Integration of Multiple Heterogeneous and Autonomous Web Services using Mediation Approach: Open Challenges

John Samuel Samuel, Christophe Rey

Integration of Multiple Heterogeneous and Autonomous Web Services using Mediation Approach: Open Challenges

John Samuel
Université de Lyon, LIRIS
France

Christophe Rey
Université Blaise Pascal, LIMOS
France

***Abstract*—Regular users and enterprises are now increasingly dependent on web services. This growing dependence on one hand has simplified routine tasks, but on the other hand it has resulted in loss of direct control over the data. Nevertheless, both users and enterprises require simplified and generic solutions to access their data. The classical mediation approach from the data integration field provides a uniform query interface to diverse data sources hiding the underlying heterogeneity. But using this approach over multiple heterogeneous and autonomous web services has several open challenges. In this article, we will take a look at some of these challenges that need to be addressed for achieving a fully automated solution.**

I. Introduction

Service providers are increasingly opting to provide service solutions over the internet to reach out to their clients. This approach often referred to as Software as a Service (SaaS) has the potential to target people with different devices and operating systems. There are several reasons for this shift. One among them is the growing usage of web browsers and mobile applications. Software solutions for personal computers limit their usage to specific platforms whereas web services are accessible to a very wide audience of people with diverse devices having internet access.

Not only regular users, even small and medium scale enterprises are now increasingly dependent on web services for their daily requirements. On one hand, it has enabled them to focus on key aspects of their business without worrying about data management. But on the other hand, this growing dependence on web services has also resulted in loss of direct control over their enterprise data. API (Application Programming Interface) provide a convenient mechanism for enterprises to access and manipulate their data managed by the service providers. It allows the clients to build their own internal applications and dashboards. Service providers expose the API so that both the clients and third party users authorized by the clients can access and manipulate the client data.

But web services managed by different providers are heterogeneous. It is practically difficult for any small enterprise to write applications to integrate with every web service they use. Therefore a generic solution is required to integrate with these multiple heterogeneous and autonomous web services [1], [2]. Such a solution will enable them to have access to their historical data. Integrating data from various sources, also called data integration is therefore an important topic of research. Several approaches have been proposed [3], [4], [5], [6] and tested on data sources like distributed databases, web pages. Mediation approach [4], [7], [8] is a virtual integration approach where a uniform query interface is provided to a multitude of heterogeneous and autonomous data sources. The data sources form the original source of information. The data source relations form the local schema. A mediated or global schema is created over which the user queries are formulated. Global schema relations and the local schema relations are linked with the help of mappings. These mappings are relation definitions expressed with respect to either global schema relations or source schema relations. Any query posed over the global schema needs to be translated to a query involving the source schemas. The users of a data integration system may or may not have direct control over the schema of the data sources (like in the case of autonomous web services).

A fully automated and generic solution to integrate with multiple web services is a challenging research and industrial problem. In this article, we take a look at various problems concerning a fully automated integration with multiple heterogeneous and autonomous web service API using the mediation approach. Section II presents the overall problem in detail, briefly presenting the various available interfaces to web services, how some of these interfaces are used for integration and the mediation approach. The section also presents a detailed example that we will use throughout the article to explain the challenges. We then discuss the various challenges in section III and finally conclude the article in section IV.

II. The Problem

Every service domain like project management, email marketing, helpdesk, accounting etc. manages specific resources. Take for example, a project management service domain manages the various projects and the related tasks of an

This is a longer and revised version of the article 'Challenges in Integrating Multiple Heterogeneous and Autonomous Web Services using Mediation Approach' presented in ICDIM 2016, Porto, Portugal.

enterprise or a user. Some of the important resources associated with project management services are projects, tasks or specific ones like open projects, archived projects, open tasks, completed tasks etc. Web service providers provide an interface for direct human consumption making use of web technologies like HTML, stylesheets (CSS), javascript etc. Clients access this interface through web browsers. Therefore web service utilization follows a client-server architecture; services are provided by the servers of the service providers and the users with their browsers constitute the clients.

For the sake of simplicity, we assume that resources can be represented by a tuple of values by the web services (especially those using relational database technologies). Recall that such a form of representation can be easily extended to database tables, XML, JSON or any other structured formats. However, the internal storage formats used by the web services may remain unknown to the end user. When the clients use a web service for the first time, they are asked to provide various authentication credentials for creating an account on the servers of the web service. These credentials are meant for subsequent usages in order to ensure only users with the right credentials are accessing the data. Using a web service means accessing, manipulating or deleting the client resources. Clients can also authorize third party users to use these web services on their behalf, for example for integrating with a third party service. For this purpose, service providers also provide various authorization mechanisms to help the clients decide what usages are permitted to the third party users.

