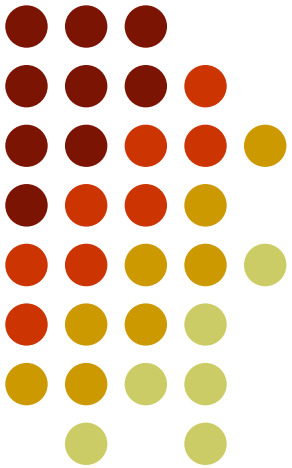
Metadatenverwaltung und kontextbasierte Personalisierung in verteilten Systemen

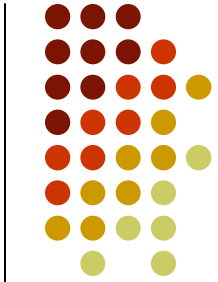
Doktorandenkolloquium

Universität Passau
Fakultät für Informatik und Mathematik
Lehrstuhl für Dialogorientierte Systeme

Markus Keidl

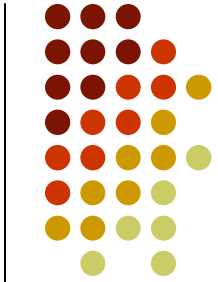


Gliederung



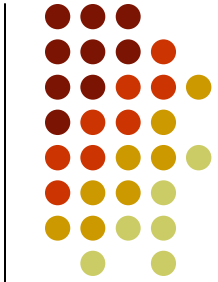
- Motivation und Einleitung
- Die Metadatenverwaltung MDV
- Die Web-Service-Plattform ServiceGlobe
  - Dynamische Dienstauswahl
  - Kontextbasierte Personalisierung von Web-Services
- Zusammenfassung

Motivation



- Neuartige Herausforderungen durch weltweite Netzwerke für Informationssysteme
  - Große, heterogene Menge von
    - Ressourcen (Daten, Funktionen, Rechenleistung),
    - Benutzern,
    - Gerätetypen
  - Pervasive Computing:
    Trend zu mobilen und ständig vernetzten Geräten
- Projekte:
  AutO, ObjectGlobe und ServiceGlobe

Einleitung

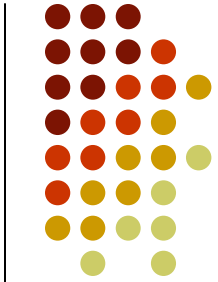


- AutO
  - Verteiltes System autonomer Objekte
  - Selbstoptimierendes System durch Datenmigration
- ObjectGlobe
  - Verteiltes Anfragesystem mit mobilen Operatoren
  - MDV: Verteilte Verwaltung von Metadaten über verfügbare Ressourcen
- ServiceGlobe
  - Web-Service-Plattform für mobile Web-Services
  - Dynamische Dienstauswahl
  - Kontextbasierte Personalisierung von Web-Services

Die Metadatenverwaltung MDV

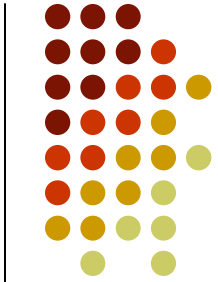
- Verteiltes System zur Verwaltung von beliebigen Metadaten
- Metadatenformat: RDF und RDF-Schema
- Eigenschaften:
  - 3-Schichten-Architektur: Ö-MDVs, L-MDVs und MDV-Klienten
  - Caching auf mittlerer Schicht (durch L-MDVs)
  - Aktualität durch Publish&Subscribe-Mechanismus
  - Deklarative Anfragesprache

RDF und RDF-Schema



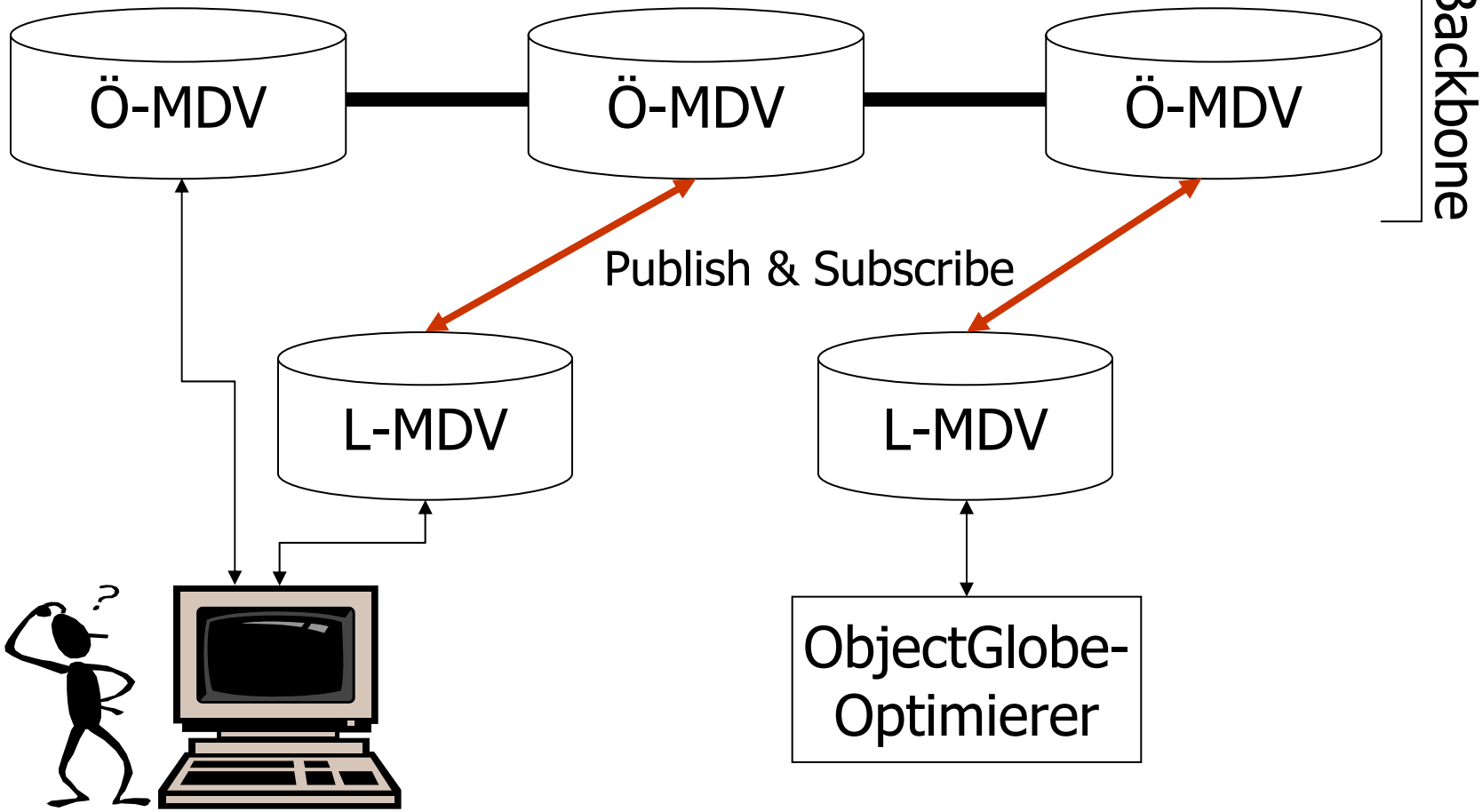
- RDF = Resource Description Framework
  - W3C Recommendation
  - Definiert Ressourcen, Eigenschaften und Werte
- RDF-Schema
  - W3C Working Draft
  - Definiert das Schema von Metadaten, ähnlich einer Klassenhierarchie

