Prison Break: A Generic Schema Matching Solution to the Cloud Vendor Lock-in Problem

Mohammad Hamdaqa, Ladan Tahvildari
Software Technologies Applied Research (STAR) Group
Department of Electrical and Computer Engineering
University of Waterloo, Canada
{mhamdaqa, ltahvild}@uwaterloo.ca

***Abstract*—Porting applications from one cloud platform to another is difficult, making vendor lock-in a major impediment to cloud adoption. Model-driven engineering could be used to determine how applications might run on different platforms, if platform schemas could be matched. However, schema matching typically relies on linguistic and structural similarities, and cloud schema terms diverge so much that such matching is impossible. To address this challenge, we introduce Prison Break: a novel, semi-automated and generic schema matching process. Prison Break solves the divergent vocabulary problem by using web search results as a similarity metric, thus incorporating domain knowledge without constructing a dictionary, lexicon or thesaurus. We tested Prison Break by matching schemas from two major cloud providers: Windows Azure and Google Application Engine. We determined that Prison Break helps solve the vendor lock-in problem by reducing the manual efforts required to map complex correspondences between cloud schemas. This brings us one step closer to automatic model migration across cloud platforms.**

I. Introduction

In cloud computing, vendor lock-in refers to the dependency of an application on a particular cloud vendor [1]. It is hard to migrate to another vendor, due to the lack of standardized protocols, APIs, data structures, and service models [2]. This migration problem has previously been attacked using model driven engineering to transform models across platforms. However, appropriate mapping between the different model schemas must first be established [3].

The process of finding correspondences between different schemas is referred to as schema matching [4]. It is a well established practice [5], widely used for database migration and ontology consolidation [6], in domains such as linked-data, telecommunications, e-commerce, and bioinformatics [7]. However, while some attempts have been made to conceptualize or deploy schema matching in the cloud [8], we are unaware of any project that has successfully used it to mitigate cloud vendor lock-in. This is because most schema matching tools exploit simple mappings between schemas that share structurally or linguistically similar concepts. These tools do not provide generic solutions for executing complex mappings, where similar domain concepts may not be be identifiable via linguistic or structural similarities. Some advanced matching tools use dictionaries to incorporate domain knowledge [9]. Unfortunately, concept glossaries are not always available, and change constantly.

Complex mappings are the norm in the cloud. There are no industry-wide conventions for naming functions or services across cloud platforms. Proprietary marketing terms get used as the structural software terms instead. For instance, the terms "Role" in Windows Azure and "Module" in Google Application Engine (GAE) refer to the same underlying concept. However, these concepts do not have any linguistic similarity. Constructing a domain specific thesaurus can strengthen this kind of approach, but its applicability will be limited to those terms existing in the domain dictionary, and it will be fragile when change happens.

This paper presents a semi-automated schema matching process that solves the problem of obtaining domain knowledge and making the complex alignments required to facilitate model driven migration of the cloud service models between different providers. We call our approach "Prison Break" because it frees users from vendor lock-in. We answer the following research questions for complex cloud schemas:

RQ1. *Is it possible to pull domain knowledge into schema matching processes without sacrificing generality?*

RQ2. *How will this generic approach to schema matching perform in comparison to existing approaches?*

The answers are addressed throughout the paper, and highlighted in the conclusion. This research work makes the following contributions: (i) apply schema matching in the cloud domain to address the vendor lock-in problem, (ii) devise a generic schema matching process that uses web-search results as a similarity metric to find correspondences between similar concepts that may not share linguistic or semantic features.

Section II of this paper illustrates why schema matching is required in the cloud, and highlights the need for a generic approach to solving the mismatch problem. Section III illustrates how generic schema matching solutions may fail to address the mismatch problem in the cloud. Section IV presents our proposed solution, and how it has been realized. Section V evaluates it against competing alternatives. Section VI presents related work, followed by conclusions and suggestions for future work in Section VII.

II. Preliminaries

We begin our investigation by exploring the schema mismatch between two cloud providers; Microsoft Azure [10] and the Google Application Engine [11]. There are several ways cloud service providers as an industry might attempt to resolve these mismatches. Given that we cannot re-start the industry on a shared vocabulary from scratch, some kind of mismatch resolution is required. Therefore, automatic or semi-automatic schema matching has a role to play in making that process more manageable.

A. Mismatches at the Service Delivery Model

Deploying an application on a cloud platform requires specifying how the application service model will use the platform resources of that particular provider. This involves specifying a set of platform specific artifacts (i.e., *definition* and *configuration* files). Listings 1 and 2 show a definition and configuration files respectively that are used to deploy an application on Windows Azure platform. Likewise, Listings 3 and 4 show a deployment descriptor (i.e., definition file) and module configuration files that are required to deploy a Java application on Google Application Engine (GAE).

The service definition file in Listing 1 describes a cloud application that uses one *role*. A *role* in Windows Azure refers to a virtual appliance that is prepared with the required software stack to run a certain family of applications (i.e., web, or back-end). The service configuration file further specifies the service definition by assigning values to the configuration settings defined in the service definition file. For example, the service configuration in Listing 2 specifies the number of instances of the worker role, as well as a connection string (*DataConnection*) that points to a Windows Azure account to allow the role to access a cloud data store. Similarly, in GAE, the application descriptor file in Listing 3 declares the list of modules that comprise the application and the types of these modules (i.e., web, java, ejb and connector). Each of the modules defined in the deployment descriptor file is further specified in the module configuration file, which also specifies the scaling type and instance class.

At a glance, the Azure and GAE files seem to share several similar concepts with similar, or at most slightly different structures. For example, the concepts of a *module* and a *role* are very similar. Both represent a software process that has a type. The type specifies the family of applications the process belongs to (i.e., a module type in GAE, a role type in Azure). The number of instances for the modules and roles are specified in the configuration files. One of the structural differences at the file level between Azure and GAE application package specifications is that the resource requirements for roles are specified in the definition file by specifying the virtual machine size (e.g., *vmsize="Small"*), while the modules underlying instance specifications are specified in the configuration file by specifying the instance class (e.g., "*instance class = F4*"). In order to port the application from one provider to another, the mismatch between the different providers' models need to be resolved. In the next subsection we highlight three different approaches that can be used to resolve such a mismatch and create mappings between the different provider-specific concepts.