Additionally, web services often provide another interface called application programming interface (API). API is meant for machine-to-machine communication; users can write their own programs and applications that can automatically communicate with the web services. The usual message format used for the communication between clients and servers is XML or JSON. A web service API has one or more operations. Operations are meant to access, manipulate or remove the resources. They are usually referred to as CRUD operations taking into account their ability to (C)reate new resources, (R)ead, (U)pdate or (D)elete some existing resources from the services. In this article, we focus only on the (R)ead operations that give the ability to access the resources and we refer them as data-providing operations [9]. Each such operation takes zero or more input parameters and returns zero or more tuples of values. API operation calls may also respond with errors like internal server errors, non-availability of services or missing resource error. The client-side applications make appropriate decisions on receiving an error (like logging the error or alerting the user).

The input of some API operations are dependent on the output of some other API operations of the same service or other services. Thus it requires that operation calls are made in a certain sequence. In some cases, to obtain complete details of resource, the same operation must be called with varying parameters like page numbers. This approach called pagination is commonly found in search engines and comes in a variety of forms. API calls also require authentication and authorization parameters.

APIs are documented in several ways. Research and industrial community have previously proposed various machine readable standards to describe the web service APIs, particularly WSDL [10] and WADL [11]. These machine readable formats are used to automatically generate codes that can integrate with web services. Another common approach is to describe the API in a human-readable format like the textual description. This human-readable documentation is used by developers to understand various API operations, to derive required operation sequences and to make appropriate decisions pertaining to the desired operations for a particular application. A third approach is to provide both human-readable and machine-readable (on the same web page) to target developers as well as to support the automated integration of applications with web services. Such an approach is seen in the proposal of a standard like hRESTS[12].

Web services catering to the needs of numerous clients often impose some limits like limiting the number of API operation calls that can be made during a period of time. These limits often grouped under service-level agreements or SLA specifies the number of API calls that can be made by a client or a particular IP address of the client.

Furthermore, service providers often make updates to their services like adding, modifying or removing resources, changing message formats for communication, deprecating certain API operations, offering new API without any backward-compatibility as well as changing the SLA. These changes are usually announced by emails, official blog posts or other social media websites. Any of the above changes requires the clients to redevelop their internal applications.

A. Integration using Web Service API

Integration with web services is a well researched field. These research works differ on whether they approach the problem from the perspective of the service provider or from that of the client. Works related to the former have given a greater thrust to promoting machine-readable API documentation by the service providers. Use of machine readable API documentation ensures that clients can make use of some generic automated code-generators to generate programs for integrating their internal applications with the concerned web service.

Several works like [13] study data integration using web services built over web standards like XML, HTTP, SOAP, WSDL, UDDI for the purpose of aggregation. Considering the rapid advancement of mobile devices, an upcoming approach is to offer API integrated software development kits (or SDK) [14] in various programming languages. This approach lets the clients download the latest version of the official SDK and easily integrate it with their internal application without the need for writing a lot of code. However, this approach is limited to making available the SDK in popular programming languages of the code. Some clients with platforms developed in other programming languages cannot conveniently make use of these SDKs. Also this SDK approach does not the

overcome the challenge of evolving API (or SDKs) that requires modification to the existing client codebase.

Research works that look at the problem from the client perspective suggest solutions to easily add or remove new web service API. ActiveXML [15] is a language that extends XML to allow the embedding of web services calls. Web mashups [16], [17] compose one or more web services to create interesting applications. One commonly used approach to creating them is by using a graphical composition tool, a visual programming approach to reducing the programming overload and to deal with different API operation invocation sequence patterns. Due to the requirement of manual efforts, the number of web services that can be integrated is still very limited. Also the key aim of web mashups is to make advantage of diverse web services to create new interesting web services.

B. Mediation Approach

Data integration [18] aims to provide a uniform query interface to multiple heterogeneous and autonomous data sources. The mediation (global) schema is primarily used to query the data sources and is not materialized (hence, the name virtual). In mediation systems, we recall that there are three common ways by which the sources (the local schema relations) can be mapped to the global schema relations: Global-as-View(GAV) mapping [7], Local as view(LAV) mapping [19] and Global-Local as view mapping (GLAV) [20]. In GAV, each relation of the global schema is defined as a query over the local source relations wheras in LAV, each (local) 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 request the sources. And rewriting algorithms have a high complexity (NP-Complete at least) [21]. 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. Choosing GAV would amount to changing the mappings on a frequent basis. Whereas in the case of LAV, a new data source will simply require a new LAV mapping and no change in the existing mappings are required.

Examples of such information systems include Infomaster[22], TSIMMIS [23] and Information Manifold[24]. Infomaster uses rules and constraints to describe the various data sources. TSIMMIS uses GAV mapping whereas Information manifold considers the query planning under the LAV settings. There are some more works focusing on the use of declarative languages especially for XML-based data sources. Enosys XML Integration Platform (EXIP) [25] is an XQuery-based integration platform. [26] makes use of semantic web standards like RDF and SPARQL and the mediation approach with (ontological) query rewriting to provide on-demand automated integration of data providing web services.

