Transaction Validation for XML Documents based on XPath

@ Informatik 2002, m-dbis

Stefan Böttcher
Adelhard Türling

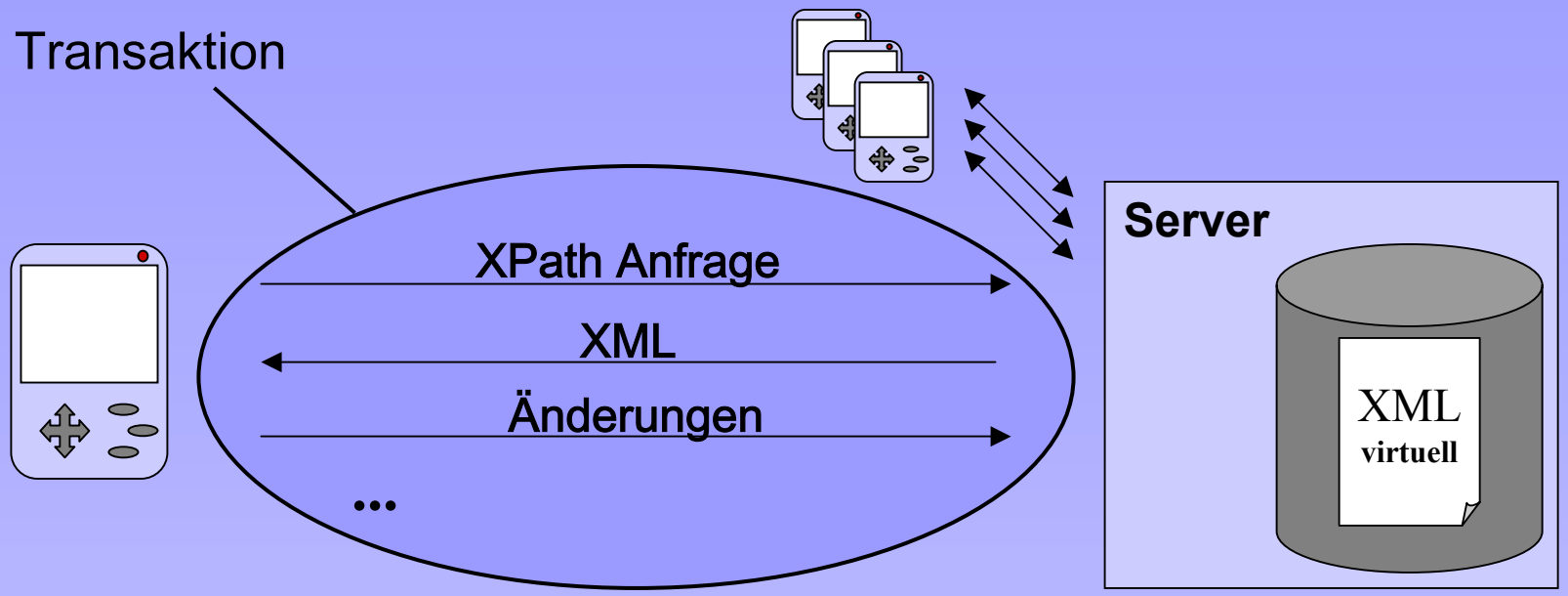
Universität Paderborn

Überblick

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 Cashing
- Zusammenfassung
- Ausblick

Szenario



Client:

- XPath Queries
- Arbeitet auf virtuellem XML

Server:

