Towards a Data Warehouse Fed with Web Services

John Samuel

LIMOS, CNRS, Blaise Pascal University, Aubière, France
samuel@isima.fr

Abstract. The role of data warehouse for business analytics cannot be undermined for any enterprise, irrespective of its size. But the growing dependence on web services has resulted in a situation where the enterprise data is managed by multiple, autonomous service providers. The goal of our work is to investigate and devise an approach to address the trade-off between scalability and adaptability in large scale integration with numerous ever-evolving web services. We present our prototype DaWeS (Data warehouse fed with Web Services) and explore how ETL using the mediation approach benefits this trade-off for enterprises with complex data warehousing requirements. The semantic web research community has proposed various standards like WSDL, WADL, hRESTS, SAWSDL for describing web service interface (API) the usage of which could have solved our requirement of automated integration. DaWeS looks to fill the current gap between the industry and research community by taking into account the key characteristics of the aforementioned description languages and using a declarative approach in order to reduce the manual effort. We also present to the semantic web research community the optimization heuristics (to reduce the API operation calls) and semantic challenges (auto-adaptability especially in the wake of an API change) devised while building DaWeS.

Keywords: #eswcphd2014Samuel.

1 Introduction

The growing dependence on the internet has influenced the rise of many small service providers offering a reduced subset of services over the internet compared to the traditional bloated computer softwares and applications. Enterprises dependent on the web services have their business data spread across multiple data centers spanning even across continents leaving them with no direct control over the web services and the associated data infrastructure and thus their own business data. Web mashups and Data as a service (DaaS) have appeared in large numbers to provide integration with some selected web services in order to provide a consolidated view of the data spread across the web services. A classical approach is to write wrappers for each such web service for extracting the relevant information. But this approach is not scalable especially if the integration is targeted towards thousands of web services.

Extracting data from web services (WS) has several constraints. The WS providers generally expose the application programming interface (API) to their services so that their clients (enterprises) can build their own internal dashboards. The APIs differ significantly among each other with respect to the resources they handle, the resource representation, the data types, the number of operations, the service level agreements or SLA (that limits the number of operations that can be made during a period of time), the authentication mechanisms (for access by enterprise and third party users), the choice of message formats and the operation request and response (or error) parameters. In addition to these, WS periodically modify their API: adding support for new resources, deprecating some operations or changing the SLA. Therefore the clients often have to deal with this volatility in the interfaces often forcing them to change their service providers or their internal applications. Therefore enterprises require a solution to be able to have an integrated view of their business data spread across multiple web services and experience a transparent continuity of their data even when they switch their service providers or when WS API undergoes a change.

Our research work looks at the problem of large scale integration with the ever-evolving web services for business analysis in providing a scalable and adaptable solution towards this end. The solution must be scalable i.e., considering the ever-increasing availability of new web service providers in the market, it must be very easy to add a new API with minimum coding effort. It must be quickly adaptable (easy update of API changes) in the wake of the versatility (or evolution) of WS. Also it must be able to manage huge and permanent storage of enterprise data coming from the WS. In addition to providing certain default performance indicators, it must be very easy to define new ad-hoc performance indicators that totally avoids hard coding. Last but not the least, it must offer a way for the enterprise to keep track of its business data, even if one of its providers is no longer available.

To address the problem previously described, we propose (and build) a *Web services fed Data Warehouse*. In section 2, we describe the current state of the art. Section 3 describes the problem and present our contributions. Section 4 describes our research approach used in our prototype DaWeS (Data Warehouse fed with Web Services). Section 5 describes the results obtained. We also discuss various scientific challenges especially in terms of dealing with reducing the number of (expensive) API operartion calls, handling incomplete information and dynamic evolution of the warehouse. Section 6 discusses how to evaluate our results. Finally in section 7, we will discuss our ongoing and future course of actions and summarize our results.

2 State of the Art

We want to offer enterprises a lightweight but powerful and online data analysis tool. So this is basically an information integration problem. Information Integration Systems [25, 5] provides a uniform query interface across multiple heterogeneous and autonomous systems.

