Klaudia Hergula, Theo Härder

# Anfrageoptimierung bei der Funktionsintegration in Föderierten Datenbanksyste-

*Unternehmen sind zunehmend mit der Integration ihrer Systeme beschäftigt, um alle ihre Informationen gemeinsam verarbeiten zu können. Dabei ist die reine Datenintegration nicht mehr ausreichend, sondern der Integrationsansatz muss um die Integration von Funktionen erweitert werden. Wir stellen hierzu eine Integrationsarchitektur auf Basis eines FDBS zur Daten- und eines Flow-Systems zur Funktionsintegration vor. Die Kopplung der beiden Systeme erfolgt über die Wrapper-Technologie.*

*Dabei ist ein neuer und wichtiger Aspekt die Anfrageverarbeitung über beide Komponenten hinweg. Wir zeigen auf, wie Funktionsaufrufe in die relationale Welt eingebettet werden können und wie zusätzliche Funktionalität im Wrapper die Anfrageoptimierung unterstützen kann. Darüber hinaus führen wir ein angepasstes Kostenmodell ein, das Funktionsaufrufe berücksichtigt.*

*Die Untersuchung und Bewertung zweier Flow-Varianten hinsichtlich ihrer Funktionalität und Performanz schließen unsere Betrachtungen ab.*

## 1 Einleitung

Unternehmen sind zunehmend mit immer schneller anwachsenden Mengen an Daten und Informationen konfrontiert. Jüngsten Studien zufolge weisen geschäftsrelevante Informationen eine jährliche Wachstumsrate von rund 50 Prozent auf, wobei ein bis zwei Exabytes ($10^{18}$) an Informationen generiert werden [Varian, Lyman 2003]. Die Verwaltung solcher Datenmengen stellt heute noch ein erhebliches Problem dar, denn Data Warehouses haben heute erst mehrere Terabytes im Griff. Noch viel interessanter ist jedoch die Tatsache, dass ein großer Teil dieser Informationen in sog. *Pakkaged Software* oder *Packaged Applications* anfällt (wie z.B. SAP [SAP 2003]). Um trotzdem alle Informationen aus den Daten in den verschiedensten Systemen zu bekommen, sind Unternehmen zunehmend mit der Integration dieser Pakete beschäftigt.

Auch die Integrationsthematik hat inzwischen mehrere Gesichter bekommen. Spielte in der Vergangenheit vor allem die Datenintegration eine große Rolle, so haben sich inzwischen aufgrund unterschiedlicher Anforderungen weitere Formen der Integration entwickelt [Jhingran et. al. 2002]: *Portal-Integration*, die unterschiedliche Anwendungen über eine Web-Seite zusammenführt, *Geschäftsprozessintegration*, die Prozesse über mehrere Anwendungen hinweg aufbaut, *Anwendungsintegration*, die Anwendungen miteinander kommunizieren lässt, und schließlich die *Informationsintegration*, die heterogene Daten zusammenführt, so dass Anwendungen auf alle relevanten Unternehmensdaten zugreifen können.

Die Informationsintegration hat zwei grundlegende Aspekte [Leymann, Roller 2002]: Datenintegration und Funktionsintegration. Die Daten aus heterogenen Quellen werden über gemeinsame Schnittstellen und integrierte Schemata zugänglich gemacht, so dass sie dem Benutzer wie eine einzige Datenquelle erscheinen. Die Integration von Funktionen hingegen soll lokale Funktionen verschiedener Systeme in einheitlicher Form bereitstellen. Dem Benutzer wird so eine homogene Menge an Funktionen an die Hand gegeben, über die er die Daten manipulieren kann, die von den verschiedenen Anwendungen gekapselt werden.

Möchte ein Unternehmen alle relevanten Daten integriert zugängig machen, unabhängig davon, wie auf sie zugegriffen werden kann, so müssen Daten *und* Funktionen integriert werden. Für die essentiellen Probleme auf dem Gebiet der Daten- bzw. Datenbankintegration liegen inzwischen mächtige Lösungen vor, wenn auch noch einige Fragen offen sind [Kim 1995; Sheth, Larson 1990]. Im Bereich der Funktionsintegration als auch der Kombination beider Integrationsformen sind bisher nur wenige Ansätze bekannt. Daher konzentrieren wir uns in dem vorliegenden Aufsatz auf diese Integrationsform.

Das folgende Beispiel soll verdeutlichen, warum die Funktionsintegration notwendig ist. Es zeigt, wie Benutzer heutzutage mit Anwendungssystemen arbeiten. Unser Beispielszenario ist in der Einkaufsabteilung eines Unternehmens angesiedelt, findet sich aber in ähnlicher Form auch in jedem anderen Bereich. In dieser Abteilung hat ein Mitarbeiter zu entscheiden, ob ein neues Produkt eines dem Unternehmen bereits bekannten Zulieferers bestellt werden soll. Ein Einkaufssystem soll dem Mitarbeiter mit der Funktion `Kaufentscheid` bei seiner Entscheidung helfen. Diese Funktion schlägt eine Entscheidung zum Kauf vor, die auf einem berechneten Qualitäts- und Zuverlässigkeitsgrad und der Nummer der zu betrachtenden Komponente basiert. Leider kennt der Mitarbeiter nur den Komponentennamen und die Nummer des Zulieferers. Da ihm die benötigten Eingabewerte nicht zur Verfügung stehen, muss der Mitarbeiter weitere Systeme heranziehen, um diese Werte zu ermitteln. Abbildung 1 illustriert die einzelnen Schritte, die er dafür ausführen muss. Ausgehend von den Werten, die ihm bekannt sind, ruft er zunächst die Funktion `Get_Qualität` des Lagerhaltungssystems und die Funktion `Get_Zuverlässigkeit` des Einkaufssystems jeweils mit der Zulieferernummer auf, um Qualität und Zuverlässigkeit des Zulieferers zu erfragen. Die resultierenden Ergebnisse werden anschließend als Eingabe für die Funktion `Get_Grad` zur Berechnung des Grades eingesetzt, um den ersten Eingabewert der Funktion `Kaufentscheid` zu erhalten. Außerdem ruft der Mitarbeiter die Funktion `Get_KompNr` des Produktdatenmanagementsystems zur Abfrage der zugehörigen Komponentennummer auf. Nun stehen ihm die Werte zur Verfügung, die für den Aufruf der Funktion `Kaufentscheid` benötigt werden.

Während des Entscheidungsprozesses muss der Mitarbeiter mit drei unterschiedlichen Anwendungssystemen und demzufolge mit drei verschiedenen Benutzerschnittstellen arbeiten. Aus technischer Sicht setzt er von Hand eine Art In-

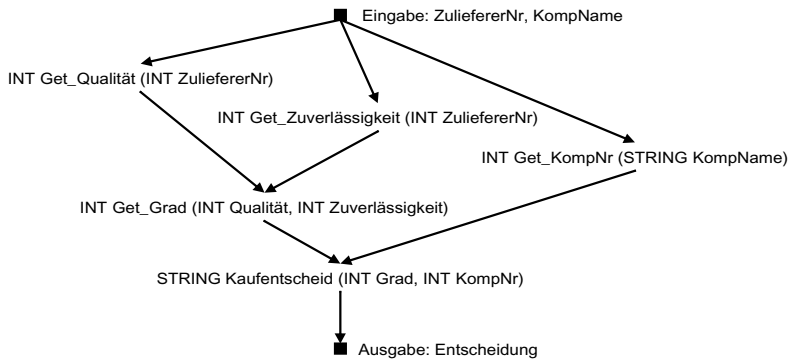


**Abb. 1: Beispiel einer föderierten Funktionsausführung.**

tegration um, indem er die Funktionen der Anwendungssysteme aufruft und deren Ergebnisse zwischen den einzelnen Systemen hin- und herkopiert. Die Interaktion des Benutzers stellt daher die Verknüpfung der Anwendungssysteme dar. Ein solches Verfahren kostet den Benutzer aber zum einen sehr viel Zeit und zum anderen ist es fehleranfällig. Als typisches Beispiel hierfür mag dienen, dass oft alte Werte im Zwischenspeicher versehentlich als Eingabe in ein anderes System weitergereicht werden. Außerdem werden bestimmte Schritte immer wieder in derselben Reihenfolge durchgeführt. Diese Beobachtung führte zu der Idee, so genannte *föderierte Funktionen* anzubieten, welche die einzelnen Aufrufe der lokalen Funktionen der Anwendungssysteme implementieren und diese Schritte vor dem Benutzer verbergen. In dem gezeigten Beispiel muss der Benutzer lediglich eine föderierte Funktion `KaufeKomponente` statt der fünf lokalen Funktionen aufrufen.

