Computing and Informatics, Vol. 37, 2018, 1231–1257, doi: 10.4149/cai_2018_5_1231

ONTOLOGY-BASED RESOLUTION OF CLOUD DATA LOCK-IN PROBLEM

Darko Andročec, Neven Vrček

Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2
42000 Varaždin, Croatia
e-mail: {dandrocec, nvrcek}@foi.hr

Abstract. Cloud computing is nowadays becoming a popular paradigm for the provision of computing infrastructure that enables organizations to achieve financial savings. On the other hand, there are some known obstacles, among which vendor lock-in stands out. Furthermore, due to missing standards and heterogeneities of cloud storage systems, the migration of data to alternative cloud providers is expensive and time-consuming. We propose an approach based on Semantic Web services and AI planning to tackle cloud vendor data lock-in problem. To complete the mentioned task, data structures and data type mapping rules between different types of cloud storage systems are defined. The migration of data among different providers of platform as a service is presented in order to prove the practical applicability of the proposed approach. Additionally, this concept was also applied to software as a service model of cloud computing to perform one-shot data migration from Zoho CRM to Salesforce CRM.

Keywords: Cloud data portability, data migration, platform as a service, software as a service, data type mappings, semantic web services

Mathematics Subject Classification 2010: 68-P20

1 INTRODUCTION

Many end users and corporations store parts of their data in clouds. For now, it is not easy to move these data from one cloud vendor to another. Due to different cloud

storage models and implementations, it is difficult for users to switch to another cloud storage solution in case of service dissatisfaction or change of users' needs. The cloud data lock-in problem is characterized by time-consuming and costly migration of data to alternative cloud solutions offered by different vendors. Currently, each cloud vendor offers its own tools, remote application programming interfaces (APIs), and cloud storage data models. The numerous heterogeneities among different vendors make cloud data portability an interesting and complex research and practical problem. The migration of data between different offers of platform as a service is the main focus of this work. The same approach was applied to software as a service model, where we have chosen to migrate data between two cloud customer relationship management (CRM) systems: Zoho CRM and Salesforce.

The presence of several coexisting cloud storage models introduces the need for translation techniques and tools. The main research questions of this paper can be stated as: How to leverage automatic data migration between cloud solutions? How to provide cross cloud data type mappings for different cloud storage systems used by various cloud offers? There are many data migration/interoperability problems among cloud providers, so the answer to the mentioned research questions is not trivial. For example, there exists a difference between data storage models of various commercial cloud providers; storage data types differ in name, value space, permitted range of values, precision of data; data import or export is often complex; some offers have a predefined standard data container or have some naming restrictions. The main contribution of this work is a flexible data migration among cloud storage systems that uses Semantic Web and AI planning to avoid point-to-point mappings. We have used a standard data model in the form of the OWL ontology representing data schema, data types, and data elements. The result of this work is the architecture design for automated data migration among different cloud providers.

This paper proceeds as follows. First, in Section 2, the related work is listed. Section 3 lists the main steps of our research. In Section 4 we present the chosen PaaS offers and storage types. Section 5 shows how we use OWL as an intermediate format for the migration of data structure and data from one PaaS storage to another. Section 6 briefly describes the developed PaaS ontology and our implementation of data type mappings between different PaaS storage offers. The following three sections show the approach we proposed, and explain the use of Semantic Web services, AI planning, and Apache CXF framework to migrate data between PaaS providers. In Section 10, the validation of our data migration approach was done by migrating a more complex set of PaaS and SaaS data. Our conclusions are provided in the final section.

2 RELATED WORK

Cloud storage interoperability problems are similar to interoperability conflicts among multiple independent database systems. These problems are well investi-

gated in the current literature. For example, Sheth and Kashyap [1] classified and defined the most important interoperability conflicts among multiple independent database systems. They listed the following main categories of incompatibilities [1]: domain definition incompatibility (attributes have different domain definitions), entity definition incompatibility (descriptors used for the same entity are partially compatible), data value incompatibility (inconsistency between related data), abstraction level incompatibility (the same entity is represented at different levels of abstraction), schematic discrepancy (data in one database correspond to schema elements in another). For each incompatibility category, Sheth and Kashyap listed possible concrete conflicts. Parent and Spaccapietra [2] listed the most relevant issues and the approaches to tackle data interoperability problems when integrating databases. They distinguished seven categories [2]: heterogeneity conflicts, generalization/specialization conflicts (different generalization/specialization hierarchies and different classification abstractions), description conflicts (types have different properties and/or their properties are described differently [2]), structural conflicts (different structures of related types), fragmentation conflicts (the same object is depicted by decomposition into different elements [2]), metadata conflicts, and data conflicts (data instances have different values for the same properties).

Park and Ram [3] conclude that semantic conflicts among databases can occur at two levels: data and schema. Data-level conflicts include data-value conflicts (the data value has different meaning in different databases), data representation conflicts (such as different representations of date and time), data-unit conflicts (different units are used in different databases), and data precision conflicts. All data-level conflicts can occur at the attribute level or at the entity level. Schema-level conflicts include [3]: naming conflicts, entity-identifier problems, schema-isomorphisms, conflicts of generalization, aggregation conflicts, and schematic discrepancies. Arenas et al. [4] tackled the problem of exchanging data between different relational databases that have different schemas. They mentioned three key problems of relational and XML data exchange: how to build target solutions; how to answer queries over target solutions; and how to manipulate with schema mappings (metadata management)[4]. Rocha et al. [5] presented their framework to support migrating from relational (MySQL) to NoSQL databases (MongoDB). Their framework consists of migration and mapping module. However, their work is not focused on cloud storage systems and how to access these storage systems. It also supports only MongoDB, a document-oriented NoSQL solution.

There are several cloud APIs and frameworks that act as intermediaries between different clouds. Apache Libcloud [6] is a Python library containing a unified API that can manage cloud resources of different providers. This library is focused on infrastructure as a service and supports cloud servers, block storage, cloud object storage, load balancers, and DNS as a service. Deltacloud API [7] contains a cloud abstraction API working as a wrapper around a large number of clouds to abstract their differences. It is also focused on IaaS providers and provides drivers for Amazon, Eucalyptus, GoGrid, OpenNebula, etc. Apache jclouds [8] is an open-source library offering blob (binary content) store and compute service abstraction

for 30 IaaS providers. There are also some commercial (industrial) approaches to tackle cloud portability and interoperability. For example, Cloutex can integrate and synchronize data between Salesforce, Quickbooks Online and Magento. A similar offer, Import2.com, enables the transfer of data between cloud applications such as Salesforce, Tumblr, Nimble, Pipedrive, SugarCRM, and Zoho CRM. Import2 is currently focused on CRM, helpdesk and blog migration of cloud data. These are all commercial offers with closed source code, and adding new cloud providers to their offers can be expensive or impossible.

There are some known approaches in the current literature to tackle cloud storage interoperability and migration problems. One of the first attempts to define cloud computing ontology to achieve cloud interoperability was introduced in Youseff et al. [9]. They presented an ontology which differentiates five main layers of cloud computing (applications, software environments, software infrastructure, software kernel and hardware). Ranabahu and Sheth [10] present the usage of semantic technologies to overcome cloud vendor lock-in issues. They distinguish four types of semantics for an application: data semantics (definitions of data structures, their relationships and restrictions), logic and process semantics (the business logic of the application), non-functional semantics (e.g. access control and logging) and system semantics (deployment descriptions and dependency management of the application).

