Adaptive Optimierung des Prefetch-Verhaltens bei objektorientierten Multi-Tier Client-Server-Systemen

Matthias Hofmann Arno Klein Gabriella Kókai
SCHEMA Electronic Documentation Solutions GmbH
Lehrstuhls für Programmiersysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg
{matthias.hofmann,arno.klein}@schema.de, kokai@informatik.uni-erlangen.de

Abstract

Prefetching – die vorsorgliche Übertragung in Zukunft wahrscheinlich benötigter Daten – stellt in verteilten Client-Server-Systemen ein wichtiges Konzept dar, um die Leistungsfähigkeit, Interaktivität und Performanz des Gesamtsystems zu verbessern. Gerade für Rich-Client-Anwendungen, bei denen die Applikationslogik in den Clients untergebracht ist, birgt ein gut an das System angepasstes Prefetch-Verhalten ein erhebliches Potential zur Steigerung der Gesamtleistung des Client-Server-Systems. Die Optimierung des Prefetch-Verhaltens eines Client-Server-Systems ist gerade bei Systemen mit wechselnder Datenmodellierung, speziellen Anpassungen für Kunden und verschiedenen Benutzerrollen sehr zeitraubend und schwierig. Ziel dieser Arbeit ist es daher, ein System zum automatisierten Erlernen einer angepassten Prefetch-Logik auf Basis der Content-Management-Plattform *SCHEMA ST4* zu entwickeln, das sich adaptiv sowohl an wechselnde Datenmodellierungen als auch an das individuelle Verhalten verschiedener Benutzer anpassen kann.

1 Einführung

Die Gesamtleistung eines Client-Server-Systems hängt wesentlich von der Effizienz der Kommunikation der beteiligten Komponenten untereinander ab. Unzureichender Datenfluss zwischen den Komponenten sorgt häufig dafür, dass in größeren Projekten die Leistung des Systems von Kunden und Benutzern schlecht bewertet wird. Verschlimmert wird die Situation dadurch, dass mit dem Aufkommen ausreichend schneller Internet-Verbindungen Teams in einem Client-Server-System oft nicht mehr lokal im Intranet, sondern weltweit zusammenarbeiten und sich damit Anforderungen an Systeme ergeben, die vor einigen Jahren noch nicht realisierbar waren. Daher ist es notwendig, effiziente und leistungsfähige Algorithmen und Architekturen zu entwickeln, die es ermöglichen, den Datenfluss zwischen den Komponenten zu verbessern.

Für die sogenannten *Thin-Client-Architekturen*, ist dies ein gut erforschtes Gebiet [Knafla, 1999][Padmanabhan and Mogul, 1996][Schäffer, 2003][Teng *et al.*, 2005]. *Thin-Clients* übernehmen dabei nur die Darstellung der Daten. Alle anderen Aufgaben wie die Applikationslogik, Datenmanagement und Datenhaltung werden von einer (*Two-Tier-Architektur*) oder mehreren Schichten *Multi-Tier-Architektur*) übernommen [Bengel, 2002][Coulouris *et al.*, 2002]. Diese Verfahren beruhen hauptsächlich auf intelligentem Caching und Prefetching derjenigen Ansichten, die durch Analyse des Nutzerverhaltens mit hoher Wahrscheinlichkeit als nächstes dargestellt werden müssen.

Rich-Clients als Konkurrenzmodell beinhalten neben der Präsentations- auch noch die Applikationslogik. In dieser Architektur-Art werden häufig Daten zwischen den Clients und der unterliegenden Serverschicht ausgetauscht und clientseitig verarbeitet. Hier besteht großes Potential in der Verbesserung des Datenflusses, da die vom Server angeforderten Daten nicht unkorrelliert angefragt werden, sondern sich anhand von Aktionen und Berechnungsschritten kategorisieren und vorhersagen lassen. Bei den meisten Frameworks – zum Beispiel *Java Data Objects* (*JDO*) oder *Enterprise Java Beans* (*EJB*) oder *SCHEMA ST4* – finden sich nur unzureichende Möglichkeiten, den Datenfluss flexibel und wenn möglich automatisiert zu optimieren.

Als Basis für die Untersuchungen dient die Content-Management-Plattform *SCHEMA ST4* der *SCHEMA Electronic Documentation Solutions GmbH*. Sie ist ein typischer Vertreter der Rich-Client-Architektur mit drei Schichten. Die Clients übernehmen Darstellung, Verarbeitungs- und Applikationslogik, während sich der Applikationsserver um das Datenmanagement und ein Datenbank-System um die Datenhaltung kümmern. *SCHEMA ST4* erlaubt es, Informationen medienneutral zu erfassen, zu organisieren und in verschiedene Ausgabeformate zu produzieren. Einsatzbereiche umfassen unter anderem Erstellung von technischen Dokumentationen, Katalogen, Verwaltung von Werkstattinformationen und Dokumentenmanagement. Die Interaktion des Benutzers mit dem Client findet mittels einer graphischen Benutzeroberfläche statt, die aus sogenannten *Viewlets* besteht. *Viewlets* sind andockbare Fenster mit eigenständiger Funktionalität, die individuell zusammengestellte und konfigurierbare Benutzeroberflächen ermöglichen.

Kennzeichen von *SCHEMA ST4* ist vor allem ein flexibles Daten- und Benutzungsmodell, das je nach Kundenwunsch in Funktionalität und Umfang angepasst werden kann. Dieses System macht eine Optimierung des Datenflusses zwischen den Komponenten wünschenswert, die einfach und flexibel anpassbar ist bzw. möglichst sogar automatisch und adaptiv den Datenfluss optimiert.

Abschnitt 2 beschreibt die grundlegenden Möglichkeiten zur Datenmodellierung sowie ein bereits vorhandenes Verfahren zur Errechung der Prefetch-Menge. Anschließend wird in Abschnitt 3 das neue Verfahren zur Lösung des Problems, das regelbasierte, adaptive Prefetching vorgestellt. In Abschnitt 4 wird die Leistung und Effizienz des Systems evaluiert und Abschnitt 5 bietet eine Diskussion der Resultate und einen Ausblick auf mögliche Erweiterungen.

2 Grundlagen

Ein großer Teil der Rechenzeit wird in der Regel mit der Kommunikation der beteiligten Schichten und Komponenten untereinander verbracht. Die konkrete Realisierung geeigneter und performanter Caching- und Prefetching-Algorithmen ist daher in großem Maße von der unterliegenden Systemarchitektur und den Wechselwirkungen zwischen den einzelnen Komponenten abhängig. Dieser Abschnitt beleuchtet zum einen die Abläufe und Datenstrukturen des Softwarepakets *SCHEMA ST4* und zum anderen *FetchExpressions*, ein serverseitiges System zur Errechnung der Prefetch-Menge, das bereits als Basislösung existierte.

2.1 Struktureller Aufbau der Daten in SCHEMA ST4

Die Optimierung der Prefetch-Daten soll durch Analyse der angefragten Daten in *SCHEMA ST4* eine Verbesserung des Datenflusses errechnen. Deshalb muss zunächst geklärt werden, welchen strukturellen Aufbau die Daten haben und welche Anfragemöglichkeiten existieren.

