Towards DaaS 2.0: Enriching Data Models

Jonathan Martí, Daniel Gasull, Anna Queralt
Barcelona Supercomputing Center
Spain
{jonathan.marti, daniel.gasull, anna.queralt}@bsc.es

Toni Cortes
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya - BarcelonaTech
Spain
toni.cortes@bsc.es

Abstract—**Current Data as a Service solutions present a lack of flexibility in terms of allowing users to customize the underlying data models by including new concepts or functionalities. Data providers either publish global APIs to make data available, or "sell" and transfer data to clients so they can do whatever they want with it. Thereby, collaboration and B2B becomes limited and sometimes is not even feasible. Our technology implements the necessary mechanisms for data providers to enable their clients to enrich data models both with additional concepts and with new methods that can be executed and, in turn, published as new services.**

Keywords—DaaS, Cloud, Cloud Storage, Data enrichment

I. INTRODUCTION

The acronym DaaS (Data as a Service) was coined to describe a model for the on-demand data management services in the context of the "as a Service" (aaS) stack [1]. DaaS is based on the concept that the product, data in this case, can be provided on demand to the user regardless of geographic or organizational separation of provider (Data Provider in this case) and consumer.

With current DaaS solutions one can store any contents according to a data model defined by the data provider. Data providers also offer a set of global APIs to enable their customers to access, download, or upload the data based on CRUD (Create, Read, Update, Delete) commands.

Google Maps is one of the most relevant examples of DaaS and we will use it to show the limitations of current approaches and what could be gained with the proposed extensions. Besides the basic downloading of maps, Google Maps enables users to create personalized maps by adding icons, or shapes for defining areas, among others, which can then be shared with other users. Although this is one example of the most flexible DaaSs in the Web today, it is still a service based on CRUD commands.

These DaaSs could be further improved by offering not only CRUD commands on data but also the possibility to let third parties to enrich the data models themselves with their own concepts and computations.

If Google Maps could share its data model and offer a mechanism for granting third-parties to enrich it with new concepts, a third party could create a new data model to represent routes with nice point of sightseeing interests and places to eat and sleep. This new data model could be added to the one offered by Google Maps and then build a service on top of it where you can search for routes following several conditions, such as routes where we can find vegan restaurants.

If we focus on adding new code, we could have a real estate company that would like to create a dynamic overlay with a gradient presenting the prices per square meter of the houses being on sale. This would need to be computed for every query depending on the current houses on sale.

Both options can be done today if you download Google Maps information to your own infrastructure, and then enrich it at your side. What would be desirable is to enable such enrichments without having to copy any data, thus enabling the new model and new objects (routes) to be resident in Google maps infrastructure (as happens today with icons and points of interest). In a similar way, it would also be desirable that the computation of the gradient could be done in Google's infrastructure. In both cases, unnecessary data movements would be avoided, and the enrichments and computations would always be made using up-to-date data.

Considering the current scenario, we can add value to current DaaS solutions by introducing capabilities that enable to enrich and share not only the data but also the data models themselves and the computation to manipulate the data. This solution increases B2B and collaboration opportunity in a win-win manner. On the one hand, new players can create new services with very little investments. And on the other hand, data providers will see a higher utilization of their data, and thus get greater benefits according to their business model.

In this paper we present the design and implementation of a new platform that enables data providers to offer their data so that third parties can enrich it in the same infrastructure. In particular, by enrichment we mean:

- Extending the original data model by adding new concepts designed by third parties.
- Extending the functionality by adding new code developed by third parties.

In section II we will comment some of the current DaaS solutions by means of some well-known examples. In section III we justify the chosen programming paradigm. In section IV we will introduce the key aspects and specific concepts of our technology. In section V we will explain some implementation details and some tricky aspects. Finally we will give our conclusions.

