Generating Semantic Web Services from Declarative Descriptions

Mohammad Sadnan Al Manir, Christopher J.O. Baker
Department of Computer Science and Applied Statistics
University of New Brunswick
Saint John, Canada
{*sadnan.almanir,bakerc*}*[at]unb.ca*

Alexandre Riazanov
IPSNP Computing Inc, Canada
alexandre.riazanov[at]ipsnp.com

Harold Boley
Faculty of Computer Science
University of New Brunswick
Fredericton, Canada
harold.boley[at]ruleml.org

***Abstract*—Semantic Web services are an effective middleware for semantic querying of relational databases. Despite the benefits of this approach, writing Web service code manually is labor-intensive and error-prone. To ameliorate this, we propose a framework to generate SADI web services from declarative service descriptions in which access to databases is achieved through semantic mappings. These mappings are scripted in the Datalog sublanguage of Positional-Slotted Object-Applicative (PSOA) RuleML. We outline a novel methodology, a system architecture, and an early stage implementation for service generation. We demonstrate the utility of this approach in a use case for querying patient data from a hospital data warehouse.**

I. Introduction

Semantic Querying (SQ) is based on the automatic application of domain knowledge formalized as ontologies and rules, which semantically capture the underlying database design. An explicit semantic correspondence between the database schema and relevant domain ontologies is established by rules. Each *domain ontology* constitutes a high-level model in the form of logical axioms using RDF(S)[1],[2] and OWL[3], which allows domain experts to pose queries in a semantic context that they are familiar with.

Existing SQ systems such as D2RQ [1], MASTRO [2], Incremental Query Rewriting (IQR) [3] typically allow database programmers to define mappings between relational schemas and domain knowledge bases, in the form of logical axioms or similar declarative constructs. These systems then translate the domain-based queries into SQL queries that can be directly executed on the data.

In recent work, HAIKU [4], [5] considers a different approach based on the deployment of Semantic Web Services on top of relational databases. This approach relies on suitable mappings written by the database programmers, allowing SADI [6] framework-based Semantic Web services to extract Hospital-Acquired Infections (HAI) data from The Ottawa Hospital (TOH) Data Warehouse (DW) [7]. One limitation of this approach is that service creation becomes labor-intensive and can be error-prone, because it requires writing code for the service. This motivated us to investigate if code generation could be automated from declarative service descriptions, specifically by incorporating the necessary input and output parameters in appropriate places of generic Web service code-blocks.

In this paper, an architecture based on SQ is presented and its implementation is outlined with the goal of generating SADI Semantic Web service code automatically from their declarative input and output descriptions. The architecture enables access to relational data via the expressive rule language PSOA RuleML [8]. The automation facilitates Web service generation without human intervention and users are able to run queries over the generated services with the help of SADI query clients like Hydra and SHARE (see, e.g., [5]).

The methodology and architecture are novel: we are not aware of another system that allows the creation of Semantic Web services on top of relational data by leveraging an expressive rule language for semantic mapping and a first-order logic reasoner for query rewriting.

The paper is organized as follows: we start with a brief description of SADI in Section II. A use case for service generation is shown in Section III and the work flow of our architecture is described in Section IV. Finally, in Section V some of the implementation challenges and an evaluation of the methodology are briefly discussed.

II. Preliminaries

A. Basic SADI Ideas

SADI [6] is a framework which utilizes Semantic Web standards and allows integration and interoperability among resources on the Web. SADI uses RDF[S], OWL for data representation and modeling, and HTTP-based recommendations (GET, POST) for interacting with the services.

The main distinguishing features of SADI services is that (1) they only exchange RDF, so "they speak the same language" (2) they can be automatically discovered and (3) orchestrated with the help of query clients like Hydra and SHARE (see, e.g., [5]).

III. Early-Stage Implementation

To show the advantages of our architecture, we walk through the generation of a simple SADI service that can

[1]http://www.w3.org/TR/rdf-concepts/
[2]http://www.w3.org/TR/rdf-schema/
[3]http://www.w3.org/TR/owl-overview/

query a database and retrieve results.

