Towards Cloud Mobile Hybrid Application Generation Using Semantically Enriched Domain Specific Languages

Ajith Ranabahu, Amit Sheth,
Ashwin Manjunatha, and Krishnaprasad Thirunarayan

Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis) Center
Wright State University, Dayton, Ohio 45435
{ajith,amit,ashwin,tkprasad}@knoesis.org,
http://knoesis.org/cloud

Abstract. The advancements in computing have resulted in a boom of cheap, ubiquitous, connected mobile devices as well as seemingly unlimited, utility style, pay as you go computing resources, commonly referred to as Cloud computing. Taking advantage of this computing landscape, however, has been hampered by the many heterogeneities that exist in the mobile space as well as the Cloud space.

This research attempts to introduce a disciplined methodology to develop Cloud-mobile hybrid applications by using a Domain Specific Language (DSL) centric approach to *generate* applications. A Cloud-mobile hybrid is an application that is split between a Cloud based back-end and a mobile device based front-end. We present *mobicloud*, our prototype system we built based on a DSL that is capable of developing these hybrid applications. This not only reduces the learning curve but also shields the developers from the native complexities of the target platforms. We also present our vision on propelling this research forward by enriching the DSLs with semantics. The high-level vision is outline in the ambitious Cirrocumulus project, the driving principle being *write once - run on any device.*

1 Introduction

Lately there have been interesting changes at both ends of the *spectrum of computing power*. On one end there has been a boom in mobile computing devices, fueled by fast growing communication networks. On the other end, there has been substantial growth in high-end data centers that offer cheap, on-demand and virtually unlimited computing resources, popularly named *Cloud computing.* In the backdrop of these advances in computing and the growth of data intensive domains such as social networks, a new class of applications have emerged taking advantage of not only the on-demand scalability of computing clouds but also the sophistication of current mobile computing devices.

This class of applications, named *cloud-mobile hybrids*, are characterized by the need for data-intensive computations or extreme scalability in the back-end and mobile device based front-ends. Figure 1 illustrates the structure of a cloud-mobile hybrid application.

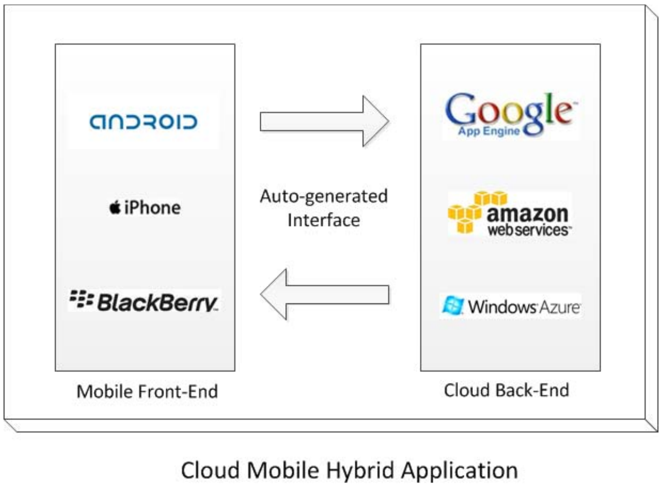


Fig. 1. Structure of a Cloud-Mobile hybrid

An illustrative example of a cloud-mobile hybrid would be an implementation of the *Privacy Score* [11] algorithm. Privacy score is a numerical indicator of the level of private details exposed by an individual in a social network. This score is a relative measure and requires substantial computations in the back-end. These computations can be performed in parallel. Presenting the score to the user is preferred to be via a mobile device, prompted by the increasing use of mobile devices to interact with social networks. Developing such an application, however, is significantly difficult than any other application development effort.

The state-of-the-art in mobile front-ends has changed from mobile-enabled Web sites to platform native applications. These native applications offer a better user experience by tightly integrating with the host platform and taking full advantage of the capabilities of the device. There is no universal development methodology and developers must pick and choose from a multitude of different mobile platforms. Similar choices need to be made in the Cloud space which is fragmented due to vendor specific service interfaces, restricted run-times and many others. Hence, developers have to cope with fragmentation at two different levels and often the efforts are focused on only selected mobile platforms and clouds. Developing portable hybrid applications in an economical and efficient manner is clearly a challenge.

