SPOON Evaluation Report

Walter Corno, Irene Celino, Emanuele Della Valle, Francesco Corcoglioniti
CEFRIEL – Politecnico di Milano
http://swa.cefriel.it/SPOON

In this document we describe the evaluation of the SPOON tool with regards to a test data set and in comparison with competing approaches.

What is SPOON?

The Web of Data vision raises the problem of how to expose existing data sources on the Web without requiring heavy manual work. Our approach facilitate SPARQL[i] queries over heterogeneous data sources, through the use of an object-oriented abstraction which can be automatically mapped and translated into an ontological one; this approach, on the one hand, helps data managers to disclose their sources without the need of a deep understanding of Semantic Web technologies and standards and, on the other hand, takes advantage of object-relational mapping technologies to deal with different types of data sources (relational DBs, but also XML sources, object-oriented DBs, LDAP, etc.). **SPOON** (**SP**arql to **O**bject **O**riented e**N**gine) is the first implementation of our approach; it is a tool that helps data managers to publish their (heterogeneous) data sources on the Semantic Web as SPARQL endpoints.

SPOON is able to manage such sources through an object-oriented virtualization layer compliant with the JDO2[ii] specification. This JDO abstraction layer is then translated in a corresponding ontological model by using a totally automated one-to-one mapping. This online wrapping of the object-oriented model allows avoiding synchronization problems between the two models.

SPARQL queries are processed and translated to JDOQL queries and then are executed over the object-oriented virtualization layer, using JPOX[iii] as persistence manager tool. The current implementation of SPOON supports only a subset of SPARQL queries (BGPs and FILTER) and does not (yet) support variables on predicates.

Evaluation Framework

To prove our approach we built a testing framework using the Gene Ontology[iv] data source, which is available both as SQL dump and RDF graph. The evaluation is then made by comparing results and performances when executing a common set of SPARQL queries with SPOON and with other competing approaches (namely, D2R over the relational source and plain SPARQL over the native RDF repository). The evaluation framework is therefore set up as in Figure **1**.

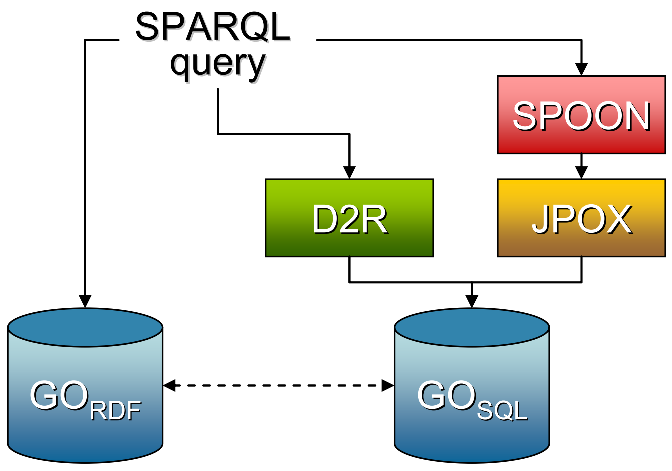


Figure 1 - SPOON Evaluation Framework

Each selected SPARQL query is executed three times on the three different systems:

- by using our system SPOON, which translates the SPARQL query into the respective JDOQL query to be executed on JPOX wrapping the SQL data source;
- by using the D2R system[v], which translates the SPARQL query into the respective SQL query on the SQL data source;
- by using a standard SPARQL processor[vi], which executes the query over the RDF data source.