Our early-stage experiment comprises of a database schema, a corresponding domain ontology, service I/O descriptions modeled according to the ontology, and an SQL template generated from inputs to the reasoner.

A schema named PatientDiseaseDB is shown in Fig. 1. A relevant ontology describing the same domain is presented next.

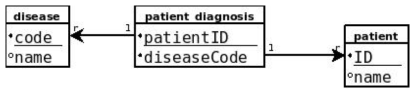

Figure 1. Database of patients and their diagnosed diseases

```
@prefix servOnt: <http://unbsj.biordf.net/servOnto.owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

servOnt:Disease a owl:Class.
servOnt:Patient a owl:Class.

servOnt:hasDisease a owl:ObjectProperty.
servOnt:isDiseaseOf a owl:ObjectProperty;
 owl:inverseOf servOnt:hasDisease.

servOnt:name a owl:DatatypeProperty;
 rdfs:range xsd:string.

servOnt:getPatientNameByDiseaseName_Input a owl:Class;
 owl:equivalentClass [a owl:Class;
   owl:intersectionOf (servOnt:Disease [a owl:Restriction
     owl:onProperty servOnt:name;
     owl:someValuesFrom xsd:string])].

servOnt:getPatientNameByDiseaseName_Output a owl:Class;
 rdfs:subClassOf [a owl:Restriction;
  owl:onProperty servOnt:isDiseaseOf;
  owl:someValuesFrom [a owl:Class;
   owl:intersectionOf (servOnt:Patient [a owl:Restriction;
    owl:onProperty servOnt:name;
    owl:someValuesFrom xsd:string])]].
```

Here we describe a simple SADI service `getPatientNameByDiseaseName` which, upon receiving the name of a disease as input, provides the corresponding patients' names.

The input class is defined by a disease name with the *name* data property attached to `Disease` class which is expressed in Protégé syntax as `Disease and` *name* `some string`.

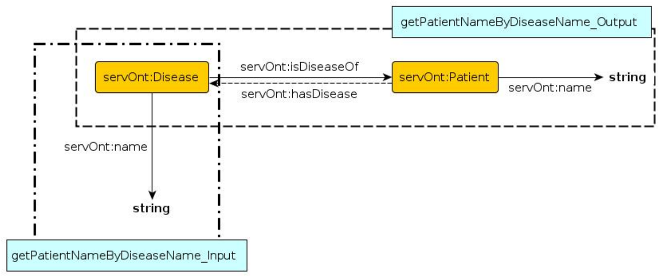

Figure 2. A Simple SADI Service getPatientNameByDiseaseName

The output class is defined by the patient names with the *name* property attached to the class `Patient`, which is attached to the `Disease` class by the *isDiseaseOf* property and is expressed as *isDiseaseOf* `some (Patient and` *name* `some string)`.

Fig. 2 depicts the modeling of both the input and the output classes. The root node for both the classes is `Disease`. The solid arrows are labeled by the object property *isDiseaseOf* and by the single string-type data property *name*. Although the inverse property *hasDisease* is defined in the ontology, it is not part of the declarative descriptions, and denoted only by a dotted arrow.

Our reasoner, VampirePrime[4] uses TPTP [9] as its primary input syntax. Hence, a translation is necessary to transform any non-TPTP syntax for generating SQL. This is accomplished by incorporating three translators into the architecture. The semantic mappings expressed in PSOA in Section IV-B can be translated into TPTP by using open-source tools such as the PSOA RuleML API [10] and PSOA2TPTP [11] (part of PSOATransRun[5]). The declarative input and output descriptions and the ontology are translated by the OWL API [12].