II. RELATED WORK

Today, the mechanisms to enable third parties to enrich both data and functionalities in the data provider's infrastructure are very basic. Third parties can add data into the original infrastructure through a data service [2], or, in some cases, add very limited functionality such as custom overlays in Google maps.

Moreover, current DaaS Servers approaches that aim to enhance data providers' experience when offering DaaS, still do not seem concerned about easing that data providers enable third parties to enrich their data model and functionalities in the same way as we propose.

A. DaaS until today

One of the most representative data sharing services is Google Maps, already mentioned in the introduction. Although the Google Maps API allows to add custom overlays that require to perform some kind of computation, they are not completely arbitrary and remain in the scope of the application computing them. On the contrary, Google Map Maker allows enriching maps with geographical data that ends up becoming part of the maps and available to the general public once it has been revised, but the data to be added must conform to a limited set of geographical items (i.e. a specific data model).

At the end, the common limitations of DaaS (until today) can be extrapolated from those present in Google Maps. With current data services, third parties are not able to:

- Extend the data set in a way not envisioned by the data provider, both by adding new concepts to the data model and new functionalities.
- Store these extensions in the data provider infrastructure so that they can be accessed by other client applications.

B. DaaS Servers

Not only current DaaS services present the limitations commented previously, but also DaaS servers do not provide the necessary mechanisms to enable data providers to offer DaaS with the dimension that we propose in this paper.

DaaS Servers facilitate data providers to offer DaaS by giving them lots of mechanisms like an all-in-one Cloud solution that joins: Cloud Databases, Cloud Storage, integration tools or security, (among others). For instance, Infopar Qualitta DaaS Server [3] provides an infrastructure based on Amazon S3 (to store data) and Amazon EC2 (to manage the computation of the services) that enables customers of Infopar to offer DaaS. Customers can also share their catalogue of data models and the databases (the actual data, information) so different Infopar accounts can enrich each other. However, Infopar does not offer the proper capabilities to enable data providers to be enriched from external third parties.

Another example is WSO2 Data Services Server [4], which provides a platform for integrating data stores, creating composite data views, and hosting data services. WSO2 enables Data Providers to combine data from multiple data sources in a single resource, allows server customization via feature provisioning of any middleware capability, or even offers a tool for automatic generation of CRUD operations/resources against existing database schemas. But again, data providers are not provisioned with mechanisms to enable third parties to enrich their models.

III. RATIONALE

Object–oriented programming (OOP), and in particular Java and Python which have been ranked as the most used programming languages in 2012 (TIOBE index [5]), is the most used programming paradigm. Thus we propose to implement a DaaS where data and code can be enriched based on the object paradigm.

In the OOP paradigm data is modeled by coding concepts as ***Classes*** that have data ***Fields*** (***Attributes*** that describe the class) and associated procedures known as ***Methods***. Then, ***Objects*** are instances of such classes that contain specific values for each data field and act as an entry point to execute their corresponding class methods.
The idea is to extend the concept of DaaS in a way that data is handled in the format of objects (as in OOP) offering the abstraction of Objects as a Service that include, in addition to the data itself, the methods needed to manipulate it. In this way, access to alien data can be naturally embedded in client applications, which also benefit from the functions that enable the manipulation of this external data.

Objects offer a very natural way to implement enrichment. On the one hand, we propose to enrich an existing data model by creating new classes and adding them to the original model. On the other hand, we propose to extend functionality by adding new methods to existing classes or new implementations for existing methods.

We have implemented a platform that supports these features, thus allowing several data providers at the same time, creating, enriching and offering classes using the same platform. This model enables chains of providers and third parties, having the latter acting as data providers too.

IV. OBJECTS AS A SERVICE

In order to offer Objects as a Service (OaaS), we have implemented the following mechanisms: i) a mechanism to enable a data provider to register his OO data models on the system, ii) a mechanism to define how to share such OO data models, i.e. which classes, attributes and methods, and iii) a mechanism to enable enrichment of the OO data model by means of extending its classes, adding new methods to them, or enabling third parties to provide their own code implementing the original methods of the classes (thus offering more than one implementation per method, potentially).

