

Transaction Validation for XML Documents based on XPath

@ Informatik 2002, m-dbis



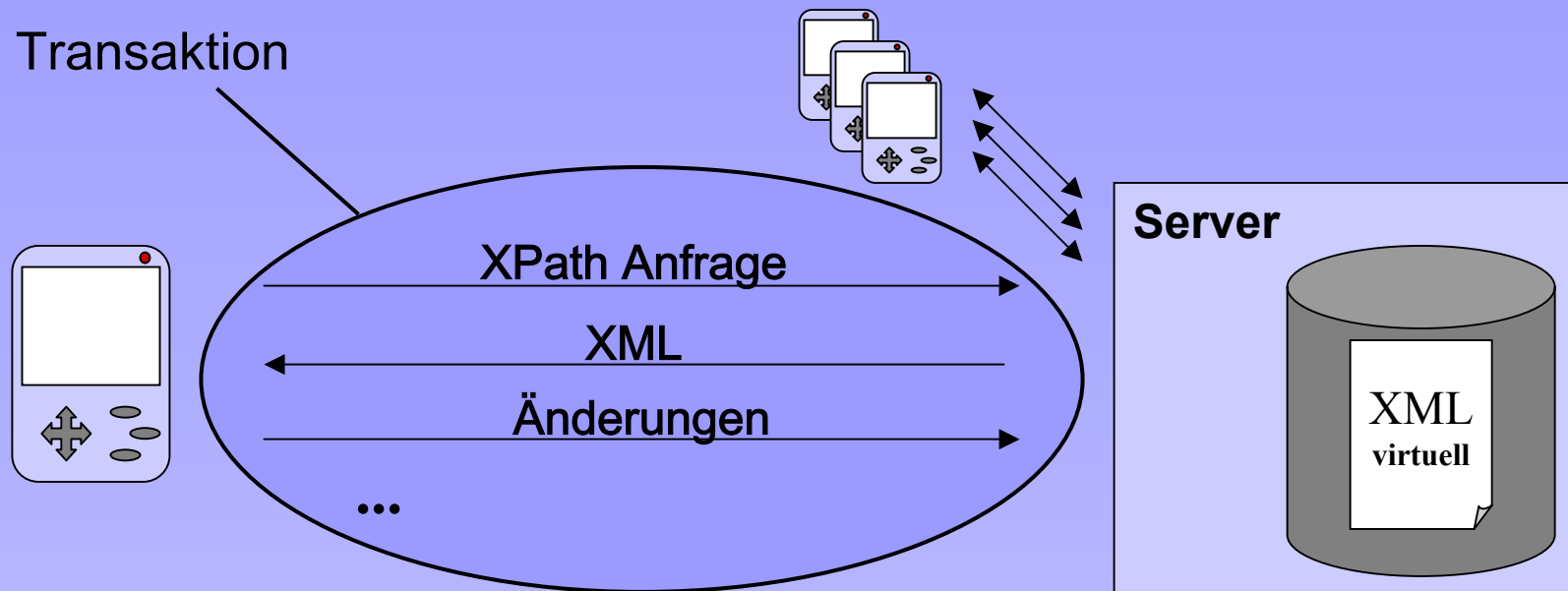
Stefan Böttcher
Adelhard Türling

Universität Paderborn

Transaktionen für XML - Daten & mobile Clients

- Motivation & Framework
- Voraussetzungen für mobile Transaktionen
- Validierung für XML Daten
- Prädikative Validierung mit XPath und XML
- Optimierung der Kommunikation & anderer Ressourcen
- Optimierung der Kommunikation durch HS Caching
- Zusammenfassung
- Ausblick

Szenario



Client:

- XPath Queries
- Arbeitet auf virtuellem XML

Server:

- Liefert virtuelles XML Dokument
- Synchronisiert parallele Zugriffe

← Standard Transaktionszusicherungen →

Voraussetzungen - Ansatz

mobiler Client:

- Lose Kopplung →
- Offline-Phasen →
- Bandbreite/Volumen →
- Geringe Speicherressource →
- Geringe Rechenressource →

Anforderung:

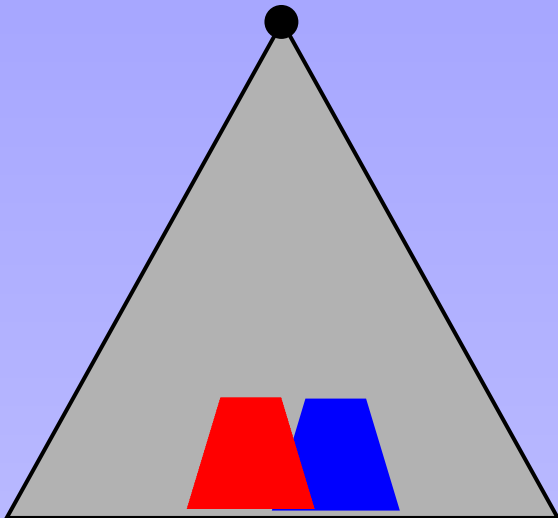
- Robust gegen Abbrüche
- Unabhängigkeit
- Kommunikation minimieren
- Datenmanagement/Caching
- Last an Server abgeben

Lösung:

- Optimistische Synchronisation (Validieren)
- Validieren auf XML Mengen
- Logische Validierung mit XPath



XPath & XML Fragmente



Virtuelles XML Dokument

```
<BookingService>
  <Connections>
    <Connection id="1">
      <destination>Paris</destination>
      <departure>London</departure>
    </Connection>
    <Connection id="2">
      <destination>Paris</destination>
      <departure>Hamburg</departure>
    </Connection>
    <Connection id="3">
      <destination>Rom</destination>
      <departure>Hamburg</departure>
    </Connection> ...
  </Connections> ...
</BookingService>
```

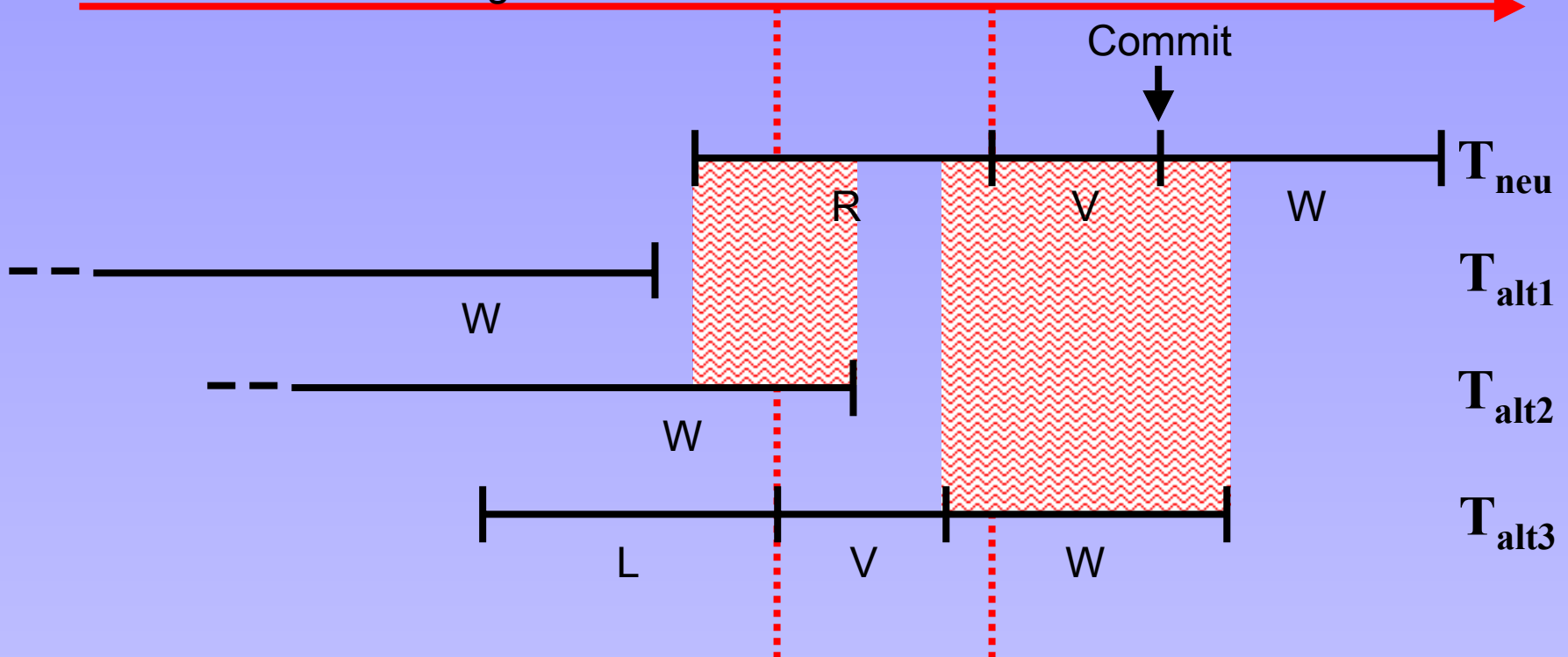
Client1: /BookingService/Connections/Connection [./destination = 'Paris']