Datenstruktur

Die Daten besitzen in *SCHEMA ST4* eine objektorientierte Zugriffsstruktur. Eine Persistenzschicht sorgt für das Mapping der Objekte auf die relationale Datenbank. Jedes Objekt besitzt eine eindeutige ID, über die es systemweit identifizierbar und ansprechbar ist.

Abbildung 1 zeigt den primären Aufbau der Daten in *SCHEMA ST4*. *Knoten* (Typ: *Node*) sind durch gerichtete Kanten, sogenannte *Links*, miteinander verbunden. In der Regel werden bedeutungstragende Inhalte an Knoten gespeichert und mit Hilfe von Links zusammengestellt oder in Kontext gesetzt. Es gibt hierarchische Links (durchgezogene Linie) und nicht-hierarchische Links (gepunktete Linie). Alle Knoten sind über mehrere Ebenen hierarchisch und azyklisch verbunden. Zwischen Knoten können zudem über nicht-hierarchische Links semantische Beziehungen (auch zyklisch) hergestellt werden. Vom Knoten aus ergeben sich somit vier verschiedene Sichtweisen: der hierarchische Link zum Elternknoten (*ParentLink*), hierarchische Links zu den Kinderknoten (*ChildrenLinks*) und beliebige, nicht-hierarchische eingehende und ausgehende Links (*IncomingLinks* bzw. *OutgoingLinks*).

Sowohl an Knoten als auch an Links ist es möglich Datenwerte zu setzen. Zusätzliche Mächtigkeit verleiht den Datenwerten das Konzept der *Dimensionen*. Eine Dimension enthält *Aspekte*, die als zusätzliche Indirektionsebene zum eigentlichen Inhalt eines Datenwerts eingefügt worden ist. Ein Beispiel wäre eine Dimension 'Sprache', welche

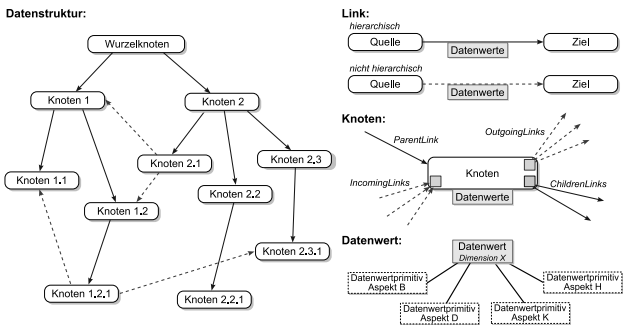

Abbildung 1: Schematische Darstellung der primären Datenstruktur in *SCHEMA ST4*

die Aspekte Deutsch, Englisch und Französisch enthält. Wird ein Datenwert in Abhängigkeit der Dimension 'Sprache' definiert, können drei verschiedene Datenwertprimitive existieren, in denen der Wert für die jeweilige Sprache enthalten ist. Wenn der Datenwert allerdings *dimensionsunabhängig* ist, gibt es nur ein Datenwertprimitiv, welches den Inhalt des Datenwerts enthält.

Informationsmodell

Neben der syntaktischen Objektstruktur ist in *SCHEMA ST4* ein semantisches Meta-Modell – das sogenannte *Informationsmodell* – vorhanden, welches eine Klassifikation und eine Festlegung von Beziehungen zwischen einzelnen Knoten, Links und Datenwerten ermöglicht.

Knoten sind je nach Bedeutung unterteilt in mehrere *Knotenklassen*, die per Vererbungshierarchie gebunden sind. Pro Knotenklasse können verschiedene Datenwerte registriert werden, die wiederum in Form von Knoten dieser Knotenklasse instanziierbar sind.

Auch Links sind je einer *Linkklasse* zugeteilt, die die semantische Bedeutung des Links zuweist. So sind innerhalb der beiden Grundformen (hierarchisch und nicht hierarchisch) feinere Unterschiede möglich. Genau wie an Knotenklassen können auch an Linkklassen Datenwerte registriert werden. Linkklassen besitzen sogenannte *Linkklassenprimitive*. Diese legen fest, welcher Klasse der Quell- und der Zielknoten angehören muss, damit der Link gezogen werden darf und ob er ein Hierarchie-Link ist.

Die Modellierungsmöglichkeiten von Datenwerten machen sich in zweierlei Hinsicht bemerkbar. Zum einen besitzen Datenwerte einen Typ, der bestimmt, welcher Datentyp in den Datenwerten gespeichert werden kann. In *SCHEMA ST4* sind folgende Typen möglich: *DateTime*, *Decimal*, *Dictionary*, *Double*, *Int32* und *String* verhalten sich analog zum .NET-Gegenstück [Liberty, 2003]. Dazu kommen noch *History* (versionierter Wert), *Label* (Versionsmarkierung), *Selection* (Enumeration), *Stream* (binärer Datenstrom), *Text* (streamingfähige Zeichenkette) und *XML* (XML-Fragment). Zum anderen sind Datenwerte gekennzeichnet durch *Datenklassen*, in denen festgelegt sind, an welcher Knoten- bzw. Linkklasse mit welchem Namen der Datenwert ansprechbar ist. Ein Datenwert kann zudem mit einem anderen Datenwert derselben Datenklasse verknüpft werden. Das bedeutet, dass die zugehörigen Datenwertprimitive die des verknüpften Datenwerts sind.

Anfragesprachen: StPath und STT

Zur Vereinfachung der Anfrage an die Objektstruktur und zur Navigation auf den Objekten selbst existiert in *SCHEMA ST4* eine spezielle Anfragesprache: *StPath*. Diese orientiert sich sehr stark an *XPath 1.0* [Consortium, 1999a] für XML. An die Stelle des „XML Document Object Model (DOM)" führt *StPath* Anfragen auf dem Objektmodell von *SCHEMA ST4* aus. Neben der *XPath*-ähnlichen Funktionalität gibt es allerdings in *StPath* zusätzlich noch spezifische Erweiterungen für das ST4 Objektmodell, zum Beispiel die Bestimmung der Klasse eines Knotens oder die Navigation über ausgehende oder eingehende Links.

Ein einfache Anfrage kann anhand von Abbildung 2 erläutert werden. Die Modellierung in *SCHEMA ST4* links entspricht dem XML-Auschnitt auf der rechten Seite. Knoten werden in XML zu Elementen, Datenwerte werden Attributen gleichgesetzt. Die Anfrage

```
child::B/@data
```

selektiert in *SCHEMA ST4* erst alle Kindknoten vom Typ *B* (des aktuellen Kontextknotens) und gibt den

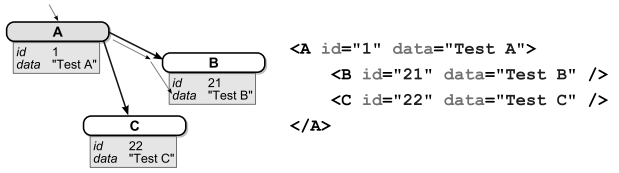

Abbildung 2: Wirkungsweise von StPath

Datenwert *content* der selektierten Knotenmenge aus. Analog werden in XML alle Kinder mit dem Elementnamen *B* gesucht und deren Attribute mit Namen *content* ausgegeben – in diesem Beispiel die Zeichenkette „Test B“.