B. Potential Approaches to Solve Vendor Lock-in

The aforementioned examples are based on Windows Azure and GAE application packaging specifications. However, the information required to specify a cloud application deployment is essentially the same (e.g., in Amazon AWS the previously described role/module is equivalent to beanstalk). In order to deploy the same application on multiple providers and facilitate its migration, there is a need to identify the mapping between the deployment-description artifacts of the different providers. There are three different approaches to address this problem:

(a) The platonic standardization approach: A standardization body or industry consortium creates a unified modeling standard and enforces it on all providers. The advantage of this approach is that if applied correctly, it can guarantee a minimal set of concepts with few to no mismatches. Unfortunately, it is not realistic to expect that a standardization body might arise

```
<ServiceDefinition>
  <WorkerRole name="ShoppingCartProcessing" vmsize="
   Small"=>
    <ConfigurationSettings>
      <Setting name.."DataConnection" />
    </ConfigurationSettings>
  </WorkerRole>
</ServiceDefinition>
```

Listing 1: Example of Service Definition File.

```
<ServiceConfiguration>
  <Role name="ShoppingCartProcessing">
  <Instances count="2" />
    <ConfigurationSettings>
      <Setting name="DataConnection"
        value="UseDevelopmentStorage=true" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

Listing 2: Example of Service Configuration File.

```
<application
  <description>Demo Java EE App</description>
  <display-name>My App</display-name>
  <module>
    <web>
      <web-uri>ShoppingCartProcessing</web-uri>
      <context-root>ShoppingCartProcessing</context-
       root>
    </web>
  </module>
</application>
```

Listing 3: Example of GAE Deployment Descriptor File.

