BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **16**, No 4
Sofia • 2016 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2016-0065

Ontologies for Platform as Service APIs Interoperability

Darko Andročec, Neven Vrček

Faculty of Organization and Informatics, University of Zagreb, Pavlinska 2, 42000 Varaždin, Croatia
Emails: dandrocec@foi.hr nvrcek@foi.hr

Abstract: *Ontologies can be used to describe common cloud functionalities and enable common terminology to assist in cloud interoperability. In this work, we have developed the ontology for resources and operations and the ontology of interoperability problems. The aim is to clearly describe and categorize the existing functionalities, features and specificities of commercial platform as a service offers. The first ontology also provides data type mappings among different PaaS storages and cross-PaaS data types used in inputs and outputs of the remote API operations to provide a common layer for information exchange and data migration among different PaaS providers. The ontologies were evaluated by tools and by human experts. Furthermore, the ontologies were used in cloud interoperability prototype to show their practical applicability.*

Keywords: *Cloud ontology, interoperability, platform as a service, remote APIs, interoperability problems.*

1. Introduction

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. The aforementioned problem is characterized by time-consuming and costly migration of application and data to alternative cloud solutions offered by different vendors, the inability or limited ability to use some computing resources, applications or data outside the selected cloud computing service and the dependence on a specific programming language used by the selected cloud computing vendor. The numerous heterogeneities among different vendors make cloud interoperability an interesting and complex research and practical problem.

Cloud computing ontologies are predominantly applied in the description, discovery and selection of the best service alternative in accordance with users' requirements. The existing cloud computing ontologies are mostly general and detailed ontologies of each cloud computing layer (software as a service, platform as a service and infrastructure as a service) are still missing. The most mature ontology is mOSAIC [1] ontology, but it is focused on infrastructure as a service model and SLA.

The ontologies presented in this paper are focused on remote operations of PaaS providers' APIs and interoperability problems among different platform as a service offers. Previously, we have published a draft of possible platform as a service ontology [2] that lists some sample cloud API operations and resources. We have built on our mentioned previous work and have completely redesigned the ontology by changing class hierarchy and including complete API operations of three platforms as service providers (Microsoft, Google, and Salesforce). Additionally, the ontology of PaaS resources, remote operations, and data types presented in this work supports data mappings among the heterogeneous APIs. 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. Furthermore, we have also developed the additional ontology that lists the technical and semantic interoperability problems of commercial platform as a service offers. On the end, the two new ontologies have been evaluated by tools and human experts, and have been refined according to evaluation results.

This paper proceeds as follows. First, in Section 2, the related work is listed. In Section 3 we show the development of our ontologies. Section 4 deals with evaluation of the developed ontologies. Our conclusions are provided in the final section.

2. Related work

2.1. Cloud ontologies

There are several existing studies involving cloud computing ontologies. One of the first attempts was introduced in Youseff, Butrico and Da Silva [3]. They presented an ontology which differentiates five main layers of cloud computing (applications, software environments, software infrastructure, software kernel and hardware). Weinhardt et al. [4] proposed a cloud business ontology model to classify current cloud services and their pricing models into three layers: infrastructure as a service, platform as a service and application as a service. Deng et al. [5] introduced a formal catalogue of cloud computing services modelled by means of ontological representation. Takahashi, Kadobayashi and Fujiwara [6] applied the ontology for cyber security to cloud computing. Martinez, Echeverri and Sanz [7] used the ontology for malware and intrusion detection based on cloud computing and created an ontological model for reaction rules that could form the prevention system.

The concepts of the mOSAIC's cloud ontology [1] were identified by analyzing standards and the existing cloud interoperability and integration works from literature. This ontology is used for retrieval and composition of cloud services in mOSAIC's usage scenarios. Bernstein and Vij [8] developed a mediator to enable collaboration among different cloud vendors. They defined the ontology of cloud computing resources using RDF. Han and Sim [9] presented a cloud service discovery system with ontology determining the similarities among different cloud offers. They created agent-based discovery system to assist users in searching the available cloud services. K a n g and S i m [10] proposed cloud ontology to define the relationship between different cloud services. They used similarity reasoning of concepts, object properties, and data properties. In the same paper, they presented their own search engine that uses the defined ontology to retrieve cloud service compatible with user's requirements. D a s t j e r d i, T a b a t a b a e i and B u y y a [11] presented an ontology-based discovery architecture providing QoS-aware deployment of virtual appliances on infrastructure as a service. M a, S c h e w e and W a n g [12] presented clouds formalism by a description of cloud services in the form of ontology. These descriptions contain service types, pre- and post-conditions, and keywords that describe the functionality of the annotated service.

