A View Integration Approach to Dynamic Composition of Web Services

Snehal Thakkar

University of Southern California/ Information Sciences Institute
4676 Admiralty Way,
Marina Del Rey, California 90292
thakkar@isi.edu

Abstract

Web services enable the user to integrate and manipulate data from distributed data sources without worrying about the underlying syntactical details. We describe extensions to the view integration approach to support dynamic integration of data from web services and support dynamic composition of web services from existing web services. In particular, we describe techniques to extend the "inverse rules" query reformulation algorithm to generate a universal integration plan to answer template user queries. To demonstrate the effectiveness of these techniques we describe a mediator-based system that dynamically composes various web services in response to a user query and provides an integrated web service that can handle a range of user requests.

1. Introduction

The emergence of web services standards has opened the doors for exciting new applications on the web that integrate and manipulate information from different web services and web sources. Several vendors have provided different tools to easily build and deploy web services. The XML-based standards for information interchange, such as WSDL, and web-based transport protocols such as SOAP, address syntactical issues involved in integrating information from different web services. The true potential of web services can only be achieved if web services are used to dynamically compose new web services that provide additional functionality.

In the context of dynamically composing new web services from existing web services, the existing web services can be viewed as data sources. In recent years various mediator systems, such as the Information Manifold [10], InfoMaster [5], InfoSleuth [3], and Ariadne [7], have been developed to provide a unified query interface to distributed data sources. At the same time the theoretical fundamentals of data integration were developed and are now well understood [6, 9]. The traditional mediator systems accept a specific user query and reformulate the query into a combination of source queries that can answer the user query.

This research focuses on developing an extensible information integration framework that can support accurate and efficient integration of a wide variety of data. The goal of the research is to produce an information integration framework that can (a) model web services as data sources, (b) dynamically compose new web services to answer template user queries by generating different query plans that integrate information from various data sources, (c) automatically model the dynamically composed web services as new data sources, and (d) provide extensible set of predicates to support accurate and efficient integration of heterogeneous data. The research on dynamically composing web services to answer template queries and automatically modelling newly generated web services as data sources is described in this paper. Several research projects that I am working on are related to the design of predicates that can be used by the mediator. The Proteus project describes work on designing new set of predicates such as, conflation, for geo-spatial data integration [8]. I am also working with other students on developing object consolidation predicate using the record linkage techniques described in [12].

In this paper, we describe an extension to the mediator based approach to support the dynamic composition of web services. In particular we propose an extension to the Inverse Rules query reformulation algorithm [4] that produces a generalized service composition in response to a user request. Instead of generating a plan limited to the specific user request, our system produces an integrated web service that can answer a much larger range of requests, in a sense it produces a universal integration plan [11].

The remainder of this paper is organized as follows. Section 2 describes an example scenario that will be used throughout the paper to explain the different concepts more clearly. Section 3 discusses relevant previous work in information integration. Section 4 describes a novel approach to dynamically compose web services from the existing web services. Finally, Section 5 discusses the contributions of this research and outlines the plans for future work.

2. Motivating Example

Consider a real estate domain scenario in which the user wants to find out the values of the properties that a given company owns in a given city. Assume that the available web services for this domain are:

CitytoCounty($city^b$, $county^f$)
LAProperty($address^b$, $value^f$, $county^f$)
SFProperty($address^b$, $value^f$, $county^f$)
OrangeProperty($address^b$, $value^f$, $county^f$)
YellowPages($name^b$, $city^b$, $address^f$, $phone^f$)

Each web service is described as a predicate with binding patterns, as it is common in describing web sources. The superscript 'b' indicates that the attribute is a required input of the service. The superscript 'f' indicates that there is no restriction on the attribute. The CitytoCounty web service requires a city as an input and outputs the county in which the city is located. The LAProperty service accepts an address in Los Angeles County and provides the value of the property located at that address. Similarly, the SFProperty and the OrangeProperty web services provide property values for addresses in San Francisco county and Orange county, respectively. The YellowPages web service accepts a business name and a city and provides the addresses for all the locations of the given business in the given city.

