The Open University

Read-Write Hyperdata

RESTful Write-oriented API for Hyperdata in Custom RDF Knowledge Bases

Jacek Kopecký
The Open University (UK)

Linked APIs workshop
ESWC 2012, Crete

Overview

- Use case description
- API for writing RDF data
  - Structure of the API
  - Operations
  - Resources
  - **Interlinking**
  - **Self-description**

Use Case Description

- App for brokering offers
  - A use case in the project SOA4All
- Needs a datastore, wants RDF
- Manages instances of several classes:
  - *User*
  - *Offer Provider*
  - *Offer*
  - Various events: *Offer Forwarded, User Response*

Use Case Data: Users

- Interests in topics
  - *hasInterest, hasDisinterest*
  - *OfferCategory* class
- "Liking" offers
  - *likes, dislikes*
- Contact information
  - Structured instances of class *Contact*

Example Data

</users/1345#this>

a uc:User ;

uc:likes </offers/439#this>,

</offers/637#this> *;*

uc:hasInterest ex:Cars .

Generalized Situation

- Developing a new application
  - With new data sources
  - Linked Data in RDF
- Needs API for control over write access
  - Assigning identifiers to new instances
  - Security
  - Validation
  - Propagation (side effects) of updates
- SPARQL Update, Graph Store Protocol not suitable for open deployment

API Structure for Clients

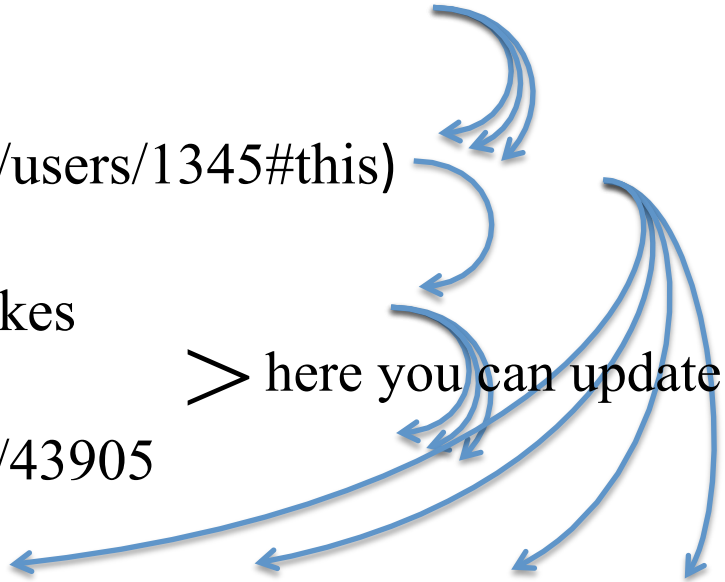
```
listUsers()
addUser(data)
getUser(id)
deleteUser(id)

    getUserLikes(id)
    addUserLike(id, uri)
    deleteUserLike(like-id)
    deleteAllUserLikes(id)
```