We believe the key in overcoming portability issues is to follow a model-driven development pattern. However, there is no one level of abstraction that can be applied to modeling. Instead, one requires multiple models with varying granularities to cover different aspects of the application. Although semantic modeling is favored at a higher level, developers prefer detailed, concrete syntactic representations such as DSLs. The relationship between the representations of different granularities need to be established, often through explicit annotations. A slicing of the Cloud modeling space and the different types of models required have been discussed in [21] and showing the relevance of semantic models. We discuss four different types of semantics, data, functional, non-functional and system in Section 4.1 where each type addresses a specific aspect in an application. For example data semantics provide platform agnostic data definitions that support data portability.

In this paper we present MobiCloud, our early attempt to introduce a methodology for developing Cloud-mobile applications using DSLs. DSLs offer a mid-level abstraction that is developer friendly but also allows room for high level models to be attached. We also discuss the role of semantic models and the vision outlined by the ambitious Cirrocumulus project[1]. The goal of Cirrocumulus is to enable platform agnostic application development, deployment and management, intended to be achieved using DSLs infused with high-level semantic models as well as semantic-enriched middleware.

2 Motivation and Background

Our motivation for this research primarily comes from the lack of a clear methodology to develop portable applications for Clouds and mobile devices. The recent attention on Cloud-mobile hybrids and the difficulty in developing such applications clearly indicated the void in this space.

Portability issues arise primarily due to the heterogeneity (fragmentation) in both Cloud and mobile platforms. There is ample evidence that such heterogeneities exist and they are indeed the root of many of the issues the industry is facing today.

The Consumer Electronics Show (CES)[2] is the premier showcase of the consumer electronics devices and is indicative of trends in the current and future mobile device markets. During the last CES event, developers openly expressed frustration over a lack of consolidation of mobile platforms [10]. This is one of many complaints about the state of the fragmentation in the mobile space where there is no standardization in how applications are developed or deployed into mobile devices.

Similar fragmentation has happened in the Cloud space with each vendor developing their own paradigm [4]. The Cloud remains a largely non-standard space despite the efforts from National Institute of Standards and Technology (NIST) to standardize it. Recent industry surveys indicate that the practitioners still consider vendor lock-in a serious hindrance to Cloud computing adoption[19].

[1] http://knoesis.wright.edu/research/srl/projects/cirrocumulus/

[2] http://www.cesweb.org/

Some experts have also suggested that vendors may purposely promote the Cloud to be a heterogeneous patchwork of frameworks for business reasons [3].

These heterogeneities force developers to be locked-in to a selected set of platforms. Catering for multiple platforms is a time consuming and highly expensive venture only a few would attempt. This research focuses on overcoming the platform lock-in by using an independent language to develop applications.

2.1 Use of DSLs

A DSL is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [2]. DSL centric approaches have been used in many domains, particularly due to the expressiveness in the domain of interest, runtime efficiency and reliability due to the narrow focus [22]. For example, mathematicians are quite familiar with specialized languages such as MATLAB [8] that provide a convenient way to write matrix oriented programs. Domain of a DSL can be arbitrarily scoped, i.e. a DSL may cater for a generic domain such as Mathematics or be extremely narrow, say configurations for a particular computer game [23].

The emergence of powerful interpreted languages, such as Ruby, have been a key enabler for many modern DSLs. A Ruby based DSL has been successfully used in the IBM Sharable Code (ISC) project [14] to provide programming abstractions for light weight service compositions (a.k.a. mashups).

DSLs are considered as the key component in the software factories approach by Greenfield et al.[7]. Some of these philosophies have played a critical role in adding features to the Microsoft Visual Studio development suite. One of the pertinent arguments Greenfield presents is that many of the goals of Object Oriented Programming (OOP) were impossible to achieve in practice due to the lack of sufficient level of abstraction. A DSL is capable of raising the level of abstraction to achieve convenience in developing and software reuse.

A DSL however is not the silver bullet that provide a universal solution. DSLs by definition, cater to only a specific domain and become inapplicable outside the targeted domain. For example, the IBM Sharable Code DSL is only useful to prepare service compositions. However, given a class of applications, a DSL greatly reduces the effort required to create programs and lowers the barriers to entry.

