2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems

# Database descriptors: laying the path to commodity web data services

Rodrigo Dias Arruda Senra Claudia Bauzer Medeiros
Institute of Computing, University of Campinas, UNICAMP
Caixa Postal 6176 – 13084-971 – Campinas – SP – Brazil

E-mail: rsenra@acm.org, cmbm@ic.unicamp.br

## Abstract

*The growth of the Internet has dramatically changed the way information is accessed and managed. The Web contains an ever growing amount of distributed, semi-structured and uncontrolled data. In this new context, we should rethink how applications couple with DBMSs. Corporate intranets allowed a tiered coupling between applications and databases. However, that model is still too constrained, and unable to accommodate the hostility, unsafety and fast pace of the Web environment.*

*Web Applications soon, if not already, will seek to dynamically negotiate their relationship to distributed database services. Prior to accomplishing autonomous application_to_DBMS binding and seamless data migration, we need to devise a "lingua franca" to request and describe DBMS and database services and capabilities.*

*Database descriptors (DBDs) are a step towards this vision. This paper presents the motivation for DBDs, their structure and architecture, examples and a use case scenario.*

## 1. Introduction

We are interested in supporting seamless switching between applications and DBMSs. In the context of this paper, applications are any software artifact, and databases refer to *Database Management Systems* (DBMS). When an application switches from using a DBMS to another, data may also have to be migrated and transformed. We attack the problem in two stages. This paper is concerned with the first stage - mechanisms to support dynamic coupling - and assumes that, once this is achieved, appropriate mechanisms will be devised to migrate data, when needed (the second stage).

Today, applications are still conceived to be tightly coupled to a given DBMS instance. Such a tight coupling is the most feasible solution to implement, since such systems differ in terms of model, operations and interface. For instance, an application written to use a relational database must be refactored to use a different DBMS. When the underlying data models are different, the way data is structured and handled is radically different, such as an XML storage or an Object-oriented database. Even if two DBMSs support the same model, they may differ on the capabilities supported, such as temporal or spatial facilities, and the DBMSs may offer a different feature set.

The term *feature set*, in this paper, refers to a set of properties that include: data model, functional capabilities, access methods and API, performance and configuration settings. Applications can only switch from one DBMS to another if the target DBMS offers a feature set compatible to what is required by the application.

Relational DBMS already achieved a good degree of interchangeability through successful standardization efforts such as *Open Data Base Connectivity* (ODBC) from the X/Open consortium, or ISO's ANSI-SQL proposed in 1989 (and revised in 1992). However, there is room for improvement. The software industry continuously improves its products with extensions that transcend the relational model and fall out of the standardization efforts. As a consequence, applications that depend on non-standard extensions become enslaved to a particular product.

Success in designing and building DBMS database products will certainly introduce additional interoperability issues. For example, given a Web application in the emergent cloud computing scenario [4], as the user base demands scalability, the solution may be to switch from a single multi-purpose database to several special-purpose database-as-a-service (DaaS) approaches.

Several research efforts have been undertaken towards more flexible coupling between application requests for data and DBMS. Examples of such efforts include: n-tiered architectures, database federations, Web services and cloud computing. Each such initiative is based on some set of standards that determine, for example, how to invoke operations or how to encapsulate data.

978-0-7695-4005-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ECBS.2010.57

There are two basic scenarios to be considered. In the first scenario, an application requests data from several DBMS, and *may need further data* from another DBMS of a different nature, i.e. with a widely different feature set. In the second scenario, the application wants to *switch* from the initial set of DBMS to another (potentially different) set of DBMS. Here, it may have to depend on additional preprocessing operations – e.g., data conversion and migration from the original set to a new one.

These two scenarios introduce many research challenges. For instance, how to choose an adequate DBMS amongst several vendors? If changes in DBMS involves data migration, what is the effort and schedule involved? Would there be any collateral effects due to compatibility mismatches? And the bottom line – could all of these questions be answered and the migration be carried out automatically and seamlessly?

In order to attack this problem we propose the use of *database descriptors* (DBDs), which are data structures that describe the feature set of a DBMS and the requirements applications have in terms of DBMS support. From a high level point of view, an application A can switch from DBMS X to DBMS Y if, $DBD_A$ is compatible with $DBD_Y$.