Weiterhin erlaubt das analog zu *XSLT 1.0* [Consortium, 1999b] aufgebaute *STT* komplexe zusammengesetzte Transformationen auf der Objektstruktur. Die Mächtigkeit der Ausgabe-Operationen von *STT* und *XSLT* sind in etwa vergleichbar. Zudem kann sowohl StPath als auch STT über in C# geschriebene Funktionen erweitert werden. Somit werden zum Beispiel auch komplexe, performante oder systemnahe Funktionalitäten möglich.

2.2 Vorhandenes Verfahren: FetchExpressions

In *SCHEMA ST4* existiert bereits ein Verfahren, welches es erlaubt, das Prefetch-Verhalten der Clients zu manipulieren. Dieses wird im folgenden Abschnitt vorgestellt.

Prinzip

Das *FetchExpression-System* ist ein serverbasiertes Verfahren zur Berechnung der durch Prefetching geholten Objektmenge. Die Clients teilen dem Server nur mit, welches Objekt sie aktuell in welcher Version vom Server geliefert bekommen wollen. Kommt eine Objekt-Anforderung beim Server an, so wird diese kategorisiert. Für jede registrierte Kategorie existieren *StPath*-Ausdrücke, die bestimmen, welche Objekte zusätzlich als Prefetch-Menge dem Client mitgeliefert werden sollen.

Das FetchExpression-System kennt zwei Strukturierungsebenen, um Anfragen in Kategorien aufzuteilen. Zum einen können *Gültigkeitsbereiche* festgelegt werden, sogenannte *(FetchExpression-)Scopes*, denn je nach durchgeführter Aktion werden bestimmte, eventuell unterschiedliche Daten, die vom angefragten Objekt abhängen, benötigt. Der Client teilt dem Server bei Anfrage immer mit, in welchem Gültigkeitsbereich er die Anfrage stellt.

Zum anderen existieren innerhalb der Gültigkeitsbereiche Regeln, die bestimmen, welche Objekte genau geholt werden sollen, die eigentlichen *FetchExpression*s. Diese funktionieren nach folgendem Prinzip: Erfolgt eine Anfrage an ein bestimmtes Objekt, wird versucht, dieses Objekt einer FetchExpression zuzuweisen. FetchExpressions können entweder auf den Typ oder auf die Klasse des Objekts reagieren. Die erste Regel, die zur Anfrage passt, wird mit dem angeforderten Objekt als Kontext ausgewertet. Alle Objekte, die der Server zur Auswertung des Ausdrucks benötigt, werden dem Client als Prefetch-Menge mit übermittelt.

Das FetchExpression-System ist eine proprietäre Technologie von *SCHEMA ST4*. Allerdings gibt es auch andere, teils standardisierte Systeme, die auf demselben Prinzip beruhen. Im kommenden Standard *JSR 243: Java Data Objects 2.0 (JDO)* [Process, 2005] zum Beispiel, das bei Erstellung dieser Arbeit als *Proposed Final Draft* vorlag, gibt es sogenannte *FetchPlans*. Diese erlauben es analog zu Gültigkeitsbereichen, je nach Anwendungsfall einen sogenannten *FetchPlan* festzulegen, der definiert, welche Daten bei Anforderung eines bestimmten Objekts zurückgeliefert werden müssen.

Probleme und Anforderungen

FetchExpressions werden in *SCHEMA ST4* bereits seit mehreren Jahren eingesetzt, um die Gesamtleistung des Client-Server-Systems zu erhöhen. Trotz der prinzipiellen Möglichkeiten, das Prefetch-Verhalten der Clients zu steuern, gibt es Einschränkungen, die eine grundlegende Verbesserung des Systems wünschenswert machen:

1. FetchExpressions werden zentral auf dem Server hinterlegt und gelten gleichermaßen für alle Clients. Es können weder Benutzerollen noch Benutzerverhalten noch Ausbaustufen der Clients hinsichtlich der Funktionalität berücksichtigt werden.
2. Zeitlich variable Anfragestrukturen können von FetchExpressions nicht abgebildet werden. Vor allem durch Änderungen an der Benutzeroberfläche – zum Beispiel das Hinzu- oder Wegnehmen von Viewlets – kann sich das Anfrageverhalten aber sehr stark ändern. Mit FetchExpressions kann darauf nicht oder nur sehr eingeschränkt reagiert werden.
3. Die manuelle Erstellung und Optimierung der FetchExpressions gestaltet sich aus zwei Gründen schwierig: Zum einen sind die Implikationen bei Änderungen an den Regeln vielschichtig und erfordern viel Spezialwissen um die Struktur und den Aufbau der Modellierung. Zum anderen ist es nicht einfach, mögliche Stellen für die Optimierung ausfindig zu machen, sofern schon eine Basisoptimierung erfolgt ist. Hierfür ist es bis jetzt notwendig, lange Log-Dateien durchzuschauen und durch Ausprobieren und Erfahrung die richtige Stelle zu finden.

Ein besseres System zur Berechnung der Prefetch-Menge sollte also nach Möglichkeit alle diese Probleme beseitigen, indem eine Optimierung per Client stattfindet, die sich adaptiv an die jeweiligen Gegebenheiten und Anfragestrukturen anpassen kann. Daraus ergibt sich insbesondere, dass ein solches System fähig sein muss, die Daten in Echtzeit ohne relevanten Einfluss auf den Betrieb des Clients zu verarbeiten und verbesserte Optimierungsregeln daraus abzuleiten. Hinsichtlich der Qualität des Prefetchings muss sich das neue, automatische System auch in etwa mit den handoptimierten FetchExpressions messen können.

3 Regelbasiertes, adaptives Prefetching (RBAP)

RBAP ist ein Prefetch-Verfahren, welches eine optimale Menge an Prefetch-Objekten durch Kombination von client- und serverseitigen Modulen errechnet. Da die Applikationslogik bei Rich-Client-Anwendungen im Client untergebracht ist, sind nur dort alle notwendigen Informationen vorhanden, die das Verfahren benötigt, um eine optimale Prefetch-Menge zu berechnen. Im Client kann bestimmt werden, welche Objekte in welcher Version mit welchem Gültigkeitsbereich angefragt wurden, welche Objekte vom Server geliefert wurden und welchen Cache-Zustand ein Objekt bei Anfrage hat. Mögliche Zustände sind *Cache-Hit* (Objekt ist im Cache vorhanden), *Cache-Miss* (Objekt ist nicht im Cache und muss vom Server geholt werden) und *Cache-Prefetch* (Objekt wurde aufgrund einer Anfrage als Prefetch-Objekt mitgeliefert).

Abbildung 3 stellt den schematischen Aufbau von *RBAP* dar. Die Optimierung des Prefetch-Verhaltens erfolgt durch ein *Lernmodul* im Client, welches die relevanten Informationen sammelt und auswertet. Anhand dieser Datenbasis kann das Lernmodul eine Verbesserung des aktuell gültigen *Prefetch-Modells* vornehmen. Dieses umfasst alle Daten, die das *Auswertungsmodul* im Server benötigt, um auf eine Anfrage vom Client die passende Objektmenge zu berechnen und zurückzuliefern. Sollte das Lernmodul feststellen, dass sein Prefetch-Modell verbessert werden kann, wird sowohl sein eigenes als auch das korrespondierende Modell im Server angepasst.

