Building Scalable, Distributed Applications with Declarative Messaging

Alexander Böhm

SAP AG
alexander.boehm@sap.com

Abstract: Im Rahmen der Dissertation wird ein neuartiger Ansatz zur Implementierung von verteilten, auf dem Austausch von XML-Nachrichten basierenden Anwendungen vorgestellt. Grundlage hierbei sind ein nachrichtenorientiertes Programmiermodell sowie eine zugehörige, deklarative Regelsprache. Sie ermöglichen die in der Arbeit diskutierten Techniken zur automatischen Optimierung, effizienten Ausführung und automatisierten Verteilung von nachrichtenbasierten Anwendungen.

1 Einleitung

Verteilte, auf Nachrichtenaustausch basierte Anwendungen sind von großer und zunehmender Bedeutung für die Realisierung von unternehmensübergreifenden Geschäftsprozessen. Charakteristische Anforderungen an solche Anwendungen sind neben einfacher und effizienter Entwicklung eine hohe Ausführungsgeschwindigkeit sowie die Möglichkeit zur parallelen Ausführung auf mehreren Computersystemen, um eine möglichst hohe Skalierbarkeit sicherzustellen. Die heute gebräuchlichen Programmiersprachen und Ausführungssysteme erfüllen diese Anforderungen nur partiell (Abschnitt 2).

Im Rahmen der Dissertation wird ein neuartiger Ansatz zur Implementierung von verteilten, auf dem Austausch von XML-Nachrichten basierenden Anwendungen vorgestellt. Es werden ein nachrichtenorientiertes Programmiermodell und eine deklarative Regelsprache diskutiert, die eine kompakte Spezifikation der Anwendungslogik erlauben (Abschnitt 3.1). Diese deklarative Sprache ermöglicht die automatische Optimierung der Anwendung durch einen zugehörigen Compiler (Abschnitt 3.2). Weiterhin wird die Architektur eines Laufzeitsystems vorgestellt (Abschnitt 3.3), das die effiziente und zuverlässige Ausführung der Anwendung ermöglicht. Mehrere Messungen belegen hierbei die gegenüber dem Stand der Technik erzielten Leistungsverbesserungen (Abschnitt 3.4).

Ein weiterer Schwerpunkt der Arbeit ist die durch die deklarative Spezifikation der Geschäftslogik ermöglichte, automatisierte Verteilung der Anwendung auf mehrere Ausführungssysteme. Dies beinhaltet die Analyse der Datenabhängigkeiten innerhalb der Anwendung (Abschnitt 4.1), die Allokation unabhängiger Anwendungsteile auf einzelne Ausführungssysteme (Abschnitt 4.2) sowie die Generierung individueller, unabhänger Teilanwendungen (Abschnitt 4.3). Des weiteren werden formale Transformationsvorschriften vorgestellt (Abschnitt 4.4), die es erlauben, die Zahl der unabhängigen Anwendungstei-

le zu erhöhen. Hierdurch lässt sich die Skalierbarkeit von verteilten, nachrichtenbasierten Anwendungen deutlich verbessern (Abschnitt 4.5).

2 Verteilte, nachrichtenbasierte Anwendungen

2.1 Anwendungsszenarien

Verteilte, nachrichtenbasierte Anwendungen bestehen aus mehreren, räumlich verteilten Teilnehmern, die über den Austausch von XML-Nachrichten kommunizieren. Solche Anwendungen lassen sich als Netzwerk von Knoten visualisieren, zwischen denen XML-Nachrichten ausgetauscht werden.

Die einzelnen Knoten stellen somit einen Teil der Geschäftslogik der Gesamtanwendung zur Verfügung. Jede solche Teilanwendung reagiert hierbei auf die von anderen Knoten eingehenden Nachrichten. Das Ergebnis der im Rahmen dieser Teilanwendung abgearbeiteten Geschäftslogik wird für die anderen Knoten erneut in Form von an sie geschickten Nachrichten sichtbar.

Verteilte, auf dem Austausch von XML-Nachrichten basierte Anwendungen werden heutzutage in einer Vielzahl von Bereichen eingesetzt. Typische Beispiele sind unter anderem verteilte Finanzanwendungen [Ltd09], unternehmensübergreifende Geschäftsprozesse [ZDGH05] und soziale Online-Anwendungen [Fac10].

