Invited Paper:

Living Objects: Towards Flexible Big Data Sharing

Jonathan Martí
Barcelona Supercomputing Center
jonathan.marti@bsc.es

Anna Queralt
Barcelona Supercomputing Center
anna.queralt@bsc.es

Daniel Gasull
Barcelona Supercomputing Center
daniel.gasull@bsc.es

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

ABSTRACT

Data sharing and especially enabling third parties to build new services using large amounts of shared data is clearly a trend for the future and a main driver for innovation. However, sharing data is a challenging and involved process today: The owner of the data wants to maintain full and immediate control on what can be done with it, while users are interested in offering new services which may involve arbitrary and complex processing over large volumes of data. Currently, flexibility in building applications can only be achieved with public or non-sensitive data, which is released without restrictions. In contrast, if the data provider wants to impose conditions on how data is used, access to data is centralized and only predefined functions are provided to the users. We advocate for an alternative that takes the best of both worlds: distributing control on data among the data itself to provide flexibility to consumers. To this end, we exploit the well-known concept of object, an abstraction that couples data and code, and make it act and react according to the circumstances.

Keywords: Data sharing, data control, offloading, enrichment, persistent objects, Data as a Service (DaaS), Big data.

1. INTRODUCTION

Traditionally, data has been seen as a passive element, with applications being in charge of consuming it to perform useful tasks. This paradigm was reasonable when computation was the main goal and data was just something needed to perform that computation. Today, data has become the key element in most computing infrastructures, both because of its relevance and because of its size, and the term *big data* has emerged to refer to the challenges resulting from data sets becoming so large, diverse and complex that they cannot be handled by traditional methods. Thus, new requirements regarding how data is managed and served have to be addressed.

First, data is becoming more open and thus building new services by **enriching** existing sources will become the general trend. Actually, initial steps in this direction are already appearing in open data initiatives [2], data markets [1] or Google Maps and Google Fusion Tables [15, 11], among others, where providers allow third parties to create new services based on data enrichments. For instance, under the philosophy of open data, many governments worldwide are releasing data for free access to promote innovation via data-centric services (a couple of examples among many others are PublicData.eu in Europe, and Data.gov in the United Sates). In the private sector, Google Maps allows third parties to add information about restaurants, hotels, etc., that are shown when querying the maps. Google Fusion Tables enables users to upload table-structured datasets, merge them with other datasets, and visualize them using, e.g., a Google Maps mash-up.

Although adding data layers on top of existing data is a good starting point to build more complex services, this simple mechanism does not help if some kind of processing is required, e.g. to offer a service that merges data from different providers. As of today, in order to build such services, data (or a subset of the data) has to be copied from the infrastructure of the data providers to the infrastructure of the service providers where it will be processed and served. This movement has many drawbacks, such as energy waste when moving the data, decreased data quality when data is simplified or cached (thus not up to date) to reduce traffic, and a potential performance reduction in the service, among others.

A possible solution consists of data providers implementing the operations needed by their clients and offering them via a data service [3]. This approach becomes unviable if many different services, with diverse and arbitrary needs, want to process the data. The provider cannot have the expertise to write the code needed by all its clients and, as a result, client services are limited by the functions offered by the data provider.

Second, in the cases where data is copied to the infrastructure of the new service providers, the owner of the data should not lose **control** over what can be done to the data, which is not the case when using today's technology. For instance, if a mandatory update has to be done to some data, this will not be possible for data copied to a third party infrastructure. Although not vital in all cases, this loss of control of the shared data may prevent providers from releasing data that they would be willing to provide if such control was guaranteed.

And third, although there are solutions to handle large amounts of data (such as parallel databases [12, 10, 22] with a MapReduce [6] interface, or NoSQL stores [5, 7]), the resources available are limited to the ones accessible to the data provider. This limitation becomes a challenging problem if the required processing, especially from multiple clients, exceeds resource capacity. In this case, the system should be able to **offload** part of the data and computation to a resource owned by the client or a third-party resource available to the client or provider. Again, as the data leaves the infrastructure of the provider, the control over what can be done needs to be guaranteed, otherwise this offloading becomes unacceptable from the provider point of view.

We argue that all these issues, which can be summarized as maximizing both the control of the data provider on his data, and the flexibility of the service provider in the deployment of new services, are important problems to be solved in big data sharing.

We advocate that the way to solve the previous problems consists in an evolution of the concept of data service, taking it to the limit by moving control even closer to data, or rather, letting data control itself.