3.1 Lernprozess

Der Lernprozess ist ständig als Hintergrund-Prozess im Client aktiv und muss schnell anhand der aktuellen Datenlage Entscheidungen treffen. Daraus ergibt sich eine wichtige Einschränkung: Das Verfahren und die Algorithmen müssen so gestaltet sein, dass der Lernprozess nicht zu viel Rechenzeit und Hauptspeicher verbraucht. Somit ist es nur schwer möglich etablierte Standard-Verfahren einzusetzen. Statt dessen kommt für *RBAP* eine Kombination aus regel- und schwellwertbasiertem Verfahren zum Einsatz, das in zwei Phasen eine Optimierung des Prefetch-Modells unter Beachtung der Beschränkungen vornehmen kann.

Knoten als grundlegender Optimierungsblock
Die wichtigen bedeutungstragenden Einheiten in *SCHEMA ST4* sind Knoten und Links zusammen mit den an beiden Objekttypen vorhandenen Datenwerten und Datenwertprimitiven. Alle anderen Typen stellen *Verwaltungsinformationen* dar. Die für den Lernprozess relevanten Daten sind daher hauptsächlich Knoten, Linkklassen und Datenwerte (mit ihren Datenwertprimitiven).

Für den Lernprozess ist es aus Gründen der Datenreduktion sinnvoll, eine Einheit zusammengehöriger Objekte zu definieren, den sogenannten *Optimierungsblock*. Um alle wichtigen Elemente aufzunehmen, gibt es zwei sinnvolle Möglichkeiten, einen Block darzustellen:

- *Knotenbasierter Optimierungsblock*
  Zum Optimierungsblock gehören alle Objekte, die von einem Knoten ausgehend erreichbar sind, ohne dass ein anderer Knoten traversiert werden muss.
- *Linkbasierter Optimierungsblock*
  Zum Optimierungsblock gehören alle Objekte, die von einem Link ausgehend erreichbar sind, ohne dass ein anderer Link traversiert werden muss.

Für FetchExpressions wird hauptsächlich in knotenbasierter Ansatz verwendet. Nur in wenige Fällen, für die eine knotenbasierte Herangehensweise keine zufriedenstellenden Ergebnisse geliefert hat, ist durch einen linkbasierten Ansatz teilweise eine Verbesserung erzielt worden. RBAP verwendet einen knotenbasierten Optimierungsblock aus zweierlei Gründen: Zum einen gestaltet sich Phase 1 (Registrierung) eindeutiger und damit einfacher. Ein Knoten (unterschieden nach Knotenklasse) kann Unterstruktur von mehreren Links (unterschieden nach Linkklasse) sein, je nach Anzahl der Linkklassenprimitiven, die diesen Knoten als Quelle oder Ziel referenzieren. Ist der Link jedoch dem Knoten untergeordnet, dann gibt es für den Link nur zwei Zuordnungsmöglichkeiten – die Quelle oder das Ziel. Zum anderen wird in der Verwendung der linkbasierten Herangehensweise kein offensichtlicher Vorteil gesehen. Zudem gestaltet sich die Vorstellung und das Nachvollziehen der Struktur anhand der eher auf Knotenvisualisierung ausgelegten Oberfläche von *SCHEMA ST4* einfacher.

Abbildung der Anfragestruktur
Die Optimierung nur mit Knoten verbundener Daten beschränkt die mehr als 100 vorhandenen Typen in *SCHEMA ST4* auf etwa 40, die wirklich für die Optimierung relevant sind – hauptsächlich Datenwerttypen, Knoten und Links. Die übrigen Objekttypen legen entweder Verwaltungsinformationen oder Strukturen des Informationsmodells fest.

In Abbildung 4 ist links die Datenstruktur für einen Optimierungsblock dargestellt, wie sie für die Realisierung von *RBAP* im Rahmen dieser Arbeit entwickelt worden ist. Der *NodeOptimizer* ist der Einstiegspunkt in den Optimierungsblock. Er beinhaltet allerdings keine weitere Optimierungslogik, sondern wird für die Optimierung nur traversiert, da der Knoten selbst nicht aus dem Prefetch-Modell entfernt werden kann. Allerdings kümmert sich der *NodeOptimizer* um alle Registrierungen am Optimierungsblock und verteilt die Registrierung weiter auf die richtigen Unterstrukturen. Die Behandlung von Datenwerten übernimmt der *DataValuesOptimizer*. Anfragen an Datenwerte werden kategorisiert über die ID ihrer Datenklasse. Für Links ist ein mehrstufiger Prozess notwendig. Eine Kategorisierung erfolgt zunächst nach ihrem Typ (*LinksOptimizer*) und dann nach ihrer Linkklasse (*LinkClassDictionary* in *LinkOptimizer*). Dies ist notwendig um eine ausreichend mächtige und zugleich flexible Handhabung von Links zu ermöglichen.

Auf der rechten Seite von Abbildung 4 ist das dem Optimierungblock entsprechende Datenlayout im *NodePrototype* aufgezeigt. Datenwerte werden sowohl an Knoten als auch an Links als eine Liste von IDs (*Int64*) repräsentiert. Das Bitfeld *_linkPolicy* fasst die Navigationsmöglichkeiten für Links zusammen. Entweder es werden nur Daten vom Link-Container abgefragt (z.B. die Anzahl der ausgehenden Links) oder es werden alle Links abgefragt und gegebenenfalls gefiltert (z.B. alle ausgehenden Links der Linkklasse *Informationshierarchie*). Im ersten Fall muss nur der Link-Container, im zweiten Fall der Link-Container mit al-

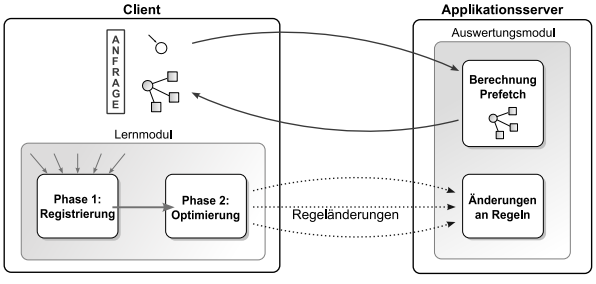

Abbildung 3: Schematische Übersicht der client- und serverseitigen Module von *RBAP*

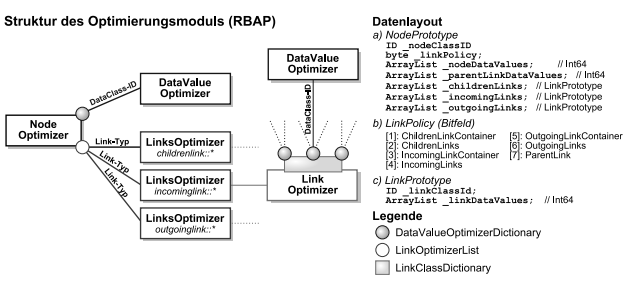

Abbildung 4: *Links*: Detaillierter Aufbau des Kernmoduls der Optimierung. *Rechts*: Datenstruktur im Prototyp

len Links geholt werden. Weitere Möglichkeiten sind navigationstechnisch für Links nicht vorgesehen.