2.2. Ontology anomalies and ontology evaluation

There are some ontology anomalies and pitfalls that can arise during ontology modelling. P o v e d a-V i l l a l ó n, S u á r e z-F i g u e r o a and G ó m e z-P é r e z [13] manually inspected pitfalls in ontologies of 26 students. They have identified 24 pitfalls and classified them into [13]: consistency (creating polysemous elements, defining wrong inverse relationships, including cycles in the hierarchy, merging different concepts in the same class, misusing "allValuesFrom", misusing "not some" and "some not", specifying wrong the domain or the range, swapping intersection and union, using recursive definitions), completeness (unconnected ontology elements, missing basic information, missing domain or range in properties, missing equivalent properties, missing inverse relationships, misusing primitive and defined classes), and conciseness (creating synonyms as classes, creating the wrong relationship, specializing a hierarchy too much, using a miscellaneous class). In their other work [14], the same authors presented a web based tool called OOPS! that can detect the mentioned anomalies in OWL ontology. B a u m e i s t e r and S e i p e l [15] explored anomalies in ontologies used with rule extensions. They distinguish four categories of anomalies: circularity (exact circularity in taxonomy and rules, circularity between rules and taxonomy, circular properties), redundancy (identity errors, redundancy by repetitive taxonomic definition, rule subsumption, redundant implication, redundant implication of transitivity or symmetry, redundancy in the antecedent of a rule, etc.), inconsistency (partition error in taxonomy, incompatible rule antecedent, self-contradicting rule, contradicting rules, multiple functional properties), deficiency (lazy class/property, chains of inheritance, lonely disjoint class, property clump).

The evaluation of ontology was discussed in many of the existing works. Ontology can be evaluated by itself, with some context, within an application, and in the context of an application and a task [16]. G o m e z-P e r e z [17] divides ontology evaluation into ontology verification and ontology validation. V r a n d e č i ć [16] analyzed the ontology quality criteria, and summarized them into the following important criteria: accuracy (the axioms of the ontology must comply to the domain expert's knowledge; classes, properties, and individuals must be correctly defined), adaptability (the ontology can be extended and specialized without the need to remove the existing axioms), clarity (ontology should clearly communicate the meaning of its elements by using concise element names and documentation), completeness (the domain of the ontology must be appropriately covered), computational efficiency (the reasoning complexity and the ability of tools to efficiently work with the ontology), conciseness (only essential ontology elements should be defined, irrelevant or redundant elements should be removed), consistency (there are no contradictions in the ontology), and organizational fitness (how easily an ontology can be used within an organization). Competency questions are defined to describe what knowledge the specific ontology must possess [16]. These questions can be formalized in a semantic query language.

B r a n k, G r o b e l n i k and M l a d e n i ć [18] differentiate four main ontology evaluation approaches: comparison of the ontology to the gold standard, using ontology in an application and evaluating the results, comparison to the data about the domain and human evaluation. Ontology is a complex structure, so B r a n k, G r o b e l n i k and M l a d e n i ć [18] propose evaluation separately on each level of the ontology: lexical layer; hierarchy; other semantic relations; context or application level; syntactic level; and structure, architecture and design level. A m i r h o s s e i n i and S a l i m [19] listed three main approaches for ontology evaluations: gold standard evaluation (comparison with benchmark ontology), task-based evaluation (Can the ontology complete the pre-defined tasks?), and criteria-based evaluation (human evaluation based on some criteria).

3. Development of the ontologies

3.1. Selected ontology development methodology, tool and language

For the purpose of this research, the Ontology Development 101 [20] 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. Ontology Development 101 is designed as a simple iterative methodology and a starting guide for new ontology designers to develop their own ontologies. Furthermore, it is also well aligned with the used tool (Protégé) and it provides working examples for this ontology editor. The open-source tool Protégé was selected because it is free and currently most used tool for ontology development. As an illustration, Protégé has more than 240,000 registered users at the moment. Protégé has many useful plug-ins, including the ones for semantic queries, ontology reasoning and ontology visualizations. Web Ontology Language (OWL) was chosen because it has the needed expressive power and is most widely used language for ontologies in the papers in the field of computer science and research projects related to this field of study.

Now, the main steps of the selected ontology will be listed. N o y and M c G u i n n e s s [20] claim that the development of the ontology includes defining classes and their hierarchy, defining their properties and instances. The ontology development process is iterative: an initial version is built, this version is checked in applications or by experts, and it is refined until usable ontology is obtained. There are seven steps in Ontology Development 101 methodology [20]:

1. Determine the domain and scope of the ontology – first step includes defining ontology's domain and scope by using competency questions (questions that the ontology should be able to answer).

2. Consider reusing the existing ontologies – checking whether the existing ontologies can be refined and extended.

3. Enumerate important terms in the ontology – write down all the possible relevant terms without worrying about the overlap between concepts.

4. Define the classes and the class hierarchy – using top-down or bottom-up approach, or the combination of the two, to define classes and their hierarchy.

5. Define the properties of classes – slots – here the internal structure of concepts is defined using data and object properties.

6. Define the facets of the slots – the value type, allowed values, domain, range, and cardinality of slots should be defined.

7. Create instances – the individual instances of classes should be defined and their slot values should be filled.

As part of their published document, N o y and M c G u i n n e s s [20] showed how to create sample Wine ontology using the above mentioned steps. In the next chapter, the Ontology Development 101 methodology is used to create ontology of PaaS resources, remote operations and data type mappings.