Quinton et al. [11] proposed SALOON, a software product lines-based platform to select among cloud environments the best one for a specific purpose. Their platform automates the deployment of cloud environment configurations through the generation of executable configuration scripts. Tsai et al. [12] proposed a service-oriented computing architecture for cloud interoperation. However, the implementation or use case of the proposed architecture is still missing. Bastiao Silva et al. [13] developed a unified API for delivering services using cloud resources of multiple vendors with abstract layers for cloud blob stores, cloud columnar data (e.g. Azure Table), and Publish/Subscribe mechanism (Channel API of Google App Engine and Azure Queue). However, the focus of their work is to allow different applications to interoperate using a normalized API interface, and the authors did not tackle the issue of the cloud data migration.

Vision Cloud project [14] was primarily concerned with developing the architecture of a cloud-based infrastructure to provide a scalable and flexible framework for optimized delivery of data-intensive storage services. Its main aim is to solve the data management conflicts in cloud federations and multi-clouds. Five areas of innovation in the VISION Cloud platform [14] include: data objects are enriched with a detailed metadata, data lock-in should be avoided, computations are put close to the data, efficient retrieval of objects is enabled, and strong QoS guarantees, security and compliance with international regulations are guaranteed. Data objects are grouped into containers that can have associated metadata descriptions. Researchers working on Vision Cloud project used CDMI standard to achieve interoperability among CDMI-compliant cloud storage vendors. They also introduced the on-boarding federation to move data from one cloud storage provider to an-

other. Vision Cloud's approach uses a cloud storage container as the basic unit of the federation. Vision Cloud offers a RESTful API to manage data federation. However, commercial cloud vendors are rarely CDMI-compliant at the moment. Scavuzzo et al. [15] propose a migration system for columnar NoSQL databases (Google App Engine Datastore and Windows Azure Tables) that consists of intermediate metamodel and database-specific translators. Our approach is more comprehensive, because it includes different types of cloud storage systems (columnar NoSQL, relational and object databases), platform as a service and software as a service models, and it is tested on four cloud providers (Microsoft Azure, Google App Engine, Salesforce and Zoho CRM). Additionally, when migration is not successful, our system returns found interoperability problems and their descriptions. Data type mappings of different types of cloud storage systems are also addressed in our solution.

Shirazi et al. [16] provided a formal way for migrating data between HBase as a column family database to Neo4j graph database. However, their approach is specific to the two mentioned data storage types, it is not general nor can be extended to include other types of cloud data storage. Ali et al. [17] proposed a cloud broker solution for the data migration between different software as a service providers. Their approach has several steps, which include: collection and analysis of the metadata, development of a mapping model, solution design, implementation and testing of the solution. The main drawbacks of their approach include the need for the definition of point-to-point data mappings, and the need to implement a broker for each specific mapping, so our approach is better since we have many different cloud storage providers and it is more flexible. Bansel et al. [18] proposed a NoSQL data migration meta-model driven framework to foster data portability across cloud-based heterogeneous NoSQL databases. Our work is more comprehensive, because it includes relational and object databases, and is tested on more cloud providers' storage systems, and their approach is tested only on Microsoft Azure (Azure Tables to MongoDB or Neo4j) and includes only NoSQL databases.

Definite solutions to interoperability and portability issues of platform as a service and software as a service remain elusive due to the technical complexity and a lack of accepted standards [19]. The vendor lock-in is omnipresent in cloud offers, and many clients have postponed their investment because they fear the significant costs if they decide to migrate to another provider. The gaps in the existing literature include the lack of data portability among different types of cloud storage systems, especially between relational and NoSQL cloud data storage systems. Furthermore, there is no existing work that solves the problem of data type mappings among different types of cloud data storage systems (NoSQL, relational database, object databases). Our work deals with the mentioned problems. Our approach aims to ease the migration of data between cloud data storage systems. Once wrappers for a platform have been built, it requires little manual intervention to migrate data between various cloud data storage systems. The big advantages of our approach arise when one has to migrate many different data types and data among multiple cloud storage systems, because it is easy to incrementally add new mapping rules. The main contribution of our work is an ontology-based architecture for the

automatic cloud data migration between cloud storage systems (including relational and NoSQL storage systems). The approach is general for cloud storage systems, and we have checked it using two use cases, one for platform as a service, and one for software as a service model of cloud computing.

3 RESEARCH METHODOLOGY

The basic steps in this research include: design and implementation of use cases, development of the ontology used for semantic annotations, definition and development of semantic web services, and identification of interoperability problems among different commercial providers of platform as a service. In the first step of the research, use cases are defined. These use cases are examined to determine technical and semantic interoperability problems among APIs of different providers of cloud data storage systems and how to detect and resolve interoperability problems. The migration of data among different providers of platform as a service (Google, Microsoft, Salesforce) is the first use case used to prove the practical applicability of the proposed approach. Additionally, the concept was also applied to software as a service model of cloud computing to perform one-shot data migration from Zoho CRM to Salesforce CRM. In the future, similar use cases can be defined. Many papers and research projects (e.g. FP7 or Horizon2020) in the computer science field use a similar research method, i.e. begin with a definition of use cases to explain requirements and to later test the approach.

The second step of this research is the development of the ontology for resources and operations. The aim is to clearly describe and categorize the existing functionalities, features and specificities of commercial platform as a service offers. Additionally, the ontology supports data mappings among the heterogeneous APIs and various data storage systems. The offerings of platform as a service often use proprietary and non-standard databases (relational and non-relational). Representing these data models by means of ontology can provide a common layer for information exchange.

The PaaS ontology developed in the previous step is used to create semantic web services that represent remote functions (APIs) of platform as a service offers. Every operation from the cloud vendor's API will be semantically described using a web application developed for this purpose. The aim of these semantic web services is to simplify determination and resolution to interoperability problems among the existing commercial vendors. All cloud providers offer APIs to access and manage their cloud storage systems, and to integrate or migrate data from multiple providers. We can use these services that are further semantically annotated in our approach to enable automatic or semi-automatic data migration. In the end, we try to determine the existing interoperability problems among the selected commercial cloud solutions by comparing their associated semantic web services. For this purpose, the AI planning methods were used. AI planning was chosen because it is one of the most promising techniques to solve the problem of web service composition.

4 CHOSEN PAAS OFFERS AND STORAGE SYSTEMS

Our approach was first applied to platform as a service model of cloud computing. There are many providers of platform as a service. We have chosen the following three prominent providers of platform as a service: Microsoft, Google, and Salesforce. These offers were chosen because they are currently among the leading offers on the platform as a service market with many current users. For example, in the magic quadrant for enterprise application platform as a service published in January 2014, Gartner [20] listed Microsoft and Salesforce.com as the only two market leaders, and Google as the only market challenger among the total of 18 reviewed commercial PaaS providers. Furthermore, the mentioned PaaS offers support different types of data storage systems that can possibly identify more data interoperability problems in comparison to moving data only among the cloud storage systems of the same types.

On the Force.com platform, data objects are called custom objects (similar to tables in databases). In Salesforce [21], an organization represents a database with built-in user identity and security. Objects are similar to tables in relational databases and they contain fields and records. Objects are related to other objects by using relationship fields instead of primary and foreign keys. There are two types of objects: standard objects (predefined, created automatically by Salesforce) and custom objects (objects that you create in your organization). Each custom object has some predefined, standard fields. Every custom object's name on Salesforce must finish with the postfix __c (e.g. Customer__c).

Next, Google App Engine has three options for data storage: App Engine Datastore, Google Cloud SQL and Google Cloud Storage. The App Engine Datastore [22] is a schema-less object datastore. The datastore holds data objects named entities; each entity has one or more properties of one of the supported data types; and each entity is identified by its kind and key. Google Cloud SQL [23] enables the usage of relational MySQL databases in Google's cloud. The Google Cloud Storage is an experimental service that provides storage for big objects and files (up to terabytes in size). The first option (App Engine Datastore) was selected because it is the only free option. Furthermore, it is a good example of key-value cloud storage. Datastore objects can be created programmatically by means of Java object classes, servlets, HTML and JavaScript.

