Automapper: Relational Database Semantic Translation using OWL and SWRL

Matthew Fisher and Mike Dean

Abstract — General purpose query interfaces to relational databases can expose vast amounts of content to the Semantic Web. In this paper, we discuss Automapper, a tool that automatically generates data source and mapping ontologies using OWL and SWRL. We also describe the use of these ontologies in our Semantic Distributed Query architecture, an implementation for mapping RDF queries to disparate data sources, including SQL-compliant databases, using SPARQL as the query language. This paper covers Automapper functionality that exploits some of the expressive nature of OWL to produce more accurate translations. A comparison with related work on Semantic Web access to relational databases is also provided.

Index Terms — Ontologies, OWL, RDBMS, Semantic Web, SWRL

1 INTRODUCTION

A wealth of information resides in relational databases, which are highly engineered for scalability, transaction management, security and access by existing applications [1]. Access to this information significantly increases the utility of the Semantic Web. Dynamic access is preferred, to accommodate high data volumes and change rates. Custom servlets or other programs can export high-quality semantic representations, but the development cost is often prohibitive and can limit reusability. This motivates the development of application-independent tools that can generate a basic ontology from a database schema and dynamically produce instance data using that ontology. This method quickly exposes the data to the Semantic Web, where a variety of tools and applications are available to support translation, integration, reasoning, and visualization.

The paper presents one such tool, Automapper, and is organized as follows: Sections 2 and 3 describe both Automapper and the overall architecture in which Automapper operates. Section 4 discusses the current use of OWL while Section 5 details the current use of SWRL. Section 6 provides a simple application of Automapper and the Semantic Distributed Query architecture. Section 7 covers related work. Section 8 concludes with future directions.

2 ARCHITECTURE

To understand Automapper's utility, a description of its subsuming architecture is provided. While that is not the focus of this paper, a general understanding is necessary to comprehend Automapper's role in the system as a whole. As shown in Fig. 1, Automapper is part of a larger system for decomposing a SPARQL query, expressed using a domain ontology, over multiple data sources and merging the corresponding data into a single result set [2]. Specifically, Automapper uses the database schema to create an OWL data source ontology and mapping instance data (the mapping ontology is discussed in Section 3) to support two layers of processing: the higher-level Semantic Query Decomposition component (SQD) and the lower level SPARQL-to-SQL translation component, also known as a Semantic Bridge for Relational Databases (SBRD). Each RDBMS has its own Semantic Bridge instantiation that can be either collocated or hosted remotely. SQD relies on a set of SWRL data source-to-domain mapping rules and domain-to-domain mapping rules to properly translate inbound queries into data source queries and vice-versa. The use of a domain ontology allows queries to be independent of any particular data source. In addition, different communities can each adopt their own domain ontology while reusing the same data sources. Mapping rules need only cover the
Fig. 1. Overview of the Semantic Distributed Query Architecture

data of interest thereby minimizing integration costs. SBRD uses both Automapper artifacts to correctly map a SPARQL query expressed using the data source ontology into SQL SELECT statements. Database query result sets are mapped back into the data source ontology and returned to SQD. As a final step, SQD recombinesthe various result sets, maps the data into the domain ontology and returns this data to the user. Automapper was developed in Java and needs to be run only once against a given relational database to automatically generate both the data source ontology and the mapping data. Using the method java.sql.Connection.getMetaData(), we are able to mine the necessary table definitions such as column names, primary keys, foreign keys, remarks and type declarations to generate the data source ontology. Automapper constructs these artifacts based on a configuration file, which contains instance data based on a simple configuration ontology. This ontology permits additional primary and foreign key information, the option to include or exclude comments in the mapping instance data, the capacity to limit the visibility of specific tables and the ability to override the declared column datatype with another XML Schema datatype [3].

3 Utilizing Automapper Mapping Data

While SQD relies on SWRL rules for mapping, SBRD depends on Automapper’s generated mapping data. SBRD employs these mappings for transforming SPARQL data source queries into SQL queries for data retrieval. Our mapping ontology, based on the mapping ontology used by D2RQ [4], defines a ClassMap OWL class whose instances represent each table in a given schema. A table has a corresponding OWL class in the data source ontology, an owning schema, a name, a uriPattern and one or more datatype property bridges and object property bridges. The uriPattern is an OWL datatype property that identifies each row in a table (instance) with a unique URI by concatenating its table name with the value of each primary key column in the table. The datatype and object property bridges correspond to columns containing literal values and foreign keys, respectively.

