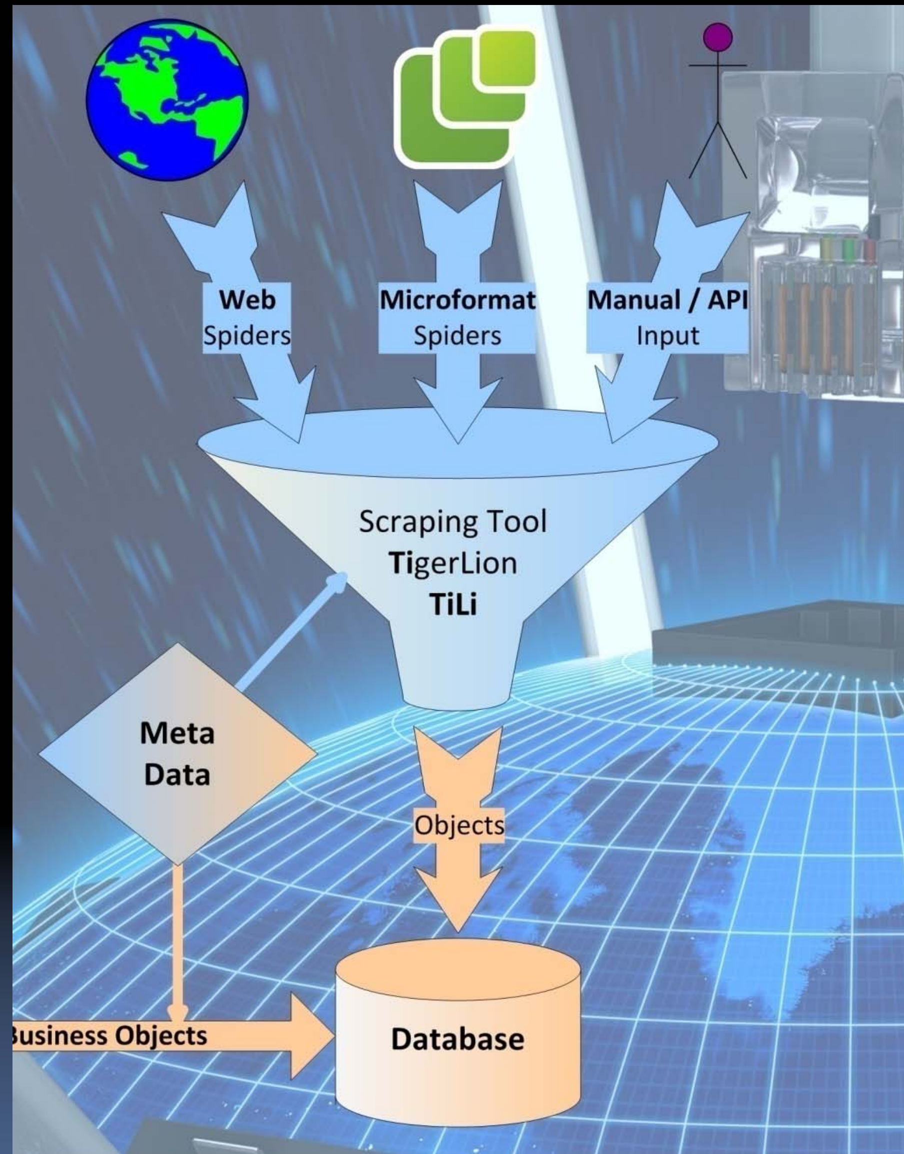
Prof. Dr. Clemens H. Cap
Universität Rostock
clemens .cap (at) uni-rostock (dot) de
www.internet-prof.de

Vortragender: Martin Garbe

GI Fachgruppe
Datenbanksysteme und InformationRetrieval
8. und 9. Mai 2008, Bamberg

SUCHE NACH OBJEKTEN

Von den Problemen des Semantic Web
zu den Chancen der JOPPA Architektur



Joppa – View

Standard Web Infrastructure plus Tools

JavaScript Object Persistent Programming Architecture

Probleme des Semantic Web

Zu schwergewichtig, weil

- Relativ komplexe Syntax (XML, Namespaces, vieles explizit)
- Komplexes Reasoning (F-Logik, OWL-*)
- Viele Co-Technologien (RDF, SparQL, RIF, RDFS)
- Hohe Anforderung an Meta-Info (Ontologie, Schemata)

