht). Die hierbei auszuführenden Teilschritte sind jedoch nicht explizit durch den Entwickler vorgegeben. Hierdurch ergibt sich ein großer Freiheitsgrad für Optimierungen und zur Durchführung von einzelnen, alternativen Ausführungsschritten, solange garantiert werden kann, dass sich das System wie beschrieben verhält. Als direkte Konsequenz können effiziente Optimierungen durch das Verarbeitungssystem durchgeführt werden und müssen nicht mehr manuell im Rahmen der Implementierung der Applikationslogik spezifiziert werden.

2.2.3 Direkte Unterstützung charakteristischer Verarbeitungsmuster

Ein Charakteristikum von Web Service Anwendungen ist die Kommunikation über den Austausch vom XML-Nachrichten. Dies führt üblicherweise zu typischen, wiederkehrenden Verarbeitungsmustern bei der Spezifikation der Anwendungslogik: Basierend auf dem Eingang einer Nachricht und deren Analyse werden zugehörige Transformationsschritte durchgeführt, neue Nachrichten erzeugt und diese an andere Systemkomponenten versandt. Jeder der genannten Schritte muss hierbei durch entsprechenden Programmcode abgebildet werden. Im Rahmen der Programmierschnittstellen (*API*s) der heutigen Messaging Systeme (wie beispielsweise [15]) werden hierfür oftmals erhebliche Mengen an Zusatzcode benötigt. Die Nachrichtensprache eines Web Service Verarbeitungssystems muss die oben genannten, für einen Web Service typischen Verarbeitungsmuster direkt und effizient unterstützen um die Anwendungsentwicklung zu vereinfachen und zu beschleunigen.

3 Systemarchitektur

Dieses Kapitel gibt einen Überblick über die von uns vorgeschlagene Architektur eines Web Service Verarbeitungssystems. Das Gesamtsystem lässt sich hierbei in vier logische Teilkomponenten gliedern (vgl. Abbildung 2).

Der Regelkompiler verarbeitet die zur Abbildung des Geschäftsprozesses erstellte, regelbasierte Anwendungslogik und speichert diese in einem Regelverzeichnis. Die Anwendungsausführung steuert das Systemverhalten zur Laufzeit und übernimmt die Auswertung der Applikationslogik sowie deren Abbildung auf verschiedene Systemprozesse und Transaktionen. Die Kommunikationskomponente stellt die Schnittstelle zu externen Web Services dar und implementiert die benötigten Übertragungsprotokolle und Erweiterungen. Die effiziente und zuverlässige Speicherung der Anwendungsdaten übernimmt die Persistenzkomponente. Sie verfügt über spezielle Datenstrukturen für die Speicherung und den Zugriff auf XML-Nachrichten.

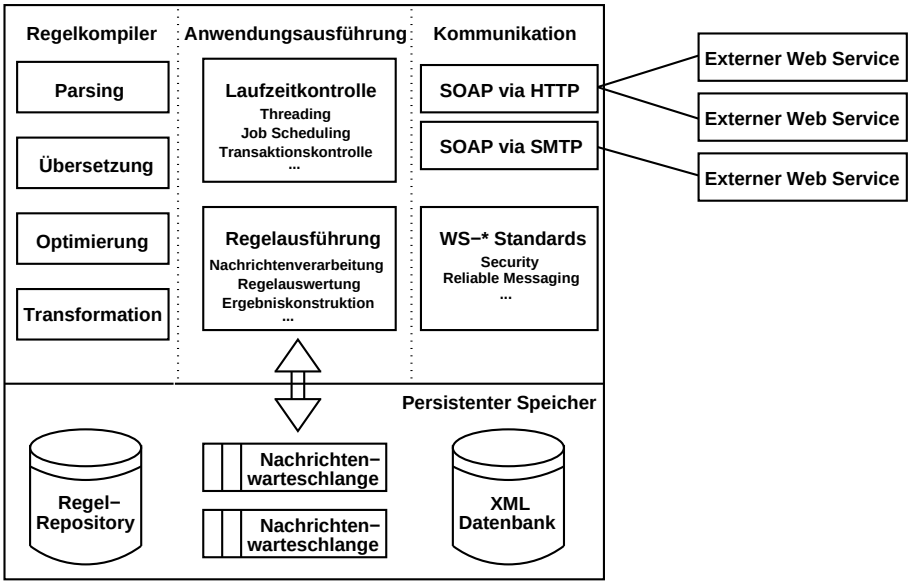

Abbildung 2: Kernkomponenten eines Web Service-Ausführungssystems

3.1 Datenhaltung

Wie bereits in Abschnitt 2.1.2 beschrieben decken sich die Anforderungen an die Datenhaltungskomponente des Verarbeitungssystems zu einem großen Teil mit denen eines Datenbanksystems. Es bietet sich konsequenterweise an, ein solches System als Grundlage für die Datenhaltungskomponente in Betracht zu ziehen. Zusammen mit den Anforderungen nach der direkten Unterstützung des XML-Datenformates sowie effizienter Speicherstrukturen zur Nachrichtenverwaltung ergeben sich zwei grundlegende Alternativen.

Eine mögliche Ausgangsbasis ist die Verwendung eines relationalen Datenbanksystems mit nativer Unterstützung für Nachrichtenwarteschlangen. Beispiele für solche Systeme sind Oracle Streams Advanced Queuing [16] oder Microsoft SQL Server Service Broker [23]. Sie erlauben die direkte und effiziente Ausführung der benötigten Nachrichtenverarbeitungsoperationen über entsprechende Datenbankschnittstellen und Speicherstrukturen, verfügen jedoch nur über eine partielle Unterstützung des XML-Formates und der zur XML-Verarbeitung benötigten Operationen.