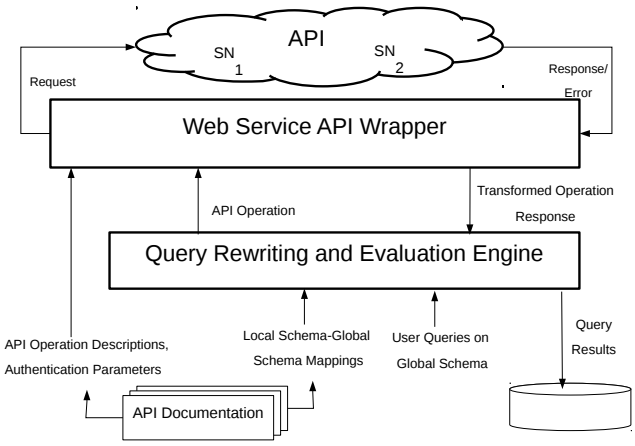

Fig. 1. Mediation approach for extracting data from web services

In the context of data integration with web services, every web service API operation is seen as a relation with access patterns[19], considering the input and output parameters of the operation as relation attributes and the operation name as relation name. These operations (seen as relations with access patterns) are defined using the global schema by applying a mapping technique: conjunctive query[27] based LAV (Local as View)[19] mapping. A translation process called query rewriting rewrites the user queries formulated over the global schema relations to the actual web service API operations. A query evaluation engine evaluates this rewritten query and the query response thus obtained form a (historical) record. These records can be stored and later used for various purposes like data analyses (e.g., to compute the performance indicators).

C. Example on Mediation Approach

Consider the domain of social networking. A user shares updates with other users and can also view the details of a particular status (or update) like the creation date of the status and the view count. Therefore two primary global schema relations in this domain can be *user* and *status*.

$user(userid,\ source,\ name)$

$status(statusid, source, userid, creationdate, viewcount)$

where $userid$ corresponds to the identifier of the user, $source$ to the name of the social networking web service, $name$ to the name of the user, $statusid$ to the identifier of a particular status, $creationdate$ to the creation date of the status and $viewcount$ to the total number of status views.

Consider two social networking web services: SN_1 and SN_2. Both expose their services through API. SN_1 uses XML data format whereas SN_2 uses JSON.

SN_1 provides the following API operations. Superscript i is used to denote input parameters and similarly o is used to denote output parameters.

1) $sn1myid^{o}(userid)$: returns the user identifier of the respective owner of the social networking service.
2) $sn1user^{io}(userid, name)$: takes as input the user identifier and returns the name of the user.
3) $sn1status^{iooo}(userid, statusid, creationdate, count)$: takes as input the identifier of a user and returns the details of statuses like the status identifier, the creation date and view count of the status.

SN_2 provides the following API operations

1) $sn2myid^{oo}(userid,\ name)$: returns the user identifier and name of the respective owner of the social networking service.
2) $sn2update^{iooo}(userid, statusid, creationdate, count)$: takes as input the identifier of a user and returns the details of all statuses of the user like the status identifier, creation date and view count.

For simplicity, we consider simple conjunctive-query [27] based LAV mapping as given below:

$sn1user^{io}(userid, name) \leftarrow user(userid,\ `SN_1',\ name)$
$sn1myid^{o}(userid) \leftarrow user(userid,\ `SN_1',\ name)$
$sn1status^{iooo}(userid, statusid, creationdate, viewcount) \leftarrow status(statusid, `SN_1', userid, creationdate, viewcount)$
$sn2myid^{oo}(userid, name) \leftarrow user(userid, `SN_2', name)$
$sn2update^{iooo}(userid, statusid, creationdate, count) \leftarrow status(statusid, `SN_2', userid, creationdate, viewcount)$

We present example XML/JSON responses for two of the above operations.

$sn1status^{iooo}(userid, statusid, creationdate, viewcount)$ returns the following message.

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
 <statuses>
    <status>
       <userid>10</userid>
       <identifier>1022</identifier>
       <text>I am on my way</text>
       <date>2016-03-04T11:20:18+02:00</date>
       <count>25</count>
    </status>
    <status>
       <userid>10</userid>
       <identifier>1023</identifier>
       <text>Giving a presentation</text>
       <date>2016-03-04T13:38:48+02:00</date>
       <count>2</count>
    </status>
 </statuses>
</xml>
```

A simple transformation using XSLT to obtain desired information is given below. It transforms the above XML message to a list comma-separated values of user identifier (userid), status identifier, its creation date and the text value.

```
<xsl:stylesheet
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fn="http://www.w3.org/2005/xpath-functions"
  version="2.0">
 <xsl:output method="text"
  omit-xml-declaration="yes"
  cdata-section-elements="namelist"/>
 <xsl:template match='/'>
  <xsl:for-each select='statuses/status'>
   <xsl:value-of select='userid'/>,
   <xsl:value-of select='identifier'/>,
   <xsl:value-of select="fn:substring(
     current()/date,1,10)"/>
   <xsl:text>&#xa;</xsl:text>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

Example JSON response for $sn2myid^{oo}(userid, name)$ is given below:

```
{
      "identifier": "345",
      "name": "Kevin Bob"
}
```

A user poses query over the global schema: Get me all statues from '2016-03-04' and their viewcounts from various social networking services.