A. Registering data models

We assume that data providers have implemented their data models as a set of ***Classes.*** In order to offer objects of these classes as OaaS, the provider registers his set of classes into the system. In particular, registering a Java class implies sending the .class file (which contains the fields and the methods of the class) to the platform.

In such an environment with several data providers registering classes, name conflicts may easily appear, since multiple data models (owned by a single provider or not) may have classes with the same name despite representing completely different things. In order to avoid name conflicts, we introduced the concept of ***Domains*** as a higher level of abstraction from classes. Every Domain can be seen as a namespace or container for classes owned by a domain responsible (i.e. one data provider). Every pair domain-class is unique in the system, so that a single domain cannot contain two classes with the same name.

B. Sharing data models

Once the data model is registered, the data provider can share and make it enrichable by creating ***Contracts***. We define a ***Contract*** as the agreement between a data provider publishing one of his ***Domains*** and a third party. The contract comprises:

- A set of what we call ***Interfaces***, one for each class to be published. Each interface includes those attributes and method signatures that will be accessible within the contract
- A set of ***Permissions*** for each interface, which define whether the third party can create, read, update or delete objects of the class corresponding to the interface.

In addition to the specific permissions associated to each interface, the fact of having a contract allows the client to enrich the data model received as explained in subsection C.

In Figure 1, we show a diagram with a data model on the data provider side (**the circle representing the *Domain* of the provider**) and a *Contract* for a third party that a developer can use. The contract is defined with an Interface with read permission for class *Team* enabling access to attribute *teamID* and the operation *getAvgAge()*. The rest of attributes and operations, or even the entire class *Person*, are hidden for the third party developer.

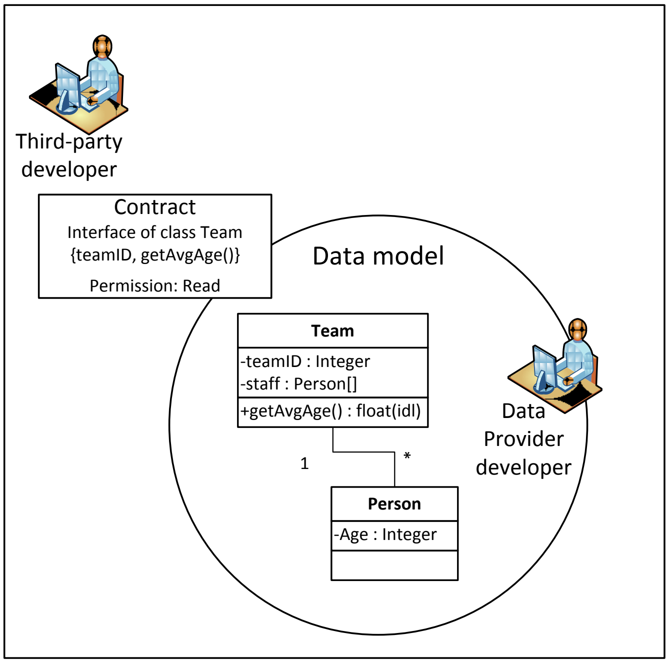

Figure 1: Contract for third parties

C. Enriching existing data models

Once a third party has some contracts with his data provider (or data providers), we offer several mechanisms to enrich the providers' data models, both with new concepts and functionalities. In particular, we offer three main possible enrichments (widely explained afterwards):

1. Adding new classes to the data model of the provider
2. Adding new methods to the data provider's classes
3. Adding new implementations for a method, thus having several implementations per method.

Importantly, previous existing contracts are not affected by enrichments, in the sense that the new classes or methods added will not be visible by the rest of clients of the data provider. In order to publish these enrichments, their creator must define a contract acting as a provider, and offer the new data model to his own clients.