Eine Alternative zu relationalen Systemen stellen native XML-Datenbanksysteme dar. Diese Systeme sind speziell zur Verarbeitung von XML-Daten entworfen und unterstützen üblicherweise XML-spezifische Anfragesprachen wie XPath [6] und XQuery [10]. Eines der ersten kommerziellen Datenbanksysteme mit nativer XML-Unterstützung ist IBM DB2 [8]. Zum gegenwärtigen Zeitpunkt bietet jedoch keines der verfügbaren XML-Systeme eine direkte Unterstützung für Nachrichtenwarteschlangen.

Da das vorgeschlagene Web Service Verarbeitungssystem exklusiv auf der XML-Repräsentation der Anwendungsdaten arbeitet und insbesondere keine relationale Datenhaltung verwendet, dient ein natives XML-Datenbanksystem (XDS) als Ausgangsbasis für die Datenhaltungskomponente. Wie bereits beschrieben muss ein solches System jedoch noch um entsprechende Datenstrukturen für Warteschlangen erweitert werden um eine effiziente und zuverlässige Nachrichtenverwaltung zu ermöglichen. Als Grundlage für unsere Implementierung dient Natix [13], ein natives XML-Datenbanksystem, das als Forschungsprojekt von der Datenbankgruppe der Universität Mannheim entwickelt wird. Die hochgradig modulare Struktur des Natix-Sytems erlaubt die Integration neuer Datentypen - wie die benötigten transaktionalen Warteschlangen - ohne zusätzliche Änderungen an den restlichen Teilsystemen. Zu den wichtigsten Aspekten der implementierten Datenstruktur gehören Ordnungserhaltung, transaktionale Semantik sowie eine möglichst hohe Nebenläufigkeit.

3.1.1 Ordnungserhaltung

Zwischen den in einem Web Service Verarbeitungssystem vorliegenden XML-Nachrichten bestehen in der Regel direkte oder indirekte Anordnungsbeziehungen, die sich beispielsweise durch den Zeitpunkt des Nachrichteneingangs oder die logische Abfolge zweier Nachrichten innerhalb eines Geschäftsprozesses ergeben (etwa erfolgt die Bestätigung eines Auftrags in der Regel erst nach dessen Erteilung). Diese Anordungsbeziehungen müssen im Rahmen des Verarbeitungssystems berücksichtigt werden, nicht zuletzt um einen gewissen Grad an Fairness bei der Verarbeitungsreihenfolge eingehender Nachrichten sicherzustellen. Die in einem XDS verwendeten, ungeordneten Dokumentkollektionen berücksichtigen diese Ordnungsbeziehnungen in der Regel nicht. Die durch unsere Implementierung hinzugefügten Erweiterungen verwalten daher einer Menge von XML-Fragmenten als geordnete Sequenz mit genau einem Anfang- und Endelement und erhalten die Ordnungsbeziehungen zwischen den einzelnen Nachrichten.

3.1.2 Transaktionale Semantik

Um die zuverlässige Verarbeitung der in den Warteschlangen abgelegten Daten sicherzustellen, müssen die zugehörigen Verarbeitungsoperationen durch den Transaktionsmanager unterstützt werden, um Datenisolation und Wiederanlaufbarkeit garantieren zu können. Die Verarbeitung einer Nachricht erfolgt im Rahmen des Verarbeitungssystems durch deren Entfernen aus der zugehörigen Warteschlange (*dequeue*) und führt in der Regel dazu, dass neue Nachrichten in andere Warteschlangen eingefügt werden (*enqueue*). Hierbei muss sichergestellt werden, dass eine Nachricht erst nach deren erfolgreicher Verarbeitung aus dem System entfernt werden kann, sowie dass die zu einer Transaktion gehörenden Warteschlangenoperationen atomar ausgeführt werden.

3.1.3 Nebenläufigkeit

Die strenge Beachtung der Ordnungsbeziehungen zwischen den einzelnen Nachrichten in Verbindung mit einer transaktionalen Zugriffsemantik schränkt die Möglichkeiten zur nebenläufigen Verarbeitung stark ein. Systeme, die strikt diesen Ansatz verfolgen (beispielsweise [23]) erlauben die Verarbeitung des nächsten in einer Warteschlange vorliegenden Datenfragments erst nachdem die Transaktion, die das vorangehende Dokument bearbeitet

hat, erfolgreich abgeschlossen wurde. Dies erlaubt es, die Ordnung der Nachrichten selbst im Fall von Transaktionsabbrüchen zu erhalten.

Unser System verfolgt hier einen anderen Ansatz, der von Bernstein et. al [7] vorgeschlagen wurde und auch in kommerziellen Systemen Verwendung findet [15], und erlaubt den parallelen Zugriff auf mehrere Einträge der selben Warteschlange. Hierdurch kann eine höhere Verarbeitungseffizienz erreicht werden, die strikte Anordnung der Nachrichten jedoch im Fall eines Transaktionsabbruchs eventuell nicht mehr gewährleistet werden. In einem solchen Fall führt das System eine logische Undo-Operation aus, bei der die von einer abgebrochenen Transaktion bearbeitete Nachricht am Anfang der betreffenden Warteschlange eingereiht wird. Hierdurch kann sichergestellt werden, dass das entsprechende XML-Fragment als nächster Eintrag verarbeitet wird, wenn auch nicht in der ursprünglich vorliegenden Reihenfolge. Im Rahmen eines Web Service Verarbeitungssystems erscheint uns diese Einschränkung als tolerabel um eine höhere Nebenläufigkeit zu erzielen.