This paper is organized as follows. In section 2 we present real problems that cannot be easily solved with current technology. Section 3 presents the idea of living objects, while in section 4 we explain how they can help to solve the real problems presented in section 2. An overview of some technological issues related to the implementation of living objects is given in section 5, and in section 6 we review related work. Finally, conclusions are presented in section 7.

2. DATA SHARING PROBLEMS IN THE REAL WORLD

The requirements described in the previous section have been identified while discussing current problems with the industry and how current technology limits their products and services. Now, we present four use cases that will demonstrate the actual needs that are not yet solved by current technology. Each of the first three use cases puts emphasis on one of these requirements (though all requirements are present in all use cases), whereas the last one combines them all.

Although the use cases are real, the names of the companies and organizations are fictitious to preserve their privacy.

Enrichment: Geospatial Data Enrichment
The National Geospatial Agency (NGA) is a public geospatial data provider. Current users download the portion of the data needed to their own infrastructure and then perform complex computations to serve their clients. Although the volume of the data for a single request may not be large, the aggregated volume that needs to be served for all requests is very large and cannot be easily transferred over the Internet. In addition, downloaded data may become inconsistent if it is not kept up-to-date in an explicit way. Avoiding these large data movements, and thus guarantee data consistency, is the main challenge for NGA and its users, since this would improve the quality of the service as well as the speed of processing.

A possible way to use the data on-site would be to deploy a WPS (Web Processing Service) platform, since it enables executing transformations and analytics directly on the data. WPS is a standardized interface, developed and maintained by the Open Geospatial Consortium, for invoking geospatial-processing services, ranging from simple coordinate transformations to complex simulations based on spatial data. However, WPS only allows execution of functionality that is predefined by the service provider, which is not necessarily the one required by a given user or application developer. In addition, WPS needs all data to be present at the location of a service to be able to process them.

Rather than this traditional scenario in which providers determine which functionality is required or which data is copied to another infrastructure where any processing can be done, NGA would like to provide to its users with the ability to deploy new processing functionality directly on their data by enriching the data with code stored in NGA infrastructure. This requires taking into account security issues in the sense that the execution of third party code is done in an isolated environment, so that it has no impact on the data itself, the general distribution of data, or on other running processes.

A concrete example of the problem can be seen in a cycling event scenario. Organisers of the event want to provide graphical representation about the positions of the cyclists on the track, which is of particular importance for TV audiences. E.g., several graphic views are available, providing the TV audience with a quick overview about the track difficulty (cumulative elevation the cyclists need to overcome), the position of the leaders vs. peloton, etc. However, this information is usually offered in a simplified form, ready for fast consumption and does not provide a complete assessment of the current situation. In this example NGA would want to offer its maps, its Digital Elevation Model and the event organizers would implement the views and graphics needed for their audience. With today's technology, the organizers' needs to download the maps and elevation models and make their computation on their own infrastructures.

Control: Exporting News Archives to the B2B Market
In a newspaper archive, such as the one from Today News (TN), there are large archives that have been obtained from digitizing old newspaper items. Once scanned, images are processed using state-of-the-art OCR systems, and users can then perform simple searches over the texts generated from them, which take them to the relevant scanned pages.

While this search functionality is enough for individual use, TN is interested in exploiting this valuable source of information by opening it to other companies outside the news domain. The main problem TN finds is related to the provision on dynamic data access policies over pieces of data. On the one hand, there are legal restrictions on some specific content of some pages, which imply that it cannot be accessed under any circumstances. Importantly, these restrictions can appear at any time, and must be immediately enforced (e.g. a person that discovers his name in an old piece of news and wants to preserve his privacy). On the other hand, there are copyright issues over some pieces of data that are not owned by TN, such as images, which can be reproduced or managed as part of the newspaper but cannot be used individually.

For all these reasons, TN needs to allow access to its data, enabling third parties to enrich it with code to offer new services, but is not willing to lose control over the data it owns. The ability for third parties, specialised in handling such type of data sources, to provide higher level views over the content that is made available, or to build applications using that content, has the potential to benefit consumers of this information (be they citizens or businesses) and more importantly will foster the creation of a business ecosystem around this information source.

A concrete example of TN's needs consists of enriching the archive data by identifying persons, events, locations, etc., using state-of-the-art techniques already successfully applied in other contexts (including the news domain). Once these relevant entities and the relationships between them have been established, TN will be able to define refined access policies. For instance, a given company is allowed only to see news concerning a specific country, or items related to crime, or conversely items related to a certain person can never be accessed. This allows TN to share its data under the precise conditions they want to impose. Then, third parties can build applications that consume this information and manage concepts instead of plain text pages, enabling more sophisticated treatment of the data.