Regarding the first feature, third parties can use and enrich original data models with their own classes. The clients of a contract are able to inherit from any class provided in a contract, and they can also define a new class using providers' classes to declare attribute types or return types for its methods. **As a result, a third party can create his own data**

models based on the provider's ones, and at the same time the data provider can enhance his services by enabling a third party to enrich the original data models with its own classes or extensions.

The second feature is more intuitive. The enrichment of existing classes by means of adding methods is an improvement on existing data models. Figure 2, shows an example of the original data model (introduced in Figure 1) that contains both classes *Team* and *Person*. The class *Team* is enhanced by adding a new method that uses the existing information (*Person[]*). Although nowadays DaaS can add new features, these features have to be implemented by the data provider. **Our novel approach is enabling third parties to perform this method modifications and/or additions themselves without interacting with the data provider**.

Furthermore, we also allow having multiple implementations for a single method, thus making an extra layer of enrichment that even extends the traditional OO paradigm. With this feature, we allow having a resource manager, which for instance can decide the implementation to be executed depending on the available resources, and we also enable data providers to explicitly define a specific set of accessible implementations for each method in a contract. **Therefore, a third party can enhance the execution of existing methods by means of adding an improved implementation.** This implementation could better exploit the provider's infrastructure, for instance if the existing implementation does not exploit parallelism and a third-party adds an OpenMP implementation. **Furthermore, it is possible to enhance business models by defining a different price for each implementation, or selecting a certain implementation depending on the customer's account type.**

V. IMPLEMENTATION DETAILS

In this section we provide several details on the implementation of our platform to provide OaaS.
Finally we also present the main modules of the middleware deployed on the data provider side, and the client library for the third parties.

A. User authentication

When talking about enriching data models we are assuming specific roles. On the one hand, we have the so-called data providers that publish their data models (or portions of them), and on the other hand there are third parties that, by means of contracts, can access providers' data models and enrich them too. Nowadays we use a typical ***User-Credential*** mechanism to manage the authentication of the users, who must register into the system before being able to use it.

B. Contract materialization

Given that a contract interface represents a part of an existing class of a provider's data model, we implemented an **OO proxy pattern based on *Stubs***. That is, a Stub class is created automatically for each accessible class corresponding to a contract interface. This Stub class only contains the part of the class that is visible according to the definition of such an interface. Then, the third party only needs to retrieve the Stubs related with the contracts he owns and use them to compile its applications or the enrichments of existing classes.

Original data model	New data model

//Data model
 	
Class Team {
 Integer teamID;
 Person[] staff;
 	
 Float getAvgAge() {
  sumAges = 0;
  foreach p in staff
  {
   sumAges+=p.age;
  }
  return sumAges /
       staff.length;
  }
}
 	
Class Person {
 Integer age;
}
	//Data model
	
	Class Team {
	 Integer teamID;
	 Person[]staff;
	
	 Float getAvgAge() {
	  Float sumAges = 0;
	  foreach p in staff
	  {
	   sumAges+=p.age;
	  }
	  return sumAges /
	       staff.length;
	  }
	
	 Float getMaxAge() {
	   Float maxAge = 0;
	   foreach p in staff
	   {
	     if (maxAge < p.age)
	       maxAge = p.age;
	   }
	   return maxAge;
	 }
	}
	
	Class Person {
	 Integer age;
	}

Figure 2: Enriching a data model by adding a method

In Figure 3, we revisit the *Team-Person* example of figure 2 and show how the third party would use the Stub for the class *Team*. Let us assume that the data provider has agreed a contract with a third party that enables the third party to access to class *Team*, but for privacy reasons the data provider hides the class *Person*. However, the data provider can specifically enable the third party to execute the method *getAvgAge()* of class *Team* (i.e. avoiding access to a single person, but allowing to retrieve aggregates or stats about teams). Now, the third party can create the class *Department* that has an array of *Team* objects, so it can compute the average age of all the staff in the whole department by using *Team Stub* to request the average age of the staff of every team (transparently as it is declared like the original *Team* class), and compute the global average from all the retrieved teams' average ages.

