MIDAS: A Middleware to Provide Interoperability between SaaS and DaaS

Tarcio Marinho, Vinicius Cidreira, Daniela Barreiro Claro, Babacar Mane
FORMAS - Grupo de Pesquisa em Formalismos e Aplicações Semânticas
Depto de Ciência da Computação – Instituto de Matemática – Universidade Federal da Bahia
Av. Adhemar de Barros, s/n, Ondina
Salvador – Bahia – Brasil 40170-110
tmmachado@dcc.ufba.br, vmcidreira@dcc.ufba.br, dclaro@ufba.br,
mbabacar@gmail.com

ABSTRACT

Software as a Service (SaaS) and Data as a Service (DaaS) proves to be two promising areas of research in the cloud computing field, however interoperability among different cloud providers is yet poorly explored. Today, clients looking for content or services from different providers need extra time and resources to learn and implement the required adaptations from the other parties. In this paper we propose MIDAS, a novel middleware to interoperate SaaS and DaaS services seamlessly and independently from provider. That is, SaaS applications will be able to get data from DaaS datasets by sending a query to our middleware and letting it mediate the communication and return the expected results. We evaluate our proposal by developing a prototype from two case studies and by analyzing the time effort to query through our middleware. Our results presented that no important overhead were required from providers nor to the final user.

Categories and Subject Descriptors

H.3.2 [**Information systems**]: Data management systems—*Information integration*

General Terms

Information Systems

Keywords

Cloud Interoperability, Middleware, DaaS, SaaS

1. INTRODUCTION

The amount of data produced around the world is doubling every two years and is estimated to achieve 7.2 zetabytes for 2016 [8]. At the same time, researchers have been facing challenges with respect to data management. There is an important need for computer scientists to develop low cost solutions to huge intensive data issues. At the same time, there is a strong tendence among governments, intitutions and companies to make public information. Typically, such data is released as open data that may be freely used and distributed. However it is not enough to distribute the data, it is important to make them available in an understandable manner and ready for use by consumers. Thus provide services for data is an envisioned mechanism to tackle with this concern.

Cloud based services have been experiencing a rapid expansion. Some cloud service approaches [30] are Platform as a Service (PaaS), which provides operating systems as a service, Infrastructure as a Service (IaaS), which provides server machines as a service, Software as a Service (SaaS), which supplies cloud applications and Data as a Service (DaaS), a novel paradigm for providing data independent from application. It is noteworthy that Data as a Service(DaaS) and Database as a Service (DbaaS) are two distinct services. The latter, often misdefined as DaaS, refers to the offer of a conventional database such as Oracle or MongoDB, but offered on demand and hosted in a cloud. The focus of Data as a Service is in the data set itself, as tables in a relational database, so that only the reading of data is allowed. In this work we deeply explain DaaS and SaaS because of their participation in our proposal.

Software as a Service (SaaS) [2] comes as a new paradigm not only for running applications in the cloud, but it has also been changing concerns about how to manage data application in the cloud and how to improve its use for better performance and scalability. In this new pay-per-use-basis model both sides benefits: the application user, since he/she will be able to access its application anywhere in the world directly from the web; and the companies who will now transfer its costs of application/database maintainer to a third party service provider.

Data as a Service (DaaS) [30] provides data on demand. Data provided by DaaS is usually provided through APIs for the consumer to query on it or, in a few cases, to download it. As a result of using DaaS, consumers do not need to maintain large availability. Instead, they find a suitable provider that owns a dataset having the desired information. On the other hand, there is a call among governments, institutions and enterprises to make information publicly available, which contributes for an increasing demand for DaaS providers and a fast expansion of data assets available.

SBSI 2016, May 17th-20th, 2016, Florianópolis, SC, Brazil

In order to better understand both new issues, we have developed an extensive research focusing on SaaS and DaaS. As a result, we built two bubble charts summarizing our discoveries, one about SaaS and the other about DaaS. Further analysis led us to perceive a gap regarding the interoperability between those two paradigms, that is, there were not enough information describing how would be possible to a SaaS application hosted on one provider to access data from a DaaS provider. In order to accomplish this interoperability issue we propose the model and implementation of our MIDAS (Middleware for DaaS and SaaS) middleware.

MIDAS is responsible for mediating the communication between different SaaS and DaaS providers, making possible that SaaS applications retrieve data seamlessly from a DaaS datasource as it would be querying its own datacenter. We presented a case study concerning two different DaaS datasources that communicate with SaaS application. We measured both experiments to analyze their overload and performance. Our results present some measures which encourage the use of MIDAS to interoperate SaaS and DaaS providers.

This paper is organized as follows. Section 2 brings a literature survey that motivated this paper and we expose two bubble charts summarizing SaaS and DaaS research directions. In Section 3 we describe our middleware proposal. Section 4 presents a case study of a prototyped scenario employing the MIDAS middleware in real world application to evaluate our proposal. In Section 5 we discuss our results and, in Section 6, the conclusion and future work.