3.2 Regelkompiler

Der Aufgabenbereich des Regelkompilers ist die Verwaltung und Verarbeitung der zur Abbildung des Geschäftsprozesses verwendeten Anwendungslogik. Hierzu werden mehrere Schritte durchgeführt. Zuerst wird die entsprechende Anwendungsspezifikation auf syntaktische Korrektheit analysiert. Anschließend erfolgt die Übersetzung in eine Zwischenrepräsentation, auf der im Anschluss Optimierungen durchgeführt werden. In einem letzten Schritt wird der resultierende, optimierte Ausführungsplan in einem entsprechenden Regel-Repository abgelegt, auf das die Anwendungsausführung zur Laufzeit zugreift.

3.3 Kommunikationsschnittstelle

Die Interaktion zwischen dem Web Service Verarbeitungssystem und externen Kommunikationspartnern wird durch die Kommunikationsschnittstelle realisiert. Diese stellt Kommunikationskanäle für die Übertragung von SOAP-Daten über verschiedene Transportprotokolle wie HTTP oder SMTP zur Verfügung.

Im Rahmen eines verteilten Geschäftsprozesses bestehen oft zusätzliche Kommunikationsanforderungen, die über die des reinen Datentransfers hinaus gehen. Examplarisch seinen hierfür die verschlüsselte Übertragung von sensitiven Informationen oder die zuverlässige Nachrichtenübertragung zwischen den einzelnen Anwendungssystemen genannt. Zu diesem Zweck existieren Erweiterungen des SOAP-Protokolls, die durch eine Reihe von ergänzenden Standards wie beispielsweise WS-Security [4] und WS-ReliableMessaging [9] definiert sind. Diese Erweiterungen müssen durch zugehörige Module der Kommunikationsschnittstelle implementiert werden um die entsprechende Funktionalität zur Verfügung zu stellen.

3.4 Anwendungsausführung

Das Abarbeiten der Anwendungslogik erfolgt durch die Anwendungsausführungskomponente. Um den in Abschnitt 2.2 aufgestellten Anforderungen für die Spezifikation der Geschäftslogik gerecht zu werden, verwendet das vorgeschlagene Web Service Verarbeitungssystem eine vollständig deklarative, regelbasierten Nachrichtenverarbeitungssprache. Mit Hilfe dieser Sprache lässt sich beschreiben wie das System auf den Eingang einer neuen Nachricht (beispielsweise von einem externen Web Service) zu reagieren hat. Ereignisse, die nicht auf dem Empfang einer Nachricht basieren, sondern beispielsweise einen Zeitbezug haben, werden durch entsprechende Nachrichten emuliert. Eine umfassende Übersicht über die einzelnen Sprachkomponenten und -konstrukte findet sich in Kapitel 4.

3.4.1 Ausführungsmodell

Das eingesetzte Ausführungsmodell ist eines der zentralen Elemente der Anwendungsausführung in einem Web Service Verarbeitungssystem. Es beschreibt, wie das System zur Laufzeit auf bestimmte Ereignisse - wie eine eingehende Nachricht - zu reagieren hat, wann die zugehörige Anwendungslogik ausgeführt wird und wie auf eventuell neue, durch die Abarbeitung der Anwendungslogik ausgelöste Ereignisse zu reagieren ist.

Das in unserem System verwendete, iterative Verarbeitungsmodell lässt sich als Verarbeitungsschleife veranschaulichen. Jede Iteration greift hierbei auf eine noch nicht verarbeitete Nachricht zu und führt die zugehörige Applikationslogik aus. Die einzelnen Regeln, die zur Spezifikation der Anwendungslogik verwendet werden, werden hierbei parallel abgearbeitet. Durch deren Ausführung werden üblicherweise neue Nachrichten erzeugt, die dann in einer darauffolgenden Iteration bearbeitet werden.

3.4.2 Abbildung auf Transaktionen

Ein weiteres, wichtiges Element der Anwendungsausführung ist die Abbildung der regelbasierten Anwendungslogik auf zugehörige Datenbanktransaktionen. Diese Abbildungsfunktion hat hierbei große Auswirkungen auf die Nebenläufigkeit des Systems und damit auch auf dessen Laufzeitverhalten und -effizienz.

Werden möglichst langlebige Transaktionen verwendet, so erstreckt sich eine Transaktion über mehrere, in der Regel warteschlangenübergreifende Anwendungsschritte. Durch die Vielzahl der von einer Transaktion gehaltenen Sperren sinkt die mögliche Nebenläufigkeit und damit die Systemperformanz. Erfolgt die Abarbeitung der Anwendungslogik im Rahmen möglichst kurzer Transaktionen, etwa einer Transaktion pro Anwendungsregel, so werden eventuell partielle Ergebnisse für andere Regeln, die auf Basis der selben Nachricht aufgeführt werden, sichtbar. Ohne entsprechende Zusatzmaßnahmen wie etwa eine eindeutige Regelpriorisierung sind die durch ein solches Ausführungssystem erzeugten Ergebnisse nicht mehr konfluent.

