A View Integration Approach to Dynamic Composition of Web Services

Snehal Thakkar, Craig A. Knoblock, and José Luis Ambite

University of Southern California/ Information Sciences Institute
4676 Admiralty Way,
Marina Del Rey, California 90292
{thakkar, knoblock, ambite}@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 user queries. To demonstrate the effectiveness of these techniques we describe a mediator-based system that dynamically integrates various web services in response to a user query and provides a integrated web service that can handle a range of user queries.

1. Introduction

The introduction of web services to the Internet has opened the doors for new exciting applications on the web that integrate 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 and web-based exchange 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 more functionality compared to existing web services.

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 [1] InfoMaster [2], InfoSleuth [3], and Ariadne [4] have been used to provide a unified query interface to various data sources. At the same time the theoretical fundamentals of data integration have been investigated and are now well understood [5, 6]. The traditional mediator systems accept a specific user query and reformulate this query into a combination of source queries that can answer such query.

In this paper we describe an extension to the mediator approach to support the dynamic composition of web services. In particular, we propose an extension to the Inverse Rules query reformulation algorithm [7] 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 range of requests. In a sense, our system produces a universal integration plan [8].

The remainder of this paper is organized as follows. Section 2 describes an example scenario that will be used throughout the paper to convey different concepts more clearly. Section 3 discusses relevant previous work in information integration. Section 4 describes a naïve way to extend the information integration framework described in Section 3 to handle web services as data sources. Section 5 describes a novel approach to dynamically compose web services from existing web services, by extending the information integration framework described in Section 3. Section 6 concludes the paper by discussing the contribution of the paper and future work.

2. Motivating Example

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

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

We have described each web service 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 the given address. Similarly, the NYProperty web service accepts an address and city in the state of New York and provides the property value and county information for such address. Finally, 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 queries to the mediator system. As a running example, we will use the query "find the property values for all 'Burger King' locations in the city of 'Torrance' in the state of 'CA'". 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 the previous work to answer the user queries similar to the query above.

3. Previous Work

Recently, we have combined a state-of-the-art query reformulation algorithm, the Inverse Rules [7] algorithm, with a streaming, dataflow-style execution engine, Theseus [9], to generate a new mediator system [10]. 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. In this section, we briefly describe this mediator system. Section 3.1 describes the Inverse Rules Algorithm that reformulates a user query into a datalog program representing a set of queries on various sources. Section 3.2 describes how such datalog programs are mapped into Theseus plans and executed.

3.1 Inverse Rules Algorithm

The Inverse Rules algorithm was utilized by the InfoMaster information integration system [2, 7]. The key advantages of the Inverse Rules algorithm are the ability to handle recursive user queries, functional dependencies, and access pattern limitations. The mediator systems that use the Inverse Rules algorithm utilize the Local-as-view model [11], i.e. they define the source relations as a view over the global relations. We will assume that the mediator system has access to the data sources described in Section 2 and the mediator has the following domain predicates:

Location(locid, address, city, county, state)
Value(locid, value)
Business(name, locid, phone)

The mediator system describes the data sources as views over the domain predicates as follows:

R1: LAProperty(address, city, value):-
Location(locid, address, city, county, state) ^ Value(locid, value) ^
county = 'LA'^
state = 'CA'

R2: NYProperty(address, city, county, value):-
Location(locid, address, city, county, state) ^ Value(locid, value) ^
state = 'NY'

R3: YellowPages(name, address, city, state, phone) :-
Business(name, locid, phone)^
Location(locid, address, city, county, state)

R4: CitytoCounty(city, state, county):-
Locations(locid, address, city, county, state)

The first step of the Inverse Rules 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 definition, the Inverse Rules algorithm analyzes all view definitions. For every view definition, $V(X)$:- $S_1(X_1),\ldots,S_n(X_n)$, where X and 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 $X_i \in X$, X'_i is the same as X_i else X_i is replaced by a function symbol [7]. For the given example, the Inverse Rules algorithm analyzes the view definitions and generates the following rules.