The user can send different requests to the mediator system. As a running example, we will use the query "find the property values for all 'Burger King' locations in 'Los Angeles'". When the mediator system receives such a query from the user, it generates a query plan that invokes the relevant web services, combines their outputs, and composes the answer. The next section describes an information integration system that we have developed in past to answer the user queries similar to the query above.

3. Previous Work

In recent past, there has been a lot of work on information integration frameworks. We combined a popular query reformulation algorithm called the Inverse Rules [4], with a streaming, dataflow style execution engine termed Theseus [2] to generate a new mediator system. The key advantages of the new mediator system are the ability to provide maximally complete answers to the user queries, support for recursion and binding patterns, and a streaming dataflow style execution system. This section briefly describes this mediator system. Section 3.1 describes the Inverse Rules algorithm to reformulate user queries into a datalog program representing a set of queries on various sources. Section 3.2 describes the execution of the datalog programs generated by the Inverse Rules algorithm.

Extended version of this paper was submitted to the ICAPS 2003 workshop on Planning for Web Services.

3.1 Inverse Rules

The Inverse Rules algorithm was utilized by the InfoMaster information integration system [4]. The key advantages of the Inverse Rules algorithm are the ability to handle recursive user queries, the ability to handle functional dependencies, and the ability to handle access pattern limitations. The mediator systems that use the Inverse Rules algorithm utilize the Local-as-view model [9], i.e. they define the source relations as a view over the global relations. For this paper, the mediator system has access to the data sources described in Section 2 and the mediator has the following global relations in its domain model.

FindCounty($city^b$, $county^f$)
Propertytax($address^b$, $value^f$)
FindLocations($name^b$, $city^b$, $address^f$, $phone^f$)

The mediator system describes the data sources as views over the global relations as follows:

```
CitytoCounty(city^b, county^f):- FindCounty(city, county)
LAProperty(address^b, value^f, county^f):-
        PropertyTax(address, value, 'LA')
SFProperty(address^b, value^f, county^f):-
        PropertyTax(address, value, 'SF')
OrangeProperty(address^b, value^f, county^b) :-
        PropertyTax(address, value, 'Orange')
YellowPages(name^b, city^b, address^f, phone^f) :-
        FindLocations(name, city, address, phone)
```

The first step of the Inverse Rules algorithm is to invert the view definitions to obtain definitions for all global relations as views over the source relations. In order to generate the inverse view definitions, the Inverse Rules algorithm analyzes all local as view definitions. For every view definition, $V(X)$:- $S_1(X_1),\ldots,S_n(X_n)$, where X, X_i refer to set of attributes in the corresponding view or relation, the Inverse Rules algorithm generates n inverse rules, for $i = 1,..,n$, $S_i(X'_i)$:- $V(X)$, where if $X_i \varepsilon X$, X'_i is the same as X_i else X_i is replaced by a function symbol [4].

When a user sends a query to the system, the Inverse Rules algorithm unions the inverse rules with the user query to produce a set of datalog rules to answer the user query. The datalog rules and the schema information are passed to the query execution engine to execute the query plan. In our example, the system generates the following set of datalog rules & queries:

Rules:

```
PropertyTax(address, value, county) :-
        LAProperty(address, value, 'LA')
PropertyTax(address, value, county) :-
        SFProperty(address, value, 'SF')
PropertyTax(address, value, county) :-
```

```
        OrangeProperty(address, value, 'Orange')
FindLocations(name, city, address, phone) :-
        YellowPages(name, city, address, phone)
Query1(name, city, address, value) :-
        FindLocations(name, address, phone)^
        PropertyTax(address, value, county)
```

Queries:
Query1('Burger King', 'Los Angeles', address, value)

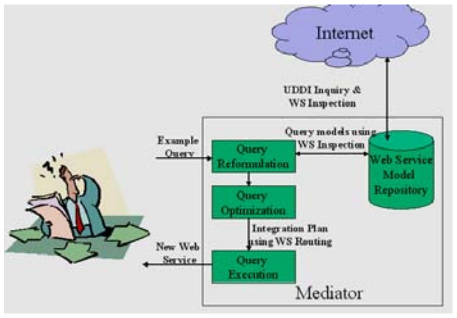
The first three rules describe that the PropertyTax global relation can be obtained by performing union on the LAProperty, SFProperty, and OrangeProperty data sources. The next rule describes that the FindLocations virtual source which can be obtained by querying the YellowPages data source. Finally, the user query specifies that first the mediator system should query the FindLocations virtual source to find all 'Burger King' locations in the city of 'Los Angeles'. Next, the mediator system should query the PropertyTax virtual source to find the property values of the locations retrieved from the FindLocations virtual source.

