Ontology-based Scientific Data Service Composition: A Query Rewriting-based Approach

Linhua Zhou
School of Computer Science,
Zhejiang University, CN
zlhyjy@zju.edu.cn

Huajun Chen
School of Computer Science,
Zhejiang University, CN
huajunsir@zju.edu.cn

Yu Tong, Jun Ma, Zhaohui Wu
School of Computer Science,
Zhejiang University, CN
{ytcs, majun, wzh}@zju.edu.cn

Abstract

Modern scientific research is generating daunting amount of data that is commonly required to be combined together to service daily research endeavors. This paper introduces an ontology-based approach to publishing and composing data-intensive web service. The approach distinguishes itself by its capability of dynamically evolving service interface and a query rewriting-based planner for service composition, enabled by richly describing service capability using the semantic web languages. The advantage of the proposed approach is that the data service can adapt itself with client request dynamically, greatly improving its query service capability. The evaluation reveals the approach scale well as service number goes large.

Motivation

Scientific advancements are growing more and dependent on collaborative researches across multiple institution boundaries. Nowadays, one typical daily research job is to search, identify, translate, and combine data stemming from disparate sources. However, the current web environment is not conducive of research productivity, as the discrepancy and disagreement in data format and interface semantics consist in everywhere around the Web. A plurality of solutions have thus far been prescribed to ease the burdensome of the issue. Chief among them is the web service technologies which have been widely applied in improving the availability of and interoperability between autonomous networked resources.

Conventional web service usually has fixed input/output interface definition and description. As the data generated by modern scientific research grow more and more complicated of format and tremendous of amount, more fined control over data requires more advanced approach to publishing, advertising, and discovering and composing data-intensive web service. However, unlike general web service such as a financial calculation web service, it is difficult to stipulate the input/output for a data service in advance. For example, given a relational database containing information about student such as *id, name,, address, phone* there could be a variety of possible combinations of input/output such as querying address by name or querying name by address, which of them will be used depends on the calling client. Since we can not assume the type of client request and the number of possible combinations increases exponentially against the schema complexity, DS calls for more flexible way of describing and publishing its service interface.

Further, data-intensive web service also raises new challenge in service composition. Data-intensive web service refer to this kinds of web service which provide and process tremendous of data sources, for example query or retrieve information from data sources. As yet, a plethora of solutions have been proposed for web service composition, most of which draws upon a matching-maker that directly matches the interface semantics between service inputs and outputs, either manually or discovered automatically by certain AI-inspired planning approaches (Pistore *et al.* 2005) (Berardi *et al.* 2005) (Esfandiari & Tosic 2005). Whereas, composing DS not only depends on service interface mapping specification, but also relies on the data content, i.e., the data schema and logic structure, for which more expressive and indicative description are required.

This paper presents a data service model characterized by a rich description of the data content, in addition to the interface description, of data service based on semantic web languages. Specifically, the model is established by describing the content semantics by using a set of RDF graph patterns. The terms used in these graph patterns are defined in the application domain ontology. Essentially, the RDF graph pattern that provides more expressive and comprehensive description about DS, defines the mapping from a local data source to a web ontology. The data mapping specification crucial of both service discovery and service composition.

Generally speaking, the role of the rich description is two fold within our approach. At first, it is used to dynamically generate WSDL/SOAP-based service interface based on the request of a service client. At second, it is the base of a novel service composition planner that is distinguished by basing on query rewriting approach inspired by conventional research over *answering and rewriting queries using views* (Halevy 2001)(Levy, Rajaraman, & Ordille 1996). Notably, this novel planner is particularly geared towards data-intensive applications, in contrast to conventional approaches for workflow-oriented applications or AI-Planning-oriented application. Because the semantic description provides more expressive and formal description over the data content and service capability of DS, it enables

more selective query interaction, more sensible service discovery and more efficient service composition.

The article is organized as follows. We first illustrate our approach in terms of modelling DS, developing evolvable WSDL/SOAP interface, and service composition. We then introduce an explanatory life science use case, and explain the evaluation result of the proposed algorithms.