IR1: Location(f_{1locid}(a, ci, v), a , ci, 'LA', 'CA') :-
LAProperty(a, ci, v)^
dom1(a)

IR2: Value(f_{1val}(a, ci, v), v) :-
LAProperty(a, ci, v)^
dom1(a)

IR3: Location(f_{2locid}(a, ci, co, v), a , ci, co, 'NY') :-
NYProperty(a, ci, co, v)^
dom1(a)

IR4: Value(f_{2val}(a, ci, co, v), v) :-
NYProperty(a, ci, co, v)^
dom1(a)

IR5: Business(n, f_{1bus}(n, a, ci, s, p), p) :-
YellowPages(n, ,a, ci, s, p)^
dom2(n)^
dom3(ci)^
dom4(s)

IR6: Location(f_{3locid}(n, a, ci, s, p), a , ci, f_{4co}(n, a, ci, s, p), s) :-
YellowPages(n, a, ci, s, p)^
dom2(n)^
dom3(ci)^
dom4(s)

IR7: Location(f_{5locid}(ci, s, co), f_{6locid}(ci, s, co) , ci, co, s) :-
CitytoCounty(ci, s, co)^
dom3(ci)^
dom4(s)

The rules IR1 and IR2 are the result of inverting the rule R1. The location id attribute is replaced with a function of different values provided by the sources. The dom_i predicates are inserted to handle the binding pattern

limitations of the sources. Recall that we model the input and outputs of web services as binding patterns. For simplicity we have omitted additional rules that fully define the dom predicates. Similarly, the rules IR3 and IR4 are the result of inverting rule R2. IR5 and IR6 are obtained by inverting the rule R3. Finally, IR7 is obtained by inverting the definition of the CitytoCounty source given by rule R4. More details on eliminating functions, handling binding patterns, and inverting view definitions are described in [7]. When a user sends a query to the system, the Inverse Rules algorithm unions the inverse rules with the user query to produce a datalog program that can 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 datalog program[1] to answer the user query:

Rules:

IR1: Location(f_{1locid}(a, ci, v), a , ci, 'LA', 'CA') :-
LAProperty(a, ci, v)^
dom1(a)

IR2: Value(f_{1val}(a, ci, v), v) :-
LAProperty(a, ci, v)^
dom1(a)

IR5: Business(n, f_{1bus}(n, a, ci, s, p), p) :-
YellowPages(n, ,a, ci, s, p)^
dom2(n)^
dom3(ci)^
om4(s)

IR6: Location(f_{3locid}(n, a, ci, s, p), a , ci, f_{4co}(n, a, ci, s, p), s) :-
YellowPages(n, a, ci, s, p)^
dom2(n)^
dom3(ci)^
dom4(s)

QR: Query1(n, ci, s, a, v) :-
Business(n, l, p)^
location(l, a, ci, co, s)^
value(l, v)^
n = 'Burger King'^
ci = 'Torrance'^
s = 'CA'

Intuitively, the user query can be answered by just accessing a YellowPages web service and LAProperty tax web services. The datalog program generated by the Inverse Rules Algorithm confirms this by utilizing the rules IR1, IR2, and IR5 in the resulting datalog program. The rule IR7 is not used in the program as the CitytoCounty source is not directly relevant to answer the user query. Also, the rules IR3 and IR4 are not used as the NYProperty tax web service is not relevant as well. The rule QR is query rule that has been added to the datalog program to answer the user query. Next, we describe how the datalog program generated by the Inverse Rules algorithm is executed.

[1] The full Inverse Rules Algorithm eliminates the function symbols to generate a program that is stricly datalog. Since the resulting rule are more complicated, we present the rules with function symbols for clarity.

3.2 Query Execution

Any datalog execution engine can execute the datalog program generated by the Inverse Rules algorithm. However, 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 [10] to map datalog programs to integration plans that can be executed by a streaming, highly parallel execution engine called Theseus [9].

The Theseus execution engine has a wide variety of operators to perform various data management tasks, access 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. Figure 1 shows the graphical representation of the Theseus plan corresponding to the example datalog program in Section 3.1

All the boxes in Figure 1 refer to different Theseus operations that have to be performed to answer the user query. Each operation in Theseus accepts one or more relations as argument and produces zero or more relations as output. A relation in Thesues is similar to tables in relational databases. A relation in Theseus can have zero of more tuples and one or more attributes.