The mediator system must execute the query plan to answer the user query. The next section describes how the generated datalog program is executed to generate the results of the user query.

3.2 Query Execution

Any datalog execution engine can execute the datalog program generated by the Inverse Rules algorithm. However, the datalog execution engines do not have ability to execute multiple operations in parallel and cannot stream data between the operations. We have developed a technique [13] to map datalog programs to integration plans that can be executed by a streaming, highly parallel execution engine called Theseus [2].

The Theseus has a wide variety of operators to perform various data management tasks, access different data sources, and communication operators such as e-mail. Among the streaming, highly parallel execution engines, Theseus is unique in its support for recursion. Theseus can execute the integration plans more efficiently compared to the traditional datalog execution engines. The key advantage of utilizing the Theseus execution engine over traditional datalog execution engine is the fact, that the datalog execution engines cannot perform several operations in parallel or stream data between operations. For example, Theseus can query all the property tax web services in parallel, while the datalog execution engines would query property tax web sites sequentially. Next, we describe how this mediator system can be extended to support web services.


Internet
UDDI Inquiry & WS Inspection
Query models using WS Inspection
Example Query
Query Reformulation
Web Service Model Repository
Query Optimization
Integration Plan using WS Routing
New Web Service
Query Execution
Mediator
Figure 1 Mediator As a Composer of Web Services

4. Mediator as a Composer of Web Services

This section describes an extension to the mediator system described in Section 3. The mediator system described in Section 3 is extended in two ways. First, the mediator system is modified to answer not a specific user query, but a template user query. For example, instead of answering the user query "find property values for all 'Burger King' location in the city of 'Los Angeles'", the mediator should generate a plan to answer a template query "find property values for all location of the given business in a given city". The template query is obtained by just making the constants in the user query input variables.

Second, the mediator is modified to return the URL of the dynamically composed web service that can answer the template user query instead of returning the query results. Figure 1 shows the architecture of the extended mediator system. The key difference between the new mediator system compared to the traditional mediator system, is that the mediator system dynamically composes new web service to answer the template user query.

Tuple level filtering technique described in [14] is used to modify the mediator to generate a universal plan [11] that can answer the template query instead of the specific user query. Section 4.1 describes how the Inverse Rules algorithm is modified to generate an integration plan to answer template query. Section 4.2 describes how the generated datalog program is mapped to Theseus integration plan.

4.1 Modified Inverse Rules Algorithm

The modified Inverse Rules algorithm differs from the original algorithm in two ways. First, the constants in the query are treated as variables. In our example, the query "find property values for all 'Burger King' locations in the city of 'Los Angeles'" has two constants 'Burger King' and 'Los Angeles'. Both constants are replaced with name and city input parameters. One direct impact of this change is the fact that the modified Inverse Rules algorithm now generates a universal integration plan [11] that obtains the maximally complete answers to the template user query given the set of sources.

Second, the constraints from the source definitions are used to filter the inputs to the sources. For all the source definitions, attributes involved in the constant constraint are changed to binding attributes and a filter is added to make sure that the attribute satisfies the constraint. For example, the model of the LAProperty tax web service has a constraint that the web service can only find property values for the properties located in 'Los Angeles' county. Therefore, before querying a property value for any address from LAProperty tax web service, the algorithm needs to verify that the address is in Los Angeles County. The algorithm changes the county attribute to a bound variable and adds a filter to ensure that the county attribute is 'Los Angeles'. One of the key problems with the universal integration plan is the fact that generated plan may send a large number of queries to the available web services. The second modification allows us to make sure that the generated plan does not send a large number of queries to any web services with incorrect parameter values. This technique is similar to the technique described in [1] to query some external data sources to reduce the number of queries to a given web service.

The modified data model and modified queries are then passed to the Inverse Rules algorithm to generate a datalog program that can answer the modified query. We will further clarify this modification by examining the datalog plan that is generated by the modified Inverse Rules algorithm.

