JSON-based interoperability applying the pull-parser programming model

Leandro Duarte Pulgatti, Marcos Didonet Del Fabro, Marina Pimentel
C3SL Labs, Federal University of Paraná, Curitiba, Brazil
Www.c3sl.ufpr.br
{ldpulgatti, marcos.ddf, marina}@inf.ufpr.br

Context

- There are several NoSQL approaches, with different formats
  - Key Value, Column Stores, Document stores, Graph, etc.
- Interoperability between all is difficult
- Several data transformations
  - NxN translators AND/OR
  - 1 (integration format) x N
- Several integration formats/solutions

How to choose/apply the best (more appropriate) solutions ?

NoSql Data Models

Representation strategies (12 from Bugiotti et al, 2013)

Database Type	Mode
Key Value	Key Value per object
	Key Value per field
	Key-hash per field
	Key-value per field Major/minor
	Key-value per atomic value
Graph	Graph

Database Type	Mode
Column Store	Column
	Super Column
	Column Family
	Super Column Family
Document Store	Document per object
	Item per object
	Cell per object

Assumption: there are more strategies, but this is a representative set

JSON format

- Largely used format
  - Interoperability, configuration (“new” XML)
  - RESTfull API exchange format
- Nested format
  - Elements, subelements, arrays
  - Several document stores

```
{ "Person":
  {"firstName":"John","lastName":"Smith","age":25,
  "phoneNumber": [
       { "type": "home", "number": "212 555-1234" },
       { "type": "fax", "number": "646 555-4567" }  ] }
}
```

Pull parser programming model

- Parsing of nested formats (Slomiski, 2001)
  - Initially with XML (Xerces, kXML, SAX APIs)
- Model
  - Given a stream of objects;
  - Read and process each object until achieving a condition (EOS, other);
    - Discard the object after each step;
- Advantages
  - Simple to implement and understand
  - Stateless, i.e., no need to keep the model in memomry
- Disavantages
  - Data model needs to be simplyfied (NESTED!)

JSON-based interoperability approach

- JSON as interoperability format
  - Not the most expressive format (e.g. relational, Ontologies)
  - Expressive enough to store several NoSQL
- Pull-parser programming model to process it
  - Need to be a nested format
  - No inter-elements cross-references
- Interoperability rules
  - From JSON to 12 NoSQL data models
  - Easy to implement following this model

Pull-parsing a JSON

Sample Input JSON **(1)**

```
{ "Person":
  {"firstName":"John","lastName":"Smith","age":25,
  "phoneNumber": [
       { "type": "home", "number": "212 555-1234" },
       { "type": "fax", "number": "646 555-4567" }  ] }
}
```

Flattening it to ease processing **(3)** (in memory)

```
ObjectId  : 8                ObjectId  : 22
DataValue : phoneNumber      DataValue : null
Label     : KEY_NAME         Label     : END_ARRAY;
FatherObj : 1                FatherObj : 9
ObjectId  : 9                ObjectId  : 23
DataValue : null             DataValue : null
Label     : START_ARRAY      Label     : END_OBJECT;
FatherObj : 8                FatherObj : 1
(the nested objects within
the phone number array)
```

Pull-parsing and tagging it **(2)**

```
{START_OBJECT
"Person"KEY_NAME:
  {START_OBJECT "firstName"KEY_NAME: "John"
    VALUE_STRING, "lastName"KEY_NAME:
    "Smith"VALUE_STRING, "age"KEY_NAME: 25
     VALUE_NUMBER,
   "phoneNumber"KEY_NAME : [START_ARRAY
      {START_OBJECT "type"KEY_NAME:
      "home"VALUE_STRING, "number"KEY_NAME:
      "212 555-1234"VALUE_STRING }END_OBJECT,
      {START_OBJECT "type"KEY_NAME:
      "fax"VALUE_STRING, "number"KEY_NAME:
      "646 555-4567"VALUE_STRING }END_OBJECT
   ]END_ARRAY
  }END_OBJECT
}END_OBJECT
```

Transformation rules

- From JSON to 12 representation stragegies
  - Key Value, Document, Column, Graph
- Most part of the rules have 2 main components
  1) Key (identification part)
  2) Value (the given object)
- The complexity of the rule may be placed in (1) or (2), depending on the data model

One strategy: key-value per field (kvpf)

Key production rule	Value production rule

`MainKey +"/"+Obj.KEY_NAME`	`for all Obj.Key_NAME do`
	`  if Value = (Array or Object) then`
	`  for all Obj.value do`
	`  Value = Value + Obj.value`
	`  else`
	`  Value = Obj.Value`
	`end`

Example output	

`Person:John/`
`firstName`
`lastName`
`age`
`phoneNumber`
	
	`John`
	`Smith`
	`25`
	`{"type":"home", "number": "212-555-1234"}`

Implementation

- Utilization of NoSQL stores with **put()** and **get()** methods
  - Key value: Oracle NoSQL Community
  - Column store: Apache Hbase
  - Document store: MongoDb
  - Graph store: Neo4J
- Data : 139,535 objects from the Chicago City Data Portal (easy to access stream)
- Transformations for all the 12 strategies
  - http://www.inf.ufpr.br/didonet/files/Jsonpullparser.zip

Conclusions

- Approach for interoperability between JSON and NoSQL formats
- JSON as central representation format has been a suitable choice
  - Not the most expressive representation
  - But expressive enough to cover several formats
- Pull parser programming model
  - Easy to implement and apply
  - Not suitable for statefull transformations
  - New rules can be added rapidly
- Future work
  - Develop more expressive formats

References (extract)

- Atzeni, P., Bugiotti, F., and Rossi, L. (2014). Uniform access to nosql systems. Information Systems, 43:117–133
- Bugiotti, F. and Cabibbo, L. (2013). A comparison of data models and apis of nosql datastores. Dipartamento di Ingegneria della Universita di Roma
- Slomiski, A. (2001). TR550: Design of a Pull and Push Parser System for Streaming XML. Technical report, CS Department, University of Indiana, US.
- Zhao, G., Lin, Q., Li, L., and Li, Z. (2014). Schema conversion model of sql database to nosql. In P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014, pages 355–362. IEEE
- Scavuzzo, M., Di Nitto, E., and Ceri, S. (2014). Interoperable data migration between nosql columnar databases. In 2014 IEEE 18th EDOCW , pages 154–162. IEEE
- Michel, F., Djimenou, L., Faron-Zucker, C., and Montagnat, J. (2014). xr2rml: Relational and non relational databases to rdf mapping language. Technical report, Research report. ISRN I3S/RR 2014-04-FR v3.
- Chung, W.-C., Lin, H.-P., Chen, S.-C., Jiang, M.-F., and Chung, Y.-C. (2014). Jackhare: a framework for sql to nosql translation using mapreduce. Automated Software Engineering, 21(4):489–508.