Beispiel für RDF-Metadaten

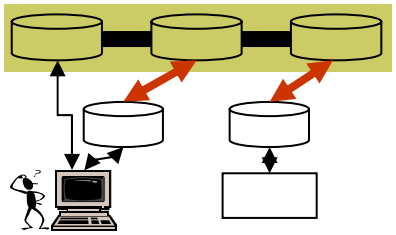


```
<CycleProvider rdf:ID="host1">
  <serverHost>pirates.uni-passau.de</>
  <serverPort>5874</serverPort>
  <serverInformation>
    <ServerInformation rdf:ID="info1"
        memory="92" cpu="600" />
  </serverInformation>
</CycleProvider>
```

Architektur der MDV

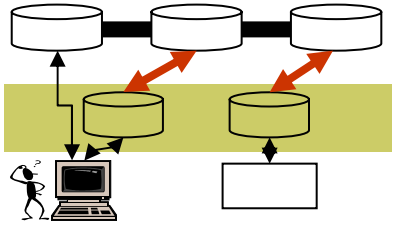


Architektur - Ö-MDVs



- Ö-MDV: Öffentliche Metadatenverwaltung
  - Öffentliche Metadaten: global und frei verfügbar
  - Backbone aus Ö-MDVs
  - Vollständige Replikation der öffentlichen Metadaten im Backbone
- Registrieren, Ändern und Löschen von Metadaten (Dokumenten-basiert)

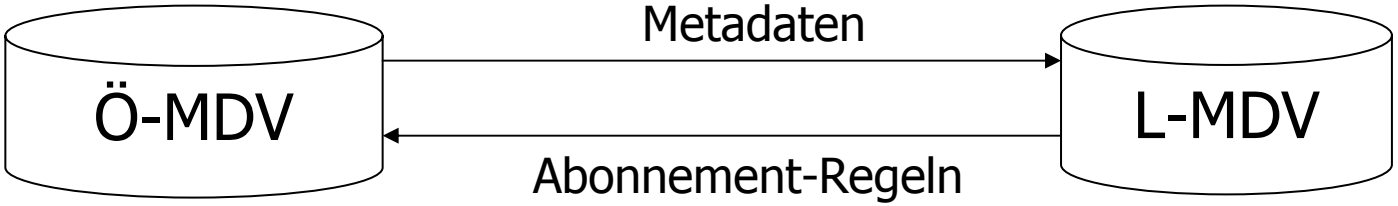
Architektur - L-MDVs



- L-MDV: Lokale Metadatenverwaltung
  - Caching von öffentlichen Metadaten
  - Speichern von lokalen Metadaten
    $\Rightarrow$ Nur lokal verfügbar, für sensitive Daten
  - Liegen nahe bei den Klienten
- Anfrageauswertung findet bei L-MDVs statt:
  - Anfrage von gepufferten und lokalen Metadaten

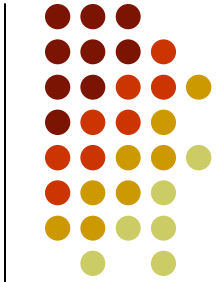
Das Publish & Subscribe-System

- Basiert auf Abonnement-Regeln:
  - L-MDVs abonnieren Metadaten (von Ö-MDVs)
  - Ö-MDVs bestimmen anhand der Regeln, welche Metadaten zu L-MDVs gesendet werden müssen



- Registrieren, Ändern oder Löschen von Metadaten erfordert Auswertung der Regeln

Abonnement-Regel-Sprache

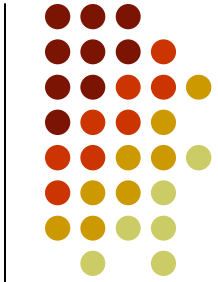


- Gezieltes Abonnement von bestimmten Ressourcen
- Beispiel:

```
search     CycleProvider c
register   c
where      c.serverHost contains 'uni-passau.de'
           and c.serverInformation.memory > 64
```

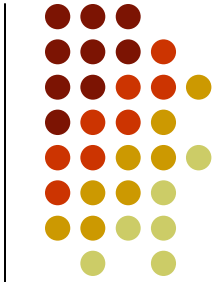
- Auswertungsgrundlage sind alle öffentlichen Metadaten
- Joins/Verbunde werden unterstützt

Der Publish & Subscribe - Algorithmus



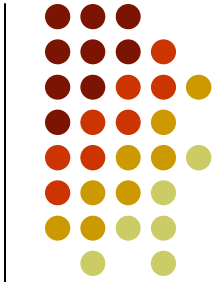
- Problem:
  - Riesige Menge von Abonnement-Regeln
- Lösung:
  - Index auf die Regelmenge
  - Auswertung einer **Teilmenge** der Regeln
- Algorithmus basiert auf bekannten, bewährten RDBMS-Technologien
  - Robustheit, Skalierbarkeit, Anfragefähigkeiten
  - Verwendung von Tabellen, Indexen, der SQL-Anfragesprache, dem Optimierer, …

Der Algorithmus im Detail



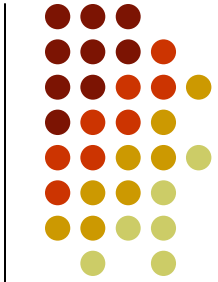
- Zerlegung:
  - RDF-Dokument
  - Regeln: Zerlegung in Trigger- und Join-Regeln
- Auswertung:
  - Ermittlung der betroffenen Trigger-Regeln
  - Iterative Auswertung der Join-Regeln

Zerlegung: RDF-Dokument



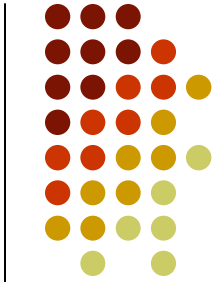
```
<CycleProvider rdf:ID="host1">
  <serverHost>pirates.uni-passau.de</>
  <serverPort>5874</serverPort>
  <serverInformation>
    <ServerInformation rdf:ID="info1"
            memory="92" cpu="600" />
  </serverInformation>
</CycleProvider>
```

Zerlegung: Tabelle



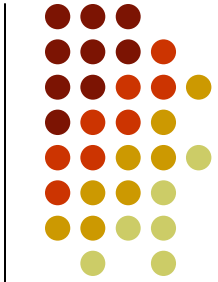
rid	class	property	value
doc.rdf#host1	CycleProvider	rdf#subject	doc.rdf#host1
doc.rdf#host1	CycleProvider	serverHost	pirates.uni-passau.de
doc.rdf#host1	CycleProvider	serverPort	5874
doc.rdf#host1	CycleProvider	serverInformation	doc.rdf#info1
doc.rdf#info1	ServerInformation	rdf#subject	doc.rdf#info1
doc.rdf#info1	ServerInformation	memory	92
doc.rdf#info1	ServerInformation	cpu	600

Der Algorithmus im Detail



- Zerlegung:
  - RDF-Dokument
  - Regeln: Zerlegung in Trigger- und Join-Regeln
- Auswertung:
  - Ermittlung der betroffenen Trigger-Regeln
  - Iterative Auswertung der Join-Regeln