- Liefert virtuelles XML Dokument
- Synchronisiert parallele Zugriffe

← Standard Transaktionszusicherungen →

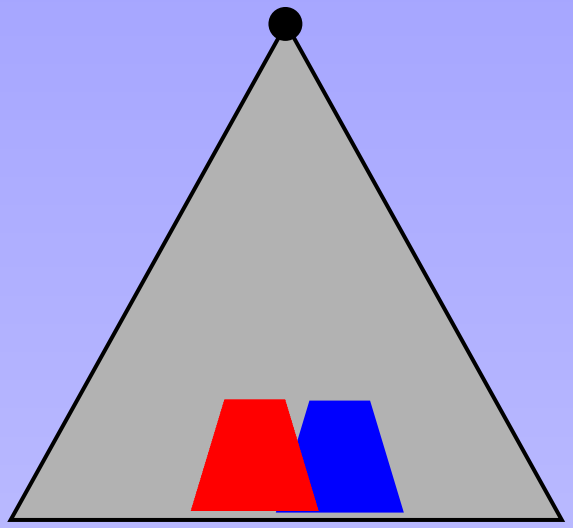
Voraussetzungen - Ansatz

Mobiler Client:		Anforderung:
• Lose Kopplung	→	Robust gegen Abbrüche
• Offline-Phasen	→	Unabhängigkeit
• **Bandbreite/Volumen**	→	**Kommunikation minimieren**
• Geringe Speicherressource	→	Datenmanagement/Caching
• Geringe Rechenressource	→	Last an Server abgeben

Lösung:

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

XPath & XML Fragmente



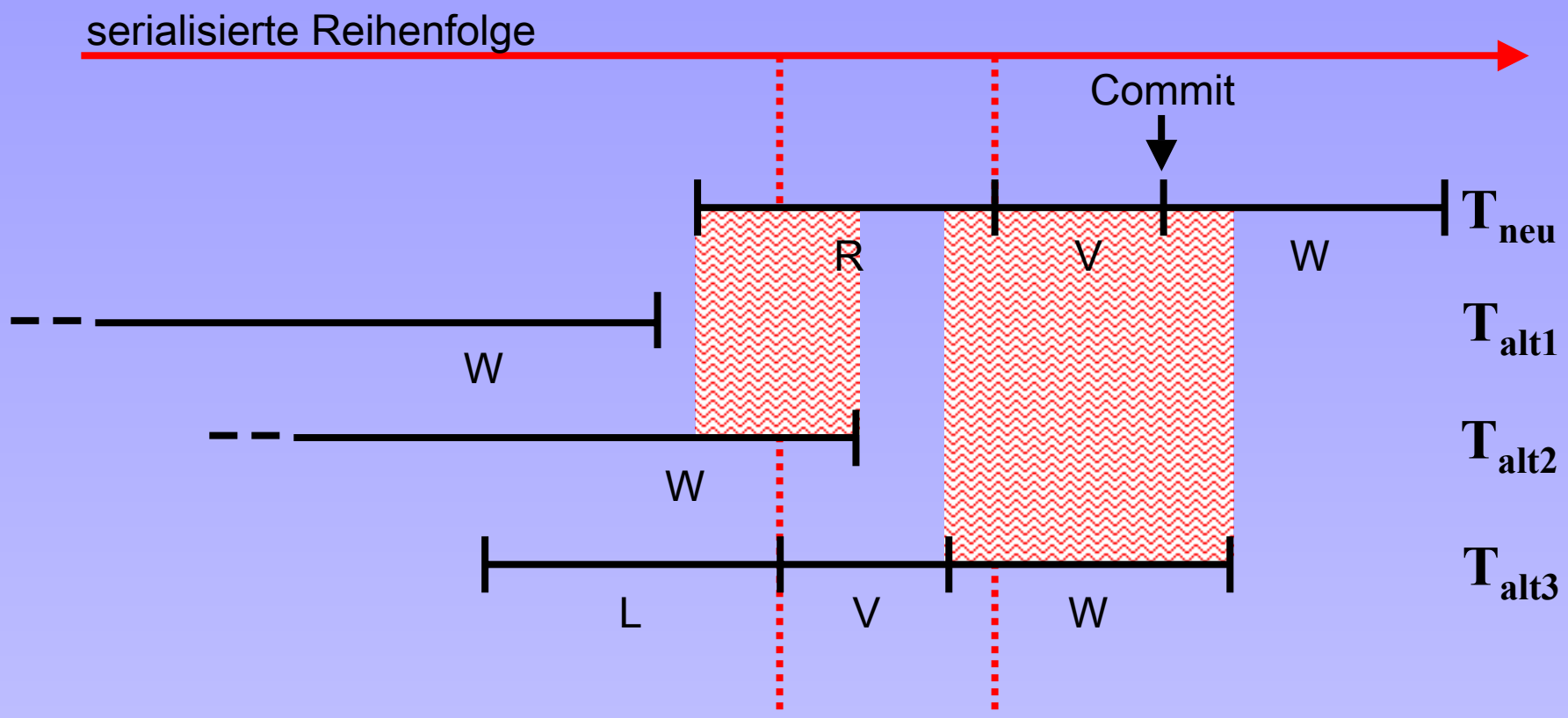
Virtuelles XML Dokument

```
<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']

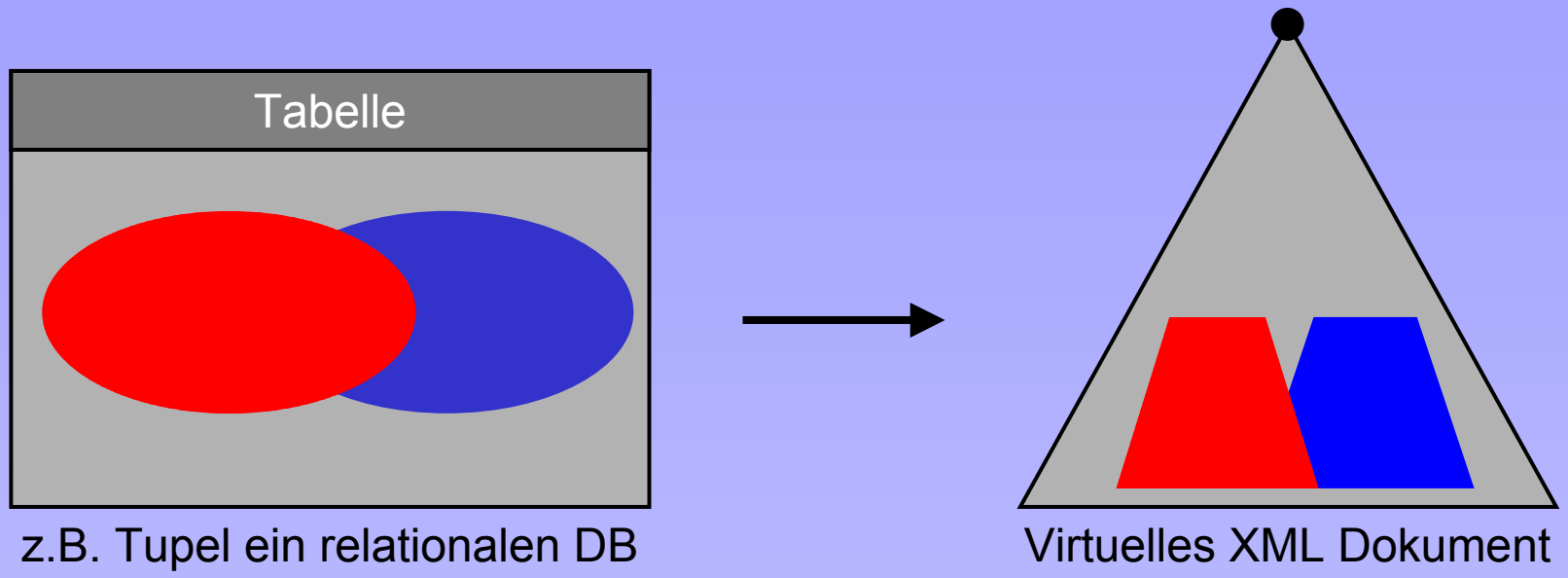
Parallel Validieren



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

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

XML Validierung



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

Mobiles XML Validieren

Client:

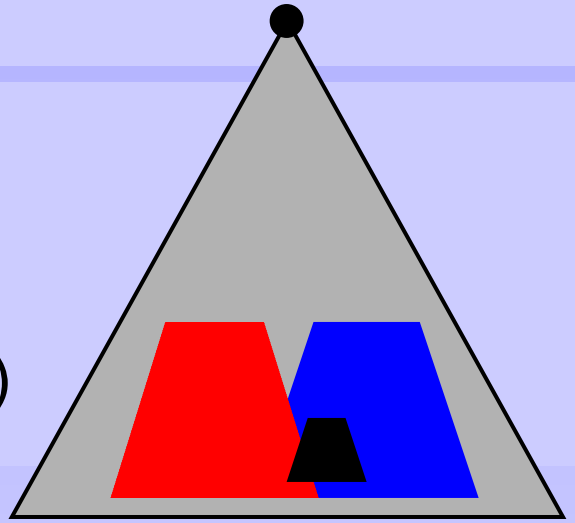
- Hält die physikalische Kopie des XML Fragments
- Änderungen auf Kopie

⇒ Kommuniziert XPath an Stelle von XML Dokumenten

⇒ **Prädikative Validierung mit XPath**

Server:

- Logisches ReadSet (XPath)
- Logisches WriteSet (XPath)
