Linked Data architectural components

How-to attach linked data services to legacy infrastructure?

Daniel Martini, Mario Schmitz, Günter Engelhardt

Berlin, 27th of June, 2017
Organization

- Registered Association (non-profit):
  - Funded ~ 2/3 by the german ministry for nutrition and agriculture
  - ~ 400 members: experts from research, industry, extension...
  - ~ 70 employees working in Darmstadt
  - Managing lots of working groups, organizing expert workshops, represented in other committees, maintaining an expert network

- Tasks:
  - Knowledge transfer from research into agricultural practice
  - Supporting policy decision making by expertises
  - Evaluating new technologies: economics, ecological impact...
  - Providing planning data (investment, production processes...) to extension and farmers

- Role of Information Technology:
  - Data acquisition: harvesting open data sources
  - Data processing: calculating planning data from raw data
  - Information provision: delivery to clients via ebooks, web, apps
Goals and requirements

Deliver KTBL planning data in human and machine readable form alike:

- Machine classes: purchase prices, useful life, consumption of supplies...
- Standard field work processes: working time, machines commonly used under different regimes...
- Operating supplies: average prices, contents...
- Facilities and buildings: stables, milking machines and their properties
- ...

...to reach a broader audience and enable further processing within software applications for the use of farmers, extension...
There’s data that wants to get shared available at an organization.

We want to comply to FAIR principles:
  - Findable
  - Accessible
  - Interoperable
  - Reusable

So we **have to** use standard specifications:
  - RDF
  - HTTP
  - SPARQL
  - ...

But alas, data exists within a legacy infrastructure.

What’s in our toolbox to get it unlocked with the least effort possible?
Graph based data model

- Every data structure can be converted to a directed graph with relative ease
- Extensions can flexibly be implemented

Resource Description Framework (RDF):

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>FarmerXY</td>
<td>owns</td>
<td>Machine0815</td>
</tr>
<tr>
<td>Machine0815</td>
<td>type</td>
<td>tractor</td>
</tr>
<tr>
<td>Maschine0815</td>
<td>purchasePrice</td>
<td>83000 Euro</td>
</tr>
</tbody>
</table>

→ Rich representation
→ Advantages when it comes to search, navigation and decision support

“A traditional relational database may tell you the average age of everyone in this pub, but a graph database will tell you who is most likely to buy you a beer.”

Andreas Kollegger
Step 1: Create vocabulary
Most important: reuse

- No name properties. Recommendation: “rdfs:label is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name.”
  http://www.w3.org/TR/rdf-schema/

- Persons, addresses, phone numbers:
  - vcard: http://www.w3.org/2006/vcard/ns#
  - foaf: http://xmlns.com/foaf/0.1/

- Units and dimensions:
  - QUDT: http://qudt.org

- Geospatial data:
  - Geovocabulary: http://geovocab.org/
  - GeoSPARQL: http://www.opengeospatial.org/standards/geosparql

- Prices, Products, etc.:
  - Good Relations Ontology: http://www.heppnetz.de/projects/goodrelations/
Problems and solution approaches

The RDF data model does not support n-ary relations

Representation of physical quantities requires n-ary relations: value and dimension form an unseparable unit

- It gets worse, if the „what“ needs to be represented as well: „consumption 7.8 l diesel per h“, „fat content 35 g/l of milk“

- Three approaches to solving the problem:
  1. *data types*
     - advantage: compact notation
     - disadvantage: deprives you of the usage of XML schema data types (e. g. xsd:float) on the numerical value of the quantity
  2. *additonal ressource nodes in the graph*
     - advantage: simplifies reasoning
     - disadvantage: difficult to represent properly in services
  3. *blank nodes*
     - advantage: compact notation in different syntaxes, intuitive, possibility to add further datatypes to values
     - disadvantage: not that easy to handle in reasoning
Infrastructure: What’s needed?

- Pull data out of a relational data base
- Efficiently query a graph based data model
- Serialize in semantic web formats
- Service HTTP requests

- Relational-to-graph/RDF mapping tool
- Triple/quad store, SPARQL query engine
- Serializer/Linked data server component
## Evaluation

<table>
<thead>
<tr>
<th>Component</th>
<th>Alternatives</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational-to-graph/RDF mapping tool</td>
<td>DB2triples</td>
<td><strong>D2RQ:</strong></td>
</tr>
<tr>
<td></td>
<td>Virtuoso</td>
<td>- Supports Oracle data bases via JDBC</td>
</tr>
<tr>
<td></td>
<td>R2RML Parser</td>
<td>- Experience was available from a project</td>
</tr>
<tr>
<td></td>
<td>Xsparql</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Karma</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Jena Fuseki:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sesame</td>
<td>- Easy to use and configure</td>
</tr>
<tr>
<td></td>
<td>Stardog</td>
<td>- (relatively) lightweight</td>
</tr>
<tr>
<td></td>
<td>4Store</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Owlim</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ELDA:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D2R server</td>
<td>- Supports different serialization formats</td>
</tr>
<tr>
<td></td>
<td>Pubby</td>
<td>- Allows adjustment of the HTML layout via</td>
</tr>
<tr>
<td></td>
<td>Callimachus</td>
<td>velocity templates</td>
</tr>
<tr>
<td></td>
<td>Apache Marmotta</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Virtuoso</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Architecture