In more detail, this paper proposes DBDs as a mechanism to describe the nature and capabilities of DBMS and application requirements. DBDs could be used to verify and validate the matching between application requirements and database capabilities, and ultimately be used as the foundation for dynamic negotiation and autonomous binding between applications and databases.

This paper is organized as follows. Section 2 introduces DBDs. Section 3 discusses situations in which they are needed. Section 4 provides a use case. Section 5 discusses a few major trends in related work. Section 6 concludes the paper.

## 2. Database Descriptors

The concept of *database descriptors* was originally presented by Madnick and Wang [14] in 1988 to describe something similar to a (relational) DBMS feature set. We extend this concept for new kinds of DBMS and applications, moreover accomodating it to the Web scenario.

The main goal behind constructing DBDs is to enforce a loose coupling between Applications and DBMS, that could help to: (i) ensure DBMS product/vendor independence, (ii) provide seamless cross-database migration, and (iii) support Applications and DaaS in the Cloud. Some of these goals depend on strategies to solve the schema integration problem. In this paper we are not focused on the data integration issue. Our main interest is to explore a mechanism that allows capability verification, validation and negotiation amongst applications and DBMS.

### 2.1 Basic Definitions and Architecture

We devise two types of *database descriptors*: desiderata descriptor and feature descriptor. The desiderata descriptor specifies what a client application needs (requirements) from a DBMS. The feature descriptor specifies the DBMS feature set. As we pointed out earlier, it refers to a set of properties that include: data model, functional capabilities, access methods and API, performance and configuration settings.

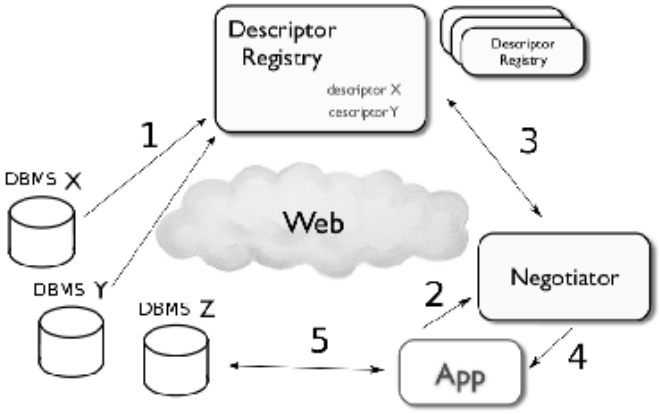


**Figure 1. Database Descriptor Architecture**

Assuming that DBDs are already available, there are many possible interaction patterns between applications and DBMSs. Figure 1 presents a generic architecture that serves as a reference for discussing interaction patterns.

The first step is taken when a given DBMS reifies its feature set as a feature descriptor. Considering that the feature descriptor is in digital form, it could be stored anywhere: as a file in the filesystem, as data in some DBMS or published in a Web page. We advocate the creation of registries in the Web. A descriptor registry consists of a publicly accessible repository specialized in storing DBDs. For instance, a minimal registry could be materialized as a Web page with links to pure-XML pages describing DBDs. Step 1 in Figure 1 illustrates DBMSs X and Y registering $DBD_X$ and $DBD_Y$ in a given registry.

The second step is taken when an application produces a desiderata descriptor that reflects its expectations in terms of a DBMS. The purpose of the desiderata descriptor is to be matched against the feature descriptors found in a registry. One possible approach is to make applications themselves responsible for discovering registries and carrying out the descriptor matching process.

We have chosen to introduce another element in the architecture called the negotiator, who is responsible for mediating the negotiation process, in which applications "ne-

gotiate" switching across DBMSs. Therefore, the negotiator can be an independent software artifact (such as a server), dwelling on the Web, and shared by potentially many applications. On the other hand, the negotiator can be a software module (such as a code library) embedded in the application or in the descriptor registry. Although we will refer to negotiator as an entity independent from the application to highlight its role, the architecture proposed is generic and accommodates different implementations.