Client2: /BookingService/Connections/Connection [./departure = 'Hamburg']



Parallel Validieren

serialisierte Reihenfolge

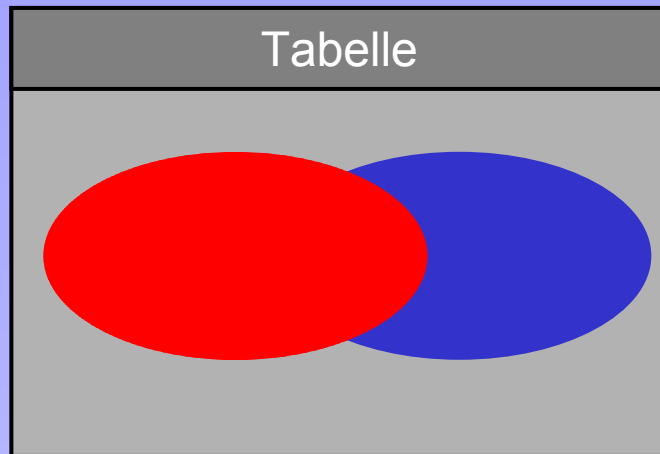


$$\text{WriteSet}(T_{alt}) \wedge \text{ReadSet}(T_{neu}) = \emptyset$$

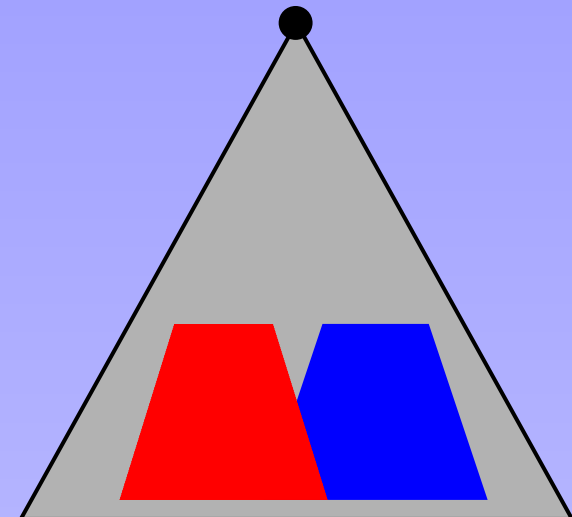
$$\text{WriteSet}(T_{alt}) \wedge (\text{ReadSet}(T_{neu}) \vee \text{WriteSet}(T_{neu})) = \emptyset$$



XML Validierung



z.B. Tupel ein relationalen DB



Virtuelles XML Dokument

- Schnittmengen-Tests XML für Fragmente
- Identifiziert z.B. durch XPath

Mobiles XML Validieren

Client:

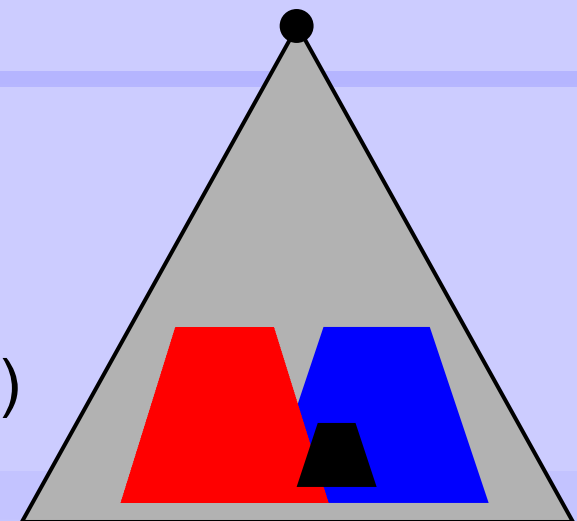
- Hält die physikalische Kopie des XML Fragments
- Änderungen auf Kopie

