Effizientes Matching von strukturierten Queries gegen XML-Dokumente

Übersicht

INHALT:

a. Grundlegende Konzepte
b. Lösungsansätze von NiagaraCQ und XFilter
c. Vergleich der Lösungsansätze

INFORMATIONEN:

Foliensatz und Ausarbeitung zum Download unter:

http://www.michael-stollberg.de

Abschnitt A

Grundlegende Konzepte:

- Benachrichtigungssysteme
- Continuous Queries
- Einsatz von XML
- Skalierbarkeit

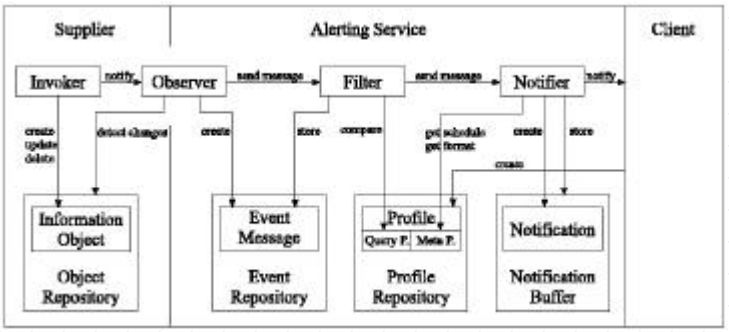
Benachrichtigungssysteme 1

- Ziel:
  Extraktion individuell relevanter Informationen aus einem großen Datenbestand
- Anforderungen:

	aus Nutzersicht:	Umsetzung:
1	aussagekräftige Profildefinition	Unterstützung entsprechender Anfragesprache
2	möglichst großer Datenbestand	Nutzung eines möglichst großen Objekt -Repositories
3	für möglichst viele Profilinhaber nutzbar	hochgradige Skalierbarkeit

Benachrichtigungssysteme 2

- allgemeine Architektur:



Nach A. Hinze, D.Faensen, FU 1999

Continuous Queries (CQs)

- technisches Konzept von Benachrichtigungssystemen:
  - Profile = persistente Anfragen im System
  - Automatisierte wiederholte Ausführung
  - Profilinhaber über aktuellen Stand seiner Interessensgebiete informiert
- Arten von CQs:

Timer-Based CQs:	Überwachung von Datenänderungen in einem bestimmten Zeitintervall
Change-Based CQs:	unverzügliche Benachrichtigung bei einer Datenänderung

- Konzept stammt aus der Datenbankforschung Anfang der 90er Jahre

Motivation zum Einsatz von XML

- Eigenschaften und Fähigkeiten von XML
  - Semistrukturiertheit / Selbstbeschreibung:

    XML-Element: <name attribute=123>content</name>

    Inhaltliche Beschreibung | Zuordnungsinfo | Data
  - anwenderspezifische Strukturierbarkeit
  - Werkzeuge (XSL, XPointer, Anfragesprachen)

  geeignet als Austauschformat und zur Datenverarbeitung, rasante Verbreitung
- Intentionen XML-basierter Benachrichtigungssysteme:
  - Unterstützung verschiedener Anwendergebiete
  - Weiterverarbeitung der Daten mit Werkzeugen
  - Bezug von Daten aus heterogen Quellen

Skalierbarkeit

- Im Einsatzgebiet Internet ist hochgradige Skalierbarkeit unabdingbar:
  - sehr viele potentielle Nutzer
  - sehr viele potentielle Datenquellen
- Problem:

  für ein Benachrichtigungssystem muss jede Datenänderung mit jedem Profil verglichen werden
- Ziel:

  Gewährleistung der Systemfunktionalität mit möglichst wenig Matches zwischen Profilen und Daten

Abschnitt B

Realisierungen der Lösungsansätze von NiagaraCQ und XFilter

- Entwicklungsumgebung

  Projekte, aus denen die Systeme hervorgegangen sind
- technische Umsetzung
  - Architektur
  - Profildefinition
  - Ansatz zum effizienten Matching

Niagara Project (Entwicklungsumgebung NiagaraCQ)

- seit 1999 / 2000
- Niagara Internet Query System:
  - Komplexere Auskünfte aus bekannten XML-Datenquellen
  - Anfrageformulierung in XML-QL
  - Entwicklung eines geeigneten Query-Prozessors und einer Suchfunktion für die Datenquellen

  Beispiel: http://www.cs.wisc.edu/niagara , java -jar niagara.jar www-db.cs.wisc.edu
- NiagaraCQ erweitert das Niagara Internet Query Systems zu einem Benachrichtigungssystem