```
<appengine-web-app
  <application>My App</application>
  <module>ShoppingCartProcessing</module>
  <version>uno</version>
  <threadsafe>true</true>
  <instance-class>F4</instance-class>
  <manual-scaling>
    <instances>5</instances>
  </manual-scaling>
</appengine-web-app>
</application>
```

Listing 4: Example of GAE Module Configuration File.

that is fully aware of all the cloud domain requirements, and has full control over the industry - especially in such a vibrant, competitive industry as cloud computing. It is just as unlikely that such a body will emerge and become dominant now as it was in earlier days when the cloud industry was born.

(b) The oligopoly approach: Providers create their own standards. Then, alignments are created to establish the mapping between the different standards. This is more realistic than the platonic approach as it relieves the providers from the constrains of following one standard. However, it suffers from several problems: (i) creating alignments manually is difficult; therefore, automatic schema matching is needed. Morover, (ii) automatic schema matching cannot identify all mismatches nor is it able to establish all mappings; human intervention is still required. Finally, (iii) the approach suffers from the combinatorial explosion problem. In which, there is a need to create mappings across every combination of the supported providers' schemas.

(c) The modest hybrid approach: Providers create their own standards. These standards along with domain knowledge are used to create a unified domain ontology/meta-model. Then, a set of schema matching techniques is used to establish mappings between the different providers' schemes and the unified domain ontology schema. This approach has the following advantages: (i) It addresses the combinatorial problem by facilitating model transformation to and from the unified domain ontology. This can improve application maintainability as changes in one model can easily propagate to other models. (ii) Even though it may use simple schemas (e.g., based on xsd) in their output format, domain ontologies represent higher level of abstraction and hence can deal with complex and hierarchical relationships, and support reasoning and validation of domain constraints. Despite all the advantages of the modest approach, it suffers from the same drawbacks as its predecessors: the creation of a unified domain ontology is a complex and time-consuming task. Moreover, automatic schema matching is an inaccurate and error prone process.

In a nutshell, schema matching is an essential component for the success of any of the approaches discussed to tackle the vendor lock-in problem. In particular, both the second and third approaches require implementing efficient schema matching processes. This paper focuses on this particular issue: how to semi-automate mappings between the different schemas. The next section defines the schema matching problem, explains the main approaches and tools used in schema matching, and highlights the challenges that may face practitioners who may want to use schema matching to address the cloud vendor lock-in problem.

III. Schema Matching in The Cloud

Despite the fact that schema matching has obvious utility for solving the vendor lock-in problem, it has not been well exploited within the cloud computing domain. Most existing approaches follow the traditional model driven approach in an attempt to create a reference domain model. This model is usually created manually. The goal of this paper is to address mismatch between the different cloud providers service models, by applying schema-matching techniques

Besides addressing the vendor lock-in problem by mapping the different providers service models, schema matching can be used for model evolution, to migrate the existing artifacts to newer versions when schemas are updated due to platform updates with the same provider. This section introduces the definition of schema matching, and highlights some of the challenges when adopting schema matching techniques in the cloud domain.

A. Schema Matching

The goal of schema matching is to find correspondences between entities of two schemas (s1, s2). As shown in Fig.1, schema matching techniques normally consist of two steps: *similarity analysis* and *elements mapping*. In similarity analysis, given two schemas (s1,s2), each element in s1 will be compared to the elements in s2. A similarity score $\sigma(e1, e2)$ will be calculated and normalized to be used in the final mapping process. The results of the similarity computation are usually presented in a similarity matrix. In which, each cell contains a measure of similarity between an element of s1 and an element of s2.

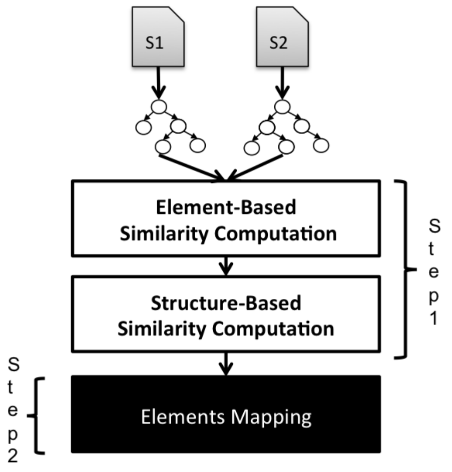

S1
S2
Element-Based Similarity Computation
Structure-Based Similarity Computation
Step 1
Step 2
Elements Mapping

Fig. 1: The Schema Matching Process

Generally speaking, the similarity analysis techniques used in schema matching can be classified into *element-based* and *structure-based* methods. Element-based techniques do not consider the relationship between the elements within the same schema. It is usually based on calculating the syntactic or semantic similarity between entities. On the other hand, structure-based techniques use the relationships between elements and apply graph or tree matching algorithms to analyze the structure of the schema. In most approaches, structural-based techniques are considered complementary, because these techniques are used to improve the overall matching results derived from element-based similarities [12]. The mapping process establishes the final matching between the elements of the two schemas by comparing the similarities and filtering the results. The most common mapping technique is threshold based filtering.

B. Problems With Schema Matching

As explained earlier, cloud models (schemas) are industry driven. They are derived form the specifications of their underlying platforms. Despite the similarity in domain concepts between these various platforms, in most cases there are huge discrepancies in the way they name these concepts. This is due to: (i) the lack of coordination between the providers at the early stages of the development of their platforms; (ii) the continuous updating of schemas due to the evolving feature sets of competitive offerings, and evolving customer needs; (iii) marketing campaigns and the need to differentiate from competitors through branding and positioning.

Purely linguistic based semantic similarity techniques fail terribly in matching elements across cloud schemas. Recall from Section II that the concept of "Role" in Azure is similar to "Module" in GAE and "Beanstalk" in AWS. When applying a linguistic pairwise similarity technique, such as *path length* [13] to discover the similarity between these concepts, we get the similarity matrix[1] in Table I. Path length similarity is a node-counting scheme, in which the relatedness score is inversely proportional to the number of nodes along the shortest path between the synsets. Path lengths generate results between zero and one, where one means identical concepts. Despite the strong domain similarity between Azure Roles and AWS Beanstalks, Table I shows very week linguistic semantic correlation. This could be even worse using other linguistic semantic techniques, such as Jiang & Conrath [13], where the similarity between a Role and Beanstalk equals zero. For this reason there is a need for techniques that captures domain similarity, even when two concepts do not share any linguistic syntactic or semantic similarity.

