# Registrierung und Suche von Diensten im Internet

Donald Kossmann

TU München

http://www3.in.tum.de

# Übersicht

1. Hintergrund:  Das ObjectGlobe Projekt
2. Speichern von XML Dokumenten
3. Der ObjectGlobe Lookup-Service
4. Zusammenfassung und Ausblick

# Das ObjectGlobe Projekt

Ziel: Offener Marktplatz für datenintensive Dienste im Internet/Intranet

Es gibt drei Arten von Diensten:

- Data Provider
- Function Provider
- Cycle Provider

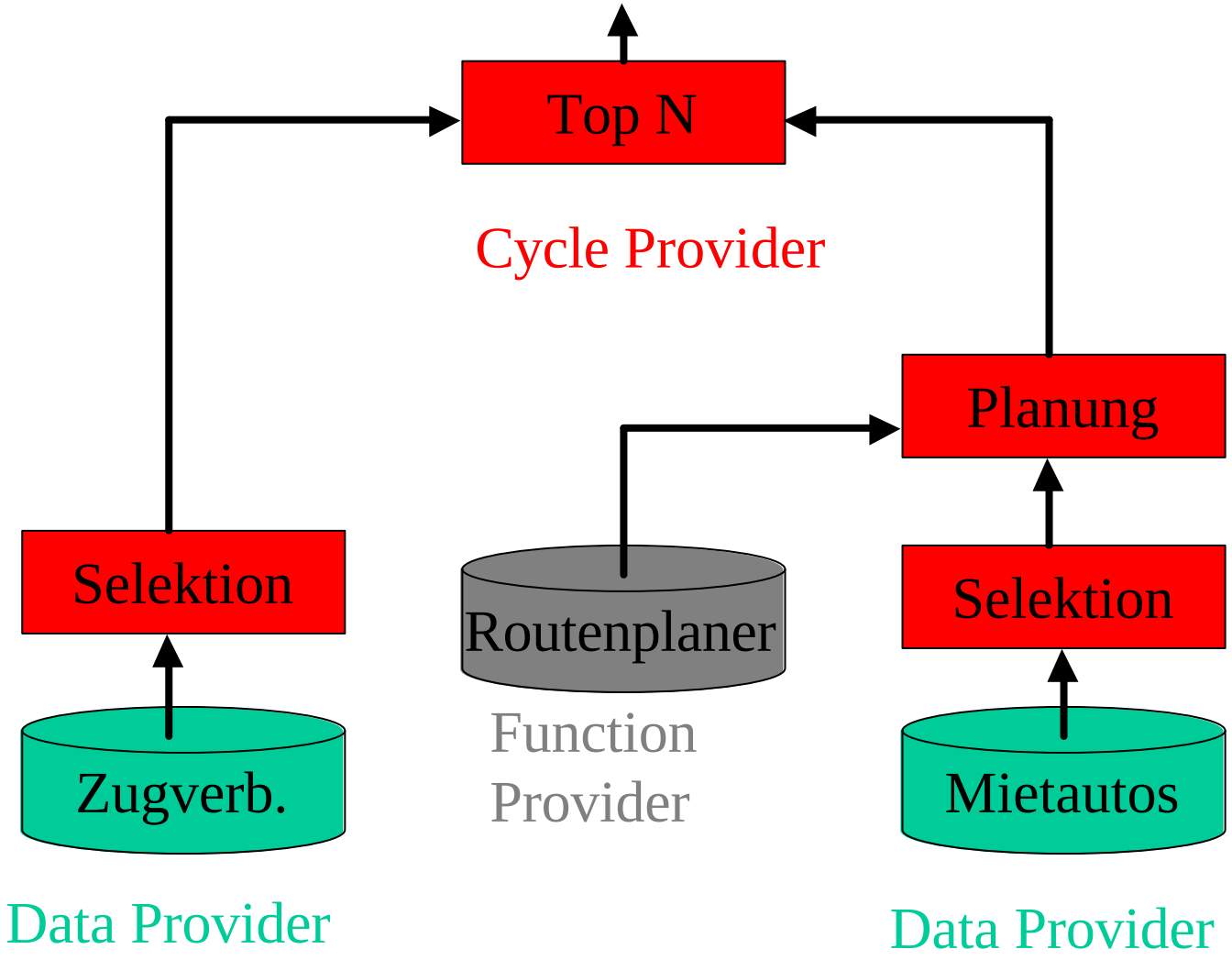
Beispiele:

- Outsourcing (ASP)
- Hochleistungsrechnen im Internet

# Beobachtungen

- Es gibt viele Anbieter von Diensten
  - intern (z.B. eigenes Rechenzentrum)
  - extern (z.B. der PC von Lieschen Müller)
- Eine Anfrage/Anwendung beansprucht oft mehrere Dienste unterschiedlicher Anbieter
- Es gibt sehr viele Möglichkeiten, eine Anfrage auszuführen
- Benutzer erwarten Transparenz

# Beispiel: Reisen von M nach KA



# Laufende Arbeiten

- Anfragebearbeitung
  - adaptive Algorithmen, Caching und Replikation, Unterstützung neuer Anfragetypen
- Sicherheit und Schutz vor Missbrauch
  - Sandbox, Verschlüsselung und Signieren von Nachrichten, Monitoring, Dienste-TÜV
- Dienstegüte (Quality of Service)
- Lookup-Service, verteilte Metadatenverwaltung
- Prototyp und Demo auf dem Internet

  http://www.db.fmi.uni-passau.de/projects/ObjectGlobe

# Dienstgüte in ObjectGlobe

- Benutzer können Ziele vorgeben
  - Kosten der Bearbeitung (z.B. 50 Pfennig)
  - Laufzeit der Bearbeitung (z.B. 5 Minuten)
  - Qualität der Antwort (z.B. Anzahl der Datenquellen)
- Aufgabe des Systems
  - möglichst viele Anfragen im Rahmen ausführen
  - möglichst früh eine hoffnungslose Anfrage abbrechen
  - wenn möglich/nötig: früh Alternativen finden
- Technische Umsetzung
  - Mehrzieleoptimierung, Monitoring der Abwicklung

**Wichtig: Viele Dienste im Konzert**

# Der ObjectGlobe Lookup-Service

- Speichern von XML (und RDF) Dokumenten in relationalen Datenbanken
- Hierarchische Architektur von Lookup-Servern
- Publish & Subscribe Mechanismus zum Abonnieren von relevanten Metadaten in lokalen Lookup-Servern

# Speichern von XML Dokumenten

- Motivation
  - viele Daten werden im XML Format erzeugt
  - insbesondere:  Beschreibung von Diensten
  - Anfragen auf XML Dokumente zur Suche von Diensten
- Ansatz
  - Einsatz eines relationalen Datenbanksystems
    - Abbildung:  XML ->  Graph
    - Speichern der Knoten und Kanten in Tabellen
  - Einsatz einer XML Anfragesprache
    - Abbildung:  XML Anfrage -> Graph
    - Implementierung der Kanten durch Joins in SQL

# Überblick über andere Arbeiten

- Einsatz eines speziellen XML Systems [Lore]
  - keine Produktreife
  - schlechte Ko-existenz mit anderen Daten
- Einsatz eines objektorientierten Systems [Abi99]
  - keine Produktreife
- Andere relationale Ansätze [AT&T00, Wis00]
  - sehr komplex, hoher administrativer Aufwand
  - nicht universell einsetzbar

# XML Beispiel

```
<person, id = 4711>
   <name>  Lilly Potter </name>
   <child>  <person, id = 314>
         <name> Harry Potter </name>
         <hobby> Quidditch </hobby>
  </child>
</person>
<person, id = 666>
   <name> James Potter </name>
   <child>  314 </child>
</person>
```

# XML ungleich Relational

| Name | Age | Child | Hobby |
|---|---|---|---|
| Harry Potter | 12 | NULL | ? |
| James Potter | NULL | ? | NULL |
| Lilly Potter | NULL | ? | NULL |