2. LITERATURE SURVEY

We have conducted an extensive research covering Software as a Service (SaaS) and Data as a Service (DaaS) areas with the objective of gathering the most relevant papers discussing data cloud related themes, which was our basic inclusion criterion. The final set of papers contains twelve articles regarding SaaS and other twelve regarding DaaS.

For the SaaS research our exclusion criteria discarded articles focusing on other aspects such as security or the application itself since our main goal was really to perceive data concerns. Papers which did not present any related work were also discarded. The remaining twelve papers [5, 10, 12, 14–16, 18, 20, 23, 28, 29, 31] were mapped into the bubble chart in Figure 1. The chart presents two readings of the results. On the left side, the contributions to each subarea (Framework, Method, Tool, Architecture and Model) and, on the right, the form of validation they used (Empirical Evaluation, Use case, Benchmark, Controlled experiment, Prototype). Bubbles grow up in accordance as the number of papers. Looking at the chart we can see that two papers did not provide any kind of validation. Also none of them proposed a way for an application to access data from a different provider, which is perceived by the empty *Interoperability* line in the chart. The major contribution observed on the chart is regarding a Method approach. Considering Validation reading, the major approach is by the use of a Controlled Experiment. Regarding Migration research context, it is observable that only Use Case approach is being used for validation.

For the DaaS research [1,3,4,7,11,13,17,19,22,25–27], papers were discarded when they were focused on security or structural aspects of the service. As our focus where aimed at data aspects of the service, we had to maintain twelve papers that were mapped into the bubble chart in Figure 2. The chart is organized following the similar structure used in SaaS chart. So, observing the chart it is possible to see that just one paper did not explain the method they used to validate their proposal. As it happened in the SaaS research, there was no paper proposing a way to work with dataset from different providers through a SaaS application, which also can be perceived looking to the empty *Interoperability* line in the chart. The major contribution observed on the chart is regarding the Architecture approach, however the difference between Architecture and Model is not too significative, thus both can be considered as a Contribution to DaaS area. Considering Validation readings, the major approach is by the use of Case Study approach. Thus, we opted to either validate our approach by a case study.

Considering both bubble charts, we can observe a lack in interoperability issues considering SaaS and DaaS approaches. Thus, our solution aims at developing a method to interoperate SaaS applications and DaaS datasources. According to IEEE, interoperabiliy means the hability of two or more systems to exchange information and use them [9]. Since the ISO [24] defined as the ability of functional units run programs, communicate and transfer data between them as to require minimal user knowledge about the unique features of these units. On the other hand, researchers define interoperability as the ability of software systems to understand and use the services one from the other [6]. Furthermore, in the context of SOA, it is defined as software systems ability to use the services of each other [21]. Finally, within the cloud computing context, interoperability is defined as the ability to write code that work in more than one service provider, regardless of the differences between them, i.e. correctly recovering data from different sources. The following sections describe our proposal and our case study.

3. MIDAS MIDDLEWARE FOR CLOUD INTEROPERABILITY

This paper is the result of an intensive endeavour to propose a method so an application hosted on a Software as a Service (SaaS) provider could not only retrieve data from its own data center, but also from an external datasource without extra effort. In sight of this goal, we propose a novel middleware, called MIDAS (Middleware for DaaS and SaaS), that commits to deliver a transparent and provider independent communication between different SaaS and DaaS providers.

The challenge is that as of today the only way to accomplish this interoperability is by doing it manually. One possibility is to acquire datasets from the DaaS providers and then populate the database manually or maybe make a query to a file. The problems here are (i) data will not be always up-to-date and (ii) the SaaS provider would have to agree in adapting to these procedures, which we assume is unlikely. Another choice would be to access the DaaS provided API directly from the application, but again, that would implicate in having the SaaS provider to adapt its application to a new method of retrieving data.

Our solution is a middleware service that will transparently mediate the communication between any SaaS and DaaS providers. The core idea is that the SaaS application will not have to adapt anyway more than it already does. For example, normally the application requests data from