$q(statusid,\ source,\ count) \leftarrow status(statusid,\ source,\ userid,\ `2016-03-04',\ viewcount)$

In the mediation approach, the above query is translated to query formulated over the local schema relations (SN_1, SN_2 API operations) using query rewriting approach and then evaluated using a query evaluation engine. Classical query rewriting algorithms include the bucket algorithm [28], minicon algorithm [29] and the inverse rules algorithm [30], [19]. An inverse rules query rewriting is given below.

$user(userid,\ `SN_1',\ f(...)) \leftarrow sn1myid^{o}(userid).$
$sn1userid(userid) \leftarrow sn1myid^{o}(userid).$
$status(statusid, `SN_1', userid, creationdate, viewcount) \leftarrow sn1userid(userid), sn1status^{iooo}(userid, statusid, creationdate, viewcount).$
$user(userid,\ `SN_2',\ name) \leftarrow sn2myid^{oo}(userid,\ name).$
$sn2userid(userid) \leftarrow sn1myid^{o}(userid).$
$status(statusid, `SN_2', userid, creationdate, viewcount) \leftarrow sn2userid(userid), sn2update^{iooo}(userid, statusid, creationdate, count).$
$q(statusid,\ source,\ count) \leftarrow status(statusid,\ source,\ userid,\ `2016-03-04',\ viewcount).$

Readers may have noticed two domain rules [19] with heads $sn1userid$ and $sn2userid$. They correspond to the respective data types of these identifiers.

For more examples of rewritten queries, refer [19], [31], [32]. The overall approach of extracting data from web services is shown in Figure 1. *Query Rewriting and Evaluation Engine* rewrites query formulated over the global schema to queries described using the API operations (or relations with access patterns) using the global-local schema mappings. During the evaluation, on encountering any API operation, the *Web Service API Wrapper* is invoked which makes use of the web service descriptions (HTTP URL, body, method, message format) and user authentication parameters, the details of which are obtained from API documentation to make

request calls to API. The API operation responses (e.g., like XML/JSON sample responses shown above) are then transformed to a desired internal format (a simple e.g. 2016-03-04T13:38:48+02:00 to 2016-03-04) and fed back to the engine which computes the final result. Query results are optionally stored to the database for any possible future use.

III. Challenges

There are several challenges to obtain a fully automated and generic solution to integrating with numerous, heterogeneous, autonomous web services with mediation based data integration approach. They are highlighted in Table I and need to be addressed for achieving a fully automated and efficient solution of Figure 1.

A. Missing Standards and Information in API Documentation

APIs are documented in several ways targeting both developers and machines. Research and industrial community previously proposed many machine readable standards [33], [34] to describe the web service APIs, particularly WSDL [10], SA-WSDL [35], SA-REST, hRESTS [12], OWL-S, WADL and USDL [36]. These (semi)-machine readable formats are useful to automatically generate codes that can integrate with web services.

Architectural styles like REST were proposed by [37] to supporting loose coupling between clients and servers. This loose coupling can be achieved by several ways including creating stateless servers, the use of self-describing messages, hypermedia as the engine of application state etc. Initial web browsers, for example use protocols like HTTP [38] and HTML for communicating with (stateless) web servers. Use of standardized protocols and status codes helps browser designers to develop a uniform solution to retrieve web pages from multiple autonomous servers.

REST approach is also used in web service API communication. [39] analyses the use REST APIs of mobile traffic (HTTP) for a period of one day analyzing the use of tunneling requests using a single HTTP method (POST), handling of resources (especially their names and associated operations) and the use of hypermedia. It concludes that there exists a lot of difference between the practice of current generation of web services usually referred to as 'RESTful' and the actual theory [37]. A similar observation has also been made by [32] focusing on API documentation of three domains of services concluding that there still exists web services that are not described using a machine-readable language (with or without semantic features) and REST principles are not fully implemented.

[39], [32] also show that request parameters of data providing operations are passed through the URL, header or the body of the HTTP request with different HTTP methods (not confined to HTTP GET). Operation request and response parameters in XML/JSON formats are often described in the documentation in human-readable texts like enumerated list of parameters. Thus it requires extra manual effort to translate this information to associated XSD/JSON-schema for validating operation responses.

Another major challenge is a missing common approach to describing API operation errors [39], [32] and authentication [40]. HTTP status codes have categorized errors to support their distinction and taking appropriate actions (like in internet browsers). There are several instances where 200 (Success HTTP status code) is used for both successful and failed API operation calls. The client, therefore, has no way to distinguish a successful message from the failed response. In such cases, new status codes are introduced by the services and passed in the response body. This requires parsing of received response, thereby losing an important benefit of HTTP status codes.

Several assumptions made by the services concerning the internal schema is missing in the documentation. Some commonly missing information include missing data dependencies assumed by the service providers. Take for example, the (primary) keys used in the resource representation are not usually documented. The clients usually need to make assumptions or infer them by making several API calls. Data dependencies like full dependencies, functional dependencies, inclusion dependencies are useful in optimizing [41] the number of API operation calls.