⇒ Kommuniziert XPath an Stelle von XML Dokumenten

⇒ Prädikative Validierung mit XPath

Server:

- Logisches ReadSet (XPath)
- Logisches WriteSet (XPath)
- Physikalisches WriteSet (Δ XML Daten)



Δ XML Fragmente

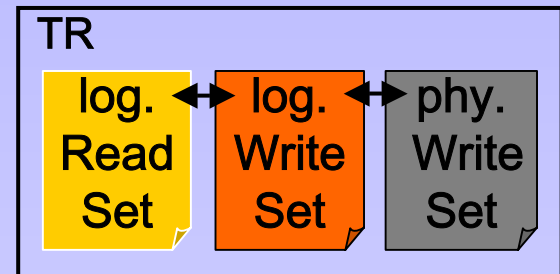
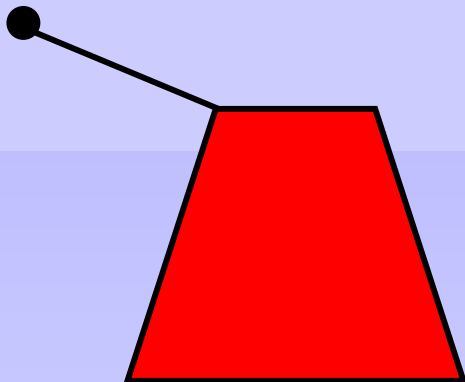
Read Phase

Client

- **Read:**
 - XPath Queries →
 - XML Fragment ←
- **Insert/Update:**
 - XPath (eindeutig*) →
 - Δ XML Fragmente →
- **Delete:**
 - XPath (eindeutig*) →

Server

- **log. ReadSet:** + XPath
- XPath Query auf DB
- **log. WriteSet:** + XPath
- **phy. WriteSet:** + XPath Query auf DB
+ Δ XML Fragment in
- **log. WriteSet:** + XPath
- **phy. WriteSet:** + XPath Query auf DB



Validierungsphase – Write Phase

- **Read/Write Konflikte**

$$\text{ReadSet}(T_{\text{neu}}) \wedge \text{WriteSet}(T_{\text{alt}}) = \emptyset \quad \Leftrightarrow$$

XPath Queries aus $\text{ReadSet}(T_{\text{neu}})$ angewendet auf phy. $\text{WriteSet}(T_{\text{alt}}) = \emptyset$

- **Write/Write Konflikte**

$$\text{WriteSet}(T_{\text{alt}}) \wedge (\text{ReadSet}(T_{\text{neu}}) \vee \text{WriteSet}(T_{\text{neu}})) = \emptyset \quad \Leftrightarrow$$

XPath Queries aus $\text{ReadSet}(T_{\text{neu}})$ + XPath Queries aus log. $\text{WriteSet}(T_{\text{neu}})$
angewendet auf phy. $\text{WriteSet}(T_{\text{alt}}) = \emptyset$



Eindeutige XPath Bestimmung

C12 XPath = /BookingService/Connections/Connection[@id='3']/destination/*

ΔXML Fragmente = Paris

Basis Algorithmus:

- füge XPath Platzhalter ein.
- folge Parent-Axe bis Wurzel
- für jeden Knoten auf dem Weg:
 - suche eindeutiges Merkmal das von Geschwistern abgrenzt
 - nehme Merkmal als Filter in XPath auf.