PHASE 1: Registrierung der Objekte
Jede Objekt-Anfrage wird zunächst vom Lernmodul im Client registriert. Das heißt das jedes Objekt seinem zugehörigen Optimierungsblock zugewiesen und die ID sowie der Cache-Zustand an geeigneter Stelle im Optimierungsmodul festgehalten wird. Zum Zweck der Datenreduktion werden Knoten nach Knotenklasse, Links nach Linkklassen und Datenwerte nach Datenklassen kategorisiert.

Für untergeordnete Strukturen (Datenwerte und Links) wird die Anzahl der Verwendungen am Knoten eingetragen (unterhalb der Struktur für die zugehörige Klasse). Für einen Cache-Hit wird die Anzahl der Verwendungen um *1* erhöht, ein Cache-Miss setzt die Zahl der Verwendungen für das Objekt auf *1* ein Cache-Prefetch auf *0*. Somit ist es möglich, herauszufinden, wie viele Objekte einer bestimmten Klasse pro Optimierungseinheit angefragt wurden und ob angefragte Objekte einer bestimmten Klasse verwendet oder nur per Prefetch geholt worden sind. Anhand dieser Informationen entscheidet Phase 2, ob das Prefetch-Modell verbessert werden kann. Wenn genügend Objekte in mindestens einem Optimierungsblock erfasst und verarbeitet worden sind, wird ein Lernprozess gestartet, der den Prefetch-Vorgang optimieren soll.

PHASE 2: Verbessern des Prefetch-Modells
Phase 2 beschreibt den eigentlichen Lernprozess. Sämtliche vorhandenen, untergeordneten Strukturen des Optimierungsblocks werden traversiert. Pro Struktur wird entschieden, ob sie ins Prefetch-Modell aufgenommen bzw. entfernt wird.

1. *Struktur ist nicht im Prefetch-Modell vorhanden*
   Die Struktur kann dem Prefetch-Modell hinzugefügt werden, wenn die Anzahl der angefragten Objekte, die dieser Struktur zugeordnet sind, einen bestimmten Schwellwert überschreitet. Der Schwellwert richtet sich nach der Anzahl der Anfragen an den übergeordneten Knoten und nach der Anzahl der verfügbaren Instanzen der Knotenklasse.
2. *Struktur ist im Prefetch-Modell vorhanden*
   Die Struktur kann aus dem Prefetch-Modell entfernt werden, wenn gewisse Zeit nach der ersten Registrierung des Objekts kein oder nur wenige Objekte der Struktur verwendet worden sind. Als Entscheidungskriterium dient ein Schwellwert, der sich an der Anzahl der Anfragen an den übergeordneten Knoten orientiert.

Änderungen am Optimierungsblock werden dem Server mitgeteilt, damit dieser künftig anhand des verbesserten Prefetch-Modells Objekte an den Client zurückliefern kann. Das Prefetch-Modell wird also angepasst, sobald genügend Daten vorhanden sind, die eine Optimierung ermöglichen. Sofern sich das Benutzerverhalten oder das Anfrageverhalten (z.B. durch Hinzufügen von Viewlets) nicht wesentlich ändert, konvergiert das Prefetch-Modell mit steigender Zahl der Optimierungsläufe gegen das aufgrund der Datenbasis bestmögliche Prefetch-Modell.

Generationen: Modellierung temporaler Eigenschaften
Für die effiziente Abarbeitung der Optimierung des Prefetch-Modells ist es wichtig, dass nur eine begrenzte Anzahl von Daten vorliegen. Die Verarbeitung zu langer Anfrageprotokolle nimmt zu viel Zeit in Anspruch. Außerdem ist die Modellierung eines zeitlichen Verhaltens für die Optimierungsphase wichtig, denn der Lernprozess muss entscheiden, ob bestimmte Strukturen gebraucht wurden oder nicht.

Zur Lösung dieser Probleme werden sogenannte *Generationen* eingeführt. Generationen gelten unabhängig pro Optimierungsblock. In die nächste Generation wird vorgerückt, sobald sich eine Regel im Optimierungsblock ändert oder eine bestimmte Zahl von Optimierungen an diesem Knoten versucht wurden. Es wird also mit Hilfe der für die Optimierung verfügbaren Daten eine zeitliche Struktur modelliert, da ein Optimierungslauf in einer Optimierungseinheit abhängig ist von der Zahl der Informationen für eine Optimierung. Aus Gründen der Datenreduktion gibt es eine maximale Anzahl von Generationen die vorgehalten werden. Im Rahmen der Tests von *RBAP* hat sich eine Zahl von maximal sechs Generationen als günstig in Hinsicht auf Geschwindigkeit und Speicherverbrauch erwiesen.

3.2 Auswertung der Daten im Server

Die Aufgabe des Applikationsservers ist es, auf eine Anfrage des Clients passende Objekte zurückzuliefern. Es muss auf jeden Fall immer das angefragte Objekt in der richtigen Version vom Server mitgeliefert werden, weil ansonsten der Client einen Fehler an der Remoting-Verbindung meldet und die Verbindung zum Server abbricht. Welche weiteren Objekte als Prefetch-Menge übertragen werden, bestimmen sowohl das für die Sitzung gültige Prefetch-Modell als auch eine Menge von Standardregeln.

Zur Auswertung einer Anfrage wird im Server in der Regel – sofern Knoten angefragt werden – das aktuell gültige Prefetch-Modell herangezogen. Im Modell wird nach einer passenden Prefetch-Regel für den Knoten gesucht und ausgewertet. Dadurch wird festgelegt, welche Objektmenge zusätzlich an den Client übertragen werden soll. Bei der Auswertung der Regeln orientiert sich das Server-Modul an den Navigationsstrukturen, die *StPath* vorgibt. Die Prefetch-Regel soll idealerweise für ein Objekt *O* all diejenigen Objekte zurückliefern, die ein *StPath*-Ausdruck benötigt, um eine Anfrage an das Objekt *O* erfolgreich auszuführen.

Zusätzlich wird diese Menge noch gefiltert, um Informationen die sich in der Regel im Client-Cache befinden (zum Beispiel Informationen über Knotenklassen) nur beim ersten Aufruf bzw. nicht mehrfach zu übertragen.

Als Beispiel dient die Abfrage eines Datenwerts an einem Knoten. In diesem Fall muss neben dem Datenwert selbst und dem Datenwertprimitiv im eingestellten Aspekt auch noch die Datenklasse und der dazugehörige Rechte-Container zurückgeliefert werden. Zudem müssen noch alle Objekte berücksichtigt werden, die zur Evaluierung notwendig sind. Zu jedem Objekt muss ein vollständiger Pfad vom Ausgangsobjekt aus verfügbar sein. Im Fall eines Datenwerts muss deswegen zusätzlich der Datenwert-Container am Knoten zur Prefetch-Menge hinzugefügt werden.