B. Expressive Mappings and Query Rewriting

End-users are usually not interested in complete details of a resource. Take for example, business-metrics requiring applications are not interested in various attributes of a resource like a detailed description of a project, task or a user status. In the above XSLT example, we didn't make use of status text ($statuses/status/text$) from the XML response. Some services, especially web search API services do offer options to selectively filter desired attributes of a resource and even desired range of values.

Another major limitation is a missing search API operation that could enable the application to specify the desired resources matching certain criteria (e.g. new resources created during a certain period or the resources that underwent some recent changes). This missing search and filter operations on one hand necessitates several unnecessary API calls and bandwidth usage and on the other hand, it also puts an additional burden on the client applications to filter out the desired result. Continuing with our above example, suppose SN_1 and SN_2 API had operations using which, the user can specify a range of dates to obtain user updates for a given time period or an operation to obtain the status on a given date or time. It would have reduced a lot of internet usage consumption and the unnecessary XML parsing to just extract the status on a given date.

But describing the above search/filter API operations with expressive global-local mappings also comes at a cost. Query answering using views with arithmetic (comparison) operations like $\geq$ or $\leq$ is another key research area in mediation approach. As mappings become more expressive, so does the associated complexity of query rewriting [21], [42] .

S.No.	Challenge	Component
A.	Missing Standards and Information in API Documentation	API Documentation, Query Rewriting and Evaluation, Web Service API Wrapper
B.	Expressive Mappings and Query Rewriting	Query Rewriting and Evaluation
C.	Errors/Failure Handling mechanism	Query Rewriting and Evaluation, Web Service API Wrapper
D.	Handling Incomplete Information from Web Services	Query Rewriting and Evaluation
E.	Optimization	Query Rewriting and Evaluation
F.	Data Transformation for Subsequent Operation Calls	Web Service API Wrapper
G.	Evolving Web Service API	API Documentation, Query Rewriting and Evaluation, Web Service API Wrapper
H.	Scheduling API Operation Calls	Query Rewriting and Evaluation
I.	Data Model	Storage

TABLE I
CHALLENGES IN AUTOMATED INTEGRATION OF WEB SERVICES USING MEDIATION APPROACH

C. Errors/Failure Handling mechanism

As discussed above, HTTP error status codes [38] are used in some web service API. Following are some of the commonly encountered errors encountered during operation calls: operation request timeout, web service internal error, permission denied, account revoked permission, API deprecation, API operation response changes, incorrect passage of operation request(input) parameters, internal web service temporarily unavailable, change in message formats (XML, JSON, plain-text), operation deprecation etc. A common approach on encountering any such error is to abort the query evaluation and record the error status. In case of web browsers, the user on encountering any of the above status codes decides whether to refresh the browser after a certain time (e.g. on internet service temporarily unavailable) or avoid any future tries (e.g. on resource not found error). However a missing common standard to describe the errors makes it difficult to make appropriate automated decisions like whether to repeat the API call after a period of time, alert the user/administrator or totally avoid making the calls of the concerned API.

As presented in the example above, (i)nput and (o)utput adornments are used to specify the input and output attributes of a relation (or an API operation). Much of the research works on mediation and access patterns discussed above concentrate on these two adornments. [43] considers additional patterns like unspecifiable and optional attributes, a common feature in web services. These patterns commonly found in the API need to be considered along with the errors. Whether error attributes need to be considered in a separate relation with access patterns or in the same relation as a new adornment/superscript i.e., (e)rror is still an open question. The associated semantics, therefore, need to be defined and studied.

D. Handling Incomplete Information from Web Services

Not all the web services of a service domain provide all the information related to the global schema relations. Take for example, if $sn2update$ did not provide the view count, several classical query rewriting algorithms including bucket algorithm, minicon algorithm and inverse query rewriting algorithm may not provide any results from $SN2$. In some cases, it is interesting to have the available information even if it is incomplete [44]. There are several ways by which the incomplete information is represented: using Codd nulls, marked nulls and horn tables. Codd nulls do not cover various interesting information (e.g., when two unknowns are known to have the same values). Even though horn tables are shown [45] to be an efficient tool for handling incomplete information in databases, they are difficult to be used along with the current relational databases. Marked nulls are useful in a mediation approach and they have been used for the purpose of query rewriting [44] by bringing in the notion of p-containment, but under limited LAV settings. Another interesting direction is to consider the case when the data from the web services are uncertain.

E. Optimization

API operation calls are expensive since they consume internet bandwidth and therefore need to be optimized. Several optimizations have been proposed in the mediation approach. These may be classified in two: static and dynamic approaches. Static approaches ensure that the final rewritten query is optimized before actually evaluating it whereas dynamic approaches ensure optimization during query evaluation. This problem studied in a limited context by [41] suggests dynamic query optimization under the functional dependencies existing among the attributes of the relation, especially considering the case where a single relation may have multiple access patterns. Optimizations have been proposed for conjunctive queries using sources with access patterns [46] by making use of the optimized dependency graph in order to reduce the number of accesses to the external data sources. [47] optimizes the query answering problem by removing any useless accesses to the sources that don't contribute to the query's answer. [31] which uses inverse-rules algorithm suggests the use of two optimizations one in the form of tuple-level filtering and an algorithm that transforms the query plan into dataflow-style streaming execution plan in order to reduce the number of calls to the web services and executing the generated query plan efficiently.