In this field, two approaches are well studied, namely the materialized and the virtual ones. A materialized information integration system often used for the purpose of data analysis and business performance measures computation is called a data warehouse (DW). A DW contains a copy of source data structured according to a single global schema. Many ETL(extraction-transform-load) tools have been studied [23] to clean and extract data from various data sources and transform them to a format according to the data warehouse schema and loading them to the DW. A majority of the works have dealt with extracting the data from existing legacy data stores (databases, spreadsheets, web pages, textual documents) in the form of of data wrappers [20], which are ad-hoc specific pieces of software to translate from the DW to the source and vice versa. It is sensible to code such specific wrappers since DW sources are usually stable. The typical example is when a big company builds a DW to analyze data coming from its various departments. In our tool, enterprise data comes from multiple sources (services), and they must be persistent in case of a service provider failure. So it is clearly a DW issue. The problem is that services are supposed to be quite unstable in time. So it is not realistic to build a specific wrapper for each source.

The virtualized information integration approach, referred to as the mediation approach, also structures sources' data into a single global schema, but without copying data. Sources' data indeed stay at the sources. The purpose of a mediator is more query answering than data analysis. User queries are formulated over the global schema, transferred to sources, and the query answers coming from the sources are then gathered by the mediator and presented to the user. There are mainly three ways of linking the data sources to the global schema: the Global As View (GAV) [5] Local As View (LAV) [7, 22] and Global Local As View (GLAV) [10]approaches. In GAV, each relation of the global schema is defined as a query over the source relations. In LAV, each source relation is defined as a query over the global schema relation. GAV mediators are known to offer good query answering properties, while facing an evolution in the sources may be difficult (e.g., adding a new source implies to potentially updating many relation definitions in the global schema). LAV mediators are known to easily handle source changes, while query answering is algorithmically more difficult. Indeed, the user query posed to the global schema must be rewritten into queries that can be posed to the source. And rewriting algorithms have a high complexity (NP-Complete at least). In GLAV, the global and local schema relations are mapped using a set of tuple generating dependencies (TGDs). LAV is easier than GLAV with respect to an algorithmic point of view. Despite the complexity of rewriting algorithms, using a LAV approach as ETL seems to be interesting for our DW. Indeed this DW has to be easily adaptable when sources (services) evolve. And that cannot be provided by GAV approaches. Obviously, this work has to be located in the field of WS. [14, 26] deals with data integration using WS, in the context of WSs using the various standards like XML, HTTP, SOAP, WSDL, UDDI. Our proposition can be seen as an implementation addressing more specifically the content and relationship aggregation [14] issues.

Many existing tools and standards concerning WS can be used to build (parts of) a DW fed with WS. They can be divided into the syntactic and the semantic ones. The most prominent syntactic ones are WSDL [24] and hRESTS [17] (both machine and human readable). The idea is that a machine readable interface format can enable automatic code generation and then automated WS integration (discovery, composition, etc.). But industry wide adoption is currently missing: majority of the WS APIs is described in human readable format (essentially HTML web pages) and are not meant for the direct consumption by software applications. This may be due to standards that are still in their infancy, to the effort based on these standards cannot handle every real situation. [21, 1] use web service composition [9] for creating data as a service (DaaS) and WS mashups based on web service standards.

Concerning semantic web technologies and ontologies, several works [19] discuss about integrating semantic web technologies (ontologies) to the WS for easier and automated information integration. DAML-S [6], OWL-S [18], SAWSDL (Semantic Annotations for WSDL and XML Schema) [13] and hRESTS are some examples of such semantic WS description languages. These approaches already provides techniques to automate integration in constrained contexts. Indeed, thanks to their formal semantics, they're adapted to automate integration issues (e.g., automatic matchmaking of services with respect to a user query). Moreover it is now known that the underlying formalisms, especially description logics, can be used to extend query answering to ontological query answering [3, 11]. We believe the industrial acceptance will increase in the upcoming years and our proposed solution can reap the benefits of these technologies for a fully automated integration.

