Taming the Complexity of Big Data Multi-Cloud Applications with Models

Marcos Aurélio Almeida da Silva[1], Andrey Sadovykh[1], Alessandra Bagnato[1], Etienne Brosse[1]

[1] R&D Department, SOFTEAM, 9 Parc Ariane, Guyancourt, France
marcos.almeida@softeam.fr, andrey.sadovykh@softeam.fr, alessandra.bagnato@softeam.fr, etienne.brosse@softeam.fr

Abstract. Private and public clouds are getting more and more common. With them comes the need to analyze data stored by different applications in different clouds. Different clouds and applications tend to enforce the use of different data stores, which makes it even harder to aggregate information. The main outcome is that integrating different data sources requires deep knowledge on how data is stored on each solution and on the trade-offs involved in moving from one system to another. This paper is part of the ongoing work on the JUNIPER FP7 EU project (http://www.juniper-project.org/). In that project we explore the power of modelling tools to simplify the design of industrial big data applications. In the present work we present an overview of our approach and its application on a simple case study.

1 Introduction

The advent of cloud computing and the multiplication of computing and storage power available to companies gave rise to the so-called **big data cloud applications**. The multiplication of cloud offers also lead to a fragmentation in the capabilities of different providers [1]. Similarly, the multiplication of data management tools lead to the fragmentation of the data representation paradigms.

Concerning the fragmentation of cloud providers. On the one hand, some companies are surely in a position to take advantage of that. On the long run, applications end up with a set of specialized applications, each of them based on a different stack of tools. The challenge to these companies is then in aggregating the data stored on different stores, on different providers and behind different systems to support their business decisions. This challenge stems from the fact that highly specialized developers are needed to deal with the different stacks and programming languages. Connecting them therefore becomes more expensive and complex the more languages and systems a company needs to integrate.

In this paper we investigate the strengths of **model driven engineering (MDE)** in such scenario. Model driven engineering in fact consists in using high level models to abstract the complexity in an application. MDE techniques excel particularly in dealing with multiple programming languages and frameworks [2] [3]. This is so because model transformations can encapsulate the complexity involved in the translation of

high level concepts into concepts in other languages, and therefore reduce the necessity of highly specialized developers and then the cost of integrating disperse data stores. Models are also useful when dealing with availability time and consistency constraints by means of model analysis. This is so because high level models on a single language are much easier to inspect and analyse (either automatically or manually) than multi-language code.

The main challenge in applying MDE to big data applications relies in **finding the right** abstract **modelling language** that is **rich enough** to abstract from the specific features of the connected platforms, but still **low level enough** to foster code generation. In the context of big data application this is even harder because, to the best of our knowledge, no language exists that includes both the concepts necessary to define the architecture of a cloud application along with its data streaming and analysis in a high level. This is therefore the main contribution of this paper.

In this paper we present the approach developed as part of the JUNIPER FP7 EU project. It consists in identifying a subset of the Unified Modelling Language (UML) and extending it with data analysis and processing concepts so that it is suitable to both defining the architecture of a big data cloud application and to generate code for it. Since the project is in the beginning of its second year, in this paper we present the initial insights and experiments behind the use of a MDE approach to support the design of multi-cloud big data applications. In order to avoid exposing sensitive details of the project use cases, in this paper we apply our approach to a similar application based on the same principles.

This paper is structured as follows: Section 2 presents an overview of multi-cloud big data applications and the requirements of a MDE approach for it, Section 3 presents the approach we put forward in this paper. Section 4 presents a case study involving a multi clouds big data application. Section 5 presents the related work and Section 6 finally concludes.This is your introduction.

2 Multi-cloud Big data applications & Model Driven Engineering

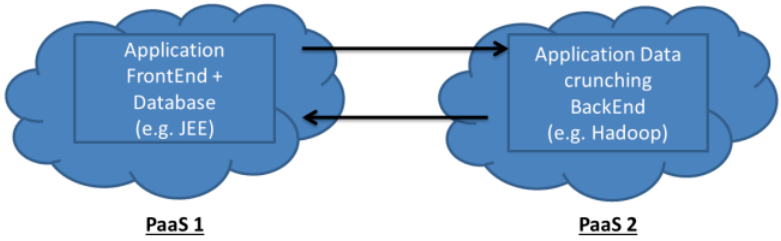

Fig. 1. Typical example of multi-cloud big data application.