Another optimization approach is to make use of caching by which the number of API operation calls can be reduced by caching the API operation responses and use appropriate cache eviction policy. Considering the various operation invocation sequences, there are some operations that may be called more often that the others. Hence there are several open questions that need to be answered: whether to cache the complete response or the transformed response, what type of cache eviction policy need to be used and under what circumstances.

Some caching heuristics have also been proposed in works like [46] to optimize the number of API operation calls.

F. Data Transformation for Subsequent Operation Calls

Wrappers [48] play an important role in data integration. In the case of web services, they play an important role [49] in the transformation of previous operation responses for subsequent operation calls. One good example is the pagination. Web search results have been paginated [50] for quite a long time. Similarly, API reduces the size of an operation response by supporting paginated API operation responses (e.g., *Paged-Collection* in Hydra [33]). There are several ways by which pagination is supported; following are some of them:

1) *(Page Number)*: This approach is commonly found in web search. Search query responses include the total number of pages on the first page. Subsequent pages can be obtained with a number less than or equal to the total number of pages. Here the default page size (or the number of individual results on a page) is fixed by the service provider.
2) *(Page Number, Page size)*: It is similar to the previous approach; the only change is that the end user is usually given a possible list of page sizes to make a choice from.
3) *(Page Links)*: In this approach, the total number of results may not be known. Every page has links to the first page, next page, previous page and last page. Navigating these links help in finally obtaining the complete result.

The above cases how desired information need to be extracted from operation responses and need to be transformed for subsequent operation calls.

Let's reconsider the above example of $sn2update^{iooo}(userid,\ statusid,\ creationdate,\ count)$. We now consider one additional input parameter, i.e., $sn2updatenew^{iiooo}(pageno, userid, statusid, creationdate, count)$ where $pageno$ corresponds to the page number, a term commonly used when all updates (or search results) are not given in a single call, but requires multiple calls. We now introduce another API operation call $sn2totalupdate^{io}(userid,\ updatecount)$ that gives the total number of updates $updatecount$ for a given user $userid$. Thus the total number of pages must be calculated (by the wrapper) making use of the value $updatecount$ and default page size before making calls to the $sn2updatenew$. [32] presents a similar but complete example.

G. Evolving Web Service API

Changes to the web services API and their impact on the clients have been extensively studied in [51], [1], [2]. Depending on the chosen global-local schema mapping (e.g. GAV, LAV), web service API evolution requires changes to these schema mappings, changes to web service descriptions as well as the need to request for new authentication and authorization from the end-users. Changes to the API are usually announced through blogs, email updates etc. and in some cases, through API operation responses. This may make it difficult to automatically shift to the newer versions. Periodical monitoring of blogs and emails for any change announcements may require some human intervention.

H. Scheduling API Operation Calls

Service providers enforce certain terms of use on API usage. These include limits to a number of API operation calls that can be made in a given period of time. These service level agreements between clients and service provide is to ensure a better quality of service (QoS e.g., availability). There is no common way [32] across service providers to specify these constraints. Some service documentation have special sections like *API limits and usage* whereas, in the case of others, it needs to be detected through error message whether some unknown limits have been exceeded. Therefore it is significantly difficult to build a generic mechanism for scheduling the API operation calls.

API operation calls are usually synchronous, i.e., the caller makes the operation request and waits for the response. A scheduler for a mediation approach must take into account client resource constraints and heterogeneous SLA. Two approaches can be considered in mediation approach: serial API operation invocation and parallel invocation. In the first approach, API operation calls are made in a sequence [52], one call at a time. Though simpler, it is not a scalable approach and also it does not make the best utilization of client resources. In the second approach, a thread pool [53] is created and operations that can be executed in parallel are made. Take for example, $sn1myid$ and $sn2myid$ calls can be made in parallel. Similarly, once the above calls have returned results, $sn1status$ and $sn2update$ can be made in parallel. Therefore the query evaluation engine in a mediation approach must be able to perform parallel operation calls, but keeping the access patterns in consideration.

I. Data Model

In mediation approach, the global schema is usually not materialized. However, for several practical reasons, it is useful to store the query responses for future use. But modeling a storage schema for different types of query responses is challenging, especially because end-user queries may come in various forms. Transforming query responses to formats like JSON (or XML) or RDF triples for storage is one possible solution for rapid visualization, for example. If it involves further querying, data storage solutions including SQL, NoSQL, triple stores, graph databases can be considered depending on the usage of these responses.

IV. Conclusion

In this article, we presented various open challenges in integrating multiple heterogeneous, autonomous and evolving web services using mediation approach. Enterprises depending on such web services require a fully automated solution for extraction and further utilization of their data. Mediation approach allows a declarative approach to the overall problem of describing and querying desired data from web services.