- viele NULL Werte
- Probleme mit Kollektionen (mehrere Hobbies)
- Verschachtelung vs. IDREFs
- wie behandelt man neue Elemente (z.B. Adresse)
- XML Elemente vs. Attribute, Reihenfolge

<person, id = 4711>

<name> Lilly Potter </name>

<child> <person, id = 314>

<name> Harry Potter </name>

</child>

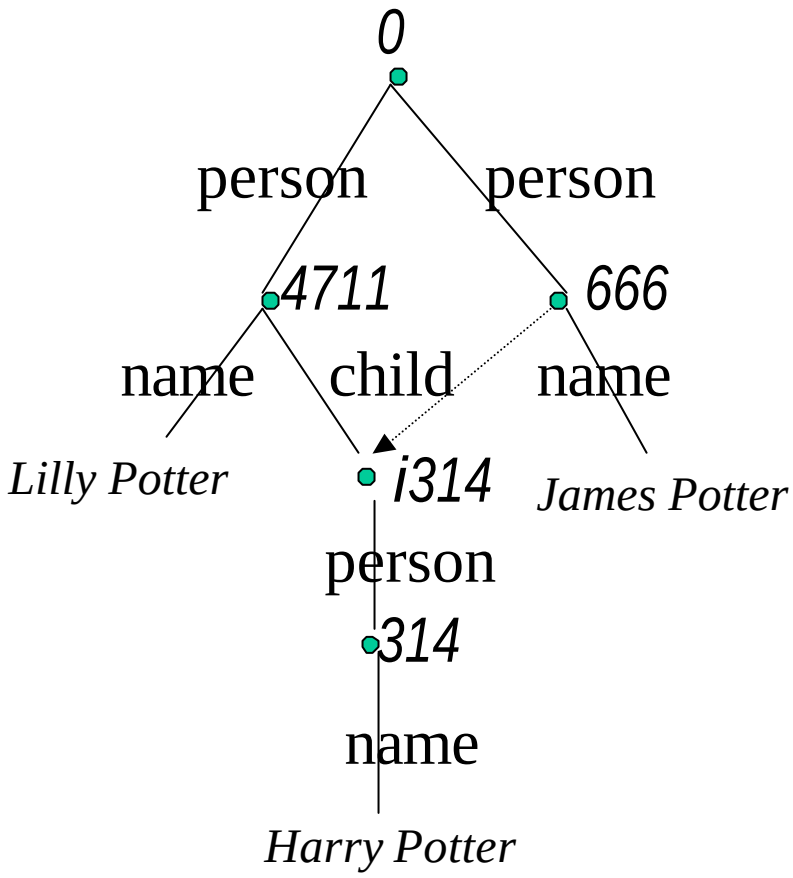
</person>

<person, id = 666>

<name> James Potter </name>

<child> 314 </child>

</person>



# Speichern eines Graphen
# Edge Approach

*Edge Table*

| Source | Label | Target |
|---|---|---|
| 0 | person | 4711 |
| 0 | person | 666 |
| 4711 | name | v1 |
| 4711 | child | i314 |
| 666 | name | v2 |
| 666 | child | i314 |

*Value Table (String)*

| Id | Value |
|---|---|
| v1 | Lilly Potter |
| v2 | James Potter |
| v3 | Harry Potter |

*Value Table (Integer)*

| Id | Value |
|---|---|
| v4 | 12 |

# Binary Approach

## Partitioniere die *Edge* Tabelle nach *Label*

*Person Tabelle*

| Source | Target |
|---|---|
| 0 | 4711 |
| 0 | 666 |
| i314 | 314 |

*Name Tabelle*

| Source | Target |
|---|---|
| 4711 | v1 |
| 666 | v2 |
| 314 | v3 |

*Child Tabelle*

| Source | Target |
|---|---|
| 4711 | i314 |
| 666 | i314 |

*Age Tabelle*

| Source | Target |
|---|---|
| 314 | v4 |

# Weitere Ansätze

- *Universal Approach*
  OuterJoin aller „Binary Tables“
- *Universal Normalized Approach*
  besondere Behandlung von Kollektion
- *Inlining*
  „Binary Tables“ OuterJoin „Value Tables“
- Mischformen und Varianten
- Besonderheit aller Varianten
  – zusätzliche Felder für Reihenfolgetreue,
    Unterscheidung von Elementen und Attributen, etc.