Die Abbildung der Anwendungslogik auf Transaktionen stellt ein noch detaillierter zu untersuchendes Teilproblem dar. Der im Rahmen unseres Systems verwendete Ansatz ist ein Kompromis zwischen den beiden obigen Strategien. Hierbei werden alle zu einem Ereignis (einer eingehenden Nachricht) gehörigen Anwendungsregeln innerhalb der selben Transaktion ausgeführt. Weiterhin werden die auszuführenden Regeln evaluiert bevor die resultierenden Aktionen durchgeführt werden. Hierdurch werden ungewollte Wechselwirkungen während der Regelausführung vermieden, da die logisch zusammengehörige Operationen der Anwendungslogik innerhalb der selben Transaktion verarbeitet werden. Eventuell aus der Regelausführung resultierende, neue Nachrichten werden bedingt durch das iterative Verarbeitungsmodell anschließend im Rahmen einer anderen Transaktion weiterverarbeitet, es entstehen also insbesondere keine rekursiven Aufrufe.

4 Deklarative Nachrichtensprache

Die Spezifikation der im Rahmen des Verarbeitungssystems auszuführenden Anwendungslogik erfolgt mit Hilfe einer Web Service-spezifischen, deklarativen Nachrichtenverarbeitungsprache. Kernbestandteile hiervon sind die Queue Definition Language (QDL), die die der Applikation zugrundeliegende Warteschlangen-Infrastruktur beschreibt, sowie die Queue Rule Language (QRL), die zur Spezifikation der zur Laufzeit abzuarbeitenden Anwendungslogik verwendet wird.

4.1 Queue Definition Language (QDL)

Die Queue Definition Language (QDL) dient der Spezifikation der Infrastruktur aus verschiedenen Warteschlangen, auf denen die mit Hilfe der QRL definierte Applikationslogik

zur Laufzeit ausgeführt wird. Neben den bereits beschriebenen, transaktionalen und persistenten Warteschlangen können mit Hilfe der QDL auch andere, spezielle Warteschlangeninstanzen definiert werden. Diese dienen beispielsweise dazu, die Performanz der Anwendungsausführung zu erhöhen oder auch um spezielle Ereignisse und Aktionen, die nicht nativ von der QRL unterstützt werden, durch diese zugreifbar zu machen.

- Transiente Warteschlangen
  Während im Allgemeinen die in einem verteilten Geschäftsprozess zwischen den Partnern ausgetauschten Daten wichtige Informationen repräsentieren, die zuverlässig und dauerhaft gespeichert werden müssen, gibt es jedoch auch einzelne Anwendungsfälle, in denen die Datenpersistenz nur eine untergeordnete Rolle spielt. Beispiele hierfür sind etwa Produktinformationen oder Nachrichten über Rabattaktionen, die nicht unbedingt den Empfänger erreichen müssen. Für solche nicht-kritischen Daten oder Informationen, die aus anderen Quellen mit geringen Aufwand rekonstruiert werden können, erlaubt das System die Definition von speziellen, transienten Warteschlangen. Der Inhalt dieser Warteschlangeninstanzen kann bei einem eventuellen Systemabsturz zwar verloren gehen, sie weisen dafür jedoch eine höhere Laufzeiteffizienz auf.

- Gateway Queues
  Eine der zentralen Aufgaben eines Web Service Verarbeitungssystems ist der Datenaustausch mit externen Kommunikationsendpunkten, wie etwa anderen Web Services. Zur Kommunikation mit diesen externen Instanzen dienen in unserem System spezielle Warteschlangen, sogenannte *Gateway Queues*. Wird eine Nachricht durch die Applikationslogik in eine Gateway Queue eingefügt, so wird diese mit Hilfe der Kommunikationsschnittstelle an den zugeordneten, externen Dienst weitergeleitet. Dieses Konzept erlaubt eine einheitliche Sicht auf sowohl lokale als auch entfernte Datenstrukturen und Dienste, da beide als Warteschlange durch die QRL adressiert werden können.

- Echo Queues
  Wie bereits beschrieben agiert die mit Hilfe der QRL spezifizierte Anwendungslogik immer reaktiv auf den Eingang einer Nachricht in einer spezifischen Warteschlange. Im Rahmen eines Geschäftsprozesses müssen jedoch auch andere, beispielsweise zeitbasierte Ereignisse berücksichtigt werden können, um etwa einzuhaltende Fristen oder Verfallsdaten zu modellieren. Die Unterstützung von zeitlichen Ereignissen erfolgt durch einen speziellen Warteschlangentyp, die *Echo Queues*. Diese erlauben die Registierung einer zu sendenden Nachricht zu einem gewissen absoluten oder relativen Zeitpunkt. Sie können somit eingesetzt werden, um zeitliche Ereignisse auf den Eingang einer Nachricht abzubilden.

4.2 Queue Rule Language (QRL)

Die Queue Rule Language (QRL) dient zur Abbildung der zur Laufzeit auszuführenden Geschäftslogik auf ein entsprechendes, auf Warteschlangen operierendes Regelwerk. Aus Gründen der Einfachheit ist das Regelwerk ist hierbei statisch, d.h. es kann zur Laufzeit des Systems nicht verändert werden.

Die spezifizierten Regeln haben eine Event-Condition-Action (ECA) Struktur: Tritt ein gewisses Ereignis (event) ein und sind die zugehörigen Bedingungen (conditions) erfüllt, so werden die entsprechenden Aktionen (actions) ausgeführt. Im Folgenden werden die einzelnen Teilkomponenten einer Regel detailliert erläutert. Abbildung 3 gibt einen zusammenfassenden Überblick über die einzelnen Regelteile und die allgemeine Struktur.

on message into [**class**] QueueName
define
    variable **as** XPathExpr
    ...
if XPathExpr
define
    variable **as** XPathExpr
    variable **as** { XQueryExpr }
    ...
do ACTION(XPathExpr,...)
  ...