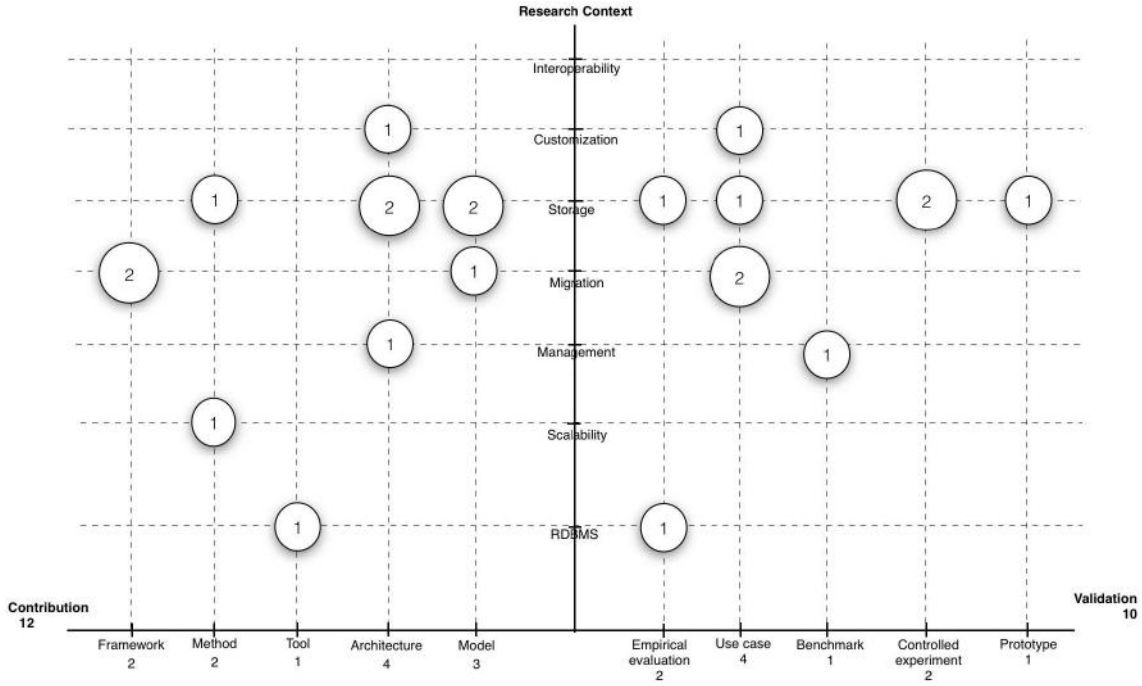

Figura 1: Bubble chart for SaaS

its server's databases and get a result set as a response to format and display on the screen. Now, the only thing that will change is the application sending the database query (regardless the language) to our middleware and it will be in charge of querying the DaaS dataset and returning the result set as expected.

MIDAS'architecture is depicted in Figure 3. The middleware is composed of two modules: Request and Result. As the names suggest, Request Module handles the incoming SaaS query while Result Module handles DaaS response.

The following subsections describe the whole architecture in detail.

3.1 SaaS Application

The SaaS application developers will benefit from MIDAS since a minimal extra effort is required to adapt it to the middleware. The SaaS query is sent to MIDAS and then our middleware takes the control.

3.2 MIDAS Middleware

MIDAS (Middleware for DaaS and SaaS) aims at providing a transparent communication between different SaaS and DaaS providers, making possible that SaaS applications retrieve data from a DaaS datasource. Our middleware is composed of four modules: *Query Decomposer*, *Data Provider Information Storage (DIS)*, *Query Builder* and *Result Formatter*. Figure 4 describes the sequence of each module of MIDAS. It is possible to understand the interactions among SaaS application, MIDAS, DaaS provider and the user. From the point of view of the user, the same query is sent and a required result is returned. Thus, MIDAS works in a transparent manner.

Each module is described in following paragraphs.