# XML Anfragen

- Finde den Namen aller Personen, die jünger als 18 sind und Quidditch als Hobby haben

```
select   $n
where  <person>
              <name>  $n </name>
              <age>     $a </age>
              <hobby> Quidditch </hobby>
          </person>,  $a < 18
```

- Semantik:  Überdeckung mit dem XML Graphen

# XML-Anfragen -> SQL

- Jede Kante in der XML Anfrage wird zu einem Join in der SQL Anfrage
- Blätter in der XML Anfrage werden zu Joins mit der „Value Tabelle“
- Reguläre Pfadausdrücke werden zu rekursiven SQL Anfragen
- Optionale Prädikate werden zu Outerjoins

**Übersetzung kann automatisch realisiert werden**

# Beispiel (Edge Approach)

```
SELECT   nv.value
FROM     Edge p, Edge n, Edge h, Value nv, Value hv
WHERE    p.label  =  „person“  AND
         p.target  =  n.source  AND
         n.label  =  „name“  AND
         n.target  =  nv.id  AND
         p.target  =  h.source  AND
         h.label  =  „hobby“  AND
         h.target  =  hv.id  AND
         hv.value  =  „Quidditch“;
```

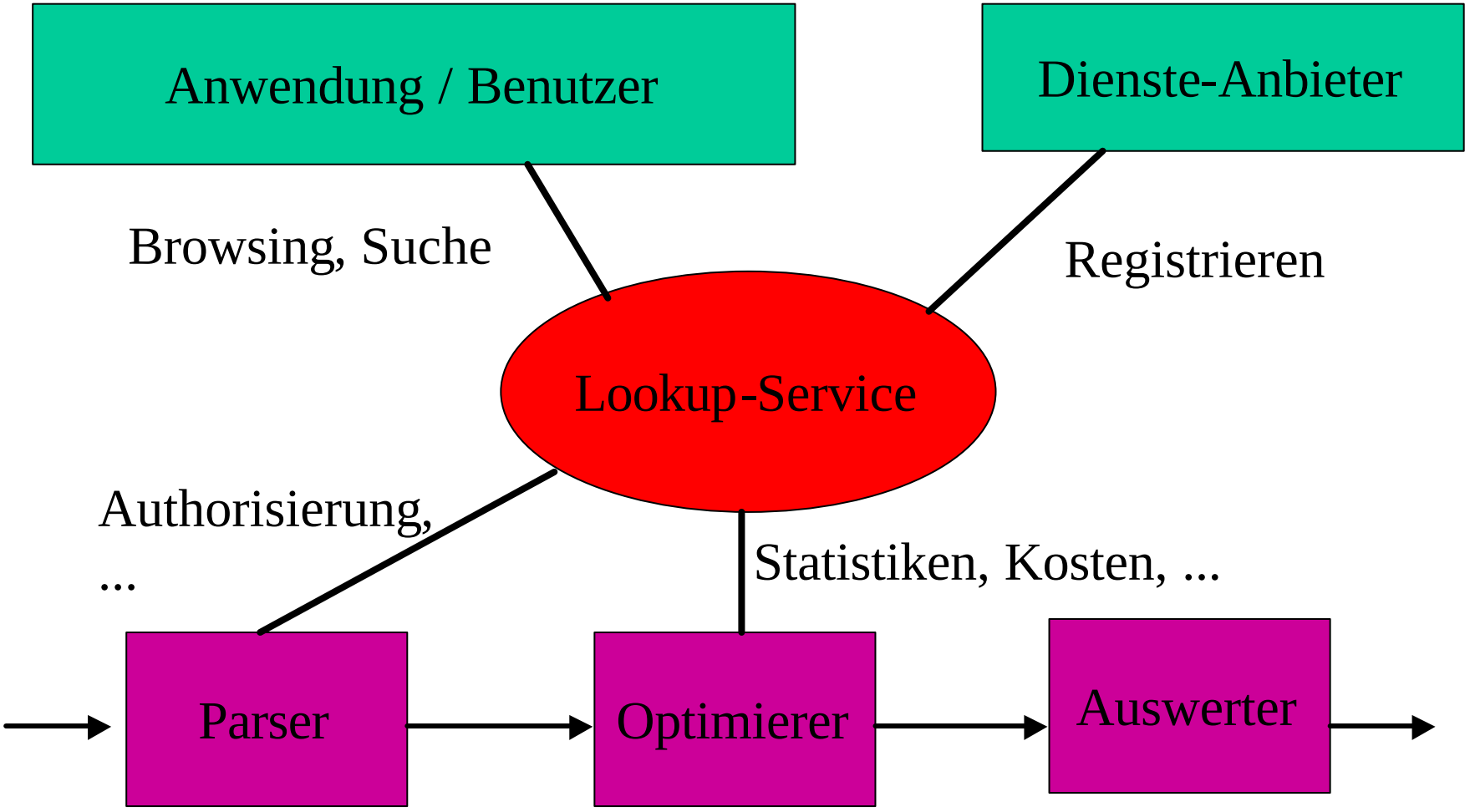
# Experimentelle Bewertung

- Synthetisches XML Dokument
  - 200.000 Elemente
  - tiefe Verschachtelung, Zyklen durch IDREFs
- 17 XML-QL Anfragen
  - einfache Punktanfragen
  - komplexe Muster, reguläre Pfadausdrücke, ...
  - unterschiedliche Selektivitäten
- 4 Update Funktionen (inserts / deletes)
- Hardware:
  - SUN Station 20, 128 MB Hauptspeicher, 1 Festplatte

# Zusammenfassung

- nicht optimal aber hinreichend
- sehr einfach zu implementieren
- kein administrativer Aufwand für Anwender
- universell einsetzbar und übertragbar
  - implementiert im Microsoft SQL Server
  - Schlüsselsuche integrierbar [WWW9 Conf.]
  - XML Datenintegration [VLDB 2000 Conf.]
  - Publish & Subscribe (nächster Abschnitt)
- offen: XML Transaktionsverarbeitung

# Der ObjectGlobe Lookup-Service



# Beschreibung von Diensten

- Es gibt ein vorgefertigtes, erweiterbares Schema zum Beschreiben von Diensten
- Viele Felder optional oder vom System belegt
- Data Provider:
  - Thema (z.B. Hotel)
  - Attribute (z.B. Preis eines Doppelzimmers)
  - Zugriffsfunktionen
  - Servercharakteristika
  - Authorisierungsinformation
  - Statistiken
  - ...

- Function Provider:
  - Signatur (z.B. foo(int, int) -> int)
  - Authorisierungsinformation
  - Hardwarevoraussetzungen (z.B. 30 MB Speicher)
  - Größe des Codes
  - ...
- Cycle Provider:
  - Hardware (z.B. 1 GB Hauptspeicher)
  - Authorisierungsinformation
  - Standort und Netzwerkanbindung
  - ...

# Beispiel:  XML Beschreibung eines Data Providers

```
<DataProvider>
   <id> 4711 </id>
   <theme>
      <name> Hotel </name>
      <desc> All hotels you ever want </desc>
    </theme>
    <Attribute>
         <topic> city </topic>
         <type>  string </type>
    </Attribute>
...
```