TABLE I: Path Length Pairwise Similarity

Terms	Role	Module	BeansTalk
Role	1	0.12	0.07
Module	0.12	1	0.10
BeansTalk	0.07	0.10	1

These issues matter whether we are attempting to map one platform schema to another, or all of them to an abstract reference model for the cloud domain.

IV. The Prison Break Approach

Fig. 2 illustrates an overall approach of model migration based on schema matching. Given two service model schemas and domain knowledge, a schema matching process is used to automate the generation of alignments between service models of the different cloud providers. A domain expert reviews the recommended model alignments and confirms or rejects them. The final alignments are used to migrate the service models between providers, or evolve the models from an older to a newer version within the same provider.

This paper focuses on the process of finding correspondences between the domain concepts. Two essential requirements should be considered in designing a schema matching process to address the vendor lock-in problem: (i) it should be generic with global applicability to be used with various providers' service models; (ii) it should be able to incorporate domain knowledge without sacrificing generality.

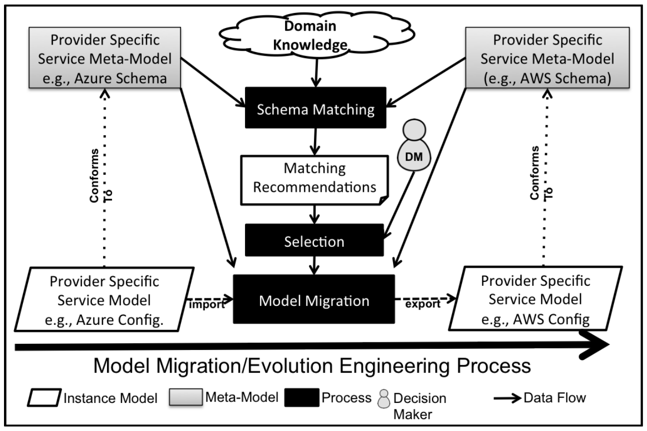

Domain Knowledge
Provider Specific Service Meta-Model e.g., Azure Schema
Provider Specific Service Meta-Model (e.g., AWS Schema)
Schema Matching
Matching Recommendations
DM
Selection
Conforms To
Conforms To
Provider Specific Service Model e.g., Azure Config.
import
Model Migration
export
Provider Specific Service Model e.g., AWS Config
Model Migration/Evolution Engineering Process
Instance Model
Meta-Model
Process
Decision Maker
Data Flow

Fig. 2: Model-Driven Migration Using Schema Matching

The schema matching approach proposed here follows the same generic schema matching approach introduced in Section III-A. However, it uses a Web Similarity Matcher as the pairwise matcher for calculating the similarity between each pair of elements of the source and target schemas based on web-search results.

A. The Web Similarity Matcher

To meet the aforementioned requirements, the approach proposed adopts a web search matching technique that is inspired by Normalized Google Distance (NGD) [14]. NGD is a measure of word relatedness that computes the degree to which one word is related to another based on their co-occurrence in search engine results [14]. NGD uses the number of hits returned by the Google search engine, however the metric can be used with the results obtained from other search engines. To avoid ambiguity, when using search results of other engines the NGD is sometimes referred to as the Normalized Web Distance ($\mathtt{NWD_{SE}}$), where SE refers to the name of the search engine used to obtain the results (e.g., Yahoo, Bing) [15]. NGD has been derived from the normalized information distance [16], which is motivated by Kolmogorov complexity [17]. Consequently, it measures the distance rather than the similarity. The distance can range from zero, which indicates 100% similarity to ∞ which means no similarity.

Equation (1) shows the NGD, where $\tau 1$ and $\tau 2$ are the two terms to compare; $NGD(\tau 1, \tau 2)$ is the distance between $\tau 1$ and $\tau 2$; $f(\tau 1)$,$f(\tau 2)$ is the page count for the terms $\tau 1$ and $\tau 2$; $f(\tau 1, \tau 2)$ is the page count for the query "$\tau 1$ AND $\tau 2$"; and finally, N is the total number of pages in Google index. As explained earlier, Equation (1) is not bounded, the output can range from 0 to ∞.

$$\mathrm{NGD}(\tau 1, \tau 2) = \frac{\max\{\log f(\tau 1),\log f(\tau 2)\} - \log f(\tau 1,\tau 2)}{\log N - \min\{\log f(\tau 1),\log f(\tau 2)\}} \quad (1)$$

In order to use the NGD as a semantic similarity measure, a transformation function is needed that can: (i) convert distance

[1]The similarity values in the matrix have been computed using the WordNet similarity web interface implementation by Ted Pedersen and Jason Michelizzi http://maraca.d.umn.edu/cgi-bin/similarity/similarity.cgi

into similarity, and (ii) normalize the unbounded results to values between 0 and 1. Morover, since most values in NGD fall between zero and one, it is desirable to have a function that falls rapidly to zero as the value of NGD reach the ∞.This research adopts a transformation function similar to the one used by Gracia et al. [15]. In this research, the exponential function $g(x) = e^{-x}$ is used in composition with the NGD. The exponential function has been selected as it satisfies the following properties:

- $(g \circ \mathrm{NGD}) : [0, \infty) \mapsto (0, 1]$
- $g(0) = 1$
- $\lim_{x \to \infty} g(x) = 0$
- $g(x)$ is decreasing, **if** $x1 > x2$ **then** $g(x1) < g(x2)$

Equation (2) shows the result of composing the exponent function after the normalized web distance ($g \circ \mathtt{NWD}$). We refer to this composition as the Web-Similarity-Metric (WSM). WSM is bounded between zero and one, where zero indicates no similarity and one is 100 % similarity.

$$\mathtt{WSM}_{\mathtt{SE}}(\tau 1, \tau 2) = e^{-\mathtt{NWD}_{\mathtt{SE}}(\tau 1, \tau 2)} \tag{2}$$

The value of this operation is calculated by making *three queries* to the search engine API, and retrieving the page counts in each case. For example, as of May 2014, the count of pages returned when querying for "Azure" on Google was 22,000,000, the count of pages when query for "AWS" was 34,600,000, and the count of pages of their co-occurrence was 2,390,000. Now, in order to calculate the WSM, one last value is needed, this is the total number of pages (N) in the search engine (e.g., Google) index. Some reference implementations of the NGD use hardwired values for this number (e.g., 30 trillion for Google as of 2014). This number changes continuously over time and is different from one search engine to another. A trick that can be used to deal with this challenge and make the approach general is first to query the search engine for the exact match of the term "the". Assuming that almost all pages on the Internet contain this term and the fact that there are about 1,000 search terms on the average page this gives the total page count (N). We use this result for the value of N. Based on these values NWD("Azure", "AWS") ≈ 0.2 and WSM("Azure", "AWS") ≈ 0.82.

TABLE II: NGD Pairwise Similarity

Terms	Azure Role	GAE Module	AWS BeansTalk
Azure Role	1	0.75	0.78
GAE Module	0.75	1	0.89
AWS BeansTalk	0.78	0.89	1

Table II shows the results obtained by applying the WSM to the same example in Table I. Each cell in Table II is the result of calculating the WSM between the two terms that intersect in that cell. For example the cell in the third column of the second row represents the outcome of the following operation: WSM("Azure Role", "AWS Beanstalk"). Note that for all the searches, we specify the domain name before the term to be searched (e.g., *Azure* Role instead of Role). Specifying the domain has significant impact on the WSM results, as it limits the search within that domain. In addition to the domain keywords (i.e., Azure, GAE, AWS), this research uses the time period and duplicate filters as tuning parameters in the search API calls in order to specify the domain context.

The results in Table II are promising; they show high correlation between the domain concepts, even when the concepts share no linguistic similarity. The evaluation section will explore this further.