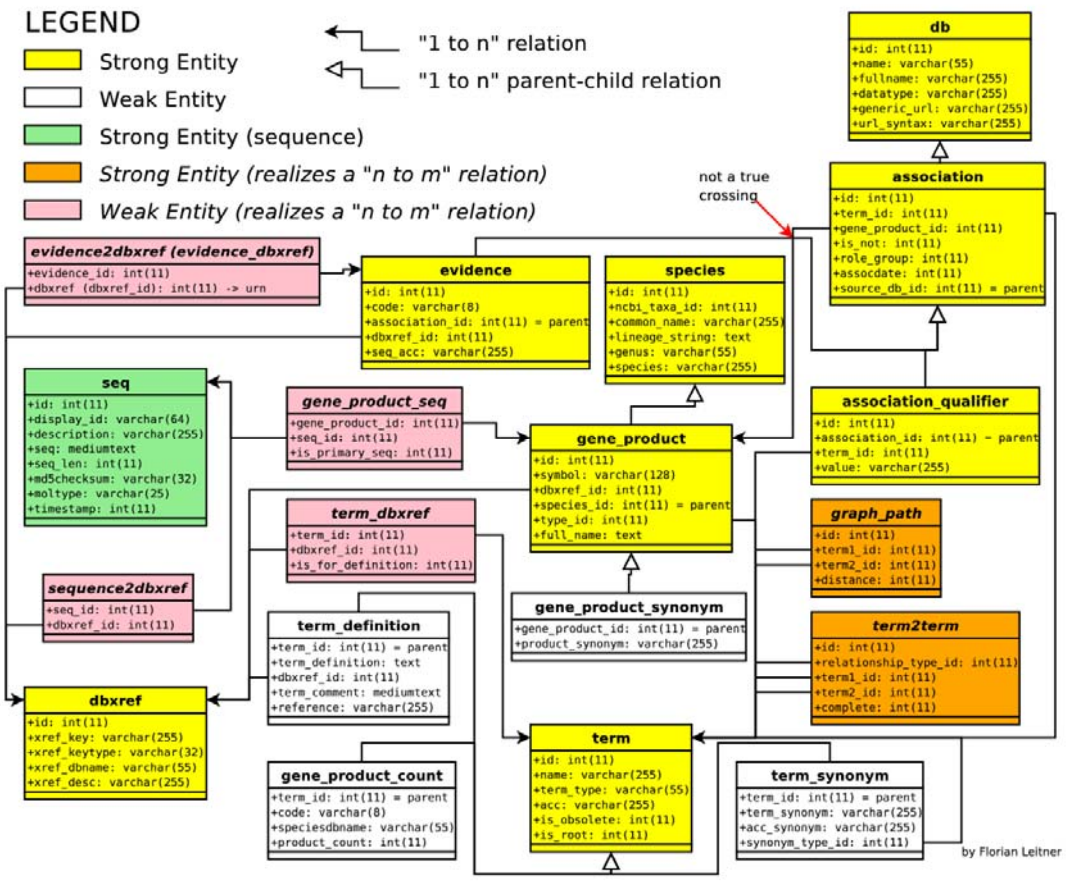
The baseline for evaluating the query execution performances of the three different systems is set by the time spent to execute the SQL query – corresponding to the SPARQL query – directly on the relational database.

GO Schema

In order to understand the test queries (reported below), we now present the GO schema (in the next page we show the E/R diagram). The most important entities are:

- **Term**: this entity represents a concept of the ontology, it can be a cellular component, a molecular function or a biological process. There are two types of relationships among terms: *IS-A* and *PART-OF* (these relationships are stored in the **term2term** table). Each term has a definition (stored in the table **term_definition**) and can have zero or more synonyms (stored in the table **term_synonym**).
- **Gene_product**: this entity represents a gene product or a gene. Each gene product has a name (*symbol*).
- **Association**: this entity represents a relationship between a term and a gene product.
- **Evidence**: this entity contains the source of an association between a term and a gene product. It can be an experimental result, a computational analysis deduction or an automatically inferred result.
- **DBXref**: this entity specify the database from which an information (a term, a gene product, an association or an evidence) is obtained.

All the other entities are not exported in the RDF graph available on the GO web page, so we don't use them.



The mappings from this E/R diagram to the object-oriented model used by JPOX, and the mapping from the diagram to the RDF graph used by D2R are available on the SPOON web page at: http://swa.cefriel.it/SPOON

SPARQL Test Queries

In order to perform our evaluation, we chose a set of queries to be executed in our testing framework. These queries are based on those publicly available on the Gene Ontology group's web-site[vii]. In the following we list the queries we selected.