Daher

- Erfolge vornehmlich in engeren Spezialbereichen
- Akzeptanz in "großer" Community geringer als Social Web / Web 2.0

Prognosen für die Zukunft

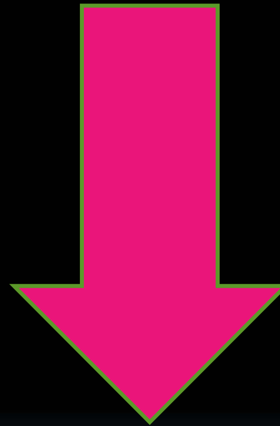
Das aktuelle Web 2.0 ist der letzte große Umbruch im Web mit

- User Generated Content zum "Socialnet"
- Anbindung der realen Außenwelt zum "Internet der Dinge"

Semantic Web Funktionen werden entstehen

- durch Socialising und Wisdom of Crowds
- in den Inhalten: bottom up
- weniger durch F-Logik, RDFs, SparQL, OWL
  so schade das wissenschaftlich gesehen ist !!
- vermutlich eher durch Microformate, Annotationen

Ausgangspunkt



Ausgangspunkt

{ class: "PKW",
age: {min:10, max:20} }

Einfache Metadaten: Objekte mit Attributen

Kleine Objekte: Bis 20 Attribute, komplett typisch 1kB

Einfachste Queries: **N**ach QBE Ansatz

- **Konjunktion** von **Value-**Queries und **Range-**Queries auf paarweise verschiedenen Attributen
- Ggf. noch **Bloomfilter** Queries (Zugehörigkeit zu Maske)

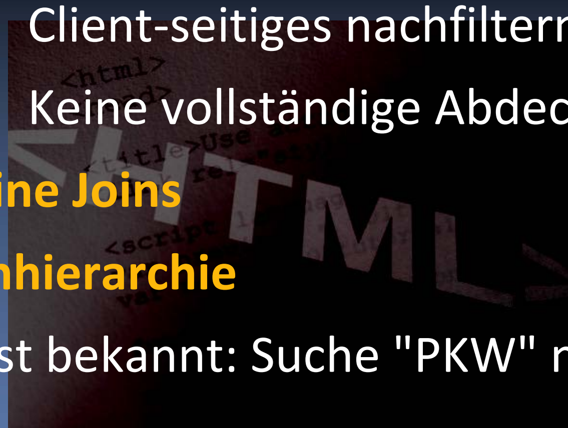
Antwort darf unpräzise sein

- Liefert zu viel: Client-seitiges nachfiltern
- Liefert zu wenig: Keine vollständige Abdeckung erwartet

Keine Relationen und keine Joins

Geringe Tiefe der Klassenhierarchie

- Most specific class meist bekannt: Suche "PKW" nicht "KFZ"



Ausgangspunkt
Beispiele

Meist Suche nach Objekten anstatt nach allgemeinen Dingen

eBay

- Temporale Restriktion: Beschränkte Gültigkeit
- Suche auf most specific class

Amazon

- Standard-Buchsuche (nicht komplexes IR)

Kinoprogramm / Restaurant / Mietwohnung

- Lokale und temporale Restriktion
- Kleine Objektanzahl

Js Obj

Js Business

Json DB

Js Obj

Js Business

UTE Template

Ähnliche Ansätze in der Literatur

Parallele Entwicklung ähnlicher Ideen bei anderen Gruppen

jsdb: Javascript DB

- nur Client-seitig

Aptana: Pure JS Development

- Jaxer: Serverside JS in SCRIPT Tag eingearbeitet, somit Spaghettis
- Keine Objektabtrennung

Persistent Javascript und Persevere

- **DB Ansatz stärker traditionell**

Vorteile von Js / Json

```
{ fname: "Clemens",
  lname: "Cap",
  lectures: ["net", "java", 11] }
```