3.2. Ontology of PaaS resources, remote operations, and data types

For the purpose of this research, the domain and scope of the model should be limited as in the first step of Ontology Development 101 [20] guide. The representation of resources and operations in APIs of platform as a service is determined as the domain of the ontology. This ontology will be used to semantically annotate API operations of platform as a service offers. The information in the ontology should provide answers to the following questions: What are the main resources of the platform as a service model of cloud computing? What are the most important remote operations on PaaS resources? How to support mappings of data types among the heterogeneous APIs? 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.

First, the work of the other authors was considered and checked if there was a possibility to refine and extend the existing ontologies for the domain and scope determined in the previous step. The most important previous work related to cloud and PaaS ontologies is listed in Section 2.1. There is no complete ontology that is focused on remote operations providers of commercial platform as a service and data type mappings among them, but some concepts from our previous work [2], mOSAIC ontology [1] and D e n g et al. [5] were used as important terms for development of this ontology.

Excel spreadsheets were used to list all relevant terms, one sheet per one relevant document. Initially, the concepts in this ontology were derived from the existing cloud ontologies (mostly from mOSAIC project), PIM4Cloud [21] metamodel from REMICS project, OASIS Reference Ontology for Semantic Service Oriented Architecture [22], relevant related works from literature ([3]), 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. Experimental remote APIs are not included, because they are subject to frequent change, and providers do not guarantee that they will keep these operations in the next versions of their APIs. Terms obtained from these sources are listed in Table 1. The list of terms was incrementally updated during the whole research.

Table 1. List of identified terms for PaaS ontology

Source	**Important terms**
D e n g et al. [5]	Service offering, composite offering
mOSAIC ontology – M o s c a t o et al.[1]	API, data storage, replicated relational database, key value stores, distributed file system, language, application, utility API, data management API, authentication API, platform provider, cloud resources
OWL-S [23]	Service, variable, parameter, input, output, result, precondition
WSMO [24]	Web service, precondition, assumption, postcondition, effect
OCCI [25]	Entity, resource, kind, action
TOSCA [26]	Properties, capabilities, interfaces, operation, requirements
CDMI [27]	Container, data object, queue object
Salesforce's APIs ([28], [29]) – list of remote operations	Convert lead, create, delete, empty recycle bin, get deleted, get updated, invalidate sessions, login, logout, merge, process, query, query all, query more, retrieve, search, undelete, update, upsert, describe global, describe data category groups, describe data categories group structures, describe layout, describe search scope order, describe SObject, describe softphone layout, describe tabs, get server timestamp, get user info, reset password, send email, send email message, set password, deploy metadata, check deploy status of metadata, retrieve metadata, create metadata, delete metadata, update metadata, check status of metadata, describe metadata, list metadata
Google App Engine APIs ([30], [31]) – list of remote operations	Put, get, delete, query, begin transaction, commit transaction, rollback transaction, resize images, rotate images, flip images, crop images, logs, send email, search application data, queues, fetch URL, authenticate users, send and receive instant messages
Microsoft Azure APIs [32] – list of remote operations	Set table service properties, get table service properties, query tables, create table, delete table, get table ACL, set table ACL, query entities, insert entity, merge entity, replace entity, update entity, delete entity, list containers, set BLOB service properties, get blob service properties, create container, get container properties, get container metadata, set container metadata, get container ACL, set container ACL, lease container, delete container, list blobs, put blob, get blob, get blob properties, set blob properties, get blob metadata, set blob metadata, delete blob, lease blob, snapshot blob, copy blob, abort copy blob, put block, put block list, get block list, put page, get page ranges, set queue service properties, get queue service properties, list queues, create queue, delete queue, get queue metadata, set queue metadata, get queue ACL, set queue ACL, put messages, get messages, peek messages, delete messages, clear messages, update message
REMICS PIM4Cloud [21]	PaaS resource, communication resource

From the list created in the previous step, the terms describing independent objects were selected to present classes in the ontology. In OWL, classes are used to group individuals that have something in common and that represent sets of individuals. A class can have subclasses, so the classes were organized into a hierarchical taxonomy. A total of 146 classes were defined that are organized in 17 top level classes (Fig. 1).