3 Problem Statement and Contributions

The goal of our work is to investigate and devise an approach for a scalable and adaptable solution to the problem of integrating data coming from a large number of WS using their API, that also reduces the coding effort required for data integration. Towards this direction, we brought forth the notion of the classical LAV mapping (with access patterns) to describe the WS API operations. Web service APIs are thus considered as the data sources, and viewed as relational sources with access patterns (to translate the presence of input and output parameters) [22]. "Translators" must be coded to present each API operation as a relational source with access pattern to ensure the translation between API and the global schema. Translators can be viewed as light-weight and declarative wrappers made operational by a single generic wrapper. For every domain (example: project management), a global schema is set up. Queries are then defined on the global schema to link with an API operations viewed as relational sources with access patterns. Records can be defined as queries on the global schema (records can be viewed as atomic performance indicators). Performance indicators can be defined as queries on the schema composed by record predicates.

As said before, we claim that our approach of using mediation as an ETL is interesting since mediation with LAV-approach is known to fit data integration problems when relational views are evolving, so fitting well with the scalability objective. Mediation is known to be a declarative process since it uses databases query languages like datalog and conjunctive queries which are declarative. Therefore it also fits our adaptability objective, since it allows to reduce the coding effort (e.g., defining a new performance indicator is simply an SQL query, adding a new operation implies to code the "translator" and then add a mapping) To compute a record, only a generic wrapper is needed (coded once for all). No need to code one wrapper for every new source.

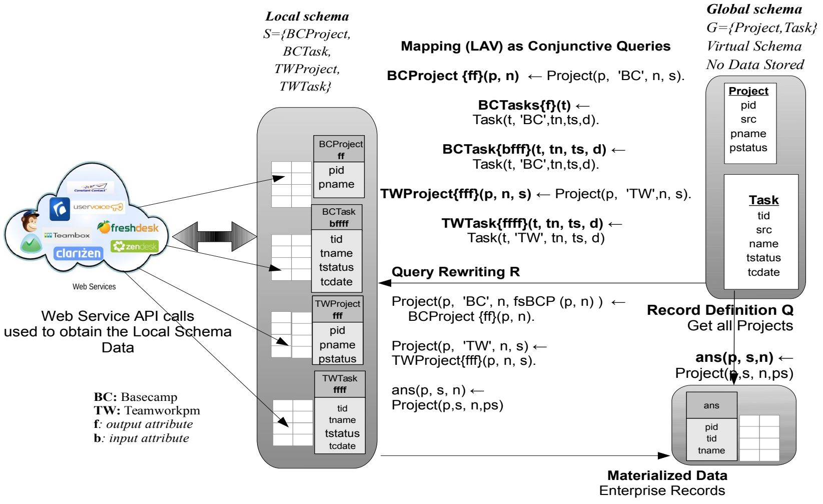
Local schema
S={BCProject, BCTask, TWProject, TWTask}
Global schema
G={Project,Task}
Virtual Schema
No Data Stored
Mapping (LAV) as Conjunctive Queries
BCProject {ff}(p, n) ← Project(p, 'BC', n, s).
BCTasks{f}(t) ← Task(t, 'BC',tn,ts,d).
BCTask{bfff}(t, tn, ts, d) ← Task(t, 'BC',tn,ts,d).
TWProject{fff}(p, n, s) ← Project(p, 'TW',n, s).
TWTask{ffff}(t, tn, ts, d) ← Task(t, 'TW', tn, ts, d)
Query Rewriting R
Project(p, 'BC', n, fsBCP (p, n)) ← BCProject {ff}(p, n).
Project(p, 'TW', n, s) ← TWProject{fff}(p, n, s).
ans(p, s, n) ← Project(p,s, n,ps)
Record Definition Q
Get all Projects
ans(p, s,n) ← Project(p,s, n,ps)
Materialized Data
Enterprise Records
Project
pid
src
pname
pstatus
Task
tid
src
name
tstatus
tcdate
BCProject ff
pid
pname
BCTask bfff
tid
tname
tstatus
tcdate
TWProject fff
pid
pname
pstatus
TWTask ffff
tid
tname
tstatus
tcdate
ans
pid
tid
tname
Web Services
Web Service API calls used to obtain the Local Schema Data
BC: Basecamp
TW: Teamworkpm
f: output attribute
b: input attribute