As you can see, **this is also an example of enabling third parties to use providers' classes (and objects) in order to define their own applications,** since the attribute *teams* of the class *Department* is of type *Team* from data-provider's data model.

C. *Stub generation*

The generation of Stubs is provided on demand. That is, the third party has a contract with a data provider during a certain period of time and, until the contract expires, the third party can retrieve the corresponding Stubs when needed (e.g. a new developer needs them, or in case they were deleted so third party wants to get them again).

Given that Stubs are generated from the original classes by filtering its attributes and methods (following the specification of a contract) and modifying the methods for remote execution, we needed a mechanism to interact with the code of the corresponding classes. But we cannot assume that we will always have access to the source code.

However, Java (the first programming language we support) is firstly compiled generating and intermediate code (byte code) which is afterwards interpreted by the Java Virtual Machine (JVM). The byte code generated by current Java compilers is a bit complex and tricky, but it is possible to manipulate it in a more comfortable way by means of existing tools like Byte Code Engineering Library [6] (BCEL) or Javassist [7].

We chose BCEL because of our previous know-how. BCEL has already built-in support for dynamically creating classes, so we use it for generating Stubs on-demand whenever a contract needs to be materialized.

D. *Transparent execution*

In our first version of the system, all the methods are executed in the data provider infrastructure. That is, if a third party uses a Stub to interact with provider's data model, the intrinsic computation (method invocation) is actually performed on the data provider side. This obviously simplifies the execution scheduling and resource management, and on the other hand lets us focus on the use cases where the data provider still wants to keep services execution under control (which at the end is a common use case).

With the constraint of having everything executed on the data provider's side, we still have the goal of making such an execution to be transparent leading us to resolve the following issues:

1. How to enable a third party to authenticate himself (with his credential) when using his contracts (with the Stubs) without having to change his applications. That is, avoiding applications to run any kind of explicit authentication against data providers.
2. How to make the Stubs act as proxies that eventually execute their methods on the data provider side.
3. How to create data objects from Stubs since providing CRUD commands on existing classes is necessary and a third party could also create new data from his enrichments.

Data Provider

```
//Data model

//Main class team
Class Team {
 Integer teamID;
 Person[] staff;

 Float getAvgAge() {
  sumAges = 0;
  foreach p in staff
  {
   sumAges+=p.age;
  }
  return sumAges /
       staff.length;
  }
}

//Main class person
Class Person {
 Integer age;
}
```

Third party

```
// Basic info of the
// stub for Team
Class Team {
 Object oid;
 Float getAvgAge() {
  return
    clientLib.execute(
     oid,
     “getAvgAge()”
    );
 }
}

// New class department
// uses Team stub
// transparently
Class Department {
  Team[] teams;
  Float getAvgAge() {
    sumAvgAges = 0;
    foreach t in teams
    {
      sumAvgAges+=
        t.getAvgAge();
  }
  return sumAvgAges /
       teams.length;
}
```

Figure 3: Data model and application example

Regarding the first issue, in order to accomplish transparent authentication, the third party must retrieve the Stubs from the data provider (of course, considering the contracts they have). That is, our system in the data provider's side accesses the registered data models and, taking the contracts into account, automatically generates the Stubs for the third party. Consequently, we needed to develop a mechanism to inject the authentication information in the Stubs in such a way that whenever a method is invoked (to be executed in the data provider) this information is passed from the third party to the data provider (where our system validates the third party, checks the contract is still in force, etc.). At the end, once having the Stubs, the developer in the third party must not be aware of the authentication mechanism since it is implicit in the Stubs.