With today's technology, this can only be done by copying the data to the new provider infrastructure, and then TN loses control over what can be done to its data, how it can be modified, who can see what, etc.

Offloading: Event Organization
Event Solutions (ES) is currently developing a B2B application that enables companies to organize events involving a large amount of people. The application must automatically negotiate and book at once resources from different providers (hotels, restaurants, cars, venues...), taking into account the obvious availability and pricing conditions, as well as a great number of additional customer-related restrictions that make the problem very complex.

An example that illustrates the problem is a British company with 200 employees that organizes its annual meeting in Barcelona. The company needs to book flights for all of them, and also 200 hotel rooms (2 suites, 188 single rooms, and 10 double rooms, and one of these double rooms must be in the 1st floor). They also need 1 private car with chauffeur, and enough buses for the rest of people. They need a venue to hold all the employees and will hire catering, taking into account that 20 employees are vegetarian. They will need pink chairs as well. All these resources have to be in a range of 500 meters from each other.

Solving this problem is very CPU intensive, because a lot of different conditions have to meet for each resource, and several resources have to be combined in order to find a valid solution. An application doing this task can easily overload the resources of the service providers (ES), and ES may thus not be able to make a timely offer to the customer, nor coordinate the bookings of the providers that contribute services to the event.

The ideal solution in this case would be to offload this computation (and the portions of data required) to third-party resources, either those of the application consuming the data, or the cloud, while guaranteeing that the access permissions and the business rules imposed by the data provider are satisfied. Otherwise, the owners of the data (hotels, airlines, etc.) would not allow such offloading.

Everything Together: Business Intelligence
Business Intelligence (BI) aims to provide organization-wide IT-based decision support, usually based on processing large volumes of strategically relevant and highly confidential data subject to a variety of legal, contractual, and intra-company restrictions.

In particular, for their planning and budgeting tasks, PharmaLab (PL) aims for a BI solution that allows them to come up with realistic budget values for their set of Key Performance Indicators. This process is intensive on data processing, since indicators are interdependent and the planning process is based on terabytes of confidential data. In addition, current processing requirements prohibit fully interactive planning scenarios.

On the other hand, since social media more and more shapes opinions, PL are interested in the analysis of data from social networks to infer customers' opinion about their products. This is a prime example of a big data application, and the combination with confidential company specific data enables this information to serve marketing and sales purposes, e.g. when assessing the potential sales impact of certain opinions or during the design of viable pricing strategies.

In both scenarios, integration with market data e.g. from sales partners would improve PL's analysis. This integration should be done under the precise conditions imposed by the different partners when sharing their data, and providing different views of the data to different players. Given the temporal nature of both processes, a solution taking advantage of the elasticity of the cloud would be ideal. However, current cloud solutions are not enough since computation (and the portions of data involved) cannot be offloaded to the cloud while guaranteeing security and privacy as if it was performed at PL infrastructures.

3. LIVING OBJECTS: CONTROL IS IN THE DATA

We propose the concept of living objects as a solution to the problems we have identified. A living object is a piece of data that is bound to all the logic needed to process it (methods), and the policies that manage its behaviour with respect to security, integrity, etc.

This approach can be seen as an evolution of the concept of data service [3], taking it to the limit by moving control even closer to data. Data services encapsulate data by providing a set of functions that guarantee that the rules and policies that the data provider wants to enforce are always satisfied. Then, instead of providing a single access point for a dataset as a whole, our proposal is to encapsulate each piece of data, i.e., an object, by providing a set of functions that protect it and guarantee its correct behavior, regardless of their physical location.

With this change in the paradigm, objects can leave the data store without losing their properties, because not just the data is moved, but also its associated methods and expected behaviour. For instance, we can move objects to the application space (or to a third-party resource) without worrying about their correct behavior. This freedom of movement increases the flexibility of object management because we can decide where an object is manipulated depending on its size (we can avoid moving it), the complexity of their methods (if they are very CPU intensive we can compute them in a high-performance computing site), the current usage and state of resources, etc.

In this way, one of the main problems of data services is overcome, since capacity is not limited by the resources accessible to the data provider.