# Anfrage an den Lookup-Server

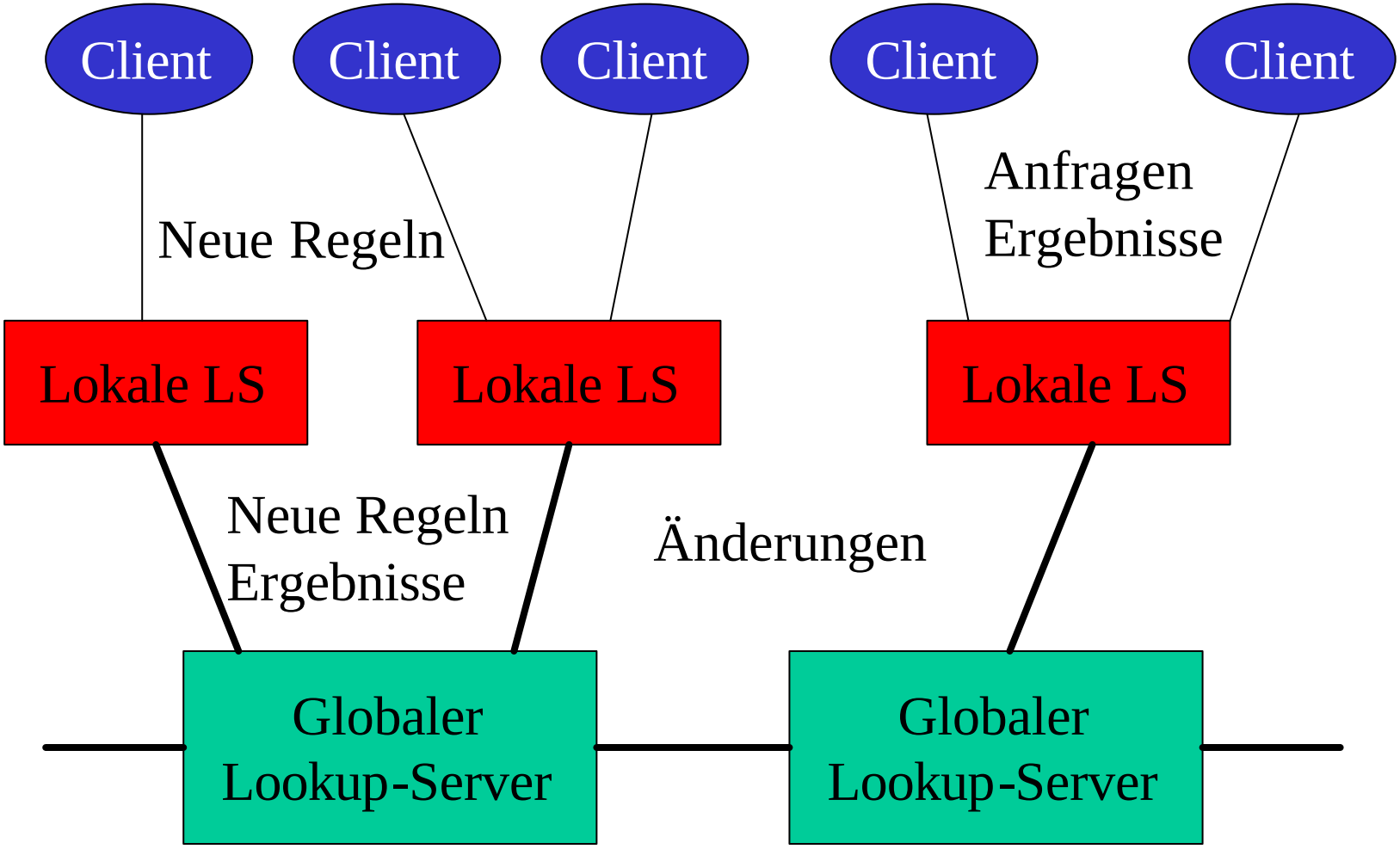
- Data Provider zum Thema Hotel, die den Ort und Preis liefern

```
search DataProvider d
select d.uniqueId, d.attr.*
where d.theme.name = „hotel“ and
        d.attr.?.topic = „city“ and
        d.attr.?.topic = „price“
```