Fig 1. illustrates a typical multi cloud big data application. It consists in leveraging the stronger points of different cloud offers to build a complex application. In this example, we consider a company that wants to leverage the cloud computing Platform As A Service (PaaS). The cloud application this company wants to build consists of a classic front end along with a database application and a specific back end for data

crunching. This backend is used to process the data in the application to deliver new insights.

The requirements in terms of short and long term data storage caps, response time and programming framework for each part of the application are different. That is therefore an **opportunity** for this company in using different PaaSes, in order to **optimize the costs** of providing the application.

The main challenge in seizing this opportunity is however in dealing with the **heterogeneity** of the cloud providers. In this specific example, the company would need to deal with the different programming languages and frameworks supported by each PaaS (e.g. Java JEE in the front end and Java Hadoop for the data crunching backend). While the application may be cheap to produce and deploy in the short term, the multiplicity of backend technologies may be a threat to its future maintenance [4].

In this paper we leverage a **model driven engineering** based approach to **seize this opportunity**. A MDE-based approach allows companies to increase the level of abstraction of architecture of the application by means of a model. This model can then be used to **simplify the development** by reducing the required familiarity of developers with the PaaSes frameworks and **maintenance** tasks by simplifying tasks such as moving data and code to other PaaSes.

There are however two challenges into coming up with a MDE approach for multi cloud big data applications:

1. **What language should we use to model applications?**
   The challenge here consists in using a language that includes the abstraction related to cloud applications, big data and the deployment of such application in multi-clouds.
2. **How to make sure this language is low level enough to foster code generation?**
   The challenge here lies in supporting as much as platforms as possible so that one can make sure that the models correspond to the effectively deployed applications.

3 A UML BASED MDE approach for big data cloud applications

3.1 General Approach

As explained in Section 2, the two challenges underlying the use of MDE approaches for handling the problem of designing multi-cloud applications are: (i) choosing a modelling language that is high-level enough to abstract from the concepts in different programming languages, and (ii) low level enough to allow for code generation. We chose the Unified Modelling Language [5] as modelling language since its object-oriented roots have been shown to be useful to model a wide range of problems while still serving as basis for code generation.

It's main drawback is however in the complexity of its specification, and therefore the steep learning curve that it represents to developers. Our approach to countering

this drawback consists in selecting a **subset** of UML suitable to address both challenges. The subset selected in this paper starts with the subset of the language usually reused by code generation tools [6] [7] [8] [9] [10].

However, reusing UML is not enough: one still needs to represent multi-cloud and big data specific concepts. In order to do that, we use the standard extension mechanism of UML called UML profiles. A UML profile allows one to add new concepts to the language by extending existing ones. On top of the extended language, we implemented code generators, so that the maps such abstract concepts into working code.

On the next section, we present the subset of UML we reuse, and the new concepts we add to it.

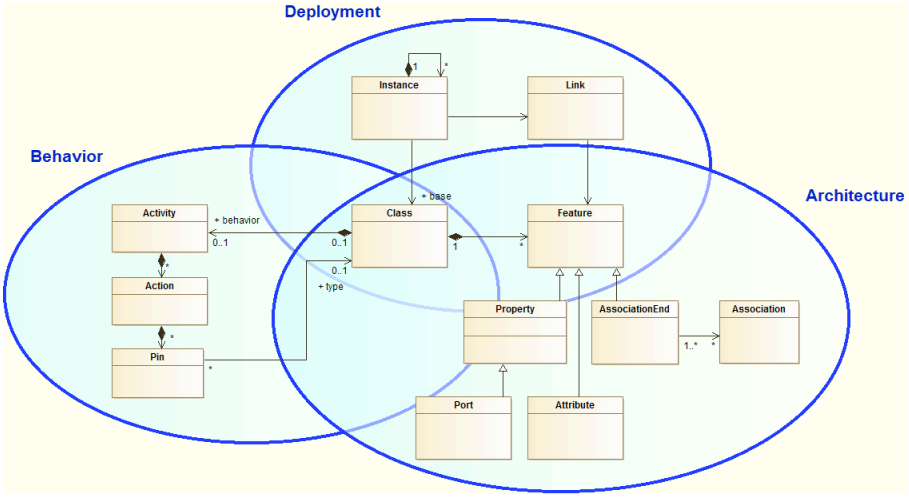

Fig. 2. Simplified overview of the reused UML subset.