Server:

- Hintereinanderausführung von
 - XPath für Read-Fragment
 - XPath für Write-Fragment

```
<BookingService>
  <Connections>
    <Connection id="1">
      <destination>Paris</destination>
      <departure>London</departure>
    </Connection>
    <Connection id="2">
      <destination>Paris</destination>
      <departure>Hamburg</departure>
    </Connection>
    <Connection id="3">
      <destination>Rom</destination>
      <departure>Hamburg</departure>
    </Connection>...
  </Connections> ...
</BookingService>
```



Beispiel

Client1 (neu)

- read(XPath1=...[./destination = 'Paris'])
- count(#connection)
- **validate:**

Client2 (alt)

- read(XPath2=...[./departure = 'Hamburg'])
- update(destination: **Rom**→**Paris**)
- validate → commit
- write

Zeit



Client2 XPath = /BookingService/Connections/Connection[@id='3']/destination/*
 Δ XML Fragmente = Paris

ReadSet(T_{neu}) = XPath1

log. WriteSet(T_{neu}) = \emptyset

phy. WriteSet(T_{neu}) = \emptyset

ReadSet(T_{alt}) = XPath2

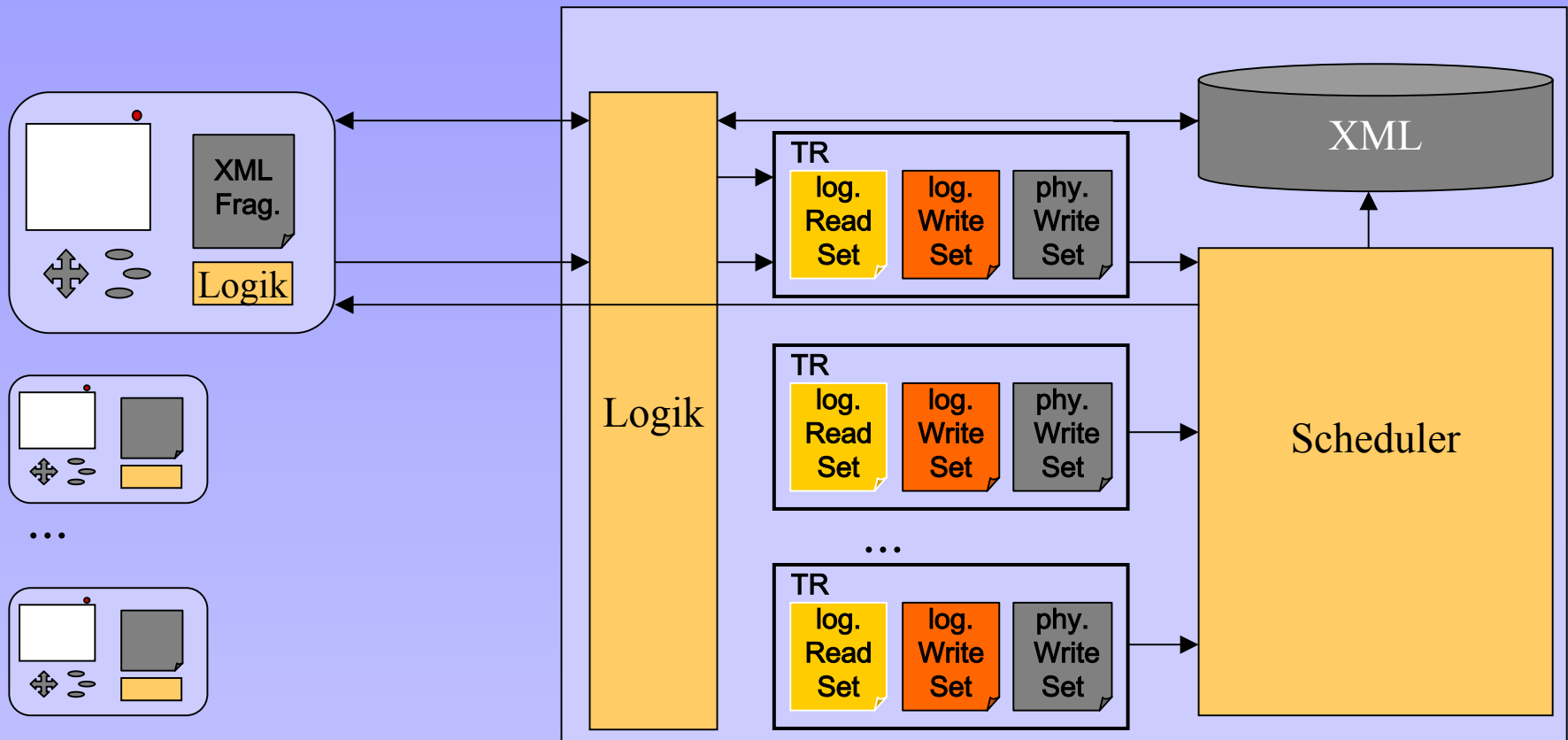
log. WriteSet(T_{alt}) = XPath2 \cap Cl2 XPath