Approach

Modelling Data Service

We define a data service as having four major components: the service input *I*, the service output *O*, the content description *CD*, and the constraints *(I, O, CD, CT)*. One typical feature of this model is that the *I/O* part is not fixed, while on the contrary, it will evolve as time goes. CD and CT together offer the base for semantic description of the content and capability of DS.

Content Description The data content refers to the underlying data model and schema used in the databases that can be in relational form or XML form. Although it is obvious that the introduced framework can be adapted to any other legacy proprietary format, we only consider relational data in this paper as it is the most popular data model used as yet.

The data content normally has its own schema definition and thus is heterogenous. Exposing them onto the Semantic Web requires mapping the legacy format to a shared semantic web ontology, making the data semantics understandable to unexpected data consumers. Conventionally, a mapping is expressed as a rule, for example, a mapping from a legacy relational database to a shared ontology can be described as like below:

Example 1 (Mapping example)

```
people(?pid, ?pname, ?age, ?page, ?oid):
            (?pid, rdf:type, :Person),
              .....
            (?pid, :workFor, ?oid).
```

In the rule description, the rule head is a relational predicate *people* , and the rule body is a set of RDF triples in which a set of variables are used. In effect, the mapping rule describe the data semantics of the legacy relational database from the perspective of the shared web ontology.

As many semantic web service proposals have been prescribed and some of them such as the OWL-S [1] has well-defined model, it is natural to develop our model upon currently available approaches. We choose OWL-S as our development foundation.

To be in accord with the OWL-S specification, we encode the mapping information in *process:inCondition* property of *process:Result*, since in OWL-S specification, the *inCondition* property specifies the condition under which the result occurs. As a matter of fact, the mapping information can be considered as a kind of constraint or condition that must be satisfied for query answering.

Further, we propose using SPARQL [2] to describe the mapping. Although some rule language such as SWRL and RIF can be used, but by present time, no standard rule language has been recommended. Taking a simple example, we describe a data service for *db1* as below. The *process:hasResultVar* part defines the variables that are used in the mapping description.

[1] OWL-S: http://www.w3.org/Submission/OWL-S/

[2] SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/

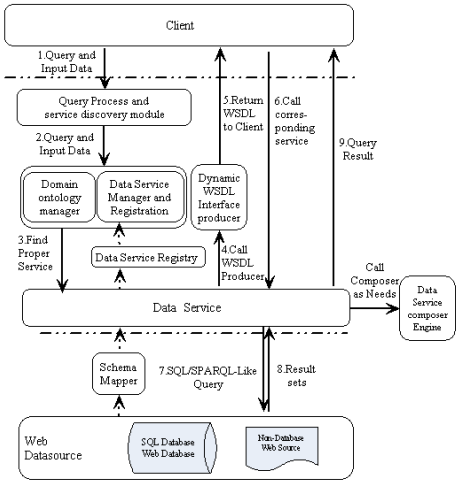

Figure 1: Evolvable WSDL/SOAP Interface.

Example 2 (Content Description Example in OWL-S)

```
<process:AtomicProcess rdf:ID="DS1">
  <process:hasInput> ... </process:hasInput>
  <process:hasOutput> ... </process:hasOutput>
    <process:hasResult>
      <process:Result>
        <process:hasResultVar>
          <process:ResultVar rdf:ID="id" />
        <process:hasResultVar>
            .....
        <process:inCondition>
          <expr:SPARQL-Condition>
            <expr:expressionBody>
              SELECT pid, pname, page, oname, oid
                        FROM db1.people WHERE
                        (pid, rdf:type, :Person),
                         .....
                        (pid, :workFor, oid).
            </expr:expressionBody>
          <expr:SPARQL-Condition>
        </process:inCondition>
        ...
         <process:Result>
     </process:hasResult>
</process:AtomicProcess>
```

Constraint Description Some data sources may have certain constraint description. For example, a data source may only provide information for younger people under age of 20. Another example is a news service in specific city may only provide local news for that city. These types of information are usually not encoded explicitly in database content, however, should be considered as important information when exposing on the semantic web.