Muss neben den Funktionen noch auf Daten zugegriffen werden, benötigen wir eine Integrationslösung, die Daten und Funktionen integrieren kann. Unseres Erachtens stellt ein FDBS eine effektive Integrationsplattform dar. Wichtige Gründe hierfür sind zum einen die deklarative Anfragesprache sowie die inzwischen ausgiebig untersuchte Integration von Datenbanksystemen. Zum anderen sind oft bereits vorhandene SQL-Kenntnisse als auch die große Anzahl an Software-Produkten, welche die SQL-Schnittstelle bedienen können, wichtige Entscheidungskriterien für den Einsatz von neuen Lösungen. Eine Anfrage, die den gemeinsamen Zugriff auf Datenbanken und Anwendungssysteme erfordert, enthält somit SQL-Prädikate und Aufrufe von externen Funktionen.

Für die Funktionsintegration ist die Flow-Technologie ein viel versprechender Ansatz, deren bekannteste Ausprägung der Workflow ist. Flows ermöglichen die Verknüpfung von lokalen Funktionen zu föderierten Funktionen. Diese können anschließend über das FDBS zugängig gemacht werden, so dass zugleich eine Integration von Daten und Funktionen unterstützt wird.

In diesem Aufsatz soll ein Ansatz zu solch einer erweiterten Integration vorgestellt werden, der auf der Kopplung eines FDBS und einem Flow-System basiert. Die beiden Systeme können nur dann sinnvoll eingesetzt werden, wenn eine übergreifende Anfrageverarbeitung bereitgestellt werden kann. Sie muss in der Lage sein, zwei völlig unterschiedliche Anfragemodelle zusammenzuführen und akzeptable Antwortzeiten zu gewährleisten. Wir legen daher den Schwerpunkt dieses Aufsatzes auf die Umsetzung einer solchen Anfrageverarbeitung und untersuchen zudem, welche Performanz unsere Architektur erwarten lässt.

Hierzu stellen wir in Abschnitt 2 zunächst unsere Integrationsarchitektur vor, in der ein Wrapper das FDBS und das Flow-System koppelt. Wie die Anfrageverarbeitung über diese beiden Komponenten hinweg aussehen kann, betrachten wir in Abschnitt 3. Anschließend bewerten wir zwei unterschiedliche Flow-Implementierungen anhand ihrer Performanz in Abschnitt 4, bevor wir abschließend unsere Ergebnisse zusammenfassen.

## 2 Integrationsarchitektur

Unsere Drei-Schichten-Integrationsarchitektur soll transparenten Zugriff zu heterogenen Datenquellen ermöglichen, unabhängig davon, ob sie SQL oder Funktionen als Schnittstelle bereitstellen (siehe Abbildung 2). Der Integrations-Server setzt sich aus zwei Komponenten zusammen: einem FDBS und einem Flow-System. Das FDBS übernimmt die Datenintegration, während das Flow-System als Komponente zur Integration von Funktionen (KIF) dient und dabei eine Ergebnistabelle (Daten) zurückliefert. Dazu kontrolliert es den Aufruf von vordefinierten lokalen Funktionen. Funktionsintegration in unserem Sinne stellt föderierte Funktionen bereit, welche die Funktionalität von einer oder mehreren lokalen Funktionen kombinieren [Hergula, Härder 2000].

Im Grunde könnte man spezielle Wrapper für den Zugriff auf jede der lokalen Funktionen einsetzen. Die lokalen Funktionen werden jedoch häufig zusammen aufgerufen, und zwar in der Form, dass die Ausgabe der einen Funktion als Eingabe einer weiteren Funktion genutzt wird. In diesem Fall könnte das FDBS die Ausführung jeder lokalen Funktion direkt steuern. Andererseits würde die Steuerung der Ausführungslogik im FDBS neben der Implementierung spezieller Wrapper zusätzlich beträchtliche Erweiterungen der FDBS-Komponente erfordern. Außerdem müsste das FDBS mit verschiedenen Anwendungen und ihren lokalen Funktionen umgehen können, die verteilt, heterogen und autonom sein können.

Als grundlegende Idee setzen wir daher einen Flow für die Ausführung der föderierten Funktionen ein. Der benötigte Flow muss einen automatisierten Prozess ohne Benutzerinteraktion umsetzen. Dabei führen die Flow-Aktivitäten die lokalen Funktionsaufrufe aus und das Flow-System kontrolliert die Parameterübergabe sowie die Abhängigkeiten zwischen den Funktionen. Auch hier müssen eine Art Adapter erstellt werden, die den Zugriff auf die heterogenen Quellen vereinheitlichen. Diese Adapter sollten erheblich einfacher vom Aufbau her sein als die Wrapper, da das Flow-System für den

Aufruf von Funktionen ausgelegt ist. Wrapper hingegen müssen die Abbildung von SQL-Befehlen auf Funktionsaufrufe umsetzen und sind somit komplexer. Darüber hinaus stellen Flow-Systeme bereits Adapter für viele Systeme bereit, während sich Wrapper auf relationale Schnittstellen konzentrieren.

Zur Isolation des FDBS von der Komplexität der föderierten Funktionen dient der Wrapper-Ansatz. Außerdem kann ein solcher Ansatz fehlende Funktionalität bereitstellen, um eine übergreifende Anfrageoptimierung zu unterstützen.

Wir haben uns für das Flow-System entschieden, da wir keine eigenen Server implementieren, sondern existierende Technologien einsetzen wollen. Das Basiskonzept der Flow-Technologie unterstützt unsere Funktionsabbildungen und ermöglicht die Umsetzung sehr komplexer Abbildungsszenarien [Hergula, Härder 2002]. Außerdem implementieren Flow-Systeme eine verteilte Programmierung über mehrere heterogene Anwendungen hinweg. Implementiert man die Funktionsintegration mit einem Software-Produkt, ist diese Lösung zudem einfacher zu verwalten.

FDBS und Flow-System werden über einen SQL/MED-konformen Wrapper verbunden [SQL 2002a], der aus zwei Teilen besteht: der eine Teil ist auf der FDBS-Seite platziert und stellt dort die SQL/MED-Schnittstelle bereit. Der zweite Teil befindet sich auf der Seite des Flow-Systems und kapselt die Flow-Schnittstelle. Somit wird die Kommunikation zwischen FDBS und Flow-System über die beiden Wrapper-Teile realisiert. Auf diese Weise stellt das Flow-System föderierte Funktionen zur Verfügung, die vom FDBS genutzt werden, um Anfragen über vielfältige externe Datenquellen ausführen zu können.

Die Anwendungen (Benutzer) greifen auf den Integrations-Server über die objekt-relationale Schnittstelle des FDBS zu und melden sich dort an. Die Anfrageverarbeitung des FDBS analysiert die Benutzeranfragen und reicht jene Teile an das Flow-System weiter, die föderierte Funktionen betreffen. Das Flow-System führt nun die Funktionsintegration aus, indem es die lokalen Funktionen in der spezifizierten Reihenfolge aufruft. Über den Wrapper liefert es das Ergebnis als abstrakte Tabelle an das FDBS zurück. Die verbleibenden Teile der Anfrage werden vom FDBS verarbeitet.

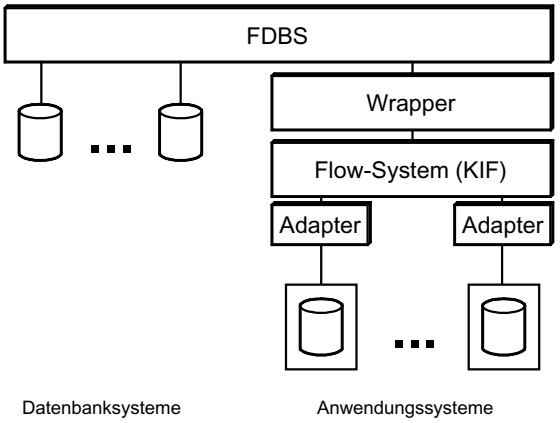


**Abb. 2: Schematische Darstellung der Integrationsarchitektur.**

Da die Probleme und Fragen der Anfrageverarbeitung im FDBS weitgehend erforscht sind, konzentrieren wir uns auf die neuen Aspekte der Anfrageverarbeitung unserer Architektur. Wie können Funktionsaufrufe nahtlos in die Anfrageverarbeitung des FDBS integriert werden? Der folgende Abschnitt geht detaillierter auf diese Frage ein, indem die Wrapper-basierte Verbindung von FDBS und Flow-System untersucht wird.