The output of the node similarity matcher is a similarity matrix that shows the similarity between each element in the first schema and the elements of the second schema. The results of the similarity matrix are filtered based on a predefined threshold as to keep only elements with highest similarity.

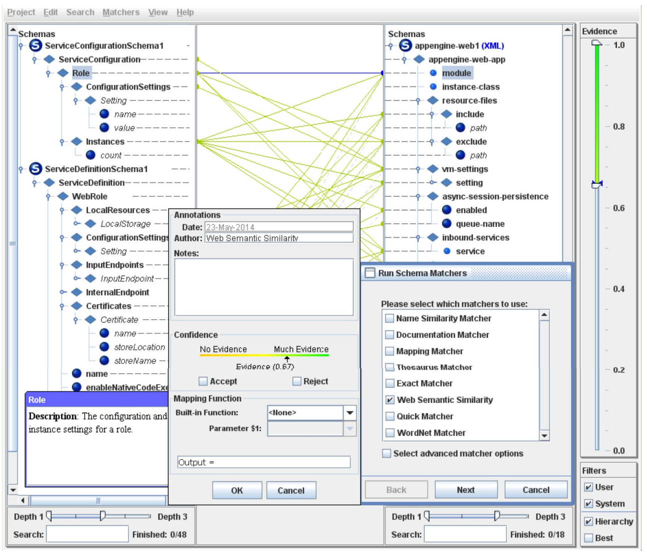
B. Implementation

The approach proposed in this paper has been realized as an extention to the OpenII Harmony framework [18] that is an open source schema matching tool. OpenII Harmony was selected because it offers many useful features, such as:

(i) **Extensibility**: Harmony has been designed in a modular fashion that makes it easy to build custom matchers tailored to a particular environment. To add new matcher all you need is to extend the matcher class, and add your logic.
(ii) **Multiple Importers**: OpenII includes several importers for many schema models. The one of interest to us is the XSD importer, which allows us to import and compare multiple XML schemas at once. This feature comes handy when comparing cloud schemas, as concepts can span multiple files and have different structures based on the cloud provider implementation of the underlying platform.
(iii) **Multi-Format Exporter**: OpenII includes a number of exporters (including spreadsheets, and cvs). Mappings can thus be easily exported to other tools for further analysis and processing.
(iv) **High-end GUI**: Harmony allows users to interactively refine the automatically-suggested mappings, confirm and reject the matches and add necessary annotations, including specifying transformation functions. This feature is extremely important for semi-automatic schema matching approaches.

Our implementation uses the OpenII loaders, mappers and code-generator. It also extends the matcher module by adding an extra matcher: *The web-semantic metric matcher*.

The web-semantic metric extends the matcher class. It has been implemented to work as a composite matcher that can be selected with other matchers implemented within the OpenII framework (e.g., WordNet, Documentation, and Thesaurus Mathcers). This facilitates combining and comparing the results of different matchers. Fig. 3 shows a snapshot that shows the results of applying the web-semantic metric on three cloud schemas: Two belong to Microsoft Windows Azure (i.e., the *Service Definition* and *Service Configuration* schemas) and one belongs to the GAE (i.e., *appengine-web* schema). The figure shows how the web-semantic metric could be used as a composite matcher, and how the $\mathtt{WSM}_{\mathtt{SE}=\text{“}\mathtt{Bing}\text{”}}$ matcher is able to map the Azure "Role" to the GAE "Module" concepts



Annotations
Date: 23-May-2014
Author: Web Semantic Similarity
Notes:
Confidence
No Evidence — Much Evidence
Evidence (0.67)
Accept Reject
Mapping Function
Built-in Function: <None>
Parameter $1:
Output =

Role
Description: The configuration and instance settings for a role.

Run Schema Matchers
Please select which matchers to use:
- [ ] Name Similarity Matcher
- [ ] Documentation Matcher
- [ ] Mapping Matcher
- [ ] Thesaurus Matcher
- [ ] Exact Matcher
- [✓] Web Semantic Similarity
- [ ] Quick Matcher
- [ ] WordNet Matcher

[ ] Select advanced matcher options

[✓] User
[✓] System
[✓] Hierarchy
[ ] Best

Fig. 3: Applying the WSM on GAE and AZURE Schemas

described earlier, with high confidence (evidence = 0.67). This experiment will be discussed further in the evaluation section (Section V).

The following are some considerations that have been taken into account in our implementation of the WSM matcher:

(i) **Response time of online requests**: The main bottleneck for the WSM approach is the time needed for performing the search requests. The complexity of the current approach based on Equation (2) is given by:

$$\texttt{TotalSearchRequests}_{\texttt{withoutcache}} = 3 \times n^2$$

assuming that the two schemas S_1 and S_2 have the same number of elements (n). We addressed this in our implementation, through implementing a caching strategy. Implanting a caching strategy can reduce the number of times we call the search engine API, by the order of three as shown in the folowing equation:

$$\texttt{TotalSearchRequests}_{\texttt{withcache}} = n \times (n + 2)$$

Unfortunately, the number of calls will still grow exponentially with the number of elements in the schemas.

(ii) **Number of API requests allowed by the search engine provider**: Search engine providers (e.g., Google, Bing) limit the number of calls per second and the maximum number of calls per month or day. In the current implementation, the user selects a license key. If the key expired while the matcher is still running it prompts the user to update the key.

The next section discusses the tests performed to evaluate the proposed system.

V. Evaluation

Revisiting our research questions, we now need to assess the generality of this approach, and evaluate its performance in comparison with other methods of schema mapping.

A. *Evaluating Prison Break Generality*

In the context of this research, generality refers to the ability of the process to perform its intended task with relatively few constraints. We want to apply traditional schema matching to the cloud domain, by allowing new matchers to discover complex matches between the source and target schemas based on concept domain similarity. The more generic the approach, the fewer constraints it imposes on its input in order to be able to find schema matches. For example, if one approach requires the user to specify more information in order to perform its computations, this makes it less generic.

By checking the input constraints of the Prison Break approach in comparison to other approaches that are able to incorporate the domain knowledge to discover schemas correspondences, we notice that pure linguistic approaches require extra sources of knowledge to distinguish domain concepts, such as thesauri, acronym lists, dictionaries, and mismatch lists. Prison Break does not require such information. Instead it uses search engine results, which can be obtained automatically at runtime to incorporate domain knowledge in the node similarity matching process. Hence, Prison Break is a more generic approach than other matching processes that take domain knowledge as input.

B. *Comparing Prison Break Performance With Other Approaches In OpenII*

Evaluating performance involves applying Prison Break to sample vendor cloud domain schemas, then evaluating the results in order to determine how it performs in comparison to other approaches. This section explains on how we set up the experiment to conduct this comparison, and what we discovered using the OpenII framework in terms of the (i) quality of results, and (ii) execution time across the methods we compared.

1) ***Experiment Setup****:* To evaluate performance, we conducted an experiment to compare Prison Break with schema matching techniques implemented within the OpenII framework. OpenII implements six different matchers: *Name Similarity*, *Documentation*, *Exact*, *Mapping*, *Thesaurus*, and *WordNet* matchers [18]. Unfortunately, due to the complexity of cloud schemas, and the lack of information regarding both domain glossaries and the documentation of schema elements, only two matchers could be considered for the comparisons: Name Similarity, and WordNet. The Name Similarity matcher is a linguistic syntactic matcher that is based on edit distance [19]. calculated through the number of operations (i.e., Insertion, Deletion, Substitution) that are required to transform one string into another. The WordNet matcher, by contrast, is a bag matcher that uses the WordNet dictionary as a thesaurus [18].