Query 1:

In this query we want to obtain the name of the term with accession "GO:0000011", and the names of all the terms "IS-A" related with this term. This query involves only terms.

```
PREFIX go: <http://www.geneontology.org/dtds/go.dtd#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?isaname
WHERE {
    ?x rdf:type  go:term.
    ?x go:Termaccession "GO:0000011".
    ?x go:Termtname ?name.
    ?x go:Termis_a ?isa.
    ?isa go:Termtname ?isaname.
}
```

Query 2:

In this query we want to obtain the names of all the gene products associated to the term with accession "GO:0000018" (we also require the name of this term). This query involves terms, gene products and associations.

```
PREFIX go: <http://www.geneontology.org/dtds/go.dtd#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?tname ?gpname
WHERE {
   ?x rdf:type  go:term.
   ?x go:Termaccession "GO:0000018".
   ?x go:Termtname ?tname.
   ?x go:Termassociation ?ass.
   ?ass go:AssociationClassgene_product ?gp.
  ?gp go:GeneProductClassgpname ?gpname.
}
```

Query 3:

In this query we want to obtain the names of all the terms "IS-A" related to the term "regulation of DNA recombination", but only if they are associated with at least a gene product (we want also the names of all these gene products). In this query we use the operator *Filter*.

```
PREFIX go: <http://www.geneontology.org/dtds/go.dtd#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?tname ?gpname
WHERE {
   ?x rdf:type  go:term.
   ?x go:Termtname ?tname.
   ?x go:Termis_a ?isa.
   ?isa go:Termtname ?isaname.
   ?x go:Termassociation ?ass.
   ?ass go:AssociationClassgene_product ?gp.
   ?gp go:GeneProductClassgpname ?gpname.
   FILTER(?isaname = "regulation of DNA recombination")
}
```

SPOON Evaluation results

All the tests have been executed on a laptop machine with the following characteristics:

- AMD Turion X2 TL-60 2GHz 64 bit, 1024kb L2 cache

- 1GB DDR2 667MHz
- SATA hard-drive 160GB 5400 rpm
- OS Linux Gentoo 2007.0 amd64

The three tested systems have been configured as follow:

- The *dump SQL* of GO has been loaded in Mysql 5.0.54
- SPOON has been deployed on Tomcat 5.5 Web-server with JPOX 1.2.2
- D2R has been used within a local application (we used directly the D2R libraries)
- The Native SPARQL processor was Sesame 2.1.2, with a Native RDF Repository configured as showed here: http://www.openrdf.org/doc/sesame2/users/ch08.html#d0e687

In the following table we compare the performances registered when executing the queries onto our evaluation framework. For the Native RDF repository we have dropped from the query the triple pattern with *rdf:type* as predicate, because this doesn't affect the results and, even if it provides an additional "clue" to the SPARQL query processor, we have observed loss of performances considering also that triple (see last table).

Executed query	**SPOON**	D2R	Native SPARQL	*SQL baseline*
Query 1	291ms	695ms	280ms	95ms
Query 2	313ms	774ms	281ms	70ms
Query 3	540ms	3808ms	63620ms	179ms

In the following table we show the translation time and execution time of the queries in SPOON:

Executed query	Translation time	Execution time
Query 1	14ms	277ms
Query 2	14ms	299ms
Query 3	177ms	363ms

Lastly we show the loss of performances observed for the Native RDF repository considering also the triples with *rdf:type* as predicate:

Executed query	Time with the triple	Time without the triple
Query 1	21561ms	280ms
Query 2	29184ms	281ms
Query 3	72801ms	63620ms

Results Discussion

From these results we can observe that the impact of the translation phase in SPOON is usually limited if compared to the query execution time which is heavily dependent on the ORM used (JPOX). In the third query the translation time is bigger because the query is more complex (we have observed that the syntactic analysis phase, performed by ARQ[viii], requires more time to parse and process the query when it contains some operators).

The loss of performances for the Native RDF repository seems to be caused by the lack of a proper index over *rdf:type* triples. The structure of the third query hampers an effective use of the indexes, and so the performances are really bad.
We have also observed different performances changing the order of triple patterns in the queries; this shows that Native RDF reporitories are still immature, so we think that the choice of a real-time mapping is really the right choice.