3.2 UML for modelling multi-cloud applications & data

Fig 2. shows the subset of UML we reuse to model big data applications. In order to represent the **architecture of the application**, we reuse the concepts in UML class diagrams, i.e. applications are represented as `Classes` along with their `Properties` and `Associations`. `Classes` are also used to represent data types in a cloud provider independent way. Still when it comes to representing big data, one needs to represent the **data flows** that need to be implemented by the application.

We reuse the UML activity diagram concepts in order to represent the data flow, i.e. `Activities` which are broken down into atomic `Actions`. `Actions` have input and output `Pins`, which describe its input and output parameters.

The deployment of the application is represented by a set of UML object diagram concepts, featuring `Instances`, their relationships (`Links`) and their `base` types.

Stereotype	Input(s)	Output
filter, split, limit	Filtering expression.	Split data streams, or filtered data stream, or limited subset of the data stream.
generate	A set of streams and generation expression.	A data stream obtained by application of the generation expression to the input streams..
group	A data stream and a grouping criteria.	A data stream formed by groups obtained by application of the grouping criteria.
union	A set of data streams.	A single data stream containing both inputs.
cross	A set of data streams.	The cross product of the input streams.
inner-join, outer-join	A set of data streams and joining expressions.	A joined data stream.
sample	The size and type of sample to generate.	A randomly generated data stream.
order, distinct	A data stream.	An ordered data stream or a data stream containing only distinct elements.
load, store	A data stream.	Loads or saves the data stream to a persistency medium.

Fig. 3. Big Data flow language. Input and output parameters should be rendered as UML input and output Pins.

3.3 UML profile for representing complex big data flows

An object diagram is made **cloud specific** by conforming to a **fixed structure**. At the top level, instances represent **multiple cloud providers**. These instances contain other instances that represent either resources provided by the cloud provider in order to deploy parts of the application (most commonly on IaaSes) or parts of the application itself (most commonly on PaaSes). This way, object diagrams can represent the required resources from different clouds, and their interconnection.

It is important to notice that UML concepts are not sufficient to represent to the full extent the details of the architecture, deployment and behaviour of a cloud application. For the sake of brevity, in this paper we only present the concepts necessary to extend activity diagrams for representing complex big data flows. Extensions of UML for representing complex cloud application architecture and deployment are part of our ongoing work on the FP7 EU projects REMICS[1] and MODAClouds[2].

The UML activity diagram concepts are however not sufficient to represent all big data flow concepts. That is why we extended them. Some of the added concepts are represented in **Fig. 3**. These stereotypes were based on the concepts behind the PigLatin language [11], which abstracts the data processing works on top of the Ha-

[1] http://www.remics.eu/

[2] http://www.modaclouds.eu/

doop framework. They extend the UML basic concept of `Action` providing a big data flow processing specific semantic to them.

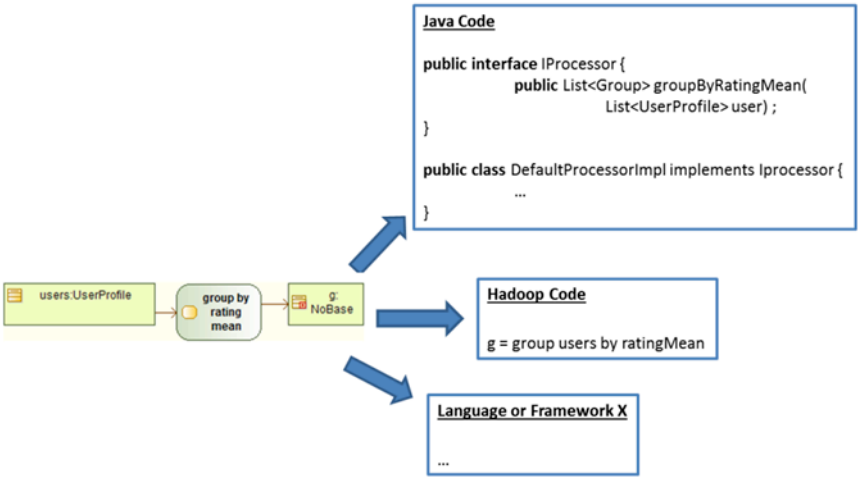

Fig. 4. Code generation approach.

As we discussed in Section 2, when it comes to supporting multi-cloud big data applications, being able to support code generation is as important as supporting modelling of such systems. **Fig. 4** overviews our code generation approach aiming to achieve this objective.