The approach is to add an additional constraint description into the mapping description which is stored in the *pro-*

cess:inCondition property of the *process:Result*. For example, the constraint $age < 20$ can be described as below. The $FILTER$ predicate is used to define the constraint.

Example 3 (Constraint Description Example in OWL-S)

```
<process:inCondition>
   <expr:SPARQL-Condition>
      <expr:expressionBody>
         SELECT pid, pname, page, oname, oid
               FROM db1.people WHERE
                  (pid, rdf:type, :Person),
                   .....
                  (pid, :workFor, oid).
                  FILTER (page, <, 20)
    </expr:expressionBody>
  <expr:SPARQL-Condition>
</process:inCondition>
```

Evolvable WSDL/SOAP Interface

As mentioned in Section 1, flexibility and evolvability of DS service interface are both valuable and necessary. The system implements the feature as illustrated in Fig. 1. The basic idea is to utilize the ontology used in both the content description and query request to dynamically generate WSDL/SOAP interface. As a matter of fact, an ontological query can be equally translated into a WSDL specification, as the selection part can be viewed as the output and the variables bound by values in the condition part can be viewed as the input, while the type information is self-described in the ontological classes and properties.

The complete working procedure is depicted in Fig. 1. In our system, it includes two kinds of process: modeling and registering DS process and query process. Modeling and registering DS process is depicted by dashed lines with arrow in Fig. 1, while query process is depicted by solid lines with arrow in Fig. 1. The modeling DS process has been described in the previous section. The content description of a DS is stored globally in a service registration for which a DS manager works to discover proper DSs. Here we focus on the query process in detail as follows: 1.the client describes their query request in terms of shared ontological types and inputs their query request. 2.The query process module tackles the query request and sends the processed query request to the manager. 3.Based on the query request description and the content description of DS, the manager finds out candidate data services. 4.According to the input/output requirement of the query, a request for generating corresponding WSDL/SOAP interface is then sent to the dynamic WSDL interface producer. 5.Then a WSDL description of data service for the new interface is sent back to the client to generate client stubs for real data retrieval. 6.The client calls corresponding data service to retrieve requested data. 7.The data service issues SQL/SPARQL-like query to the real data sources. 8.The result sets return to the data service. 9.Finally, the data service returns the query result to the client.

More complex case may occur when several sources needs to be combined to fulfill the query request. The manager has then one more thing to do, which is to determine what kinds of sources can be combined together to fulfill the query request, other than merely selecting out candidate DSPs. This process requires a query rewriting procedure to determine if these two sources can be combined and on what properties they can be combined, which will be introduced in the following section. After finding out the candidate sources, the requests are sent out simultaneously to dynamic WSDL interface producer to generate the WSDL/SOAP interface, and a query pipe is also generated.

We note that the additional interface generating process may cause a performance issue. To alleviate this issue, the newly generated WSDL/SOAP interface will be saved for future interaction if similar query request comes in. So as time goes, the DS will become full-fledged if sufficient number and types of queries are processed.

Service Composition Based-on Query Rewriting

The rich description provides the foundation for implementating more selective algorithm for service composition. Our algorithm mainly draws upon the query rewriting approach introduced in traditional database community (Halevy 2001). The chief goal of the algorithm is to find out an executable query plan that combines a sequence of data service based on the content description and input/output requirement of the DS. In general, the algorithm has two steps. In the first, we find out candidate data services, and in the second we try to order the services to ensure that they are executable.

The first step of the algorithm is described in details in Algorithm 1. It takes a SPARQL query and a set of service description as input. Firstly, it splits the triples in the query body into a set of tripe groups based on the subject variables used in them.