## 3 Anfrageverarbeitung

Wie kann eine Anfrageverarbeitung über Daten und Funktionen hinweg aussehen? In unserem Ansatz dienen abstrakte Tabellen als Schnittstelle zwischen Daten- und Funktionsverarbeitung. Die Abbildung zwischen Tabelle und Funktion erfolgt so, dass die Ein- und Ausgabeparameter die Spalten der Tabelle festlegen. Ein Funktionsaufruf entspricht dann exakt einer SQL-Anweisung, die alle Ausgabeparameter in der Select-Klausel aufführt und alle Eingabeparameter in der Where-Klausel konjunktiv verknüpft.

Ziel der Anfrageoptimierung ist die Verschiebung größtmöglicher Anteile der Verarbeitung der Daten an deren Quelle (das sog. *Pushdown*), um die Anzahl der zu transportierenden Ergebniszeilen zwischen den Systemen zu minimieren. Dazu führen wir zunächst eine Klassifikation möglicher Wrapper-Funktionalitäten ein, welche die Basis unserer weiteren Betrachtungen hinsichtlich Optimierung und Kostenmodell darstellt.

Unter *Kernfunktionalität* des Wrapper verstehen wir all jene grundlegende Funktionen, die zur Abbildung zwischen Tabellen und Funktionen bzw. SQL-Anweisungen und Funktionsaufrufen notwendig sind. Sie stellt die Minimalfunktionalität zur Verknüpfung der beiden Welten dar. Für jede Referenz auf eine abstrakte Tabelle innerhalb einer SQL-Anweisung wird deren Inhalt materialisiert und bereitgestellt. Dabei stellt der Wrapper keinerlei Funktionalität bereit, die über die Funktionalität des Flow-Systems hinausgeht. Dies ist die schlankeste Implementierung des Wrappers.

*Basisfunktionalität* erweitert die Funktionalität der lokalen Systeme, um die Anfrageoptimierung zu unterstützen. Hier ist vor allem die Reduzierung der zu übertragenden Datenmenge zwischen Flow-System und FDBS das Ziel. Die Basisfunktionalität beinhaltet die Projektion und Selektion auf eine Teilmenge der Spalten bzw. Zeilen der abstrakten Tabellen. Überdies sind Boolesche Kombinationen von Vergleichsprädikaten mit Konstanten möglich.

Die *erweiterte Funktionalität* beinhaltet mächtigere Operationen wie Gruppierung und Aggregation, Teilanfragen bzgl. abstrakter Tabellen und Mengenvergleiche.

Damit wir den Wrapper um die vorgestellten Funktionalitäten erweitern können, müssen zwei Voraussetzungen berücksichtigt werden:

- Die Kardinalität $K(T_a, i)$ jedes Eingabeparameters $i$ einer abstrakten Ta-

belle $T_a$ ist endlich.

- Die Elemente seines Wertebereiches $W(T_a, i)$ sind aufzählbar.

Wir beschreiben im Folgenden, wie die Basis- und erweiterte Funktionalität des Wrapper umgesetzt werden können. Wir erläutern diese anhand einer beispielhaften föderierten Funktion `GetBestand(IN LiefNr, IN KompNr, OUT Lager, OUT Order)`, die über das FDBS eingebunden wird. Sie liefert für eine gegebene Lieferanten- und Komponentennummer den aktuellen Lagerbestand und die Anzahl bestellter Komponenten zurück. Tabelle 1 enthält die verfügbaren Werte der lokalen Systeme.

## 3.1 Basisfunktionalität

Bei der *Selektion* unterscheiden wir zwischen Selektion auf Basis von Eingabe- und Ausgabeparametern. Spezifiziert man alle benötigten Eingabewerte in der Where-Klausel, reicht genau ein Funktionsaufruf, an dessen Ausgabewerte die korrespondierenden Eingabewerte konkateniert werden, um die abstrakte Tabelle aufzubauen. Werden zudem die Ausgabewerte in der Select-Klausel aufgeführt, so liefert der eine Funktionsaufruf das gewünschte Ergebnis ohne weitere Verarbeitung im Wrapper. Fehlt jedoch ein Eingabewert, müssen kompensierende Funktionsaufrufe durchgeführt werden.

**Tabelle 1: Abstrakte Tabelle GetBestand**

| LiefNr | Komp Nr | Lager | Order |
|---|---|---|---|
| 1 | 11 | 5 | 20 |
| 1 | 13 | 10 | 10 |
| 2 | 11 | 2 | 10 |
| 2 | 12 | 3 | 10 |
| 2 | 13 | 0 | 15 |
| 3 | 12 | 6 | 15 |
| 3 | 13 | 7 | 10 |

Für jedes Element der Wertemenge des fehlenden Eingabeattributs muss eine Funktion hinzugefügt werden. Die restlichen, bekannten Eingabewerte bleiben dieselben. Folglich wird die Anzahl der Aufrufe multipliziert mit der Kardinalität jedes unspezifizierten Eingabewertes. Das folgende Beispiel verdeutlicht diesen Aspekt.

***Beispiel**:*

Der SQL-Befehl

```
SELECT Lager, Order
FROM GetBestand
WHERE LiefNr=1
```

spezifiziert nur den Eingabewert von `LiefNr`, während `KompNr` unspezifiziert bleibt. Sei W=(GetBestand, KompNr)={11, 12, 13} beispielsweise die Wertemenge von `KompNr`, dann sind drei Funktionsaufrufe notwendig, um die SQL-Anweisung abzubilden:

`GetBestand(1,11,:Lager,:Order)`,
`GetBestand(1,12,:Lager,:Order)`,
`GetBestand(1,13,:Lager,:Order)`.

Allgemein ergibt sich die folgende Anzahl an kompensierenden Funktionsaufrufen für Selektionen auf Basis von Eingabeparametern (wobei die Menge $U$ alle unspezifizierten Eingabeparameter beschreibt):

$$fc_{in-sel} = \prod_{n \in U} K(T_a, n)$$

Erfolgt die Selektion ausschließlich auf Basis der Ausgabeparameter, müssen alle Daten der abstrakten Tabelle über Funktionsaufrufe wie bei einem Table Scan ausgelesen werden. Da keiner der Eingabewerte vorgegeben ist, müssen alle Kombinationen der Eingabeparameter ermittelt werden. Dies entspricht dem Produkt der Kardinalitäten aller Eingabeparameter. Anschließend wird im Wrapper die Selektion ausgeführt.

Bei der *Projektion* unterscheiden wir ebenfalls zwischen der Projektion auf Basis von Eingabe- und Ausgabeparameter. Erfolgt die Projektion über Eingabewerte, kann das Ergebnis in bestimmten Fällen ohne Funktionsaufruf ermittelt werden. Dazu prüft man, ob die spezifizierten Eingabewerte in der Wertemenge der Parameter enthalten sind. Trifft dies für einen der Parameter nicht zu, ist auch die Ergebnismenge leer.

Erfolgt die Projektion auf Basis der Ausgabeparameter, entspricht sie exakt der Ausgabe von Funktionen. Sind zudem alle Eingabewerte vorgegeben, ist genau ein Funktionsaufruf notwendig. Fehlen Eingabewerte, sind Funktionsaufrufe zur Kompensation zu ermitteln. Sollen nicht alle Ausgabewerte projiziert werden, muss der Wrapper die zurückgelieferte Ergebnismenge der Funktion um die überflüssigen Ausgabewerte kürzen.

***Beispiel**:*

Die SQL-Anweisung

```
SELECT Lager, Order
FROM GetBestand
WHERE LiefNr=2
```

projiziert exakt die Ausgabeparameter der Funktion. Somit sind lediglich kompensierende Funktionsaufrufe für den fehlenden Wert von `KompNr` notwendig:
`GetBestand(2,11,:Lager,:Order)`,
`GetBestand(2,12,:Lager,:Order)`,
`GetBestand(2,13,:Lager,:Order)`.
Da alle Funktionsaufrufe ein Wertepaar zurückliefern, sind diese drei Wertepaare das Ergebnis der SQL-Anweisung.