We will not explore in this paper the trade-offs among the options to implement negotiators, either embedded in the application, in the registry or as an independent external mediator. We leave this topic to be explored in the future.

Step 2 in Figure 1 depicts application App presenting its desiderata descriptor $DBD_{App}$ to the negotiator. The negotiator should discover the available registries and run the descriptor matching algorithm (see Section 2.4) against them. This is represented by step 3. After the matching process, the resulting collection of feature descriptors is ranked by similarity with the desiderata descriptor and returned back to App in step 4. The process concludes with dynamically binding application and DBMS.

## 2.2 DBD Structure

The desiderata and the feature descriptors both share a common structure, composed by three distinct parts: metadata, dimensions, dimensional values.

The *metadata part* describes the descriptor itself, and we propose the adoption of a Dublin Core (DC) [1] subset. The following DC fields should be mandatory: identifier, format, date, creator, title and type.

The *dimensions part* are the DBMS properties described by the DBD, such as: connectivity, data model, type system, indexing resources, DDL/DML support, security, provenance, versioning, replication, scalability, etc. For each dimension mentioned in the DBD, there should be an associated *dimensional value*, which composes the third part.

The desiderata DBDs could have additional dimensions that express limitations or trade-offs from the application perspective, for example: prioritize up-to-date information, prioritize low latency for data delivery, enforce size constraints for result sets, determine quality of data (e.g., accuracy or completeness). Other (non-functional) requirements include privacy, security, costs, legal issues.

Desiderata descriptors could also include a fourth algorithmic part that specifies criteria for matching against feature descriptors. This algorithmic part can be represented by code embedded in the DBD, or it could be just a textual reference to some algorithm well known by the negotiators (further explained in Section 2.4). Once more, the trade-offs derived from these implementation choices are left for the future.

## 2.3 DBD Representation

The wide spectrum of the dimensions exemplified in Section 2.2 suggests that the DBD representation format should be extensible. We assume that distinct DBD instances will have a different collection of dimensions. As a result, the representation format and the matching algorithms should cope with partial information and heterogeneous structures. Given these constraints, XML might be a sound technological choice for representing DBDs. XML satisfies the heterogeneity condition but it is not sufficient for DBD representation, as explained by Wilde et al [20].

We propose, therefore, that DBDs be represented by the semantic annotations of [15, 13].

**Annotation Units**. An annotation unit a is a triple $< s, m, v >$, where $s$ is the subject being described, $m$ is the label of a metadata field and $v$ is its value or description.

**Annotation**. An annotation A is a set of one or more annotation units.

**Semantic Annotation Units**. A semantic annotation unit sa is a triple $< s, m, o >$, where $s$ is the subject being described, $m$ is the label of a metadata field and $o$ is a term from a domain ontology.

**Semantic Annotation**. A semantic annotation SA is a set of one or more semantic annotation units.

In fact, annotation units describe data using natural language; semantic annotations use ontology classes and can be processed by a machine. Since semantic annotations rely on ontologies, they provide the necessary interoperability basis to accommodate the needs of DBDs. However, there still remains the need to define the notion of *compatibility*, which is discussed next.

## 2.4 Matching DBDs

An application can couple to a DBMS if the former's desiderata descriptor matches the latter's feature descriptor. In an ideal world, both DBDs should be the same (i.e., identical metadata, dimensions and values). However, this restricts application-DBMS coupling. Thus, we have to look for more flexible matching criteria – e.g., borrowing notions from (i) programming languages or from (ii) content-based retrieval mechanisms in image databases.

In the first case (i), we can use an analogy from interface matching, where a subroutine invocation (message in object-oriented jargon) should match one function or procedure (resp., method) considering its context (i.e., namespace) and signature (i.e., name and parameters). Similarly, we can consider DBDs to represent signatures, where the dimension values are ontology terms.

Moreover, if we use semantic annotations, then matching can be performed using ontological relations. Borrowing from Santanche [18], two DBD values (A and B) are

considered equivalent if they refer to the same concept in an ontology (equal URIs), or if they point to two concepts related by OWL equality relationships (equivalentClass or sameAs). Moreover, A is said to be more general than B if A subsumes B and conversely B is more specific than A. For instance, if B is OWL subClass of A, or B is related with A through the partOf property (B partOf A), then A subsumes B. Consider A and B vertices of a graph, whose edges are properties. The subsumption relationship between A and B is a path formed by one or more edges. Therefore, the similarity rank value between A and B is inversely proportional to the number of edges which connect A and B in a subsumption relationship.