4 Use of OWL

The following class descriptions, axioms and restrictions are currently generated by Automapper:

1. maxCardinality is set to 1 for all nullable columns and is used for descriptive purposes
2. cardinality is set to 1 for all non-nullable columns and is used for descriptive purposes
3. All datatype and object properties that represent columns are marked as FunctionalProperties. To ensure global uniqueness and class specificity, these columns are given URLs based on concatenating the table and column names
4. allValuesFrom restrictions reflect the datatype or class associated with each column and is used for descriptive purposes

5 Use of SWRL

Automapper generates SWRL rules to identify identical individuals based on their primary key values. These rules use swrl:SameIndividualAtom statements to express class-specific inverse functional relationships, including those classes involving multiple properties (primary key columns), neither of which is directly supported by OWL [5]. The inclusion of these rules can reduce the number of SPARQL variables and statements used by both SQD during the query decomposition process and Semantic Bridges during translation. The resulting SQL queries are more concise and, in certain cases, execute in a shorter period of time.

6 Example

Assume a simple OWL domain ontology of personnel information used by a human resources department. This ontology defines
classes Person and Dept and datatype properties gender, name, code and projectName\(^1\). It also defines an object property, department, used to associate a Person with a Dept. A Dept is uniquely identified by its code. Below is a fragment represented in Turtle:

```turtle
:Person a owl:Class;
    rdfs:subClassOf
    [ a owl:Restriction;
      owl:onProperty :name;
      owl:maxCardinality "1"^^xsd:nonNegativeInteger 
    ],
    [ a :Restriction;
      owl:onProperty :name;
      owl:allValuesFrom xsd:string ]

:Dept a owl:Class;
    rdfs:subClassOf
    [ a :Restriction;
      owl:onProperty :code;
      owl:maxCardinality "1"^^xsd:nonNegativeInteger 
    ],
    [ a :Restriction;
      owl:onProperty :code;
      owl:allValuesFrom :Dept ]
```

A domain model will often incorporate data from multiple databases but not all terms defined in a data source ontology will map to the domain ontology. For the sake of brevity, we limit our example to a single database (hresources) that holds staffing information; Tables 1 and 2 list the contents of the Staffing and Departments tables, respectively. Note that the Department ID column is only partial dependent on the primary key (an employee belongs to one department, independent of a project) and therefore the table is not in second normal form. Unfortunately, such usage is not uncommon in practice and is included here as a real-world example.

From this schema, Automapper creates the data source ontology and class-specific inverse functional rules, of which fragments are listed below:

```turtle
dsont:Hresources.Departments a owl:Class;
    rdfs:subClassOf
    [ a owl:Restriction;
      owl:onProperty dsont:hresources.departments.id;
      owl:allValuesFrom xsd:decimal 
    ],
    [ a owl:Restriction;
      owl:onProperty dsont:hresources.departments.id;
      owl:maxCardinality "1"^^xsd:nonNegativeInteger 
    ].

dsont:Hresources.Staffing a owl:Class;
    rdfs:subClassOf
    [ a owl:Restriction;
      owl:onProperty dsont:hresources.staffing.name;
      owl:maxCardinality "1"^^xsd:nonNegativeInteger 
    ],
    [ a owl:Restriction;
      owl:onProperty dsont:hresources.staffing.name;
      owl:allValuesFrom xsd:string 
    ],
    [ a owl:Restriction;
      owl:onProperty dsont:hresources.staffing.deptid.Object;
      owl:maxCardinality "1"^^xsd:nonNegativeInteger 
    ].
```

\(^1\) The association between Person and a Project could also be appropriately modeled using an object property.