## 3.2 Erweiterte Funktionalität

*Gruppierung und Aggregation* stufen wir als erweiterte Funktionalität ein. Sie reduzieren im Allgemeinen die Ergebnismenge erheblich, so dass sie die Anfrageoptimierung maßgeblich beeinflussen können. Da die Betrachtungen zur Aggregation denen zur Projektion sehr ähneln, gehen wir nicht näher auf sie ein [Hergula 2003]. Für die Gruppierung untersuchen wir die Fälle ohne Where-Klausel, da wir die Selektion bereits beschrieben haben. Außerdem gehen wir davon aus, dass die Mengen der gruppierenden und aggregierenden Attribute disjunkt sind.

Bei der Gruppierung auf Basis von Eingabeparametern müssen unabhängig von der Anzahl spezifizierter Eingabewerte alle Daten mit den entsprechenden Funktionsaufrufen ausgelesen werden. Dies ist auch unabhängig davon, ob die Aggregation von Eingabe- oder Ausgabeparametern geschieht. Sobald Selektionen stattfinden, reduziert sich die Zahl kompensierender Funktionsaufrufe, da nicht alle Daten gelesen werden müssen.

Wird auf Basis der Ausgabeparameter gruppiert, müssen ebenfalls alle Daten basierend auf allen Kombinationen der Eingabewerte ausgelesen werden. Um die verschiedenen Gruppen zu identifizieren, muss die Ergebnismenge im Wrapper nach dem Ausgabeparameter

sortiert werden. Anschließend wird für jede Gruppe die Aggregation durchgeführt.

***Beispiel**:*

Die SQL-Anweisung aggregiert auf Basis von Eingabeparametern und gruppiert anhand eines Ausgabeparameters. Die Anweisung liefert die unterschiedlichen Lagerbestände und wie viele Lieferanten diesen Bestand haben:

```
SELECT Lager, COUNT(LiefNr)
FROM GetBestand
GROUP BY Lager
```

Dazu werden alle kompensierenden Funktionsaufrufe ausgeführt, die sich aus dem kartesischen Produkt der Wertemengen von `LiefNr` und `KompNr` ergeben. Nachdem alle Rückgabewerte für `Lager` sortiert wurden, zählt man die gleichen Bestände und bekommt das gewünschte Ergebnis. Bei Hinzufügen der Having-Klausel reduziert sich die Anzahl kompensierender Funktionsaufrufe nur, wenn sich die Having-Klausel auf Eingabeparameter bezieht.

Abschließend betrachten wir die Unterstützung von *Teilanfragen und Mengenvergleichen* im Wrapper. Dabei bezieht sich die Ausführung von Teilanfragen innerhalb des Wrapper ausschließlich auf abstrakte Tabellen. Außerdem dürfen die Anfragen nur Operationen enthalten, die der Wrapper zusammen mit den Funktionen unterstützt. Nur bei Einhaltung dieser Anforderungen kann der Wrapper die Anfrage vollständig ausführen und den Optimierungszweck erfüllen.

Wir unterscheiden bei den Teilanfragen zwischen korrelierten und unkorrelierten Teilanfragen. Im unkorrelierten Fall wird die Teilanfrage einmal ausgeführt und bringt somit keine neuen Aspekte mit sich. Daher fokussieren wir auf die korrelierte Teilanfrage, die für jede Zeile der äußeren Tabelle ausgewertet werden muss. Da die meisten Teilanfragen durch Prädikate zum Mengenvergleich wie IN oder EXISTS eingeleitet werden, sollte der Wrapper auch solche Prädikate unterstützen. So kann die Zahl der Wrapper-Aufrufe und auch die zu transportierende Datenmenge erheblich reduziert werden. Das folgende Beispiel veranschaulicht diese Aspekte.

***Beispiel**:*

Neben den bereits bekannten abstrakten Tabellen existiert eine Basistabelle im FDBS, die für bestimmte Lieferanten alternative Lieferanten nennt. Die folgende SQL-Anweisung ermittelt alternative Lieferanten für all jene Lieferanten, deren gelieferte Komponenten derzeit keinen Lagerbestand aufweisen und somit auf Lieferprobleme hinweisen:

```
SELECT Alternative
FROM GetLiefAlternative LA
WHERE 0 IN (
SELECT Lager
FROM GetBestand
WHERE LiefNr=LA.LiefNr
```

Beim ersten Wrapper-Aufruf werden alle Lieferantennummern der Basistabelle sowie der Operator IN mitgegeben. Dieser führt die Teilanfrage für jeden dieser Werte mit den entsprechenden Funktionsaufrufen aus. Alle Lieferantennummern, für welche die Bedingung zutrifft, werden an das FDBS zurückgeliefert.

Nachdem wir gezeigt haben, welche Funktionalität der Wrapper implementieren sollte und wie diese umgesetzt werden kann, gehen wir im nächsten Abschnitt auf ein angepasstes Kostenmodell ein.

## 3.3 Kostenmodell

Abschließend betrachten wir Aspekte des Kostenmodells, die durch die Hinzunahme von Funktionen hinzukommen. Dazu passen wir zunächst die traditionelle Kostenberechnung an und zeigen für ausgewählte Operationen, wie sich die Kosten für eine Ausführung im FDBS oder im Wrapper unterscheiden.

Im Allgemeinen enthalten die Kosten einer Anfrage die Kosten für die Kommunikationszeit, Prozessorzeit, Platten-E/A und Hauptspeicher. Überträgt man diese Aufteilung auf die vorgestellte Integrationslösung, stellen die Kosten für die Plattenzugriffe und die Kommunikation aufgrund der Verteilung der Architekturkomponenten den Löwenanteil bei der Anfrageausführung dar. Die existierenden Kostenmodelle für die heterogene Anfrageverarbeitung sind hinsichtlich der Kommunikationszeit auch für unseren Fall geeignet.

Zur Minimierung der Kommunikationszeit zwischen FDBS und Datenquelle sollte die zu transportierende Datenmenge möglichst stark reduziert werden. Dies wird in erster Linie durch das Verschieben möglichst vieler Operationen zur Datenquelle erreicht. Sitzt der Wrapper auf Seiten der Datenquelle und stellt dieser zusätzliche Funktionalität bereit, können die Operationen zum Wrapper verschoben werden.

Trotzdem muss die traditionelle Kostenberechnung für Tabellenzugriffe überarbeitet werden, wenn zukünftig die Kosten für Funktionsaufrufe berücksichtigt werden sollen. Bisher galt im Wesentlichen die folgende Systematik für die Kostenabschätzung einer Anfrage basierend auf Plattenzugriffen und Prozessorzeit [Selinger et. al. 1979]:

$$C = pagefetches + W * systemcalls$$

Diese Kostenformel beschreibt die erwarteten E/A- und CPU-Kosten für einen Anfrageplan. Diese Kostenanteile lassen sich über einen Faktor W gewichten, um bei der Bestimmung des kostengünstigsten Anfrageplans die Art der Auslastung des Systems berücksichtigen zu können. W kann dabei als das effektive Verhältnis des Aufwandes für die DBS-interne Verarbeitung zur Ein-/Ausgabe (beispielsweise in System R der Aufruf des Zugriffssystems zu einem Seitenzugriff) gewählt werden.

Für die Berechnung der Kosten über mehrere Komponenten hinweg gehen wir davon aus, dass die Systemaufrufe für eine Operationsausführung synchron nacheinander ausgeführt werden. Die Last in den Knoten und im Netz wird nur statisch berücksichtigt. Somit ergibt sich folgende Kostenformel, welche die Kosten in den Knoten und der Kommunikation addiert:

$$C = LocalCost + CommCost$$

Da die Anfrageverarbeitung über mehrere Komponenten und Knoten verteilt ist, präzisieren wir die Formel:

$$\begin{aligned} C = {} & LocalCost_{FDBS} \\ & + CommCost_{FDBS/Wrapper} \\ & + LocalCost_{Wrapper} \\ & + CommCost_{Wrapper/KIF} \\ & + LocalCost_{KIF} \end{aligned}$$

Wir gehen davon aus, dass der Wrapper zusammen mit dem Flow-System auf demselben Rechnerknoten liegt. Daher

können wir $CommCost_{Wrapper/Flow}$ vernachlässigen.