For example, the input and output declarative descriptions are translated by the OWL API-based translator into a single TPTP rule below ('- -' labels conditions while '++' labels the conclusion, `X, N, D` are variables):

```
--p_Patient(X),--p_name(X, N),--p_isDiseaseOf(D, X)
, --p_Disease(D), --p_name(D, "?"), ++answer(N)
```

The tuple `N` in the unary predicate *answer* denotes the patient tuple `X`'s names who have the disease tuple `D` with a name `"?"`, which is like a formal parameter and its actual values come from actual service inputs in run time. This rule is created by merging the input and output class based on the SADI principle that both the input and output class have a common root node.

VampirePrime generates the following SQL query template from the semantic mapping, ontology and the TPTP rule. Although in this specific case no reasoning is necessary, potentially VampirePrime can do very complex reasoning to rewrite queries. The template query contains the WHERE clause with the condition `disease.name = "?"`, where the symbol `'?'` is extracted from the TPTP predicate `p_name(D, "?")` above.

```
SELECT patient.name AS patName
FROM patient, disease, patientdiagnosis
WHERE  disease.name = "?"
AND patient.id = patientdiagnosis.patient_id
AND disease.code = patientdiagnosis.code
```

Due to space constraints, we refrain from documenting the complete Java code for the SADI Web service.

One of the most important tasks of our system is to extract the inputs from an RDF input instance and to place them

[4]http://riazanov.webs.com/software.htm, last accessed on June 14, 2013
[5]http://wiki.ruleml.org/index.php/PSOA_RuleML

precisely where they are needed. Once invoked, the Web service determines the string-type input value `Arthritis`, extracts it from the RDF input and replaces `'?'` with `Arthritis` in the `WHERE` clause, making the instantiated SQL query executable over the database:

```
SELECT patient.name AS patName
FROM patient, disease, patientdiagnosis
WHERE  disease.name = "Arthritis"
AND patient.id = patientdiagnosis.patient_id
AND disease.code = patientdiagnosis.code
```

After the call and execution, the service returns the output RDF file containing a list of patient names `John Doe`, `Bob`, `Alice` etc. having `Arthritis`, each extracted from the tables in Fig. 1.

The following figure shows a graphical representation of the above RDF input and output instances:

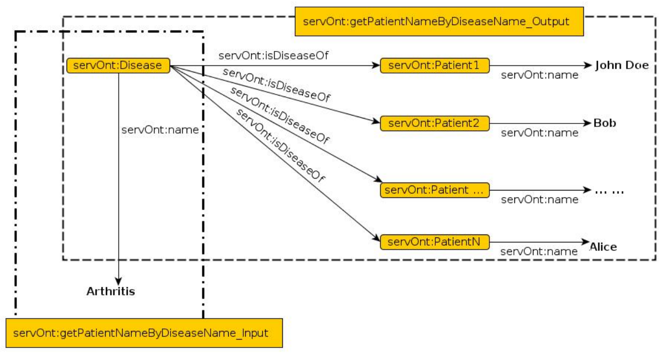

Figure 3. Service Input and Output Instances

The generated SADI service can be invoked and tested by issuing a simple SPARQL query in SADI query clients such as Hydra and SHARE. User asking *Which patients have Arthritis*?, issues the following SPARQL query:

```
PREFIX servOnt: <http://unbsj.biordf.net/servOnto.owl#>
SELECT DISTINCT ?patientName
WHERE
{ servOnt:Patient servOnt:name ?patientName.
  servOnt:Disease servOnt:isDiseaseOf servOnt:Patient.
  servOnt:Disease servOnt:name "Arthritis". }
```

IV. Architecture

The Web service generation process is best described by the main components (modules) of the architecture shown in Fig. 4.

A. Module for Declarative Descriptions of the Service

Declarative service descriptions are composed of the properties along with the class names and various logical connectives from the ontology(ies) as shown by the input and output classes in Section II using Protégé syntax.

B. Module for Semantic Mapping of Databases in PSOA

This module provides mappings between ontologies and databases using the Datalog sublanguage of the expressive Web rule language PSOA RuleML. The SQL queries and pseudo-RDF indicate how relational data is mapped.