In this experiment, the Prison Break WSM was set to use Microsoft Bing as a search engine, as it provides 5000 free requests a month. The search was bound to a context by appending the keywords “Azure” before all the first schema concepts, and “GAE” before each of the second schema concepts. The time between requests was set to 0.25 seconds to avoid being classified as a denial of service (DoS) attack.

The experiment was conducted using real cloud schemas for two main cloud platform providers: Microsoft and Google.

In particular, we compared the Azure applications definition and configuration schemas with the GAE "appengine-web" schema configuration file. No additional information was provided (e.g., domain dictionaries or documentation).

We ran the experiment using the different matchers that have been selected, then filtered the results based on different threshold evidence. The mappings were then exported to be evaluated. To make the comparisons fair, no manual selection of mappings have been considered in this evaluation.

*2) **Evaluation of Results**:* To evaluate the quality of the matching outcomes a set of reference alignments were used. These alignments have previously been created manually for the same schemas, as part of our efforts to devise platform independent domain models for cloud applications [20]. For each alignment, the values of precision, recall, and F-measure were computed according to Equations 3, 4, and 5 respectively. The outcome of these equations is between zero and one, the higher the value the better the result.

$$\text{Precision} = \frac{|\{\text{ReferenceAlignments}\} \cap \{\text{MatchResults}\}|}{|\{\text{MatchResults}\}|} \quad (3)$$

$$\text{Recall} = \frac{|\{\text{ReferenceAlignments}\} \cap \{\text{MatchResults}\}|}{|\{\text{ReferenceAlignments}\}|} \quad (4)$$

$$\text{F} - \text{measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (5)$$

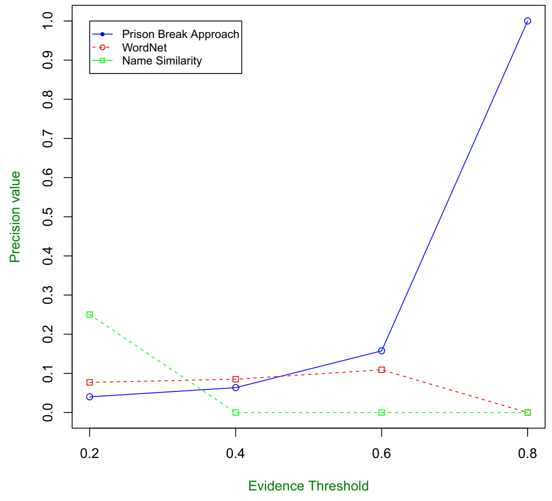


Fig. 4: Precision

The goal of calculating precision is to assess how many of the matches discovered by the matcher are correct in comparison to the total number of matches retrieved. The higher the precision value, the smaller the set of wrong matches. Fig. 4 shows that as the evidence threshold increases, the Prison Break approach tends to provide higher precision than other approaches. The precision of both Prison Break and WordNet increases until the evidence reach 60% at that point WordNet hits zero while Prison Break jumps to 100% precision before hitting zero after that. The reason for this sudden change is that at 80% confidence Prison Break only retrieves one value, which is also a correct value. On the other hand the WordNet matcher retrieves zero matches at confidence higher than 60% for the set of schemas used. The worst precision results were obtained by the Name Similarity matcher. While the precision of the Name Similarity matcher starts higher than both other matchers. It falls down rapidly to hit zero when the evidence required is greater than 40%. The reason is that most matches returned by the Name Similarity matcher have low evidence. If most of the values returned by a matcher have low evidence, this means that the matcher is not suitable for discovering matches for that type of schema.

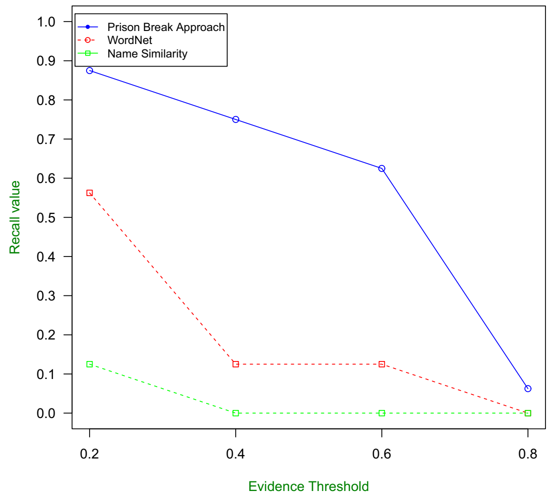


Fig. 5: Recall

The goal of calculating recall is to assess how many correct matches are retrieved in comparison to the total number of correct matches in the reference alignment. The higher the recall value, the smaller the set of the matches that have not been found. Fig. 5 shows that the number of matches found decreases as the evidence threshold increases for all the three matchers. This is expected, because when the evidence is low the matcher retrieves more results; both correct and incorrect. Fig. 5 also shows that Prison Break always obtains higher recall results, followed by WordNet and lastly the Name Similarity matcher. The values of recall of Prison Break is on average 38% higher than those of WordNet and 55% higher than Name similarity. The reason Prison Break has the highest recall is due to its dependency on WSM. WSM is a statistical measure. Given the large number of pages on the Internet, in most cases, the value WSM is greater than zero, which indicates some statistical relatedness. However, this is not the case for linguistic based similarities, where the similarity can easily hit zero.

To have a better perception of the quality of the different matchers, the values of both precision and recall should be

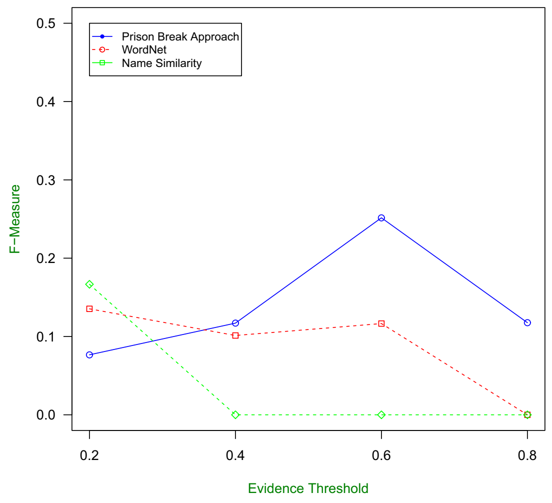


Fig. 6: F-measure

combined. One way to combine both of them is through the F-measure, which is the harmonic mean of precision and recall. The F-measure provides a global measure of the matching quality. A high value of F-measure indicates that the matching results are of a good quality.

As shown in Fig. 6, for evidence thresholds greater than 30% the value of the F-measure for Prison Break is always higher than both the WordNet and the Name Similarity matchers. On average the value of the F-measure is 5.2% higher than the WordNet and 10% higher than the Name Similarity matcher. For Prison Break, the maximum F-measure value occurs at evidence threshold of 0.6, while the F-measure values for both other approachs reach their maximum at an evidence threshold of 0.2 . The low F-measure values obtained in this experiment are an indication that pure linguistic approaches are not suitable for discovering matches between the cloud domain concepts of the different vendors. The high F-measure value provides evidence of the advantage of using web search similarity to incorporate domain knowledge in the element matching process to discover domain concepts similarities.

To summarize, Prison Break has higher precision recall and F-measure values than the pure linguistic approaches, when applied to the cloud schemas. Unfortunately, one of the major drawbacks of the Prison Break approach is the execution time. While the execution time for the pure linguistic approaches are measured in seconds, the Prison Break approach execution time is measured in minutes. For instance, for the example presented in this section, it took around five minutes for Prison Break matching to run.

C. Threat to Validity

This section discusses three validity threats to our approach. Two external validity threats (i.e., accuracy of search results, and experiment size) that affect the generalizability of the approach and one internal threat (i.e., reference alignments) that questions the current results.

- **Accuracy of search results:** The web similarity metric depends on the page counts returned in search results. Consequently, the accuracy of the web semantic similarity measures is based on the correctness of these results. In our work, we noticed that some search APIs return misleading information based on estimates (e.g., Google, Yahoo). This is becuase these search engines uses cached results for search queries and do not perform deep search (a more sophisticated search that provides more accurate results) unless the cached results are below a certain threshold, or deep search is enforced through parameter tuning. For example searching the term ”car” on Google will return 965 million results, while ”car games” will return over than a billion pages. In order to address this issue, in our implementation we enforce deep search, allowing users to use different search engines and combine search results.