Standardregeln
Das Standardregelwerk sorgt dafür, dass für alle angefragten Objekte, die keine Knoten sind, eine sinnvolle Objektmenge an den Client übertragen wird. Die einfachste Realisierungsmöglichkeit besteht darin, nur das angefragte Objekt selbst zurückzuliefern. Allerdings ist dies von Seiten

	Anfragen	Objekte
Unoptimiert	884	884
FetchExpressions	239 (≈ 0.27)	2324 (≈ 2.63)
RBAP (initial)	322 (≈ 0.36)	1612 (≈ 1.82)
RBAP (optimal)	292 (≈ 0.33)	1578 (≈ 1.79)

Tabelle 1: Vergleichswerte der Verfahren für *Test 1: Start des Clients*

	Anfragen	Objekte
Unoptimiert	916 ($+ 32$)	916 ($+ 32$)
FetchExpressions	431 ($+ 192$)	3992 ($+ 1668$)

Tabelle 2: Anzahl der Anfragen bei eingeschaltetem Lernmodul (*Test 1: Start des Clients*). Der Wert in Klammern kennzeichnet die Änderung der Werte gegenüber nicht aktivem Lernmodul.

der Performanz sehr bedenklich, da in diesem Fall unnötig viele einzelne Anfragen an den Server gestellt werden.

Die Standardregeln sind vom Prinzip her den ursprünglichen FetchExpressions sehr ähnlich. Sie agieren als zusätzliche Prefetching-Instanz, die eine serverseitig optimierte Objektmenge zurückzuliefert, wo das Prefetch-Modell aufgrund der Granularität nicht weiterhilft. Der Unterschied zu den FetchExpressions besteht darin, dass alle vom Prefetch-Modell behandelten Regeln nicht mehr beachtet werden müssen und die Regeln wesentlich einfacher und allgemeiner gehalten sind.

Die Standardregeln sind fest in den Server einprogrammiert. Ihre Gestaltung ist durch Analyse von Protokolldaten der Interaktion zwischen Client und Server manuell festgelegt worden und stellt implizites Wissen über Anfragestrukturen des Systems dar. Prinzipiell scheint es auch möglich, die Standardregeln über einen Lernprozess zu definieren. Im aktuellen System wurde dies aber noch nicht realisiert, da das Verfahren zur Ableitung dieser Regeln wesentlich komplexere, zeitaufwändigere Untersuchungen (und somit eine Offline-Auswertung) benötigen würde und explizites Wissen über alle Navigationsmöglichkeiten des Datenmodells modelliert werden müssten.

Intern werden für Standardregeln zwei verschiedene Fälle berücksichtigt: Objekte außerhalb und Objekte innerhalb eines Optimierungsblocks. Eine Unterscheidung kann in der Regel durch den Objekttyp getroffen werden.

Objekte außerhalb der Optimierungsblöcke sind hauptsächlich Verwaltungsinformationen. Ihre Anzahl ist im Vergleich zu Knoten, Links und Datenwerten in der Regel sehr gering. Die Standardregeln sorgen hier hauptsächlich dafür, dass diese Objekte nicht einzeln vom Server abgeholt werden, sondern blockweise zusammen. Dies stimmt auch mit der Art überein, wie sie vom Client benötigt werden.

Innerhalb der Optimierungsblöcke werden Standardregeln für Objekte eingeführt, die wegen Performanz und Granularität nicht mit optimiert werden. Geregelt wird vor allem der Zugriff auf einzelne Datenwertprimitive. So ist zum Beispiel festgelegt, dass beim Zugriff auf einen Datenwert gleich das zugehörige Datenwertprimitiv im aktuell eingestellten Aspekt mitgeliefert wird. So können unnötige Latenzen verhindert werden. Allerdings muss auch der Client angemessen auf diese Objekte reagieren und in der Registrierung berücksichtigen, dass diese Objekte serverseitig – ohne bestehende Optimierungsregel im Client – mitgeliefert worden sind.

4 Ergebnisse

In diesem Abschnitt wird die Gesamtleistung von *RBAP* als adaptives System zum Erlernen des Prefetch-Verhaltens analysiert, bewertet und mit FetchExpressions verglichen. Eine Analyse der Systeme findet hauptsächlich qualitativ statt. Gemessen werden die Anzahl der Anfragen an den Server und die Gesamtzahl der zurückgelieferten Objekte. Zeitmessungen gestalten sich aufgrund möglicherweise unterschiedlicher Netzwerktopologien als nicht sehr aussagekräftig. Liegt eine sehr hohe Latenz vor dann nimmt die Anzahl der einzelnen Anfragen maßgeblich Einfluss auf die Geschwindigkeit. Ist das Problem der Datendurchsatz der Netzwerkverbindung, dann verlangsamt eine große Gesamtzahl an vom Server zurückgelieferten Daten das Gesamtsystem.

Außerdem ist zu bedenken, dass die Verfahren, die miteinander verglichen werden, einen von Grund auf verschiedenen Aufbau haben. FetchExpressions sind ein rein serverseitiges Verfahren, während RBAP client- und serverseitig agiert. Es mussten viele Kernkomponenten (vor allem im Server) ausgetauscht werden, die nicht gleichwertig ersetzt wurden. Die Registrierung und das Lernen im Client benötigen zudem mehr Daten als eigentlich von der Anwendung in der Regel gebraucht werden.

Beide Verfahren werden anhand von ausgewählten Testfällen objektiv beurteilt. Als Datenbasis dient der Standard-Export der Datenbank für die Version *1.3.0* des *ST4 DocuManagers 1.3*.

- *Test 1: Start eines Clients*
  Der Start eines Clients testet die Güte der Standardregeln. Es werden viele Verwaltungsinformationen angefragt, aber nur wenige Knoten. Die Werte sind als Basis für die weiteren Testfälle zu sehen, da die anderen Testdaten immer unter der Voraussetzung gewonnen werden, dass der Client gestartet wurde und sich die dafür notwendigen Daten bereits im Cache befinden.
- *Test 2: Produktion eines Dokuments*
  Nach Start des Clients wird direkt ein HTML-Dokument produziert. Als Daten dienen das Beispielprojekt *Projekt HTML*. Dieser Test behandelt die Fähigkeit des Optimierungsmoduls, die für die Produktion notwendigen Datenwerte zu erlernen. Eine Produktion läuft in einem gesonderten Gültigkeitsbereich. Die Regeln überschneiden sich daher nicht mit den vorher gewonnenen Optimierungsdaten.
- *Test 3: Aufklappen aller Ebenen des Dokumentbaums*
  Dieser Testfall behandelt vor allem die Anpassungsfähigkeit des Verfahrens, wenn viele Links und zugehörige Linkdatenwerte ausgewertet werden müssen. Der Startknoten ist der Wurzelknoten des Dokumentbaums. Von dort aus werden alle Unterebenen rekursiv in einer Aktion aufgeklappt.