Zerlegung: Regeln

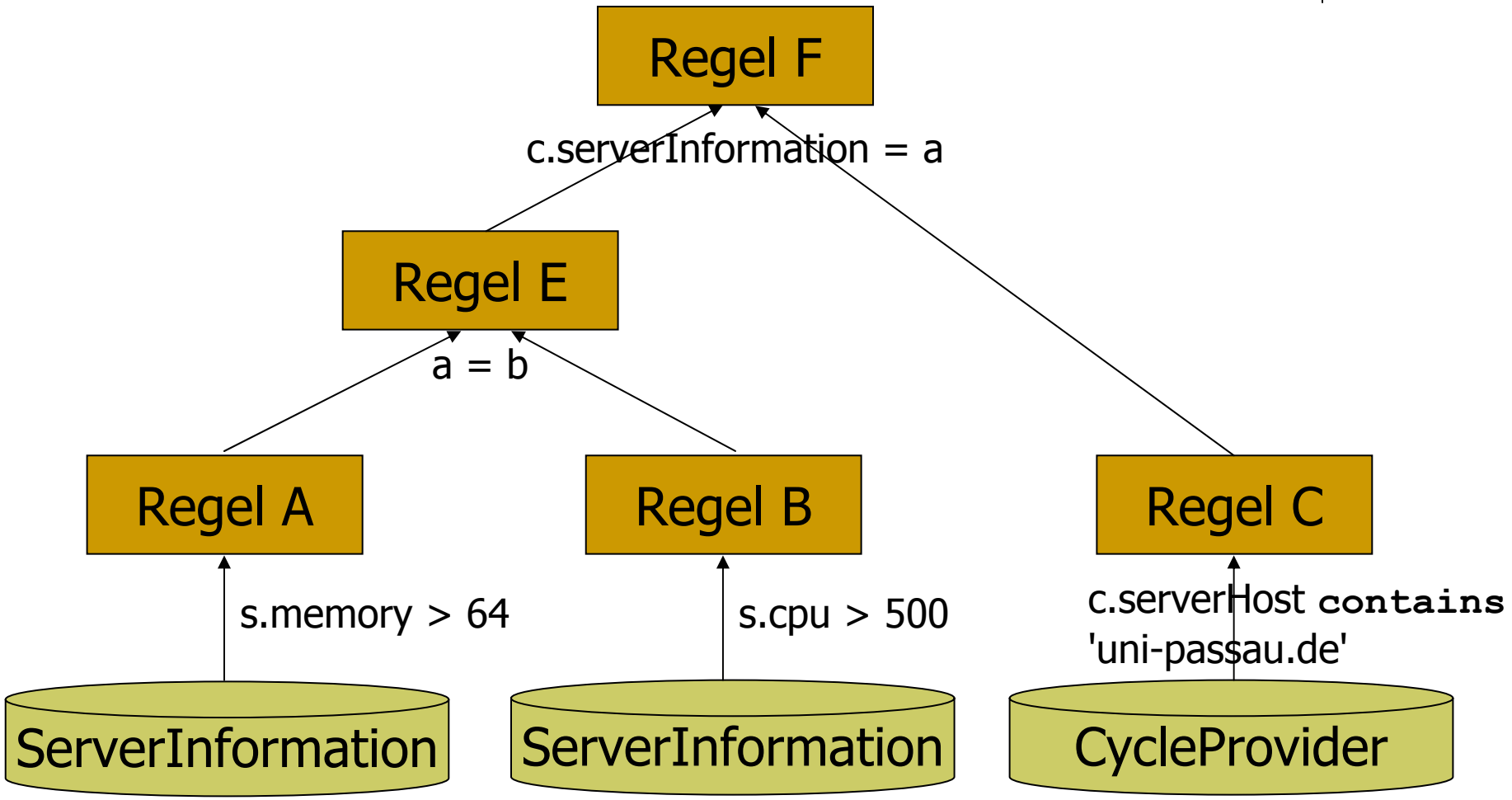


- Normalisierung der Regel
- Beispiel:

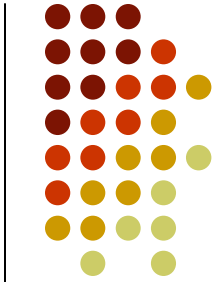
```
search    CycleProvider c, ServerInformation s
register  c
where     c.serverHost contains 'uni-passau.de'
          and c.serverInformation = s
          and s.memory > 64 and s.cpu > 500
```

- Zerlegung in Teilregeln: Trigger-Regeln und Join-Regeln

Zerlegung: Regel - Abhängigkeitsbaum



Der Algorithmus im Detail



- Zerlegung:
  - RDF-Dokument
  - Regeln: Zerlegung in Trigger- und Join-Regeln
- Auswertung:
  - Ermittlung der betroffenen Trigger-Regeln
  - Iterative Auswertung der Join-Regeln

Auswertung: Trigger-Regeln bestimmen

rid	class	property	value
doc.rdf#host1	CycleProvider	rdf#subject	doc.rdf#host1
doc.rdf#host1	CycleProvider	serverHost	pirates.uni-passau.de
doc.rdf#host1	CycleProvider	serverPort	5874
doc.rdf#host1	CycleProvider	serverInformation	doc.rdf#info1
doc.rdf#info1	ServerInformation	rdf#subject	doc.rdf#info1
doc.rdf#info1	ServerInformation	memory	92
doc.rdf#info1	ServerInformation	cpu	600

Neue Metadaten

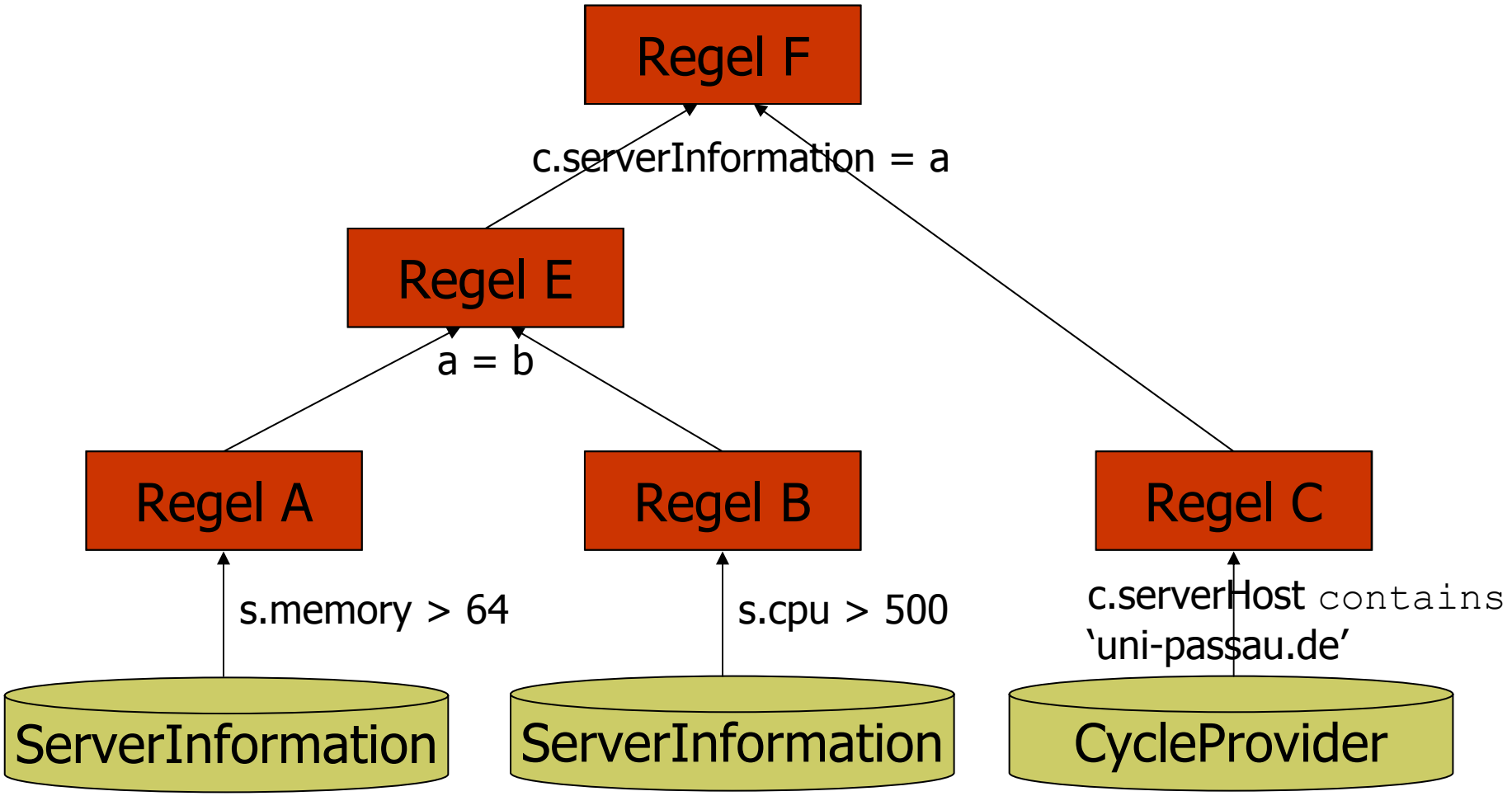
rule_id	class	property	value
Regel A	ServerInformation	memory	64
Regel B	ServerInformation	cpu	500