There are three main storage offerings on the Azure platform [24]: Local Storage, Windows Azure Storage, and SQL Database. Local Storage provides temporary storage for a running application and it represents a directory that can be used to store files. Windows Azure Storage consists of blobs (storage of unstructured binary data), tables (a schema-less collection of rows such as entities, each of which can contain up to 255 properties) and queues (storage for passing messages between applications) that are accessible by multiple applications. SQL Database is based on SQL Server technology and provides a relational database for the Azure platform. For the purpose of these use cases, SQL Database option was chosen. To be better at detecting interoperability problems among different types of PaaS storage, this

relational storage option was chosen, because in the first two providers different types of PaaS storage were selected. More various interoperability problems can be detected if different types of PaaS storage were chosen, instead of choosing the same or similar storage types (such as key-value datastore, relational database-like storage, or object storage) for each PaaS provider. A database can be created by means of Microsoft Azure management portal (`https://windows.azure.com`). It can also be created programmatically.

Most API data operations deal with one data container (table, entity, or custom object), so if users want to migrate all data, they must first learn how to get names or identifiers of data containers. All three chosen PaaS providers enable CSV export, and these files can be used to obtain the required names or identifiers. The obtained basic structure can be used to call remote API functions to retrieve detailed information about the structure of data and data types from cloud storage systems.

5 TRANSFORMATION OF DATA STRUCTURES AND DATA TO ONTOLOGY

Data structures and data of each platform as a service's storage will be represented as the unified data model ontology, so OWL will be used as an intermediate format to migrate data between PaaS vendors. We have chosen OWL, because we also use it for semantic web services. The mappings could be implemented in any other format. This architecture is inspired by a direct mapping approach [25] proposed by the RDB2RDF Working Group. The transformation of data structures from cloud storage to ontologies is based on mapping rules that specify how to map PaaS data constructs to the ontological models. Astrova et al. [26] proposed an approach to automatic transformation of relational databases to ontologies. They listed the mapping rules [26] which inspired the rules presented later in this paper. Inevitably, some of the semantics captured in a relational database will be lost when transforming the relational database to the ontology [26], the same situation will certainly also happen when dealing with PaaS storage systems.

Due to many differences among cloud storage types supported by major commercial providers of platform as a service, the basic transformation rules were defined to build data model ontology's classes, data properties and instances (see Table 1). The mappings in the other direction (from OWL ontology to cloud storage) could also be defined, so representing these data models by means of the OWL ontology can provide a common layer for information exchange. The web services for reading and writing OWL data ontologies were created using the above specified transformation rules and the Apache Jena framework [27] for building Semantic Web applications in Java programming language. Jena provides an API to work with OWL and RDFS files and a rule-based reasoning inference engine.

Azure SQL	**GAE Datastore**	**Salesforce**	**OWL**
table	entity kind	object	**OWL class**
column	property	field	**data type property**
row	entity	record	**instance**
primary key	identifier from an entity key	identifier of an object (recognized as a field of Salesforce's ID data type)	**data property identifier in an instance**
foreign key	relationship between two entities	relationship between two objects (recognized as a field of Salesforce's reference data type)	**object property hasLinkToObject with the appropriate domain and range in an instance**

Table 1. Basic transformation rules to and from an intermediate OWL file

6 PAAS ONTOLOGY AND DATA TYPE MAPPINGS

Our architecture for PaaS data migration is using OWL as data intermediate format. Additionally, we have used PaaS ontology to define mappings between data types of different PaaS providers and to list all relevant providers' API operations for manipulating and management of underlying PaaS data. For the purpose of the development of PaaS ontology, the Ontology Development 101 [28] methodology was selected. This methodology was chosen among others, because it is the simplest and it is really focused on the results, i.e. building the first ontology version very fast and then refining it according to requirements. Web Ontology Language (OWL) was chosen because it has the needed expressive power and is the most widely used language for ontologies in the papers in the field of computer science and research projects related to this field of study. The aim of the ontology is to describe clearly and to categorize the existing functionalities and features of commercial providers of platform as a service. Our ontology is publicly available at `https://github.com/dandrocec/PaaSInterop/tree/master/PaaSOntology5`.

Initially, the concepts in this ontology were derived from the existing cloud ontologies (mostly from mOSAIC project), PIM4Cloud [29] metamodel from REMICS project, OASIS Reference Ontology for Semantic Service Oriented Architecture [30], relevant related works from literature [9], remote cloud functions specified in the API documentation of the most prominent commercial providers of platform as a service (Google App Engine, Microsoft Azure, Salesforce), standards for Semantic Web services such as OWL-S and WSMO, relevant cloud computing standards (OCCI, TOSCA, CDMI), and using personal experience in building applications for platform as a service. A total of 146 classes were defined that are organized in 17 top level classes (Table 2). The ontology is described in more detail in our previous work [31].

For this paper, the most important part of the PaaS ontology is its application for data type mappings which will be explained in the next chapter.

Each platform as a service provider supports its own set of data types. Data types differ in their name, value space, permitted range of values, precision of data, etc. Data types from the three chosen PaaS storage types (Google App Engine Datastore [32], Microsoft Azure SQL Database [33], Salesforce [34]) are mapped to XSD (because OWL uses Schema Data Types – [35] and [36]), more specifically to an OWL data property's range of data model ontology. OWL (XSD) data types are chosen as a baseline system.

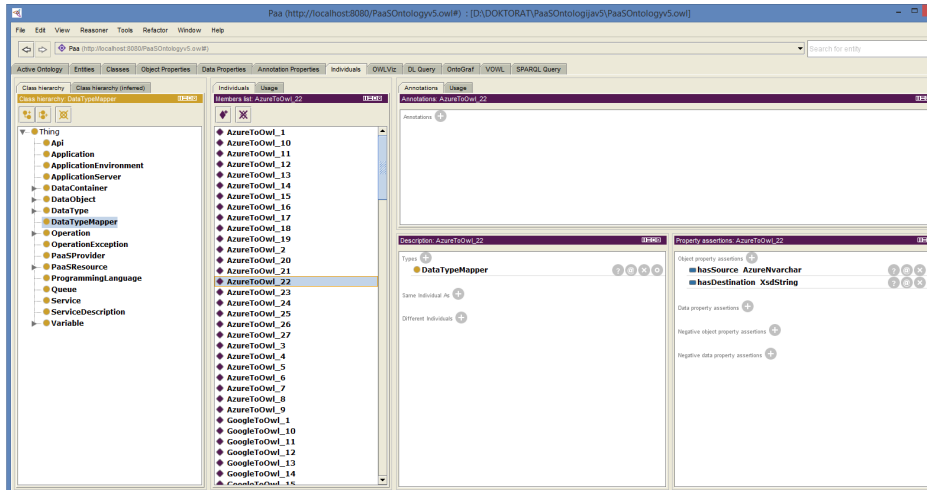

Figure 1. Instances of DataTypeMapper class for mapping different cloud providers' data types

Data type mappings were implemented by means of instances of the ontology. Two classes deal with data types mappings between different PaaS storage systems: *DataType* and *DataTypeMapper*. The subclasses of the *DataType* class are OWL data types and data types of each platform as a service storage model (*AzureDataType*, *GoogleDataType*, *OWLDataType*, and *SalesforceDataType*). Each data type is represented by an individual (instance) of the associated class. As an illustration, *XsdDate* is an instance of the OWL class *OWLDataType* and it represents the xsd:date type. The second important class is *DataTypeMapper*. This class has two object properties (*hasSource* and *hasDestination*), and instances of this class are actually data type mappings (see Figure 1). For instance, *SalesforceToOwl__2* is an instance of the *DataTypeMapper hasSource SalesforceBoolean* and *hasDestination XsdBoolean*, so it shows that the Salesforce's Boolean data type is mapped to the OWL Boolean data type.