3 A DSL for Cloud Mobile Hybrids

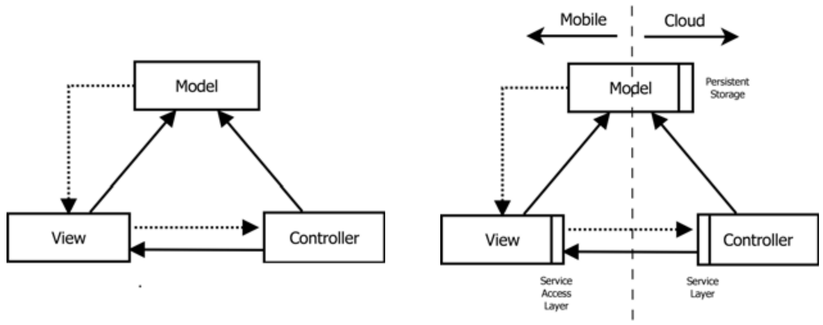
We now present the prototype DSL focused in this research. Named *MobiCloud* to indicate the presence across mobile and Cloud spaces, the DSL caters for interactive Web applications driven by Create, Retrieve, Update and Delete (CRUD) operations. These applications typically use multiple data structures in a data centric back-end and use a mobile or Web based front-end to manipulate these data structures. The use of Cloud in these applications is primarily for scalability, i.e., the application itself may not require a massive processing capability but is likely to receive a large number of simultaneous requests and hence needs to scale accordingly. Typically, these can be horizontally scaled, i.e. the load can be shared across multiple replicas.

An example of such an application is a *to-do list manager* similar to the very popular task manager application offered by *Remember the Milk*[3]. This application allows users to create *to-do items* using their mobile devices and stores them in a Cloud data store. These reminders can later be retrieved as a list, either on a mobile device or on the Web. Creating an application of this nature from scratch requires developing the following components:

(1) A data storage mechanism tied to the storage technology of choice.
(2) A service layer capable of exposing the operations on the data store.
(3) A service access layer in the targeted front-end capable of accessing the services defined on the server side.
(4) Relevant user front-end components.

Long running software engineering research on design patterns has identified the most appropriate design pattern for this type of applications is the Model-View-Controller (MVC) pattern. Figure 2(a) illustrates the major components present in a MVC based design. Figure 2(b) illustrates the split of these components across the back-end and the front-end.

We designed a DSL that closely resembles the MVC pattern giving separate specifications of each of the major components. This has been a conscious design decision since many developers are already familiar with the MVC pattern thus it would be natural for them to use this DSL.



(a) The Model View Controller (MVC) Design Pattern

(b) Splitting Model, View and Controller across the front-end and back-end

Fig. 2. MVC Design Pattern and the its usage for Cloud-Mobile Hybrids

We now present a *hello world* application written using this DSL to exemplify the features of the language. Listing 1.1 depicts the DSL script for this application. The intention of this application is to *illustrate* the main language features.

[3] http://www.rememberthemilk.com/

(1) A minimal *model* with only one attribute.
(2) A minimal *controller* with only one action.
(3) A minimal *view* demonstrating a minimal user interface.

This application displays a greeting message on the mobile device by fetching it from the remote, cloud based data storage via a RESTful service interface.

Listing 1.1. The DSL script for the *hello world* application

```
recipe :helloworld do
  metadata :id => 'helloworld-app'
    # models
     model :greeting,
                {:message => :string}

    #controllers
     controller :sayhello do
      action :retrieve,:greeting
    end

   # views
     view :show_greeting,
     {:models =>[:greeting],
      :controller => :sayhello,
      :action => :retrieve}
 end
```

We limit the elaboration on our DSL for brevity. Further details of the implementation of this prototype language is available from the MobiCloud technical report [12]. An on-line tool kit and a number of examples, including the complete language specification in BNF are also available [4]. The current system is capable of generating functionally equivalent back-end applications for Google Appengine and Amazon EC2. The front-end capabilities include Android 1.5 and Blackberry platforms.