Definition 1 (Triple Group) *A triple group is a set of triples that have the same subject. It represents a fragment of the query body that can later be matched and combined together, thus making the query rewriting to be easier to implement.*

In the algorithm, the triple group is called a query chunk. Next, for each query chunk, we find out all candidate data services and form a candidate set. The criterion used for the determination is based on whether or not the query chunk is contained or subsumed by the content description of a data service.

The result of Algorithm 1 is a list of candidate sets, each of which corresponds to a query chunk. A candidate plan is generated by selecting one candidate service from each candidate sets, namely, apply a cartesian product operation over these candidate sets.

In the second step, we consider a candidate plan and try to order the candidate services in such a way that the plan will be executable, i.e., will adhere to the input/output requirements of the services. Algorithm 2 describes a process that given a candidate plan, finds a ordering on them, if such an ordering exists. It proceeds by maintaining a list of available parameters, and at every point adds to the ordering any new services whose input requirements are satisfied. Finally, the algorithm outputs a ordered service sequence.

A Life Science Use Case

To describe how the proposed methodology works in practice, we consider a life science scenario. It aims at con-

Require: a set of service descriptions $\mathcal{S} = \{ds_1, ds_2, ..., ds_n\}$, a SPARQL query statement $\mathcal{Q}$.

Split $\mathcal{Q}$ into k chunks; $\mathcal{C} = \{q_1, q_2, ..., q_k\}$ based on the variables used in $\mathcal{Q}$;
Let SC be a empty ordered set.
for all $q_i \in \mathcal{C}$ **do**
  Set $sc_i = \emptyset$
  for all $ds_j \in \mathcal{S}$ **do**
    if $q_i \subseteq ds_j$ **then**
      $sc_i = sc_i \cup ds_i$
    end if
    if ds_j is the last one in $\mathcal{S}$ **then**
      Return "invalid query"
    end if
  end for
  Add sc_i to SC.
end for
Output candidate set list SC

Algorithm 1: Candidate DS selecting Algorithm

necting modern western medicine knowledge system with traditional Chinese medicine(TCM) system to enable cross-culture medical research.

The Case Description TCM physicians have developed an ancient system of TCM syndromes to interpret human morbid states. Electronic medical records show that the relations between TCM syndromes and diseases of western medicine are very complex. For example, the Kidney Yang Deficiency syndrome (KYDS) has relations with hundreds of diseases introduced in western medicine, including diabetes that in turn has relations with several other TCM syndromes. More scientifically sound evidence from modern biomedicine for the principles behind TCM syndromes can promote traditional therapies in personalized healthcare. TCM physicians can conduct an interactive knowledge discovery process based on the domain model targeted at KYDS . In addition to direct relations such as syndrome-disease and syndrome-herb, indirect relations such as syndrome-disease-gene relations are also considered. We will present a use case of our system, which aggregates TCM syndrome knowledge from disparate resources, and facilitate the interpretation of TCM syndrome in terms of modern biomedicine.

The Ontology. We start with a domain analysis to map the two separately developed medical systems into a global ontology, and to seek potential connecting points, among which we currently focus on: (1) the patient that represents Electronic Medical Records and Clinical Trials that faithfully record the methods and results of integrative clinical practices of Western Medicine and TCM, and (2) the chemical that refers to bioactive compounds from Chinese Herbal Medicine, which serve for drug discovery and safety analysis. We then develop an ontology that consists of key concepts from both medical systems as illustrated in Fig. 2. The ontology includes concept description for TCM herb, TCM diagnosis, TCM disease, TCM drug, western medical disease, gene, protein, etc.

Require: a SPARQL query statement $\mathcal{Q} = \{q_1, q_2, ..., q_k\}$ and a corresponding candidate service set $DS = \{ds_1, ds_2, ..., ds_k\}$, .