---

**TABLE 1**

**STAFFING TABLE**\(^a\)

<table>
<thead>
<tr>
<th>Name</th>
<th>Project</th>
<th>DeptID</th>
<th>Hours</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>MattF</td>
<td>Alpha</td>
<td>1</td>
<td>100.5</td>
<td>Developer</td>
</tr>
<tr>
<td>MikeD</td>
<td>Alpha</td>
<td>2</td>
<td>50.2</td>
<td>Tech Lead</td>
</tr>
<tr>
<td>MattG</td>
<td>Beta</td>
<td>1</td>
<td>92.0</td>
<td>Architect</td>
</tr>
<tr>
<td>DaveK</td>
<td>Beta</td>
<td>1</td>
<td>120.0</td>
<td>Developer</td>
</tr>
<tr>
<td>MikeD</td>
<td>Beta</td>
<td>2</td>
<td>30.8</td>
<td>Consultant</td>
</tr>
<tr>
<td>DaveK</td>
<td>Alpha</td>
<td>1</td>
<td>87.8</td>
<td>Indigator</td>
</tr>
</tbody>
</table>

\(^a\) Name and Project together form a primary key and DeptID is a foreign key.

**TABLE 2**

**DEPARTMENTS TABLE**\(^b\)

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System Solutions</td>
</tr>
<tr>
<td>2</td>
<td>Research and Development</td>
</tr>
<tr>
<td>3</td>
<td>Management</td>
</tr>
</tbody>
</table>

\(^b\) ID is a primary key.

```turtle
owl:onProperty
dsont:hresources.departments.id;
owl:allValuesFrom xsd:decimal,
[ a owl:Restriction;
  owl:onProperty
dsont:hresources.departments.id;
  owl:maxCardinality "1"^^xsd:nonNegativeInteger 
].
```

```turtle
owl:onProperty
dsont:hresources.staffing.name;
owl:maxCardinality "1"^^xsd:nonNegativeInteger,
[ a owl:Restriction;
  owl:onProperty
dsont:hresources.staffing.name;
  owl:allValuesFrom xsd:string,
  [ a owl:Restriction;
    owl:onProperty
dsont:hresources.staffing.deptid.Object;
    owl:maxCardinality "1"^^xsd:nonNegativeInteger 
].
```

```turtle
[ a swrl:ClassAtom;
  swrl:argument1 :A;
  swrl:classPredicate dsont:Hresources.Departments ]
[ a swrl:ClassAtom;
  swrl:argument1 :B;
  swrl:classPredicate dsont:Hresources.Departments ]
[ a swrl:DatavaluedPropertyAtom;
  swrl:argument1 :A;
  swrl:argument2 :Var0;
```
swrl:propertyPredicate
dsont:hresources.departments.id ] ) ;
ruleml:body
( [ a swrl:DataValuedPropertyAtom ;
swrl:argument1 :B ;
swrl:argument2 :Var0 ;
swrl:propertyPredicate
dsont:hresources.departments.id ] ) ;
ruleml:head
( [ a swrl:SameIndividualAtom ;
swrl:argument1 :A ;
swrl:argument2 :B ] ) .

The corresponding mapping data is also generated. Below are two datatype property bridges, an object property bridge (representing a foreign key) and a class map all relating to the Departments table:

:HRESOURCES.DEPARTMENTS.ID a map:DatatypePropertyBridge;
map:column "ID";
map:datatype xsd:decimal;
map:language "en";
map:property
dsont:hresources.departments.id .

:HRESOURCES.DEPARTMENTS.NAME a map:DatatypePropertyBridge;
map:column "NAME";
map:datatype xsd:string;
map:language "en";
map:property
dsont:hresources.departments.name .

:HRESOURCES.STAFFING.DEPTID.OBJ a map:ObjectPropertyBridge;
map:constraint
[ a map:KeyConstraint;
map:objectColumnOperand "ID";
map:operator :EqualsOperator;
map:subjectColumnOperand "DEPTID" ];
map:objectClassMap
dsont:Hresources.Departments;
map:property
dsont:hresources.staffing.deptid.Object .

:)Hresoures.Departments a map:ClassMap;
map:datatypePropertyBridge
 :HRESOURCES.DEPARTMENTS.ID,
 :HRESOURCES.DEPARTMENTS.NAME;
map:table "DEPARTMENTS";
map:type dsont:Hresources.Departments;
map:uriPattern
"http://example.org/2007/08/ds-ont#
Hresources.Departments@@ID@@" .

In the final step, we create SWRL rules to map between the data source Departments class to the domain ontology Dept class including the department ID:

:RuleDeptClass a ruleml:Imp ;
ruleml:body
( [ a swrl:ClassAtom ;
swrl:classPredicate
dsont:Hresources.Departments ;
swrl:argument1 :d ] ) ;
ruleml:head
( [ a swrl:ClassAtom ;
swrl:classPredicate domont:Dept ;
swrl:argument1 :d ] ) .