# Zweistufige Architektur

- Es gibt lokale Lookup-Server
  - speichern alle Informationen für relevante Dienste für ein Subnetz (z.B. ein Unternehmen oder eine Filiale)
  - üblicherweise werden alle Anfragen lokal ausgeführt
  - Inhalt wird durch Anfragen / Regeln spezifiziert
- Es gibt globale Lookup-Server (Backbone)
  - speichern alle Beschreibungen von Diensten
  - speichern alle Regeln von lokalen Lookup-Servern
  - propagieren neue Dienste und Änderungen
  - surfen im Backbone möglich
- Hierarchiebildung möglich (z.B. Regionalisierung)

# Architektur



- Ausführung von Anfragen (Lookup Requests)
  - lokale LS hält XML/RDF Beschreibungen in RDB
  - Anfragen werden wie XML Anfragen ausgewertet
- Registrieren neuer Dienste
  - XML/RDF Beschreibung wird von globalen LS in RDB gespeichert
  - globale LS prüfen, ob neue Beschreibung auf eine Regel eines lokalen LS passt (Vorfiltern)
  - Propagieren der Beschreibung des neuen Dienstes an lokale LS
- Löschen / Ändern von Diensten
  - Ähnlicher Vorfilter Mechanismus zur Ermittlung von betroffenen lokalen LS

# Vorfiltern (Publish&Subscribe)

- Zerlege Regeln der lokalen LS in „atomare“ Teilregeln und speichere die Teilregeln in der relationalen Datenbank des globalen LS
- (Beschreibung von Diensten wird auch zerlegt.)
- Verwende SQL Join-Anfragen um matchende Teilregeln und XML Elemente zu finden
- Verwende SQL Join-Anfragen, um matchende Teilregeln zu matchenden Gesamtregeln zusammenzubauen

# Beispiel: Speichern von Regeln

- Finde Datenquellen zum Thema Börse, die weniger als 500 DM kosten:

  search DataProvider d
  where d.theme.name = „Börse“ and
  d.cost < 500

- Zerlegung in drei Teilregeln:

  R1: search Theme t where t.name = „Börse“
  R2: search DataProvider d where d.cost < 500
  R3: search R1 a, R2 b where b.theme = a

- Speichere Regeln als Tupel in relationaler Datenbank

| Regel | Typ | Operator | Attribut | Konstante |
|---|---|---|---|---|
| R1 | Thema | = | name | Börse |
| R2 | DataProv. | < | cost | 500 |

# Beispiel: Matching

| Regel | Typ | Operator | Attribut | Konstante |
|---|---|---|---|---|
| R1 | Thema | = | name | Börse |
| R2 | DataProv. | < | cost | 500 |

| Objekt | Typ | Attribut | Wert |
|---|---|---|---|
| O1 | Thema | name | Börse |
| O1 | Thema | description | B.Info.System |
| O2 | DataProv. | theme | O1 |
| O2 | DataProv. | attr | O3 (kurs) |
| O2 | DataProv. | attr | O4 (wkn) |
| O2 | DataProv. | cost | 70 |

Ergebnis des Joins: (R1, O1); (R2, O2)

# Verwandte Arbeiten

- Lookup-Services: z.B. Jini (SUN),
  UDDI (Ariba, IBM, Microsoft), Ninja (UCB)
  - sehr viel eingeschränkter (z.B. Jini nur im LAN)
- Repositories fürs Software-Engineering
  (Software Wiederverwendung)
  - zentrale Architektur
- Publish & Subscribe (Niagara, SIFT, Inria)
  - wesentlich einfacheres Modell

# Zusammenfassung

- Einsatz von Standard DB-Technologie
- Prinzip:
  - Zerlege Daten, Anfragen und Regeln in kleinste Bestandteile
  - Matche die kleinsten Bestandteile
  - Baue Ergebnisse zusammen
- hoher Aufwand, aber universell einsetzbar, geringer administrativer Aufwand und in bisherigen Untersuchungen hinreichend effizient

# Ausblick

- Finetuning, viele Experimente (Zusammenarbeit mit anderen Gruppen)
- Bedienkomfort des Prototypen verbessern
- Anwendungsszenarien implementieren
- Modell erweitern (Anfragen -> Workflow)
- **Das Projekt ist noch ganz am Anfang**