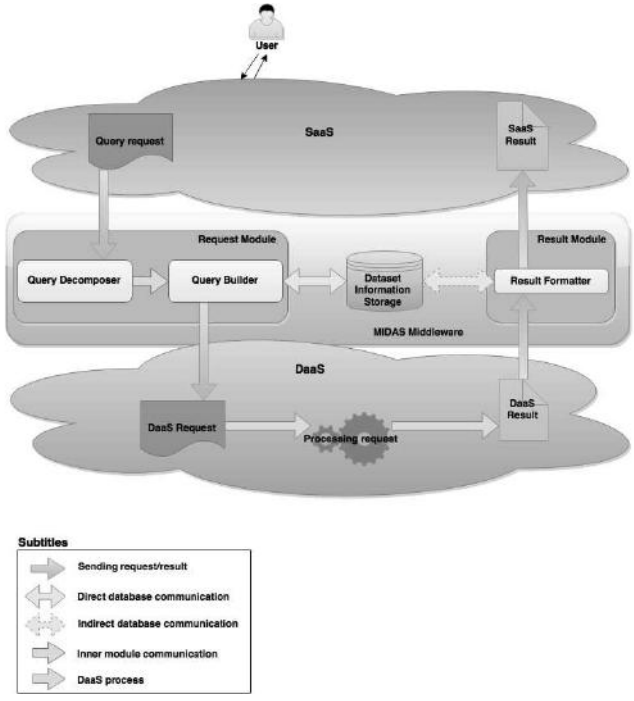

Figura 3: MIDAS middleware model

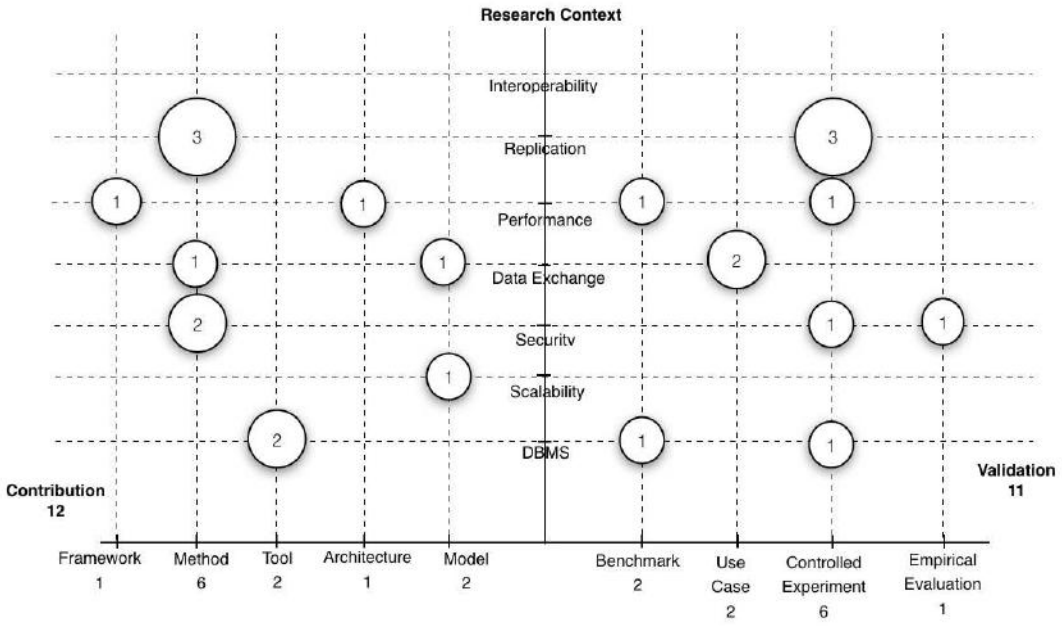

Figura 2: Bubble chart for DaaS

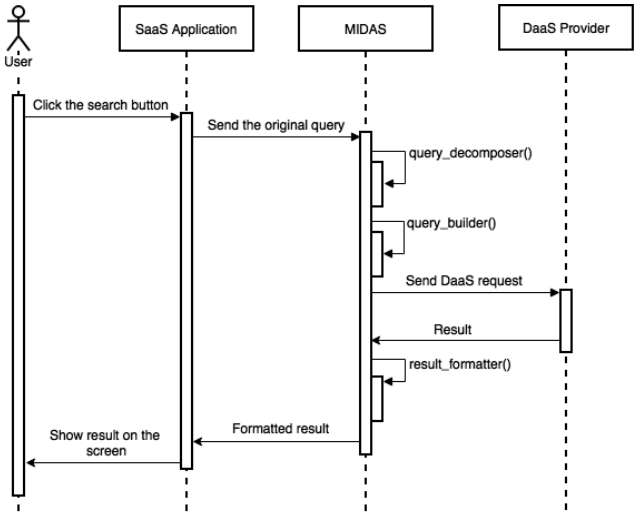

Figura 4: Sequence Diagram of MIDAS

3.2.1 Query Decomposer

Query Decomposer is responsible for breaking the original query into an array, which maps each part of the query to an independent key-value format. This format is more interoperable and will assure that the *Query Builder* will be able to build the DaaS request independently of their API.

3.2.2 Dataset Information Storage (DIS)

Dataset Information Storage (DIS) aims to keep information about DaaS providers' datasets and APIs. In order to persist it, we proposed the use of a relational database. The Logical Database Diagram for DIS is depicted in Figure 5 within three tables. Since there is yet no way to get this data automatically, the job of adding providers to the storage and maintaining them will have to be done manually for now.

- **Provider(id, name, domain, id_api)**. In this table we keep only essential data about the DaaS provider, which are **name** and **domain**. The **domain** is the first part of the DaaS request URL built by the *Query Builder*. The attribute **id_api** is a foreign key to the API by the provider and the **id** is a numeric sequence.

- **Dataset(id, id_provider)**. The attribute **id** is a name that uniquely identifies the dataset among the providers. Because of that we are able to find its respective provider and API, which affords the SaaS application to work with nothing more than the dataset **id** as a targeted table.

- **Api(id, search_path, dataset_param, query_param, sort_param, limit_param)**. The attribute **id** here is a numeric sequence as in the Provider relation. The attribute **search_path** represents the piece of string in

the DaaS request that concatenates to the provider domain and gives the path to access the API. The attributes **dataset_param**, **query_param**, **sort_param** and **limit_param** represent respectively the API parameter names to inform dataset, query filters, ordering field and the number of rows in the result.