Web services were created to handle these mappings automatically by reading the OWL ontology and performing the needed mappings and transformations.

Top Class	**Description**
Api	It represents vendors' Application Programming Interfaces (APIs).
Application	It contains all instances of applications that are deployed to a PaaS offer and run in the ApplicationEnvironment.
ApplicationEnvironment	PaaS application environment such as Google App Engine Java runtime environment.
ApplicationServer	Application server dedicated to efficient execution of cloud applications on vendor's servers.
DataContainer	This class is an abstraction of containers of data objects, e.g. tables, entities, objects, files directories.
DataObject	This class includes instances of data objects of various storage options such as NoSQL, relational database, object database and cloud file systems.
DataType	Data types in cloud storage systems or cloud services.
DataTypeMapper	Its instances are used for data type mappings between different storage systems of different PaaS vendors.
Operation	It represents all instances of remote operations defined in various vendors' APIs.
OperationException	It includes all instances of possible exceptions thrown by remote operations defined in vendors' APIs.
PaaSProvider	It includes instances of commercial vendors who offer platform as a service.
PaaSResource	A generic resource provided by a PaaS vendor.
Programming-Language	It contains instances of computer languages used for developing applications in vendor's environment.
Queue	It covers all instances of FIFO queues supported by commercial providers of platform as a service.
Service	It includes all kinds of services provided by commercial vendors of platform as a service.
ServiceDescription	A description of the functionality provided by service
Variable	Its subclasses include input, output, and results of APIs' web services.

Table 2. Top level classes of our ontology

For now, *DataTypeMapper* has approximately 150 instances (mappings). If some mappings are not correct, they can be fixed in the PaaS ontology and data type conversion will work. If another platform as a service provider is added, another subclass of *DataType* must be added, as well as instances for each data type of the new PaaS storage, and mapping instances from and to OWL data types must be created. Web services for data mapping to deal with the new storage provider also need to be slightly upgraded. This enables great flexibility regarding the mapping of data types supported by different PaaS providers. Some data types have unsupported mappings (for example, for Salesforce's *anyType* we can not find anything similar in OWL). In these cases, data migration will stop and error will be shown to the user suggesting that there is an interoperability problem connected to data types of different PaaS storage systems.

Ontology Development 101 methodology does not have an explicit evaluation step and it lacks evaluation procedure and recommendations, but evaluating the ontologies is useful to refine the ontologies and see whether they can be used in applications as expected. The ontology was evaluated by four human experts working in the field of cloud computing interoperability and related science projects. Their feedback was used to refine the ontology. After their initial feedback, the ontologies were revised and improved, and contact was kept (by email) with the experts who offered more comments on the newer versions of the ontologies. Several pitfalls were found by four experts and the ontology was improved.

7 SEMANTIC PAAS WEB SERVICES

All cloud providers enable developers to access their cloud storage systems using their application programming interfaces (APIs). They also provide custom tools or scripts, but these are specific for a certain cloud platform and not usable on another platform. If we want to integrate different cloud storage systems, the best solution is to use APIs (RESTful or SOAP service designed to be used exactly for this purpose, i.e. to integrate custom application or systems with cloud storage systems). Current web services provide only syntactical descriptions, so web service integration must be done manually. Semantic web services are the integration of Semantic Web and service-oriented architecture implemented in the form of web services. Semantic web services are aimed at an automated solution to the following problems: description, publishing, discovery, mediation, monitoring and composition of services. For this reasons, we have decided to use semantic web services. Web services that encapsulate remote API operations of three commercial providers (Google, Microsoft, and Salesforce) were developed to access these services in a unique way (providers offer their remote APIs in different forms - REST, SOAP or programming language libraries). These services directly call remote vendors' APIs. Some composite services (that call more than one cloud API operation and perform some additional tasks) were also developed (e.g., some of the services used for the data migration between PaaS storage types). Web services and all other parts of the authors' pro-

totype were implemented in Java. The source code of the tool is publicly available at `https://github.com/dandrocec/PaaSInterop`.

SAWSDL (W3C's Semantic Annotations for WSDL) [37] lightweight annotation was used to define semantic web services. SAWSDL was chosen due to its simplicity, its rich ontology-based data mediation mechanism for mapping inputs to outputs of web services and tool availability. The SOWER tool developed as part of the SOA4All FP7 project [38] was used to facilitate the manual annotation of WSDL service descriptions with the semantic information [38]. The web services that invoke API operations of the providers of platform as a service were developed, and each particular API operation with a term defined in this ontology of platform as a service can now be annotated. For instance, the Azure's *createTable* web service operation can be referenced to *CreateDataOperation* class of the OWL ontology.

8 AUTOMATED PAAS SERVICE COMPOSITION

As we have already mentioned in the previous section, what all cloud providers have in common is that they provide APIs for cloud storage management. To move data among different cloud providers, we have chosen to use semantically annotated web services representing cloud APIs. The next step is to automatically or semi-automatically compose semantic web services to retrieve data from one cloud storage, perform data type mappings, and store data into another cloud storage.

In the current literature, the automated composition of web services was performed using numerous methods, such as: Event Calculus, Petri Nets, Colored Petri Nets, Linear Logic theorem proving, AI planning, logic programming, Markov process, States Machines, etc. AI planning is one of the most promising techniques to solve the problem of the automated web service composition, and was chosen for the mentioned task in this work. We also wanted to determine the existing data interoperability problems among the selected commercial cloud solutions by comparing their associated Semantic Web services to find out which of these problems can be solved using the currently available API operations of commercial vendors. For this purpose, the AI planning methods were used and are described in more detail in Section 9.

A JSHOP2 planner was used for the AI planning process. The JSHOP2 planner was chosen because it is implemented in Java and can be easily incorporated into other parts of the prototype system that was developed using Java technologies, and it was used in the past for similar purposes, i.e. the composition of web services in various contexts. JSHOP2 is a Java version of Simple Hierarchical Ordered Planner (SHOP). It is used to generate sequential plans. It is based on ordered task decomposition where tasks are planned in the same order as later in the execution [39]. The objective of JSHOP2 and other HTN planners is to accomplish a set of tasks where each task can be decomposed, until primitive tasks [40]

are reached. The inputs of JSHOP2 are a planning domain and a planning problem. In JSHOP2, primitive tasks are called operators whose name must begin with an exclamation mark. The body of an operator consists of a precondition (must be satisfied to execute the action), a delete list (a set of properties that will be removed), and an add list (a set of properties that will be added) [39]. Solving a planning problem in JSHOP2 is done in three steps: the domain description file is compiled into Java code, the problem descriptions are converted into Java class, and the second Java class should be executed to initiate the planning process and inspect the planning results. These three steps were incorporated into the authors' prototype.

The problem description file is composed of logical atoms showing the initial state and a task list [39]. The task list and the initial state are created on the fly, when the user executes some interoperability actions using the client web application. Based on the choices of the user, the tasks that need to be completed are generated and saved in JSHOP2 problem description file. For example, if the user selects the data migration between Salesforce's and Google App Engine's PaaS storage, it looks like this:

```
((migrateData SalesForce GoogleAppEngine))
```

Java class was developed that handles this and writes the appropriate content to file using standard Java I/O and file classes and methods. In this case, the task lists are simply methods defined in the domain description file described later. The initial state (a set of logical atoms) is also created programmatically based on SAWSDL files and the PaaS ontology. A SAWSDL parser was developed in Java by using an EasyWSDL open-source library and its extension EasySAWSDL. The class for parsing the OWL ontology was implemented by using the Apache Jena library. Based on these two files, various logical atoms could be generated to represent the initial state. The most important logical atoms regarding PaaS data migration, together with their definition and the description of their creation are systematically listed in Table 3.