Another drawback of data services is the lack of flexibility from the point of view of data consumers, since they can only access data in the ways that the data owner has defined. But if the shared data is protected as we propose, third parties can safely enrich the original data with new types of information, change how information is exported to applications, and even add new functionality to process the integrated dataset, while the data provider is sure that his rules are never broken, since rules are an indivisible part of the objects.

In order to do so, we revisit the concept of persistent object, which is a mechanism to store data close to how applications understand and deal with it. We argue that the potential of objects has been underexploited because they become passive when made persistent. If persistent objects could take care of themselves, working with data would be a much simpler task. Thus, we propose bringing objects to life and make the objects themselves be the ones that manage their own integrity, privacy, security, synchronization, and even their own life cycle. Objects should also be in charge of choosing the most appropriate resources to carry out a given task depending on the circumstances and their relationships with other objects. These actions should be performed by the objects regardless of whether there is one, many, or no application accessing them.

In the following subsections we present some relevant tasks that can be assigned to objects. This list is by no means exhaustive, but it should be enough to show the potential of this approach.

They Enforce Integrity

In our target environment, different applications want to access and modify the same data, while the data provider wants to guarantee that his data always satisfies the integrity constraints that formalize his business rules. In this scenario, it is clear that enforcing integrity from the applications is unfeasible, since they may be built by third party businesses. Thus, the solution in these cases is to move the responsibility of integrity close to the data by means of data services or constraints and triggers when possible.

Following the idea of moving integrity enforcement closer to the data, the next and final step is to move this task to the objects themselves. This converts passive data managed by a third party code (the storage system, e.g. a DBMS) into an active object that makes sure that all integrity constraints that affect it are enforced.

In addition to the common benefits explained above, assigning the responsibility of integrity enforcement to the objects also simplifies the enforcement of integrity constraints involving third-party data, for instance referential integrity across remote databases or services, since they can be accessed inside the methods. Furthermore, if we cannot guarantee that the third-party data store will enforce our constraints, the object can wake up every while (or every access, or…) to check whether the conditions have changed and react to maintain integrity. It will not be an immediate enforcement, but will do the job in many cases.

They Control Privacy and Security

Privacy and security are becoming the key issue when managing data, especially in the emerging scenarios where the same data is to be shared by different users, applications, organizations, etc.

Analogously to integrity enforcement, we can move the responsibility of privacy and security to the object itself, so the expressiveness available in the application is maintained and the security is kept independent of it. If living objects themselves are in charge of security and privacy, any application built on top of them will follow the rules imposed by the owner of the data, which provides extra flexibility. Taking this idea to the limit, an individual object should be able to decide, for example, in which conditions it appears in query results (for instance depending on the location from which the query is executed), or even that it never appears.

Besides, it can perform these decisions even after migrating among different services or when it raises from lowest layers to the application. The object itself keeps the necessary security and privacy rules for it.

They Keep Sync

With today approaches, if data is always under the control of the data store (i.e. like in data services), all modifications to the data need to go through the data services and this protocol may add significant latency to data-update operations and potential overload of the data services in general. This approach implies contacting the data service even if the data does not need to be made persistent. For instance, a given object may require being stored immediately when a critical value is modified, but an asynchronous update may be sufficient for the rest of its properties. In this last case, the application cannot know this policy, and the data service is always contacted.

It would be ideal if applications could take the decision on when contacting a data store is a must and when it can be delayed. Unfortunately, this would imply that this kind of code is replicated in all applications, and we have already discussed that this approach is complex and prone to errors.

With our new paradigm, objects take their decisions regardless of their location. Thus, the objects themselves decide when they need to become persistent and when not.

Following the same rationale, the objects could take care of keeping their replicas consistent with the consistency semantics defined by the data programmer (which gives the programmer full control and flexibility over these policies).

They Manage Their Life Cycle

Another clear example of why objects should come to life is the management of their life cycle. By data life cycle we understand things such as when an object can be removed/modified as well as auto compressing when seldom accessed, or migrating to new data formats (i.e. from mp3 to mp4).

If the object is the responsible of its own life cycle, we can add methods that are triggered at some intervals, or when a given event or condition occurs, that perform the needed action. This approach has several advantages. First, the possible actions to be done are not limited to a restricted set known by the storage system, but are as open as anything that can be expressed in a method (similar to the data services case). Second, like in the previous examples, these kinds of operation are also enforced while the object is managed in the application space, and not only in its persistent state in the data store, avoiding code replication or contacting a given service constantly. Third, given that the object itself decides when such actions need to be performed, no scanning through the whole set of objects is needed (like in backup operations today). Furthermore, as we have already seen, if these operations are resource intensive, they can be easily offloaded to a third-party resource given that they are self-contained as part of the object.