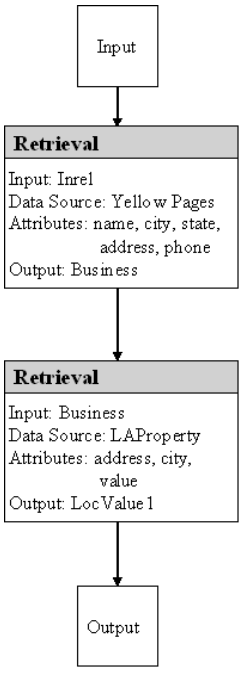

Figure 1 Example Theseus Plan

In our example, the Theseus plan receives a relation containing the business name, city, and state attributes as an input, which is termed inrel in the Figure 1. The first step of the plan is to use the input relation to retrieve the business locations from the YellowPages web service. We can perform this retrieval operation first as the binding patterns for the YellowPages web service can be satisfied using just the input relation. This step corresponds to the rules IR5 in the datalog program. The rule IR6 does not produce any tuples as the function f_{4co}(n, a, ci, s, p) can not be evaluated. The output relation from the YellowPages web service contains a set of locations for each business in the city of 'Torrance'. The locations obtained from the YellowPages web service are used to query property values for the locations from the LAProperty tax web service. The retrieval operation that access the data from the LAProperty tax web service is the result of translating the datalog rules IR1 and IR2 to Theseus plans. More details on translating datalog programs to Theseus plans can be found in [10].

The key advantage of utilizing the Theseus execution engine over traditional datalog execution engine is the fact, that Theseus can perform several operations in parallel and stream data between operations. For example, all the property tax web services are queried in parallel. Next, we describe how this mediator system can be modified to support web services.

4. Mediator as a Web Service

A naïve way to extend the mediator system describe in Section 3 to support web service is to view the mediator as a web service that can accept a user query and return the results of the user query by integrating and manipulating data from various web services.

As shown in Figure 2, the user can send query to the mediator web service. The mediator web service utilizes the Inverse Rules Algorithm to reformulate the user query into a datalog program representing a set of source queries. Next, the mediator maps the datalog program to an integration plan, which can be executed in streaming, highly parallel manner by the Theseus execution engine. The query results are returned to the user in form of a XML document via SOAP. Thus, the mediator is a web service that can accept a query and depending on the query provide different query results.

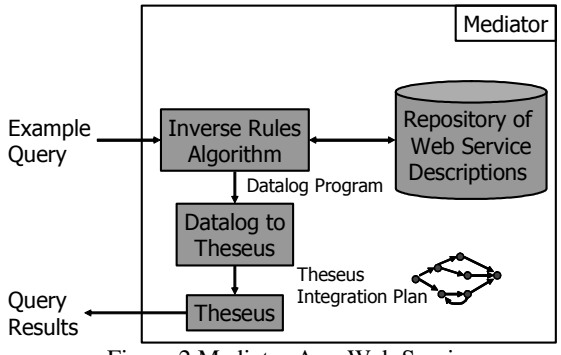

Figure 2 Mediator As a Web Service

While the approach to turn the mediator into a web service that accepts user queries and returns query results works fine, there are two issues. First, the mediator must regenerate the query plan for every user query. If the user queries may be same queries with the different constant parameters, then the mediator may be able to reuse some of the plans. For example, in our example system if several use queries are of the form find property values of the given business in the given city, the mediator may be able to reuse the plans it has generated. Second, the mediator web service does not fit well in the web service model. In the web service model each web service provides syntactical description of the web service through Web Service Description Language (WSDL). As a part of the WSDL description, the web service must describe the arguments accepted by the web service as input and the resulting output of the web service. The mediator web service accepts a user query and the structure of the resulting output can be different depending on the user query. Therefore, the arguments accepted by the mediator web service and the output of the mediator web service can not be accurately described to fit the web services model. To address these two issues, we describe a different approach in next section.

5. Mediator as a Web Service Generator

Instead of encapsulating the mediator as a web service, we can use the mediator as a generator of web services. In fact, for each user query the mediator generates a composition of web services that answers the query, so the system could save this integration plan and provide it as a new web service. However, such approach would produce overly specific web services. In this section we describe how to use a mediator to generate web services that are reusable, efficient, and well typed.

We utilize two techniques to enhance the mediator system described in Section 3 to fit the web services model. First, we change the mediator to generalize the user queries, so that it produces reusable web services. Second, we adapt a tuple-level filtering technique, originally introduced in [12], to reduce the number requests to each web service and produce a more efficient composite web service.

Our mediator system accepts a user query and generates a new web service that can answer not just the particular user query, but also a class of queries similar to the user query. In order to do so, the mediator generalizes the user query before producing the composite service. The user query can typically be generalized in many different ways. For example, the query in our example can be generalized over the city, the state, the business name, or any combination of the three parameters. In this paper we only consider the complete generalization of the query, i.e. generalize the query across all the parameters. The resulting web service provides answer not just for a particular set of values of the parameters for the query, but all valid sets of values of the parameters of the query. Therefore, in our example the mediator will generate a web service that can answer the following query: "find property value of all properties of a given business in the given city and

state". This is similar to the notion of generating the universal plan [8] to answer the queries.