Fig. 1. Mediation approach using Local as View (LAV) Mapping

Example 1. Consider the domain *Project Management* that manages mainly two resources, Project and Task; hence two global schema relations:

$Project(pid, src, pname, pstatus)$ and $Task(tid, src, tname, tstatus, tcdate)$. and two WS: Basecamp[1] and Teamwork[2]. We consider here a simplified version of their API operations. Basecamp provides two operations related to task: the first operation provides all the task identifiers and the second requires the task identifier as input to give the complete task details. This information is captured by the access patterns. Consider LAV mapping $BCTask^{iooo}(t, tn, ts, td) \leftarrow Task(t,' BC', tn, ts, td)$. It corresponds to the fact that the operation $BCTask$ takes as input the task identifier t and gives the details of the task (name, status and creation date. The operation has been mapped to the global schema relation $Task$ with source value as BC to signify Basecamp, its source. Now consider a

[1] http://www.basecamp.com

[2] http://www.teamworkpm.net

record definition q: *Get all Projects*, (a conjunctive query formulated over the global schema). $q(pid, src, pname) \leftarrow Project(pid, src, pname, pcdate, pstatus)$. The query rewriting generated by the inverse rules algorithm generates queries using the WS API operations that are then used for the actual calls. The mediation approach for this example is described in Fig. 1.

4 Research Methodology and Approach

Our research methodology relies on the implementation of our approach to data integration and warehousing in a prototype, that can then be validated and evaluated against the requirements expressed above. We built the first version of the data warehouse, DaWeS handling every essential aspect required to make use of the WS API to extract information. The basic underlying architecture of DaWeS is shown in Figure 2. The component *Enterprise Record Computation*, is responsible for computing the enterprise records. It takes as input the record definition (query formulated over the global schema relations) and makes use of the *answer builder* to execute the query. Answer builder consists of a (datalog) query engine, that executes the query plan obtained by rewriting the query according to the inverse rules algorithm [7, 8].

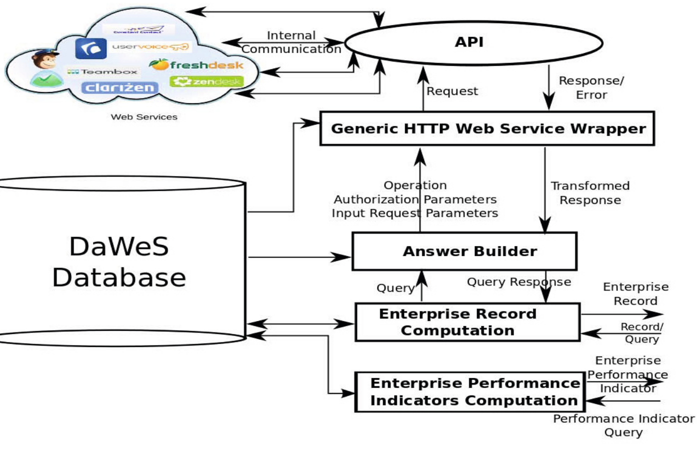
Internal Communication
API
Request
Response/ Error
Web Services
Generic HTTP Web Service Wrapper
Operation Authorization Parameters Input Request Parameters
Transformed Response
DaWeS Database
Answer Builder
Query
Query Response
Enterprise Record
Enterprise Record Computation
Record/ Query
Enterprise Performance Indicator
Enterprise Performance Indicators Computation
Performance Indicator Query

Fig. 2. DaWeS: Basic Architecture

We use the inverse rules algorithm given its capability to handle recursive datalog queries, handling access patterns in the source relations and to specify full and functional dependencies in the global schema relation (we used this capability to specify the primary keys). The *generic HTTP WS wrapper* is used to make the WS API operation calls and transform the response in a manner understood by the answer builder. The wrapper is generic since it can make any HTTP WS API operation call given the right URL, valid input and authentication parameters (in HTTP header and/or body). A response validator serves to validate the response before performing any transformation and to catch any unannounced (or unexpected) response schema changes. We also used cache with the wrapper in order to reduce the number of API operation calls so that any subsequent use of an operation call makes use of the cached response. On the completion of the datalog query evaluation, the query response is saved in the database to be later used by *Enterprise Performance Indicator Computation* component to compute interesting business measures.