The domain description file is defined manually. A method called *migrateData* shows which operators should be called to migrate data from one PaaS storage to another:

```
(:method (migrateData ?from ?to)
         ()
         ((!checkDataTypeMappings ?from)
         (!createDataModelOntology ?from)
         (!createDataElementsFromOntology ?to))
)
```

First, the existence of the needed data type mappings are checked, then data model ontology is created, and finally data is migrated to target PaaS storage. The operator *!checkDataTypeMappings* includes preconditions that check whether all data types from data to be migrated have the appropriate data type mappings defined

Logical Atom (with Example)	**Description and Generating Method**
hasApiOperation (hasApiOperation Azure CreateDataOperation)	– it claims that a specific PaaS API has a specific API operation – cross-PaaS operation names are specified in the PaaS ontology, and services are annotated using SAWSDL – it is generated based on SAWSDL files – if Java class parsing SAWSDL finds a semantic annotation by means of sawsdl:modelReference on a service operation, it then generates hasApiOperation logical atom in JSHOP2 problem description file
typeInCurrentData (typeInCurrentData salesforcecurrency)	– it shows which type is present in storage system of the chosen PaaS offers – present data types in PaaS storage are obtained calling remote APIs of PaaS providers
dataTypeMappingExists (dataTypeMappingExists azuresmallmoney xsddecimal)	– it specifies the data type mapping between data types of different PaaS storage systems – the PaaS ontology is parsed to obtain all instances of DataTypeMapper OWL class that represent data type mappings between PaaS storage systems

Table 3. Logical atoms in the initial state

in the JSHOP2 problem file. The operators *!createDataModelOntology* and *!createDataElementsFromOntology* include preconditions to check whether the selected PaaS providers have the appropriate operations to execute the migration of data from source to target PaaS offer.

9 ARCHITECTURE FOR CLOUD DATA MIGRATION

After the domain and problem description files were successfully created, these definitions are forwarded to a component in the prototype which invokes JSHOP2 planner to get a plan if it exists. The domain and problem descriptions are dynamically compiled into Java code, and the resulting Java files are redeployed to Glassfish server. AI planning process can then be started. If JSHOP2 planner finds a plan, this plan is printed on the client web application, and an option to execute the plan (to invoke relevant web services) is given to the user. If the planner finds the appropriate plan, then no interoperability problems were found at this stage. The plan given by JSHOP2 is parsed to retrieve adequate web services from SAWSDL files that need to be executed. Apache CXF framework [27] was used to dynamically invoke the appropriate web services.

If there is no suitable plan returned by the JSHOP2 planner, the client web application displays the error message. In this case, some interoperability problems exist and the cause of the failure needs to be determined. In the existing literature,

there are few approaches to tackle gaps in the planning domains. Our approach is similar to the one proposed by Goebelbecker and Keller [41]. They proposed to change the initial state, when no plan can be found, with the aim to find reasons why some tasks cannot be solved. They named this change "an excuse"; they created a method for finding the candidates for the excuse where they replan with new initial states to find out whether they found the cause why the plan is not found. Our approach differs from the one proposed by Goebelbecker and Keller [41], because it does not need replanning that is an expensive and time-consuming task. This algorithm consists of four main steps:

Find problematic operator or method – The domain description file of JSHOP2 is simple. The data migration action is represented by a method that describes a set of operators that need to be executed. There is only one way to successfully get a plan – all operators defined in a particular JSHOP2 method must be successfully finished. JSHOP2 supports a function to programmatically inspect every step in the planning process. This function was used to get the list of all the steps of the planner. This list of steps was programmatically parsed in Java, and an operator or a method were found where the first BACKTRACKING action occurs. This action occurs when some preconditions of the operator or the method are not satisfied, and then the JSHOP2 planner goes back up in the tree to try to find another path to the solution. In this case, the first BACKTRACKING action in a plan step represents the problematic atom (problematic method or operator where interoperability problem had occurred).

Parse concrete preconditions – The next step is to parse preconditions of the problematic operator or method. JSHOP2 domain file is directly parsed to get all the relevant preconditions. A list of preconditions was created, and in the next step it was determined which of the preconditions is the cause of the problem.

Check whether the preconditions are satisfied in the end state – The end state (the last state after the AI planner fails to get a plan) is parsed to compare which of the preconditions are not satisfied in this state, and one or more preconditions are listed as indicators of interoperability problems.

List interoperability problems – Each logical atom that is used in states and preconditions has some meaning (for example, *hasApiOperation* describes that one PaaS offer has a particular API operation annotated with the cross-PaaS concept from the ontology). Using this meaning, error messages were programmatically created to explain the found interoperability problem in the client web application to a user. For example, if the problematic precondition contains *hasApiOperation*, then there is a missing API operation problem in the concerned PaaS offer.

Let us now examine the proposed architecture for data migration. The user starts data migration using the client web application (Figure 2). The CSV files

are parsed and data, data structures and types are retrieved by calling remote API functions. The data model ontology is created, and data is ready for migration to another PaaS provider. There are also internal web services that can read the data ontology, perform mappings, create data and data structures and move them into target PaaS storage. The mentioned internal web services parse SAWSDL files and the PaaS ontology, and use the techniques described in the previous sections. The AI planning component deals with the AI planning files and executes the required semantic web services. If the user chooses only one data container (table, entity, or Salesforce's custom object) the migration flow is the same, only data container name is forwarded as a filter to include only the chosen container and disregard the other remaining data during migration.

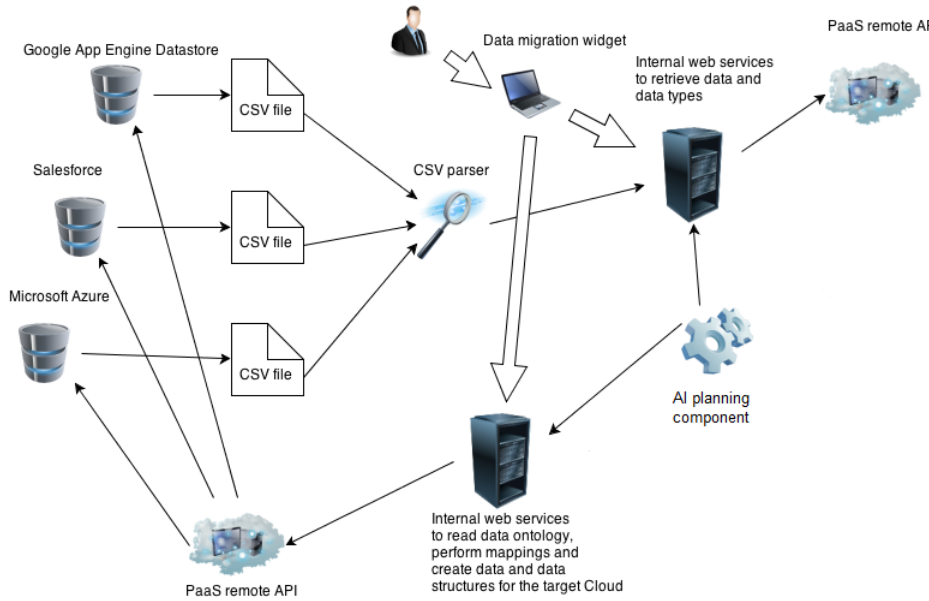

Figure 2. Architecture for data migration between PaaS providers

10 VALIDATION AND ASSESSMENT

10.1 Migration of PaaS Storage Data

The validation of the data migration architecture was done by migrating a more complex set of data and manually checking all of the migrated data elements. For this purpose, sample data of an open-source content management system (CMS) Vosao [42] was chosen. Vosao CMS uses Google App Engine Datastore and initially

consists of 19 entity kinds, 186 entites, and 203 properties (see Table 4 for more details).