Let B be the set of variables bound by values in the query, which can be viewed as the input requirement of the query.
Let b_i be the set of variables bound by values either from the original query or from output of some data service at step i.
Let O be the set of variables occurring in the selection part of the query, which can be viewed as the overall output requirement of the query.
Let (In_i, Ou_i) be the input/output requirement of a data service ds_i.
Let S be an empty set.
for all i=1,...,n **do**
  Choose a candidate service $ds_j \in DS$ that was not chosen earlier and the parameters in In_j is in $b_(i-1)$.
  if There is no such a data service **then**
    Return "plan not executable".
  else
    $b_i = b_{i-1} \cup Ou_j$
  end if
  Add ds_j to S
end for
Output a service sequence S.

Algorithm 2: Create executable plan algorithm

Example Data Provider Services. We suppose there are six data services as shown in Table 1. Each data provider service is comprised of four components: content description, input/output description , and an optional constraint description corresponding to the DS model introduced in Section 2. Each service contains data corresponding to one or more than one classes in Fig. 2.

Algorithm Simulation. Suppose the client submits a SPARQL query request that has input information for (patient name (pName), drug name (drgName)) and output information for (gene name (gName) and its id (gid)). The first analysis reveals that the query involves ontological classes for patient, diagnosis, disease, and gene. The first step is to find out candidate DS by making comparison between the input query body and the content description of all DS. This results in a list of candidate DS set {(DS1, DS2), (DS3), (DS5)}. According to the input and output specification and applying our planning algorithm of executable services sequence, we can get two executable service sequence: (DS2, DS3, DS5) (DS1, DS3, DS5).

Evaluation

The first goal of our experiment is to validate that our algorithm can scale up to deal with services composition of large service sets. Since there are no standard datasets for web service composition, we evaluate planner performance by measuring planning time on increasingly large randomly generated sets of services. We consider two kinds of services matching algorithm: greedy algorithm and candidate DS se-

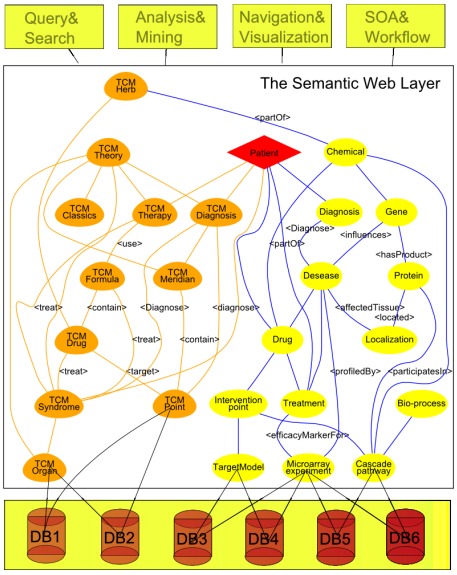

Figure 2: The ontology integrates data from the territories of TCM (Left) and Western Medicine (Right), and supports service composition.

Table 1: Example Data Provider Service

DS1: patients and their diagnosis disease information. **Contents**: [Patient, Diagnosis, Disease] **Input**: pName, pId, dName, **Output**: diagName, diagId, dName, dId, drgId
DS2: patients and their TCM diagnosisdisease information. **Contents**: Patient, TCMDiagnosis, disease **Input**: pName,pId, dName], **Output**:TCMdiagName, TCMdiagId, dName, dId, TCMdrgId, **Constraint**: pAge =18
DS3: Disease, Drug and Treatment information. **Contents:** [Disease, Drug, Treatment] **Input**: dName, dId, drgName, tName, **Output**: dName, dId, drgId , drgName, tName, tId, bpId
DS4: Disease, TCMDrug and TCMTreatment information. **Contents**: [Disease, TCMDrug, TCMTreatment] **Input**: dName,dId, TCMdrgName, TCMtName, **Output**: dName, dId, TCMdrgName, TCMdrgId,TCMtName, TCMtId, bpId
DS5: Gene and Protein information. **Contents**: [Gene, Protein] **Input**: gName, pName **Output**: gName,gId, pName , pId, cpId
DS6: Cascade Pathway and Bio-Process information **Contents**: [Cascade Pathway, Bio-Process] **Input**: cpId,cpName, bpName, bpId, **Output**: bpId, bpName, cpId , cpName, bpExplain

lecting algorithm. In these two case, we consider queries and services that have the same shape and size. Since there are no standard datasets for web service composition, we evaluate planner performance by measuring planning time on increasingly large randomly generated sets of services. All experiments are performed on a PC with a single Intel Celeron 2.4GHz CPU and 1024MB RAM, JRE 1.5.0.