During the course of the development of DaWeS, we identified various scientific locks. Reducing the number of expensive (bandwidth and cost perspective) API operation calls is important. The domain rules do not take into account any functional dependencies existing among the input attributes of an API operation. Suppose there is a functional dependency $t \rightarrow p$ between project identifiers p and task identifiers t, two input attributes for an API operation (e.g., in Figure 1, if $BCTasks$ is replaced by $BCTasks^{oo}(p,t)$ and $BCTask$ is replaced by $BCTask^{iiooo}(p,t,tn,ts,td)$. If there are 10 p and 20 t, the domain rules generated result in 200 (10×20) $BTask$ API calls (and not 20) because domain rule generate the following r.h.s $dom_p(p), dom_t(t), BCTask^{iiooo}(p,t,tn,ts,td)$, where $dom_p(p)$ and $dom_t(t)$ are the abstract domains corresponding to project identifier and task identifier respectively. Secondly, classical query rewriting algorithms [12] including inverse rules algorithm are shown not to be able to generate any rewritings for such queries. Thirdly, WS evolve and currently there are no standard mechanisms to track when an API (or an operation) is deprecated or changed. Fourthly, expecially considering the multi-domain WS environment, new global schema relations (especially when a domain of service has to be added) and LAV mappings updations occur regularly. Therefore contrary to the traditional warehouse settings, there is a dynamic evolution of the data sources, the warehouse schema, the queries formulated over the mediated schema and the performance indicators defined using these queries (for new domains).

5 Intermediate Results

DaWeS was tested with Intel(R) Pentium(R) Dual CPU @ 2.16GHz processor, system memory of 3GiB, Ubuntu 13.04 (32 bits) operating system, Oracle 11g (11.2.0.1.0) database and was developed and run using Java 1.7.0_25. We chose IRIS (Integrated Rule Inference System) [15] as the datalog engine to perform query evaluation. We configured IRIS to make use of the generic HTTP WS wrapper capable to make WS API operation calls to any web service.

We considered different domains and the associated WS APIs. For every WS API operation, we required the input and output parameters, the XSD [2] schema of the response and XSLT [16] to extract interesting information from the response. Various record definitions (mediated schema queries) were considered and the enterprise records (query responses) were stored in two tables.

We developed heuristics to deal with the scientific locks discussed before. First, in order to handle functional dependencies existing among two or more input attributes of a source relation, we implemented a static optimization heuristics by considering only valid input values (obtained from previous operations). Dynamic optimization [4] for this problem has been proposed recently. Secondly, a more recent paper [12] has shown the capability of inverse rules mechanism to handle incomplete information. We implemented this into DaWeS making use of the functional terms generated during inverse rules rewriting. Finally, we use a XML response validator to identify any unexpected API changes. We periodically compute certain records and compare the obtained response to the expected response (calibration). Response validator and calibration are our current semi-automatic error handling mechanisms to trigger a manual intervention on the event of WS evolution. DaWeS database employs two tables each for describing the mediated schema, the data sources, the mediated schema queries and the performance indicator queries, the enterprise records and performance indicator values. Advanced analytics queries like OLAP (used in conjunction with the star schema) under these settings is an open database research question.