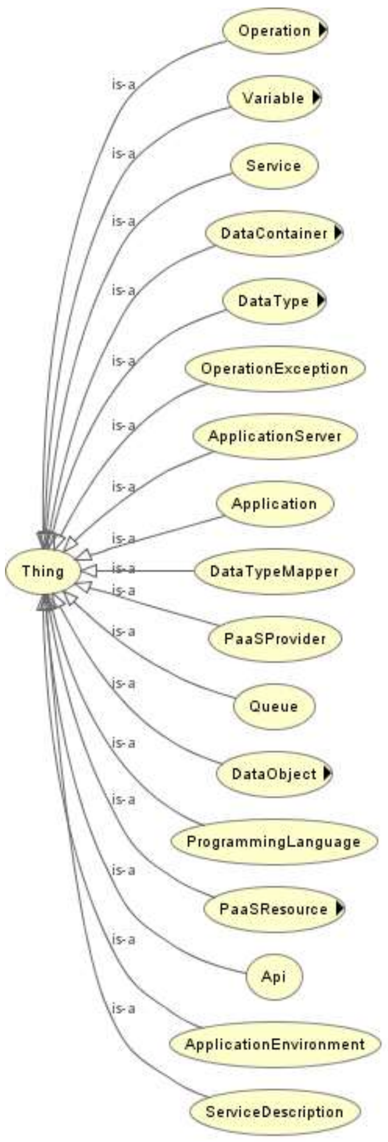

Fig. 1. Top level classes of PaaS ontology

The properties of classes describe the internal structure of concepts. Properties specify how the instances of a class relate to other instances. Property cardinality defines how many values a property can have. The allowed classes for a property instance are called a range of a property, and the classes that the property describes are called the domain of the property [20]. Apart from having a domain and a range, an object property may have super- and sub-properties, inverse properties, equivalent properties and property chains. A total of 34 object properties were defined in the first ontology. Additionally, instances can be described by data values. For this purpose, OWL provides data type properties that relate instances to data values (instead of relating them to other instances). A total of 30 data properties were defined. The last step in the methodology devised by N o y and M c G u i n n e s [20] is filling in the values for instances. It requires the creation of individual instances of each relevant class. For now, a total of 426 individuals were created. This number is obtained from ontology documentation created by using *OWLDoc* plugin in Protégé, and DL Query was used to obtain the number of instances per each OWL class. Most of the created instances are used for data type mappings between cloud storage of different PaaS vendors. For example, OWL class *DataTypeMapper* has 178 instances, and *CloudStorageDataType* has 124 instances. Our ontology is publicly available at **https://github.com/dandrocec/PaaSInterop/blob/master/PaaSOntology5/PaaSOntologyv5.owl**.

3.3. Ontology of platform as service interoperability problems

The second ontology was also developed using Ontology Development 101 methodology [20], OWL and Protégé tool. The domain of this ontology is the representation of the technical and semantic interoperability problems of commercial platform as a service offers. The ontology will be used in the methodology for detecting interoperability problems among providers of platform as a service as a comprehensive list of possible interoperability issues. The information in the ontology should give answers to the following question: What are the most important interoperability problems among different platform as a service offers?

N a u d e t et al. [33] developed a general ontology of interoperability that can be used as a starting point for this ontology of platform as service interoperability. Their ontology is based on system theory and aims at defining interoperability in a more formal way and it is the basis for allowing interoperability problem detection, and suggesting solutions [33]. The general interoperability concepts from their ontology that can be applied to platform as a service APIs interoperability (e.g., Interoperability, *AprioriSolution*, *AposterioriSolution*, *Problem*, etc.) and relations between them will be directly used in our ontology.

Again, Excel spreadsheets were used to list all the relevant terms. The concepts of the ontology of interoperability problems were derived from Naudet et al.'s ontology of interoperability [33], interoperability problems between different databases listed in the literature ([34-37]) metadata interoperability problems [38], interoperability problems of web services ([39], [40]) the ATHENA Interoperability Framework [41] and problems identified by the author of this dissertation when working on use cases. Terms obtained from these sources are listed in Table 2.

Table 2. List of identified terms for PaaS interoperability ontology

Source	Important terms
Naudet et al. [33]	InteroperabilitySolution, Indicator, InteroperabilityProblem, InteroperabilityExistenceCondition, Model, ConformancePoint, AntiPattern, InteroperabilitySolution, AprioriInteroperabilitySolution, AposterioriInteroperabilitySolution, Incompatibility, Misalignment, Heterogeneity, actsOnApi, actsOnModel, actsOnRepresentation, canInduceNewProblem, concernsApi, concernsModel, concernsRepresentation, definesCondition, existsIf, solvesProblem
Park and Ram [34]	DataLevelConflict, DataValueConflict, DataRepresentationConflict, DataUnitConflict, DataPrecisionConflict, SchemaLevelConflict, NamingConflict, EntityIdentifierConflict, SchemaIsomorphismConflict, GeneralizationConflict, AggregationConflict, SchematicDiscrepancies
Cloud4SOA [42]	Different data models, different APIs, different query languages
Haslhofer and Klas [38]	Metadata heterogeneities, structural heterogeneities, domain representation conflicts, abstraction level incompatibility, multilateral correspondences, meta-level discrepancy, domain coverage, element definition conflicts, naming conflicts, identification conflicts, constraints conflicts, semantic heterogeneities, domain conflicts, terminological mismatches, scaling/unit conflicts, representation conflicts
Parent and Spaccapietra [36]	Generalization/specialization conflicts, description conflicts, structural conflicts, fragmentation conflicts, metadata conflicts, data conflicts
Sheth and Kashyap [35]	Domain definition incompatibility, naming conflicts, data representation conflicts, data scaling conflicts, data precision conflicts, default value conflicts, attribute integrity constraint conflicts, entity definition incompatibility, database identifier conflicts, union compatibility conflicts, schema isomorphism conflicts, missing data item conflicts, data value incompatibility, known inconsistency, temporary inconsistency, acceptable inconsistency, aggregation conflicts, generalization conflicts, data value attribute conflict, attribute entity conflicts, data value entity conflicts
Ponnekanti and Fox [39]	Structural, value, encoding and semantic incompatibilities, missing methods, extra fields, missing fields, facet mismatches, cardinality mismatches
Zhu et al. [37]	Naming synonyms, naming homonyms, different composite structure, different value representation, differences in semantic meaning, differences between data models, changes over time of the structure and the representation of attributes and values, different query languages, different transaction mechanisms
AIF [41]	Interoperability at enterprise/business level, interoperability of processes, interoperability of services, interoperability of information/data