Comparing the performances of SPOON with those of D2R, we can observe that the use of an ORM tool allows the generation of cleaner and more compact SQL queries (the generated SQL queries are shown below). In particular D2R adds many useless joins (in the first two queries it adds a join for each triple), and for the third query we can observe that D2R generates a first preliminary SQL query (that returns all the terms "IS-A" related with the term "regulation of DNA recombination"), and then for each result of this SQL query it generate a new SQL query (to obtain the gene products). So the total number of SQL queries depends by the number of results returned by a preliminar SQL query.
Moreover D2R doesn't include the "NOT NULL" constraints in its SQL queries, so it must perform many post-processing operations.

Translations of Queries

For completeness, we report now the JDOQL and SQL queries translated by SPOON + JPOX and by D2R

Translations of Query 1:

<u>JDOQL translated query by SPOON</u>:

```
SELECT this.tname, varisa.tname
FROM go.Term
WHERE this.tname!=null && varisa.tname!=null &&
        this.accession == 'GO:0000011' &&
        this.is_a.contains(varisa)
VARIABLES go.Term varisa;
```

<u>SQL</u>:
(directly executed)

```
SELECT term.name , term1.name
FROM term, term2term, term as term1
WHERE term.id =term2term.term2_id AND
        term.acc = "GO:0000011" AND
        term1.id = term2term.term1_id AND
        term.name IS NOT NULL AND
        term1.name IS NOT NULL AND
```

```
            term2term.relationship_type_id = 2
```

(translated by SPOON)

```
SELECT this.`name`,UNBOUND_varisa.`name`
FROM term this CROSS JOIN term UNBOUND_varisa CROSS JOIN term2term this_is_a
WHERE this_is_a.term2_id = this.id AND (this.`name`) IS NOT NULL AND
            (UNBOUND_varisa.`name`) IS NOT NULL AND
            this.acc = 'GO:0000011'  AND
            UNBOUND_varisa.id = this_is_a.term1_id AND
            this_is_a.relationship_type_id = 2
```

(translated by D2R)

```
SELECT DISTINCT `T0_term`.`id`, `T1_term`.`id`, `T2_term`.`id`, `T2_term`.`name`,
          `T3_term2term`.`term1_id`, `T3_term2term`.`term2_id`, `T4_term`.`name`, `T4_term`.`id`
FROM `term` AS `T0_term`, `term` AS `T1_term`, `term` AS `T4_term`, `term` AS `T2_term`,
          `term2term` AS `T3_term2term`
WHERE (`T0_term`.`id` = `T1_term`.`id` AND `T0_term`.`id` = `T2_term`.`id` AND `T0_term`.`id` =
            `T3_term2term`.`term2_id` AND `T1_term`.`acc` = 'GO:0000011' AND `T1_term`.`id` =
            `T2_term`.`id` AND `T1_term`.`id` = `T3_term2term`.`term2_id` AND `T2_term`.`id` =
            `T3_term2term`.`term2_id` AND `T3_term2term`.`relationship_type_id` = 2 AND
            `T3_term2term`.`term1_id` = `T4_term`.`id`)
```

Translations of Query 2:

JDOQL translated query by SPOON:

```
SELECT this.tname, varass.gene_product.gpname
FROM go.Term
WHERE this.tname!=null && varass.gene_product.gpname!=null &&
            this.accession == 'GO:0000018' && this.association.contains(varass)
VARIABLES go.AssociationClass varass;
```

SQL:
(directly executed)

```
SELECT term.name , gene_product.symbol
FROM term, association, gene_product
WHERE term.id = association.term_id AND gene_product.id = association.gene_product_id AND
            term.acc = 'GO:0000018' AND term.name IS NOT NULL AND
            gene_product.symbol IS NOT NULL
```