Als Kommunikationskosten setzt man die benötigte Zeit zur Übertragung einer Anzahl an Nachrichten bei einer bestimmten Bandbreite des Netzwerkes an: *CommCost = #Msg/Bandwidth*.

Die Kosten im FDBS ergeben sich aus Addition der Systemaufrufe und der E/A-Operationen, die synchron und asynchron anfallen können:

$$LocalCost_{FDBS} = Speed * \#SystCalls + w_1 * \#SynchI/O + w_2 * \#AsynchI/O$$

Dabei wird die Geschwindigkeit des Knotens gemessen oder rechnerspezifisch angesetzt. $w_1$ bzw. $w_2$ beschreiben den Transfer, wobei im synchronen Fall die Latenz hinzuzufügen ist. $w_2$ kann beispielsweise oft herangezogen werden, wenn ein Table Scan durch Prefetching unterstützt wird. Die Berechnung der lokalen Kosten setzt eine genaue Kenntnis des Operationsverhaltens voraus, um zu wissen, wann synchrone und asynchrone Operationen angesetzt werden müssen.

Für den Wrapper sind die Kosten ähnlich aufgebaut. Jedoch werden die Systemaufrufe den Hauptteil der Kosten verursachen, da der Wrapper E/A-Operationen nur zum Auslesen von Abbildungsinformationen benötigt.

Für die lokalen Kosten der KIF bzw. des Flow-Systems muss die Formel um die Anzahl der Aufrufe lokaler Funktionen erweitert werden:

$$LocalCost_{KIF} = Speed * \#SystCalls + w_1 * \#SynchI/O + w_2 * \#AsynchI/O + \#FuncCalls$$

Wobei sich *FuncCalls* wiederum aus mehreren *LocalCost* der integrierten Systeme und *CommCost* zusammensetzt.

Sind die Auswertungskosten in FDBS und Wrapper mit denen der KIF vergleichbar, so ist ein Pushdown von Operationen zum Wrapper vorzuziehen, um die Kommunikationskosten gering zu halten. Lediglich bei komplexen und teuren Prädikaten, für die Wrapper und KIF ungeeignet sind, sollten die Operationen im FDBS durchgeführt werden. Folglich besteht kaum eine Wahl bei den Ausführungsplänen.

Abschließend betrachten wir den Fall, dass das Netz stark belastet ist. Es gilt nach wie vor, dass die Operationen möglichst von Wrapper und KIF ausgeführt werden sollten. Doch was ergibt sich für die komplexen Prädikate, die eigentlich im FDBS verarbeitet werden sollten? Hier kann keine pauschale Aussage gemacht werden. Eine Verlagerung solcher Operationen in den Wrapper macht nur dann Sinn, wenn die geringeren Ausführungskosten im FDBS die Übertragungskosten der größeren Datenmenge nicht ausgleichen können. Dies ist wiederum stark von den Daten und Prädikaten abhängig. Bewirkt das Prädikat eine starke Reduzierung der ursprünglichen Datenmenge, dann kann es sinnvoll sein, auch die komplexen Prädikate im Wrapper durchzuführen. Wird die Datenmenge hingegen nur geringfügig reduziert, dann könnte sich nach wie vor die Ausführung im FDBS als günstiger erweisen.

Im Folgenden untersuchen wir, wie sich die Unterstützung zusätzlicher Funktionalität im Wrapper auf die Kosten auswirkt. Was kann an Operationen im FDBS und an zu transportierenden Datenvolumina eingespart werden, wenn die entsprechende Funktionalität im Wrapper bereitgestellt wird? Wir werden nicht die Kosten für die beiden Varianten explizit berechnen, sondern untersuchen, wie sich die folgenden Größen verändern: zu transportierende Datenmenge sowie Anzahl der Wrapper- und Funktionsaufrufe. Wir fokussieren auf diese Größen, da unseres Erachtens die Übertragung von Daten zwischen Wrapper und FDBS und die Anzahl der Funktionsaufrufe die Kosten stärker beeinflussen als die Rechenzeit von Wrapper und FDBS. Wir unterstellen dabei, dass die reinen Kosten für die einzelnen Operationen in Wrapper und FDBS in der gleichen Größenordnung liegen.

Bei der *Selektion* auf Basis von Eingabeparametern hat man keine Wahl des Ausführungsortes, denn diese Operation findet immer in der KIF statt. Dies ist durch unsere Definition der Abbildung von Funktionen auf Tabellen bereits vorgegeben.

Für eine Selektion auf Basis der Ausgabeparameter werden Unterschiede deutlich. Um diese Selektion durchführen zu können, müssen zunächst alle Ausgabewerte durch entsprechende Funktionsaufrufe ermittelt werden. Sind keine Eingabewerte vorgegeben, ist sogar ein Table Scan notwendig. Wird die Selektion nicht im Wrapper unterstützt, muss die vollständige Ergebnismenge zur Selektion an das FDBS übertragen werden. Lässt sich hingegen die Selektion im Wrapper durchführen, kann die zu übertragende Datenmenge in der Regel erheblich reduziert werden. Aus Sicht des FDBS wirkt sich eine Selektion im Wrapper folglich hauptsächlich auf die Einsparung von CPU-Kosten für die aktuelle Selektion und für die nachfolgenden Evaluationskosten aus, da diese erheblich durch die Größe und Sortierordnung der zurückgelieferten Ergebnismenge bestimmt werden.

Der Kostenvergleich für die *Projektion* verhält sich ähnlich wie bei der Selektion. Auch in diesem Fall bleiben die Wrapper- und Funktionsaufrufe dieselben, lediglich die zu übertragende Datenmenge und damit die Kommunikationskosten sind unterschiedlich. Kann der Wrapper die Projektion durchführen, müssen weniger Daten zum FDBS transportiert werden. Folglich werden die Gesamtkosten für die Verarbeitung der Anweisung geringer.

Die Ausführung von *Gruppierungen und Aggregationen* erfordert je nach optionaler Spezifikation von Selektionen mehrere oder gar alle kompensierende Funktionsaufrufe, um einen Table Scan durchzuführen. Werden Gruppierung und Aggregation erst im FDBS durchgeführt, muss die gesamte Datenmenge zum FDBS transportiert werden. Dabei entstehen die gleichen Kosten wie für Selektionen oder Projektionen.

Werden diese Operationen bereits im Wrapper durchgeführt, reduziert sich die zu übertragende Datenmenge deutlich. Außerdem sind weniger Funktionsaufrufe notwendig, wenn überdies die HAVING-Klausel im Wrapper unterstützt wird. Da die HAVING-Klausel einer Selektion entspricht, müssen nicht alle Daten über kompensierende Funktionsauf-

rufe ausgelesen werden.

Ein Beispiel soll dies verdeutlichen. Die folgende SQL-Anweisung ermittelt, von wie vielen Lieferanten eine Komponente geliefert wird:

```
SELECT COUNT(LiefNr), KompNr
FROM GetBestand
GROUP BY KompNr
```

Bei Durchführung der Gruppierung und Aggregation im FDBS fallen folgende Kosten an: 1 Wrapper-Aufruf, 7 Funktionsaufrufe für alle Kombinationen der Eingabewerte und 7*2 Werte für 7 Ergebniszeilen à 2 Spalten. Werden die Operationen dagegen im Wrapper ausgeführt, ergeben sich folgende Kosten: 1 Wrapper-Aufruf, 7 Funktionsaufrufe für alle Kombinationen der Eingabewerte und 3*2 Werte für 3 Ergebniszeilen à 2 Spalten. Folglich kann die Datenmenge von 14 auf 6 reduziert werden.

Wir erweitern nun das Beispiel um eine HAVING-Klausel:

```
SELECT COUNT(LiefNr), KompNr
FROM GetBestand
GROUP BY KompNr
HAVING KompNr<=12
```

Dabei ergeben sich folgende Kosten: 1 Wrapper-Aufruf, 4 Funktionsaufrufe für alle Kombinationen der Eingabewerte und 2*2 Werte für 2 Ergebniszeilen à 2 Spalten. In diesem Fall konnten weitere drei Funktionsaufrufe eingespart und die Kommunikationskosten nochmals um 2 Einheiten verringert werden.