… Dislike, Interest, Disinterest, Contact

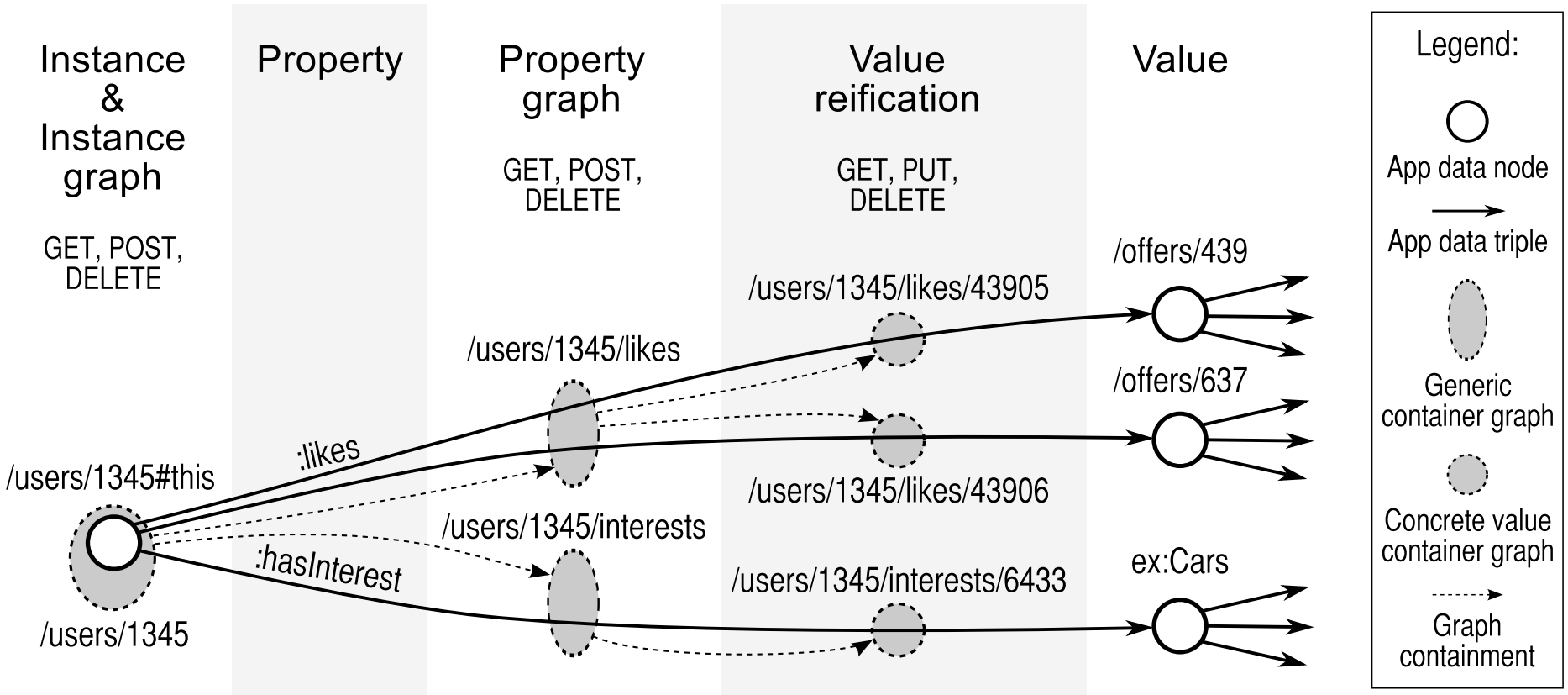
API in Resources

- Users – /users
  - GET, POST
- User – /users/1345 (user is /users/1345#this)
  - GET, POST, DELETE
- User's Likes – /users/1345/likes
  - GET, POST, DELETE
- One like – /users/1345/likes/43905
  - GET, PUT, DELETE
- Same two above for dislikes, interests, disinterests, contacts



> here you can update my centroid

Graph and Meta-graph



Self-description

</**users/1345**> a g:Graph ;
g:defines </users/1345#this> ;
g:contains </**users/1345/likes**>,
</**users/1345/interests**> ;
g:isContainedIn </**users**> .

</**users/1345/likes**> a g:Graph ;
g:contains </**users/1345/likes/43905**> ;
g:contains [
a rdf:Statement ;
rdf:subject </users/1345#this> ;
rdf:predicate uc:likes ;
rdf:object [ ]
].

</**users/1345/likes/43905**> a g:Graph ;
g:contains [
a rdf:Statement ;
rdf:subject </users/1345#this> ;
rdf:predicate uc:likes ;
rdf:object </offers/439#this>
].

Implementation, Future Work

- Prototype implementation for SOA4All
  - Hard-wired clients, though
- Extensiblity
  - Can allow “unexpected” properties
  - Can handle RDF data about values, with GC
- Using non-standard graph vocabulary
- Not done: Security, Validation, Side effects
  - Partly solvable with configuration
  - Partly hooks for app-specific code

Summary

- We need app-specific APIs for writing data
- For RDF, we propose **hyperdata**
  - Self-describing with a graph vocabulary
  - Interlinked
  - Discoverable
  - With implied or explicit update capabilities

Extra Questions for Discussion

- How does self-description actually help clients?
  - Can they be easier to develop?
  - Will they adapt to changed structures?
- Hypermedia as client platform?
  - Browser is not the client for Facebook – it's the javascript that's running in the browser
  - Does this break the client/server separation?