Abbildung 3: Allgemeine Struktur einer ECA-Regel

4.2.1 Event

Das einzige Ereignis, das im Rahmen der QRL betrachtet wird, ist der Eingang einer Nachricht in einer Warteschlange, oder einer Klasse von Warteschlangen. Wie bereits in Abschnitt 4.1 beschrieben können andere, für die Umsetzung eines Geschäftsprozesses relevante Ereignisse mit Hilfe spezieller Warteschlangentypen emuliert werden, so dass die Existenz nur eines einfachen, zentralen Ereignistyps sich nicht negativ auf die Mächtigkeit des Systems auswirkt.

Neben dem Nachrichteneingang in einer spezifischen Warteschlange kann im Event-Teil einer Regel auch eine Warteschlangenklasse angegeben werden, zu denen eine Reihe von einzelnen Warteschlangeninstanzen gehört, beispielsweise alle Datenstrukturen, die Kundenaufträge enthalten. Hierdurch lässt sich gegebenenfalls die Modellierung der Anwendungslogik vereinfachen, zusätzlich können durch die Warteschlangenklassen trotz des statischen Regelwerks auch Regeln für dynamische, durch eine Aktion zur Laufzeit erzeugte Warteschlangen angegeben werden.

4.2.2 Variablenbindung

Innerhalb jeder Regel existieren zwei Variablendeklarationsblöcke in denen einzelne Variablen an Ausdrücke der XML-Anfragesprachen XPath [6] und XQuery [10] gebunden werden können.

Der erste Deklarationsblock befindet sich vor dem konditionalen Teil (condition) der Regel. Er erlaubt die Spezifikation von Variablen mit Hilfe von XPath. Diese Variablen können sowohl in der Regelbedingung als auch gegebenenfalls im Aktionsteil verwendet werden. Wird eine Nachricht in die der Regel zugehörige Warteschlange oder Warteschlangenklasse eingefügt, so müssen die im ersten Deklarationsblock definierten Variablen und der konditionale Teil für jede der für diese Warteschlange definierten Regeln ausgewertet werden. Die effiziente Auswertung dieses ersten Regelteils ist also für die Performanz des gesamten Verarbeitungssystems von entscheidender Bedeutung. Aus diesem Grund können im ersten Deklarationsblock nur XPath-Variablen verwendet werden, da die XPath-Auswertung deutlich effizienter möglich ist als die der komplexeren XQuery-Ausdrücke.

Der zweite Variablendeklarationsblock findet sich direkt nach dem konditionalen Teil der Regel. Hier können sowohl XPath- als auch XQuery-Ausdrücke zur Variablenbindung und Ergebniskonstruktion verwendet werden.

Neben den üblichen, standardisierten Sprachkonstrukten von XPath und XQuery verwenden wir im Rahmen unser Regelsprache noch benutzerdefinierte Funktionen, mit denen auf die aktuell durch eine Regel verarbeitetet Nachricht (`qs:message()`), deren zugehörige Warteschlange (`qs:queue()`) oder den Inhalt einer andere Warteschlange im System

(`qs:queue(„name“)`) zugegriffen werden kann.

4.2.3 Condition

Im Rahmen des konditionalen Regelteils werden die für die Auswertung der zugehörigen Aktionen zu erfüllenden Bedingungen überprüft. Dies geschieht unter Verwendung eines XPath-Ausdrücks, der gegebenenfalls die oben beschriebenen, im ersten Deklarationsblock gebundenen Variablen mit einbezieht. Nur wenn die spezifizierten Bedingungen erfüllt sind wird der restliche Teil der Regel ausgeführt.

4.2.4 Action

Die Liste der durch eine Regel potentiell ausführbaren Aktionen besteht aus vier grundlegenden Operationen. Dieses stark eingeschränkte Aktions-Set erleichtert einerseits potentiell die Optimierung des Regelwerks, zum anderen reichen diese Aktionen in Verbindung mit den in Abschnitt 4.1 vorgestellten Warteschlangentypen aus um die benötigte Systemfunktionalität zu implementieren. Als Parameter für die Aktionen werden XPath-Ausdrücke verwendet, die auch die in den Variablendeklarationsblöcken definierten Variablen referenzieren können.

Die `enqueue`-Aktion erlaubt das Anfügen einer Nachricht an eine beliebige Warteschlange im System (oder auch das Versenden an einen externen Web Service mit Hilfe einer Gateway Queue). Der Name der Ziel-Warteschlange sowie der Nachrichteninhalt werden hierbei als Parameter übergeben. Mit Hilfe der `delete`-Aktion können Nachrichten explizit aus einer Warteschlange gelöscht werden. Als Parameter kann hierbei eine Menge von Nachrichten übergeben werden. Durch die `createQueue`-Aktion können zur Laufzeit dynamisch neue Warteschlangeninstanzen erzeugt werden. Hierbei werden der eindeutige Warteschlangenname sowie die zugehörige Warteschlangenklasse als Parameter übergeben. Durch die `destroyQueue`-Aktion kann eine dynamisch erzeugte Warteschlange wieder entfernt werden.

4.3 Anwendungsbeispiele

Im folgenden Abschnitt werden einige examplarische Anwendungsfälle für die obige Regelsprache im Rahmen eines fiktiven Einkaufs-Web Services vorgestellt. Die jeweils erzeugten Antwortnachrichten sind aus Gründen der besseren Lesbarkeit deutlich vereinfacht.