- Am Client universell verfügbar
- Einfache Objektnotation Json – selber Js, daher nur "eval"
- OO (Prototypen-basiert, keine Typprüfung)
- Dynamische Attribute flexibler als in C++, Java
- Syntax leichtgewichtig
- Co-Technologien oftmals trivial (Bsp: JPath)
- Funktionen sind first class citizens (Speichern in Variable)
- Interpreter ist Teil des Sprachumfangs
- Stark steigende Akzeptanz der Community
- Als RFC 4627 standardisiert

Persistenz
Meta-Prinzip: Schwein ja, Müll nein

Persistenz
Meta-Prinzip: Wissenschaftlicher

Kein Anspruch an

- Algebraische Vollständigkeit
- Normalformen
- Effizienz für alle Queries
- Effiziente Joins
- Langfristige Archivierung
- Transaktionale Korrektheit
- Hoher Recall, hohe Precision

Anspruch an

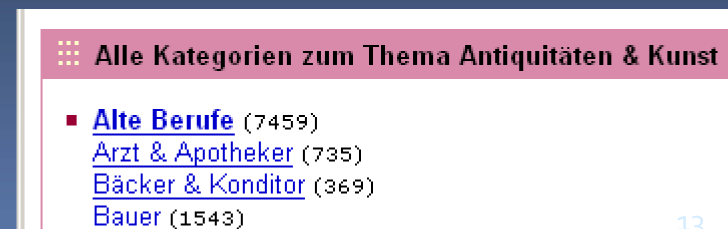
- Abdeckung genannter Use Cases
- Chance auf sinnvolle Struktur wird nicht dauerhaft und total zerstört
- Flexibles Schema
- Do the best you can

Persistenz
Elemente des Schemas

Klassen	Dienen semantischer Gruppierung (isA, subClass)
	Jede Klasse hat generischen Typ zugeordnet
Typen	**string, number, boolean, null**
	array
	object
	reference via OID referenziert
	weak als Substruktur inkludiert

Query, Load, Store, Instanziierung & co. nur nach **most specific classes**

- Via Use Case Annahmen und GUI gewährleistet



Persistenz
Sicht des Clients

Erlaubt ist für Attribute

additional zusätzliches Attribut, das der Typ nicht vorsieht

nullable jedes Attribut darf (in DB & Client) fehlen
fehlt $OID, dann nicht persistable

pending funktionalwertiges Attribut (Js Funktion)
Attribut wäre vorhanden, Wert wurde nicht geladen
Aufruf der Funktion (stub) lädt Wert dynamisch nach

Erlaubt ist für Methoden

Alles Dh: Fehlt, Parameter-Signatur falsch, Exceptions, usw.

Grund Server bereitet für Client korrekt auf
Client kann beliebig sich und andere betrügen
Server muß Client-Daten ohnehin massiv prüfen
Dem Client ist partial load gestattet (nullable, pending)
Pending load kann fehlschlagen

Persistenz
Methoden

Als Js Text in Schema / Typdefinition enthalten

Dynamisch dazugepatched

- über Js Prototype-Chain
- über Reflection Methoden __noSuchMethod__

Größere Flexibilität um den Preis geringerer Sicherheit

Persistenz
Sicht des Servers

Instanziieren: oid **new** (registeredClassId, [templateObject])

- Server generiert OID, speichert Klasse (& Typ), init nach Template

Speichern: object.**persist** ([jsonMask])

- Object kennt eigene OID in $OID
- Geschriebene Attribute: Alle - außer jsonMask*ierte* & pending (nicht mit null überschreiben, wenn nicht sicher weiß daß null ist)

Laden: object **load** (oid, [jsonMask])

- Relativ zu opt. Json-Mask Ausdruck als Typ-Filter (Col Liste im SELECT)
  Was wird eager (ie. sofort) geladen ?
  Was wird lazy (ie. pending als stub) geladen ?
  Was wird nicht geladen ? (Attribut null)