In the second case (ii), one can consider an analogy between DBDs and descriptors of images. From a high-level perspective, image similarity mechanisms are based on the notions of *feature descriptor* and *distance function* [19]. A feature descriptor is typically a set of values, organized according to some structure, that summarizes a given object (here, an image). Vectors are the most common structure used, and feature descriptors are therefore often called *feature vectors*. Two objects are considered similar if the distance between their descriptors is below some threshold. Similarity depends on the features selected – thus, distance functions are intimately associated with the algorithms used to create the vectors. Examples of distance functions involve *Manhattan* (also known as *L1*) and Euclidean – *L2*. If we borrow from this second kind of domain, then DBDs are our image feature descriptors and we can devise different distance functions (e.g., edit distance for each string in an annotation unit) to compare two DBDs.

We propose the adoption of the first definition, which borrows from interface matching in programming languages. Nevertheless, we point out that other matching mechanisms can be used.

## 3. DBD Example

Figure 2 exemplifies, from a high level point of view, a hypothetical DBD for an RDF DBMS. This example represents the feature DBD whose identifier is DBD1 (created in Dec 18, 2009 by Claudia). The dimensions and values indicate that the 2PL concurrency protocol is used, there is no versioning, storage uses RDF format, and the query language supported is RDQL. In order to represent DBD1 we have chosen the RDF data model rendered in XML markup language, which is acceptable in such a simple example. Posterior and more complete versions of DBD1 could be rendered in more expressive semantic languages, e.g. an OWL dialect (FULL/DL/Lite), depending on the need to express more accurately the identities, relationships and restrictions on the DBD dimensions.

Figure 3 presents a desiderata descriptor that matches

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:dbd="http://www.lis.ic.unicamp.br/purl/DBD">
  <rdf:Description rdf:about="http://www.lis.ic.unicamp.br/purl/DBD/DBD1">
    <!-- metadata -->
    <dc:creator>Claudia Bauzer Medeiros</dc:creator>
    <dc:description>Hypothetical DBD for an RDF DBMS</dc:description>
    <dc:identifier>DBD1</dc:identifier>
    <dc:format>application/rdf+xml</dc:format>
    <dc:type>
       <rdf:Description>
        <dbd:Type>Feature DBD</dbd:Type>
       </rdf:Description>
    </dc:type>
    <dc:title>Descriptor of an RDF DBMS</dc:title>
    <dc:date>2009-12-18</dc:date>
    <dc:language>EN</dc:language>
    <!-- dimensions and values  -->
    <dbd:concurrency>Two phase lock</dbd:concurrency>
    <dbd:versioning>unsupported</dbd:versioning>
    <dbd:storage>RDF triples</dbd:storage>
    <dbd:DML>
       <rdf:Bag>
          <rdf:li>RDQL</rdf:li>
          <rdf:li>SPARQL</rdf:li>
       </rdf:Bag>
    </dbd:DML>
  </rdf:Description>
</rdf:RDF>
```

**Figure 2. Feature DBD Example using annotations**

with the feature DBD exemplified in Figure 2. We point out that these examples are artificial, in the sense that feature and desiderata descriptors were both created by the same people. However, they serve the purpose of illustrating the look-and-feel of DBDs.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:dbd="http://www.lis.ic.unicamp.br/purl/DBD">
  <rdf:Description rdf:about="http://www.lis.ic.unicamp.br/purl/DBD/DBD1">
    <!-- metadata -->
    <dc:creator>Rodrigo Dias Arruda Senra</dc:creator>
    <dc:description>Desiderata DBD for an hypothetical application</dc:description>
    <dc:identifier>DBD2</dc:identifier>
    <dc:format>application/rdf+xml</dc:format>
    <dc:type>
       <rdf:Description>
        <dbd:Type>Desiderata DBD</dbd:Type>
       </rdf:Description>
    </dc:type>
    <dc:title>Desiderata descriptor of an hypothetical application</dc:title>
    <dc:date>2010-01-05</dc:date>
    <dc:language>EN</dc:language>
    <!-- dimensions and values  -->
    <dbd:concurrency>Two phase lock</dbd:concurrency>
    <dbd:storage>RDF triple store</dbd:storage>
    <dbd:DML>RDQL</dbd:DML>
  </rdf:Description>
</rdf:RDF>
```