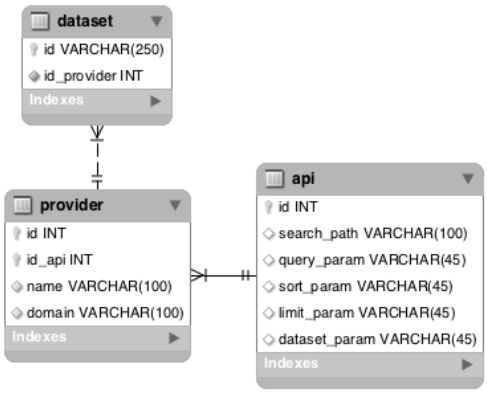

Figura 5: DIS Entity Relationship Diagram

3.2.3 Query Builder

After receiving the decomposed query, the *Query Builder* will proceed with the assembling process of transforming the SaaS query in a DaaS request. For this matter, the *Query Builder* search the *Dataset Information Storage (DIS)* in order to get information about the provider that owns the dataset being queried (Figure 5). With this information, the *Query Builder* will follow-up matching the fields in our independent format with the parameters that compose the DaaS provider request using the column names in the *Dataset Information Storage* as a reference to the fields as its correspondent in the DaaS API parameters. After finishing the parameters matching, the *Query Builder* will then compose the complete request parameter and send it to the DaaS provider.

3.2.4 Result Formatter

The data received from the DaaS provider comes in a JSON object packed along with other information that will not be used by the SaaS application that initially requested the information. For this reason, the *Result Formatter* is responsible to prune the returned object keeping just the relevant information that needs to be sent to the SaaS application.

3.3 DaaS Provider

The DaaS provider does not have to make any changes in its service offering. Thus, the provider will receive the request sent by MIDAS and proceed with the processing within the dataset. After gathering all the data requested, the provider will return all the information to MIDAS as a JSON object.

4. CASE STUDY

Our case study aims to validate our MIDAS approach. It illustrates a tourism agency web application that wants to host their application in the Software as a Service (SaaS) Red Hat cloud. Though the cloud provider offers the storage space of a SQL data center, their interest is to assemble data from different Data as a Service (DaaS) providers to build their application.

In the current prototype, MIDAS is able to understand simple SQL Standard queries, containing *Select*, *From*, *Where*, *Order By* and *Limit* clauses. However, it is implemented in a way that will easily incorporate new query languages in their full potential as suited. In other words, the application will send the query and wait for the result as the SQL Standard database would return it.

For our case study we have prototyped a web application that concerns displaying two different data sources: a list of Wi-Fi hotspots (*OpenDataSoft2* dataset) and a list of hotels in the New York City metropolitan region (*NYC OpenData3*) from OpenDataSoft's public service[1]. Our web application is hosted in the Red Hat SaaS public cloud[2]. The application is based on HTML5 + CSS3 + JavaScript front end along with a PHP 5.4 and MySQL 5.5 back end. Bootstrap and jQuery were also used additionally and the framework *Codeigniter 2* was employed for the PHP implementation.

To better illustrate the scenario, we will assume for now on Code 1 is the original SaaS query.

Code 1. Original SaaS query in SQL language

```
SELECT id, name, address, city
FROM nyc-wifi-hotspot-locations
WHERE city = 'New York'
ORDER BY id
LIMIT 10
```

As we can see in Code 2, the method takes the query as a parameter and creates an array called $indexes, which will collect the first index of each clause of the SQL Standard syntax that is acceptable for now. They are: 'from', 'where', 'order by' and 'limit'. The clause 'select' is assumed as index 0 since it has to head the query in SQL. Next, we store the query columns, the ones between 'select' and 'from' in the query, as an array inside our independent format array in the position 'fields'. The 'dataset' key relates to the value passed in the 'from' clause, this will be the dataset **id** the DaaS provider will look for. This means the SaaS application will query the dataset as it would be querying a relational table. The 'filters' key maps the 'where' clause value, 'order' maps 'order by's and the 'limit' key maps the 'limit' clause.