They Are Versatile
An important feature of object-orientation is method overloading, consisting on the existence of several versions of a method with the same name and different signature in a given class. In our living objects we can take this further and allow not only several versions of a method, but also several implementations for each version. In this way, a method with the same name and signature can also have different versions, which may differ in the required resources such as memory needs, possibility of Graphics Processing Unit (GPU) optimization, use of object processors [16], etc. At runtime, when a method is called on an object, this object proactively chooses the most appropriate implementation according to the available resources.

We can use the same mechanism to offer several implementations of a method coded in different languages, and thus allow a given method to be executed in the application space from a java application as well as from a C++ (or any other language). The data provider, depending on the expected clients, would decide what languages need to be supported for each class.

Not only can an object choose the method implementation to be executed, but also the location. Depending on factors such as the current load or the available resources, an object may decide to execute a method either in the same resource the application is running (client machine), in the local resources of the data store, or using a third-party resource such as a Cloud computing or high-performance computing (HPC) datacenter. For instance, assume a scientific application executed in a smartphone that includes a method requiring plenty of computational power. In this case, the object can decide to offload execution to an HPC resource, as long as it has an appropriate implementation and available resources.

4. HOW LIVING OBJECTS SOLVE REAL DATA SHARING PROBLEMS

The objective of this section is to demonstrate the benefits of taking the living objects approach, showing how they help in the scenarios described in Section 2.

Enrichment: Geospatial Data Enrichment
To address issues related to publishing data from a cycling event, NGA would allow the organizers to enrich their maps and digital elevation model with the new objects (data and logic) needed to compute the graphical representations needed to be shown to the TV audience. This graphical representation will be computed in NGA infrastructure and the result will be sent to the organizers (which is less than the elevation model and maps for the whole race).

This enrichment of the information at NGA can be achieved by merging, in a single infrastructure, objects from data providers and objects from third parties that will perform the needed computation without breaking the rules imposed by NGA.

Living objects enable cycling-event organizers to include its objects into NGA infrastructure because the behaviour will be as defined by the cycling-event organizers even if it is stored and computed in NGA infrastructure because both methods and behaviour are part of the objects themselves. On the other hand, original data can be enriched with new methods because living objects guarantee that even new methods will follow the security, integrity, sync, and life cycles behaviour defined by NGA.

Control: Exporting News Archives to the B2B Market
To allow third parties to use TN's archive without losing control over the data, TN should allow third parties to insert new objects into their infrastructure and thus they would be able to get the needed information for their new business. On the other hand, data would not leave TN and thus all security and privacy rules would be applied immediately. For instance, if somebody asks that his name disappears from the news, it will disappear for all services because no copies of the data are out of TN's control.

Again, this secure enrichment of the information at TN can be achieved by merging, in a single infrastructure, objects from data providers and objects from third parties that will add value to the original data, using only the information that TN allows at any given time. This can be done because the policies are embedded in the objects and, thus, enrichments or new methods cannot break the rules imposed on existing data.

Offloading: Event Organization
To avoid overloading the providers with many queries and computations, these providers will allow objects to migrate and computation to be offloaded as long as the privacy, security and integrity rules are met regardless of the destination of the data and computation.

With living objects, providers would be able to define their security policies, integrity constraints, sync policies, or life cycle behaviour and no application consuming their data would be able to break their rules, even if the objects have been migrated to a different infrastrcutre, since they are delegated to the objects themselves. In particular, this greatly facilitates the development of ES' application and guarantees the conditions imposed by the data provider.

Everything Together: Business Intelligence
A good way to address typical BI pain points and in particular those of PL, like the flexibility needed to respond to fluctuating workloads, would be outsourcing to the cloud. However, the privacy and security issues regarding highly confidential data are the reason for the slow adoption of cloud-based BI.

Living objects do not only address these issues by combining performance, cloud based elasticity and built-in security, but will go beyond and above by allowing fast data sharing within and across enterprise borders and the integration of external (and possibly unstructured) data for analytical purposes, thus enabling new types of BI solutions. This is possible since data management logic and data are kept together in living objects, which enforce the rules of the data owner. Performance and flexibility are achieved also in this way, since living objects can be offloaded to any infrastructure (including the cloud) to perform computation.