**Figure 3. Desiderata DBD Example**

There are many other domains where DBDs are applicable. We just point out a few examples, to illustrate their utility. In *Online Transaction Processing Systems* (OLPTs) applications should require support for lots of small concurrent transactional workloads (e.g. debit/credit). In *Digital Libraries*, applications may demand for corpora text-indexing and, for more sophisticated libraries, multimedia indexing. In *Web Portals* there is an increasing demand for multimedia delivery (audio, image and video streaming) and the basic operation is to serve pages. *Scientific Grid* applications are interested in accessing voluminous data cubes and in number crunching.

In each of these situations, applications frequently may

have to migrate from one DBMS to another, and in all cases DBDs have to capture and express the applications' requirements and the DBMS capabilities in each of those domains.

## 4. Use Case

This section presents a hypothetical use case for DBDs. It is based on two real life projects in agro-environmental planning - the WebMAPS [16, 17] and the eFarms [13] projects, both conducted at the Laboratory of Information Systems of the Institute of Computing at UNICAMP.

The task to accurately create feature and desiderata DBDs is far from trivial. However, manually performing a compatibility analysis between an application and cloud DBMS services, and designing and executing a migration plan is not trivial either, not to mention time consuming. The purpose of this use case is to illustrate a concrete scenario where DBDs are useful, and show some of the difficulties involved in devising and applying DBDs. The DBD negotiation process presents its own set of relevant research challenges that fall out the scope of this use case discussion.

Consider a web application designed to support crop monitoring. This application handles three types of data: satellite images, farm/county geometries (coordinate sets), and time series of temperature and pluviosity measurements. Suppose the images and geometries are stored in the filesystem (as NetCDF and Shapefiles respectively), while the temporal series are stored in a relational database. The first prototype has been tested, and the application must now be released on the Web to end-users (farmers and agronomers). User demand is expected to scale from dozens to thousands of active sessions within a month's period.

Consider furthermore that we decided to adapt this application to use Internet-scale computing platforms, but we must choose from the available cloud computing storage services such as: Amazon's SimpleDB, S3 and Relational; Microsoft SQL Azure; or Google Data Services: Docs, Base and DataStore/BigTable.

The first challenge in this use case is to convey feature DBDs for these storage services – one just needs to compare such services to see that there are countless factors to take into account. There are, in fact, many dimensions to consider to properly describe each service – e.g., volume restrictions, pricing, scalability, data model, security. Comparing Amazon's SimpleDB and S3, both services provide: high availability, low latency, a key-value data model, a REST-based API and an access control lists (ACLs) security model. On the other hand, these two services differ in terms of volume restrictions and pricing. For example, S3 focuses on large raw data items (i.e. BLOBs), while SimpleDB focuses on small textual items (described by textual attributes) with implicit indexing. Considering that the pricing model mimics the data model and the latter is different amongst services, then the pricing model becomes harder to be compared automatically amongst services.

All of Google's Data Services provide: high availability, low latency and a REST-based API, though they differ in terms of data model, access restrictions and specific APIs. Google Docs is adequate to store textual documents and spreadsheets. Google Base is similar to Amazon's S3 – it is a web storage service for structured content as a set of descriptive attributes. Differently from S3, Google Base has no access restriction mechanisms, all items published are always publicly available. Google DataStore is a non-relational key-value-based web service DBMS that is part of the App Engine development stack, and it is more adequate for request-oriented applications (optimized for read operations). BigTable is a sparse, distributed, persistent multi-dimensional sorted map. It is the technology that lies underneath the DataStore service, and was built with access restrictions that prevent careless or malicious users from causing a query overload. For instance, no query can use an inequality operator $(<, <=, >=, >, ! =)$ on more than one property (a.k.a field) and the result sets are limited to 1000 entries.