One of the problems with generating a universal plan to answer the user queries is that the plan may send a lot of queries to different data sources. In our example, the plan may send a lot of queries to different property tax services. We solve this problem by using a tuple-level filtering technique [12]. The idea behind the tuple-level filtering technique is to ensure that each data source is only queried for the information present in that data source. The mediator introduces a filter operation before querying any data source to ensure that the input arguments to the data sources are acceptable to the data source based on the description of the source.

In order to fit with the web services model, the mediator web service must be able to clearly describe the input arguments to the web service and the outputs of the web service. The mediator system described in this section accepts a user query and returns a URL of a new dynamically composed web service that can answer a class of user queries similar to the user query. Furthermore, both the mediator web service and all dynamically composed web services can be described clearly as they all of them accept certain input arguments and return output with the same structure for all requests.

Figure 3 shows the architecture of the extended mediator system. The key difference between the new mediator system compared to the mediator system in Section 4, is the fact that the mediator system dynamically composes a new web service to answer a class of user queries.

Section 5.1 describes how the Inverse Rules algorithm is modified to generate an integration plan to answer template queries. Section 5.2 describes how the generated datalog program is mapped to Theseus integration plan.

5.1 Modified Inverse Rules Algorithm

The modified Inverse Rules algorithm differs from the original algorithm in two ways. First, constants in the query are treated as the variables. In our example, the query "find property values for all 'Burger King' locations in the city of 'Torrance' in the state of 'CA'", has three constants 'Burger King', 'Torrance' and 'CA'. Both constants are replaced with name and city input parameters. One direct impact of this change is the fact

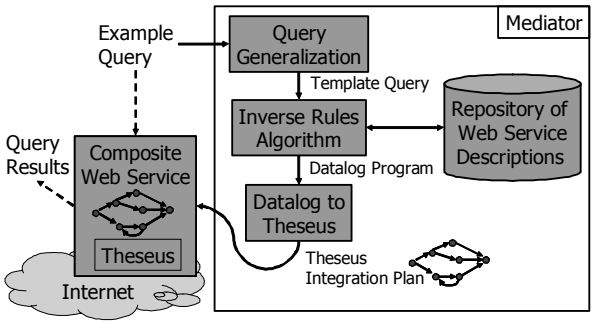

Figure 3 Mediator As a Web Service Generator

that the modified Inverse Rules algorithm now generates a universal integration plan [8] that obtains the maximally complete answers to the template user query given the set of sources.

As the value of the input parameters is not known in advance the resulting datalog program queries both LAProperty tax web service and the NYProperty tax web service for property values. Our second modification ensures that both web services are only queried with input arguments that are valid for them, i.e. the LAProperty tax web service is queried with only the locations in 'Los Angeles' county and the NYProperty tax web service is queried for the addresses in the state of 'New York'. 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 variable 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 [13] 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 using the modified data model.

Another key research issue here is how to handle scenarios when the modified Inverse Rules algorithm fails to answer the user query because the value for the bound variable can not be generated by any source. For example, if in our example, we did not have the CitytoCounty source, the modified Inverse Rules algorithm would not be able to answer the user query. We are looking at extending the plan generalization techniques described in [14, 15] to support web services framework. Next, the modified datalog program is passed to the query execution engine.

5.2 Query Execution

The mediator system maps the generated datalog program to an integration plan that can be executed by Theseus execution engine. The datalog program generated in Section 4.1 is translated to a Theseus plan shown in Figure 4. The first operations in the Theseus plan are to query YellowPages and CitytoCounty web services using the input parameters. Note that the binding constraints given by the dom predicates are

satisfied for both sources as the input arguments provide a city, a state, and a business name. 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. In parallel, Theseus selects the tuples with the state = 'NY' from the results of the YellowPages web service. The selected records are then used to query the NYProperty tax web service.

Next, tuples with the county = 'LA' are selected from the results of the join operation. The selected tuples are used to query the LAProperty web service. The filters specified by the select operations in Theseus plan ensure that the web services are not queried with invalid input arguments. Finally, the union operator is used to combine the results of both property tax web services. 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.