Rules:
FindCounty(city, county):- CitytoCounty(city, county)
PropertyTax(address, value, **'LA'**):-
    LAProperty(address, value, 'LA')
PropertyTax(address, value, **'SF'**):-
    SFProperty(address, value, 'SF')
PropertyTax(address, value, **'Orange'**):-
    OrangeProperty(address, value, 'Orange')
FindLocations(name, city, address, phone) :-
    YellowPages(name, city, address, phone)
Query1(name, city, address, value) :-
    FindLocations(name, city, address, phone)^
    FindCounty(city, county)^
    PropertyTax(address, value, county)

Queries:
Query1(**name**^{b}**, city**^{b}, address, value)

Listing 1 New Datalog Rules

The modifications to the original datalog plan form Section 3 are shown in bold. In the modified datalog program the query Query1 has no constants instead; the name and the city parameters are bound, i.e. the values for these parameters must be specified by the user. This change is due to the first modification of the Inverse Rules algorithm that replaces the constants in the user query with input parameters.

The Inverse Rules algorithm also ensures that the binding patterns for all sources are satisfied. In the modified data model the LAProperty, the SFProperty, and the OrangeProperty sources require an address attribute and a county attribute. The Inverse Rules algorithm adds a query to FindLocations source to obtain the county information before querying the PropertyTax virtual source. Finally, filters are added to ensure that for the tuples passed to the LAProperty source have value 'Los Angeles' for the county attribute, the tuples passed to the SFProperty source have value 'San Francisco' for the county attribute, and the tuples passed to the OrangeProperty source have value 'Orange' for the county attribute. This step ensures that the generated plan does not send irrelevant queries to various data sources.

4.2 Query Execution

The mediator system maps the generated datalog program to an integration plan that can be executed by the Theseus execution engine. The datalog program generated in Section 4.1 is translated to a Theseus plan shown in Figure 2. The first operations in the Theseus plan are to query YellowPages and CitytoCounty web services using the input parameters. The Theseus execution engine queries both web services in parallel and streams the data between the two services to the join operator that joins the information from both web services. Based on the county attribute of the joined data, the joined data is routed to LAProperty web service, SFProperty web service, or OrangeProperty web service. One major difference between this plan and the plan shown in Section 3.2 is the fact that only one of the property tax web services is queried for a given specific query. Moreover, which property tax service to query for a given specific query is based on the information queried from the CitytoCounty web service. This idea is very similar to the idea of interleaving plan execution and plan generation. However, the key difference here is the fact that the plan is generated before the execution begins and the conditions to decide which property tax web service to query is encoded in the plan based on the model of difference property tax web services.

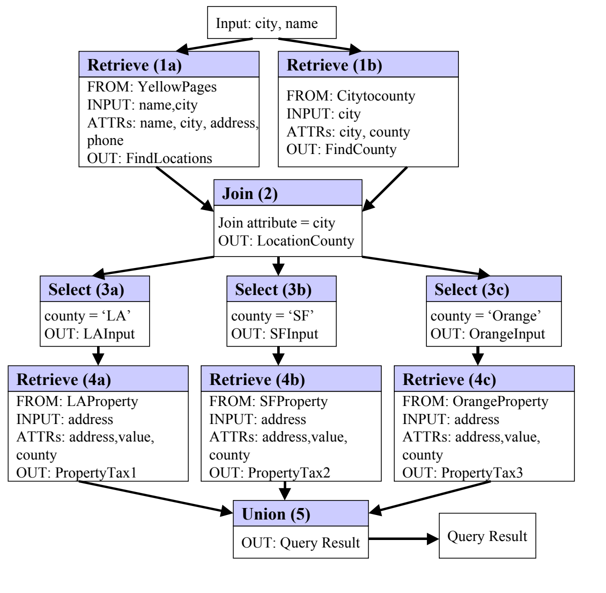


Figure 2 Modified Theseus Plan

The mediator system utilizes the template integration plan to generate a new web service that can answer the template user query. For our example, the mediator generates a new web service that accepts city and a name of business as input and returns the property values of the all the locations of the business in a given city. The mediator returns URL of the new web service to the user.