Auch Gruppierung und Aggregation beeinflussen in erster Linie die Größe der zu übertragenden Datenmenge. Wird zudem die HAVING-Klausel genutzt, dann wird auch die Anzahl der Funktionsaufrufe verändert, da kein Table Scan notwendig ist.

Abschließend betrachten wir die Verarbeitung von *Teilanfragen und Mengenvergleichen*. Insbesondere der korrelierte Fall von Teilanfragen verursacht häufige Interaktionen zwischen FDBS und Wrapper bzw. KIF, da die Teilanfrage nicht auf einmal ausgeführt werden kann, sondern in mehreren Schritten aufgerufen werden muss. Folglich unterscheiden sich die Kosten zumindest in der Anzahl der Wrapper-Aufrufe. Unterstützt der Wrapper zusätzlich den Mengenvergleich, kann auch die zu übertragende Datenmenge reduziert werden, wie das nächste Beispiel zeigt. Die folgende SQL-Anweisung ermittelt die alternativen Lieferanten all jener Zulieferer, für die keine Komponenten auf Lager sind:

```
SELECT Alternative
FROM GetLiefAlternative LA
WHERE 0 IN (SELECT Lager
FROM GetBestand
WHERE LiefNr = LA.LiefNr)
```

Wenn das FDBS alle Operationen abwickelt, ist ein Table Scan von *GetBestand* durchzuführen. Es entstehen folgende Kosten: 1 Wrapper-Aufruf, 7 Funktionsaufrufe für alle Kombinationen der Eingabewerte und 7*4 Werte für 7 Ergebniszeilen à 4 Spalten.

Wird die Teilanfrage im Wrapper ausgeführt und der Mengenvergleich im FDBS, ergeben sich folgende Kosten: 4 Wrapper-Aufrufe, 7 Funktionsaufrufe für alle Kombinationen der Eingabewerte und 7*1 Werte für 7 Ergebniszeilen mit jeweils einer Spalte. In diesem Fall konnten die Kommunikationskosten zwar auf ein Viertel reduziert werden, im Gegenzug hat sich aber die Zahl der Wrapper-Aufrufe vervierfacht.

Wird auch der Mengenvergleich im Wrapper unterstützt, kommen wir zu folgenden Kosten: 1 Wrapper-Aufruf, 7 Funktionsaufrufe für alle Kombinationen der Eingabewerte und 1*2 Werte für 1 Ergebniszeile à 2 Spalten. Offensichtlich konnten die Wrapper-Aufrufe erneut reduziert und die Kommunikationskosten verringert werden.

Allgemein gilt für die Unterstützung von Teilanfragen und Mengenvergleichen im Wrapper, dass sich einerseits die Anzahl der Wrapper-Aufrufe deutlich erhöht, andererseits sich aber die Zahl der Funktionsaufrufe verringert. Zudem wird das Datenvolumen kleiner, da sich die Zahl der Ergebniszeilen verringert und diese durch Projektion kürzer sind.

Die Untersuchungen der zusätzlichen Wrapper-Funktionalität haben verdeutlicht, dass Operationen, die auf Seiten von KIF und Wrapper ausgeführt werden, die Kosten erheblich reduzieren können. Folglich ist die Implementierung der Basisfunktionalität auf jeden Fall gerechtfertigt. Nichtsdestotrotz zeigt sich, dass das Ziel, die komplette Funktionalität von SQL durch kompensierende Funktionsaufrufe unterstützen zu wollen, einige Nachteile mit sich bringt. Dies liegt vor allem an der Zahl der benötigten kompensierenden Funktionsaufrufe, die regelrecht explodieren kann. Ein kleines Rechenbeispiel verdeutlicht dies. Angenommen eine Funktion hat drei Eingabeparameter mit einer Kardinalität von lediglich zehn für jeden dieser Parameter. Wenn eine Selektion vorliegt, die den Wert für nur einen dieser Parameter spezifiziert, sind bereits 10*10=100 kompensierende Funktionsaufrufe notwendig, um die fehlenden Werte auszugleichen. Noch gravierender ist das Ergebnis, wenn gar kein Wert spezifiziert wird, denn dann erhält man sogar 10*10*10 = 1000 Funktionsaufrufe.

Daher macht die Betrachtung alternativer Einbindungsmöglichkeiten der föderierten Funktionen mittels abstrakter Tabellen Sinn, um die große Anzahl der Funktionsaufrufe zu verringern. Dies führt leider auch dazu, dass sich der Benutzer der Einschränkungen oder zumindest der besonderen Handhabung abstrakter Tabellen bewusst sein muss.

Eine Alternative könnte ein erweiterter Parser sein, der bei der Zergliederung der SQL-Anfrage feststellt, ob beispielsweise für alle Eingabeparameter Werte spezifiziert wurden. Ist dies nicht der Fall, so wird die Anfrage mit einer Fehlermeldung zurückgewiesen und der Benutzer auf die fehlenden Informationen aufmerksam gemacht. Auf diese Weise könnten alle kompensierenden Funktionsaufrufe aufgrund fehlender Eingabewerte vermieden werden. Ein anderer Ansatz verschiebt die Spezifikation der Eingabewerte von Vergleichsprädikaten zu benutzerdefinierten Funktionen in der WHERE-Klausel, die immer zusammen mit den referenzierten abstrakten Tabellen eingesetzt werden müssen. Mit Hilfe der benutzerdefinierten Funktionen können die benötigten Eingabewerte übergeben werden, so dass auch hier keine kompensierenden Funktionsaufrufe aufgrund fehlender Werte nötig sind. Als Nachteil dieser Vorgehensweise muss der Benutzer jedoch wissen, bei welchen Tabellen es sich um abstrakte Tabellen handelt und welche benutzerdefinierte Funktion er in diesem Zusammenhang einzusetzen hat. Die Entscheidung, wel-

che Alternative anzustreben ist, hängt sicherlich von den Systemen und den Benutzern ab.

## 4 Flow-Varianten

Die Flow-Komponente und ihre Ausführung der föderierten Funktionen nimmt einen großen Teil der Gesamtdauer der globalen Anfragen in Anspruch. Daher wollen wir im folgenden Abschnitt zwei Varianten von Flow-Systemen näher betrachten und vergleichen.

Den geläufigsten Ansatz für ein Flow-System stellen sicherlich Workflow-Systeme dar, die schon seit einigen Jahren insbesondere in der Prozessunterstützung eingesetzt werden. Solche Flow-Systeme sind jedoch ein sehr mächtiges Konzept mit entsprechendem Verwaltungsaufwand. Da für die Integration von Funktionen aber viele dieser Funktionalitäten nicht benötigt werden, sind andere Flow-Systeme interessant, die eher leichtgewichtig sind. Ein Beispiel hierfür sind die Microflows, die IBM in den Applikations-Server integriert hat [IBM 2002]. Im Folgenden stellen wir diese beiden Flow-Varianten gegenüber und zeigen ihre Unterschiede in Funktionalität und Performanz auf.

### 4.1 Work- und Microflows

Workflows und Workflow-Managementsysteme (WfMS) wurden ursprünglich zur rechnergestützten Durchführung betrieblicher Prozesse entwickelt. Sie steuern, überwachen und koordinieren Aktivitäten. Eine Aktivität entspricht hierbei entweder einer Benutzeraktion oder einer automatisch abzuwickelnden Funktion. Die Steuerung führt die Aktivitäten in der definierten Reihenfolge aus. Diese strukturierte Folge von Aktivitäten definiert man als Prozess.

Die grundlegende Idee des WfMS liegt in der Trennung von Anwendungsfunktionen und Prozesslogik. Ein WfMS lässt sich als eine Middleware auffassen, die Datenverwaltungsaufgaben von Anwendungsfunktionen übernimmt und den Daten- und Kontrollfluss in einer heterogenen und verteilten Umgebung sicherstellt. Ihre Aufgaben umfassen Netzwerkkommunikation, Datenkonvertierung, Anbindung von Datenquellen, Transaktionsunterstützung, dynamische Zuteilung von Aktivitäten und Überwachungsaufgaben. Ein Workflow spannt dabei drei Dimensionen auf – Prozesslogik, Organisation und Infrastruktur –, die durch Funktionalität bedient werden müssen.