Nevertheless, there are several open challenges namely optimization of a number of API operation calls, handling incomplete information and error responses, dealing with different service level agreements and inferring required information from web service documentation.

ACKNOWLEDGEMENT

Much of this work was done at LIMOS, Université Blaise Pascal. We thank the Conseil General of the Region of Auvergne (France) and FEDER for funding our research project. We also thank Farouk Toumani of Université Blaise Pascal, Franck Martin and Lionel Peyron of Rootsystem for their feedback during the development of DaWeS [32].

REFERENCES

[1] S. Wang, W. A. Higashino, M. A. Hayes, and M. A. M. Capretz, "Service evolution patterns," in *2014 IEEE International Conference on Web Services, ICWS, 2014*, 2014, pp. 201–208.

[2] T. Espinha, A. Zaidman, and H. Gross, "Web API growing pains: Loosely coupled yet strongly tied," *Journal of Systems and Software*, vol. 100, pp. 27–43, 2015.

[3] C. T. Kwok and D. S. Weld, "Planning to gather information," in *In Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996, pp. 32–39.

[4] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying heterogeneous information sources using source descriptions," in *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. Morgan Kaufmann, 1996, pp. 251–262.

[5] L. Yu, W. Huang, S. Wang, and K. K. Lai, "Web warehouse - a new web information fusion tool for web mining," *Information Fusion*, vol. 9, no. 4, pp. 501–511, 2008.

[6] P. Vassiliadis, "A survey of extract-transform-load technology," in *Integrations of Data Warehousing, Data Mining and Database Technologies*, D. Taniar and L. Chen, Eds. Information Science Reference, 2011, pp. 171–199.

[7] A. Y. Halevy, "Answering queries using views: A survey," *VLDB J.*, vol. 10, no. 4, pp. 270–294, 2001.

[8] K. Tomingas, M. Kliimask, and T. Tammet, "Data integration patterns for data warehouse automation," in *New Trends in Database and Information Systems II - Selected papers of the 18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events, ADBIS 2014 Ohrid, Macedonia, September 7-10, 2014 Proceedings II*, 2014, pp. 41–55.

[9] M. Masud and M. Rouached, "A web service based integration model of data-providing sources," in *Eighth International Conference on Digital Information Management (ICDIM 2013)*, 2013, pp. 320–325.

[10] W3C, *Web Service Description Language 1.1*, 2001. [Online]. Available: http://www.w3.org/TR/wsdl

[11] M. J. Hadley, "Web application description language (wadl)," Mountain View, CA, USA, Tech. Rep., 2006.

[12] J. Kopecký, K. Gomadam, and T. Vitvar, "hRESTS: An html microformat for describing restful web services," in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, 2008, pp. 619–625.

[13] M. Hansen, S. E. Madnick, and M. Siegel, "Data integration using web services," in *DIWeb*, Z. Lacroix, Ed. University of Toronto Press, 2002, pp. 3–16.

[14] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," in *2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013, Sardinia, Italy, July 1-5, 2013*. IEEE, 2013, pp. 323–328. [Online]. Available: http://dx.doi.org/10.1109/IWCMC.2013.6583580

[15] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber, "Active xml: Peer-to-peer data and web services integration," in *VLDB*. Morgan Kaufmann, 2002, pp. 1087–1090.

[16] M. Matera, "Web mashups," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer US, 2009, pp. 3482–3483. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-39940-9_5019

[17] D. Benslimane, S. Dustdar, and A. P. Sheth, "Services mashups: The new generation of web applications," *IEEE Internet Computing*, vol. 12, no. 5, 2008.

[18] J. D. Ullman, "Information integration using logical views," in *ICDT*, ser. Lecture Notes in Computer Science, F. N. Afrati and P. G. Kolaitis, Eds., vol. 1186. Springer, 1997, pp. 19–40.

[19] O. M. Duschka, M. R. Genesereth, and A. Y. Levy, "Recursive query plans for data integration," *J. Log. Program.*, vol. 43, no. 1, pp. 49–73, 2000.

[20] M. Friedman, A. Y. Levy, and T. D. Millstein, "Navigational plans for data integration," in *AAAI/IAAI*, J. Hendler and D. Subramanian, Eds. AAAI Press / The MIT Press, 1999, pp. 67–73.

[21] S. Abiteboul and O. M. Duschka, "Complexity of answering queries using materialized views," in *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 1998, pp. 254–263.

[22] M. R. Genesereth, A. M. Keller, and O. M. Duschka, "Infomaster: An information integration system," in *SIGMOD Conference*, J. Peckham, Ed. ACM Press, 1997, pp. 539–542.

[23] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "The tsimmis project: Integration of heterogeneous information sources," in *In Proceedings of IPSJ Conference*, 1994, pp. 7–18.

[24] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava, "The information manifold," in *In Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, 1995, pp. 85–91.