Persistenz
Implementierungs-Strategien

Jede **most-specific Klasse** wird durch eine Tabelle umgesetzt

- reference Typen: Durch OID
- simple Typen: Durch Spalte
- weak Typen: Durch flattening
- array Typen: Durch separate Tabelle
- pending: Durch Js Code Generierung für Client
- additionals: Durch Json Text
  Damit nicht mehr (effizient) durchsuchbar

person.name.first = "Joe"

Ergebnis

Persistenz-Architektur für Javascript Objekte

Homogen: Durchgängig, Client & Serverseitig Javascript-Nutzung

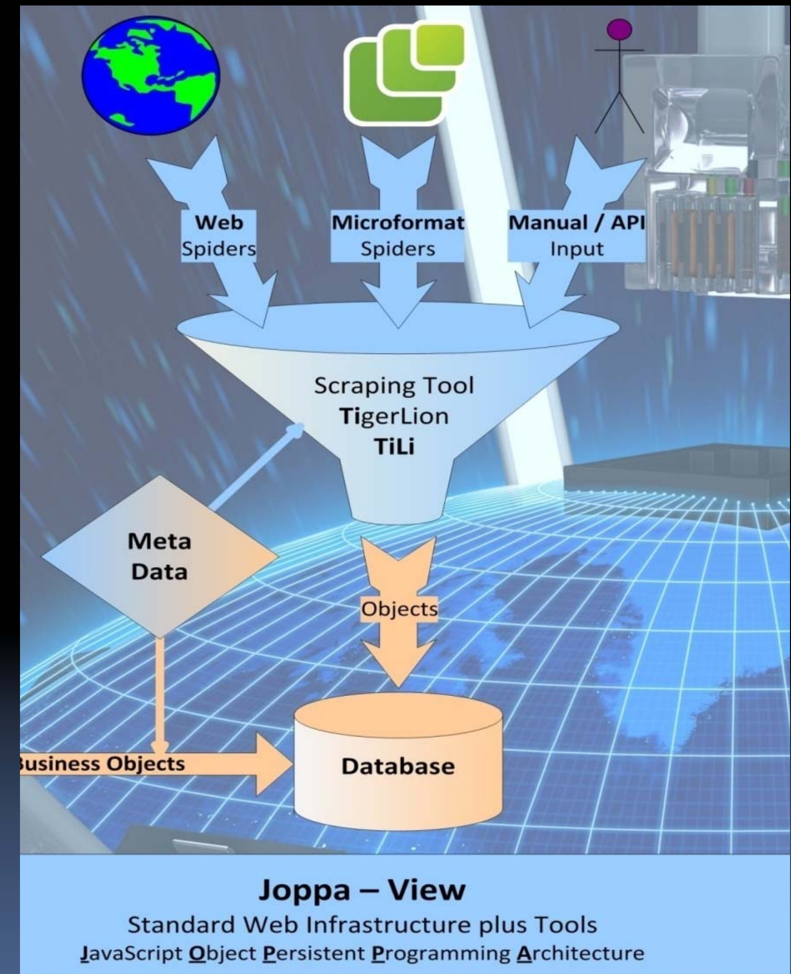
Realisierungsstand

- **Persistenz** 30% proof-of-idea (MySQL, PHP)

Geplanter Zugriff auf Suchmaschine/Datenbank

Auslieferung von Ergebnissen als

- **Reines Objekt**
- **Objekt + Darstellungsbeschreibung (Template)**



Realisierungsstand

- **Json RPC** 95% implementiert, PHP Server
- **UTE** 60% raw prototype

Vielen Dank für die Aufmerksamkeit.

Fragen?

Übersicht
Gesamtablauf der Anwendung

Client sendet QBE-Objekt

Server antwortet mit

1. Daten-Objekt und
2. Template-Funktion

Präsentation durch Anwendung der Js Funktion auf Objekt

Effekt: Client-side templating, skaliert besser

Und: Js Funktionen sind Domänen-weites Look %& Feel
wird 1x geladen
Damit: Sehr einfaches Skinning möglich

Business Logik: Server-side und Client-side in Javascript
Kann sogar identischer Code sein !
Geht auch wenn Objekte "Paragraphen" sind
Bsp: Wikis, Blogs, Annotationen