(translated by SPOON)

```
SELECT this.`name`,UNBOUND_varass_gene_product_gpname.symbol
FROM association UNBOUND_varass LEFT OUTER JOIN gene_product
            UNBOUND_varass_gene_product_gpname ON UNBOUND_varass.gene_product_id =
            UNBOUND_varass_gene_product_gpname.id CROSS JOIN term this
WHERE UNBOUND_varass.term_id = this.id AND (this.`name`) IS NOT NULL AND
            (UNBOUND_varass_gene_product_gpname.symbol) IS NOT NULL AND
            this.acc = 'GO:0000018' AND UNBOUND_varass.id = UNBOUND_varass.id
```

(translated by D2R)

```
SELECT DISTINCT `T0_term`.`id`, `T1_term`.`id`, `T2_term`.`id`, `T2_term`.`name`,
            `T3_association`.`term_id`, `T3_association`.`id`, `T4_association`.`id`,
            `T4_association`.`gene_product_id`, `T5_gene_product`.`id`, `T5_gene_product`.`symbol`
FROM `association` AS `T4_association`, `association` AS `T3_association`, `term` AS `T0_term`, `term`
             AS `T1_term`, `gene_product` AS `T5_gene_product`, `term` AS `T2_term`
WHERE (`T0_term`.`id` = `T1_term`.`id` AND `T0_term`.`id` = `T2_term`.`id` AND
           `T0_term`.`id` =`T3_association`.`term_id` AND `T1_term`.`acc` = 'GO:0000018' AND
           `T1_term`.`id` = `T2_term`.`id` AND `T1_term`.`id` = `T3_association`.`term_id` AND
           `T2_term`.`id` = `T3_association`.`term_id` AND `T3_association`.`id` = `T4_association`.`id`
           AND `T4_association`.`gene_product_id` = `T5_gene_product`.`id`)
```

Translations of Query 3:

JDOQL translated query by SPOON:

```
SELECT this.tname, varass.gene_product.gpname
FROM go.Term
WHERE this.tname!=null &&
    varisa.tname!=null &&
    varass.gene_product.gpname!=null &&
    varisa.tname == "regulation of DNA recombination" &&
    this.is_a.contains(varisa) &&
    this.association.contains(varass)
VARIABLES go.Term varisa; go.AssociationClass varass;
```

SQL:
(directly executed)

```
SELECT term.name, gene_product.symbol
FROM association, gene_product, term, term as term1, term2term
WHERE term2term.term2_id = term.id AND
    term2term.term1_id = term1.id AND
    association.term_id = term.id AND
    term.name IS NOT NULL AND
    term1.name = 'regulation of DNA recombination' AND
    gene_product.id = association.gene_product_id AND
    gene_product.symbol IS NOT NULL
```

(translated by SPOON)

```
SELECT this.`name`,UNBOUND_varass_gene_product_gpname.symbol
FROM association UNBOUND_varass LEFT OUTER JOIN gene_product
   UNBOUND_varass_gene_product_gpname ON
   UNBOUND_varass.gene_product_id = UNBOUND_varass_gene_product_gpname.id
   CROSS JOIN term this CROSS JOIN term UNBOUND_varisa
   CROSS JOIN term2term this_is_a
WHERE this_is_a.term2_id = this.id AND UNBOUND_varass.term_id = this.id AND
    (this.`name`) IS NOT NULL AND (UNBOUND_varisa.`name`) IS NOT NULL AND
    (UNBOUND_varass_gene_product_gpname.symbol) IS NOT NULL AND
    UNBOUND_varisa.`name` = 'regulation of DNA recombination' AND
    UNBOUND_varisa.id = this_is_a.term1_id AND
    UNBOUND_varass.id = UNBOUND_varass.id
```