- Physikalisches WriteSet (ΔXML Daten)



ΔXML Fragmente

Read Phase

Client

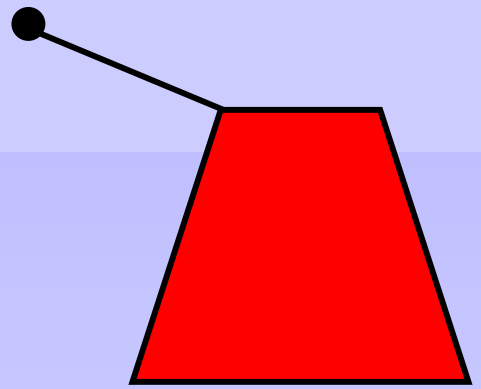
- Read:
  - XPath Queries →
  - XML Fragment ←
- Insert/Update:
  - XPath (eindeutig*) →
  - Δ XML Fragmente →
- Delete:
  - XPath (eindeutig*) →

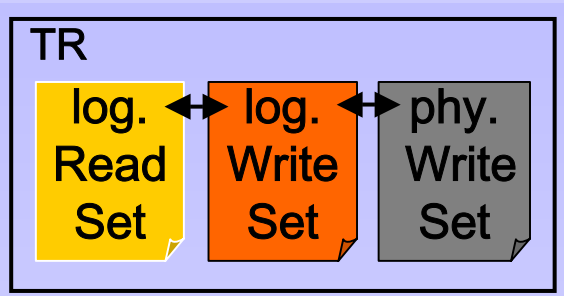
Server

- 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**

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

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

- **Write/Write Konflikte**

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

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

Eindeutige XPath Bestimmung

Cl2 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)	Client2 (alt)