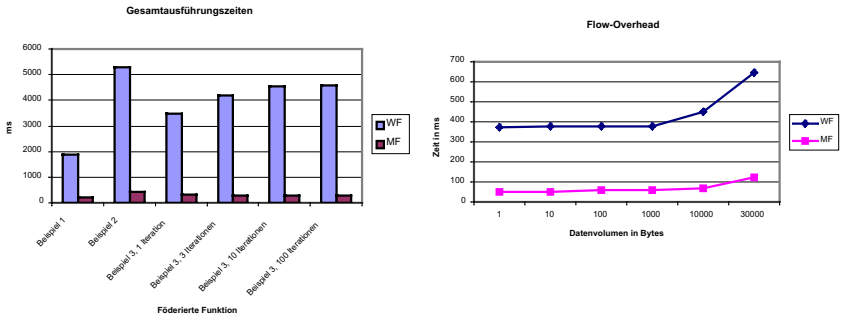


**Abb. 3: Messergebnisse für die implementierten Beispielanwendungen.**

Offensichtlich stellt ein WfMS deutlich mehr Funktionalität bereit, als wir für die reine Funktionsintegration benötigen. Eine mögliche Alternative stellen die sog. Microflows dar, die IBM in der Entwicklungsumgebung ihres Applikations-Servers anbietet. IBM definiert einen Microflow als einen leichtgewichtigen Workflow, der sich von diesem maßgeblich durch eine kurze, nicht unterbrechbare und zustandslose Abwicklung unterscheidet [IBM 2002]. Im Gegensatz zu Workflows gibt es für den Microflow die Aspekte einer Organisation und einer Infrastruktur nicht. Es wird nur die Prozesslogik modelliert, die existierende Funktionen zu einer neuen (föderierten) Funktion zusammenfügt. Da Microflows von IBM auf Web Services [W3C 2002] basieren, werden lokale Funktionen als Web Services bereitgestellt, die von den Aktivitäten des Microflow aufgerufen werden.

Microflows sind von kurzer Zeitdauer, da eine Benutzerinteraktion nicht vorgesehen ist, sondern nur der Service bzw. die Funktion aufgerufen wird. Nicht unterbrechbar bedeutet, dass der Microflow die Zustände *Suspended* und *Terminated* nicht kennt, sondern immer komplett durchläuft und anschließend das Ergebnis bekannt gibt. Es findet keine persistente Speicherung des Gesamtfortschritts eines Microflow statt, so dass eine transaktionale Ausführung nicht gewährleistet werden kann. Daher liegt eine zustandslose Abwicklung von Microflows vor. Hierbei wird der erforderliche Verwaltungsaufwand zugunsten einer effizienten Ausführung reduziert. Zusammenfassend stellen wir fest, dass Microflows wie gewünscht eine schlankere Lösung darstellen, dabei aber Funktionalitäten einsparen.

### 4.2 Testaufbau

Neben den funktionalen Unterschieden erwarten wir auch Unterschiede in der Ausführungsgeschwindigkeit von Flows und damit von föderierten Funktionen. Um diese herauszufinden, wurden föderierte Funktionen mit beiden Flow-Varianten implementiert und die Ergebnisse verglichen.

Die umgesetzten Architekturen entsprechen der in Abbildung 2 gezeigten allgemeinen Integrationsarchitektur. Anfragen gehen an das FDBS, das Referenzen auf föderierte Funktionen zum Flow-System weiterleitet. In unseren Implementierungen setzen wir DB2 als FDBS ein und koppeln das Flow-System über benutzerdefinierte Tabellenfunktionen, die in Java geschrieben sind [SQL 2002b], da es derzeit keinen SQL/MED-konformen Wrapper seitens der DB-Hersteller gibt.

Das Flow-System wird einmal mit Workflows (MQ Series Workflow von IBM) und einmal mit Microflows realisiert. Im Folgenden konzentrieren wir uns auf die Flow-Komponente der Architekturen, da die restlichen Komponenten identisch sind.

Die Workflow-Variante besteht aus vier Schichten. Die oberste Schicht bildet die Java-Tabellenfunktion, die in das FDBS eingebettet ist. Da aus Sicherheit-

saspekten und daraus resultierenden technischen Gründen der direkte Aufruf des WfMS aus der Tabellenfunktion heraus nicht möglich ist, führen wir den Workflow-Controller ein. Dieser kommuniziert mit der Tabellenfunktion mittels RMI und läuft in einer eigenen JVM. Auf diese Weise erfolgt die Entkopplung der Adressräume. In dieser Umgebung kann der Java-Agent des WfMS gestartet werden, der wiederum den angeforderten Workflow startet und zusammen mit dem WfMS die dritte Schicht bildet.

Ein eigenständiger Workflow-Controller hat den weiteren Vorteil, dass er eine Verbindung zum WfMS aufbaut und diese aufrecht erhält, so dass die Zeit hierfür nur einmal und nicht bei jedem Aufruf einer föderierten Funktion anfällt.

Für den Aufruf der lokalen Funktionen benötigt man als vierte Schicht Adapter. Diese werden vom WfMS gestartet und führen den Aufruf der lokalen Funktionen durch. Adapter sind API-spezifisch und werden entweder bereits mit dem WfMS ausgeliefert oder müssen für die lokalen Systeme speziell erstellt werden.

Auch die Microflow-Umsetzung besteht aus vier Schichten. Die oberste Schicht bildet erneut die Java-Tabellenfunktion im FDBS. Darüber hinaus benötigt man ebenfalls einen Controller, hier Microflow-Controller genannt, der jedoch nicht mit dem Workflow-Controller zu verwechseln ist. Der Microflow-Controller wurde aus Kompatibilitätsgründen eingefügt, um von Java-Version 1.1.8 von DB2 zur Version 1.3 des Applikations-Server zu kommen. Der Controller stellt daher eine Art Kommunikationsschicht auf Basis RMI dar und kommuniziert mit dem Applikations-Server über SOAP [W3C 2000]. Der RPC-Router und der Applikations-Server bilden die dritte Schicht der Microflow-Umsetzung. Hier leitet der RPC-Router die SOAP-Anfrage an die entsprechende Enterprise JavaBean weiter, die wiederum den Microflow startet.

Der Aufruf der lokalen Funktionen erfolgt über JavaBeans, die den Adaptern der Workflow-Lösung entsprechen. Sie sind ebenfalls eine spezifische Anbindung an die lokalen Systeme. Da sie innerhalb des Applikations-Servers ablaufen, nutzen sie den Vorteil, in einer bereits gestarteten JVM ausgeführt zu werden und damit schneller zu sein.

## 4.3 Leistungsmessungen

Zum Vergleich der vorgestellten Flow-Ansätze wurden mehrere föderierte Funktionen unterschiedlicher Heterogenitätsklassen implementiert [Hergula, Härder 2002]. Mit diesen Beispielen können wir die Leistungsfähigkeit sowie die Mächtigkeit der Funktionsabbildung beider Implementierungen bewerten. Darüber hinaus wurde ein Beispiel mit minimaler Laufzeit der Aktivitäten umgesetzt, um den Verwaltungsaufwand zu messen.

Aus Platzgründen beschreiben wir die einzelnen Beispiele nicht im Detail, sondern zeigen die wesentlichen Ergebnisse auf. Für eine genaue Beschreibung der Implementierung und der Messergebnisse verweisen wir auf [Steiss 2002].

Die Tests beschränken sich auf die Messung der Zeit, d. h., andere Ressourcen wie Speicherverbrauch und E/A-Operationen wurden nicht berücksichtigt. Da alle beteiligten Software-Komponenten lokal auf einem Rechner laufen, entfällt die Datenübertragung über ein Netzwerk. Einerseits würde eine verteilte Ausführung ein reales Einsatzgebiet besser widerspiegeln. Andererseits sind durch Ausschluss der Netzwerkbandbreite als limitierenden Faktor die Messwerte zur Abschätzung der Leistungsobergrenze auch im verteilten Fall geeignet.

Die lokalen Systeme und ihre Funktionen wurden durch Java-Programme nachgestellt, die vordefinierte Funktionen verkörpern. Die in SQL-Anweisungen aufzurufenden föderierten Funktionen basieren auf diesen lokalen Funktionen. Jede föderierte Funktion wird wiederum dem FDBS als eine Tabellenfunktion zur Verfügung gestellt.