Microsoft SQL Azure Database is a cloud-based relational database service built on SQL Server technologies. It provides a highly available, fault tolerant, scalable, multi-tenant database service. Amazon Relational Database Service (Amazon RDS) is a similar service, based on MySQL technology. These relational DBMS in the cloud are cost-efficient, resizable (in capacity), managed by the respective service providers (for time-consuming database administration tasks).

Another challenge in this use case is to materialize the desiderata DBD that describes our hypothetical crop monitoring web application's DBMS needs. There are two options: build a single desiderata DBD, or build three separate DBDs – one for each type of data handled by the application, assuming that it will be easier to find DBMSs that will handle some, but not all data types. In the former option, a single DBD would be expressing the wish to integrate different data sources in a single DBMS. In the latter option, three separate desiderata DBDs might suggest that data sources will be kept within isolated repositories. Moreover, it is important to reify several application requirements, such as access patterns (read and write) and indexing needs, relationships between its data types, and data model used by the data structures to be persisted.

## 5. Related Work

As pointed out in the introduction there are several initiatives to foster interoperability between applications and DBMS, from n-tiered architectures, passing through database federations, and reaching web services and cloud

computing.

DBDs provide a basis for self-describing DBMS, and as such can be seen as a means of structuring (unstructured) facilities of these DBMS. As such, they can be used within the so called UIMA (Unstructured Information Management Architecture) – http://docs.oasis-open.org/uima/v1.0/os/uima-spec-os.html. Unstructured information may be contrasted with the information in classic relational databases where the intended interpretation for every data field is explicitly encoded in the database by column headings – similar to a schema. Unstructured information represents the largest, most current and fastest growing source of knowledge available to businesses and governments worldwide. For unstructured information to be processed by applications that rely on specific semantics, it must be first analyzed to assign application-specific semantics to the unstructured content. The added headings structure provides an initial support to deriving such semantics. By the same token, DBDs provide a high level description of a DBMS (and of application requirements), in which attributes (dimensions) can be used to derive semantics.

We believe that DBDs are specially well suited to the cloud computing scenario [21]. The Open Cloud Manifesto [2] states that: (i) The challenges to cloud adoption are addressed through open collaboration and the appropriate use of standards. (ii) Cloud providers must not use their market position to lock customers into their particular platform. (iii) Cloud providers must use and adopt existing standards. In this scenario, DBMS can be seen as a special kind of provider within the cloud, and DBDs can describe their features, thereby enabling applications to look for the appropriate databases, with help from the repositories.

The Claremont Report on Database Research [3] states that a significant long-term research goal is to transition from managing schemata-based structured data to the managing of structured, semi-structured and unstructured data spread over many repositories in the enterprise and on the Web. This is referred to as the challenge of managing dataspaces. Again, DBDs can be seen as a kind of high level descriptor of dataspaces.

Federations and integration are two facets of the problem of enabling applications to access heterogeneous data sets. Federations [5, 11, 10] allow applications to access distinct DBMS via some kind of mediation layer, which concentrates the "intelligence" needed to transform an application request into a sequence of requests to federation members. The integration approach, on the other hand, assumes that data or schemata have to be adapted, in order to provide a unifying view to all applications. In the same vein, Haas has introduced a model for information integration [8, 9] that consists of four phases: Understanding, Standardization, Specification, Execution. The *Execution* phase is divided in: Materialization (ETL, replication), Virtualization (Federation), and Search. We believe that DBDs are orthogonal to all of these phases, meaning that they could be used to describe them as a whole or separately. Our initial focus is not on schema integration, nor data cleansing, nor self-tuning [12] DBMS – rather, DBDs offer a means for DBMS self-description. In this sense, they can be used by mediators in a federation.

Last but not least, web services [6, 7] are another means to allow flexibility in application execution. Web services can encapsulate a DBMS, and serve as a layer that receives requests for data and returns the appropriate data. In this sense, instead of looking for the appropriate feature DBD, an application could look for appropriate services, and request for data invoking these services, according to standard protocols (e.g., SOAP). Finding the appropriate services would also require looking for service directories (as opposed to looking for matching DBDs in a DBD registry). This offers the advantage of not requiring the specification of feature and desiderata DBDs, nor requires negotiation; on the other hand, this demands extending web service capabilities beyond those normally found – e.g., to accomodate versioning or concurrency requests.