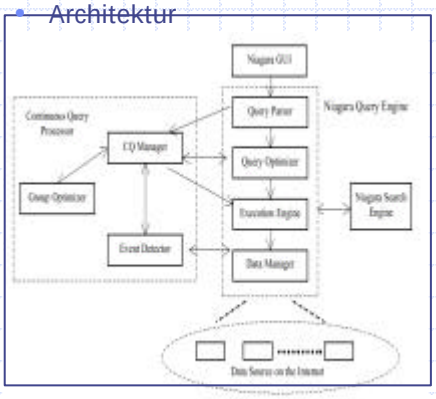
DBIS Project (Entwicklungsumgebung XFilter)

- seit 1997, Projektarbeit scheinbar eingestellt
- u.a. Entwicklung des DBIS Toolkit:
  - Framework für Dissemination Based Information Systems
  - definiert Aufgaben / Verantwortlichkeiten:
    - Data Sources
    - Information Broker
    - Client Library

    } Ähnliche Intention wie in der allgemeinen Architektur von Benachrichtigungssystemen
  - Schwerpunkt auf der Informationsverteilung
- XFilter ist eine Implementierung des DBIS Toolkit Frameworks

NiagaraCQ: Architektur

- ~~Architektur~~



PROZESSE:

1. neue CQ aufnehmen:
   - Eingabe über GUI
   - Erstellung des Query Plans durch QP
   - CQM beauftragt GO (Gruppierung) und ED (Eventüberwachung)
2. CQ ausführen
   - Timer-Based: ED fragt DM nach relevanten Datenänderungen
   - Change-Based: DM benachrichtigt ED bezüglich Datenänderung
   - CQM beauftragt EE (Ausführung der entsprechenden CQs)

NiagaraCQ: Profildefinition

Definition einer CQ:

```
CREATE CQ_name
XML-QL query
DO action
[START start_time] [EVERY time_interval] [EXPIRE
expiration_time]
```

Anfrage in XML-QL:

```
Where <Quotes> <Quote>
            <Symbol>INTC</>
          </> </> element_as $g
in "http://www.cs.wisc.edu/db/quotes.xml"
construct $g
```

Niagara Command Language:

- Name und Benachrichtigungsaktion
- Lebensdauer der CQ (START / EXPIRE)
- Ausführungsintervall (EVERY)
- Angaben umrahmen die eigentliche XML-QL Anfrage

XML-QL (deklarative Anfragesprache):

- Anfrage nach XML-Substrukturen mit Selektionsprädikaten
- Komplexe Anfragen durch entsprechende Konstruktion von Variablen möglich
- Angabe der Datenquelle (auch mehrere)
- Definition der Ausgabe möglich

NiagaraCQ: Ansatz zum effizienten Matching

- Ziel:

  Nutzung der existenten Anfragebearbeitung und Datenquellenorganisation

  ⇓

  Optimierung der CQ-Organisation statt Modifikation der Mechanismen

- Hypothese:

  Viele Anfragen sind gleichartig

  ⇓

  Gleichartige Anfragen werden zu einer Gruppe zusammengefasst

  ⇓

  nur ein Match für alle Anfragen einer Gruppe nötig

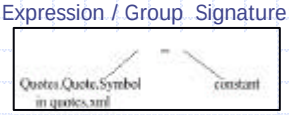
  ⇓

  höhere Skalierbarkeit

NiagaraCQ: inkrementelle Gruppenoptimierung 1

Anfrage:

Expression / Group Signature:



Group:
- Group Signature
- Group Constant Table
- Group Plan

Group Constant Table:

....	....
INTC	Dest. i
MSFT	Dest. j
....	....

PROZESS:

Parsen der Anfrage

inkrementelle Gruppierung:

? Gruppe:= Group Signature == Expression Signature

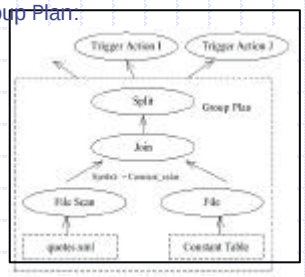
ja / nein

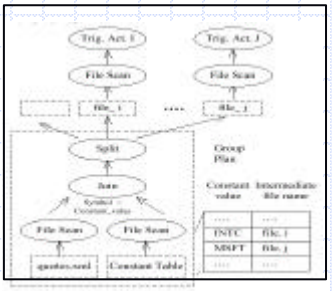
zu Gruppe hinzufügen:
Konstanten in die Group Constant Table aufnehmen, Destination Buffer zuweisen

neue Gruppe:
Group Signature == Expression Signature der Anfrage

NiagaraCQ: inkrementelle Gruppenoptimierung 2

Group Plan:





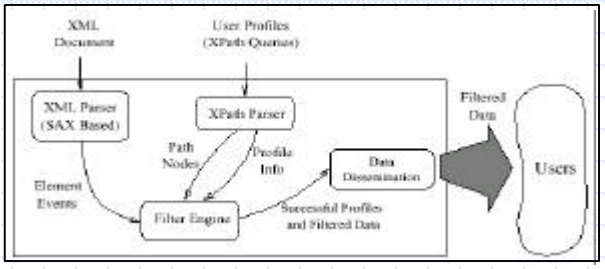
Ausführung des Group Plan:
- einmaliger Scan der Datenquelle
- für jeden Wert der Group Constant Table wird der Destination Buffer aktiviert

Nur 1 Match mit der Datenquelle für alle Anfragen in einer Gruppe nötig

Query Split:
- Komplexe Anfragen werden in Teilanfragen zerlegt
- Ergebnisse der 1. Teilanfrage stehen in der Constant Table
- Intermediate File dient als Datenquelle der 2. Teilanfrage
- Weniger Gruppen nötig

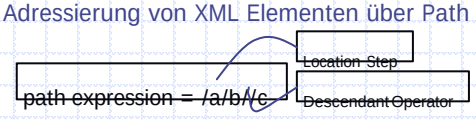
XFilter: Zielsetzung und Architektur

- Zielsetung: selektive Informationsverteilung
  aus eingehenden Dokumenten werden die jeweils relevanten an die Profilinhaber weitergeleitet (Dokumenten-Filter-System)
- Architektur:



XFilter: Prozesse und Profildefinition

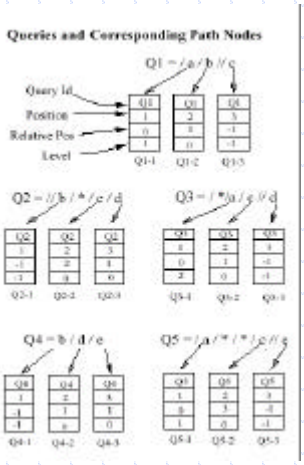
- Prozesse:
  - neues Profil:
    - Anfrageparsierung durch XPath Parser, Indizierung im System
  - Anfrage feuern:
    - detektierte Events eines eingehenden Dokuments löst Filtermechanismus aus
- Profildefinition über XPath:
  - Adressierung von XML Elementen über Path Expression:



- Filterdefintion
- Pfadangaben: absolut – relativ

  - Einsatz in XFilter: enthält ein Dokument die Path Expression einer Anfrage, wird es an den Profilinhaber weitergeleitet

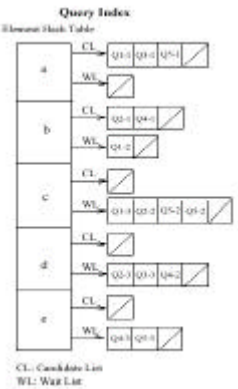
XFilter: Ansatz zum effizienten Matching 1



Path Nodes:

- als Final State Maschine einer Query
  - Zerlegung durch XPath Parser
  - Jeder Path Node ist ein State der FSM
- Werte:
  - Query ID: zugehörige Query
  - Position: Location Step der Expression
  - Relative Position:
    - -1 für Decendant Operator
    - 0 für ersten Location Step
    - (+) 1 sonst
  - Level: Hierarchiestufe im Dokument, auf der dieser Path Node geprüft werden soll
- NextPathNodeSet
  Pointer auf nachfolgende Path Nodes

XFilter: Ansatz zum effizienten Matching 2



Query Index:

Path Nodes sind ihrem Elementnamen zugeordnet.

- Candidate Lists:
  enthält den Current State des FSM einer Query (zu Beginn den Initial State)
- Wait Lists:
  enthält alle weiteren States der FSM einer Query

XFilter: Ansatz zum effizienten Matching 3

Filtermechanismus:

- XML Parser detektiert Events (Start Element, End Element, Element Characters)
- Event Handler:
  - Start Element: überprüft alle Path Nodes in der Candidate List
    - Levelcheck: Levelwert des Path Nodes gegen Dokumentenlevel
    - Filtercheck: Prüfung eventueller untergeordneter Filterdefinitionen

  ⇩

  Wenn Checks bestanden: Traversion der FSM in den nächsten State:
  PN löschen && Folge-PN aus der Wait List in die Candidate List
  - End Element: verschiebt Path Nodes in die Wait List
  - Element Charakter: überprüft Filterdefinition für Elementinhalt
    - Löst ebenfals FSM-Traversion aus

Wiederholung, bis der Final State der FSM einer Query erreicht ist:
dann ist das Dokument als passend zum Profil ermittelt.