Für beide Flow-Implementierungen wurden vergleichbare Messpunkte definiert, so dass nicht nur die Gesamtlaufzeit, sondern einzelne Phasen der Ausführung verglichen werden können. Wir betrachten nicht alle Phasen im Detail, sondern beschreiben auffällige Besonderheiten.

Unsere Untersuchungen zeigen, dass die Microflow-Implementierung für die durchgeführten Messungen der Funktionsbeispiele Gesamtausführungszeiten aufweist, die um den Faktor neun bis 17 kürzer sind als die der Workflow-Lösung (vgl. Abbildung 3, links). Die Microflows weisen bei fast allen Messpunkten die kürzeren Zeiten auf. Dieser beachtliche Unterschied in der Ausführungszeit ist auf die unterschiedliche Funktionalität der beiden Flow-Varianten zurückzuführen. Während das WfMS den Prozesszustand und den Datenfluss zwischen den Aktivitäten in einem persistenten Speicher sichert, spart sich die Microflow-Umsetzung diesen Aufwand und damit auch Zeit.

Überdies hat Java als eingesetzte Programmiersprache erheblichen Einfluss. Während das WfMS immer wieder die JVM neu starten muss, laufen die Beans der Microflows in einer bereits gestarteten JVM und sparen so erneut deutlich Zeit. Dies führt zur schnelleren Ausführung einer einzelnen Aktivität als auch des gesamten Workflows. Auch mit steigender Anzahl der Ausführungen derselben Aktivität stellen wir deutliche Vorteile seitens der Microflows fest. Schließlich folgt, dass der reine Verwaltungsaufwand beim WfMS um den Faktor sieben größer ist als bei der Microflow-Lösung (vgl. Abbildung 3, rechts). Mit steigendem Datenvolumen ist eine Änderung ab 1000 zu übertragenden Zeichen zu beobachten. Durch die relativ hohe Grundzeit eines WfMS-Aufrufes ist es ungünstig, viele Aufrufe mit kleinen Datenmengen durchzuführen. Stattdessen sollten möglichst wenig Workflow-Aufrufe mit großen Datenmengen angestrebt werden.

Übertragen wir diese Ergebnisse auf unsere übergreifende Anfrageverarbeitung, so ist offensichtlich, dass die Microflow-Variante eine schnellere Ausführung der föderierten Funktionen und damit der ganzen Anfrageverarbeitung unterstützt. Dem stehen jedoch fehlende Transaktionsunterstützung gegenüber, so dass eine verteilte Anfrageverarbeitung über Transaktionsgrenzen hinweg nicht möglich ist.

Bei der Wahl der Flow-Variante muss man daher entscheiden, welche Anforderungen die Flow-Komponente unterstützen soll. Erlaubt man nur lesenden Zugriff auf die integrierten Systeme und sollte dieser möglichst schnell sein, so kann der Einsatz der Microflow-Lösung durchaus Sinn machen. Möchte man hingegen auch schreibende Zugriffe und systemübergreifende Transaktionen unterstützen, so muss man derzeit auf die klassischen Workflow-System zurückgreifen.

## 5 Zusammenfassung

Die immer schneller wachsende Menge an Daten und Informationen in unterschiedlichsten Systemen zwingt Unternehmen zur Integration ihrer Systeme, um den besten Nutzen daraus zu ziehen. Da immer mehr Packaged Software die Daten verwaltet, ist eine reine Datenintegration nicht mehr ausreichend. Stattdessen müssen Daten und Funktionen integriert werden. Wir stellten hierzu eine Integrationsarchitektur auf Basis eines FDBS und Flow-Systems vor, in welcher das FDBS die Integration der Daten übernimmt und Flows zur Integration von Funktionen eingesetzt werden.

Eine besondere Herausforderung stellt die übergreifende Anfrageverarbeitung dar, welche die Abbildung zwischen einem relationalen Datenmodell und einem funktionalen Modell herstellen muss. Wir zeigten auf, wie eine Abbildung von SQL-Anweisungen auf Funktionsaufrufe realisiert werden kann. Zur Unterstützung der Anfrageoptimierung schlugen wir eine zusätzliche Funktionalität im Wrapper vor, der als Kopplungsmechanismus zwischen FDBS und Flow-System eingesetzt wird. Anhand eines angepassten Kostenmodells erläuterten wir die Auswirkung von Funktionsaufrufen und zeigten, dass die Wrapper-Funktionalität bei der übergreifenden Anfrageverarbeitung eine wichtige Rolle spielt.

Da Flow-Systeme sehr funktionsreiche Systeme mit beachtlichem Verwaltungsaufwand sind, untersuchten wir zwei Varianten von Flows, Work- und Microflows, hinsichtlich ihrer Performanz und Funktionalität. Es zeigt sich, dass abhängig von den Anforderungen beide Flow-Varianten Sinn machen.

## 6 Literatur


[Hergula, Härder 2000] *Hergula, K.; Härder, T.*: A Middleware Approach for Combining Heterogeneous Data Sources – Integration of Generic Queries and Predefined Function Access. Proc. 1st Int. Conf. on Web Information Systems and Engineering, Hongkong, S. 22-29, 2000.

[Hergula, Härder 2002] *Hergula, K.; Härder, T.*: Coupling of FDBS and WfMS for Integrating Database and Application Systems: Architecture, Complexity, Performance. Proc. 8th Int. Conf. on Extending Database Technology, Prag, S. 372-389, 2002.

[Hergula 2003] *Hergula, K.:* Daten- und Funktionsintegration durch Föderierte Datenbanksysteme (Arbeitstitel), Dissertationsschrift in Vorbereitung, 2003

[IBM 2002] *IBM*: WebSphere Studio Application Developer Integration Edition, Version 4.1.1, 2002.

[Jhingran et. al. 2002] *Jhingran, A.D.; Mattos, N.; Pirahesh, H.*: Information Integration: A Research Agenda. IBM Systems Journal 41:4, S. 555-562, 2002.

[Kim 1995] *Kim, A. (Hrsg)*: Modern Database Systems – The Object Model, Interoperability, and Beyond. Addison-Wesley, 1995.

[Leymann, Roller 2002] *Leymann, F.; Roller, D.*: Using Flows in Information Integration. IBM Systems Journal 41:4, S. 732-742, 2002.

[SAP 2003] *SAP*. Zu beziehen über www.mysap.com.

[Selinger et. al. 1979] *Selinger, P.; Astrahan M.; Chamberlin, D.; Lorie, R.; Price, T.*: Access Path Selection in a Relational Database System. Proc. ACM SIGMOD Conf. on Management of Data, Boston, S. 251-260, 1979.

[Sheth, Larson 1990] *Sheth, A.P.; Larson, J.A.*: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys 22:3, S. 183-236, 1990.

[SQL 2002a] *Final Committee Draft 9075-9:200x.*: Information Technology – Database Language – SQL – Part 9: Management of External Data (SQL/MED). Derzeit unter Abstimmung, zu beziehen über http://sql-standards.org/SC32/WG3/Progression_Documents/FCD/4FCD1-14-XML-2002-03.pdf, 2002.

[SQL 2002b] *Final Committee Draft 9075-2:200x*: Information Technology – Database Language – SQL – Part 2: Foundation, 2002.

[Steiss 2002] *Steiss, E*: Vergleich von Konzepten zur Kooperation zwischen Datenbank- und Workflow-Systemen. Studienarbeit, Universität Stuttgart, 2002.

[Varian, Lyman 2003] *Varian, H.; Lyman, P.*: How Much Information? Zu beziehen über http://www.sims.berkeley.edu/research/projects/how-much-info/.

[W3C 2000] *World Wide Web Consortium*: Simple Object Access Protocol (SOAP) 1.1, zu beziehen über http://www.w3.org/TR/SOAP, 2000.

[W3C 2002] *World Wide Web Consortium*: Web Services Activity, W3C Architecture Domain, zu beziehen über http://www.w3.org/2002/ws/, 2002.



Dipl.-Inf. Klaudia Hergula
DaimlerChrysler AG · IT Management
Data Management (TOS/TD) · HPC 096-0516
Epplestr. 225, D-70546 Stuttgart
klaudia.hergula@DaimlerChrysler.com

Prof. Dr.-Ing. Dr. h.c. Theo Härder
Technische Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
67653 Kaiserslautern
haerder@informatik.uni-kl.de