Entity Kind	Number of Entities	Number of Properties
ConfigEntity	1	23
ContentEntity	23	9
ContentPermissionEntity	1	10
FieldEntity	3	15
FileChunkEntity	43	8
FileEntity	43	11
FolderEntity	15	8
FolderPermissionEntity	3	8
FormConfigEntity	1	1
FormEntity	1	14
GroupEntity	1	6
LanguageEntity	1	7
PageEntity	26	30
PageTagEntity	13	7
StructureEntity	1	7
StructureTemplateEntity	2	11
TagEntity	5	9
TemplateEntity	2	8
UserEntity	1	11

Table 4. Quantitative information about sample Vosao's data

Its data were then migrated to the other two chosen PaaS offers (Microsoft Azure and Salesforce) to check whether the developed prototype migration tool works smoothly when there is a real cloud application with more data containers and data objects. The implementation of the migration consists of eight main steps:

Prepare migration – We developed two composite web services to transform the source data into an intermediate OWL file, and vice versa: from the intermediate OWL data ontology to concrete data in target PaaS offer's storage. The mentioned composite web services invoke remote PaaS APIs and call our developed services that parse CVS, SAWSDL and JSHOP2 files. Next, they were semantically annotated (classes *CreateDataModelOntologyOperation* and *CreateDataElementsFromOntologyOperation* from the PaaS ontology describe this functionality) using SAWSDL. Finally, a JSHOP2 domain description file is defined manually (see Section 7 for more details).

Export CSV files from PaaS provider – Our prototype first needs to identify the names of entities used by Vosao, because most API data operations deal with only one data container, and entity name is a required input parameter for these operations. Google provides the bulk loader tool in Python SDK that provides

functionalities to download CSV data from a specific application's Google App Engine Datastore (an instance of Vosao CMS application deployed on Google App Engine in our example). Vosao's data consist of 19 entities, so we get 19 separate CSV files, where each file represents one entity.

User chooses action, source, and target of the migration – We start the data migration using our developed client web application. We have an option to choose to port all data, or only one chosen data element (e.g., a specific entity of GAE Datastore), and we select source and target PaaS offers. In our case, we selected Vosao instance as source and two other PaaS offers (our test instances of Salesforce and Microsoft Azure) as targets. For now, our web application enables migration to one target PaaS offer at a given time, so we need to repeat this step twice to migrate Vosao data to the two mentioned target platforms.

Generate problem description file and execute AI plan – JSHOP2 problem description file is generated based on the user's choices, SAWSDL files, and the PaaS ontology (see Section 7). For example, when we choose to move Vosao's data from Google App Engine to Salesforce, the generated goal is:

```
((migrateData GoogleAppEngine SalesForce))
```

Next, the AI planner is executed to see whether there are interoperability problems (e.g. missing operations, missing or impossible data type mappings). If there is no suitable plan returned by the JSHOP2 planner, the client web application displays the error message. In this case, some interoperability problems exist and the cause of the failure needs to be determined. The detailed algorithm is presented in Section 8. In our migration use case there were no identified problems, so we could proceed to the next step described below.

Dynamic invocation of web services – The JSHOP2 and SAWSDL files are parsed to execute adequate web services using Apache CXF framework. For instance, the web services annotated with *CreateDataModelOntologyOperation* in SAWSDL representing Google App Engine's operations and *CreateDataElementsFromOntologyOperation* in SAWSDL file representing Salesforce's operation are executed. Two transformations were performed:

a) *Transformation to unified data model ontology* – On the server side, we implemented web services in Java that parse CSV files, call the appropriate data manipulation remote provider's API to extract details about data (attributes, identifiers, relationship between different data elements), perform data type mappings defined in the PaaS ontology, and use Jena framework to construct the data model ontology according to the rules presented in Section 4. For each Vosao's data store entity, attributes, identifiers, data types, and the number of instances were checked and the conclusion was that the transformation was successful. The data ontology contained all the

entities and data from Vosao's data stored in Google App Engine Datastore.

b) *Transformation to target data model* – During this transformation, the intermediate OWL data file is transformed to data elements in the target PaaS storage. The verification of migration of Vosao's data to Salesforce and Google App Engine was done manually in Excel. All data containers, their names, the names and the number of their attributes, and the number of records were listed there. Additionally, all the data for randomly chosen entities were also checked. Some errors were initially found and bugs in the prototype were fixed until the migration was properly done. In Salesforce, custom objects must have __c postfix, so it is necessary to add these to the names of entities stored in Google App Engine's Datastore. The names of custom fields must also end with __c string. In Excel, the number of properties (of entities from GAE Datastore) and custom fields of each custom objects were compared, and the numbers were identical. Salesforce automatically creates an ID standard field for each object, so the identifier__c custom field was created to save the Google's identifier. When creating a new object, Salesforce always adds some obligatory standard fields (Name, CreatedBy, LastModifiedBy, and Owner). Then, the data record numbers in Google's and Salesforce's platforms were compared, and identical values were obtained. ApexDataLoader tool was used to get data records from Salesforce. Next, the data migrated from Google App Engine to Microsoft Azure was checked in the similar way. The number of properties (of entities from GAE Datastore) and columns of tables created in Microsoft Azure were compared, and the numbers were identical. Then, the data record numbers in Google's and Azure's platforms were compared, and identical values were obtained. Microsoft SQL Server Management Studio tool was used to inspect the data migrated to the Microsoft Azure instance. Finally, some entities were randomly chosen and all the data and mappings of data types in one and the other platform were checked.

Furthermore, the migrated data was taken and put again using the migration tool to a new instance of Google App Engine (datafromazure.appspot.com and datafromsalesforce.appspot.com) and then this data was compared to the original Vosao's data in its original instance of Google App Engine and its underlying datastore. The number and the names of entities, properties and identifiers were manually checked. When migrating from Salesforce, the only difference in data is the identifier, because Salesforce platform automatically assigns identifiers (ID field of each custom object). The same procedure was repeated to migrate data back from Microsoft Azure to the new instance of Google App Engine. The video of the sample PaaS data migration using our developed tool and ontology is available at `https://www.youtube.com/watch?v=tmwoV6XgIhs`.