4 Discussion

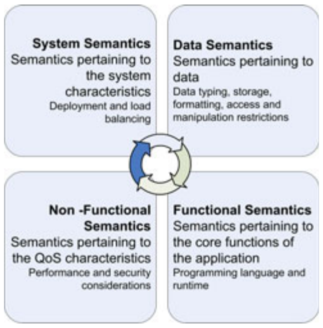
4.1 Role of Semantics

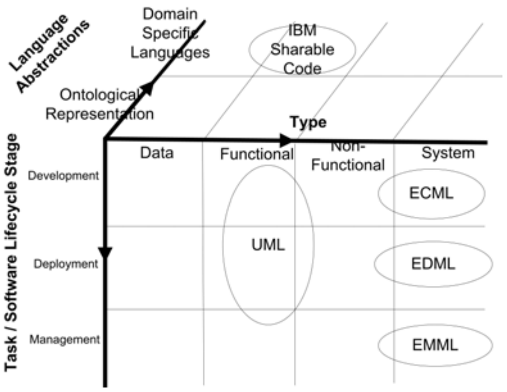
Semantic models have been applied in many domains to provide platform-neutral specifications. For example, our faceted classification and search system APIHut uses a taxonomy to organize the functional characteristics of Web APIs [6]. Another domain dependent example is GoodRelations, a standardized vocabulary for product, price, and company data [9].

[4] http://knoesis.org/mobicloud

Semantic Web community has been using semantic models to overcome issues of portability and interoperability for years and these are the very issues the Cloud computing community is facing today. A particularly relevant research work was on semantics for Web services where four types of semantics has been identified for a service [20]. Figure 3(a) illustrates an adaptation of the four types of semantics to the Web application domain.

Figure 3(b) illustrate the analysis of the modeling space we presented in [21]. This slicing indicates the applicability of the existing models as well as the voids that are present.


System Semantics
Semantics pertaining to the system characteristics
Deployment and load balancing
Data Semantics
Semantics pertaining to data
Data typing, storage, formatting, access and manipulation restrictions
Non -Functional Semantics
Semantics pertaining to the QoS characteristics
Performance and security considerations
Functional Semantics
Semantics pertaining to the core functions of the application
Programming language and runtime
(a) Four types of Semantics



(b) 3 dimensional slicing of the modeling space

Fig. 3. Four types of semantics and its relevance to the modeling space

The key in creating such a breakdown is to direct the model creation and usage towards specific aspects. For example semantic data models (ontologies) can be applied independent of the other aspects of the application. While the direct adaptation of the service oriented semantic categorization may not be the best in the application domain, a similar categorization would be immensely helpful in applying semantics to Cloud-mobile hybrids.

There are three potential uses of semantics in cloud-mobile hybrid application generation.

(1) A key limitation, even with the use of the DSL is reusable data modeling. This is very important in the back-end when the need to migrate applications arise. Existing clouds use a myriad of data models, making the task of migrating across data stores that follow different models a challenge. Model-agnostic semantic data definitions, coupled with the lifting-lowering data migration strategy [16] originally proposed for Web service data mediation is directly applicable here. A semantic data model can be referenced with in the DSL rather than defining one in-line. There are many well established, public semantic data models such as Friend-of-a-friend (FOAF) that can be reused in data definitions.
A mechanism to enable data references is illustrated in Listing 1.2.

(2) Non-functional details of an application are generally interleaved into the logic. However, many of these capabilities can be separated from the functionality and layered on the core functional implementations. Aspect Oriented Programming (AOP) [5] is a relatively new philosophy that advocates a clean separation of cross cutting non-functional concerns. Semantic models can be used to specify these non-functional capabilities and linked to the DSL via annotations.
(3) System details for the application including the deployment parameters and scaling configuration can be expressed via semantic models. In fact such descriptions are being used commercially today. Elastic Computing Modeling Language (ECML), Elastic Deployment Modeling Language (EDML) and Elastic Management Modeling Language (EMML) by Elastra Inc. [1], highlighted in Figure 3(b) is a prime example of a system oriented semantic model. This type of configuration may also be linked to the DSL via annotations.

Listing 1.2. Using a Reference to Define Data Types

```
model :person, {:ref => "foaf:Person"}
```

4.2 Application UI Features

A potential limitation of the current tool is the generic nature of the applications that are being generated. The generated UI's use minimal decorations and are focused on functionality, rather than visual appeal. Even if the generic UI features can be improved, developers may want to customize their application's visual components. There are two possible solutions:

(1) Use a secondary DSL to define custom UI components and attach them to the views. The XAML [15] UI language is one such well established DSL.
(2) Use the generated projects to bootstrap custom development. This is similar to the model driven development process followed by many major software companies where a high level model, such as UML diagram, is used to bootstrap the development process.

Listing 1.3 is a UI description written in XAML and Listing 1.4 shows how this could be incorporated in to the DSL.