[25] Y. Papakonstantinou and V. Vassalos, "Architecture and implementation of an xquery-based information integration platform," *IEEE Data Eng. Bull.*, vol. 25, no. 1, pp. 18–26, 2002.

[26] M. Barhamgi, D. Benslimane, and A. M. Ouksel, "Composing and optimizing data providing web services," in *WWW*, J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, Eds. ACM, 2008, pp. 1141–1142.

[27] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[28] J. D. Ullman, "Information integration using logical views," *Theor. Comput. Sci.*, vol. 239, no. 2, pp. 189–210, 2000.

[29] R. Pottinger and A. Y. Levy, "A scalable algorithm for answering queries using views," in *VLDB*, A. El Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, Eds. Morgan Kaufmann, 2000, pp. 484–495.

[30] O. M. Duschka and M. R. Genesereth, "Answering recursive queries using views," in *PODS*, 1997, pp. 109–116.

[31] S. Thakkar, J. L. Ambite, and C. A. Knoblock, "Composing, optimizing, and executing plans for bioinformatics web services," *The VLDB Journal*, vol. 14, no. 3, pp. 330–353, 2005.

[32] J. Samuel and C. Rey, "Dawes: Data warehouse fed with web services," in *INFORSID*, 2014.

[33] M. Lanthaler and C. Guetl, "Hydra: A vocabulary for hypermedia-driven web apis," in *Proceedings of the WWW2013 Workshop on Linked Data on the Web, Rio de Janeiro, Brazil, 14 May, 2013*, 2013. [Online]. Available: http://ceur-ws.org/Vol-996/papers/ldow2013-paper-03.pdf

[34] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decades overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[35] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007.

[36] O. Terzidis, D. Oberle, A. Friesen, C. Janiesch, and A. Barros, *The Internet of Services and USDL*. Springer US, 2012, pp. 1–16.

[37] R. T. Fielding, "Architectural styles and the design of network-based software architectures," 2000.

[38] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[39] C. Rodríguez, M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella, *REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices*. Cham: Springer International Publishing, 2016, pp. 21–39.

[40] M. Maleshkova, C. Pedrinaci, J. Domingue, G. Alvaro, and I. Martinez, "Using semantics for automating the authentication of web apis," in *ISWC on The Semantic Web - Volume Part I*, 2010, pp. 534–549.

[41] A. Calì, D. Calvanese, and D. Martinenghi, "Dynamic query optimization under access limitations and dependencies," *J. UCS*, vol. 15, no. 1, pp. 33–62, 2009.

[42] F. N. Afrati, C. Li, and P. Mitra, "Answering queries using views with arithmetic comparisons," in *PODS*, L. Popa, S. Abiteboul, and P. G. Kolaitis, Eds. ACM, 2002, pp. 209–220.

[43] R. Yerneni, C. Li, H. Garcia-Molina, and J. D. Ullman, "Computing capabilities of mediators," in *SIGMOD Conference*, 1999, pp. 443–454.

[44] G. Grahne and V. Kiricenko, "Towards an algebraic theory of information integration," *Inf. Comput.*, vol. 194, no. 2, pp. 79–100, 2004.

[45] G. Grahne, "Horn tables - an efficient tool for handling incomplete information in databases," in *PODS*. ACM Press, 1989, pp. 75–82.

[46] A. Calì and D. Martinenghi, "Querying data under access limitations," in *ICDE*, 2008, pp. 50–59.

[47] C. Li and E. Y. Chang, "Query planning with limited source capabilities," in *ICDE*, 2000, pp. 401–412.

[48] M. T. Roth and P. M. Schwarz, "Don't scrap it, wrap it! a wrapper architecture for legacy data sources," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 266–275. [Online]. Available: http://dl.acm.org/citation.cfm?id=645923.670992

[49] J. Samuel and C. Rey, "Generic web service wrapper for mediation based data warehousing," in *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '16. ACM, 2016, pp. 34:1–34:4.

[50] J. Kim, P. Thomas, R. Sankaranarayana, T. Gedeon, and H. Yoon, "Pagination versus scrolling in mobile web search," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*. ACM, 2016, pp. 751–760. [Online]. Available: http://doi.acm.org/10.1145/2983323.2983720

[51] J. Li, Y. Xiong, X. Liu, and L. Zhang, "How does web service API evolution affect clients?" in *2013 IEEE 20th International Conference on Web Services*, 2013, pp. 300–307.

[52] J. Samuel, "Feeding a data warehouse with data coming from web services. A mediation approach for the dawes prototype." Ph.D. dissertation, Blaise Pascal University, Clermont-Ferrand, France, 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01086964

[53] U. Zdun, M. Völter, and M. Kircher, "Design and implementation of an asynchronous invocation framework for web services," in *Web Services - ICWS-Europe 2003, International Conference ICWS-Europe 2003, Erfurt, Germany, September 23-24, 2003, Proceedings*, ser. Lecture Notes in Computer Science, M. Jeckle and L. Zhang, Eds., vol. 2853. Springer, 2003, pp. 64–78. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-39872-1_6