2.2 Stand der Technik

Die heute verfügbaren Lösungen zur Realisierung verteilter, nachrichtenbasierter Anwendungen ([ACKM04] gibt einen Überblick) basieren größtenteils auf existierenden Middleware-Lösungen, die durch zusätzliche Komponenten zur Nachrichtenverarbeitung erweitert werden. Dieser Ansatz erlaubt die schnelle Integration der benötigten Schnittstellen in bestehende Infrastruktur, erschwert jedoch die Implementierung von effizienten, nachrichtenbasierten Anwendungen:

- 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 *Impedance Mismatch* verringert die Ausführungsgeschwindigkeit des angebotenen Dienstes. Darüber hinaus hat die starke Segmentierung negativen Einfluß auf die Wartbarkeit, Stabilität und Flexibilität des Gesamtsystems [Sto02].

- Die Implementierung der Geschäftslogik im Rahmen existierender Applikationsserver erfolgt weitgehend durch objektorientierte Programmiersprachen, wie bei-

spielsweise 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 effizienten Anwendungen.

- Die Skalierbarkeit einer mit dem aktuellen Stand der Technik realisierten, nachrichtenbasierten Anwendung ist durch die Leistungsfähigkeit des (einzelnen) Rechners begrenzt, auf dem sie ausgeführt wird. Um auf mehreren Computersystemen parallel ausgeführt werden zu können, müssen solche Anwendungen üblicherweise manuell umgeschrieben werden (z.B. um die Synchronisation von gemeinsam verwendeten Daten sicherzustellen). Dies verursacht einen erheblichen Mehraufwand an Entwicklungszeit und Kosten.

3 Deklarative Nachrichtenverarbeitung

Im Rahmen der Arbeit wird ein neuartiges Programmiermodell entwickelt, das die Geschäftslogik eines Teilnehmers in einer verteilten, nachrichtenbasierten Anwendung auf Basis zweier grundlegender Komponenten beschreibt. *Nachrichtenwarteschlangen* stellen eine Schnittstelle für asynchrone Kommunikation zur Verfügung und dienen darüber hinaus zur zuverlässigen Speicherung von XML-Nachrichten. Eine Menge von *deklarativen Regeln* implementiert die Anwendungslogik auf der Grundlage der in den Warteschlangen vorliegenden Nachrichten. Jede einzelne Regel spezifiziert hierbei, wie auf den Eingang einer neuen Nachricht in einer Warteschlange zu reagieren ist, und erzeugt als Konsequenz wiederum neue Nachrichten.

Im folgenden Abschnitt 3.1 wird vorgestellt, wie diese beiden grundlegenden Komponenten zur Implementierung der Geschäftslogik einer verteilten, nachrichtenbasierten Anwendung eingesetzt werden können. Die deklarative Formulierung der Anwendungslogik erlaubt eine weitestgehend automatische Optimierung der Anwendung durch einen entsprechenden Compiler (Abschnitt 3.2). Dieser überführt die initiale Anwendungsspezifikation in ein ausführbares Programm für das zugehörige Laufzeitsystem (Abschnitt 3.3).

3.1 Programmiermodell

3.1.1 Nachrichtenwarteschlangen

Das entwickelte Programmiermodell stellt zwei unterschiedliche Typen von Warteschlangen zur Verfügung. *Gateway queues* dienen zur nachrichtenbasierten Kommunikation mit externen Systemen (z.B. Web Services). Warteschlangen dienen zudem als zuverlässige, dauerhafte Speicherstukturen für XML-Nachrichten. Diese *basic queues* erlauben es einer Anwendung, Zwischenergebnisse und Hilfsnachrichten lokal abzuspeichern, ohne diese an externe Systeme zu versenden. Als Konsequenz können alle Nachrichten, unabhängig davon, ob sie von einem externen Teilnehmer oder einer lokalen Anwendungsregel erzeugt wurden, durch einheitliche Speicher- und Zugriffsmechanismen verarbeitet werden.

3.1.2 Deklarative Regeln

Die Geschäftslogik einer verteilten, nachrichtenbasierten Anwendung wird mit Hilfe einer Menge von deklarativen Regeln ausgedrückt, die die in den Warteschlangen vorliegenden XML-Nachrichten verarbeiten und neue Nachrichten erzeugen. Diese neuen Nachrichten dienen wiederum als Eingabe für weitere Regeln oder können mit Hilfe von gateway queues an externe Systeme verschickt werden.