6 Evaluation Plan

Our approach will be evaluated based on the requirements indicated previously, through measuring performance at different scales and the reduction in coding effort compared to other state of the art approach in data integration and ETL. We already performed various qualitative and quantitative experiments on DaWeS. We created 100 test organizations to simulate a multi enterprise environment. Following are the twelve different WS considered from three different domains; *Project management services*: Basecamp, Liquid Planner[3], Teamwork, Zoho Projects[4]. *email marketing services*: MailChimp[5], Campaign Monitor[6], iContact[7]. *support/ helpdesk services*: Zendesk[8], Desk[9], Zoho Support[10], Uservoice[11], FreshDesk[12]. We took into consideration 35 different operations of the 12 WS having the following characteristics: required no input arguments, required one or more input arguments or required paginated requests.

The queries formulated over the global schema consisted of a mix of (union of) conjunctive queries, and (recursive) datalog queries. We considered the following record definitions (queries), each computed on a *Daily* basis: New Projects(1), Active Projects (2), On-Hold Projects(3), On-Hold or Archived Projects(4), New Tasks(5), Open Tasks(6), Closed Tasks(7), Todo-Lists(8), Same Status Projects(9), New Forums(10), All Forums(11), New Topics(12), New Tickets(13),

[3] `http://www.liquidplanner.com`
[4] `http://www.zoho.com/projects`
[5] `http://www.mailchimp.com`
[6] `http://www.campaignmonitor.com`
[7] `http://www.icontact.com`
[8] `http://www.zendesk.com`
[9] `http://www.desk.com`
[10] `http://www.zoho.com/support`
[11] `http://www.uservoice.com`
[12] `http://www.freshdesk.com`

Open Tickets(14), Closed Tickets(15), New Campaigns(16) and Campaign Statistics(17). Twenty performance indicators were defined using SQL queries and record predicates. Example performance indicator queries include average resolution time of tickets during the last 30 days. An average (Mean) Time of *104.82 seconds* was taken to compute all the records of a single organization. Without incomplete data handling heuristics, many queries would have remained unanswered.

Figure 3 shows the time taken to compute the records using the WS API operations serially. The spikes point the situations when data is not available locally in the cache and the API operations have to be made. Use of cache and the proposed heuristics helped us to reduce the number of API operation calls, without which it takes too much time (many hours at least). Currently we are working on the precise semantics of the conjunctive query with access patterns in order to quantify (or predict) the number of API operation calls. This will help us to drive our heuristics to a complete algorithm.

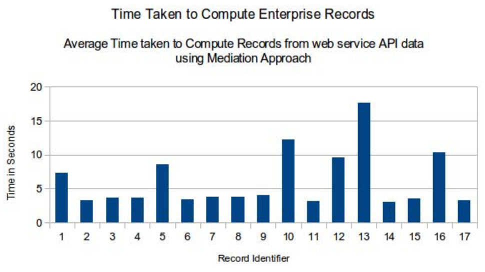

Fig. 3. DaWeS: Record Computation Time

7 Conclusion

Mediation as an ETL approach is very useful for building a WS fed data warehouse. Given its scalable and adaptable nature, it can be easily adapted by small and medium scale enterprises considering the minimum amount of coding effort while handling WS API. Inverse rules query rewriting used in conjunction with our proposed heuristics is useful to handle access patterns in the sources, data dependencies on the global schema, recursive datalog queries, incomplete information and optimize the number of expensive API operation calls.

DaWeS has been tested with real web services. Our further extensions include automatic error handling with an error ontology (e.g., to handle API changes, temporary API failures), adding more constraints in the form of TGDs to the global schema to get close to the kind of constraints used in the ontological query answering, but keeping recursive queries and defining the precise semantics of conjunctive queries with (or without) access patterns.

Given the interest of cloud computing within the industry, our current approach of storing the complete WS API enterprise data on just two big tables can be easily adapted to the cloud infrastructure. Also we want to design the system to avoid the complexity so that it may be more widely adopted than SAWSDL or other such standards.

Acknowledgement. I thank the Conseil General of the Region of Auvergne (France) and FEDER for funding our research project. I also thank Christophe Rey and Farouk Toumani of LIMOS, Blaise Pascal University, Franck Martin and Lionel Peyron of Rootsystem for their guidance and feedback for DaWeS.