phy. WriteSet(T_{alt}) \cong XPath2 \cap Cl2 XPath
 + Δ XML Fragment

XPath Queries aus ReadSet(T_{neu}) angewendet auf phy. WriteSet(T_{alt}) = \emptyset ?



Ressourcenbetrachtung



Logik des Servers:

Verwaltung der Sets

Logik des Clients:

Eindeutige XPath Pfade finden (einfach)

Kommunikationsbetrachtung

- Read-Operation:
 - Nur benötigte Fragmente des kompletten XML Dokumentes senden
- Delete-Operation:
 - Nur logischen Ausdruck senden
- Update-Operation:
 - Es werden nur neue Werte kommuniziert

⇒ Kommunikationsaufwand optimiert !



Lost Connection

Vorteil optimistischer Synchronisation

- Abbrüche beeinflussen andere Transaktionen nicht
- Offline auf aktuellen Daten arbeiten

Wiederaufnahme der Verbindung:

- **Vor Beginn der Validierungsphase**
 - Transaktion verwerfen und Daten neu lesen
 - Ohne Spezialbehandlung fortfahren
 - **Intermediate Validation** (kein oder Δ Datentransfer nötig)
- **Nach Beginn der Validierungsphase**
 - Status nach Reconnect mitteilen



Einsatzspektrum

- Szenario:
 - Lose gekoppelte Client/Server XML Transaktionen
 - Geringe Bandbreite
 - Hohe Datenvolumen-Kosten
 - Abbrüche
 - Geringe Clientanforderungen
- Mögliche Implementation:
 - Webservice für XML Transaktionen
 - Spezifische Ausprägungen
 - Z.B. für Clients mit sehr kleinem Hauptspeicher
 - ...



Nachladestrategien in Read-Phase

Fragestellung:

- Wie erkennt Server welches XML-Fragment dem Client fehlt?
- Muss der Client XPath-Anfrage überhaupt stellen?
- Wie verdrängt der Client?

Lösung:

- Identische Abbildung des Clientspeichers im Server (\neq ReadSet!)
- Fixe Größe des Speichers
- Datenstruktur identifiziert fehlende Fragmente
- Merkmal für Verdrängung und TR-Zugehörigkeit
- Client und Server benutzen gleiche Verdrängungsstrategie
- Operationen des Clients werden vom Server nachvollzogen



Speicher Recycling

- Test des Clients:
 - Vollständigkeit
 - Transaktionszugehörigkeit
- Szenarien:
 - Beide Bedingungen des Test erfüllt ⇒ keine Serverinteraktion nötig
 - Vollständigkeit nicht erfüllt ⇒ Δ XML nachladen
 - Daten aus alten Transaktionen ⇒ falls veraltet Δ XML nachladen
- Nur wenn keine Knoten bereits im Speicher sind keine Optimierung



Zusammenfassung

- ✓ Validierungstest: nur XPath Query Auswertung
- ✓ XPath Ausdrücke nur gegen kleine XML Fragmente testen
- ✓ Informationsoptimale Nutzung der Verbindung
- ✓ Verwaltung von XPath Listen im Hauptspeicher möglich
- ✓ Robust gegen Unterbrechungen
- ✓ Client braucht kein XPath oder XML auswerten können
- ✓ Transaktionszusicherung: Serializable
- ✓ Wiederverwendung über Transaktionsgrenzen hinaus



Ausblick

- XML Differenztests
- Transparente Nutzung über MDOM_{TR}
- Verdrängungsstrategien
- Preload Heuristiken
- Writeback Heuristiken



Transaction Validation for XML Documents based on XPath

@ Informatik 2002, m-dbis



Stefan Böttcher
Adelhard Türling

Universität Paderborn