Trigger-Regeln mit Operator >

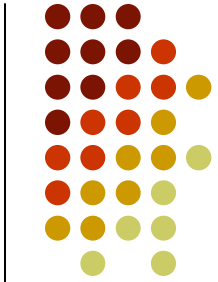
rule_id	rid
Regel A	doc.rdf#info1
Regel B	doc.rdf#info1

Ergebnis

Auswertung: Iterative Auswertung der Join-Regeln

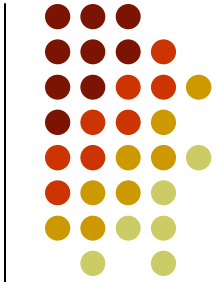


Funktionsweise des Algorithmus



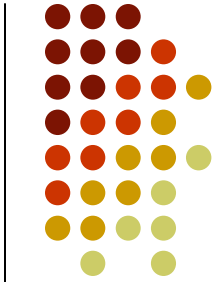
- Zerlegung der Abonnement-Regeln
- Einfügen von neuen Metadaten:
  - Zerlegung des RDF-Dokuments
  - Ausführung des Algorithmus liefert:
    - Regeln, deren Ergebnis sich ändert
    - Neue Metadaten im Ergebnis
  - Regeln bestimmen zu benachrichtigende L-MDVs
  - Benachrichtigung dieser L-MDVs

Löschen bzw. Ändern von Metadaten



- Mehrmaliges Ausführen des Algorithmus
- Jeweils mit unterschiedlichen Eingabedaten

Gliederung

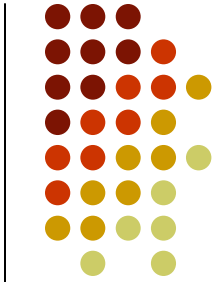


- Motivation und Einleitung
- Die Metadatenverwaltung MDV
- Die Web-Service-Plattform ServiceGlobe
  - Dynamische Dienstauswahl
  - Kontextbasierte Personalisierung von Web Services
- Zusammenfassung

Die Web-Service-Plattform ServiceGlobe

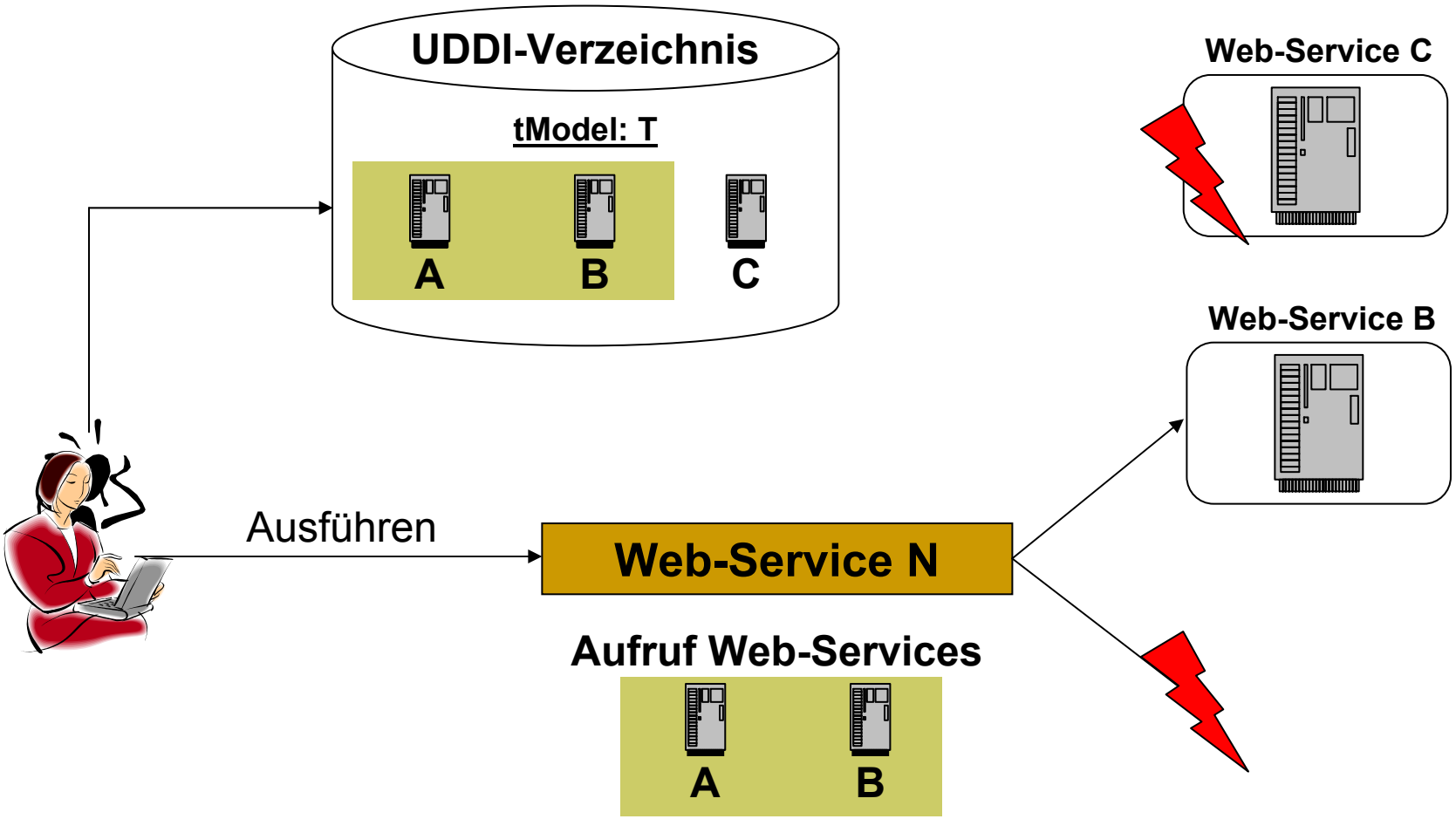
- Forschungsplattform für **mobile** Web-Services
  - Web-Services sind mobiler Code, d.h. sie können nahe bei Daten bzw. anderen Web-Services ausgeführt werden
  - Vollständig in Java implementiert
  - Basiert auf Standards wie XML, SOAP, UDDI, WSDL, …
- Standardfunktionalität für eine Dienstplattform:
  - Transaktionssystem
  - Sicherheitssystem ausgelegt auf mobilen Code
- Unterstützt die Entwicklung von personalisierbaren, flexiblen und zuverlässigen Web-Services