In addition, since living objects include the rules imposed by the data provider, different views (or enriched versions of PL's data) can be offered, thus allowing to perform BI directly on the data, without the need to move it to a dedicated data store to analyse it. At the same time, this provides the ability to offload computation to the cloud or to any other resource implies an important performance gain for parallelizable tasks.

5. CHALLENGES TO REALIZING LIVING OBJECTS

Although the idea of living objects may seem very philosophical, it is based on a very simple technical concept: keeping the object methods with the data when an object is made persistent and let the objects manage the precise conditions under which their methods are executed, and even let objects execute methods independently of applications.

Object-oriented programming languages already couple data and processing logic. In addition, there are some languages, such as Java, that provide a mechanism to control the access to an object by means of GuardedObjects [13]. GuardedObjects wrap an object together with its access control policies, implemented in a separate Guard object. Our proposal innovates by including the policies in the objects themselves, thus providing transparent access to them even when they include policies. This mechanism applies not only to access-control policies, but also to any kind of rule that the object must satisfy (i.e. integrity constraints, synchronization rules, life-cycle management...). In this way, third-party developers can add new methods or refine existing ones, always following the rules defined by the data providers, which cannot be separated from the objects.

The technology supporting living objects should enable the combination of these policies and rules (integrity constraints, security policies, etc.) at different levels such as country of origin, infrastructure, data owner, etc. The result of this combination could then be compiled and injected into the methods of the object; thus the methods of a living object would contain the original logic plus whatever is needed to check/enforce the afore-mentioned policies and rules. It is important to notice that this injection will be done to all relevant methods regardless of whether they are original or enrichments, thus guaranteeing the right behaviour independently of who developed the methods code. This way of implementing rules has two key advantages. On the one hand, it allows objects to embed policy and rule behaviour into the object logic and thus enable its migration to any infrastructure. On the other hand, policy enforcement becomes more efficient and scalable because we eliminate the process of searching all rules when checking a given object. In most cases, this process will be managed at rule-compilation time. This scalability comes at the cost of an overhead when policies are added or changed. In this case, all needed rules have to be recompiled and re-injected in the class methods. With current trends, the frequency of policy updates is much less that the frequency of policy checking. Thus, the benefits of faster and more scalable policy checking outweigh this initial overhead.

In order to implement this process of rule injection into the code, we need to make sure that anything that can be expressed as a rule can be compiled and then injected. A solution could be using a Domain Specific Language (DSL) to specify policies and rules in areas such as security or integrity, which should be especially designed to ease the task of rule and policy combination, and their further injection into the adequate object methods.

The last step that needs to be solved before living objects can become a reality relies on the security of the infrastructures themselves. If we send an object with all its data and code to a third-party infrastructure, data privacy could be endangered. For this reason different security levels should be defined for the infrastructure and in the objects. Then, the system will be able to guarantee that a given object will never be offloaded to a resource with a lower security level than the one specified in the object.

Another way of sharing data consists of building new services by iteratively enriching the available data. As we have seen, current technology is still quite simplistic when trying to share data and allow third parties to exploit it. Today, mechanisms to enrich both data and code in an arbitrary way using the same infrastructure where the original data resides are very basic. In particular, if we want to modify/enrich how this data is processed in an arbitrary way, we need to move the data from the infrastructure where the data is provided to an infrastructure where this arbitrary code can be added.

In order to enable third parties to create/modify how data is seen by applications without having to move any data around, a new mechanism is needed that allows the definition of data and code enrichments without compromising the security or integrity of the original data. For instance, we need to be able to guarantee that third party code does not break a rule that specifies that no single value can be returned, but only aggregates with at least 1000 elements, which is a condition that the data provider wants to enforce. To guarantee that the third-party logic does not break any rule, these methods should be automatically checked with the relevant rules and policies. Only when this checking guarantees that no rules will be broken by the code, these methods can become part of the enriched object in the infrastructure. As this automatic checking cannot be done on arbitrary code, the language should be limited by removing those abstractions that make this automatic checking impossible, while trying to remain as general as possible.

6. RELATED WORK

Persistence of object-oriented programming language objects, our ground–stone abstraction, has been around since the late eighties. Their usage was first standardized in 1993 with the ODMG standard, which was revised for the last time in 2003 [4]. This standard applies both to object database management systems that store objects directly, and to object-to-database mappings that convert and store objects in a relational or other database system representation.