10.2 Migration of SaaS Storage Data

The same approach was used to migrate sample data of Zoho CRM to Salesforce. We believe that our approach for migrating cloud storage data is general, and we have used the mentioned sample use case to further analyse, test and validate our approach. Zoho CRM [43] provides REST API to access its data. It is a cloud based customer relationship management (CRM) software. We have created a free instance and filled it with initial data. The sample Zoho's data contains 20 standard objects and a total of 553 columns. Each object has been filled with a couple of rows. Zoho's API for the free version supports method *getRecords* for the following standard objects: *Account*, *Campaign*, *Contact*, *Lead*, *Potential*, and *Task*. We have created new mappings to the data ontology from Zoho CRM (similar to the one described in Section 5 for the other three cloud providers) and a new web service that calls Zoho CRM REST API to get the records. Then we have semantically annotated the mentioned web service and added the needed data type mappings in the ontology. The new cloud provider was also added to the AI planning problem. We started the migration, and it was successfully finished, besides that custom objects were created in Salesforce. Salesforce CRM offer has standard objects similar to Zoho's account, campaign, contact, lead, potential and task objects, and to use data directly in Salesforce's cloud application, the migration to standard objects needs to be performed. To successfully migrate data at software as a service level, the standard objects of one CRM offer (e.g. Zoho CRM) need to be translated into standard objects of another CRM (e.g. Salesforce) cloud offer. For this purpose, the data migration architecture described in Section 9 of this work was changed only in the step where intermediate data ontology is created. An adapter Java class that reads the XML file representing standard objects mappings of different SaaS offers (in our test case two different cloud CRM offers) and changes intermediate data ontology representing cloud storage data was developed. Apache Jena was used to rename or delete names of standard data objects (OWL classes in the ontology) and objects' attributes (data properties in the intermediate ontology). The migration was finished successfully, and with this example we have shown that our approach with minor modifications (the addition of simple intermediate data ontology transformation according to mappings of standard objects and their attributes) can also be used for the data migration at SaaS level. The samples of migrated data and generated intermediate OWL data ontologies are available at `https://github.com/dandrocec/PaaSInterop/tree/master/migratedData`.

11 CONCLUSION

There are many data migration problems among cloud providers. To minimize the possible data migration problems in the cloud domain, users should carefully choose a cloud offer, underlying cloud storage systems, and features. It is best to avoid using vendors' specific features that are not supported in any other cloud

offer. For example, most data types problems can be avoided if the established variants of data types (for example, integer, string, etc.) were used and if the usage of new or innovative data types (e.g., Salesforce's anyType, calculated, or DataCategoryGroupReference data type) that cannot be mapped to data types of different cloud storage is avoided. The more users use advanced and innovative functionalities that are vendor specific, the more difficult it will be for migration and interoperability to occur.

In this paper, we proposed a flexible data migration architecture. Our work aims to develop a configurable method for batch migration of data from one cloud data store to another provider's platform. The chosen approach rests on a standard intermediate representation of data and meta-data (in OWL), along with custom-built wrappers for the data manipulation operations available on the source and target platforms. This architecture answers two research questions: How to leverage automatic data migration between cloud solutions? How to provide cross cloud data type mappings for different cloud storage systems used by various PaaS offers? Automatic data migration between cloud offers was done by using OWL as an intermediate format, PaaS ontology, semantic web services implemented in SAWSDL, and AI planner JSHOP2. The validation of the data migration architecture was done by migrating a more complex set of PaaS and SaaS data (concretely, data of the open-source content management system Vosao and data of Zoho CRM) and manually checking all of the migrated data elements. Data type mappings were implemented by means of the ontology, where instances of data type mapping classes were used for this purpose. Our approach enables one-shot migration of cloud data between different types of cloud storage systems (e.g. NoSQL and relational cloud databases). It uses a flexible approach to avoid point-to-point mappings. The main novelty of the paper is a specific application domain (ontology-based migration of data between different cloud offers) and the implementations of mappings among different types of cloud data storage types (NoSQL, relational database, object database). We also provide a comprehensive description of the design and an implementation of the automated cloud data migration solution using state of the art Semantic Web and AI planning techniques. The identified cross-PaaS concepts of the defined PaaS storage data types and their mappings improve the understanding of PaaS and SaaS models in more detail than any other models and ontologies in the existing literature. These concepts also enable semantic annotations and help solve known interoperability problems.

There are several limitations of this work that need to be considered. The AI planning component of this system does not take into consideration the non-determinism of the domain (as an example, some of the remote API operations could be unavailable at specific time; the output of one web service could differ from the expected one, etc.). For this purpose, a contingent planner could be used for planning under uncertainty. Four prominent commercial cloud offers (Google App Engine, Salesforce, Zoho CRM, and Microsoft Azure) were used in this work, and it would be certainly beneficial to include other providers as well.

Some possible future research topics could arise by solving the mentioned limitations. In addition to the JSHOP2 planner that is used in this approach, our architecture could be upgraded to use some contingent planners to address the nondeterminism of the domain. The presented PaaS ontology can be extended including the other cloud providers. The ontology is designed to be easily extended with additional API operations, data types and mappings of data types. Our approach could be integrated with TOSCA and/or other existing multi-cloud orchestration systems, e.g. to migrate web applications together with their data. TOSCA aims to leverage the portability of application layer services between different clouds. Generally, the automatic migration of cloud data is a very complex research and practical problem, and we hope that our paper will be a solid foundation for future research in this field.

Aknowledgement

This work has been fully supported by the Croatian Science Foundation under the project IP-2014-09-3877.

REFERENCES

[1] Sheth, A. P.—Kashyap, V.: So Far (Schematically) Yet So Near (Semantically). Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems, North-Holland Publishing Co., 1993, pp. 283–312, doi: 10.1016/B978-0-444-89879-1.50022-1.

[2] Parent, C.—Spaccapietra, S.: Database Integration: The Key to Data Interoperability. In: Papazoglou, M. P., Spaccapietra, S., Tari, Z. (Eds.): Advances in Object-Oriented Data Modeling, MIT Press, 2000.

[3] Park, J.—Ram, S.: Information Systems Interoperability: What Lies Beneath? ACM Transactions on Information Systems, Vol. 22, 2004, No. 4, pp. 595–632.

[4] Arenas, M.—Barceló, P.—Libkin, L.—Murlak, F.: Foundations of Data Exchange. Cambridge University Press, Cambridge, New York, 2014.

[5] Rocha, L.—Vale, F.—Cirilo, E.—Barbosa, D.—Mourão, F.: A Framework for Migrating Relational Datasets to NOSQL. Procedia Computer Science, Vol. 51, pp. 2593–2602.

[6] The Apache Software Foundation: Welcome to Apache Libcloud's Documentation! 2013, available at: `https://ci.apache.org/projects/libcloud/docs/#main`.

[7] Apache: About Deltacloud. 2013, available at: `http://deltacloud.apache.org/about.html`.

[8] Apache: What Is Apache jClouds? 2013, available at: `http://jclouds.incubator.apache.org/documentation/gettingstarted/what-is-jclouds/`.

[9] Youseff, L.—Butrico, M.—Da Silva, D.: Toward a Unified Ontology of Cloud Computing. Grid Computing Environments Workshop (GCE '08), IEEE, 2008, pp. 1–10, doi: 10.1109/GCE.2008.4738443.

[10] Ranabahu, A.—Sheth A.: Semantics Centric Solutions for Application and Data Portability in Cloud Computing. IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom 2010), Indianapolis, 2010, pp. 234–241, doi: 10.1109/CloudCom.2010.48.

[11] Quinton, C.—Romero, S.—Duchien, L.: SALOON: A Platform for Selecting and Configuring Cloud Environments. Software: Practice and Experience, Vol. 46, 2016, No. 1, pp. 55–78.

[12] Tsai, W.-T.—Sun, X.—Balasooriya, J.: Service-Oriented Cloud Computing Architecture. 2010 Seventh International Conference on Information Technology: New Generations (ITNG), IEEE, 2010, pp. 684–689, doi: 10.1109/ITNG.2010.214.

[13] Bastião Silva, L. A.—Costa, C.—Oliveira, J. L.: A Common API for Delivering Services over Multi-Vendor Cloud Resources. Journal of Systems and Software, Vol. 86, 2013, No. 9, pp. 2309–2317, doi: 10.1016/j.jss.2013.04.037.

[14] Gogouvitis, S. V.—Kousiouris, G.—Vafiadis, G.—Kolodner, E. K.—Kyriazis, D.: OPTIMIS and VISION Cloud: How to Manage Data in Clouds. In: Hutchison, D., Kanade, T., Kittler, J. et al. (Eds.): Euro-Par 2011: Parallel Processing Workshops. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7155, 2012, pp. 35–44.