- **Reference alignments:** The evaluation presented in this paper depends on the correctness of the manual alignments created by the domain expert. Such alignments could be questionable, as they may involve human error. The credibility of the manually created alignments used in this study emerges from the fact that these alignments have already been used to create a domain independent modeling language [20], and were realized as mapping rules to automate the generation of cloud provider-specific deployment artifacts from existing provider-independent models.
- **Experiment size:** Another issue is the number of sample schemas used. In order to generalize the results of this study, Prison Break needs to be applied to other schemas for other cloud vendors. We are planning to address this in our future work.

Despite these external and internal threats, the approach is promising and presents a progressive step toward automatic model migration for cloud applications.

VI. Related Work

The risk of vendor lock-in motivates research in cloud portability [1]. Several techniques and solutions have been devised to enable portability and facilitate migration in the cloud [2], [21]–[26]. The goal is to promote reusability and ensure that applications will work in the same manner, despite the underlying platform or provider specifications.

The most common way to address portability is through abstraction approaches (e.g., meta-modeling, feature modeling, ontologies) [27], [28]. Abstraction approaches highlight the commonalities and differences between the different provider models. Several standardization bodies [29] have created cloud domain reference models and ontologies in an attempt to tackle the portability problem. The literature [1] distinguishes three types of portability at different levels of abstraction in the cloud: (i) Portability of virtual machines at the infrastructure level. The Open Virtualization Format [30] is an example of a standard that supports this type of portability [30]. (ii) Portability of service models at the platform level. StratusML [31], mOSAIC [32], MODACloud [33], and CloudML [34] are some examples of languages and tools that enable this type of portability. Finally, (iii) portability of data. The goal here is to maintain data integrity while importing and exporting data between different providers. While this type of portability has been widely studied in the literature, cloud computing creates new challenges, such as migration for shared nothing databases [35] .

As argued in Section III, it is almost impossible to impose a single platonic standard. In fact, several contributions in the literature have also concluded that having a single standard is not advisable in practice [36]–[38]. This makes schema matching a mandatory requirement for portability. Matching schemas enables the knowledge and data of the source schema to be expressed with respect to the matched target schema, which facilitate portability [6]. Several solutions for schema matching have been devised in the last decades [12], [39]. Many surveys [4], [40], and books [41] have thoroughly covered the topic. Morover, several schema-matching tools have been developed by the database community in the past two decades; Cupid , COMA++ [42], AgreementMaker [43] and OpenII Harmony [18], are possibly the most noticeable.

Table III shows a comparative overview of these tools. These tools integrate useful matchers based on linguistic, and structural based matching techniques, they provide comprehensive GUIs, module importers, and facilitate domain knowledge matching based on external dictionaries. Unfortunately, none of these tools currently support semantic matching based on web search results. For this reason the work presented in this paper not only paves the way for solving the cloud vendor lock-in problem, it also contributes to the schema matching community by making search-based semantic similarity matching available as part of one of the most comprehensive schema matching tools (OpenII).

TABLE III: Comparison of selected match tools

Comparison Criteria		Cupid	COMA++	Agreement Maker	OpenII Harmony
First time introduced		2001	2002/2005	2007	2008
Comprehensive GUI		×	√	√	√
Matchers	Linguistic	√	√	√	√
	Structural	√	√	√	√
Use of External Dictionary		√	√	√	√
Extensibility Support		×	×	×	√

The role that schema matching can play to address the vendor lock-in problem is unquestionable. To date, the only research initiative we are aware of that plans to adopt schema-matching is mOSAIC [44]. As part of mOSAICs work toward this goal, Cretella and Di Martino [8] described a schema matching prototype system that aims to support mapping of providers functionalities and resources between the different providers APIs. Different from our approach, Cretella and Di Martino approach uses a traditional linguistic schema matching approach based on syntactic analysis and WordNet thesaurus. Our study uncovers the problems in such approaches and provides a solution through incorporating domain knowledge using web-search based semantic matching (Web Similarity Measure).

The relationship between cloud computing and schema matching is mutual. Schema matching contributes to solving the vendor lock-in problem, while cloud computing can be used to develop efficient schema matching systems to address some limitations of existing mapping processes (i.e., scalability). For example in [45] the authors use cloud computing to support the collaborative reconciliation of schemas.

Using web metrics for semantic disambiguation is not new. Cilibrasi and Vitanyi paper "The Google similarity distance" [14] has been cited over a thousand times since 2007. Our approach distinguishes itself by virtue of our application domain and implementation. The work we present provides a case study from the cloud domain, and evaluates the applicability of web-metrics in addressing the vendor lock-in problem.

VII. Conclusions and Future Directions

To help mitigate the platform lock-in problem imposed on application owners by cloud vendors, this paper proposed a novel schema matching approach called Prison Break. Prison Break uses web search results to compute the similarity between any two schema-elements. This allows it to discover alignments between schema elements that do not share linguistic similarities. Prison Break is a generic approach, as it does not require a dictionary of domain concepts to incorporate domain knowledge.

Prison Break was compared with other matchers implemented within the OpenII framework. The results showed that Prison Break has higher recall and greater precision for results with high evidence. The F-measure shows that the results of the Prison Break approach are always better when matching evidence is higher than 30%.

As in all schema-matching approaches scalability is still a major issue that affects matching performance in terms of execution time. However, due to its dependency on online search results, Prison Break takes even more time than other processes to discover the alignments.

The current implementation of Prison Break addresses the scalability problem by caching. The performance of the current approach can be further improved in the future by implementing the similarity-matching task as a parallel process and using partitioning to allow several machines to perform similarity matching at the same time.

Schema matching plays an important role in the process of automating model migration and evolution. It is a key for solving the vendor-lock in problem. However, vendor driven schemas are unconventional schemas. These schemas pose challenges for schema matching techniques. Particularly, there is a need for approaches that incorporate more domain knowledge in the matching process. Moreover, there is ultimately a need for a repository for thesauri, acronyms, dictionaries, and mismatch lists for schemas across cloud vendors.