Again, from the list created in the previous step, the terms that describe independent objects were selected, because they present classes in the ontology. A total of 78 classes were defined. Also, a total of 14 object properties were defined. For now, the ontology does not contain any data properties. Our PaaS interoperability OWL ontology is publicly available at **https://github.com/dandrocec/PaaSInterop/blob/master/InteroperabilityProblemsOntology.owl**.

4. Evaluation of the ontologies

Ontology evaluation gathers information about some properties of the ontology, compares the results with a set of requirements, and assesses the suitability of the ontology for some specified purpose [43]. 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 question of choosing the ontology evaluation method is still one of the biggest problems in ontology engineering. There is no consensus on the best ontology evaluation approach and there exist no universally agreed metrics for ontology evaluations [43], but evaluating the ontology systematically during its whole lifecycle will certainly raise its quality. Ontology anomalies and main approaches to tackle ontology evaluation are presented in Section 2.2 of this work. Neuhaus et al. [43] claim that ontology evaluation should be incorporated into all ontology development lifecycle phases based on carefully identified ontology requirements. Due to a lack of gold standards and corpus of data, the evaluation by humans and application-based evaluation was chosen. Additionally, some tools were used to eliminate OWL syntax errors and known ontology anomalies. In the next subsections, the evaluation process of developed ontologies will be shown.

4.1. Evaluation by tools

First, the logical consistency of the developed ontologies was checked by means of the Pellet reasoner that checks hierarchies, domains, ranges, conflicting disjoint assertions and calculates the resulting inferred hierarchy and other properties. Pellet uses logic to draw inferences from the facts and axioms defined in the OWL ontology. Pellet reasoner plug-in for Protégé 4 was installed and executed, and no consistency problems were found.

Next, the DL Query was used to check whether the ontology meets the basic requirements. DL Query is a Protégé 4 plug-in [44], and the supported query language is based on Manchester OWL syntax. For example, DL Query "Operation" can be executed to get all subclasses, descendant classes and individuals of the Operation class. Then vendor's documentation of their remote API operations can be observed, and it should be checked if all the relevant operations were included in the ontology. Other relevant DL Query can be

“DataTypeMapper” to check whether all relevant data type mappings are present as individuals in our ontology.

Furthermore, the web based tool called Ontology Pitfall Scanner! (OOPS!) [14] was used to detect possible ontology anomalies. The mentioned tool can currently identify 40 ontology pitfalls. The two ontologies in this dissertation were evaluated using publicly available OOPS! tool. One critical (swapping intersection and union) and three important (untyped property) pitfalls were found and eliminated.

4.2. Evaluation by humans

Ontology was also evaluated by four human experts working in the field of cloud computing interoperability and related science projects (Contrail [45], mOSAIC [1]). The questionnaire was sent to four cloud researchers. They were sent a brief ontology description document with figures of class hierarchy, and asked to answer the following questions:

1. Completeness

Do the ontologies cover the major concepts regarding PaaS API operations and PaaS interoperability problems? Are there any concepts/terms that you recommend to add to the ontologies and where?

2. Conciseness

Can you identify some redundant or ambiguous concepts in the ontologies? Do you think that some concepts should be removed and why?

3. Consistency

Can you identify some inconsistencies (for example, contradictions, semantic duplication, or circular definitions) in the provided ontologies?

4. Flexibility

Can new concept/s be included into the ontologies without revising their existing structures?

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 which offered more comments on newer versions of the ontologies. Several pitfalls were found by four experts. The findings, together with the actions taken, are shown in Table 3.

4.3. Application based evaluation

The aim was to validate the usability of these ontologies to semantically annotate remote vendors’ PaaS API operations, to enable mapping between their inputs and outputs, and to enable mappings of different types between different PaaS storages. The prototype was developed in Java and it uses Jena library to work with the ontologies. The developed prototype demonstrates the feasibility of applying the ontologies to semantically annotate API operations, find interoperability problems, and try to find solution for the problems found. The source code of the prototype is publicly available at
https://github.com/dandrocec/PaaSInterop