Storing objects as seen by the program was first investigated in the eighties in object-oriented databases (OODB). Although they are used in several niche markets where performance or a rich data model is needed, OODBs did not succeed as a general-purpose data management system since, among other reasons, it was difficult to have a database shared by several applications. This, together with the fact that relational database management systems were an established technology that could already handle the amounts of data managed at that time, harmed the adoption of OODBs. However, the success of relational databases did not remove the need of programmers to store objects as seen by the program instead of explicitly managing a set of relations in a relational database. For this reason, solutions such as Hibernate [14] or TopLink [18], which implement the Java Persistence API to map objects into relational databases, have appeared and are taking over the market. However, it is interesting to note that today, when relational databases have problems scaling and becoming elastic as needed in cloud environments, OODB such as the Versant Object Database [23], ObjectStore [20] or Perst [17] are claiming to be the solution for these problems. Though their popularity is still far from that achieved by relational databases, a new interest in OODBs is raising in the industry.

Our approach is different from both approaches because it takes the whole concept of objects into account, not just its data. The ability to store methods together with the data has been implemented in a few products, such as EyeDB [8] or Oracle Objects [19]. They implement the basic mechanism that allows storing methods and executing them in the server, with the aim of making applications easier to understand and maintain, and to bring computation closer to data. However, self-contained objects require making persistent not just the data and the methods, but also the rules and policies that restrict their behaviour so that they can be safely used by third party applications, or migrated to other infrastructures for the sake of scalability.

In addition, this technology should be further matured to make living objects come true. In particular, to realize the envisioned versatility, several implementations of a method are needed, as well as the ability to choose where they are executed. Existing products store a single implementation of each method, which in Oracle Objects is always executed in the server, while in EyeDB can be either executed in the server or in the client, but always in the same site, which must be specified by the data owner. In addition, the methods that are to be executed in the client are not stored in the database. This does not allow offloading computation to the most appropriate site depending on the circumstances.

Also, to be able to implement the rest of responsibilities of living objects, it is necessary that objects react to several kinds of events. Triggers can be useful in some situations, mainly related to maintaining integrity and synchronization, but they are only fired due to events that occur on an object (e.g. creation, update...) or in the database (e.g. startup, error...). Some other issues regarding these and other responsibilities of living objects can be incorporated in the methods, but they are only executed after an invocation. To make objects proactive, objects should be able to respond also to time events and to conditions on their environment and their relationships with other objects.

Another important feature of living objects is the idea of moving computation closer to the data and thus avoiding unnecessary data movements. This technique is not new and has been applied in many different fields.

Stored procedures are the mechanism to this end offered by database systems. Using them to move application logic to the DBMS allows avoiding the overheads of the client/server model [21], but all the weight of the computation must be supported by the system. Modern data services are based on this idea [3], and encapsulate one or several data stores so that applications have services executed disregarding their internal details. This also benefits data providers, since it alleviates the problems derived from the lack of control they have over their client applications, and at the same time allows distributing the computational load. In fact, a data service can be seen as a huge object encapsulating its data, but the fact that it must be managed as a whole does not provide the flexibility we gain by moving computation even inner and managing everything at the finest granularity.

The idea of having computation close to data also appears in other fields of the storage technology, such as file systems management. For instance, active disks [9] allow moving computation to the disk drive itself, but the kind of operations is very limited, and changing them implies a change in the firmware. This idea of joining computation and data is also present in MapReduce [6], where data is partitioned and processed in parallel by different machines on a cluster. Data is distributed in such a way that each storage node can perform local access to the data and then return the merged results, thus avoiding moving unnecessary data. This approach places computation close to data, but does not decouple it from applications, which is the essence of bringing data to life.

7. CONCLUSIONS

In this paper we propose a change in the view of how data is seen. So far data has been passive, with applications or services being in charge of handling it. We propose to change this view and make data an active element that controls itself.

We have presented the notion of living objects, which are born when control on data is pushed to the most atomic element: a meaningful piece of data.

Bringing objects to life has many implications. On the one hand, it moves issues such as integrity or security enforcement, or life cycle operations into the data, facilitating data sharing among many different applications and enabling custom enrichments, with the full guarantee that data behaves as desired. At the same time, since objects remain tied to their responsibilities, computation can be more easily distributed and parallelized, and the usage of resources can be optimized.

Summarizing, bringing objects to life means guaranteeing that the data provider has full control over his data, while the flexibility of building services based on external data and the elasticity in the usage of resources are maximized. This radically changes how applications and services are designed and opens the door to a new dimension of business models.