References

[1] D. Petcu, "Portability and interoperability between clouds: Challenges and case study," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, vol. 6994. Springer, 2011, pp. 62–74.

[2] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, "Portable cloud applications-from theory to practice," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1417–1430, 2013.

[3] K. Garcés, F. Jouault, P. Cointe, and J. Bézivin, "Managing model adaptation by precise detection of metamodel changes," in *Model Driven Architecture-Foundations and Applications*, ser. Lecture Notes in Computer Science, vol. 5562. Springer, 2009, pp. 34–49.

[4] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *the International Journal on Very Large Data Bases*, vol. 10, no. 4, pp. 334–350, 2001.

[5] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *the International Journal on Very Large Data Bases*, vol. 4, no. 11, pp. 695–701, 2011.

[6] P. Shvaiko and J. Euzenat, "Ontology matching: State of the art and future challenges," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 158–176, 2013.

[7] J. Euzenat and P. Shvaiko, *Ontology Matching*, 2nd, Ed. Springer, 2007.

[8] G. Cretella and B. Di Martino, "Semantic and matchmaking technologies for discovering, mapping and aligning cloud providers's services," in *the International Conference on Information Integration and Web-based Applications & Services*, 2013, pp. 380–384. [Online]. Available: http://doi.acm.org/10.1145/2539150.2539204

[9] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *the International Conference on Very Large Data Bases*, vol. 1, 2001, pp. 49–58.

[10] M. Corporation. (2014) Windows azure. [Online]. Available: http://azure.microsoft.com/

[11] G. Inc. (2014) Google app engine. [Online]. Available: https://appengine.google.com

[12] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in *the International Conference on Data Engineering*, 2002, pp. 1–12.

[13] J. Jiang and D. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," in *the International Conference on Research in Computational Linguistics*, 1997, pp. 1–15.

[14] R. L. Cilibrasi and P. M. Vitanyi, "The google similarity distance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 370–383, 2007.

[15] J. Gracia and E. Mena, "Web-based measure of semantic relatedness," in *the International Conference on Web Information Systems Engineering*, 2008, pp. 136–150.

[16] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi, "The similarity metric," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[17] M. Li and P. M. Vitányi, *An introduction to Kolmogorov complexity and its applications*, 3rd, Ed. Springer, 2009.

[18] P. Mork, L. Seligman, A. Rosenthal, J. Korb, and C. Wolf, "The harmony integration workbench," in *Data Semantics XI*. Springer, 2008, pp. 65–93.

[19] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.

[20] M. Hamdaqa, T. Livogiannis, and L. Tahvildari, "A Reference Model for Developing Cloud Applications," in the International Conference on Cloud Computing and Services Science, 2011, pp. 98–103.

[21] E. Brandtzæg, S. Mosser, and P. Mohagheghi, "Towards CloudML, a model-based approach to provision resources in the clouds," in *the European Conference on Modelling Foundations and Applications*, 2012, pp. 18–27.

[22] D. K. Nguyen, F. Lelli, Y. Taher, M. Parkin, M. P. Papazoglou, and W.-J. van den Heuvel, "Blueprint template support for engineering cloud-based services," in *Towards a Service-Based Internet*. Springer, 2011, pp. 26–37.

[23] M. Kostoska, M. Gusev, and S. Ristov, "A new cloud services portability platform," *Procedia Engineering*, vol. 69, pp. 1268–1275, 2014.

[24] F. Gonidis, A. J. Simons, I. Paraskakis, and D. Kourtesis, "Cloud application portability: an initial view," in *th Balkan Conference in Informatics*, 2013, pp. 275–282.

[25] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4–1, 2009.

[26] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using tosca." *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, 2012.

[27] A. Ranabahu, E. Maximilien, A. Sheth, and K. Thirunarayan, "Application portability in cloud computing: An abstraction driven perspective," *IEEE Transactions on Services Computing*, vol. 99, no. 1, pp. 1–14, 2013.

[28] M. Smit, "Supporting software evolution to the multi-cloud with a cross-cloud platform," in *the International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, 2013, pp. 85–85.

[29] G. A. Lewis, "Role of standards in cloud-computing interoperability," in *International Conference on System Sciences*, 2013, pp. 1652–1661.

[30] *Open Virtualization Format Specification*, Distributed Management Task Force, Inc. Std. DSP0243, Rev. 2.1.0.

[31] M. Hamdaqa and L. Tahvildari. (2013, November) Stratusml: A layered cloud applications modeling language. [Online]. Available: https://www.youtube.com/watch?v=24YZd2MqhZY

[32] D. Petcu, B. Di Martino, S. Venticinque *et al.*, "Experiences in building a mosaic of clouds," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, pp. 1–22, 2013.

[33] D. Ardagna, E. Di Nitto, P. Mohagheghi *et al.*, "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds," in *The ICSE Workshop on Modeling in Software Engineering*, 2012, pp. 50–56.

[34] G. Gonçalves, P. Endo, M. Santos, D. Sadok *et al.*, "Cloudml: An integrated language for resource, service and request description for d-clouds," in *the International Conference on Cloud Computing Technology and Science*, 2011, pp. 399–406.

[35] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi, "Zephyr: live migration in shared nothing databases for elastic cloud platforms," in *the International Conference on Management of data*, 2011, pp. 301–312.

[36] B. Haslhofer and W. Klas, "A survey of techniques for achieving metadata interoperability," *ACM Computing Surveys*, vol. 42, no. 2, pp. 1–42, 2010.

[37] D. E. O'Leary, "Impediments in the use of explicit ontologies for kbs development," *International Journal of Human-Computer Studies*, vol. 46, no. 2, pp. 327–337, 1997.

[38] R. Santodomingo, S. Rohjans, M. Uslar, J. Rodríguez-Mondéjar, and M. Sanz-Bobi, "Ontology matching system for future energy smart grids," *Engineering Applications of Artificial Intelligence*, vol. 32, pp. 242–257, 2014.

[39] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM computing surveys*, vol. 18, no. 4, pp. 323–364, 1986.

[40] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *Data Semantics IV*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3730, pp. 146–171.

[41] Z. Bellahsene, A. Bonifati, and E. Rahm, *Schema matching and mapping*. Springer, 2011.

[42] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with coma++," in *the International Conference on Management of Data*, 2005, pp. 906–908.

[43] I. F. Cruz, F. P. Antonelli, and C. Stroe, "Agreementmaker: Efficient matching for large real-world schemas and ontologies," *the VLDB Endowment*, vol. 2, no. 2, pp. 1586–1589, 2009.

[44] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, and V. Munteanu, "An analysis of mosaic ontology for cloud resources annotation," in *the Federated Conference on Computer Science and Information Systems*, 2011, pp. 973–980.

[45] H. Q. V. Nguyen, X. H. Luong, Z. Miklós, T. T. Quan, and K. Aberer, "Collaborative schema matching reconciliation," in *On the Move to Meaningful Internet Systems*. Springer, 2013, pp. 222–240.