The PSOA rule below embodies the semantic mapping of HAI-related data from TOH DW. Lines 12-18 essentially represent the SQL query while lines 1-9 and 19-23 capture the meaning of the pseudo-RDF. The relations among SQL queries and pseudo-RDF with these rules are exemplified in [4], [5] in detail.

```
1 And
2 (
3   ?diagnosis # haio:Diagnosis()
4   haio:is_performed_for(?diagnosis ?patient)
5   haio:identifies(?diagnosis ?disease)
6   ?disease # haio:Disease()
7   ?disease # ?diseaseClass()
8 )
9   :-
10 And
11 (
12  ?encounterRow #
13     dwt:Nencounter(dwa:encWID->?encounterID
14                    dwa:encPatWID->?patientID)

15  ?diagnosisRow #
16     dwt:NhrDiagnosis(dwa:hdgWID->?diagnosisID
17                      dwa:hdgHraEncWID->?encounterID
18                      dwa:hdgCd->?diseaseCode)
19  ?patient = External(modf:Patient_by_patWID(?patientID))
20  ?diagnosis = External(modf:Diagnosis_by_hdgWID(?diagnosisID))
21  ?diseaseClass
22         = External(modf:disease_class_by_ICD10(?diseaseCode))
23  ?disease = External(modf:Disease_by_diagnosis(?diagnosisID))
24 )
```

C. SQL Query Template Generation Module

The generation of SQL queries requires declarative service descriptions, semantic mapping of the database, and ontology (semantic schema) as the inputs. Our architecture will be using the IQR technique because it facilitates such SQL generation. The IQR technique takes the inputs and generates a (possibly infinite) number of SQL queries.

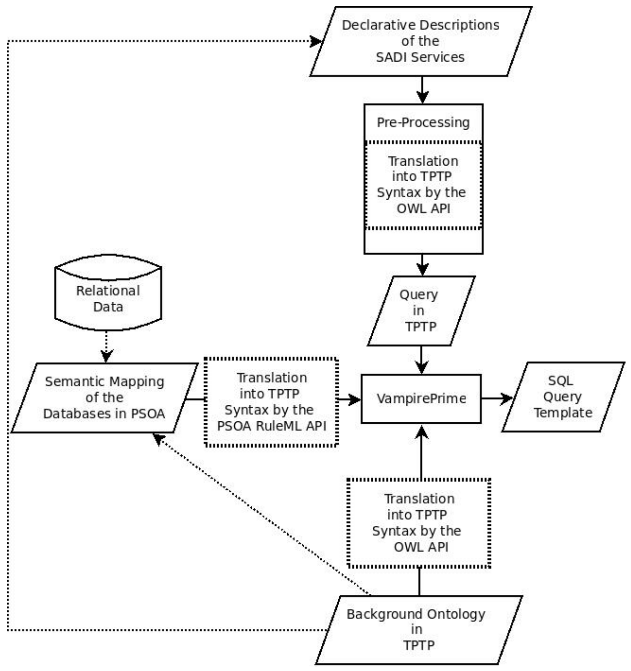
Declarative Descriptions of the SADI Services
Pre-Processing
Translation into TPTP Syntax by the OWL API
Query in TPTP
Relational Data
Semantic Mapping of the Databases in PSOA
Translation into TPTP Syntax by the PSOA RuleML API
VampirePrime
SQL Query Template
Translation into TPTP Syntax by the OWL API
Background Ontology in TPTP

Figure 4. Architecture

D. Service Generator Module

The service generator module generates Java code for the Web service. The service code consists of three parts: reading input RDF, business logic and writing RDF output. The code for reading input and writing output are generated based on the input and output class definitions, respectively. The service code when executed, reads input RDF and places all input values in appropriate places of the generated code as well as in the generated SQL template. Finally, the data drawn from the database is presented as RDF output according to the modeling of the output class. Thus, the module ensures automatic generation of a fully functional Web service code with no human intervention.