4.1 Vergleich zwischen FetchExpressions und RBAP

Im folgenden Abschnitt wird überprüft, wie gut sich *RBAP* als Prefetch-Verfahren im Vergleich zum FetchExpression-System eignet. Dies wird anhand der drei Testfälle bewertet, indem untersucht wird, wie viele einzelne Anfragen an den Server gestellt und wie viele Objekte vom Server zurückgeliefert wurden. Als Basis dienen die Werte des unoptimierten Clients. Generell gilt: Je weniger einzelne An-

	Anfragen	Objekte
Unoptimiert	3465	3465
FetchExpressions	600 (≈ 0.17)	5708 (≈ 1.65)
RBAP (initial)	1045 (≈ 0.30)	4310 (≈ 1.24)
RBAP (optimal)	522 (≈ 0.15)	4819 (≈ 1.39)

Tabelle 3: Vergleichswerte der Verfahren für *Test 2: HTML-Produktion*

	Anfragen	Objekte
Unoptimiert	2090	2090
FetchExpressions	350 (≈ 0.17)	3279 (≈ 1.57)
RBAP (initial)	577 (≈ 0.28)	2558 (≈ 1.22)
RBAP (optimal)	233 (≈ 0.11)	2632 (≈ 1.26)

Tabelle 4: Vergleichswerte der Verfahren für *Test 3: Aufklappen aller Ebenen des Dokumentbaums*

fragen an den Server gestellt werden und je näher die Anzahl der zurückgelieferten Objekte am Ausgangswert des unoptimierten Clients liegt, desto besser ist das Prefetch-Verhalten.

Test 1: Start des Clients
Die Messergebnisse aus Tabelle 1 zeigen, dass *RBAP* mit dem FetchExpression-System durchaus mithalten kann. Das neue Verfahren stellt zwar etwas mehr Anfragen an den Server, allerdings werden auch wesentlich weniger Objekte vom Server zurückgeliefert. Ist die Latenz im Netzwerk der dominierende Faktor, dann haben FetchExpressions einen leichten Vorteil. Wenn allerdings der Datendurchsatz im Netzwerk eher problematisch ist, hat *RBAP* einen großen Vorsprung.

Ein wichtiger Faktor, der nicht außer Acht gelassen werden sollte, besteht in dem Mehraufwand, den das neue Verfahren leisten muss. In Tabelle 2 ist aufgelistet, wie sich ein unoptimierter Client und ein Client mit FetchExpressions verhält, wenn das Optimierungsmodul im Client eingeschaltet ist. Insgesamt werden nur 32 Werte mehr angefragt – gegenüber den ca. 900 Objekten sonst eine Vergleichsweise geringe Zahl. Interessant ist der doch sehr hohe Zuwachs an Anfragen bei FetchExpressions. Dieser lässt sich hauptsächlich dadurch erklären, dass sich durch das Einschalten des Optimierungsmoduls die Reihenfolge der Objektanfragen ändert und somit Objekte, die sonst durch einen Prefetch mitgeliefert worden wären, einzeln angefragt werden. Dieser Wert unterstreicht die gute Integration der Standardregeln in das neue Verfahren.

Test 2: HTML-Produktion
Bei der HTML-Produktion (Tabelle 3) zeigen sich die Stärken von *RBAP*. Es kann sich genau darauf einstellen, welche Daten für die Produktion gerade benötigt werden. Das neue Verfahren benötigt etwa 10% weniger Anfragen und sorgt noch dazu für wesentlich weniger zurückgelieferte Objekte als FetchExpressions. Wichtig für diesen Testfall ist, dass die Produktion immer in einem gesonderten Gültigkeitsbereich abläuft. So kann das Optimierungsmodul unabhängig von anderen Einflüssen sehr schnell eine angepasste Prefetch-Lösung bereitstellen, die nach wenigen Optimierungszyklen gegen das Optimum konvergiert. Neben der HTML-Produktion wurden noch andere Produktionen untersucht. Diese werden aber nicht separat aufgeführt, da die Ergebnisse bei allen Produktionen abgesehen von minimalen Abweichungen gleichwertig sind.

	Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5
Test 1	322 (3)	295 (1)	292 (0)	292 (0)	292 (0)
Test 2	1045 (33)	689 (10)	579 (4)	522 (0)	522 (0)
Test 3	577 (16)	346 (7)	233 (0)	233 (0)	233 (0)

Tabelle 5: Anpassungsfähigkeit von *RBAP* nach *n* Optimierungsläufen. Der Wert in Klammern ist die Zahl der Änderungen am Prefetch-Modell im Zug des Optimierungslaufs.

Anhand dieses Testfalls lässt sich gut verfolgen, dass *RBAP* nicht nur aufgrund von Standardregeln gute Ergebnisse erzielt. Im initialen Zustand dominiert noch der Einfluss der Standardregeln. Allerdings bewirkt erst die Anpassung des Prefetch-Verhaltens mit der Zeit, dass ein deutlicher Vorsprung zum FetchExpression-System erzielt wird.

Test 3: Aufklappen aller Ebenen des Dokumentbaums
Bei der Interpretation der Ergebnisse von *Test 3* (Tabelle 4) muss bedacht werden, dass dieser Test im selben Gültigkeitsbereich abläuft, wie viele andere Aktionen. Das Ergebnis kann stark von früheren Optimierungszyklen beeinflusst sein. In der Regel wird deshalb der optimale Zustand bei *RBAP* nicht erreicht werden. Im optimalen Zustand ist *RBAP* dem FetchExpression-System deutlich überlegen, sowohl was die Anzahl der Anfragen als auch die Menge der zurückgelieferten Objekte anbelangt. Der initiale Zustand beinhaltet die erwartete maximale Anzahl an Anfragen für *Test 3*, da noch keine Unterstrukturen am Anfang mitgeliefert werden. Die Zahl der zurückgelieferten Objekte kann sehr stark schwanken, je nachdem, wie viele Datenwerte an Knoten und Links bei Start des Tests im Prefetch-Modell als mitzuliefern markiert waren.

4.2 Anpassungsfähigkeit

Eines der wichtigsten Merkmale des neuen Verfahrens ist die Fähigkeit, sich adaptiv an Anfragedaten und Modellierung anzupassen. In Tabelle 5 finden sich Messwerte, wie sich das Prefetch-Verhalten für die drei Testfälle über die Zeit ändert. Als Ausgangspunkt dient immer ein initiales Prefetch-Modell ohne Daten. Deutlich zu sehen ist, dass bereits nach dem ersten Optimierungslauf eine deutliche Verbesserung des Prefetch-Verhaltens eintritt. Nach drei Durchläufen ist das Prefetch-Modell in der Regel optimal angepasst. Dies hat damit zu tun, dass im ersten Durchlauf bereits die eindeutigen Änderungen durchgeführt werden und meist nur noch Anpassungen an bereits modifizierten Knoten übrig bleiben, die aufgrund der Datenlage und fortgeschrittenen Generationen nicht berücksichtigt wurden. Obwohl in den Testfällen hauptsächlich Strukturen hinzugefügt werden, kann es durchaus vorkommen, dass sie im späteren Lauf wieder entfernt werden. Die Testfälle haben also nicht rein additiven Charakter.

Je nach Leistungsfähigkeit des Clients ist es möglich, dass sich die konkreten Zahlen für das Prefetch-Verhalten geringfügig ändern. Da der Lernvorgang asynchron verläuft, ist es möglich, dass er aufgrund der Auslastung des Clients erst später durchgeführt wird als bei Rechnern, die noch Kapazitäten frei haben. In seltenen Fällen kann es vorkommen, dass sich aufgrund dieser Effekte das finale Prefetch-Modell geringfügig unterscheidet. Allerdings halten sich die Schwankungen in engen Grenzen. Auf fünf getesteten Rechnern unterschiedlicher Leistungsklasse lieferte *RBAP* jeweils ähnliche Ergebnisse und einen ähnlichen Verlauf der Prefetch-Optimierung.