Greedy Algorithm In our experiment, we consider the query that involves ten data services (DS1,DS2, ..., DS10). They can be viewed as a line of data services that can be joined one by one to achieve user's query goal. Each service on the chain can be selected from a service set and joined with the subsequent service, which is selected from the same service set accordingly. This scenario simulates the case when each service on the composition chain is directly selected from the whole service set and does not adopt any filter algorithm. The Fig. 3 shows the performance in the chain model against the increasing number of the service.

Pre-Filter Algorithm . For this scenario, we again consider the query that involves ten data services (DS1,DS2,..., DS10) composed one by one as like a chain. In this schema, each node on the chain has a candidate service set that is constructed from the whole service set by our candidate DS selecting algorithm. The Fig. 4 shows the performance of adopting our filter algorithm against the increasing number of the candidate data service set for each service node on the chain. The experiments show that our pre-filtering approach can makes service composition planning more practical by improving planner performance and scalability. We can see that even for total 500 services, the planner is able to produce the plan in less than 9 seconds, which is an acceptable performance. The experiments also illustrate that the services composition adopting our Pre-Filter Algorithm works better than the general greedy algorithm.

Related Work

Web service composition has been always gaining great attention since the technology was first introduced. Popular approach is to use workflow-based methods to describe, develop and manage service compositions. There are already a good body of projects and work adopting workflow technology for service composition (Esfandiari & Tosic 2005) (Martin *et al.* 2007). Many of them are based on the standard W3C web service language to support web service composition. Two major efforts are the BPEL4WS (Andrews & others 2003) and OWL-S (Martin *et al.* 2007) (Ankolekar 2002), which defines a standard for so-called semantic web services(McIlraith & Martin 2003).

Existing AI planning and reasoning web service composition works draw upon various AI planning techniques and different kinds of logics (Sirin & Parsia 2004). In general, all of these works are targeted towards workflow-oriented applications or AI-Planning-oriented applications, rather than data-intensive applications getting data from multiple web services via SQL-oriented queries, as addressed in this paper. Although many of them have utilized sematic technologies to address the issue of service interface mismatching, they do not have, as far as we are aware, specific planning models or techniques that designed specially for data-

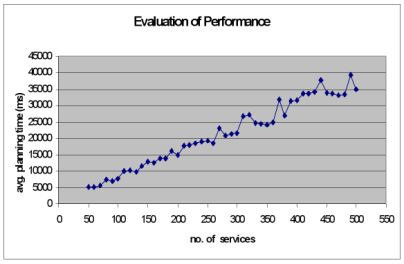

Figure 3: Greed Algorithm.

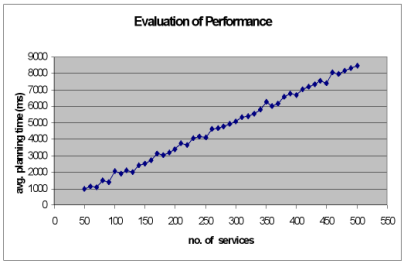

Figure 4: Pre-Filter Algorithm.

intensive web service that requires more expressive description and more fined control over its data content.The Grid community is devoting great attention toward the management of data. The most significant examples of such works are the OGSA-DAI(Antonioletti *et al.* 2005) and the OGSA-DQP(Alpdemir *et al.* 2004) projects. But only few of those projects(Comito & Talia 2006) actually establish semantic connections among heterogeneous data sources. Our services composition strategy has some similarity to mediators in data integration systems (Halevy 2001) (Rachel Pottinger 2001) (Chen *et al.* 2006). However, they do not consider web service-based data resources that have specific limitation over query interface.