Dynamische Dienstauswahl

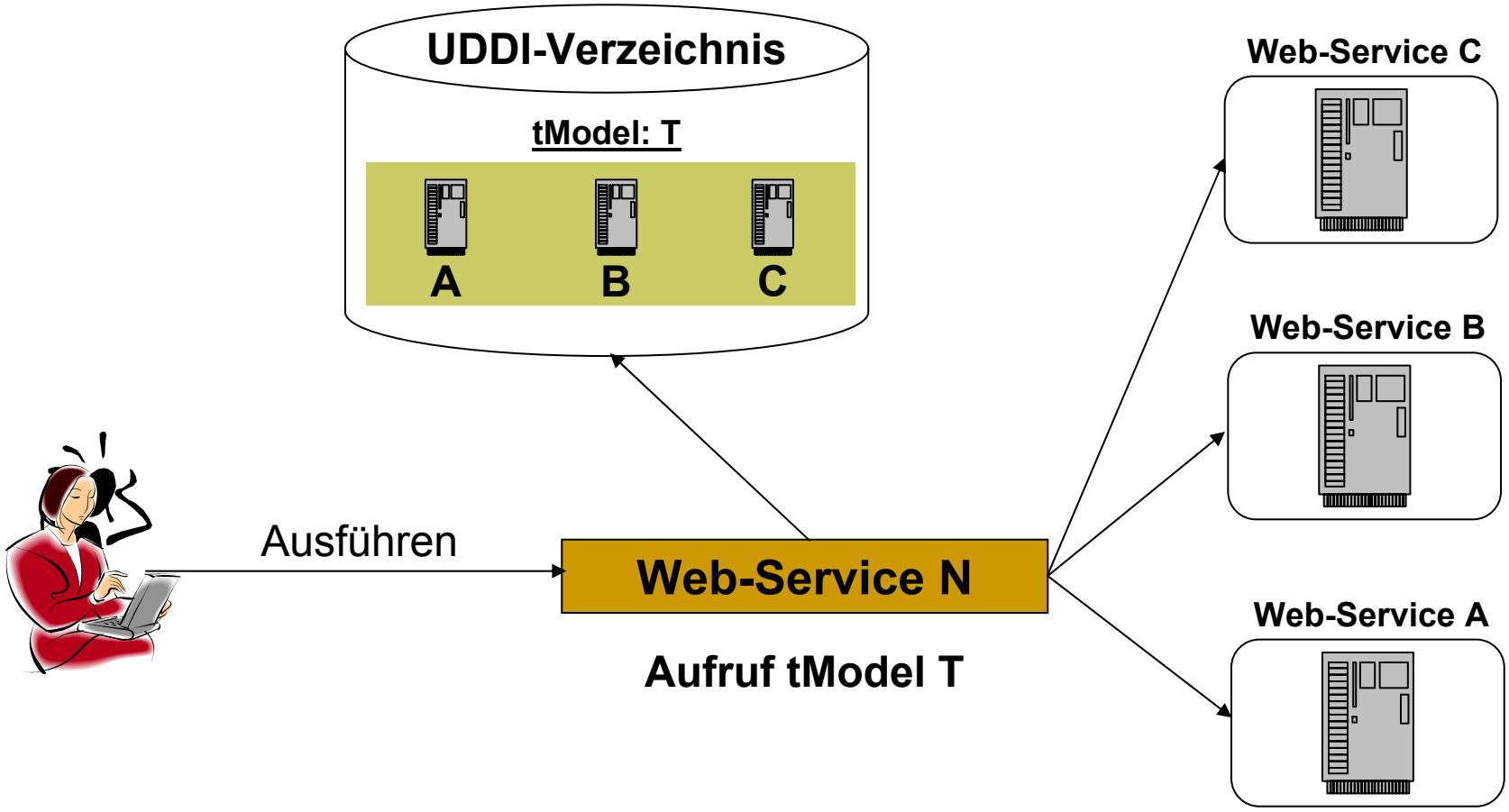


- Aufgabe
  - Dynamische Fusion von Web-Services
  - Robuste Ausführung von Web-Services
- Verwendung von UDDI:
  - Verzeichnis für Web-Services
  - UDDI: Zuordnung von Diensten zu tModels
  - tModel: Semantische Klassifikation der Funktionalität und formale Beschreibung der Schnittstelle