**Jena TDB**
- Graph Storage

**Fuseki**
- SPARQL Endpoint

**ELDA**
- Linked Data API Server
- Serialization: Turtle, XML, JSON
- Presentation: HTML, Velocity, XSLT

**Apache**
- Proxy
- Content Negotiation
- Static Content
- 303 Redirect

**D2RQ**
- Relational-to-Graph-Mapping

**SearchHaus**
- Semantic Search

**Website**

** „traditional“ KTBL Applications (e.g. FeldAV)**

**Full text publications**

**Apps**

**Desktop**

**Browser**
Website: http://www.epimorphics.com/web/tools/elda.html

Source code: https://github.com/epimorphics/elda

- used e.g. by data.gov.co.uk

- an implementation of the Linked Data API as specified at: https://github.com/UKGovLD/linked-data-api

- using the Apache velocity template engine http://velocity.apache.org

- one template for the whole server: templates can become rather complex, if you want to do path specific rendering or localization

- no native content negotiation: that requires Apache upfront

Next Generation Data API in Go

- Allow for differing HTML Renderings and SPARQL backend queries depending upon URL path requested and Accept*-headers:
  - template driven HTML frontend
  - SPARQL query templates with variable expansion
  - Each URL path can have its own HTML as well as SPARQL templates
- Content-Negotiation (HTTP Accept: header + filename suffix)
- Frontend Localization Support (HTTP Accept-Language: header + LDA _lang parameter)
- Support most of the additional query parameters in the LDA spec
- Replace LDA spec JSON compliant serialization by JSON-LD as specified by the recent W3C recommendation
Why Go?

http://golang.org

Created by some Google Engineers and former AT&T/Bell Labs Unix System Laboratories employees around 2009: Rob Pike, Robert Griesemer, Ken Thompson
Inspired by their former work at Bell Labs: Plan9

● Features:
  – The best of three worlds: Python, C/C++, Java
  – Compiled language with a clean, portable compiler design
  – Consistent syntax
  – Easy to use build and packaging framework included
  – Adjusted to modern hardware architectures: concurrency, networking
  – Performant (~ C++)
  – Non-object-oriented, but has interfaces and methods
  – Static typing, pointers but no pointer arithmetic, function closures...

● Used in some high profile, large-scale projects:
  – Soundcloud’s Prometheus monitoring system:  http://prometheus.io
  – Google’s download server:  http://dl.google.com serving Chrome, Android SDK, Earth... downloads
Architektur LOD-Service am KTBL

D2RQ
- Relational-nach-Graphen-Mapping

Jena TDB
- Graphen-Speicher

Fuseki
- SPARQL Endpoint

Apache
- Proxy

dangg
- Serialisierung: Turtle, N-Triples
- Content Negotiation: Format und Sprache
- HTML-Präsentation
- 303 Redirect und #-basierte Auslieferung

SearchHaus
- Semantische Suche

„traditionelle“ KTBL-Anwendungen (z. B. FeldAV)

KTBL-Oracle Datenbank

Mapping-Beschreibung

Service-Beschreibung

Volltexte

Apps

Desktop

Browser

Suchmaschinen

Website

Entscheidungsunterstützung, Softwareagenten
Dangg: features so far

● Done:
  – Content Negotiation
  – Per-Endpoint-Templates: SPARQL and HTML (standard go template engine: https://golang.org/pkg/html/template/)

● Not yet:
  – JSON-LD
  – IP-based logging
  – Configuration files
  – Complete LDA/LDP support

● In-memory label processing:
  – Speed (avg. 8 ms page load time -> huge improvement vs. ELDA)
  – Might require redesign with datasets with lots of labels

● ~2000 SLOC
Dangg: Core data structure

- "Item" struct:
  - Either a Subject/Object or a Predicate in a RDF triple

- Item struct fields:
  - P (Parent Node)
  - T (Node Type at parse time: subj/obj or pred)
  - L (human readable Label, filled from in-memory map)
  - U (URL)
  - V (Value: only filled for literals)
  - D (Dimension: only filled for physical quantities, requires units to be represented as blank nodes)
  - N (Next Level of Items)

- All fields referencable from HTML templates

- Can feed any RDF data to it, as long as units are represented using the blank node strategy
<table>
<thead>
<tr>
<th>Eigenschaften</th>
<th>Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typ</td>
<td>Ladewerkzeug, Maschinenklasse,</td>
</tr>
<tr>
<td>Anschaffungspreis</td>
<td>3050 EUR</td>
</tr>
<tr>
<td>Nutzungsdauer</td>
<td>10 a</td>
</tr>
<tr>
<td>Nutzungspotenzial nach Massendurchsatz</td>
<td>57000 t</td>
</tr>
<tr>
<td>Reparaturkosten nach Masse</td>
<td>0.01 EUR/t</td>
</tr>
<tr>
<td>Abstellmaß: Breite</td>
<td>2600 mm</td>
</tr>
<tr>
<td>Abstellmaß: Höhe</td>
<td>950 mm</td>
</tr>
<tr>
<td>Abstellmaß: Länge</td>
<td>1000 mm</td>
</tr>
</tbody>
</table>
Conclusions

- Free tools, not too difficult to setup are available
  - Usually, the problem exists between keyboard and chair
  - There are rough edges
  - Replacing certain components by own code is doable, when you are fluent in graph based data models

Alternatives:

1. Buy an all in-one-solution with a service contract
2. Program each and every data service from scratch
Thanks for listening!

Questions?

Contact: d.martini@ktbl.de