Das erste Beispiel in Abbildung 4 zeigt die Reaktion auf eine von einem externen Web Service in der „hotline“-Warteschlange eintreffende Nachricht, hier die Anmeldung eines Benutzers. Erfüllt die Nachricht den konditionalen Teil der Regel indem sie ein entsprechendes „body“- und „action“-Element enthält, wird die zugehörige Aktion ausgeführt. Hierbei wird das Ziel (*targetqueue*) sowie die zu sendende Nachricht (*content*) mit Hilfe von XPath- und XQuery-Ausdrücken konstruiert und durch die `enqueue` Aktion an eine andere Warteschlange versandt. Ist diese Warteschlange vom Typ Gateway Queue so wird die Nachricht an den entsprechenden externen Web Service weitergeleitet.

Abbildung 5 zeigt wie im Rahmen einer Regel auf den Inhalt anderer Warteschlangen zugegriffen werden kann, beispielsweise um einen Lieferauftrag zu erteilen. Trifft hier eine neue Nachricht in der „hotline“-Warteschlange ein und wird der konditionale Teil durch entsprechende Nachrichtenelemente erfüllt sollen hier die in einer seperaten Warteschlange gespeicherten Bestellungen eines Kunden an eine andere Systemkomponente („packaging“) zur weiteren Bearbeitung weitergeleitet werden. Der Zugriff auf die seperate Warteschlange erfolgt hierbei über die `qs:queue`-Funktion, die die zugehörigen Elemente zurückliefert. Das resultierende XML-Dokument (*content*) wird dann mit Hilfe eines XQuery-Ausdrucks erzeugt und kann anschließend über die `enqueue`-Aktion weitergeleitet werden.

```
on message into „hotline“
if //body/action=„logIn“
define
        targetqueue as concat(„reply“,//body/username)
        content as { <text>Welcome</text> }
do ENQUEUE($targetqueue, $content)
```

Abbildung 4: Direkte Reaktion auf eine eingehende SOAP-Nachricht

```
on message into „hotline“
if //body/action=„confirmOrder“
define
        orderq as concat(„orders“,//body/username)
        orderid as //body/orderID
        items as {
                for $entry in qs:queue($orderq)//entry
                where $entry/entryType='orderItem'
                        and $entry/orderID=$orderid
                return <item>$entry</item>}
        content as { <action> assembleOrder
                <items>$items</items> </action>}
do ENQUEUE(„packaging“, $content)
```

Abbildung 5: Zugriff auf andere Warteschlangen im Rahmen einer ECA-Regel

```
on message into „orderProcessing“
define
        process as qs:queue()[//orderID=
        qs:message()/orderID]
if ($process//item/@status=„OK“)
        and ($process//creditRating=„OK“)
define
        targetqueue as concat(„reply“,//body/username)
        contents as {<text>We’ll ship your order...</text>}
do ENQUEUE($targetqueue, $contents)
```

Abbildung 6: Synchronisation des Verarbeitungskontrollflusses im Rahmen einer ECA-Regel

Das Beispiel in Abbildung 6 zeigt die Synchronisation des Verarbeitungsflusses im Rahmen einer ECA-Regel. Ziel dieser Regel ist es, sicherzustellen, dass eine Auftragsbestätigung nur versandt wird, falls sowohl alle bestellten Güter lieferbar sind als auch der Kunde über eine ausreichende Bonität verfügt. Hierfür müssen in der Warteschlange der Auftragsabwicklung (*orderProcessing*) die entsprechenden Informationen zum jeweiligen Vorgang (*orderID*) vorliegen, bevor eine Bestätigung verschickt werden kann. Trifft eine neue Nachricht zu einem spezifischen Vorgang ein wird auf die zugehörigen Nachrichten der aktuellen Warteschlange mit Hilfe der `qs:queue()` Funktion zugegriffen. Erfüllen diese die genannten Bedingungen kann eine Bestätigung verschickt werden.

5 Verwandte Arbeiten

Nachrichtenwarteschlangen spielen eine zentrale Rolle in Systemen mit asynchronen, nachrichtenbasierten Verarbeitungsmustern. Sie erlauben es dem System, die Annahme von Datenfragmenten von deren Verarbeitung zu entkoppeln und hierdurch beispielsweise temporäre Überlastung oder Kommunikationsprobleme zwischen Systemen zu kompensieren. Als Konsequenz werden Nachrichtenwarteschlangen als zugrundeliegende Datenstruktur von einer Vielzahl von Messaging- und Middleware-Lösungen verwendet. Neben der üblichen Verwendung als (temporäre) Nachrichtenspeicher finden sich in diesem Rahmen auch exotischere Anwendungsszenarien für Warteschlangen, wie beispielsweise die Verwendung zur Modellierung von zuverlässigen Automaten [21]. Auch in anderen Anwendungsgebieten werden Warteschlangen eingesetzt, wie beispielsweise als Kommunikationskanäle in einem robusten Workflow-Management System [19].

Die hohe Flexibilität, sowie die universelle Einsetzbarkeit und nicht zuletzt der erfolgreichen Einsatz von Warteschlangen in verschiedenen Anwendungsgebieten führte schließlich zu der Forderung, diese Datenstrukturen als natives Speicherformat direkt in Datenbanksysteme [17] und allgemein in verteilte Rechnerarchitekturen zu integrieren [7].