5 Bewertung und Ausblick

Das regelbasierte, adaptive Prefetching erfüllt die Erwartungen und scheint zumindest nach den ersten Testläufen in der Lage, als automatisiertes Verfahren die manuell erstellten FetchExpressions vollwertig und teilweise sogar besser zu ersetzen. Obendrein besitzt es auch ein größeres Repertoire an Möglichkeiten, wie zum Beispiel die flexible Anpassung an neue Datenmodellierungen ohne weiteren Aufwand und die Fähigkeit sich an das individuelle Verhalten des jeweiligen Benutzers anzupassen.

In der Regel ist die zurückgelieferte Datenmenge besser an das eigentliche Ziel der Anfrage angepasst und die Anzahl der gestellten Anfragen für ein bestimmtes Problem ist oft sogar geringer als bei FetchExpressions. Bei statischen Aktionen in einem eigenen Gültigkeitsbereich (zum Beispiel Produktionen) wird dieses Optimum auch erreicht, bei dynamischen Aktionen liegen die Ergebnisse in der Regel unter dem Optimum – je nach Art und Beschaffenheit der Anfragen und der konkreten Ausprägung des Prefetch-Modells. Trotz eines erhöhten Rechenaufwands im Client ist gegenüber FetchExpressions keine erhöhte Laufzeit festzustellen, wenn Effekte durch Latenzen und Datenübertragung weitgehend ausgeschlossen werden.

Das Softwarepaket, welches im Laufe der Untersuchungen entwickelt wurde, erfüllt somit weitgehend die notwendigen Anforderungen für den Einsatz in *SCHEMA ST4*. Allerdings gibt es durchaus noch Ansätze, wie das Verfahren bzw. die Software verbessert werden kann:

- *Echte Multi-Level-Optimierung*
  RBAP ist durch den Optimierungsblock festgelegt auf die Reichweite einer Knotenebene. Aus der Praxis zeigt sich, dass viele Aktionen Einfluss auf mehrere Knotenebenen nehmen können. Durch eine Erweiterung des Optimierungsblocks über weitere Knoten hinweg könnte eine Verbesserung der Qualität der Lernverfahrens erreicht werden. Allerdings steigt gleichzeitig auch die Komplexität und der Aufwand für die Berechnungen.
- *Reduktion der Rechenzeit im Client*
  Gerade um komplexere Verfahren zu realisieren, müssen Möglichkeiten gefunden werden, die Rechenzeiten in Grenzen zu halten.
  Ein Ansatzpunkt ist die feste Definition dynamischer und statischer Gültigkeitsbereiche. Im ersten Fall ist eine Änderung der Anfragestruktur jederzeit möglich und der Lernvorgang muss immer ausgeführt werden. Das Prefetch-Verhalten für statische Aktionen (wie zum Beispiel Produktionen) müsste hingegen nur einmal erlernt werden. Sofern sich in der Datenmodellierung und am Ablauf der Aktion nichts ändert, muss kein weiterer Lernvorgang ausgeführt werden.
  Außerdem sollte es möglich sein, den Lernvorgang so umzugestalten, dass eine statistische Auswertung ermöglicht wird. Dabei werden nicht mehr alle Anfragen registriert, sondern im Durchschnitt nur jede *n*te Anfrage.
- *Erweiterte Lernmöglichkeiten*
  In Kombination mit einer intensiven Offline-Analyse von Protokolldaten sollte es möglich sein, bisher manuell festgelegte Erfahrungswerte, zum Beispiel Schwellwerte für die Optimierung oder das Standardregelwerk, über einen Lernprozess zu definieren, um möglichst optimal ans jeweilige System angepasste Regeln zu erhalten.
- *Temporäre Gültigkeitsbereiche*
  In etlichen Fällen ist es wünschenswert, Daten für Aktionen, die in mehreren Gültigkeitsbereichen ablaufen, zu exportieren. Anstatt für diesen Export eine neue Aktion und einen neuen Gültigkeitsbereich zu definieren, könnte ein kombinierter Gültigkeitsbereich (*Combined Scope*) geschaffen werden, der als Prefetch-Modell die additive Hülle aller beteiligten Gültigkeitsbereiche erhält. Somit würde ein erneuter Lernprozess entfallen und die Aktion würde von sämtlichen Verbesserungen in jedem einzelnen Gültigkeitsbereich profitieren.
  Das Gegenteil stellt das Konzept der *temporären Differenz von Gültigkeitsbereichen* (*Delta Scopes*) dar. Bei Wechsel von einem Gültigkeitsbereich *G* zu einem anderen Gültigkeitsbereich *H* kann es vorkommen, dass sich Daten von angefragten Objekten bereits im Cache befinden. Wird ein Cache-Miss auf einem Datenwert unter *H* registriert, obwohl der zugehörige Knoten sich im Cache befindet, kann durch eine Übertragung der Differenzmenge von *G* und *H* eine effektivere und sparsamere Prefetch-Menge für die Anfrage geliefert werden als durch reinen Prefetch mit Gültigkeitsbereich *H*.

Literatur

[Bengel, 2002] Günther Bengel. *Verteilte Systeme – Client-Server-Computing für Studenten und Praktiker*. Vieweg, 2nd edition, Jan. 2002. ISBN 3-528-15738-0.

[Consortium, 1999a] World Wide Web Consortium. XML Path Language (XPath) Version 1.0, 11 1999. `http://www.w3.org/TR/xpath`.

[Consortium, 1999b] World Wide Web Consortium. XSL Transformations (XSLT) Version 1.0, 11 1999. `http://www.w3.org/TR/xslt`.

[Coulouris *et al.*, 2002] George Coulouris, Jean Dollimore, and Tim Kindberg. *Verteilte Systeme – Konzepte und Design*. Addison-Wesley, 3rd edition, Jan. 2002. ISBN 3-8273-7022-1.

[Knafla, 1999] Nils Knafla. *Prefetching Techniques for Client/Server, Object-Oriented Database-Systems*. PhD thesis, University of Edinburgh, 1999.

[Liberty, 2003] Jesse Liberty. *Programming C#*. O'Reilly, 3rd edition, 2003. ISBN: 0-596-00117-7.

[Padmanabhan and Mogul, 1996] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, 1996.

[Process, 2005] Java Community Process. JSR 243: Java^{TM}Data Objects 2.0 – An Extension to the JDO specification (Proposed Final Draft), Aug. 2005.

[Schäffer, 2003] Bruno Schäffer. Durch dick und dünn – Probleme bei Thin-Clients. *JavaSPEKTRUM*, Jan. 2003.

[Teng *et al.*, 2005] Wei-Guang Teng, Cheng-Yue Chang, and Ming-Syan Chen. Integrating web caching and web prefetching in client-side proxies. Technical Report 15, IEEE Transactions on Parallel and Distributed Systems, May 2005.