

WSIRD: Web Services Integration via Rules for Data Transformation

Sandy Liu and Bruce Spencer

Institute for Information Technology – e-Business

National Research Council of Canada

{sandy.liu,bruce.spencer}@nrc.gc.ca

Abstract

We suppose that two Web Services with semantically similar data models are to be composed so that the output of one contributes to the input of the other. The task is made easier by having access to OWL-S descriptions of each Web Service, in particular the syntax and semantics of their data models. This paper describes a running prototype of the Web Services Integration via Rules for Data Transformation (WSIRD) system, which consists of two parts: an off-line system that infers data transformation rules, and an on-line system that runs the rules effecting the transformation. The rules can incorporate calls to data conversion and type casting procedures to accommodate differences in the two data models.

1 Introduction

Many works in Semantic Web Services composition have focused on matchmaking at the conceptual level where the details of how data will be passed among a set of composed services are often ignored. For instance, the output of a Web Service that can provide lists of available flights ideally can be used as input for another Web Service for flight reservation. In reality, with the distributed and independent nature of Web Services, the output of the first Web Service often cannot be used directly as input to the second one. To bridge this gap, the objective of our work is to generate an appropriate message required by the downstream Web Service from the message(s) of the upstream Web Services. This paper gives an overview on how this integration process can be achieved with our proposed solution: WSIRD (Web Service Integration via Rules for Data transformations).

2 WSIRD: An Overview

WSIRD is designed based on the assumption that there exist rich semantic descriptions for both the upstream and downstream Web Services. WSIRD uses our customized Description Logic reasoner to generate a set of data transformation rules based on semantic descriptions of the services to be integrated. This set of transformation rules is then imported to our Inference Queue, a queuing inference engine. During runtime, the upstream message is passed through the Inference Queue, which runs the rules and produces the required input message for the next service.

As OWL became a W3C Recommendation, we use OWL-S [Coalition, 2004], the OWL-based ontology for Web Services, for describing the semantic information for each Web Service. OWL-S describes Web Services in three levels of abstraction: the *service profile*, the *process model*, and the *service grounding*, describing respectively what the service does, how it does it, and how to access it. In WSIRD, we assume that matchmaking or service composition is done based on the abstract description (i.e. Service profile or/and process model). WSIRD then handles the semantic Web Services integration at the grounding level, where the complete details are found for each of the message components. WSIRD can be divided into two sub-systems: the off-line system and the on-line system.

2.1 The Off-line System

The off-line system is used before the actual service invocation in order to prepare necessary transformation rules for the on-line system. Figure 1 shows the components of the off-line system. The following section describes the two key components in the off-line system: the *Rule Compiler* and the set of *Converters*.

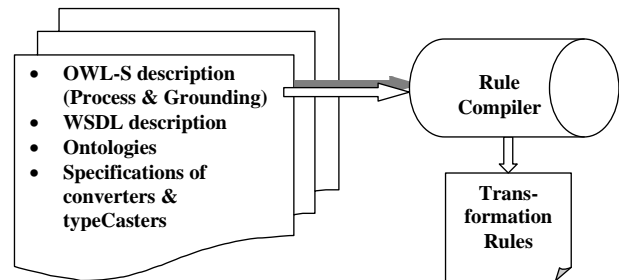


Figure 1: The components of the off-line system

The Rule Compiler

As illustrated in Figure 1, the input of the rule compiler includes the OWL-S descriptions, the WSDL descriptions, and auxiliary ontologies for both upstream and downstream Web Services. It also includes the specifications of converters, which will be discussed in the subsequent section. The rule compiler makes use of a tableau *ALC* Description Logic reasoner (written in Prolog). According to these semantic information, the rule compiler can find the correspondences between components of the upstream message and the downstream one. It generates data transformation rules in first-order logic. Should the integration is deemed

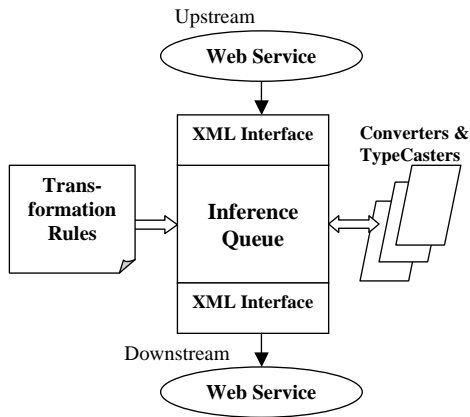


Figure 2: The components of the on-line system

to be infeasible, the rule compiler will be unable to complete its task; other than informing the user we do not consider in this paper what other actions might be appropriate.