(translated by D2R)

```
SELECT DISTINCT `T0_term`.`id`, `T1_term`.`id`, `T1_term`.`name`,
      `T2_term2term`.`term2_id`, `T2_term2term`.`term1_id`, `T3_term`.`id`,
      `T3_term`.`name`
FROM `term` AS `T3_term`, `term` AS `T0_term`, `term` AS `T1_term`,
    `term2term` AS `T2_term2term`
WHERE (`T0_term`.`id` = `T1_term`.`id` AND `T0_term`.`id` = `T2_term2term`.`term2_id` AND
     `T1_term`.`id` = `T2_term2term`.`term2_id` AND
     `T2_term2term`.`relationship_type_id` = 2 AND
     `T2_term2term`.`term1_id` = `T3_term`.`id`)

SELECT DISTINCT `T0_association`.`id`, `T1_association`.`id`,
      `T1_association`.`gene_product_id`, `T2_gene_product`.`id`,
      `T2_gene_product`.`symbol`
FROM `association` AS `T1_association`, `association` AS `T0_association`,
    `gene_product` AS `T2_gene_product`
WHERE (`T0_association`.`id` = `T1_association`.`id` AND `T0_association`.`term_id` = 31 AND
     `T1_association`.`gene_product_id` = `T2_gene_product`.`id`)

SELECT DISTINCT `T0_association`.`id`, `T1_association`.`id`,
      `T1_association`.`gene_product_id`, `T2_gene_product`.`id`,
      `T2_gene_product`.`symbol`
FROM `association` AS `T1_association`, `association` AS `T0_association`,
    `gene_product` AS `T2_gene_product`
WHERE (`T0_association`.`id` = `T1_association`.`id` AND
     `T0_association`.`term_id` = 8081 AND
     `T1_association`.`gene_product_id` = `T2_gene_product`.`id`)

SELECT DISTINCT `T0_association`.`id`, `T1_association`.`id`,
      `T1_association`.`gene_product_id`, `T2_gene_product`.`id`,
      `T2_gene_product`.`symbol`
FROM `association` AS `T1_association`, `association` AS `T0_association`,
    `gene_product` AS `T2_gene_product`
WHERE (`T0_association`.`id` = `T1_association`.`id` AND
     `T0_association`.`term_id` = 8089 AND
     `T1_association`.`gene_product_id` = `T2_gene_product`.`id`)

SELECT DISTINCT `T0_association`.`id`, `T1_association`.`id`,
      `T1_association`.`gene_product_id`, `T2_gene_product`.`id`,
      `T2_gene_product`.`symbol`
FROM `association` AS `T1_association`, `association` AS `T0_association`,
    `gene_product` AS `T2_gene_product`
WHERE (`T0_association`.`id` = `T1_association`.`id` AND
     `T0_association`.`term_id` = 19604 AND
     `T1_association`.`gene_product_id` = `T2_gene_product`.`id`)

SELECT DISTINCT `T0_association`.`id`, `T1_association`.`id`,
      `T1_association`.`gene_product_id`, `T2_gene_product`.`id`,
      `T2_gene_product`.`symbol`
FROM `association` AS `T1_association`, `association` AS `T0_association`,
    `gene_product` AS `T2_gene_product`
WHERE (`T0_association`.`id` = `T1_association`.`id` AND
     `T0_association`.`term_id` = 20288 AND
     `T1_association`.`gene_product_id` = `T2_gene_product`.`id`)
```

```
SELECT DISTINCT `T0_association`.`id`, `T1_association`.`id`,
      `T1_association`.`gene_product_id`, `T2_gene_product`.`id`,
      `T2_gene_product`.`symbol`
FROM `association` AS `T1_association`, `association` AS `T0_association`,
     `gene_product` AS `T2_gene_product`
WHERE (`T0_association`.`id` = `T1_association`.`id` AND
      `T0_association`.`term_id` = 20289 AND
      `T1_association`.`gene_product_id` = `T2_gene_product`.`id`)
```

---

i SPARQL: http://www.w3.org/TR/rdf-sparql-query/

ii JDO2 Specifications: http://jcp.org/aboutJava/communityprocess/final/jsr243/index.html

iii JPOX: http://www.jpox.org/

iv Gene Ontology: http://www.geneontology.org/

v D2R: http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/; we didn't make use of the D2R Server, but we used directly the D2R library.

vi Sesame, used with its RDF native store: http://www.openrdf.org/

vii Gene Ontology queries: http://wiki.geneontology.org/index.php/Example_Queries

viii ARQ: http://jena.sourceforge.net/ARQ/