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

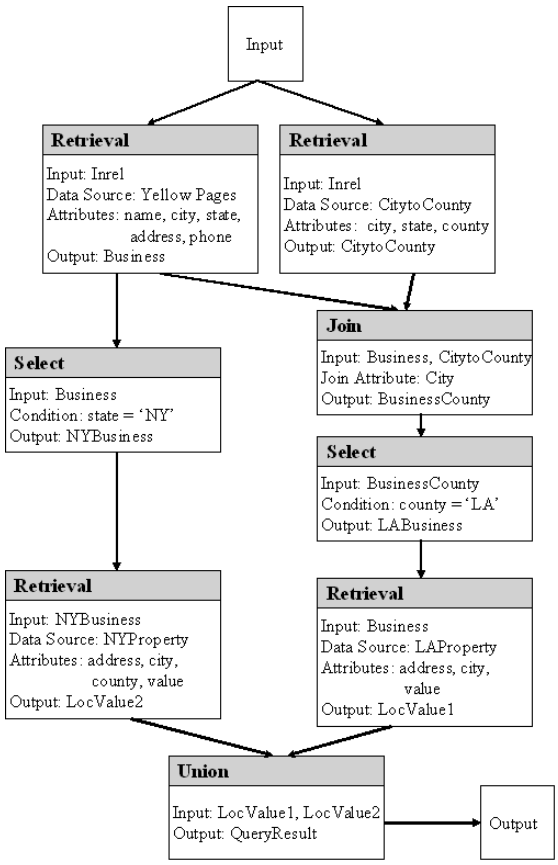

Figure 4 Modified Theseus Plan

6. Discussion & Future Work

In this paper, we described techniques to extend the Inverse Rules [7] Algorithm to generate a composite web service that can answer a class of user queries, similar to universal integration plans [8]. We described a mediator web service that utilizes the extended Inverse Rules Algorithm to dynamically integrate data from various web services and to dynamically compose 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 same template query.

In addition to working on strategies to efficiently determine how to determine which constraints should be used to generalize the query, 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 [16] as an intelligent join operator in the mediator. The object consolidation techniques allow "soft-matching" the records extracted from various web services.

References

1. Levy, A.Y., 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.
2. Genesereth, M.R., A.M. Keller, and O.M. Duschka. *InfoMaster: An information integration system*. in *In Proceedings of ACM SIGMOD-97*. 1997.
3. Bayardo Jr., R.J., et al. *Infosleuth: Agent-based semantic integration of information in open and dynamic environments*. in *In Proceedings of ACM SIGMOD-97*. 1997.
4. Knoblock, C., et al., *The ARIADNE Approach to Web-Based Information Integration.* International Journal on Intelligent Cooperative Information Systems (IJCIS), 2001. **10**(1-2): p. 145-169.

5. Levy, A.Y., *Logic-Based Techniques in Data Integration*, in *Logic-Based Artificial Intelligence*, J. Minker, Editor. 2000, Kluwer Publishers.
6. Levy, A.Y., et al., *Answering Queries Using Views*, in *Proceedings of the 14th ACM Symposium on Principles of Database Systems*. 1995: San Jose, California. p. 95--104.
7. Duschka, O.M., *Query Planning and Optimization in Information Integration*, in *Computer Science*. 1997, Stanford University. p. 92.
8. Schoppers, M. *Universal plans for reactive robots in unpredictable environments*. in *Proceedings of the InternationalConference on Artificial Intelligence, IJCAI-87*. 1987.
9. Barish, G., et al. *A dataflow approach to agent-based information management*. in *Proceedings of the 2000 International Conference of on Artificial Intelligence*. 2000. Las Vegas, NV.
10. Thakkar, S. and C.A. Knoblock. *Efficient Execution of Recursive Integration Plans*. in *To Appear In Proceeding of 2003 IJCAI Workshop on Information Integration on the Web*. 2003. Acapulco, Mexico.
11. Levy, A., *Logic-Based Techniques in Data Integration*, in *Logic Based Artificial Intelligence*, J. Minker, Editor. 2000, Kluwer Publishers.
12. Thakkar, S., et al. *Dynamically Composing Web Services from On-line Sources*. in *In Proceeding of 2002 AAAI Workshop on Intelligent Service Integration*. 2002. Edmonton, Alberta, Canada.
13. Ashish, N., C.A. Knoblock, and A. Levy. *Information Gathering Plans with Sensing Actions*. in *European Conference on Planning, ECP-97*. 1997. Toulouse, France.
14. Kambhampati, S. and S. Kedar, *A Unified framework for Explanation Based Generalization of Partially ordered and partially instantiated plans.* Artificial Intelligence, 1994. **67**(1).
15. Kambhampati, S. *Formalizing a spectrum of Plan Generalization based on Modal Truth Criteria*. in *Canadian Conference on Artificial Intelligence*. 1994.
16. Tejada, S., C.A. Knoblock, and S. Minton, *Learning Object Identification Rules for Information Integration.* Information Systems, 2001. **26**(8).