Seit Version 8 bietet die Datenbanklösung der Firma Oracle eine native Unterstützung für Nachrichtenwarteschlangen [16]. Besondere Merkmale hierbei ist die große Anzahl von unterstützten imperativen Programmiersprachen, Schnittstellen, sowie Datentransformations- und Zugriffsfunktionen [15]. Die Interaktion mit den im System gespeicherten Daten erfolgt hierbei mittels proprietären, callback-basierten Sprachanbindungen oder dem Java Messaging Service (JMS).

Auch andere Anbieter von Datenbanklösungen betrachten die Integrationsmöglichkeit von Warteschlangen als nativen Datentyp. Microsoft führte im Rahmen des SQL Server 2005 den Service Broker ein, eine Warteschlangen-basierte Erweiterung zur asynchronen Nachrichtenverarbeitung [23].

Im Gegensatz zu dem von unserer Lösung und den beiden obigen Systemen verfolgten Ansatz einer direkten Integration untersuchten Forscher von IBM einen anderen Möglichkeit zur Kombination von asynchroner Nachrichtenverarbeitung und Datenbanksystem [12]. Der Grundgedanke ist hierbei eine lose Kopplung des DB2-Datenbanksystems mit der WebSphere Middleware-Lösung über ein eigenes Wrapper-Modul, das die von der WebSphere-Plattform verwendeten JMS-Objekte in relationale Datensätze übersetzt. Hierdurch entsteht ein Impedance Mismatch bei der Verarbeitung jeder einzelnen Nachricht. Schon im Rahmen eines Prototyps mit sehr eingeschränkter Funktionalität traten bei dem verfolgten Ansatz gravierende Einschränkungen des Systems zutage, wie beispielsweise beim linearen Zugriff auf alle in einer Warteschlange gespeicherten Nachrichten, der einheitlichen, systemübergreifenden Transaktionsverwaltung sowie der Handhabung von Seiteneffekten.

Widom et al [22] gibt einen Überblick über das Konzept der unserer Anwendungssprache zugrundeliegenden Event-Condition-Action Regeln sowie deren Einsatzmöglichkeiten in aktiven Datenbanksystemen. In der Literatur finden sich einige Anwendungsvorschläge und potentielle Einsatzmöglichkeiten für ECA-Regeln in XML-basierten Datenverarbeitungssystemen, beispielsweise zur aktiven Sicherstellung von Konsistenzbedingungen bei der Modifizierung von Dokumentteilen [5]. Andere potentielle Einsatzgebiete für ECA-Regeln sind offene Datenbank-Anwendungen wie die von Bonifati et al [11] untersuchten Möglichkeiten, interessierte Abonnenten - beispielsweise per SMS - über Änderungen am Datenbestand zu informieren. Die einem Web Service Verarbeitungssystem inhärenten Anforderungen nach zuverlässiger Datenverarbeitung und performanten Datenstrukturen wird von den jeweils vorgestellten Ansätzen nicht berücksichtigt.

Die Idee, Konzepte und Forschungsergebnisse aus dem Datenbankbereich zur Unterstützung und Optimierung von verteilten Anwendungen wie Web Services einzusetzten, wird neben unserem Ansatz auch von der XL Plattform [14] und dem ActiveViews System [1] verfolgt.

Grundlage des ActiveViews Systems [1] ist eine deklarative Sprache zur Definition von Sichten, sowie deren Unterstützung durch entsprechende Sichten- und Triggermechanismen in einem der Ausführungsumgebung zugehörigen Datenbanksystem. Die durch den Benutzer deklarierten Sichten werden durch einen Anwendungsgenerator in Java-Code umgewandelt, der sowohl das durch einen Applikationsserver auszuführenden Anwendungsprogramm als auch eine browserbasierte, grafische Benutzeroberfläche generiert. Das Architektur des Active Views Systems unterscheidet sich grundlegend von unserem, eng mit dem Datenbanksystem integrierten Ansatz. Es verwendet eine Drei-Schichten-Architektur, mit einem nativen XML-Datenbanksystem zur persistenten Speicherung, einem Java-basierten Anwendungsserver, sowie einer Browser- und/oder Java-basierten Benutzerschnittstelle. Hierbei werden Java-Objekte sowohl zur Implementierung der Anwendungslogik als auch für Teile der Benutzerschnittstelle verwendet, die gegebenenfalls über Java-RMI kommunizieren. Zur persistenten Datenhaltung werden diese Java-Objekte in das XML-Format konvertiert.

Die XL-Plattform [14] vermeidet solche Impedance Mismatches durch die ausschließliche Verwendung von XML als Datenformat. Zur Datenverarbeitung verwendet das System eine eigens definierte Programmiersprache, XL. Grundlage von XL ist eine erweiterte Version von XQuery, die auch Dokument-Aktualisierungen unterstützt. Zusätzlich integriert XL Teile der Java-Syntax, beispielsweise um Fehlerbehandlung zu ermöglichen. Mit XL erstellte Programme werden von einem zugehörigen Anwendungssystem ausgeführt, das auch automatische Code-Optimierungen durchführt.

Wie die Autoren des Systems schon feststellen [14] werden für die Ausführung von verteilten Geschäftsprozessen entscheidende Anforderungen wie Zuverlässigkeit und Transaktionsunterstützung nicht durch XL berücksichtigt. Eine weitere Einschränkung der vorgeschlagenen Programmiersprache sind deren nur eingeschränkte Deklarativität, die dadurch das automatische Optimierungspotential limitiert. Im Gegensatz hierzu erlaubt der von uns vorgestellte Ansatz einer engen Integration des Web Service Verarbeitungssystems mit ei-

ner nativen XML-Datenbank die zuverlässige und transaktionale Ausführung der entsprechenden Anwendungslogik, sowohl mit lokalen als auch entfernten Kommunikationsendpunkten. Darüber hinaus bietet unserer Ansicht nach eine vollständig deklarative, regelbasierte Anwendungssprache ein deutlich höheres Potential für automatische Optimierungen.