The code generation approach consists in providing cloud generators upfront, along with the definition of the language. In our present case, we experimented with code generation for Java EE cloud applications and Hadoop PigLatin map reduce data flows. As illustrated in **Fig. 4**, we use the same UML model, along with the provided transformations to generate both target languages.

4 Case Study

In this section we put the models and transformations we defined in the previous section into action. Notice however that this is a proof of concept that illustrates the work that is being performed on the FP7 EU project JUNIPER case studies. To avoid publishing sensitive information, we base the present case study in a cloud application that can be found in the literature. It consists in the MiC application (Meeting in the Cloud) [12]. The modelling and code generation tools presented here were implemented using the Modelio modelling tool[3].

The MiC application is a social network which allows users to maintain user profiles in which they register they topics of interests. The MiC application then groups

[3] Modelio, the open source modelling environment. Website: http://www.modelio.org

users by similarity, allowing users to interact with their "best contacts", based on ratings provided by each user in their profiles.

The use case we will analyse here is the one of a company that intends to provide different levels of service for different categories of users. Some users have paying accounts while other have free accounts. The company providing the MiC service wants then to support updates to the similarity computing service to paying users as fast as possible. In order to do that, it will use a IaaS cloud provider to compute the similarity so that the resources of the provider may adapt to the number of user requests and therefore absorb any increase in demand. For free account users, the company will resort to a low cost PaaS. This will lead to minimum cost and management costs.

The challenge is that the selected clouds support different platforms: on the IaaS the company may use any technology, while on the low cost PaaS, it needs to use the Hadoop crunching platform as a limitation of the PaaS.

This section intends to show how a MDE approach can help the company in designing the application so that it can be deployed on both clouds. This section is divided into two sections, in the first one, we show the UML models that describe the MiC applications and the second the code generation techniques we developed from these models.

4.1 UML Models

Fig. 7 and **Fig. 6** display part of the UML models that describe the MiC application. **Fig. 7** is divided into three parts:

- The **data model** is centred on the `UserProfile` class that represents a profile on the system, and stores the `UserRatings` provided by the user and the `UserSimilarity` which group profiles by similar ratings.
- The **architecture model** shows that there are basically two components in the application: the `CRUD` which is responsible for displaying the CRUD user interface and the `SimilarityCimputer` to update similarity of the users.
- Finally, the **deployment model** states that part of the application is deployed on `iaas1` and part of it on `paas1`. The PaaS is used to compute the similarity of part of the user base.

Fig. 6 models part of the behaviour of the `SimilarityComputer` component. It consists in loading user profiles and ratings from the data base, joining them, computing the means of the ratings and then grouping profiles by similarity.

4.2 Code Generation

Fig. 8 displays the generated code for the model described in the previous subsection. On the right side we see the code generated for the Hadoop cloud and on the left side we see the code generated for the Java EE Cloud. Thanks to the MDE approach, developers can be sure to find the same behaviour on both clouds. Data types and

structures are translated accordingly and the code generation makes sure that the data flow of both implementations is similar.

In case a new cloud platform is added on the future, developers will need to spend less time re-implementing the same data flow and data structures on the new cloud.