Zur Spezifikation dieser Anwendungsregeln kommt eine speziell entwickelte Regelsprache zum Einsatz. Diese basiert auf XQuery [BCF+07], einer deklarativen Anfragesprache für XML-Dokumente. Sie ermöglicht Entwicklern die direkte Verarbeitung der in den Warteschlangen vorliegenden XML-Nachrichten ohne aufwändige Typumwandlungen oder zusätzliche Programmierschnittstellen. Zusätzlich zu den bereits durch XQuery zur Verfügung gestellten Verarbeitungsfunktionen wie Dokumentkonstruktion, Datenextraktion und Schemavalidierung bietet die entwickelte Sprache einige notwendige Erweiterungen:

Regeldefinition Jede Regel ist auf einer Warteschlange der Anwendung definiert und hat einen anwendungsweit eindeutigen Namen. Um dem Anwendungsentwickler die Zuordnung von Regeln zu Warteschlangen zu ermöglichen, bietet die Regelsprache ein entsprechendes Definitionskonstrukt an.

Nachrichtenversand Das Ergebnis jeder Anwendungsregel ist eine (potentiell leere) Sequenz von Nachrichten, die in andere Warteschlangen eingefügt werden sollen (entweder zur Weiterverarbeitung durch eine andere Regel oder zum Versand an ein externes System). Da die der Regelsprache zugrunde liegende Anfragesprache XQuery keine Seiteneffekte unterstützt, wurde hierzu die Regelsprache in Anlehnung an die XQuery Update Facility [CDF+09] erweitert. Jede Anwendungsregel kann hierdurch mit Hilfe eines entsprechenden Konstrukts (`enqueue message`) Nachrichten in andere Warteschlangen einfügen.

Deklarative Sichten auf die Nachrichtenhistorie Die dem im Rahmen der Arbeit entwickelten Programmiermodell zugrunde liegenden Warteschlangen dienen nicht nur als temporäre Zwischenspeicher, sondern als dauerhafte Ablage für Nachrichten. So wird eine Nachricht nach ihrer Verarbeitung nicht gelöscht, sondern bleibt weiterhin im System vorhanden und zugreifbar. Hieraus ergibt sich die Möglichkeit, den aktuellen Zustand der Anwendung mit Hilfe einer deklarativen Anfrage an die komplette Nachrichtenhistorie auszudrücken. Dies ermöglicht die einheitliche Behandlung von Nachrichten und Zustandsinformationen und bietet hierdurch weitere Freiheitsgrade bei der Anwendungsoptimierung und -ausführung durch den zugehörigen Compiler (Abschnitt 3.2) und das Laufzeitsystem (Abschnitt 3.3).

Um den Zugriff auf für die Anwendung relevante Teile der Nachrichtenhistorie zu vereinfachen, bietet die entwickelte Programmiersprache entsprechende deklarative Sichten an. Konzeptuell ähneln diese *slicings* den für relationale Datenbankmanagementsysteme entwickelten parametrisierten Sichten [Toy86]. Sie erlauben die Definition von logischen Nachrichtengruppen, auf die im Rahmen von Anwendungsregeln mit einem einfachen Funktionsaufruf zugegriffen werden kann.

Beispiel

Im folgenden Beispiel wird die praktische Verwendung der oben diskutierten Komponenten der Regelsprache kurz erläutert. Zuerst wird eine Warteschlange (basic queue) mit dem Namen `incomingRequests` definiert (Zeile 1).

Anschließend wird eine neue Regel mit dem Namen `sendOrderConfirmation` definiert (Zeile 3). Diese Regel wird ausgeführt, falls eine Nachricht in die zuvor definierte Warteschlange eingefügt wird. Mit Hilfe eines XPath-Ausdrucks wird nun überprüft, ob es sich bei der eingehenden Nachricht um eine Bestellung handelt (Zeile 4). Ist dies der Fall, so wird auf die Inhalte der Nachricht (Zeile 6 und 7) zugegriffen, des weiteren werden mit Hilfe einer deklarativen Sicht die Stammdaten des Kunden aus der Nachrichtenhistorie abgefragt (Zeile 8). Abschließend wird eine Ergebnisnachricht erzeugt und mit Hilfe des `enqueue message`-Konstrukts in die Warteschlange `customerReplies` eingefügt.

```
1  create queue incomingRequests kind basic mode persistent;
2
3  create rule sendOrderConfirmation for incomingRequests
4  if(//order)
5  then
6    let $items := //order/items
7    let $customerID := //customerID/text()
8    let $address := qs:slice($customerID, "customerMasterdata")
9    let $result := <confirmation>
10   <status>OK</status> ... </confirmation>
11   return enqueue message $result into customerReplies
12 else();
```