• read(XPath1=...[./destination = 'Paris']) • count(#connection) • validate:	• read(XPath2=...[./departure = 'Hamburg']) • update(destination: Rom→Paris) • validate → commit • write

Zeit

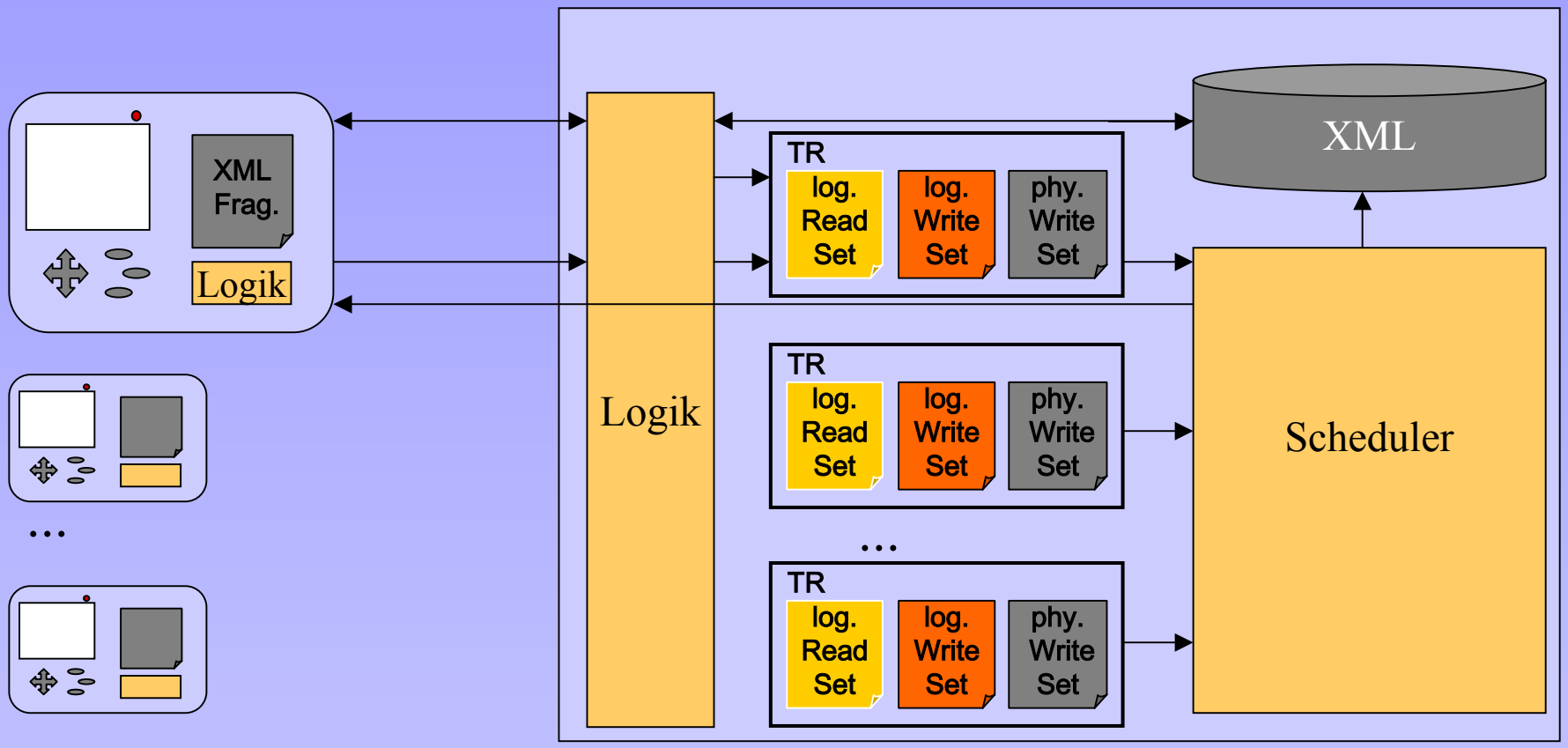
Client2 XPath = /BookingService/Connections/Connection[@id='3']/destination/*
ΔXML Fragmente = Paris

ReadSet(T_{neu}) = XPath1
log. WriteSet(T_{neu}) = ∅
phy. WriteSet(T_{neu}) = ∅

ReadSet(T_{alt}) = XPath2
log. WriteSet(T_{alt}) = XPath2 ∩ Cl2 XPath
phy. WriteSet(T_{alt}) ≙ XPath2 ∩ Cl2 XPath
+ Δ XML Fragment

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

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 (≠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 ⇒ ΔXML nachladen
  - Daten aus alten Transaktionen ⇒ falls veraltet ΔXML 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