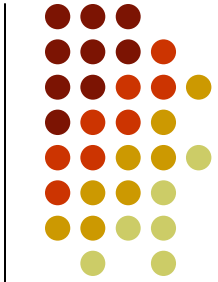
Dynamische Dienstauswahl – Motivation



Dynamische Dienstauswahl – Beispiel

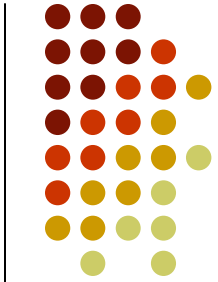


Dynamische Dienstauswahl



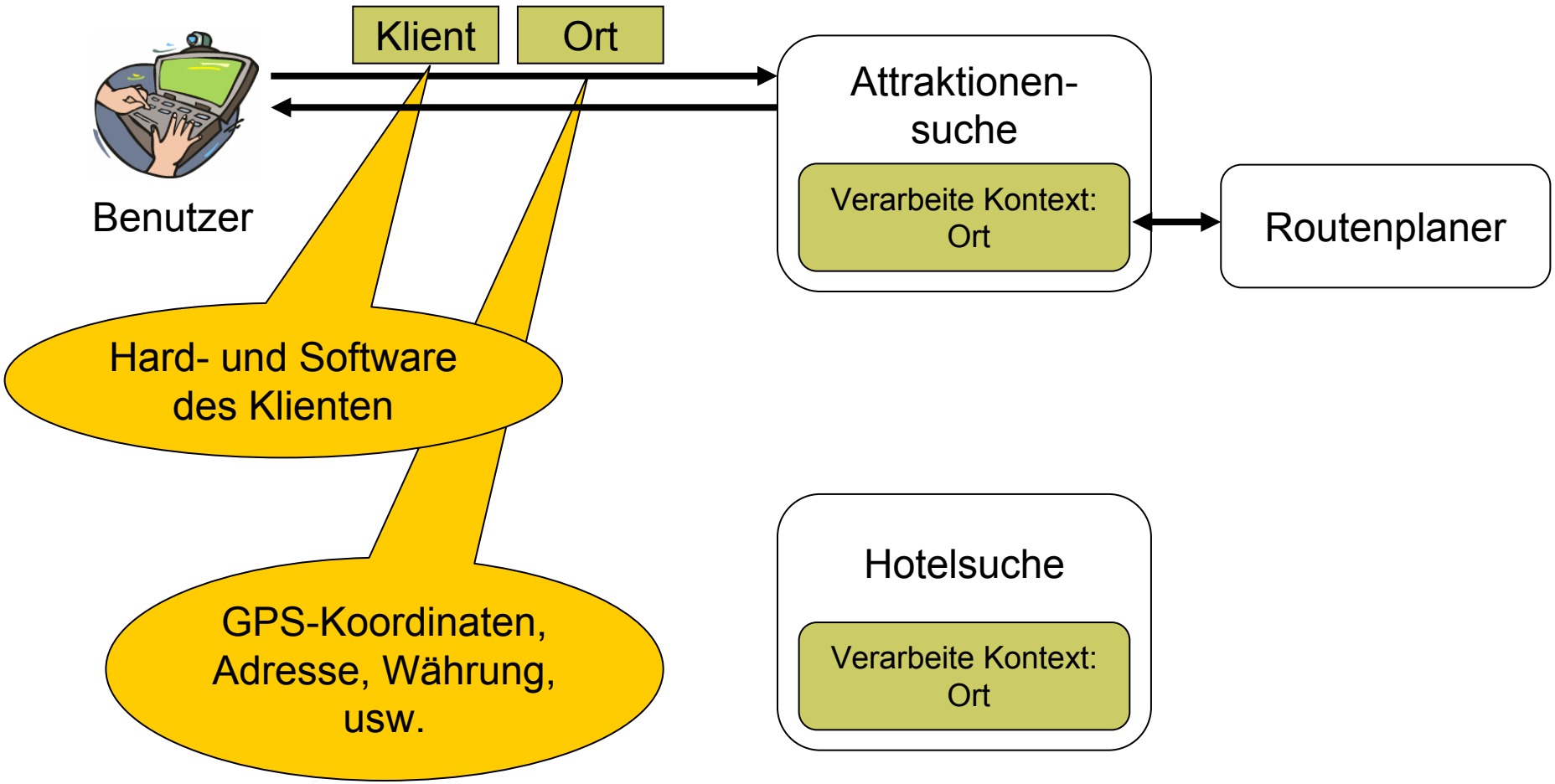
- Aufgabe
  - Dynamische Fusion von Web-Services
  - Verlässliche Ausführung von Web-Services
- Ergebnis
  - „Aufruf eines tModels“ anstatt des traditionellen „Aufruf eines Web-Services“
  - Abstraktion von tatsächlichen Web-Services in der Implementierung
  - Web-Services werden zur Laufzeit ausgewählt, nicht bereits während der Entwicklung
  - Vorgaben können verwendet werden, um Dienstauswahl und –aufruf zu beeinflussen

Kontextbasierte Personalisierung von Web-Services

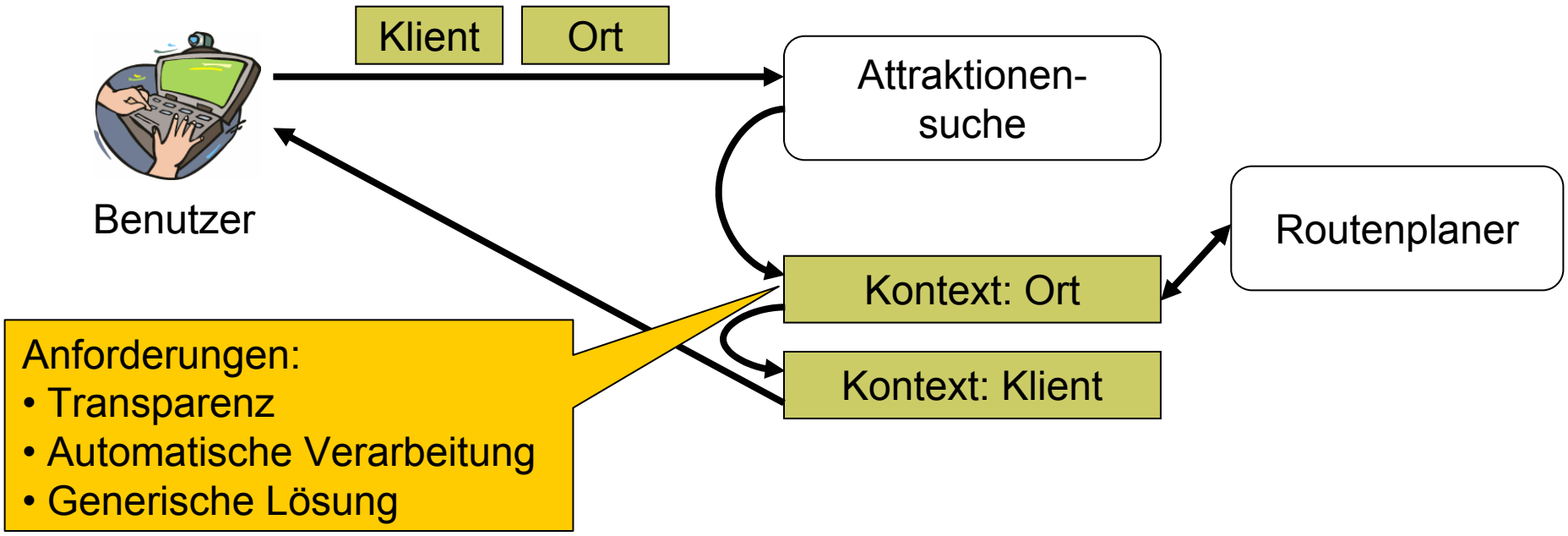


- Kontext:
  Informationen über Benutzer und ihre (aktuelle) Umgebung, die von Web-Services eingesetzt werden, um ein personalisiertes Verhalten zu erzielen
- Beispiele:
  - Informationen über Aufenthaltsort des Benutzers,
  - Art des Klienten, usw.

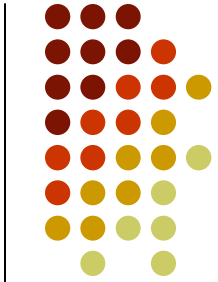
Beispiel-Szenario



Beispiel-Szenario

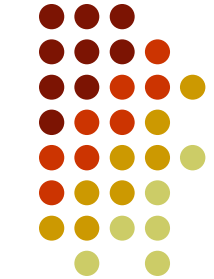


Das Kontext-Framework in ServiceGlobe



- Ziele und Eigenschaften:
  - Flexibel und zur Laufzeit erweiterbar
  - Automatische Auswertung von Kontext-Informationen, ohne Beteiligung des Web-Services selbst
  - Trennung der Kontext-Verarbeitung von den eigentlichen Web-Services
- Wesentliche Teile: Kontext-Infrastruktur und Kontext-Typen

