OilC: Schema Mappings Library

Ivo Jimenez

March 15, 2012

Abstract. Currently, there is no open source tool that provides with the functionality required to implement schema mappings. In this work we introduce OilC, a language-agnostic schema mapping library that can be extended through the use of plugins. We describe its internals and sample use cases, as well as a brief description of the algorithms that implements.

Introduction

Current schema mapping tools are closed-source. In this work we describe the design and implementation of OilC (pronounced oil-sea), a query language-agnostic framework to implement schema mappings that can be extended through the use of plugins.

The rest of the paper is organized as follows. Section 2 describes the background and related tools. Section 3 gives a high-level overview of the architecture. Section 4 details the internals of each component. Section 5 describes a sample use case. We conclude and outline future work in Section 6.

Background

Query Graph-based Approach

OilC implements the Query Graph-based approach first introduced in [Popa et~al., 2002] and described in more detail in [Fagin et~al., 2009].

Existing Tools

Existing tools such as IBM Clio[1], HePToX, Altova MapForce, Stylus Studio, MS Biztalk Mapper, BEA Aqualogic are all closed-source schema mapping

[1]The careful reader might have noticed that OilC is Clio backwards.

implementations. The goal of OilC is to provide the same functionality, through an open source implementation.

OilC Architecture

As described in [Fagin et~al., 2009], the schema mapping flow can be seen as comprised of six phases:

1. Metadata access (reading of source and target schemas)
2. Associations Generation (schema- and user-defined)
3. Generation of mappings from associations
4. Obtain the query graph
5. Annotate the graph
6. Translate to specific language
7. Merge multiple mappings

The work described in this paper implements 4-6, that is, we don't access the metadata catalogs in order to read the whole source and target schemas and thus we can't check the semantics of a mapping (1); consequently, we assume that mappings given by the user are correct and that have been correctly generated (2-3); and lastly, we don't merge multiple mappings (7).

The architecture of OilC is shown in Figure 1. Given a schema mapping in literal form (`String` type), the parser generates a schema mapping specification. This specification is then passed to the query generator and then to the annotator, both of which implement the procedures described in section 5.2 of [Fagin et~al., 2009]. Subsequently, the

OilC Implementation

OilC is implemented in Scala (specifically Scala 2.10) and uses the *Graph For Scala* library [2] to handle graph-related actions. It also uses ScalaTest 1.9 [3] to implement the tests. This section describes how the source code maps the architecture described in the previous section.

[2]https://www.assembla.com/spaces/scala-graph/wiki
[3]http://www.scalatest.org/

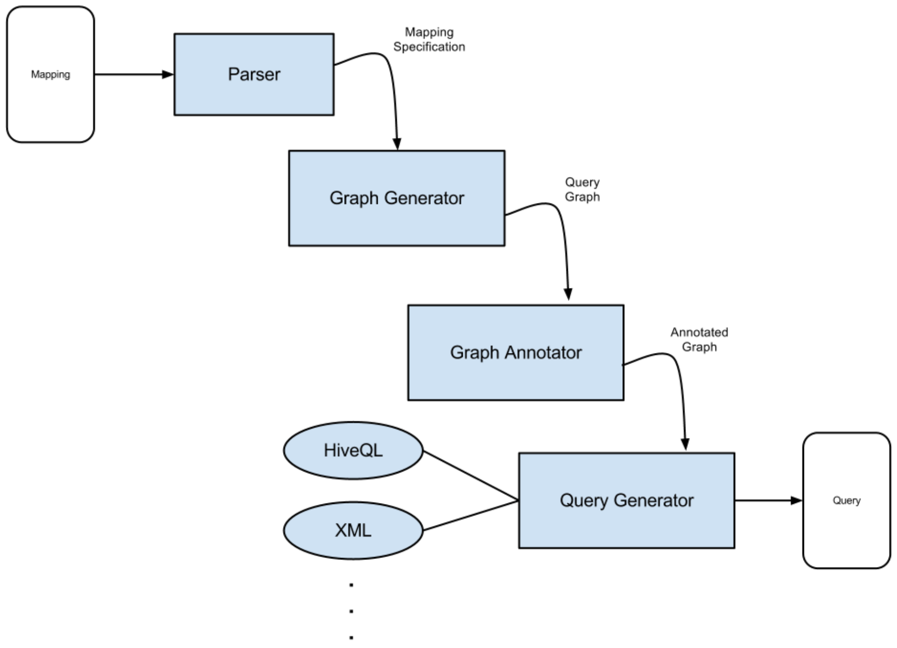

Figure 1: Architecture of OilC

Schema Mapping Parser

The schema mapping parser is contained in `edu.ucsc.oilc.SchemaMappingParser`. It implements the schema mapping DSL through the use of Scala's parser combinators[4]. As mentioned before, the output of the parser is a schema mapping specification that separates each of a schema mapping's clause contents.

Query Generator

The query generator, implemented in `edu.ucsc.oilc.QueryGraph#generateGraph` takes as input the schema mapping specification and generates the query graph.

Query Annotator

The query annotator is implemented in `edu.ucsc.oilc.QueryGraph#annotateGraph` and implements the annotation propagation algorithm described in Section 2. The outcome is the annotated graph, which contains all the information that is needed in order to generate a query that can populate the target instance.

Query Generation

The last piece of our current implementation is the language-specific translation. In our current implementation, we have a HiveQL translator (located in `edu.ucsc.oilc.HiveTranlator`) that takes as input the schema mapping specification (from which it generates the subquery corresponding to the source subgraph) and the annotated graph. It implements the depth-first traversal described in Section 2.

Use Case

Sample use cases can be found in the `src/test/` folder.

Conclusion and Future Work

In this paper we have described OilC, a schema mapping open source library. As future work, we would like to add the ability to read the source and target schemas from actual metadata catalogs. This could be implemented in a plugin-based way, such that different types of catalogs are supported, eg. HCatalog.

[4]http://www.codecommit.com/blog/scala/the-magic-behind-parser-combinators

Also, generating mappings (from associations) and merging multiple mappings would allow to support the entire schema mapping workflow.

Lastly, creating a test suite that takes its inputs from the STBenchmark [Alexe et~al., 2008] and checks against the expected mappings would allow to prove the robustness of the library.

References

Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proc. VLDB Endow.*, 1(1):230–244, August 2008. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1453856.1453886.

Ronald Fagin, Laura~M. Haas, Mauricio Hernández, Renée~J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In Alexander~T. Borgida, Vinay~K. Chaudhri, Paolo Giorgini, and Eric~S. Yu, editors, *Conceptual Modeling: Foundations and Applications*, number 5600 in Lecture Notes in Computer Science, pages 198–236. Springer Berlin Heidelberg, January 2009. ISBN 978-3-642-02462-7, 978-3-642-02463-4. URL http://link.springer.com.oca.ucsc.edu/chapter/10.1007/978-3-642-02463-4_12.

Lucian Popa, Yannis Velegrakis, Mauricio~A. Hernández, Renée~J. Miller, and Ronald Fagin. Translating web data. page 598–609, 2002. URL http://dl.acm.org/citation.cfm?id=1287369.1287421.