Code 2. Query decomposer' indexes structure

```
private function _query_decomposer($query){
    $sql = strtolower(trim($query));

    $indexes[] = strpos($sql, "from");
    $indexes[] = strpos($sql, "where");
    $indexes[] = strpos($sql, "order by");
    $indexes[] = strpos($sql, "limit");

    $jsonArray["fields"] = $this->_getFields($sql,
        $indexes[0]-6);
    $jsonArray["dataset"] = $this->_getDataset($sql,
        $indexes[0]+4, $indexes, 1);
```

[1] http://public.opendatasoft.com/explore/
[2] https://www.openshift.com

```
    $jsonArray["filters"] = $indexes[1] === false ?
         false : $this->_getFilters($sql, $indexes[1]+5,
         $indexes, 2);
    $jsonArray["order"] = $indexes[2] === false ? false
         : $this->_getFilters($sql, $indexes[2]+8,
         $indexes, 3);
    $jsonArray["limit"] = $indexes[3] === false ? false
         : $this->_getLimit($sql, $indexes[3]+5);

    return $jsonArray;
  }
```

After being processed by our middleware's Request Module, the original query will be turned in a DaaS request. Information about DaaS requests had already been stored in DIS. Each DaaS provider needs to store its relevant information manually, since until now there is no way to obtain it automatically. A brief example about DaaS information stored in DIS can be seen as follows. This example was used in our case study.

http://public.opendatasoft.com/api/records/1.0/search?dataset=nyc-wifi-hotspot-locations&q=city+%3D+'New+York'&rows=10&sort=id

Then, the request is processed by the DaaS service and a JSON result is generated. Because the object returned is too large, we only present its first record in Code 3. This result is returned to the middleware where it will be handled by the Result Module.

Code 3. JSON result from DaaS

```
{"nhits": 712, "parameters": {"dataset": ["nyc-wifi-
     hotspot-locations"], "timezone": "UTC", "q": "city
     = 'New York'", "rows": 10, "sort": "id", "format":
     "json"}, "records": [{"datasetid": "nyc-wifi-
     hotspot-locations", "recordid": "
     a4e6f313395a4eab66019164ea30b18a7103ca6a", "fields
     ": {"city": "New York", "name": "Metropolitan
     Museum of Art", "zip": "10028", "url": "http://www.
     metmuseum.org/", "geom_x_y": [40.779426050712786,
     -73.96343466448646], "geom": {"type": "Point", "
     coordinates": [-73.96343466448646,
     40.779426050712786]}, "address": "1000 Fifth Avenue
     ", "type": "Fee-based", "id": 1359.0}, "geometry":
     {"type": "Point", "coordinates":
     [-73.96343466448646, 40.779426050712786]}, "
     record_timestamp": "2015-04-22T22:08:01+00:00"}]}
```

As we can see in Code 3, the JSON object provided by the DaaS service contains too much information that does not concern the SaaS application, so when it specifies the name of the attributes within the SELECT parameter in the query, the Result Formatter will remove the unwanted attributes from the JSON object. To do that pruning, the Result Formatter retrieves, from the DIS, the name of the field into the object that contains the target information. After doing this, the Result Formatter will turn the returned object in another JSON object but following the structure of a result set as would be returned from SQL Database, which means that it holds both the numeric and associative keys containing the information from the dataset (Code 4).

Code 4. JSON from Result Formatter

```
[{"0":"New York","city":"New York","1":"Metropolitan
     Museum of Art","name":"Metropolitan Museum of Art
     ","2":"1000 Fifth Avenue","address":"1000 Fifth
     Avenue","3":1359,"id":1359}]
```

Now that the result is in the SaaS expected format it is returned as a response to the original query. All steps can be better visualized in Figure 6, where each step (1-5) is described concerning its respective module in MIDAS.

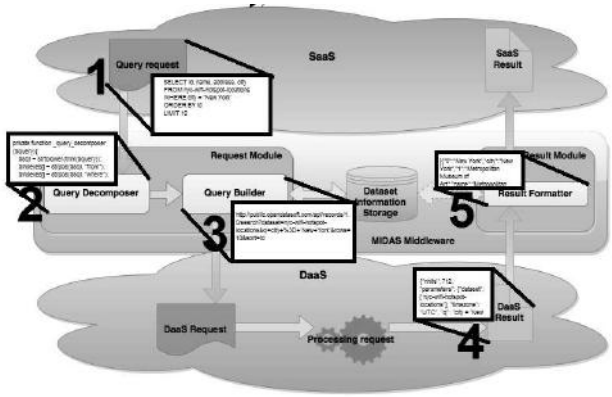

Figura 6: Steps of MIDAS

Is it important to observe that our MIDAS provides interoperability approach, following step-by-step as proposed in our model. The first step depicts the query request described in our case study as SQL statement. The second step corresponds to the indexes in Query Decomposer where each clause in query request is decomposed into key value approach. The third step concerns the communication between the Query Builder and DIS. The fourth step is a pure DaaS result and the fifth step is actually the DaaS result formatted as following the SaaS application's request.

5. EVALUATION

Our evaluation was made based on the comparison between the direct access to a datasource and access through MIDAS through all steps since the initial query. For that we used a tool called Hurl[3] to test HTTP requests.