References

[1] Benslimane, D., Dustdar, S., Sheth, A.P.: Services mashups: The new generation of web applications. IEEE Internet Computing 12(5), 13–15 (2008)
[2] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (xml). World Wide Web Journal 2(4), 27–66 (1997)
[3] Calì, A., Calvanese, D., Lenzerini, M.: Data integration under integrity constraints. In: Seminal Contributions to Information Systems Engineering, pp. 335–352. Springer (2013)
[4] Calì, A., Calvanese, D., Martinenghi, D.: Dynamic query optimization under access limitations and dependencies. J. UCS 15(1), 33–62 (2009)
[5] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The tsimmis project: Integration of heterogeneous information sources. In: Proceedings of IPSJ Conference, pp. 7–18 (1994)
[6] Coalition, D., Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D., Mcdermott, D., Narayanan, S., Mcilraith, S.A., Paolucci, M., Payne, T., Sycara, K.: Daml-s: Web service description for the semantic web (2002)
[7] Duschka, O.M., Genesereth, M.R.: Answering recursive queries using views. In: PODS, pp. 109–116 (1997)
[8] Duschka, O.M., Genesereth, M.R., Levy, A.Y.: Recursive query plans for data integration. J. Log. Program. 43(1), 49–73 (2000)
[9] Dustdar, S., Schreiner, W.: A survey on web services composition. International Journal on Web and Grid Services 1(1), 1–30 (2005)
[10] Friedman, M., Levy, A.Y., Millstein, T.D.: Navigational plans for data integration. In: AAAI/IAAI, pp. 67–73. AAAI Press / The MIT Press (1999)
[11] Gottlob, G., Schwentick, T.: Rewriting ontological queries into small nonrecursive datalog programs. In: KR. AAAI Press (2012)
[12] Grahne, G., Kiricenko, V.: Towards an algebraic theory of information integration. Inf. Comput. 194(2), 79–100 (2004)
[13] Group, S.W.: Semantic annotations for wsdl, w3c working draft. The World Wide Web Consortium, W3C (2006)
[14] Hansen, M., Madnick, S.E., Siegel, M.: Data integration using web services. In: DIWeb, pp. 3–16. University of Toronto Press (2002)
[15] IRIS: Integrated Rule Inference System - API and User Guide (2008), `http://www.iris-reasoner.org/pages/user_guide.pdf`
[16] Kay, M.: Xsl transformations (xslt) version 2.0. W3C Recommendation 23 (2007)
[17] Kopecký, J., Gomadam, K., Vitvar, T.: hrests: An html microformat for describing restful web services. In: WI-IAT 2008. IEEE Computer Society Press (2008)
[18] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T.: Owl-s: Semantic markup for web services. W3C Member Submission 22, 2007–4 (2004)
[19] Noy, N.F.: Semantic integration: A survey of ontology-based approaches. SIGMOD Record 33 2004 (2004)

[20] Roth, M.T., Schwarz, P.M.: Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In: VLDB 1997, pp. 266–275 (1997)
[21] Truong, H.L., Dustdar, S.: On analyzing and specifying concerns for data as a service. In: Kirchberg, M., Hung, P.C.K., Carminati, B., Chi, C.H., Kanagasabai, R., Valle, E.D., Lan, K.C., Chen, L.J. (eds.) APSCC, pp. 87–94. IEEE (2009)
[22] Ullman, J.D.: Information integration using logical views. Theor. Comput. Sci. 239(2), 189–210 (2000)
[23] Vassiliadis, P., Simitsis, A.: Extraction, transformation, and loading. In: Encyclopedia of Database Systems, pp. 1095–1101. Springer (2009)
[24] W3C: Web Service Description Language 1.1 (2001), http://www.w3.org/TR/wsdl
[25] Wiederhold, G.: Mediators in the architecture of future information systems. Computer 25(3), 38–49 (1992)
[26] Zhu, F., Turner, M., Kotsiopoulos, I.A., Bennett, K.H., Russell, M., Budgen, D., Brereton, P., Keane, J.A., Layzell, P.J., Rigby, M., Xu, J.: Dynamic data integration using web services. In: ICWS. IEEE Computer Society (2004)