Easy Client-side Reasoning

Mehdi Terdjimi, Lionel Médini
Introduction

The Web as a an Application Platform

- For: work, entertainment, physical devices...
- More and more dynamic, reactive, etc.
- Mobile / front end first
- Lots of development tools
Introduction

The Web as a an Application Platform
- For: work, entertainment, physical devices...
- More and more dynamic, reactive, etc.
- Mobile / front end first
- Lots of development tools

The SemWeb as...
- Heavy backends
- Unreliable endpoints
- Complicated technologies
Introduction

The Web as a an Application Platform
- For: work, entertainment, physical devices...
- More and more dynamic, reactive, etc.
- Mobile / front end first
- Lots of development tools

The SemWeb as...
- Heavy backends
- Unreliable endpoints
- Complicated technologies

Well, maybe... But then, how to hide it?
Benefits of SemWeb for Web applications

- **Linked Data**
  - Lots of resources available
  - Reusability
  - Interoperability

- **Reasoning**
  - Automatic data deduction
  - Different levels of expressivity
  - High level of declarativity
Benefits of SemWeb for Web applications

- **Linked Data**
  - Lots of resources available
  - Reusability
  - Interoperability

- **Reasoning**
  - Automatic data deduction
  - Different levels of expressivity
  - High level of declarativity

→ No particular reason to do that on the server side
SemWeb on the client: fears

- **Limited storage / memory**
  - "Small data" approach
  - Only load what’s necessary onto the client
  - Use asynchronous data loading/update

- **Limited computing power**
  - Limited expressivity $\Rightarrow$ limited calculations
  - Choose the constructs that fit your application

- **Heterogeneous clients**
  - Resource-based adaptation
  - Detect client resources
  - Choose reasoning location

- **Loss of client Data**
  - Semantic data upload
  - Upload high-level data on the server
SemWeb on the client: tools

- Community group
  - RDF JavaScript Libraries Community Group

- Libraries
  - rdflib, rdfStore, N3...

- Reasoners
  - CHR: constraint solver
  - JSW: partial OWL2 EL
  - EYE: FOL & OWL2 reasoner – proof support – RDF streams
  - HyLAR: OWL2 RL – incremental – extensible – adaptable – NPM & Bower packages – dev-friendly GUI – Backbone, Angular 1 & Angular 2-compliant...
Simple application development scenario

- **Domain:** e-commerce
  - Locate products in stores

- **Developer’s objective:** code less
  - Reuse
    - Vocabularies
    - Data sources
    - Web APIs
    - Abstract business logic
    - Simplify (& pre-process) queries

- **Company’s objectives:** save resources
  - Servers
  - Network
S[i/a]mple application development scenario

**Step 1**
- Search vocabularies on the LOV (**at design time**)
  - GoodRelations
  - ProVoc
  - Part of Schema.org
- Convert to JSON-LD
- Load vocabularies onto the reasoner (**at runtime**)
- Launch classification task
  - Class subsumptions
  - Property subsumptions
S[i/a]mple application development scenario

Step 2
- Identify data sources \textit{(at design time)}
- Integrate actual data \textit{(at runtime)}
  - SPARQL INSERT DATA
- Launch transitive closure of the graph
  - Class assertions
  - Property assertions
Step 3

User request (at runtime)
The user searches for a 4G compatible tablet closeby

- Geolocation API
- Google Geocoder

```
SELECT ?product ?store {
  ?product a vocab:Tablet .
  ?offer gr:includes ?product .
  ?store gr:hasPOS ?location .
}
```

Query result bindings
S[i/a]mple application development scenario

Step 4
- Optimization against business logic scenarios
  - Identify complex processes **(at design time)**
  - Simplify them using rules

```
(?store http://purl.org/goodrelations/v1#hasPOS ?location)
  ^ (?location http://schema.org/place ?place)
  ->
  (?store http://www.my-online-store.fr/isNearBy
   http://www.w3.org/2001/XMLSchema#true)
```
S[i/a]mple application development scenario

Step 5

- Use the rules to precompute business facts \textbf{(at runtime)}
- Use the rules to query the triple store

\begin{verbatim}
SELECT ?product ?store {
  ?product a vocab:Tablet .
  ?offer gr:includes ?product .
  ?store vocab:isNearBy xsd:true .
}
\end{verbatim}

Query result bindings
Conclusion

- **ROI**
  - Reduces development time
  - Reduces infrastructure costs
  - Ensures best QoS

- **Performance**
  - Incremental
  - Ahead-of-time
  - Asynchronous
  - Cross-domain

- **Ease of use**
  - Integrates with JS frameworks
  - SPARQL decorator

- **Limits**
  - Requires basic Knowledge Engineering skills
  - Expressivity / performance tradeoff
  - “Small data” approach

- **Perspectives**
  - Improve adaptation parameters
  - Allow SWRL syntax
  - Improve authoring tools
That’s all!

References

For developers
- [https://github.com/ucbl/HyLAR-Reasoner](https://github.com/ucbl/HyLAR-Reasoner)
- [https://github.com/ucbl/HyLAR-Framework](https://github.com/ucbl/HyLAR-Framework)
- [https://www.npmjs.com/package/hylar](https://www.npmjs.com/package/hylar)

For academics