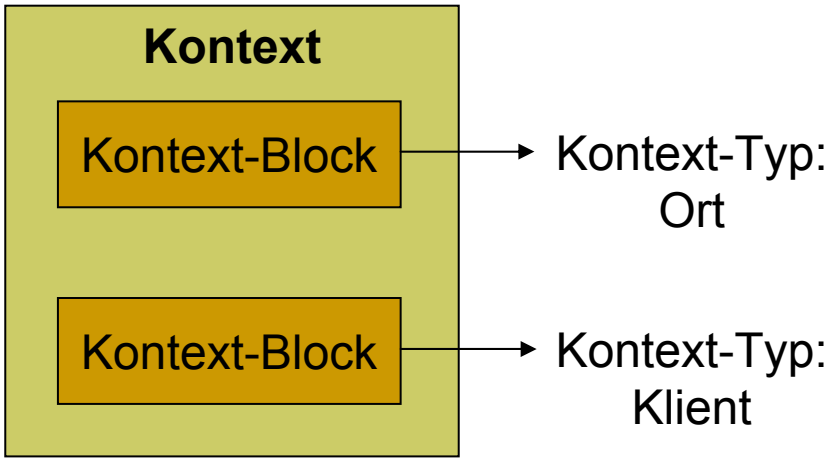
Die Kontext-Infrastruktur



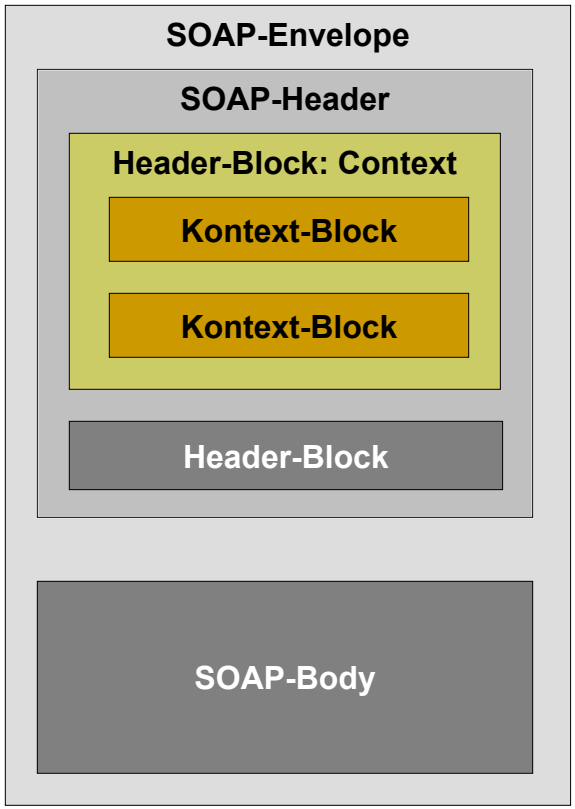
- Kontext-Modell
- Übertragung von Kontext zu den Web-Services
- Automatische Verarbeitung von Kontext
  - Arbeitsweise
  - Komponenten
- Verarbeitungsinstruktionen

Das Kontext-Modell

- Kontext besteht aus mehreren Kontext-Blöcken
- Ein Kontext-Block ist einem Kontext-Typ zugeordnet
- Ein Kontext-Typ definiert die Art der Kontextinformation, z.B. Aufenthaltsort, Klient
- Kontext darf max. einen Kontext-Block pro Kontext-Typ enthalten

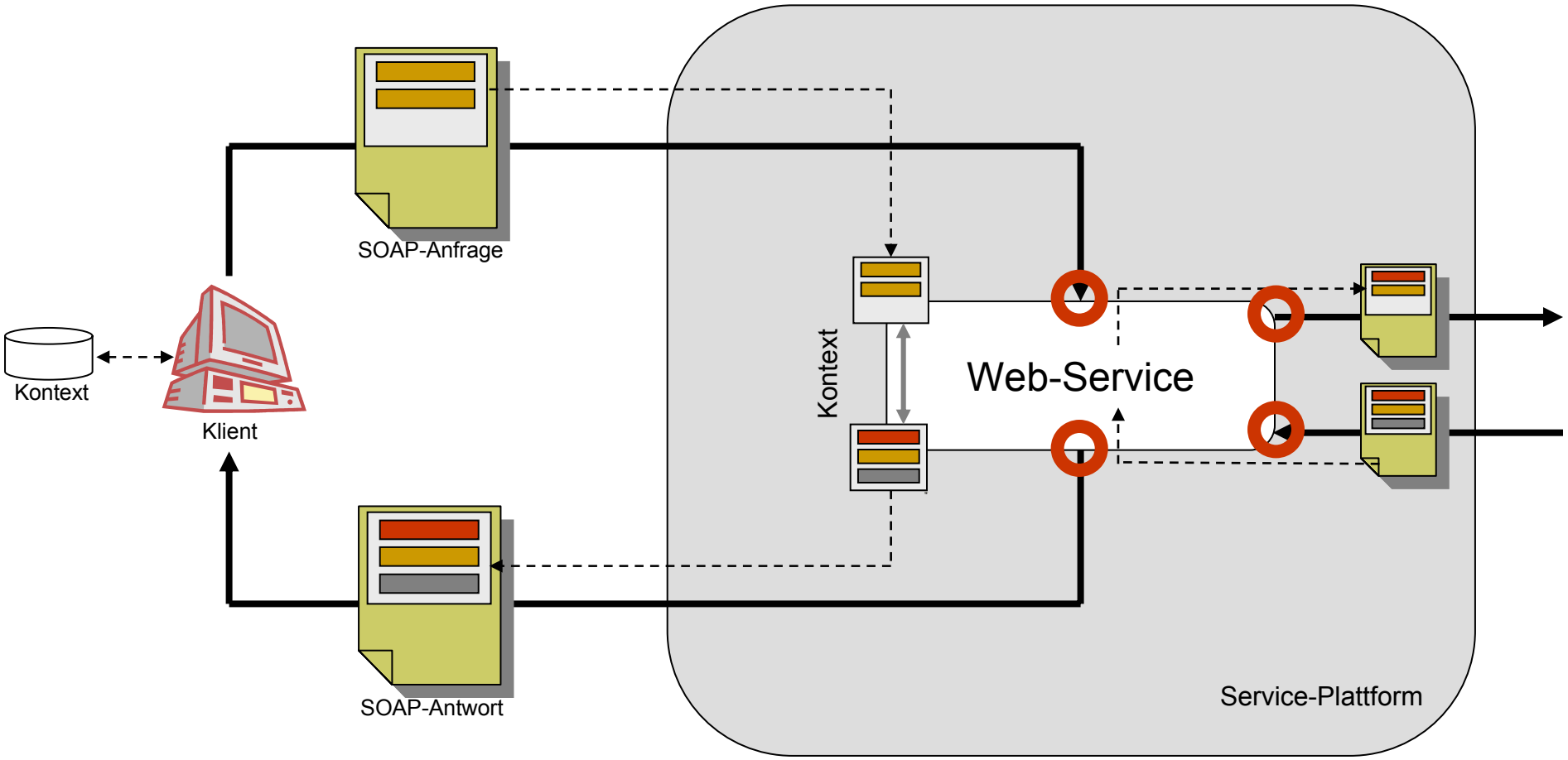


Kontext in einer SOAP-Nachricht

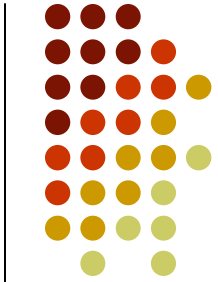



```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/.../">
  <env:Header>
    <Context
       xmlns="http://sg.fmi.uni-passau.de/context">
      <Client>
        <Hardware>
          <Defaults>
            http://example.com/PDA
          </Defaults>
          <ScreenSize>320x200</ScreenSize>
          <IsColorCapable>Yes</IsColorCapable>
        </Hardware>
      </Client>
    </Context>
  </env:Header>
  <env:Body>
    <!-- serialized object data -->
  </env:Body>
</env:Envelope>
```

Übertragung von Kontext

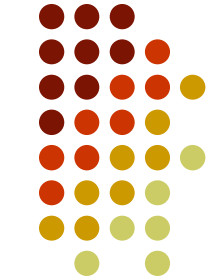


Automatische Verarbeitung von Kontext



- Kontext-Operationen:
  - Vorverarbeitung einer Web-Service-Anfrage
  - Nachbearbeitung einer Web-Service-Antwort
  - Nachbearbeitung von ausgehenden Nachrichten (Anfragen an andere Web-Services)
  - Vorverarbeitung von eingehenden Nachrichten (Antworten von anderen Web-Services)
- Komponenten:
  - Kontext-Plugins und Kontext-Services