[15] Scavuzzo, M.—Di Nitto, E.—Ceri, S.: Interoperable Data Migration Between NoSQL Columnar Databases. 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), Ulm, 2014, pp. 154–162, doi: 10.1109/EDOCW.2014.32.

[16] Shirazi, M. N.—Kuan, H. C.—Dolatabadi, H.: Design Patterns to Enable Data Portability Between Clouds' Databases. 12th International Conference on Computational Science and Its Applications (ICCSA '12), Salvador, Brazil, IEEE, 2012, pp. 117–120, doi: 10.1109/ICCSA.2012.29.

[17] Ali, H.—Moawad, R.—Hosni, A. A. F.: A Cloud Interoperability Broker (CIB) for Data Migration in SaaS. IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA '16), Chengdu, China, IEEE, 2016, pp. 250–256, doi: 10.1109/ICCCBDA.2016.7529566.

[18] Bansel, A.—González-Veléz, H.—Chis, A. E.: Cloud-Based NoSQL Data Migration. 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP '16), Heraklion, Greece, IEEE, 2016, pp. 224–231, doi: 10.1109/PDP.2016.111.

[19] Di Martino, B.: Applications Portability and Services Interoperability among Multiple Clouds. IEEE Cloud Computing, Vol. 1, 2014, No. 1, pp. 74–77, doi: 10.1109/MCC.2014.1.

[20] Natis, Y.—Pezzini, M.—Driver, M.—Smith, D. M.—Iijima, K.—Altman, R.: Magic Quadrant for Enterprise Application Platform as a Service (Jan. 2014). Available at: http://www.gartner.com/technology/reprints.do?id=1-1P502BX&ct=140108&st=sb.

[21] Salesforce: Database.com Workbook (Jun 2013). Available at: http://www.salesforce.com/us/developer/docs/workbook_database/workbook_database.pdf.

[22] Google: Google App Engine – Storing Data (Jun 2013). Available at: https://developers.google.com/appengine/docs/java/datastore/.

[23] Google: Google Cloud SQL (May 2013). Available at: https://developers.google.com/cloud-sql/.

[24] Franks, L.: Data Storage Offerings on the Windows Azure Platform (Oct. 2010). Available at: http://social.technet.microsoft.com/wiki/contents/articles/1674.data-storage-offerings-on-the-windows-azure-platform.aspx.

[25] Auer, S.—Feigenbaum, L.—Miranker, D.—Fogarolli, A.—Sequeda, J.: Use Cases and Requirements for Mapping Relational Databases to RDF. W3C working draft (Jun 2010). Available at: http://www.w3.org/TR/rdb2rdf-ucr/.

[26] Astrova, I.—Korda, N.—Kalja, A.: Rule-Based Transformation of SQL Relational Databases to OWL Ontologies. Proceedings of the 2nd International Conference on Metadata and Semantics Research, 2007. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.8189.

[27] Apache: Apache CXF Dynamic Clients. 2013, available at: http://cxf.apache.org/docs/dynamic-clients.html.

[28] Noy, N. F.—McGuinness, D. L.: Ontology Development 101: A Guide to Creating Your First Ontology. 2001, available at: http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf.

[29] SOFTEAM SINTEF Tecnalia: REMICS Deliverable d4.1 PIM4cloud. Project deliverable, REMICS Consortium (Mar. 2012). Available at: http://www.remics.eu/system/files/REMICS_D4.1_V2.0_LowResolution.pdf.

[30] Norton, B.—Kerrigan, M.—Mocan, A.—Carenini, A.— Cimpian, E.—Haines, M.—Scicluna, J.—Zaremba, M.: Reference Ontology for Semantic Service Oriented Architectures. Public Review Draft 01, OASIS (Nov. 2008). Available at: http://docs.oasis-open.org/semantic-ex/ro-soa/v1.0/pr01/see-rosoa-v1.0-pr01.pdf.

[31] Andročec, D.—Vrček, N.: Ontologies for Platform as Service APIs Interoperability. Cybernetics and Information Technologies, Vol. 16, 2016, No. 4, pp. 29–44, doi: 10.1515/cait-2016-0065.

[32] Google: Entities, Properties, and Keys. 2013, available at: https://cloud.google.com/appengine/docs/standard/java/datastore/entities.

[33] Microsoft: Data Types (Windows Azure SQL Database). 2013, available at: http://msdn.microsoft.com/en-us/library/windowsazure/ee336233.aspx.

[34] Salesforce: SOAP API Developer's Guide Version 28.0. 2013, available at: https://developer.salesforce.com/docs/atlas.enus.api.meta/api/sforce_api_quickstart_intro.htm.

[35] Bechhofer, S.—van Harmelen, F.—Hendler, J.—Horrocks, I.—McGuinness, D. L.—Patel-Schneider, P. F.—Stein, L. A.: OWL Web Ontology Language Reference. 2004, available at: http://www.w3.org/TR/owl-ref/.

[36] W3C: XML Schema Part 2: Datatypes Second Edition. 2004, available at: http://www.w3.org/TR/xmlschema-2/.

[37] Sheth, A. P.—Gomadam, K.—Ranabahu, A.: Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST. IEEE Data Engineering Bulletin, Vol. 31, 2008, No. 3, pp. 8–12.

[38] SOA4All Consortium: SOWER. 2010, available at: http://technologies.kmi.open.ac.uk/soa4all-studio/provisioning-platform/sower/.

[39] Ilghami, O.: Documentation for JSHOP2. 2006, available at: http://sourceforge.net/projects/shop/files/JSHOP2/.

[40] Ilghami, O.—Nau, D. S.: A General Approach to Synthesize Problem-Specific Planners. Technical report CS-TR-4597 and UMIACS-TR-2004-40, 2003. Available at: http://www.cs.umd.edu/~nau/papers/ilghami2003general.pdf.

[41] Goebelbecker, M.—Keller, T.—Eyerich, P.—Brenner, M.—Nebel, B.: Coming Up with Good Excuses: What to Do When No Plan Can Be Found. Proceedings of the 20th International Conference on Automated Planing and Schedulling (ICAPS 2010), 2010, pp. 81–88.

[42] Husted, T.: Vosao – Google App Engine CMS. 2013, available at: https://code.google.com/p/vosao/.

[43] Zoho: Zoho CRM – An Overview. 2017, https://www.zoho.eu/crm/help/overview.html.

Darko Andročec is Senior Teaching Assistant at the University of Zagreb, Faculty of Organization and Informatics Varaždin, where he was awarded his Ph.D. degree in 2015. He is a member of the Department for Information System Development. Before joining the faculty, he was a computer security incident handler at CARNet (Croatian Academic and Research Network) and Java developer of banking information systems. He currently teaches computer labs in the following courses: Information Systems Development, E-Business, and Business Processes in Organizations. His main research areas are cloud computing, interoperability, semantic web, and internet of things.

Neven Vrček is Full Professor at the Faculty of Organization and Informatics, University of Zagreb. He is the main lecturer at several courses: Software Engineering, Software Analysis and Design, E-Commerce, ERP Systems and Customer Relationship Management. He graduated at the Faculty of Electrical Engineering, University of Zagreb. He defended his master theses as well as his Ph.D. dissertation at the same faculty. His fields of interests are: strategic planning of information systems development, e-commerce and IT applications in business sector, business performance measurement, and digital signal processing. He was a leader or a team member of several scientific projects financed by the EU institutions, Croatian Ministry of Science, Education and Sports, and Croatian Science Foundation and of more than 30 commercial projects. He was appointed as Dean of the Faculty of Organization and Informatics, University of Zagreb in October 2015.