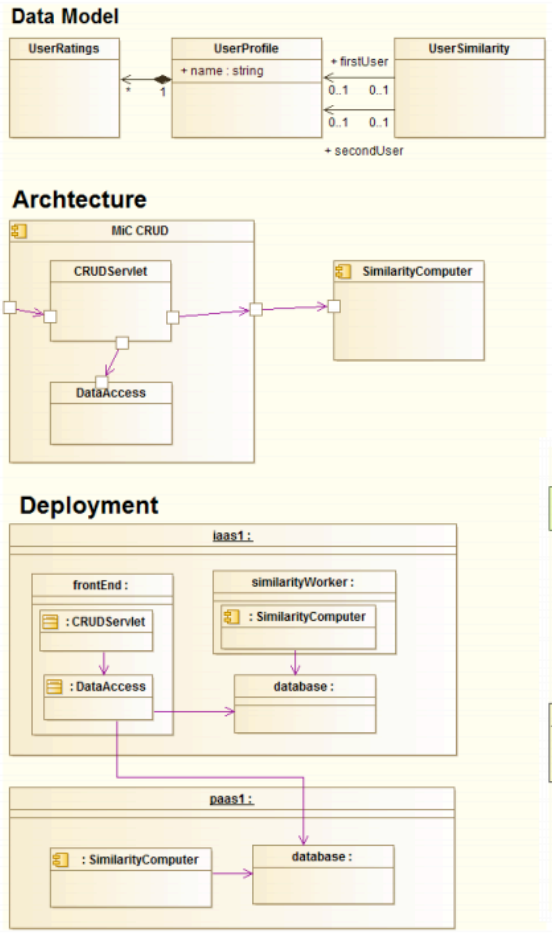

Fig. 7. Overview of the MiC Application models

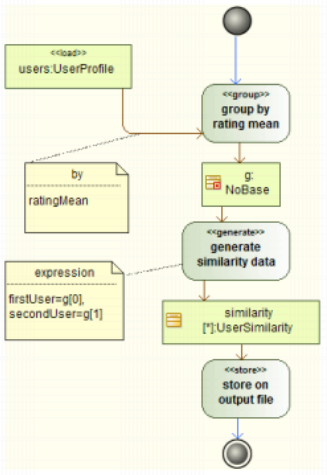

Fig. 6. Similarity Computer Behaviour model

Code for Java EE Cloud

```
@Entity
public class UserProfile {
…
}
@Entity
public class UserRatings {
…
}
public class ProcessSimilarity {
…
   List<Group> groupByRatingMean(
List<UserProfile> users) {
   …
```

Code for Hadoop Cloud

```
users = load 'input.csv' using PigStor-
age(',') s
   (logn:chararray,
      meanRating:float);

g = group users by ratingMean;

similarity = foreach g
   generate (first);

store similarity
   into 'output.csv'
               using PigStorage(',');
```

Fig. 8. Generated Code.

5 Related Work

In this section we are going to study similar works in the literature that are used to model applications, either related to the high level architecture of the application, its deployment or data. We are also going to analyse their support concerning the generation of running code from the models, and therefore of being suitable to a MDE approach for big data multi-cloud applications.

5.1 Architecture modelling languages

Languages such as SoaML [13]and SoaMF [14] are used to define the high level architecture of cloud applications. SoaML defines a MOF metamodel and a UML profile while SOMF defines a completely new language for defining service related concepts. It reuses and extends the UML concepts of components and ports to define respectively the services and their interfaces. SOMF also includes a sublanguage called Cloud Computing Modelling Notation (CCMN), whose concepts include IaaS, PaaS and SaaS clouds, and clouds of clouds; and service orchestration based.

The main weak point of these languages is that they are not intended to support code generation neither the data types manipulated by the application, they are limited therefore on high level concepts.

5.2 Cloud deployment languages

Some languages focus on providing "DevOps" tools such as Chef [15], Puppet [16] and CloudML [17]. They intend to automate the deployment of applications and services, as well as the management of cloud capabilities. With visibility and control on both IaaS and PaaS levels, developers can exploit the peculiarities of cloud solutions at each level of the cloud stack. The main weak point of such languages is that they target DevOps, not developers, and therefore do not include the architecture and data types of the application, and their impact to its execution.

Another group of languages uses the existing UML deployment diagrams to model the physical distribution of data. As an example we have the work of S. Lujan-Mora and J. Trujillo [18]. They define a UML profile that they can use to specify the deployment of Data Warehouses, which can be considered a former kind of private clouds, and could be extended to potentially represent multi-cloud big data deployment. Similarly, data integration tools such as Pentaho [19] and Yahoo! Pipes [20] offer visual editors that allow one to describe the partitioning of data in different data stores.

In both categories of work, the data structures are considered but not the architecture of the application, neither the code generation is envisioned.

5.3 Data Modelling

Many tools and approaches exist to help data modellers and application developers to describe data models. Object oriented models can be produced with the help of languages like UML [5] or Entity Relationship models [21], and relational models can be produced with the help of the numerous UML Profiles for relational modelling [22] [23] [24] [25] [26]. Purely object oriented databases are however rarely used in practice when it comes to storing data. Translations between both paradigms were created with the purpose of facilitating the use of relational data stores by object oriented applications [22] [27] [28]. This comes with the drawback of the inherent loss of information in the translation process. As in the other cases, these languages do not support modelling the application platform neither its deployment on the cloud.