Arbeitsweise: Automatische Kontext-Verarbeitung

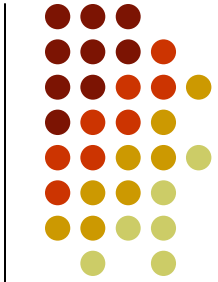


- Arbeitsweise der Kontext-Operationen:
  - Verarbeitung von Kontext-Blöcken in der SOAP-Nachricht in beliebiger Reihenfolge
  - Auswahl eines Kontext-Blocks
  - Aufruf von geeigneten Kontext-Plugins und Kontext-Diensten
- Kontext-Plugins und Kontext-Services:
  - Zuordnung zu genau einem Kontext-Typ
  - Eingabe: Kontext-Block und Nachricht
  - Ausgabe: Ggf. modifizierte Nachricht
  - Nachricht variiert je nach Kontext-Operation

Komponenten zur Kontext-Verarbeitung

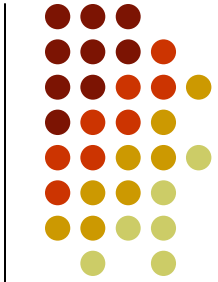
- Kontext-Plugins
  - Java-Objekte, die ein bestimmtes Interface implementieren
  - Installation auf lokalem Rechner
  - Verarbeitung von Kontext-Blöcken eines Kontext-Typs
  - Vorteile:
    - Generische Lösung für alle Web-Services
    - Transparente Verarbeitung von Kontext
    - Zugriff auf interne Datenstrukturen der Dienstplattform
  - Nachteil:
    - Lokale Installation notwendig

Komponenten zur Kontext-Verarbeitung



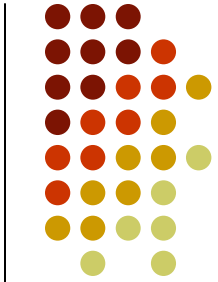
- Kontext-Dienste
  - Web-Services, die ein spezielles Interface implementieren
  - Verarbeitung von Kontext-Blöcken eines Kontext-Typs
  - Vorteile:
    - Irgendwo im Netz verfügbar
    - Ansonsten wie Kontext-Plugins
  - Nachteile:
    - Nur Zugriff auf Kontext und Web-Service-Nachricht

Komponenten zur Kontext-Verarbeitung



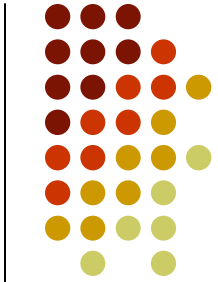
- Web-Service
  - Zugriff auf alle Kontext-Informationen
  - Modifikation des Kontextes
  - Vorteil:
    - Beeinflussung des internen Kontrollflusses
  - Nachteil:
    - Aufwändige Implementierung

Komponenten zur Kontext-Verarbeitung



- Klient
  - Verarbeitung von Kontext auf dem Endgerät, z.B. Formatierung XML nach HTML
  - Speichern von Kontext-Informationen

Verarbeitungsinstruktionen



- Herausforderungen:
  - Welche Komponenten sollen zur Verarbeitung der Kontext-Blöcke eingesetzt werden?
  - Auf welchen Rechnern sollen die Kontext-Blöcke Kontext verarbeitet werden?
- Lösung: Angabe von
  - Instruktionen für Kontext-Dienste und Kontext-Plugins
  - Verarbeitungsrichtlinien

  für jeden Kontext-Typ einzeln

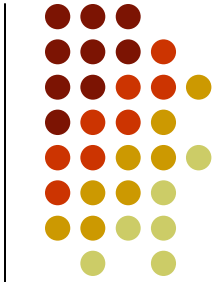
Verarbeitungsinstruktionen: Beispiel

```
<ContextProcessingInstructions>
  <ContextType ID="http://sg.fmi.uni-passau.de/context:Location">
    <ContextService>
      <AccessPoint useType="http">
        http://example.com/services/CurrencyConverter
      </AccessPoint>
      <ContextOperations>post</ContextOperations>
    </ContextService>
    <ProcessingGuideline>
      <ServiceHost>Next</ServiceHost>
      <Components>ContextPlugin+ContextService</Components>
    </ProcessingGuideline>
  </ContextType>
</ContextProcessingInstructions>
```

Verwendung der Instruktionen

- Im Kontext selbst:
  - Einfügen der Verarbeitungsinstruktionen als eigenen Kontext-Block
- In den UDDI-Metadaten eines Web-Services
- In UDDI allgemein:
  - Zuordnung von Kontext-Diensten zum tModel ihres Kontext-Typs
  - Web-Service-Plattform sucht nach passenden Kontext-Diensten

Kontext-Typen



- Klient: Hard- und Software des Klienten
  - Kontext-Plugin: Auswahl eines XSL-Stylesheets, passend zum Klient-Typ
  - In UDDI: Web-Service legt XSL-Stylesheets für verschiedene Klient-Typen fest
- Ort:
  - GPS-Koordinaten, Adresse, Zeit, Währung; aber auch: semantische Informationen, z.B. „in der Arbeit“
  - Kontext-Dienst: Konvertierung von Geldbeträgen in die Währung des aktuellen Aufenthaltsorts
- Kontakt-Daten, DSS-Vorgaben, …

Und damit …

Benutzer

Klient

Ort

Attraktionen-
suche

Routenplaner
Kontext-Dienst

Routenplaner

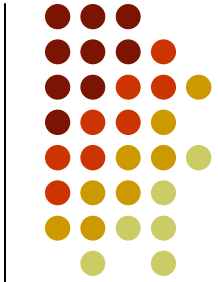
Währungsumrechner
Kontext-Dienst

Währungs-
umrechner

StylesheetFinder
Kontext-Plugin

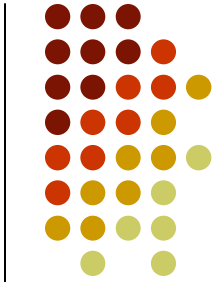
**UDDI-
Verzeichnis**

Zusammenfassung



- AutO: Verteiltes System autonomer Objekte
  [BTW'99]
- Das verteilte Anfragebearbeitungssystem ObjectGlobe
  - Die Metadatenverwaltung MDV
    [IEEE Data Engineering Bulletin 24(1), VLDBJ 10(3), BTW'01, ICDE'02, Informatik Forschung & Entwicklung 17(3)]
  - Authentifizierung und rollenbasierte Autorisierung
  - Verteilung der Autorisierungsinformationen mit Hilfe der MDV
- Die Web-Service-Plattform ServiceGlobe
  - Offene, verteilte Dienstplattform für mobile Web-Services
  - Dynamische Dienstauswahl
    [GI-Workshop'02, Demo VLDB'02, BTW'03, Poster WWW'03, TES'03]
  - Kontextbasierte Personalisierung von Web Services
    [BTW'03, Demo EDBT'04]

Ausblick



- Weitere Untersuchung von Verarbeitungsinstruktionen
- Kontext-Sicherheit:
  - Zugriffskontrolle
  - Datenschutz
- Weitere Kontext-Typen und entsprechende Einsatzszenarien

Vielen Dank für Ihre Aufmerksamkeit!