:RuleDepartmentCode
ruleml:body
( [ a swrl:ClassAtom;
swrl:classPredicate
dsont:Hresources.Departments ;
swrl:argument1 :d ]
[ a swrl:DataValuedPropertyAtom ;
swrl:propertyPredicate
dsont:hresources.departments.id ;
swrl:argument1 :d ;
swrl:argument2 :c ] ) ;
ruleml:head
( [ a swrl:DataValuedPropertyAtom ;
swrl:propertyPredicate domont:code ;
swrl:argument1 :d;
swrl:argument2 :c ] ) .

Running the following query for the names of all people in department 1 and their associated project names:

PREFIX :<http://example.org/2007/08/domain-ont#>
SELECT ?pName ?name
WHERE {
[ a :Person ;
 :projectName ?pName ;
 :name ?name ;
?d :code 1 ]
}
yields:
Alpha MattF
Alpha DaveK
Beta MattG
Beta DaveK

7 Related Work

Various tools have been developed to provide Semantic Web interfaces to relational databases including D2RQ, Gnowsis [6], ISENS [7], Relational.OWL [8] and OntoGrate
We initially used D2RQ and incorporated several modifications which we submitted to the D2RQ development team\(^2\). The changes involved query optimizations such as eliminating duplicate table prefixing, increased selectivity of property bridges (to limit the desired number of tables in a query), and SQL SELECT query partitioning for smaller queries. As SQD became more sophisticated, we determined that SBRD and other Semantic Bridges did not require the full power of D2RQ.

Automapper corresponds to D2RQ’s generate-mapping\(^3\) script. It continues the D2RQ use of instance data for SPARQL-to-SQL mappings, entity concepts (e.g. ClassMap, uriPattern) and separate datatype and object property bridges. Object property bridges more precisely model foreign key relationships, although our model is not the only possibility [10]. The d2r:join property has been replaced with a Constraint class in the mapping ontology. A Constraint is modeled as a binary operation with a single operator and two operands. Further precision is captured in KeyConstraint, a subclass of Constraint, which limits the operator to equality. Thus, KeyConstraint conceptually combines d2r:refersToClassMap and d2r:join. d2r:AdditionalProperty and d2r:additionalProperty are not used since the data source ontology is a straight-forward model of the RDBMS schema. This simplified representation makes Automapper-generated artifacts easier to create and understand.

8 FUTURE WORK

We are currently applying Automapper’s approach to other Semantic Bridges. Specifically, we are exploring its use for both SOAP and RESTful services in our Semantic Bridge for Web Services (SBWS). Regarding SBRD’s use of Automapper, we want to expand inspection into the data to better refine the database model, such as data typing. For example, instead of relying solely on the database schema to declare a column to contain integers, we could restrict that type to only natural numbers using xsd:positiveInteger\(^4\). Automapper currently achieves this goal by overriding the datatype in the configuration file but automated discovery would be beneficial. The use of type definitions and data range restrictions as part of OWL 1.1 [11] would be helpful for SQD in determining valid queries during decomposition. Testing the applicability of new object property axioms based on extant data (e.g. reflexiveObjectProperty) is also planned. Currently, URIs returned by SBRD are unique but generally not resolvable. We intend to address this issue in future versions by generating resolvable URIs.

REFERENCES


Matthew Fisher earned a Bachelor’s Degree in Computer Science from The Pennsylvania State University in 1995 and a Master’s Degree in Computer Science from George Mason University in 2000. He is currently a Senior Software Engineer at BBN Technologies and is a member of both IEEE and ACM.

Mike Dean earned a Bachelor’s Degree in Computer Engineering from Stanford
University and performed graduate work at the University of Rochester. Mike is a principal engineer at BBN Technologies. He has been involved in the Semantic Web since 2000, serving as Principal Investigator for the DARPA Agent Markup Language (DAML) Integration and Transition contract, chair of the Joint US/EU ad hoc Agent Markup Language Committee (responsible for DAML+OIL and SWRL), an Invited Expert on the W3C RDF Core and Web Ontology Working Groups, and as Principal Investigator for various Semantic Web pilot projects. Mike is on the program committee for the Sixth International Semantic Web Conference and is a senior member of IEEE.