6 Zusammenfassung und Ausblick

Die voranstehenden Kapitel beschreiben einen neuen Ansatz zur effizienten Implementierung von Web Services durch den Einsatz einer deklarativen, regelbasierten Sprache und eines entsprechenden Verarbeitungssystems. Ausgehend von einer Analyse der heute verfügbaren, meist gewachsenen Systeme stellen wir eine Architektur auf Basis eines nativen XML-Datenbanksystems mit speziellen Erweitungen zur Verwaltung von zuverlässige Nachrichtenwartenschlangen vor. Die Implementierung der Geschäftslogik erfolgt in unserem System mit Hilfe einer deklarativen Regelsprache, deren Verwendung wir anhand einiger exemplarischer Anwendungsfälle veranschaulichen.

Das vorgestellte Web Service Verarbeitungssystem bietet noch erhebliches Forschungspotential. Eine interessante Fragestellung ist die automatisierte Abbildung von Hochsprachen, die zur Definition von Geschäftsprozessen verwendet werden können (wie beispielesweise BPEL [3]), auf die unserem System zugrunde liegende, deklarative Regelsprache. Weitere Forschungsgebiete sind die Integration eines Web Services spezifischen Transaktionsmodells, sowie die Optimierung von Regelsätzen, beispielsweise auf Basis logischer Zusammenhänge zwischen den einzelnen Regeln [20].

Literatur

[1] Serge Abiteboul, Bernd Amann, Sophie Cluet, Adi Eyal, Laurent Mignet, and Tova Milo. Active views for electronic commerce. In *VLDB*, pages 138–149, 1999.

[2] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applictions*. Springer-Verlag, 2004.

[3] Tony Andrews et al. Business process execution language for web services version 1.1. Technical report, 2003. http://www.ibm.com/developerworks/library/ws-bpel.

[4] Bob Atkinson et al. Web services security (WS-Security). Technical report, 2002. http://www.ibm.com/developerworks/library/ws-secure/.

[5] James Bailey, Alexandra Poulovassilis, and Peter T. Wood. An event-condition-action language for xml. In *WWW*, pages 486–495, 2002.

[6] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Siméon. XML path language (XPath) version 2.0. Technical report, World Wide Web Consortium (W3C) Working Draft, 2002.

[7] Philip A. Bernstein, Meichun Hsu, and Bruce Mann. Implementing recoverable requests using queues. In *SIGMOD Conference*, pages 112–122, 1990.

[8] Kevin Beyer, Roberta J. Cochrane, Vanja Josifovski, Jim Kleewein, George Lapis, Guy Lohman, Bob Lyle, Fatma Özcan, Hamid Pirahesh, Normen Seemann, Tuong Truong, Bert Van der Linden, Brian Vickery, and Chun Zhang. System RX: One Part Relational, One Part XML. In *SIGMOD Conference*, pages 347–358, 2005.

[9] Ruslan Bilorusets et al. Web services reliable messaging protocol (WS-ReliableMessaging). Technical report, 2005. http://www.ibm.com/developerworks/library/ws-rm/.

[10] S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML query language. Technical report, World Wide Web Consortium, November 2005. W3C Candidate Recommendation.

[11] Angela Bonifati, Stefano Ceri, and Stefano Paraboschi. Active rules for XML: A new paradigm for e-services. *VLDB J.*, 10(1):39–47, 2001.

[12] Sangeeta Doraiswamy et al. Reweaving the tapestry: Integrating database and messaging systems in the wake of new middleware technologies. In *Data Management in a Connected World*, pages 91–110, 2005.

[13] Thorsten Fiebig, Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, Julia Neumann, Robert Schiele, and Till Westmann. Anatomy of a Native XML base management system. *VLDB Journal*, 11(4):292–314, 2003.

[14] Daniela Florescu, Andreas Grünhagen, and Donald Kossmann. XL: a platform for web services. In *CIDR*, 2003.

[15] Craig B. Foch. Oracle streams advanced queuing user's guide and reference, 10g release 2 (10.2), 2005.

[16] Dieter Gawlick and Shailendra Mishra. Information sharing with the Oracle database. In *DEBS*, 2003.

[17] Jim Gray. Queues are databases. In *Proceedings 7th High Performance Transaction Processing Workshop. Asilomar CA, Sept 1995.*, 1995.

[18] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP version 1.2. Technical report, The World Wide Web Consortium (W3C), 2003.

[19] Frank Leymann and Dieter Roller. Building a robust workflow management system with persistent queues and stored procedures. In *ICDE*, pages 254–258, 1998.

[20] Norman W. Paton, Andrew Dinn, and M. Howard Williams. Optimization. In *Active Rules in Database Systems*, pages 69–80. Springer, New York, 1999.

[21] Stefan Tai, Alexander Totok, Thomas A. Mikalsen, and Isabelle Rouvellou. Message queuing patterns for middleware-mediated transactions. In *SEM*, pages 174–186, 2002.

[22] Jennifer Widom and Stefano Ceri. *Active Database Systems: Triggers and Rules For Advanced Database Processing.* Morgan Kaufmann, 1996.

[23] Roger Wolter. An introduction to SQL server service broker. Technical report, Microsoft, 2005.