Regarding the second issue, we already commented that Stubs are built from the original classes and this is performed by firstly getting the corresponding contract interface in order to know the accessible attributes and methods and, afterwards, generating the actual Stub containing only such visible parts of the original class. Therefore, the remaining issue is how to eventually execute original methods in the data provider's side. To that end, when generating the Stub the system substitutes the original method calls of the class implementation by Remote Procedure Calls (RPCs) containing all the needed information, i.e. not only the method signature and the parameters but also the authentication information and the contract in use (the contract which the Stub comes from). Then, the RPC is eventually executed in the data-provider's infrastructure by means of loading the original class and executing the corresponding method.

Finally, the third issue mentioned is resolved by means of intercepting the Stub constructors in such a way that when instantiating such a Stub (e.g. *"Team x = new Team();"*) this actually creates an object on the data-provider's side as an instance of the original corresponding class (e.g. an object of the original *Team* class). **This enables the third party to give feedback to data provider since it can create new information directly within its applications by means of the Stubs** and with the appropriate permissions.

E. Execution plan through multiple Domains

Although Domains are designed to offer a complete data model by themselves, they might be related. For instance, let us refer to Figure 3 again, where a third party has added the *Department* class that refers to the *Team* class of the provider's data model. Now, this third party wants to offer the class *Department* with all its contents to one of his customers. In order to accomplish it, the third party creates its own *Domain* and offers a contract with its customer that contains an *Interface* for the class *Department* (as the provider did when he shared his class *Team*). Now the third party's customer develops an application that executes *getAvgAge()* of class *Department*. This method is resolved by internally calling the method *Team.getAvgAge(),* so the execution plan is starting from the third party *Domain* to the original provider's *Domain*.

In this scenario the security could be jeopardized if, for instance, the contract from the third party to his customer was defined to be longer (in terms of expiration dates) than the original one between the data provider and the third party. In this case, the customer could end up being able to execute *Team.getAvgAge()* while the original contract between the provider and the third party has already expired.

For the first version of our approach, we implemented an execution plan that checks the current contracts involved on the execution of an implementation. That is, when the third party's customer attempts to execute *Department.getAvgAge()*, the system will check not only the contract that enables the customer to do it, but also validates the contracts involved in the execution of the selected implementation. As a result, in Figure 3 where there is only one implementation for the method *Department.getAvgAge()*, the system will validate the contract between the responsible of such an implementation (the third party) and its provider for the required internal method *Team.getAvgAge()* (i.e. the *Domain* of the original data provider).

F. Client library and middleware for Data Provider

We implemented a client library in order to encapsulate the management of remote execution from Stubs to data providers. This library is configured to know where to find the data-provider's service and consequently how to resolve the RPC invocations to be executed in such a data provider.

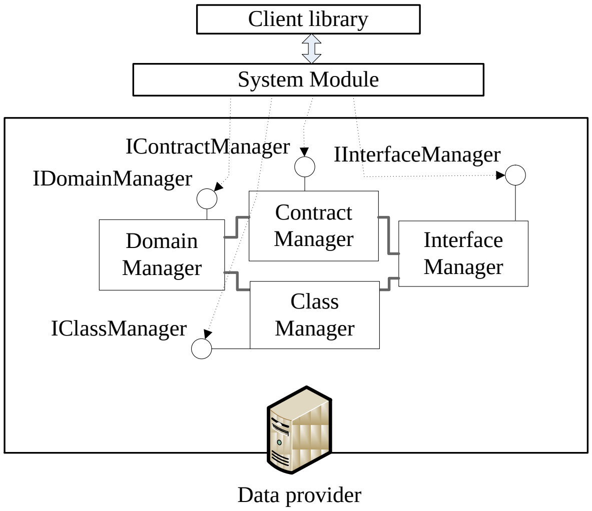

Figure 4: Middleware and client library

In Figure 4, we show a diagram with the client library and the main components of the **middleware** system deployed in the data provider infrastructure. The ***Contract Manager*** deals with the creation and validation of contracts between data providers and third parties. It is related with the ***Interface Manager*** which handles the Interfaces registered in the system (an Interface can be reused in several contracts). The ***Domain Manager*** and the ***Class Manager*** are in charge of managing the domains and the classes of the data models respectively. Finally, the ***System Module*** publishes the service that enables the client library to request any of the offered features and handles the requests with the support of the managers.