XFilter: Ansatz zum effizienten Matching 4

Erweiterte Filteralgorithmen:

- List Balancing:
  statt des ersten Path Nodes einer Query wird derjenige als Inital State genommen, dessen Candidate List im Query Index am kürzesten ist.

  ⇩

  Ausgewogene Anzahl von Path Nodes für die Elementnamen

- Prefiltering:
  vor dem Filtern werden alle Queries aus dem Query Index entfernt, die XML-Elemente aufweisen, die nicht im eingehenden Dokuments sind.

  ⇩

  Weniger potentielle Queries im Query Index

Abschnitt C

Vergleich der Lösungsansätze

- Ausdrucksfähigkeit der Profildefinitionssprachen im Zusammenhang mit der Verwendung im System
- Skalierbarkeit
- Ansätze zum effizienten Matching

Ausdrucksfähigkeit der Profildefinitionssprachen 1

Vergleich der Anfragesprachen

Aspekt	XML-QL	XPath
Selektionsqualität:	Wertselektion	Strukturselektion
Deklaration der Datenquelle:	explizit in der Anfrage	von außen an Quelle herantragen
Ergebniskonstruktion:	explizit in der Anfrage	Implizite Rückgabe: XML-Subtree / FALSE

⇒ unterschiedliche funktionale Konzeptionen:

XML-QL := aufarbeitende Informationsgewinnung aus bekannten Datenquellen

XPath := inhaltliche Evaluation unbekannter XML-Dokumente

Ausdrucksfähigkeit der Profildefinitionssprachen 2

Verwendung der Sprachen in Systemen

- NiagaraCQ:
  - Funktionale Intention: Auskunftssystem
  - Anwendungssystem von XML-QL als deklarative Anfragesprache
- XFilter:
  - Funktionale Intention: Dokumenten-Filter-System
  - Nutzung von XPath zur inhaltlichen Evaluation von Dokumenten

Funktionale Intention der Systeme == Funktionale Konzeption der Anfragesprachen

Skalierbarkeit

Anfragenhomogenität

Anfragen (in XPath):

```
Q1 = //a/b     Q3 = //b/c
Q2 = //a/b/c   Q4 = /a/b
```

Art der Homogenität:

gleich:	Q1,Q2
gleichartig:	Q1,Q2; Q1,Q3; Q2,Q3
ungleich:	Q1,Q4; Q2,Q4; Q3,Q4

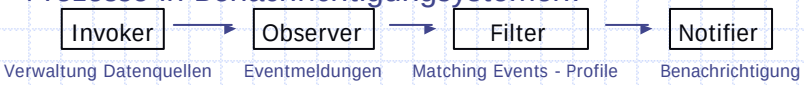
- NiagaraCQ:
  Skalierbarkeit desto größer, je homogener bzgl. Gleichartigkeit
- XFilter:
  Skalierbarkeit desto größer, je heterogener bzgl. Gleichheit
  (nur beim Basic Algorithmus – beim List Balancing besteht keine Abhängigkeit)

NiagaraCQ ist stärker abhängig als XFilter

Ansätze zum effizienten Matching

Vergleich der Optimierungsansätze

- Prozesse in Benachrichtigungsystemen:



Verwaltung Datenquellen — Eventmeldungen — Matching Events - Profile — Benachrichtigung

- NiagaraCQ:
  - Unterstützung aller Prozesse (abgeschl. Benachrichtigungssystem)
  - Optimierungsansatz: Aufgabendelegation der Filterkomponente
- XFilter:
  - Invoker fehlt (als Filterkomponente für größere Informationssysteme)
  - Optimierungsansatz: algorithmischer Filtermechanismus

Zusammenfassung

Unterschiede:

Aspekt	NiagaraCQ	XFilter
Funktionale Intention:	Auskunftssystem	Dokumenten-Filter-System
Profildefinitionssprache:	XML-QL: Informationsaufarbeitung	XPath: Inhaltsevaluation
Skalierbarkeit: (bzgl. Anfragehomogenität)	Abhängig bzgl. Gleichartigkeit	Abhängig bzgl. Gleichheit, bzw. gar nicht abhängig
Optimierungsansatz:	Aufgabendelegation	Leistungsoptimierung

Statement

NiagaraCQ:

- abgeschlossenes Auskunftssystem
- komplexe Anfragen möglich
- Skalierbarkeit nur durch ausgefeiltes Zusammenspiel gewöhnlicher Technologien

XFilter:

- Dokumenten-Filter-System als Komponente
- weniger komplexe Anfragen möglich
- Skalierbarkeit durch algorithmischen Filtermechanismus gesichert

Vielen Dank und schönes Wochenende noch…