6 Conclusion

On the one hand the multiplication of cloud providers represents an opportunity to companies willing to reduce the costs of maintenance of cloud applications by choosing the set of providers that best adapts to the uses of the application. On the other hand, multiple cloud providers come with extra technical requirements on programming languages, data structures and framework support. Integrating data and applications from different clouds then becomes more and more expensive and complex as the number of cloud providers increases.

In this paper we presented the first steps in dealing with this problem by means of a MDE approach. The core of the approach consists in defining a language, based on a

UML subset, extended with big data analysis specific concepts that can be used to generate multi-cloud enabled code to aggregate data in different sources. This approach is going to be fully implemented in the foregoing year of the JUNIPER FP7 EU project and will be applied on two industrial case studies.

Acknowledgements

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8- 318763 (JUNIPER).

References

1. R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers," in *10th IEEE/ACM International Conference on Grid Computing*, 2009.
2. P. Baker, S. Loh and F. Weil, "Model-Driven Engineering in a Large Industrial Context: Motorola Case Study," in *Model Driven Engineering Languages and Systems*, Springer Berlin Heidelberg, 2005, pp. 476-491.
3. F. Fleurey, E. Breton, B. Baudry, A. Nicolas and J.-M. Jézéquel, "Model-Driven Engineering for Software Migration in a Large Industrial Context," in *Model Driven Engineering Languages and Systems*, Springer Berlin Heidelberg, 2007, pp. 482-497.
4. T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld and M. Jazayeri, "Challenges in software evolution," in *Eighth International Workshop on Principles of Software Evolution*, 2005.
5. OMG, "UML: OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.4.1," 2001.
6. "ArgoUML Code Generation Window," [Online]. Available: http://argouml.tigris.org/tours/classgen.html. [Accessed 14 January 2014].
7. [Online]. Available: http://www.nomagic.com/products/magicdraw.html.
8. "IBM," [Online]. Available: https://www-304.ibm.com/support/docview.wss?uid=swg21259513. [Accessed 14 January 2014].
9. "Modelio," [Online]. Available: http://www.modeliosoft.com/en/modelio-store/modules/generators/java-designer-open-source.html.
10. "Uml to Java Generator 2.0.2," [Online]. Available: http://marketplace.eclipse.org/content/uml-java-generator#.UtPwpfRDvfk. [Accessed 14 January 2014].
11. "Pig Latin Reference Manual," [Online]. Available: http://pig.apache.org/docs/r0.7.0/piglatin_ref1.html. [Accessed 14 January 2014].

12. G. F., L. D., S. Y. M., A. D. and D. N. E., "An Approach for the Development of Portable Applications on PaaS Clouds," in *Proceedings of the 3rd International Conference on Cloud Computing and Service Science (CLOSER 2013)*, 2013.
13. OMG, "Service oriented architecture Modeling Language (SoaML)," 2009..
14. B. Michael, "Introduction to Service-Oriented Modeling," in *Service-Oriented Modeling: Service Analysis*, Wiley & Sons.
15. "Chef," [Online]. Available: http://www.opscode.com/chef/.
16. [Online]. Available: https://puppetlabs.com/.
17. [Online]. Available: http://cloudml.org/.
18. J. T. Sergio Luján-Mora, "Physical Modeling of Data Warehouses Using UML Component and Deployment Diagrams: Design and Implementation Issues," *Database Management,* pp. 12-42, 2006.
19. [Online]. Available: http://www.pentaho.com/.
20. [Online]. Available: http://pipes.yahoo.com/pipes/.
21. C. Batini, S. Ceri and S. B. Navathe, Conceptual Database Design, an Entity-Relationship Approach, Benjamin and Cummings Publ. Co., 1992.
22. [Online]. Available: http://argouml-db.tigris.org/.
23. Rational, "The UML and Data Modeling," [Online]. Available: http://bit.ly/15hxqgj.
24. [Online]. Available: http://www.modeliosoft.com/en/modules/sqldesigner.html.
25. D. Gorni, "UML Data Modeling Profile," 2002. [Online]. Available: http://bit.ly/VYJVGw.
26. D. Silingas and S. Kaukenas, "Applying UML for Relational Data," 2004. [Online]. Available: http://bit.ly/VkrNtw.
27. "Relational Persistence for Java and .NET," [Online]. Available: http://hibernate.org.
28. Oracle, "Java™ Persistence 2.0, JSR 317".
29. S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News,* pp. 51-59, 2002.