Table 3. Summary of ontology evaluation by experts

Expert's comments	**Actions taken**
• Authentication describes authT towards the PaaS portal? AuthT against application developed within the PaaS? If second, maybe alternative (e.g., ×509) authentication operations can be added (there is *GetPublicCert* operation)? • You could add *RegistrationOperation* in parallel to *AuthenticationOperation* • I have not seen any operations/concepts related to accounting/monitoring/billing/alerting. How is that? Is this maybe included in some operation? • However, I believe that your concepts cover most of the operations • New operations can be added without revising other concepts in the ontology	• *AddServiceCertificateOperation* and *DeleteServiceCertificateOperation* were added • *RegistrationOperation* is added to the ontology • *MonitoringOperation, ResourceUsageOperation, BillingOperation, UpdateAlertRuleOperation, ListAlertRulesOperation, GetAlertRuleOperation, DeleteAlertRuleOperation, CreateAlertRuleOperation* were added to the ontology
• The ontology seems pretty extensive and consistent to me, although slightly different from the one developed in mOSAIC	• None
• My first impression is that the ontologies are too abstract, i.e., not very "practical" • The best way to proceed would be to include some instance data in Protégé and prepare some SPARQL queries that would be useful in your given context – that would demonstrate its usage	• More instance data was included
• I would suggest inspecting Cloud API-s such as Dasein Cloud API, Apache jclouds, etc, where standardization has been performed for accessing clouds in a provider-independent way • I saw some potential anomalies, such as e-mail address being a concept/class • Go through the instances to add more assertions • What about mappings between complex types? • With respect to ontology sources I suggest to also look at the REMICS-related metamodels • Also, please unify the naming of classes and properties • You model all data structures of specific PaaS solutions in the ontology with dedicated entities instead of defining cross-PaaS concepts – why was this choice made? This means that in order to add support for other PaaS' you need both – extend the ontology and create new mappings, while with cross-PaaS conceptualization creation of new mapping might suffice	• Additional ontology sources were inspected • Email class is removed from the ontology because it was an anomaly • More instance assertions were added • Complex types mappings were listed in the PaaS ontology • The naming of classes and properties were unified • In the final version of PaaS ontology, cross-PaaS concepts are used to model simple and complex data types of services' inputs and outputs

5. Conclusion

This work described the development of two ontologies. The mentioned ontologies describe functionalities, features and interoperability problems among APIs of different providers of platform as a service. The first ontology provides data type mapping among different PaaS storages and cross-PaaS data types used in inputs and outputs of the operations. This functionality provides a common layer for information exchange and data migration among different PaaS providers. The logical consistency of the ontologies was checked and four human experts evaluated the ontologies. Furthermore, the ontologies were used in cloud interoperability prototype to show their practical applicability.

Key indicators of the existence of interoperability problems among the available platform as a service APIs can be found in the description of the subclasses of *InteroperabilityProblem* OWL class in the second shown ontology. The developed ontologies improve the understanding of PaaS offers, their operations and data type, and enable mappings to overcome their differences. Identified cross-PaaS concepts of operation, input and output data types, as well as defined PaaS storage data types and their mappings improve the understanding of platform as a service model in more detail than other models and ontologies in the existing literature. These concepts also enable semantic annotations with aim to solve known interoperability problems.

Three prominent commercial offers of platform as a service (Google App Engine, Salesforce and Microsoft Azure) were used to define main types of API functions in the ontology. Their APIs represent most of the functionalities found today in platform as a service offers, but it would be certainly beneficial to also include other providers. The ontology is designed to be easily extended with additional API operations, data types and mappings of data types. Another direction for future work could be to extend the ontology to support API functions and interoperability problems of the other two main models of cloud computing (software as a service and infrastructure as a service). Generally, the interoperability of platform as a service and cloud computing in general are very complex and important issues, and hopefully, the ontologies presented in this work will extend the knowledge of cloud APIs and their interoperability problems and allow for gradual resolution of cloud interoperability problems.

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

References

1. M o s c a t o, F., R. A v e r s a, B. D i M a r t i n o, T.-F. F o r t i s, V. M u n t e a n u. An Analysis of mOSAIC Ontology for Cloud Resources Annotation. – In: Proc. of Federated Conference on Computer Science and Information Systems, Szczecin, 2011, pp. 973-980.
2. A n d r o č e c, D., N. V r č e k. Platform as a Service API Ontology. – In: Proc. of 12th European Conference on eGovernment, Barcelona, 2012, pp. 47-54.