As a first test, a set of 40 requests were sent through MIDAS from the SaaS application using both DaaS Providers: NYC Open Data and OpenDataSoft. The second test was made by the direct access to both datasources, where either 40 requests were sent to each datasource provider. As a result, we obtained an average of 228.24 ms on the direct access to NYC OpenData and 286.88ms by the use though our middleware MIDAS. The difference between the direct access and MIDAS was 58.64ms.

Considering the second data source, OpenDataSoft, the average response time within direct access was 370.08ms, while using MIDAS was 450.04ms. Thus the difference between direct access and MIDAS was 79.96ms. The time difference between the datasets was due to the number of fields and data returned. The OpenDataSoft DaaS has a larger number of fields and data than NYC Open Data.

Analyzing the result times obtained, a low dispersion (LS - LI) on query returning through MIDAS is observed. In both providers there is a negative asymmetry in delay time,

[3]https://www.hurl.it/about

thus finding not very high values for returning time using our middleware. In the case of NYC OpenData, the turnaround time via MIDAS was within the range between $Q3$ and LS boxplot on comparison with direct access to provider. Thus the overtime for accessing through MIDAS does not reflect into performance. It was observable that the JSON object returned has a large number of fields and data, requiring more operations in our Formatter Result of MIDAS. This conclusion comes from the fact that there is an intersection between the graphic access via MIDAS and direct access at the point that goes between the Q1 and the LI of the first and the LS and the second of Q3.

In addition, the median (Med) was analyzed for access via MIDAS and direct access to NYC OpenData whose results were $277ms$ and $215ms$ respectively. For OpenDataSoft the results are $438ms$ and $345ms$, respectively. Finally, the turnaround time to query via MIDAS was between 20% and 25% for both providers, which is offset by the benefit provided by use of our middleware, since it adds the ability to query different data sources. It is worth noting that this time also includes the access time for a provider that hosts our middleware, which could increase the total return time.

6. DISCUSSION

With our case study we were able validate the effectiveness of MIDAS through a step-by-step explanation of how we achieved the interoperability between our SaaS application and both DaaS datasets. We chose those providers for their reputation with cloud services and the availability of a public service, which enabled the implementation and hosting of our current prototype.

Concerning time efforts, we could verify that no response overtime was perceived from the users' perspective. Search results were displayed on the screen in approximately one second from the instant the search button is clicked. As the best of our knowledge, there is no similar proposal covering a middleware implementation and its evaluation to interoperates SaaS and DaaS.

6.1 Threats of Validity

We can position some threats of validity. Concerning our case study we have application and middleware running in the same domain, which will not be the case for real world use. In those cases, the middleware would run on an independent cloud service. However this does not diminish our proposal as our case study can be located abroad.

Another threat can be the use of a public dataset. One of the advantages of choosing a public dataset is that we did not need to worry about authentication concerns, nevertheless this also means that our current implementation does not cover authentication issues in protected domains. In that case, the SaaS provider would have to send its authentication key along with the query so it can be validated by the DaaS service. Despite that, it will not interfere with the current functionality.

7. CONCLUSION AND FUTURE WORK

In this paper we proposed a novel middleware called MIDAS that enables the interoperability between Software as a Service (SaaS) and Data as a Service (DaaS) cloud providers. With MIDAS, SaaS applications will be able to query DaaS datasets transparently as it would be querying its own databases, which means a minimal adaptation for SaaS and DaaS providers. Results have shown that our middleware was able to deliver the expected results to the application with a low overtime, from the moment the user clicks on the search button to the moment the result table is filled. This shows how our project can encourage the conception of a middleware in despite of the cross-provider communication.

In future work we intend to address issues left open in this paper, such as offering a more robust support to SQL Standard queries and authentication concerns for protected domains. We also plan to build a new case study demonstrating the middleware operation with a non-relational-database application.

ACKNOWLEDGMENTS

Babacar Mane would like to thanks FAPESB BOL3693/2014.

8. REFERÊNCIAS

[1] S. Ahirrao and R. Ingle. Scalable transactions in cloud data stores. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 116–119. IEEE, 2013.

[2] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1195–1206. ACM, 2008.

[3] S. Barouti, D. Alhadidi, and M. Debbabi. Symmetrically-private database search in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 671–678. IEEE, 2013.

[4] N. Bonvin, T. G. Papaioannou, and K. Aberer. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 205–216. ACM, 2010.

[5] E. Boytsov and V. Sokolov. Multi-tenant database clusters for saas. *proceedings of BMSD*, page 144, 2012.

[6] D. Chen, G. Doumeingts, and F. Vernadat. Architectures for enterprise integration and interoperability: Past, present and future. *Comput. Ind.*, 59(7):647–659, Sept. 2008.

[7] G. Chen, H. T. Vo, S. Wu, B. C. Ooi, and M. T. Özsu. A framework for supporting dbms-like indexes in the cloud. *Proceedings of the VLDB Endowment*, 4(11):702–713, 2011.