3.1.3 Ausführungsmodell

Die Datenverarbeitung in einer verteilten, nachrichtenbasierten Anwendung lässt sich als einfacher Zyklus beschreiben. In jeder Iteration dieses Zyklus wird entschieden, welche Nachrichten aktuell zu verarbeiten sind, wie diese Nachrichten zu verarbeiten sind und welches Ergebnis sich aus der Nachrichtenverarbeitung ergibt. Um eine deterministische Verarbeitung von Nachrichten sicherzustellen und gleichzeitig möglichst hohe Freiheitsgrade für automatische Anwendungsoptimierungen zu ermöglichen, werden die folgenden Anforderungen an einen solchen Ausführungszyklus definiert.

1. Jede Nachricht wird genau einmal verarbeitet. Die Verarbeitung erfolgt durch die Auswertung aller Regeln, die auf der Warteschlange, in der die Nachricht eingefügt wurde, definiert sind.

2. Die Regelverarbeitung erfolgt durch die Auswertung der Regeln nach der XQuery (Update) Semantik unter Einbeziehung der zusätzlichen Funktionen zum Zugriff auf die Nachrichtenhistorie (Abschnitt 3.1.2). Das jeweilige Ergebnis ist eine Sequenz von Ergebnisnachrichten.

3. Das Ergebnis der Regelauswertung für eine Nachricht ist eine beliebige Verkettung aller entstandener Nachrichtensequenzen.
4. Die Verarbeitung der Ergebnisnachrichten ist atomar, d.h. alle aus der Regelverarbeitung entstandenen Nachrichten werden im Kontext einer atomaren Transaktion in die entsprechenden Ziel-Warteschlangen eingefügt. Ebenfalls wird die aktuell verarbeitete Nachricht im Rahmen dieser Transaktion als verarbeitet gekennzeichnet.
5. Jeder Regelauswertungsvorgang für eine gegebene Nachricht sieht die gleiche Nachrichtenmenge beim Zugriff auf die Nachrichtenhistorie des Systems. Insbesondere sind nur solche Nachrichten sichtbar, die *vor* der aktuell zu verarbeitenden Nachricht eingefügt worden sind.

Diese Anforderungsliste beinhaltet starke, transaktionale Garantien (z.B. atomare Verarbeitung, Isolation paralleler Auswertungsprozesse), die zur Implementierung von zuverlässigen, verteilten Anwendungen benötigt werden. Gleichzeitig bietet sie jedoch auch Spielraum für weitgehende Optimierungsmöglichkeiten, die durch einen zugehörigen Compiler ausgenutzt werden können.

3.2 Regelcompiler

Die Aufgabe des Regelcompilers ist es, eine gegebene Anwendungsspezifikation in eine durch ein zugehöriges Laufzeitsystem (Abschnitt 3.3) ausführbare Form zu überführen. Durch die deklarative Spezifikation der Geschäftslogik ergeben sich hierbei weitgehende Optimierungsmöglichkeiten für den Compiler, durch die die spätere Ausführungsgeschwindigkeit erhöht werden kann. Optimierungspotentiale existieren hierbei auf verschiedenen Ebenen.

Umschreiben des Regelsatzes Das Ausführungsmodell erlaubt die weitreichende Modifikation des einer Anwendung zugrunde liegenden Regelsatzes. Beispielsweise kann der Compiler alle auf einer gemeinsamen Warteschlange definierten Regeln in eine einzelne, kombinierte Regel zusammenfassen. Dies erlaubt u.a. die Faktorisierung gemeinsamer Teilausdrücke und vermeidet den wiederholten Aufruf der Regelauswertungskomponente des Laufzeitsystems für eine einzelne Nachricht.

Optimierung einzelner Regeln Ein erheblicher Teil der entwickelten Regelsprache basiert auf der deklarativen Anfragesprache XQuery. Hierdurch lassen sich viele der für XQuery entwickelte Optimierungstechniken zur Optimierung der Anwendungsregeln nutzen.