V. Discussion, Ongoing Work and Evaluation

The open-source D2RQ platform uses a declarative language and employs a tool called D2R server, which uses a customizable D2RQ mapping to map database contents into RDF and allows users to issue SPARQL queries which are rewritten into SQL queries via the mapping. MASTRO is an ontology-based data integration tool. The mapping language in MASTRO allows for expressing Global-As-View mappings, answers unions of conjunctive queries, and it provides a sound and complete query answering algorithm for a rather restricted logic fragment. For our work, we plan to adopt the IQR technique which is based on a sound and complete algorithm that works with a full first-order logic, but without a general termination guarantee and rewrites TPTP queries into SQL queries. Results from initial experiments show that simple SQL queries can be generated without problem. We plan to address complex query generation, case-by-case, in future.

Unlike D2RQ which exposes the database as a virtual RDF graph, in our approach, semantic mappings are written to map the existing ontology and the relational database. Any changes occurring in the database schema must be reflected in the mappings and such modifications are to be written by the database programmers. The mappings allow decoupling of applications from the database design. Should there be changes in design, the applications need not be changed provided that suitable mappings can be written for the new design. A detailed description of the semantic mappings is beyond the scope of this paper, we plan to address this issue in future.

For generating SQL queries by the VampirePrime engine, three inputs are required: semantic mappings, the ontology and the declarative descriptions. As VampirePrime can process only TPTP syntax, three translators are necessary for processing these inputs. We plan to reuse and modify existing tools such as the OWL API, the PSOA RuleML API, and PSOA2TPTP for the translation tasks.

In general, relational data are URI-free while any entity in a Web ontology is identified by a URI. Hence, efficient handling of URIs is important. As formulas in a Web rule language, PSOA rules can easily use entities with or without URI. We plan to use URI constructing functions for URI handling.

A list of HAI use cases has been identified in [5]. A thorough evaluation of our system can be performed by generating HAIKU SADI services that leverage these use cases and run on HAI data stored in the TOH DW.

References

[1] C. Bizer and A. Seaborne, "D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs," in *ISWC2004 (posters).*

[2] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Lembo, A. Poggi, and R. Rosati, "MASTRO-I: Efficient Integration of Relational Data through DL Ontologies." in *Description Logics*.

[3] A. Riazanov and M. A. T. Arago, *Incremental Query Rewriting with Resolution*, ser. Canadian Semantic Web II. Springer US, 2010.

[4] A. Riazanov, G. W. Rose, A. Klein, A. J. Forster, C. J. Baker, A. Shaban-Nejad, and D. L. Buckeridge, "Towards clinical intelligence with SADI semantic web services: a case study with hospital-acquired infections data," in *Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences*, ser. SWAT4LS '11.

[5] A. Riazanov, A. Klein, A. Shaban-Nejad, G. W. Rose, A. J. Forster, D. L. Buckeridge, and C. J. O. Baker, "Semantic querying of relational data for clinical intelligence: a semantic web services-based approach," *J. Biomedical Semantics*.

[6] M. Wilkinson, B. Vandervalk, and L. McCarthy, "The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation," *Journal of Biomedical Semantics*, vol. 2, no. 1, p. 8, 2011.

[7] G. Rose, V. Roth, K. Suh, M. Taljaard, C. Van Walraven, and A. Forster, "Use of an electronic data warehouse to enhance cardiac surgical site surveillance at a large canadian centre." *Clin Invest Med*, vol. 31, no. 4, p. S21, 2008.

[8] H. Boley, "A RIF-style semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules," in *Proceedings of the 5th international conference on Rule-based reasoning, programming, and applications, in RuleML'2011.*

[9] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0," *Journal of Automated Reasoning*, vol. 43, no. 4, pp. 337–362, 2009.

[10] M. S. A. Manir, A. Riazanov, H. Boley, and C. J. O. Baker, "PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes." in *RuleML'2012*.

[11] G. Zou, R. Peter-Paul, H. Boley, and A. Riazanov, "PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners." in *RuleML'2012*.

[12] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Semantic Web*, vol. 2, no. 1, pp. 11–21, 2011.