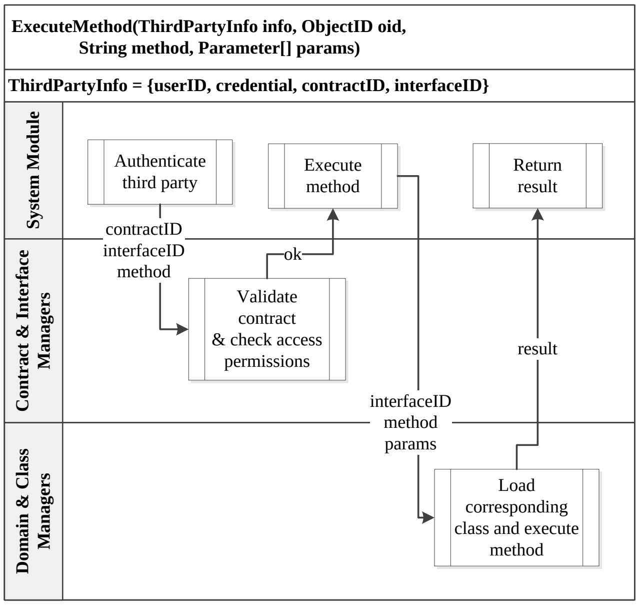
ExecuteMethod(ThirdPartyInfo info, ObjectID oid, String method, Parameter[] params)
ThirdPartyInfo = {userID, credential, contractID, interfaceID}
System Module
Contract & Interface Managers
Domain & Class Managers
Authenticate third party
Execute method
Return result
contractID interfaceID method
ok
Validate contract & check access permissions
result
interfaceID method params
Load corresponding class and execute method

Figure 5: Method execution

Besides, the system module is also in charge of authentication.

The flowchart in Figure 5 shows the common activity between the System Module and the other Managers for the execution of a method. The System Module firstly authenticates the third party, then validates the contract (expiration date, method is accessible, etc.) and finally loads the corresponding class to execute the byte-code of the selected method.

VI. CONCLUSIONS

In this paper we have presented a new opportunity to add value on DaaS solutions. We have introduced the first version of our approach both conceptually and giving details of our current implementation.

In particular, we have proposed a new abstraction of DaaS: Object as a Service. With our OaaS approach we enable third parties to enrich providers' data models and functionalities in the context of the Object Oriented paradigm which nowadays is the most used programming paradigm.

For this reason, we strongly believe that these mechanisms can improve B2B models and collaboration among different organizations.

VII. ACKNOWLEDGEMENTS

This article has been realized with the support of the Spanish government under grants SEV-2011-0067 of Severo Ochoa Program, and TIN2012-34557. In addition, it has also been supported by the Catalan Government under the 2009-SGR-980 grant.

REFERENCES

[1] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, A. Vakali, "Cloud computing: distributed Internet computing for IT and scientific research", IEEE Internet Computing, vol. 13, no. 5, 2009, pp 10-13.

[2] M.J. Carey, N. Onose, M. Petropoulos. "Data services". *Communications of the ACM 55(6),* 2012, pp. 86-97.

[3] Infopar, Qualitta DaaS Server. http://www.infopar.com/en/database-as-a-service-daas-dbaas.html (accessed 02-21-2013)

[4] WSO2 Data Services Server, http://wso2.com/products/data-services-server/ (accessed 02-21-2013)

[5] TIOBE Programming Community Index, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (accessed 02-21-2013)

[6] Apache Commons BCEL, http://commons.apache.org/bcel/ (accessed 02-21-2013)

[7] JBoss Community, Javassist, http://www.jboss.org/javassist (accessed 02-21-2013)