5. Conclusions & Future Work

This paper describes techniques to extend the Inverse Rules [4] algorithm to generate a universal integration plan to answer the template user query. The modified Inverse Rules algorithm was used to develop a mediator web service that dynamically integrates data from various web services and dynamically composes new web services from the existing web services. The mediator web service accepts user queries and returns a URL of dynamically composed web service that can answer not only the specific user query, but also the all user queries that fit the template query.

In future, we plan to extend our mediator framework to automatically model the newly generated web service as a data source in the mediator's domain model. This can be done very easily as the template query can be used to describe the new web service. We are also planning to extend the operations supported by the mediator to facilitate intelligent integration of data from different web services. For example, one of the key issues when integrating data from various web services is to consolidate information extracted from various data sources. We plan to incorporate object consolidation techniques from [12] as an intelligent join operator in the mediator. The object consolidation techniques allow "soft-matching" the records extracted from various web services.

Acknowledgements

This work has been advised by Dr.Craig A. Knoblock, Dr.Cyrus Shahabi, and Dr. Jose Luis Ambite. This research is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory under contract/agreement numbers F30602-01-C-0197 and F30602-00-1-0504, in part by the Air Force Office of Scientific Research under grant numbers F49620-01-1-0053 and F49620-02-1-0270, in part by the United States Air Force under contract number F49620-01-C-0042,in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, under cooperative agreement number EEC-9529152, and in part by a gift from the Microsoft Corporation.

References

[1] N. Ashish, C.A. Knoblock, and A. Levy. *Information Gathering Plans with Sensing Actions. European Conference on Planning, ECP-97*. 1997. Toulouse, France.

[2] G. Barish, D. DiPasquo, C.A. Knoblock, and S. Minton. *A dataflow approach to agent-based information management. Proceedings of the 2000 International Conference of on Artificial Intelligence*. 2000. Las Vegas, NV.

[3] R.J. Bayardo Jr., W. Bohrer, R. Brice, A. Cichocki, J. Flower, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. *Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In Proceedings of ACM SIGMOD-97*. 1997.

[4] O.M. Duschka, *Query Planning and Optimization in Information Integration*, in *Computer Science*. 1997, Stanford University. p. 92.

[5] M.R. Genesereth, A.M. Keller, and O.M. Duschka. *InfoMaster: An information integration system. In Proceedings of ACM SIGMOD-97*. 1997.

[6] A.Y. Halevy, *Answering queries using views: A survey.* The VLDB Journal, 2001. **10**(4): p. 270--294.

[7] C. Knoblock, S. Minton, J.L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada, *The ARIADNE Approach to Web-Based Information Integration.* International Journal on Intelligent Cooperative Information Systems (IJCIS), 2001. **10**(1-2): p. 145-169.

[8] C. Knoblock, C. Shahabi, J.L. Ambite, S. Thakkar, and C.-c. Chen, *Composing Web Sources with .NET*. 2002, University of Southern California.

[9] A. Levy, *Logic-Based Techniques in Data Integration*, in *Logic Based Artificial Intelligence*, J. Minker, Editor. 2000, Kluwer Publishers.

[10] A.Y. Levy, A. Rajaraman, and J.J. Ordille, *Query-Answering Algorithms for Information Agents*, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. 1996: Portland, OR. p. 40--47.

[11] M. Shoppers. *Universal plans for reactive robots in unpredictable environments. Proceedings of the InternationalConference on Artificial Intelligence, IJCAI-87*. 1987.

[12] S. Tejada, C.A. Knoblock, and S. Minton, *Learning Object Identification Rules for Information Integration.* Information Systems, 2001. **26**(8).

[13] S. Thakkar and C.A. Knoblock. *Efficient Execution of Recursive Integration Plans. To Appear In Proceeding of 2003 IJCAI Workshop on Information Integration on the Web*. 2003. Acapulco, Mexico.

[14] S. Thakkar, C.A. Knoblock, J.-L. Ambite, and C. Shahabi. *Dynamically Composing Web Services from On-line Sources. In Proceeding of 2002 AAAI Workshop on Intelligent Service Integration*. 2002. Edmonton, Alberta, Canada.