8. ACKNOWLEDGMENTS

We thank the following researchers that have helped us to mature the idea of bringing objects to life in our frequent discussion: Ramón Nou and Ernest Artiaga from BSC, Maria S. Pérez from the UPM, André Brinkmann from JGU, Angelos Bilas from FORTH, Henning Baars from USTUTT, and Oscar Romero from UPC.

This work has been realized within the frame of the BSC-CNS Severo Ochoa program and under grant TIN2007–60625, both with the support of the Spanish Government.

REFERENCES

[1] M. Balazinska, B. Howe, and Dan Suciu. "Data Markets in the Cloud: An Opportunity for the Database Community." PVLDB 4, no. 12, 2011, pp. 1482-1485.

[2] D. Bennett, and A. Harvey. "Publishing Open Government Data". W3C Working Draft. 2009. http://www.w3.org/TR/gov-data/ (accessed January 2013)

[3] M. J. Carey, N. Onose, and M. Petropoulos. "Data Services." Commun. ACM 55, no. 6, 2012, pp. 86-97.

[4] R. G. G. Cattell, and D. K. Barry. The Object Data Standard: ODMG 3.0. *Morgan Kaufmann*, 2000.

[5] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. "Bigtable: A distributed storage system for structured data", ACM Trans. on Computer Systems 26, no. 2, 2008.

[6] J. Dean, and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Symposium on Operating System Design and Implementation, OSDI, 2004, pp. 137-150.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store", Symposium on Operating Systems Principles, SOSP, 2007, pp. 205-220.

[8] EyeDB - Open Source Object Database. http://www.eyedb.org (accessed July 2012).

[9] B. G. Fitch, A. Rayshubskiy, M. C. Pitman, T. J. Ward, and R. S. Germain. "Using the Active Storage Fabrics Model to Address Petascale Storage Challenges", Annual Workshop on Petascale Data Storage, PDSW, 2009, pp. 47-54.

[10] E. Friedman, P. M. Pawlowski, and J. Cieslewicz "SQL/MapReduce: A practical approach to self-describing, polymorphic and parellelizable user-defined functions", PVLDB 2 no. 2, 2009, pp. 1402-1413.

[11] H. Gonzalez, A.Y. Halevy, C. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen. "Google Fusion Tables: Data Management, Integration, and Collaboration in the Cloud.", ACM Symposium on Cloud Computing, SoCC, 2010, pp. 175-180.

[12] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar: "The MADlib Analytics Library or MAD Skills, the SQL", PVLDB 5, no. 12, 2012, pp. 1700-1711.

[13] Java Platform, Standard Edition 7, API Specification, http://docs.oracle.com/javase/7/docs/api/java/security/GuardedObject.html (accessed July 2013).

[14] JBoss Community, Hibernate, http://www.hibernate.org (accessed January 2013).

[15] J. Madhavan, S. Balakrishnan, K. Brisbin, H. Gonzalez, N. Gupta, A. Y. Halevy, K. Jacqmin-Adams, H. Lam, A. Langen, H. Lee, R. McChesney, R. Shapley, and W. Shen "Big Data Storytelling Through Interactive Maps." IEEE Data Engineering Bulletin 35, no. 2, 2012, pp. 46-54.

[16] N. Markovic, D. Nemirovsky, O. Unsal, M. Valero, and A. Cristal. "Object Oriented Execution Model (OOM)". In *NDCA,* held in conjunction with *ISCA*, 2011.

[17] McObject, Perst, http://www.mcobject.com/perst (accessed January 2013).

[18] Oracle TopLink, http://www.oracle.com/technetwork/middleware/toplink/overview/index.html (accessed January 2013).

[19] Oracle Database Application Developer's Guide - Object-Relational Features, http://docs.oracle.com/cd/B19306_01/appdev.102/b14251.pdf (accessed January 2013).

[20] Progress Software, ObjectStore, http://www.progress.com/en/objectstore/index.html (accessed January 2013).

[21] M. Stonebraker, S. Madden, D. J. Abadi, S. Haziropoulos, N. Hachem, and P. Helland. "The End of an Architectural Era (It's Time for a Complete Rewrite)", International Conference on Very Large Data Bases, VLDB, 2007, pp. 1150-1160.

[22] Teradata Aster, http://www.asterdata.com (accessed January 2013).

[23] Versant Object Database, http://www.versant.com/products/versant-object-database (accessed January 2013).

Received: February 2013. Accepted: July 2013.