Listing 1.3. An Example XAML template for the Greetings UI

```
<Canvas>
    <Rectangle  Fill="PowderBlue" />
    <TextBlock
      Foreground="Teal"
      FontFamily="Verdana"
      FontSize="18"
      FontWeight="Bold"
      Text="<%@model.message%>" />
</Canvas>
```

Listing 1.4. Using a Reference to XAML based UI template

```
view :show_greeting,
{:models =>[:greeting],
  :controller => :sayhello,
  :action => :retrieve,
  :uiref => "hello.xaml"}
```

4.3 User Defined Back-end Functions

Custom actions beyond the simple CRUD operations become an absolute necessity when the applications grow in complexity. Similar to the customization of the UI, this DSL may be enhanced to enable plug-in-in actions using user defined functions. These actions may also be written in other DSLs such as PIGLatin [18] scripts. Listing 1.5 shows a possible extension of the DSL to embed a PIGLatin script for a custom back-end function based on Apache Hadoop.

Listing 1.5. Embedding a PIGLatin script in a custom action

```
 action :sort_items,
         :item,{:lang => 'PIG'} do
     %{
          A=load 'items' using PigStorage()
                    as (a, b, c);
          B=sort A by a;
         }
 end
```

4.4 Deployment Complexity

Although the generated applications can be tested on the provided mobile device emulators, deployment to the actual device may require a signing step (using an authenticated key) and optionally an upload to a vendor controlled *app store.* Some of these work flows have been deliberately kept as human centric operations by the vendors. Even if there are Web APIs present, managing keys, security certificates and other deployment operations require the presence of a different layer of automation. Although such facilities are out of scope of this work, adding a middleware layer capable of managing deployments and subsequent management tasks, such as Altocumulus [13], would improve the reach and the usability of the DSL.

5 Vision for the Future

Our vision on the future of the Cloud applications indeed include DSLs as well as semantic-aware middleware layer [17]. The mobicloud DSL is the early attempt to realize this vision as outlined in Figure 4.

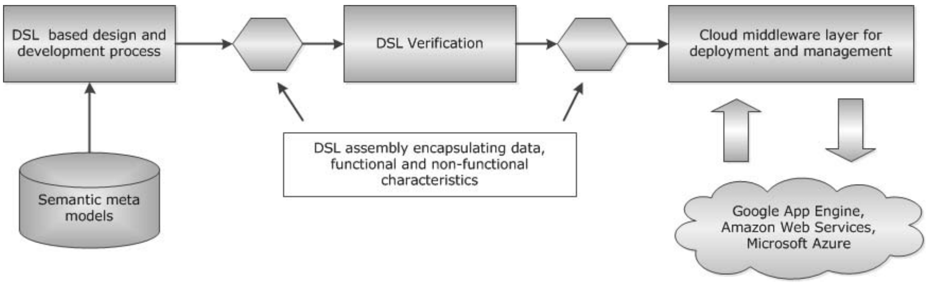

DSL based design and development process
DSL Verification
Cloud middleware layer for deployment and management
Semantic meta models
DSL assembly encapsulating data, functional and non-functional characteristics
Google App Engine, Amazon Web Services, Microsoft Azure

Fig. 4. The High Level Objective of the Cirrocumulus Project

The Cirrocumulus project attempts to provide the following capabilities to support portability and interoperability objectives.

(1) The ability to design and develop (program) with no assumptions about a specific target platform, data model or runtime behavior.
(2) The ability to deploy the artifacts to multiple platforms with no re-architecture or re-programming.
(3) The ability to manage and tune the deployed artifacts with no consideration of *where* they are deployed and migrate them to a different platform if necessary. *Management* refers to the tasks such as taking backups, moving log files etc.

Although these objectives were formed to cater for Cloud portability, they also apply to Cloud-mobile hybrids and to mobile application portability as well. MobiCloud system gave us an opportunity to *test the water* with respect to our development philosophy.

Several insights were gained from the feedback received on the MobiCloud on line tool kit.

(1) Some experienced developers considered the top-down design and development process not flexible enough to create presentable applications. This is indeed the case with high levels of abstractions. However, the default applications with the basic functionality would serve the majority case. Experienced developers can still use the DSL to generate the boiler plate code and continue to customize it as mentioned in Section 4.2.
(2) There was great interest in having a reverse engineering tool to convert an existing application to the DSL. Such a tool in practice would be semi-automated rather than fully automated. A conversion, even with human involvement, would bring value by enabling migrations at a later point and act as an incentive to convert existing programs to the DSL.