3. Y o u s e f f, L., M. B u t r i c o, D. D a S i l v a. Toward a Unified Ontology of Cloud Computing. – In: Grid Computing Environments Workshop (GCE'08), Austin, Texas, 2008, pp. 1-10.
4. W e i n h a r d t, C., A. A n a n d a s i v a m, B. B l a u, J. S t o s s e r. Business Models in the Service World. – IT Professional, Vol. **11**, March 2009, No 2, pp. 28-33.
5. D e n g, Y., M. R. H e a d, A. K o c h u t, J. M u n s o n, A. S a i l e r, H. S h a i k h. Introducing Semantics to Cloud Services Catalogs. – In: IEEE International Conference on Services Computing (SCC'2011), Washington, DC, 2011, pp. 24-31.
6. T a k a h a s h i, T., Y. K a d o b a y a s h i, H. F u j i w a r a. Ontological Approach Toward Cybersecurity in Cloud Computing. – In: Proc. of 3rd International Conference on Security of Information and Networks (SIN'10), Rostov-on-Don, Russian Federation, 2010, p. 100.
7. M a r t i n e z, A. C., G. I. E c h e v e r r i, A. G. C. S a n z. Malware Detection Based on Cloud Computing Integrating Intrusion Ontology Representation. – In: IEEE Latin-American Conference on Communications (LATINCOM'10), Bogota, 2010, pp. 1-6.
8. B e r n s t e i n, D., D. V i j. Intercloud Directory and Exchange Protocol Detail Using XMPP and RDF. – In: 6th World Congress on Services (SERVICES-1 2010), Miami, Florida, 2010, pp. 431-438.
9. H a n, T., K. M. S i m. An Ontology-Enhanced Cloud Service Discovery System. – In: Proc. of International MultiConference of Engineers and Computer Scientists 2010, Hong Kong, Vol. **I**, 2010, pp. 644-649.
10. K a n g, J., K. M. S i m. Ontology and Search Engine for Cloud Computing System. – In: International Conference on System Science and Engineering (ICSSE 2011), Macao, 2011, pp. 276-281.
11. D a s t j e r d i, A. V., S. G. H. T a b a t a b a e i, R. B u y y a. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. – In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'2010), Melbourne, Australia, 2010, pp. 104-112.
12. M a, H., K.-D. S c h e w e, Q. W a n g. An Abstract Model for Service Provision, Search and Composition. – In: IEEE Asia-Pacific Services Computing Conference (APSCC'09), Singapore, 2009, pp. 95-102.
13. P o v e d a-V i l l a l ó n, M., M. C. S u á r e z-F i g u e r o a, A. G ó m e z-P é r e z. A Double Classification of Common Pitfalls in Ontologies. – In: Proc. of Workshop on Ontology Quality (OntoQual'10), Co-Located with EKAW 2010, Lisbon, Portugal, 2010.
14. P o v e d a-V i l l a l ó n, M., M. C. S u á r e z-F i g u e r o a, A. G o m e z-P e r e z, A s u n c i o n. Validating Ontologies with OOPS! – In: Proc. of 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW'12), Galway City, Ireland, 2012, pp. 267-281.
15. B a u m e i s t e r, J., D. S e i p e l. Anomalies in Ontologies with Rules. – Web Semantics: Science, Services and Agents on the World Wide Web, Vol. **8**, March 2010, No 1, pp. 55-68.
16. V r a n d e č i ć, D. Ontology Evaluation. PhD Thesis, Karlsruher Instituts Fuer Technologie (KIT), Karlsruhe, 2010.
17. G ó m e z-P é r e z, A. Ontology Evaluation. – In: Handbook on Ontologies, S. Staab and R. Studer, Eds. Berlin, Heidelberg, Springer, 2004, pp. 251-273.
18. B r a n k, J., M. G r o b e l n i k, D. M l a d e n i ć. A Survey of Ontology Evaluation Techniques. – In: Proc. of Conference on Data Mining and Data Warehouses SiKDD 2005, Ljubljana, Slovenia, 2005.
19. A m i r h o s s e i n i, M., J. S a l i m. OntoAbsolute as a Ontology Evaluation Methodology in Analysis of the Structural Domains in Upper, Middle and Lower Level Ontologies. – In: International Conference on Semantic Technology and Information Retrieval (STAIR'11), Putrajaya, 2011, pp. 26-33.
20. N o y, N. F., D. L. M c G u i n n e s s. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford University, 2001.
21. SOFTEAM, SINTEF, Tecnalia. REMICS Deliverable D4.1 PIM4Cloud. REMICS Consortium, Project Deliverable, March 2012.
22. D o m i n g u e, J., M. Z a r e m b a, B. N o r t o n, M. K e r r i g a n, A. M o c a n, A. C a r e n i n i, E. C i m p i a n, M. H a i n e s, J. S c i c i l u n a, M. Z a r e m b a. Reference Ontology for Semantic Service Oriented Architectures. OASIS, Public Review Draft 01, November 2008.