The Converters

The *converters* are built-in components for converting data from one type to another. In the real world many data types that are not identical syntactically are still convertible, such as temperatures expressed in either Fahrenheit or Celsius, currency in either dollars or Yen, etc.

A *TypeCaster* is a special type of converter. It performs low-level type conversion such as converting between string and numeric. The functions of all the converters and typeCasters can be specified, then they can be used by the rule compiler to generate proper transformation rules. As a result, a number of these standard conversions can be expressed as relations in the bodies of rules, and at runtime the inference engine will perform the conversion.

2.2 The On-line System

The on-line system performs the actual service integration at runtime with the fundamental component: the *Inference Queue*. Figure 2 shows the components of the on-line system.

The Inference Queue (IQ) [Spencer and Liu, 2003] is a resolution-based inference engine based on jDREW [Spencer, 2004], which can reason with definite clauses. It is implemented as a priority queue with two main operations: insert and remove, each can be called by separate threads in any order.

The IQ is loaded with the set of transformation rules generated by the off-line system. At runtime it accepts the output message from the upstream Web Service in XML format, encodes it as facts, and draws inferences, so that the inferred results are transmitted as output. Finally, the output is then be converted back to XML and delivered to the downstream Web Service. The inferred results produced by IQ are sound, complete, and irredundant.

2.3 An Example

Suppose that a European-based Web Service producing flight information is selected to provide input to a Canadian-based Web Service accepting flight information. The upstream European flight record has three data items: a source city, a destination city and a cost quoted in Euros. The downstream service expects the flight to be described with two pieces of information: the cost in Canadian dollars and a manifest, consisting of the codes of the departure

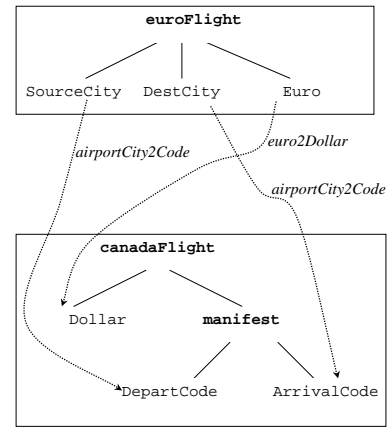


Figure 3: The Upstream to Downstream transformations

and arrival airports. We have a converter from Euros to dollars, and from cities to airport codes. There are three tasks (Figure 3): creating the appropriate structure, converting the currency, and converting the two airport locators. This rule is produced off-line:

```
down(canadaFlight(Dollar, manifest(DepartCode,
    ArrivalCode))):-
    up(euroFlight(SourceCity, DestCity, Euro)),
    euro2Dollar(Euro, Dollar),
    airportCity2Code(SourceCity, DepartCode),
    airportCity2Code(DestCity, ArrivalCode).
```

When the fact `up(euroFlight(Fredericton, Frankfurt, 1200))` is entered into the inference queue, the fact `down(canadaFlight(2400, manifest(YFC, FRA)))` is produced, (assuming that a Euro is two Canadian dollars).

3 Conclusion

We have introduced a rule-based approach for semantic Web Services integration. Given two semantically compatible Web Services, our current prototype WSIRD is capable of generating data transformation rules at composition time, and producing message for the downstream Web Service with preserved meaning during service invocation. This approach compensates for the inflexibility of the third-party code and completes the integration solution after high-level service composition.

References

- [Coalition, 2004] The OWL Service Coalition. OWL-S 1.0 Release. “<http://www.daml.org/services/owl-s/1.0/>”, 2004.
- [Spencer and Liu, 2003] Bruce Spencer and Sandy Liu. Inference Quenes for Communicating and Monitoring Declarative Information between Web Services. In *Proceedings of RuleML-2003*, number 2876 in Lecture Notes in Artificial Intelligence, 2003.
- [Spencer, 2004] Bruce Spencer. A Java Deductive Reasoning Engine for the Web. www.jdrew.org, 2004. Accessed 2004 Jan 12.