Summary and Future Work

This paper introduces an ontology-based approach to publishing and composing data-intensive web service. In summary, the main contribution is three fold.

1. A data service model that extends the description capability of conventional semantic web service langauge such as OWL-S to address the requirement for exposing rich data semantics of data-intensive applications.

2. A evolvable approach to publishing data service that make the data service more adaptable to client quest.

3. A query-rewriting based approach to composing data services that take advantage of ontology language and efficiency of query-rewriting-based planning.

One remained issue is to deal with non-sparql request that is more common in real-life web service applications. Dealing with non-sparql request calls for more sophisticated approach for query planning, which requires more advance approaches to performance optimization.

Acknowledgement

This work is supported in part by National Key Technology R&D Program under grant No. 2006BAH02A01; National Program(NO.A1420060153); National Program(NO.51306030101).

References

Alpdemir, M. N.; Mukherjee, A.; Gounaris, A.; Paton, N. W.; Watson, P.; Fernandes, A. A. A.; and Fitzgerald, D. J. 2004. Ogsa-dqp: A service for distributed querying on the grid. In *EDBT*, 858–861.

Andrews, T., et al. 2003. *Business Process Execution Language for Web Services*. 2nd public draft release, Version 1.1.

Ankolekar, A. 2002. Daml-s: Web service description for the semantic web.

Antonioletti, M.; Atkinson, M.; Baxter, R.; Borley, A.; Hong, N. P. C.; Collins, B.; Hardman, N.; Hume, A. C.; Knox, A.; Jackson, M.; Krause, A.; Laws, S.; Magowan, J.; Paton, N. W.; Pearson, D.; Sugden, T.; Watson, P.; and Westhead, M. 2005. The design and implementation of grid database services in ogsa-dai: Research articles. *Concurr. Comput. : Pract. Exper.* 17(2-4):357–376.

Berardi, D.; Calvanese, D.; Giacomo, G. D.; Hull, R.; and Mecella, M. 2005. Automatic composition of transition-based semantic web services with messaging. In *VLDB2005*, 613–624. VLDB Endowment.

Chen, H.; Wu, Z.; Wang, H.; and Mao, Y. 2006. Rdf/rdfs-based relational database integration. *icde* 0:94.

Comito, C., and Talia, D. 2006. Grid data integration based on schema mapping. In K?gstr?m, B.; Elmroth, E.; Dongarra, J.; and Wasniewski, J., eds., *PARA*, volume 4699 of *Lecture Notes in Computer Science*, 319–328. Springer.

Esfandiari, B., and Tosic, V. 2005. Towards a web service composition management framework. In *ICWS2005*, 419–426. Washington, DC, USA: IEEE Computer Society.

Halevy, A. 2001. Answering queries using views: A survey. *Journal of Very Large Database* 10(4):53–67.

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *VLDB*, 251–262.

Martin, D.; Burstein, M.; Mcdermott, D.; Mcilraith, S.; Paolucci, M.; Sycara, K.; Mcguinness, D. L.; Sirin, E.; and Srinivasan, N. 2007. Bringing semantics to web services with owl-s. *World Wide Web* 10(3):243–277.

McIlraith, S. A., and Martin, D. L. 2003. Bringing semantics to web services. *IEEE Intelligent Systems* 18(1):90–93.

Pistore, M.; Traverso, P.; Bertoli, P.; and Marconi, A. 2005. Automated synthesis of composite bpel4ws web services. *icws* 0:293–301.

Rachel Pottinger, A. H. 2001. MiniCon: A scalable algorithm for answering queries using views. *VLDB Journal: Very Large Data Bases* 10(2–3):182–198.

Sirin, E., and Parsia, B. 2004. Planning for semantic web services. In *Planning for Semantic Web Services. In Semantic Web Services Workshop at 3rd International Semantic Web Conference, 2004.*