[8] J. GANTZ and D. REINSEL. Extracting value from chaos. (1142):9–10, 2011.

[9] A. Geraci. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. IEEE Press, Piscataway, NJ, USA, 1991.

[10] I. Gorti, N. Shiri, and T. Radhakrishnan. A flexible data model for multi-tenant databases for software as a service. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pages 1059–1066. IEEE, 2013.

[11] J. Han, M. Song, and J. Song. A novel solution of distributed memory nosql database for cloud computing. In *Computer and Information Science (ICIS), 2011 IEEE/ACIS 10th International Conference on*, pages 351–355. IEEE, 2011.

[12] D. Hou, S. Zhang, and L. Kong. Placement of saas cloud data and dynamically access scheduling strategy. In *Computer Science & Education (ICCSE), 2013 8th International Conference on*, pages 834–838. IEEE, 2013.

[13] M.-J. Hsieh, C.-R. Chang, L.-Y. Ho, J.-J. Wu, and P. Liu. Sqlmr: A scalable database management system for cloud computing. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 315–324. IEEE, 2011.

[14] M. Hui, D. Jiang, G. Li, and Y. Zhou. Supporting database applications as a service. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 832–843. IEEE, 2009.

[15] L. Jiang, J. Cao, P. Li, and Q. Zhu. A mixed multi-tenancy data model and its migration approach for the saas application. In *Services Computing Conference (APSCC), 2012 IEEE Asia-Pacific*, pages 295–300. IEEE, 2012.

[16] K. Lanju, L. Qingzhong, and W. Xue. Multi-level index model for saas application. In *Web Information System and Application Conference (WISA), 2013 10th*, pages 23–28. IEEE, 2013.

[17] Y. Li, L. Guo, and Y. Guo. Cacss: Towards a generic cloud storage service. In *CLOSER*, pages 27–36, 2012.

[18] H. Liu and S. Wu. Data storage schema upgrade via metadata evolution in saas. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 3148–3151. IEEE, 2012.

[19] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In *Data and Applications Security XX*, pages 89–103. Springer, 2006.

[20] J. Ni, G. Li, J. Zhang, L. Li, and J. Feng. Adapt: adaptive database schema design for multi-tenant applications. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2199–2203. ACM, 2012.

[21] S. Pokraev. *Model-Driven Semantic Integration of Service-Oriented Applications*. PhD thesis, University of Twente, October 2009.

[22] A. M. Segura, J. S. Cuadrado, and J. D. Lara. Odaas: Towards the model-driven engineering of open data applications as data services. In *Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International*, pages 335–339. IEEE, 2014.

[23] W. Shengqi, Z. Shidong, and K. Lanju. A dynamic data storage architecture for saas. In *Multimedia Information Networking and Security (MINES), 2010 International Conference on*, pages 292–296. IEEE, 2010.

[24] M. J. Standard. *Information Technology - Vocabulary - Part 1: Fundamental Terms (ISO/IEC 2382-1:1993, IDT)*. Malaysian standard. Department of Standards Malaysia, 2005.

[25] O. Terzo, P. Ruiu, E. Bucci, and F. Xhafa. Data as a service (daas) for sharing and processing of large data collections in the cloud. In *Complex, Intelligent, and Software Intensive Systems (CISIS), 2013 Seventh International Conference on*, pages 475–480. IEEE, 2013.

[26] H.-L. Truong, S. Dustdar, J. Götze, T. Fleuren, P. Müller, S.-E. Tbahriti, M. Mrissa, and C. Ghedira. Exchanging data agreements in the daas model. In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 153–160. IEEE, 2011.

[27] Q. H. Vu, T.-V. Pham, H.-L. Truong, S. Dustdar, and R. Asal. Demods: A description model for data-as-a-service. In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pages 605–612. IEEE, 2012.

[28] C. Weiliang, Z. Shidong, and K. Lanju. A multiple sparse tables approach for multi-tenant data storage in saas. In *Industrial and Information Systems (IIS), 2010 2nd International Conference on*, volume 1, pages 413–416. IEEE, 2010.

[29] Z. Xuxu, L. Qingzhong, and K. Lanju. A data storage architecture supporting multi-level customization for saas. In *Web Information Systems and Applications Conference (WISA), 2010 7th*, pages 106–109. IEEE, 2010.

[30] M. Zhou, R. Zhang, D. Zeng, and W. Qian. Services in the cloud computing era: A survey. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 40–46. IEEE, 2010.

[31] X. Zhou, D. Zhan, L. Nie, F. Meng, and X. Xu. Suitable database development framework for business component migration in saas multi-tenant model. In *Service Sciences (ICSS), 2013 International Conference on*, pages 90–95. IEEE, 2013.