Physische Optimierung Physische Optimierungen erlauben die selektive Anpassung von Anwendungsregeln an spezifische Merkmale des verwendeten Laufzeitsystems. Beispielsweise bietet das im Rahmen der Arbeit entwickelte Laufzeitsystem (Abschnitt 3.3) besonders effiziente Mechanismen zum Weiterleiten von unveränderten Nachrichten zwischen einzelnen Warteschlangen an. Erkennt der Compiler einen solchen Fall, kann er die entsprechenden Primitiven des Laufzeitsystems verwenden, um die spätere Ausführung zu beschleunigen.

3.3 Laufzeitsystem

Die Ausführung des durch den Compiler erzeugten Ergebnisses wird durch ein im Rahmen der Arbeit entwickeltes Laufzeitsystem übernommen. Dieses System besteht aus drei Hauptkomponenten.

Transaktionaler Nachrichtenspeicher Die Grundlage für das Laufzeitsystem bildet Natix [FHK+03], ein an der Universität Mannheim entwickeltes, natives XML-Datenbankmanagementsystem. Natix erlaubt die effiziente und zuverlässige Speicherung von XML-Dokumenten sowie deren transaktionale Verarbeitung. Zusätzlich wurde das System erweitert, um eine Warteschlangen-basierte Nachrichtenverwaltung, effiziente Mehrbenutzersynchronisation sowie mit Hilfe von Indexstrukturen schnellen Zugriff auf die Nachrichtenhistorie (slicings) zu ermöglichen.

Regelauswertungskomponente Die Regelauswertungskomponente implementiert das in Abschnitt 3.1.3 beschriebene Ausführungsmodell. Auf Basis der Anwendungsspezifikation entscheidet sie, wann und in welcher Reihenfolge Nachrichten abgearbeitet, wann neue Nachrichten in den Nachrichtenspeicher eingefügt und wann Nachrichten mit externen Systemen ausgetauscht werden. Neben einer modellkonformen Regelausführung liegt der Schwerpunkt hierbei auf einer möglichst schnellen und effizienten Regelverarbeitung, z.B. durch den Einsatz von mehreren parallelen Ausführungsprozessen.

Kommunikationssystem Das Kommunikationssystem ermöglicht dem Laufzeitsystem den Nachrichtenaustausch mit den anderen, an der verteilten Anwendung teilnehmenden Knoten. Hierzu werden verschiedene, sowohl synchrone wie auch asynchrone Transportprotokolle (HTTP, SMTP, ...) unterstützt.

3.4 Evaluation

Mächtigkeit und Benutzbarkeit der Regelsprache Um die praktische Tauglichkeit der Regelsprache zu erproben, wurden im Rahmen der Arbeit verschiedene Fallstudien von nachrichtenbasierten Anwendungen implementiert. Des weiteren wurde untersucht, inwieweit sich für nachrichtenbasierte Anwendungen charakteristische Entwurfsmuster [HW04, vdAtHKB03] realisieren lassen. Hierbei zeigte sich, dass sich selbst komplexe Entwurfsmuster mit nur geringem Entwicklungsaufwand umsetzen lassen. Des weiteren ermöglichte die Regelsprache die schnelle Entwicklung der untersuchten Fallstudien, die sich im Vergleich zu existierenden Programmiersprachen, wie beispielsweise Java, deutlich kompakter implementieren ließen.

Leistungsfähigkeit des Laufzeitsystems Um die Leistungsfähigkeit des entwickelten Laufzeitsystems zu evaluieren, wurde die Ausführungsleistung des Systems mit der eines kommerziell verfügbaren Anwendungsservers verglichen. Als Anwendung wurde hierbei eine vereinfachte Version des TPC-App Benchmarks [Tra08] verwendet. Die Ausführungsgeschwindigkeit des Laufzeitsystems liegt dabei sehr deutlich über der des kommer-

ziellen Systems. Besonders anschaulich wird dies mit zunehmender Größe des zu verwaltenden Zustands sowie bei einer steigenden Zahl von parallelen Anwendungsinstanzen.