23. M a r t i n, D., M. B u r s t e i n, J. H o b b s, O. L a s s i l a, D. M c D e r m o t t, S. M c I l r a i t h, S. N a r a y a n a n, M. P a o l u c c i, B. P a r s i a, T. P a y n e, E. S i r i n, N. S r i n i v a s a n, K. S y c a r a. OWL-S: Semantic Markup for Web Services. W3C Member Submission, November 2004.
24. R o m a n, D., H. L a u s e n, U. K e l l e r, J. de B r u i j n, C. B u s s l e r, J. D o m i n g u e, D. F e n s e l, M. H e p p, M. K i f e r, B. K ö n i g-R i e s, J. K o p e c k y, R. L a r a, E. O r e n, A. P o l l e r e s , J. S c i c l u n a, M. S t o l l b e r g. D2v1.4. Web Service Modeling Ontology (WSMO). WSMO Working Draft, February 2007.
25. P a h l, C., L. Z h a n g, F. F o w l e y. A Look at Cloud Architecture Interoperability through Standards. – In: 4th International Conference on Cloud Computing, Grids, and Virtualization, Valenica, Spain, 2013, pp. 7-12.
26. OASIS. Topology and Orchestration Specification for Cloud Applications Version 1.0. OASIS, OASIS Committee Specification Committee Specification 01, March 2013, Accessed 9 November 2016.
http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf
27. SNIA. Cloud Data Management Interface (CDMI^{TM}) Version 1.0.2. SNIA, SNIA Technical Position, Jun. 2012, Accessed 9 November 2016.
http://snia.org/sites/default/files/CDMI%20v1.0.2.pdf
28. Salesforce. SOAP API Developer’s Guide Version 28.0. 21-Jun-2013, Accessed 9 November 2016.
http://www.salesforce.com/us/developer/docs/api/
29. Salesforce. Metadata API Developer’s Guide. Salesforce, 05-Oct-2013., Accessed 9 November 2016.
https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_intro.htm
30. Google. Entities, Properties, and Keys. 16-Aug-2013, Accessed 9 November 2016.
https://cloud.google.com/appengine/docs/java/datastore/entities
31. Google. Google App Engine Java API. Google, Accessed 9 November 2016.
https://cloud.google.com/appengine/docs/java/javadoc/
32. Microsoft. Windows Azure Storage Services REST API Reference. Microsoft, 31-Aug-2011, Accessed 9 November 2016.
http://msdn.microsoft.com/en-us/library/windowsazure/dd179355.aspx
33. N a u d e t, Y., T. L a t o u r, W. G u e d r i a, D. C h e n. Towards a Systemic Formalisation of Interoperability. – Computers in Industry, Vol. **61**, February 2010, No 2, pp. 176-185.
34. P a r k, J., S. R a m. Information Systems Interoperability. – ACM Transactions on Information Systems, Vol. **22**, October 2004, No 4, pp. 595-632.
35. S h e t h, A. P., V. K a s h y a p. So Far (Schematically) yet So Near (Semantically). – In: Proc. of IFIP WG 2.6. Database Semantics Conference on Interoperable Database Systems, 1993, pp. 283-312.
36. P a r e n t, C., S. S p a c c a p i e t r a. Database Integration: The Key to Data Interoperability. – In: Advances in Object-Oriented Data Modeling. MIT Press, 2000.
37. Z h u, F., M. T u r n e r, I. K o t s i o p o u l o s, K. B e n n e t t, M. R u s s e l, D. B u d g e n, P. B r e r e t o n, J. K e a n e, P. L a y z e l l, M. R i g b y. Dynamic Data Integration Using Web Services. – In: Proc. of IEEE International Conference on Web Services (ICWS’04), San Diego, USA, 2004, pp. 262-272.
38. H a s l h o f e r, B., W. K l a s. A Survey of Techniques for Achieving Metadata Interoperability. – ACM Computing Surveys, Vol. **42**, February 2010, No 2, pp. 1-37.
39. P o n n e k a n t i, S. R., A. F o x. Interoperability among Independently Evolving Web Services. – In: Proc. of 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware’04), Toronto, Canada, 2004, pp. 331-351.
40. N a g a r a j a n, M., K. V e r m a, A. P. S h e t h, J. A. M i l l e r. Ontology Driven Data Mediation in Web Services. – International Journal of Web Services Research, Vol. **4**, 34 2007, No 4, pp. 104-126.

41. Berre, A.-J., B. Elvesæter, N. Figay, C. Guglielmina, S. G. Johnsen, D. Karlsen, T. Knothe, S. Lippe. The ATHENA Interoperability Framework. – In: Enterprise Interoperability II. R. J. Gonçalves, J. P. Müller, K. Mertins, and M. Zelm, Eds. London, Springer, pp. 569-580.
42. Loutas, N., E. Kamateri, K. Tarabanis, F. D'Andria. D 1.2 Cloud4SOA Cloud Semantic Interoperability Framework. 2 Juny 2011, Accessed 9 November 2016. **http://www.cloud4soa.com/sites/default/files/D1.2_Cloud4SOA%20Cloud%20Semantic%20Interoperability%20Framework.pdf**
43. Neuhaus, F., A. Vizedom, K. Baclawski, M. Bennett, M. Dean, M. Denny, M. Grüninger, A. Hashemi, T. Longstreth, L. Obrst, S. Ray, R. Sriram, T. Schneider, M. Vegetti, M. West, P. Yim. Towards Ontology Evaluation across the Life Cycle. Applied Ontology, 2013, No 3, pp. 179-194.
44. ProtegeWiki. DL Query Tab. Protege Wiki. 18-Mar-2013, Accessed 9 November 2016. **http://protegewiki.stanford.edu/wiki/DLQueryTab**
45. Carlini, E., M. Coppola, P. Dazzi, L. Ricci, G. Righetti. Cloud Federations in Contrail. – In: Euro-Par 2011, Parallel Processing Workshops. Vol. **7155**, M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Traff, G. Vallée, and J. Weidendorfer, Eds. Berlin, Heidelberg, Springer, 2012, pp. 159-168.