6 Conclusion

Our experimental DSL has clearly demonstrated the applicability of DSLs to generate cloud-mobile hybrids, as part of a larger goal of bring portability to

Cloud applications. Although there are many possible improvements, we believe that our philosophy is promising in transforming the Cloud-mobile hybrid application development process. By reusing many existing and well-established semantic technologies, this approach will be able to create, deploy, and manage Cloud-mobile hybrids efficiently and cost-effectively.

References

1. Charlton, S.: Model Driven Design and operations for the Cloud. In: Towards Best Practices in Cloud Computing Workshop, pp. 17–26 (2009), `http://bit.ly/cSPAin` (last accessed August 27, 2010)
2. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. SIGPLAN Not. 35(6), 26–36 (2000)
3. Durkee, D.: Why cloud computing will never be free. Communications of the ACM 53(5), 62–69 (2010)
4. Economist Opinion Section: Clash of the Clouds. The Economist (2009), published online at `http://bit.ly/cBRAfB` (last accessed August 27, 2010)
5. Elrad, T., Filman, R.E., Bader, A.: Aspect-oriented programming: Introduction. Communications of the ACM 44(10), 29–32 (2001)
6. Gomadam, K., Ranabahu, A., Nagarajan, M., Sheth, A.P., Verma, K.: A faceted classification based approach to search and rank web apis. In: IEEE International Conference on Web Services, pp. 177–184 (2008)
7. Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. In: Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 16–27. ACM (2003)
8. Hanselman, D., Littlefield, B.C.: Mastering MATLAB 5: A comprehensive tutorial and reference. Prentice Hall PTR, Upper Saddle River (1997)
9. Hepp, M.: GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 329–346. Springer, Heidelberg (2008)
10. Johnson, A.: Apps call, but will your phone answer?, published online, at `http://bit.ly/7OfKeO` (last accessed August 27, 2010)
11. Liu, K., Terzi, E.: A Framework for Computing the Privacy Scores of Users in Online Social Networks. In: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, pp. 288–297. IEEE Computer Society (2009)
12. Manjunatha, A., Ranabahu, A., Sheth, A., Thirunarayan, K.: A Domain Specific Language Based Method to Develop Cloud-Mobile Hybrid Applications. Tech. rep., Kno.e.sis Center, Wright State University (2010), `http://knoesis.wright.edu/library/publications/MobiCloud.pdf` (last accessed August 27, 2010)
13. Maximilien, E., Ranabahu, A., Engehausen, R., Anderson, L.: Toward cloud-agnostic middlewares. In: Proceeding of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, pp. 619–626. ACM (2009)
14. Maximilien, E.M., Ranabahu, A., Gomadam, K.: An Online Platform for Web APIs and Service Mashups. IEEE Internet Computing 12(5), 32–43 (2008)
15. Microsoft Corporation: Extensible Application Markup Language. Microsoft Developer Network, MSDN (2008)

16. Nagarajan, M., Verma, K., Sheth, A.P., Miller, J., Lathem, J.: Semantic Interoperability of Web Services-Challenges and Experiences. In: IEEE International Conference on Web Services (ICWS), pp. 373–382 (2006)
17. Oberle, D.: Semantic Management of Middleware (Semantic Web and Beyond: Computing for Human Experience). Springer-Verlag New York, Inc., Secaucus (2006)
18. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig Latin: A not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1099–1110. ACM (2008)
19. Rightscale.com: The Skinny on Cloud Lock-in (2009), published online at `http://bit.ly/LZc80` (last accessed August 27, 2010)
20. Sheth, A.: Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration. In: Workshop on E-Services and the Semantic Web (ESSW 2003) in 12th International World Wide Web (WWW) Conference, Budapest, Hungary (2003) (invited presentation)
21. Sheth, A., Ranabahu, A.: Semantic modeling for cloud computing, part 1. IEEE Internet Computing 14, 81–83 (2010)
22. Spinellis, D.: Notable design patterns for domain-specific languages. The Journal of Systems & Software 56(1), 91–99 (2001)
23. Sweeney, T.: Unreal Script Language Reference (1998), `http://unreal.epicgames.com/UnrealScript.html` (last retrieved August 27, 2010)