4 Automatische Anwendungsverteilung

Im letzten Teil der Arbeit wird die Möglichkeit zur automatischen Verteilung einer mit Hilfe der Regelsprache realisierten Anwendung auf mehrere Ausführungssysteme (Knoten) untersucht. Ziel hierbei ist es, die Skalierbarkeit der Anwendung zu verbessern, ohne - wie bei den meisten heute verfügbaren Programmiersprachen - aufwändige manuelle Anpassungen durchführen zu müssen. Das entwickelte Verfahren besteht aus vier einzelnen Schritten, die im Rahmen eines entsprechenden Compilers implementiert wurden und in den folgenden Abschnitten kurz vorgestellt werden.

4.1 Anwendungsfragmentierung

In einem ersten Schritt wird die Anwendung in einzelne, voneinander unabhängige Anwendungsfragmente, jeweils bestehend aus Warteschlangen und zugehörigen Regeln, zerlegt. Zwei Fragmente werden dabei als unabhängig angesehen, wenn die auf den Warteschlangen definierten Regeln des einen Fragments nicht auf die in den Warteschlangen des anderen Fragments abgelegten Nachrichten zugreifen. Diese Datenabhängigkeiten lassen sich leicht durch die mit Hilfe der deklarativen Sichten (Abschnitt 3.1.2) modellierten Zugriffsbeziehungen erkennen. Das Ergebnis der Anwendungsfragmentierung ist eine Menge von Fragmenten $F = \{f_1, \ldots, f_n\}$. Jedes Fragment $f \in F$ enthält eine disjunkte Teilmenge der Warteschlangen der Anwendungsspezifikation.

4.2 Fragmentallokation

Nachdem die einzelnen Fragmente, in die eine Anwendung zerlegt werden kann, identifiziert sind, werden in einem nächsten Schritt diese n Fragmente auf eine zur Verfügung stehende Menge von h Knoten verteilt. Ziel ist eine möglichst gleichmäßige Verteilung des erwarteten Lastenaufkommens auf diese Knoten, um eine möglichst hohe Skalierbarkeit und Ausführungsgeschwindigkeit zu erreichen. Zu diesem Zweck versucht die Fragmentallokation die Netzwerkkommunikation zwischen den einzelnen Knoten zu minimieren, indem oft miteinander kommunizierende Fragmente auf dem gleichen Knoten platziert werden. Gleichzeitig wird versucht, die erwartete CPU- und I/O-Last möglichst gleichmäßig auf die h verfügbaren Knoten zu verteilen. Hierzu werden durch Sampling gewonnene Kostenabschätzungen sowie diverse Allokationsheuristiken verwendet.

Basierend auf einer gegebenen Anwendungsspezifikation A und einer zugehörigen Fragmentierung F erzeugt die Fragmentallokation eine Zuordnung $G = \{g_1, \ldots, g_h\}$ der Fragmente zu den h zur Verfügung stehenden Knoten. Jede Zuordnung $g \in G$ enthält eine disjunkte, potentiell leere Teilmenge der Anwendungsfragmente.

4.3 Generierung einzelner Teilanwendungen

Nachdem die Anwendungsfragmente den zur Verfügung stehenden Knoten zugeordnet wurden, wird die initiale, vom Programmierer erstellte Anwendung A in eine Menge von unabhängigen Teilanwendungen $\{A_1, \ldots A_h\}$ transformiert. Jede dieser Anwendungen ist selbstständig auf dem Zielknoten ausführbar und stellt einen Teil der Funktionalität der ursprünglichen Anwendung zur Verfügung.

4.4 Skalierbarkeitstransformationen

Im günstigsten Fall wird durch die oben besprochenen drei Schritte eine initiale Anwendungsspezifikation A in eine Menge von unabhängigen Teilanwendungen $\{A_1, \ldots A_h\}$ transformiert, die das zu erwartende Lastenaufkommen optimal auf eine Menge von h verfügbaren Knoten verteilen.

Üblicherweise schränken jedoch die innerhalb einer Anwendung bestehenden Datenabhängigkeiten die Anzahl der unabhängigen Fragmente, in die die Anwendung zerlegt werden kann, stark ein. Selbst wenn die Anwendung in eine ausreichende Anzahl von Fragmenten zerlegt werden kann, so dass mindestens ein Fragment jedem verfügbaren Knoten zugeordnet werden kann ($F = \{f_1, \ldots, f_n\} : n > h$), kann das Ergebnis suboptimal sein: Wie sich in mehreren Fallstudien zeigte, ist die Anwendungslast typischerweise ungleichmäßig zwischen den einzelnen Fragmenten verteilt, d.h. ein einzelnes, nicht weiter zerlegbares Fragment muss einen signifikanten Anteil der Anwendungslast bewältigen.