## 6. Conclusions and future directions

This paper presented the concept of database descriptors (DBDs). DBDs could be the foundational bricks to build dataspaces, becoming an indispensable tool to allow applications to switch across DBMSs, in a loosely coupled scenario.

DBDs can thus contribute to help to commoditize data services in the cloud, by supporting dynamic switching between DBMSs and applications. Moreover, they can also be seen as a different way of tackling the information integration problem, from a connectivity point of view, in which applications and DBMSs can negotiate their coupling.

This work is part of our initiatives towards interoperability in an eScience context. Future directions include: to create a catalog of concrete feature descriptors, to design a descriptor negotiation framework, to do a proof-of-concept implementation, with real DBMS products and services.

## Acknowledgments

We would like to thank the Brazilian National Institute of Science and Technology INCT de Ciência na Web (Web Science), and the support from Brazilian financing agencies CNPq, CAPES and FAPESP. This research was partially financed by the BioCORE project, and the Fapesp-Microsoft Research Virtual Institute (eFarms project). We would also like to thank Karin Breitman and Roy Sterritt for pointing out relevant references related to this work.

## References


[1] Dublin core metadata initiative. http://dublincore.org/, 2009.

[2] http://www.opencloudmanifesto.org/open Web, 2009.

[3] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, and G. Weikum. The claremont report on database research. *SIGMOD Rec.*, 37(3):9–19, 2008.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[5] S. Berger and M. Schrefl. From federated databases to a federated data warehouse system. *Hawaii International Conference on System Sciences*, 0:394, 2008.

[6] E. Cerami. *Web Services Essentials*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[7] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6:86–93, 2002.

[8] L. Haas. Beauty and the beast: The theory and practice of information integration. *ICDT*, 2007:28–43, 2007.

[9] L. M. Haas, M. Hentschel, D. Kossmann, and R. J. Miller. Schema and data: A holistic approach to mapping, resolution and fusion in information integration. In *ER*, pages 27–40, 2009.

[10] D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Trans. Inf. Syst.*, 3(3):253–278, 1985.

[11] D. Jonscher and K. R. Dittrich. An approach for building secure database federations. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 24–35, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[12] V. Kriakov, G. Kollios, and A. Delis. Self-tuning management of update-intensive multidimensional data in clusters of workstations. *The VLDB Journal*, 18(3):739–764, 2009.

[13] C. G. N. Macário and C. B. Medeiros. Specification of a framework for semantic annotation of geospatial data on the web. *SIGSPATIAL Special*, 1(1):27–32, 2009.

[14] S. E. Madnick and Y. R. Wang. Evolution towards strategic applications of databases through composite information systems. *J. Manage. Inf. Syst.*, 5(2):5–22, 1988.

[15] G. Z. Pastorello, Jr, J. Daltio, and C. B. Medeiros. A mechanism for propagation of semantic annotations of multimedia content. *Journal of Multimedia*, 2010. Accepted for publication.

[16] G. Z. Pastorello, Jr, R. D. A. Senra, and C. B. Medeiros. Bridging the gap between geospatial resource providers and model developers. In *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–4, New York, NY, USA, 2008. ACM.

[17] G. Z. Pastorello, Jr, R. D. A. Senra, and C. B. Medeiros. A standards-based framework to foster geospatial data and process interoperability. *Journal of the Brazilian Computer Society*, 15(1):13–26, March 2009.

[18] A. Santanchè. *The Fluid Web and Digital Content Components: from a document-centered to a content-centered view*. PhD thesis, Institute of Computing - Unicamp, August 2006.

[19] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *Transactions on Pattern Analysis and Machine Intelligence*, 22:1349–1380, December 2000.

[20] E. Wilde and R. J. Glushko. Xml fever. *Commun. ACM*, 51(7):40–46, 2008.

[21] L. Youseff, M. Butrico, and D. D. Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, 2008.