Diesem Problem lässt sich mit den im Rahmen der Arbeit entwickelten Skalierbarkeitstransformationen begegnen. Ziel dieser Transformationen ist es, die Anzahl der unabhängigen Fragmente, in die eine Anwendung zerlegt werden kann, zu erhöhen. Dies geschieht durch das Umschreiben der initialen Anwendungsspezifikation A in eine Spezifikation A', die eine potentiell bessere Lastverteilung und Skalierbarkeit der Anwendung ermöglicht. Eine grundlegende Idee hierbei ist es, Teile der Nachrichtenhistorie, die eine weitere Zerlegung von Teilfragmenten verhindern, zu replizieren und dadurch eine weitere Aufteilung von Fragmenten zu ermöglichen. Des weiteren lassen sich Zugriffe auf die Nachrichtenhistorie mittels deklarativer Sichten (Abschnitt 3.1.2) in expliziten Nachrichtenfluss transformieren sowie aus dem Datenbankbereich bekannte Techniken wie vertikale und horizontale Datenpartitionierung einsetzen, um das Verteilungspotential weiter zu erhöhen.

4.5 Evaluation

Die praktische Anwendbarkeit des entwickelten Ansatzes zur automatischen Verteilung von nachrichtenbasierten Anwendungen wurde mit Hilfe von mehreren Fallstudien verifiziert. Hierbei konnten alle untersuchten Anwendungen ohne manuelle Anpassungen auf mehrere Knoten verteilt werden. Bereits durch die Anwendung von wenigen Skalierbarkeitstransformationen (Abschnitt 4.4) ließ sich das Verteilungspotential (und damit die Skalierbarkeit) der untersuchten Anwendungen deutlich erhöhen.

Literatur

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno und Vijay Machiraju. *Web Services: Concepts, Architectures and Applictions*. Springer, 2004.

[BCF+07] Scott Boag, Donald Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie und Jérôme Siméon. XQuery 1.0: An XML Query Language. Bericht, World Wide Web Consortium (W3C), January 2007.

[CDF+09] Donald Chamberlin, Michael Dyck, Daniela Florescu, Jim Melton, Jonathan Robie und Jérôme Siméon. XQuery Update Facility 1.0. Bericht, World Wide Web Consortium (W3C), June 2009.

[Fac10] Facebook Homepage. World Wide Web, March 2010. `http://facebook.com/`.

[FHK+03] Thorsten Fiebig, Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, Julia Neumann, Robert Schiele und Till Westmann. Anatomy of a native XML Base Management System. *VLDB Journal*, 11(4):292–314, 2003.

[HW04] Gregor Hohpe und Bobby Woolf. *Enterprise Integration Patterns*. Pearson Education, Inc., 2004.

[Ltd09] FIX Protocol Ltd. Financial Information eXchange (FIX) Protocol. Bericht, April 2009.

[Sto02] Michael Stonebraker. Too Much Middleware. *SIGMOD Record*, 31(1):97–106, 2002.

[Toy86] Motomichi Toyama. Parameterized View Definition and Recursive Relations. In *Proceedings of the Second International Conference on Data Engineering, February 5-7, 1986, Los Angeles, California, USA*, Seiten 707–712. IEEE Computer Society, 1986.

[Tra08] Transaction Processing Performance Council (TPC). TPC BENCHMARK App (Application Server) Specification Version 1.3. Bericht, February 2008.

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski und Alistair P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[ZDGH05] Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler und Kerard Hogg. Service-oriented architecture and business process choreography in an order management scenario: Rationale, concepts, lessons learned. In Ralph Johnson und Richard P. Gabriel, Hrsg., *OOPSLA Companion*, Seiten 301–312. ACM, 2005.

Alexander Böhm wurde am 29. April 1979 in Heidelberg geboren. Nach dem Studium der Wirtschaftsinformatik an der Universität Mannheim mit den Schwerpunkten Datenbankmanagementsysteme, Computernetzwerke und Operations Research arbeitete er im Rahmen der Forschungsgruppe Datenbanken der Universität Mannheim an seiner Doktorarbeit über deklarative Nachrichtenverarbeitung. Seit dem erfolgreichen Abschluß seiner Promotion im Jahr 2010 ist er bei SAP im Bereich der In-Memory Datenbanksysteme tätig.