Application Programming Interfaces (APIs) Based Interoperability of Cloud Computing

Andročec, Darko

University of Zagreb

Faculty of Organization and Informatics

Darko Andročec

APPLICATION PROGRAMMING INTERFACES (APIs) BASED INTEROPERABILITY OF CLOUD COMPUTING

DOCTORAL THESIS

Varaždin, 2015.

DOCTORAL DISSERTATION INFO

I. AUTHOR

First and last name	Darko Andročec
Date and place of birth	October 11th, 1981, Čakovec
Name of the institution and date of obtaining graduate degree	Faculty of Organization and Informatics, November 19th, 2004.
Current employment	Faculty of Organization and Informatics

II. DOCTORAL DISSERTATION

Title	Application Programming Interfaces (APIs) Based Interoperability of Cloud Computing
Number of pages, figures, tables, annexes, bibliographic references	174 pages, 10 figures, 27 tables, 226 bibliographic references
Scientific area and field	Social Sciences, Information and Communication Sciences
Supervisors	Prof Neven Vrček
Name of the institution where the doctoral dissertation is publically defended	Faculty of Organization and Informatics, University of Zagreb

III. EVALUATION AND DEFENCE

Date of the meeting of Faculty Council when the topic is accepted	September 4th, 2012.
Date of submission	December 19th, 2014.
Date of the meeting of Faculty Council when the positive evaluation is accepted	March 3rd, 2015
Committee appointed for dissertation evaluation	Prof.dr.sc. Mirko Maleković, Prof.dr.sc. Neven Vrček, Dr.sc. Peep Küngas, Senior Researcher
Date of dissertation public defense	April 20th, 2015
Committee appointed for dissertation public defense	Prof.dr.sc. Alen Lovrenčić, Prof.dr.sc. Neven Vrček, Dr.sc. Peep Küngas, Senior Researcher
Date of promotion	

University of Zagreb

Faculty of Organization and Informatics

DARKO ANDROČEC

APPLICATION PROGRAMMING INTERFACES (APIs) BASED INTEROPERABILITY OF CLOUD COMPUTING

DOCTORAL THESIS

Research supervisor:

Prof. Neven Vrček

Varaždin, 2015.

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Darko Andročec

INTEROPERABILNOST USLUŽNOG RAČUNARSTVA POMOĆU APLIKACIJSKIH PROGRAMSKIH SUČELJA

DOKTORSKI RAD

Varaždin, 2015.

To my beloved wife Maja

and our three little angels Marta, Marija, and Gabrijel

Acknowledgements

First, I would like to thank my research supervisor Neven Vrček for his continual support, given freedom to choose research domain of my interest, and for his valuable comments. Next, I would like to thank Peep Küngas (University of Tartu, Estonia) for investing his time to evaluate my ontologies and my whole research approach, and for his detailed comments on the previous versions of the text. These comments have greatly improved the quality of my dissertation. Many thanks to Aleš Červinec (XLAB Research, Slovenia), Vlado Stankovski (University of Ljubljana, Slovenia), and Miha Stopar (XLAB Research, Slovenia) for their contribution in the evaluation of my two ontologies presented in this dissertation. I would also like to thank Irena Mandić for proofreading the text of the dissertation. I thank my friends and co-workers from Faculty of Organization and Informatics for their encouragement.

Finally, I wish to express my gratitude to my beloved family who have always supported me through the difficult times. Marta, Marija, Gabrijel and Maja have been my constant source of love, support, and strength. I thank all four of you for your patience during my work on this dissertation.

Abstract

Cloud computing paradigm is accepted by an increasing number of organizations due to significant financial savings. On the other hand, there are some issues that hinder cloud adoption. One of the most important problems is the vendor lock-in and lack of interoperability as its outcome. The ability to move data and application from one cloud offer to another and to use resources of multiple clouds is very important for cloud consumers.

The focus of this dissertation is on the interoperability of commercial providers of platform as a service. This cloud model was chosen due to many incompatibilities among vendors and lack of the existing solutions. The main aim of the dissertation is to identify and address interoperability issues of platform as a service. Automated data migration between different providers of platform as a service is also an objective of this study.

The dissertation has the following main contributions: first, the detailed ontology of resources and remote API operations of providers of platform as a service was developed. This ontology was used to semantically annotate web services that connect to providers' remote APIs and define mappings between PaaS providers. A tool that uses defined semantic web services and AI planning technique to detect and try to resolve found interoperability problems was developed. The automated migration of data between providers of platform as a service is presented. Finally, a methodology for the detection of platform interoperability problems was proposed and evaluated in use cases.

Keywords

Cloud interoperability, cloud data portability, platform as a service, AI planning, semantic web services, ontology, cloud APIs

Sažetak

Zbog mogućnosti financijskih ušteda, sve veći broj poslovnih organizacija razmatra korištenje ili već koristi uslužno računarstvo. Međutim, postoje i problemi koji otežavaju primjenu ove nove paradigme. Jedan od najznačajnih problema je zaključavanje korisnika od strane pružatelja usluge i nedostatak interoperabilnosti. Za korisnike je jako važna mogućnost migracije podataka i aplikacija s jednog oblaka na drugi, te korištenje resursa od više pružatelja usluga.

Fokus ove disertacije je interoperabilnost komercijalnih pružatelja platforme kao usluge. Ovaj model uslužnog računarstva je odabran zbog nekompatibilnosti različitih pružatelja usluge i nepostojanja postojećih rješenja. Glavni cilj disertacije je identifikacija i rješavanje problema interoperabilnosti platforme kao usluge. Automatizirana migracija podataka između različitih pružatelja platforme kao usluge je također jedan od ciljeva ovog istraživanja.

Znanstveni doprinos ove disertacije je sljedeći: Najprije je razvijena detaljna ontologija resursa i operacija iz aplikacijskih programskih sučelja pružatelja platforme kao usluge. Spomenuta ontologija se koristi za semantičko označavanje web servisa koji pozivaju udaljene operacije aplikacijskih programskih sučelja pružatelja usluga, a sama ontologija definira i mapiranja između pružatelja platforme kao usluge. Također je razvijen alat koji otkriva i pokušava riješiti probleme interoperabilnosti korištenjem semantičkih web servisa i tehnika AI planiranja. Prikazana je i arhitektura za automatiziranu migraciju podataka između različitih pružatelja platforme kao usluge. Na kraju je predložena metodologija za otkrivanje problema interoperabilnosti koja je evaluirana pomoću slučajeva korištenja.

Ključne riječi

Interoperabilnost oblaka, prenosivost podataka na oblacima, platforma kao usluga, AI planiranje, semantički web servisi, ontologija, aplikacijska programska sučelja oblaka

Extended Abstract

The numerous heterogeneities among different vendors make cloud interoperability an interesting and complex research and practical problem. Cloud computing is nowadays becoming a popular paradigm for the provision of computing infrastructure, but 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. This dissertation has tackled vendor lock-in problem in platform as a services offers by using Semantic Web services and AI planning to detect and try to solve the identified interoperability problems.

The basic steps in this research include: design and implementation of use cases, development of the ontology of platform as a service, definition and development of semantic web services, identification of interoperability problems among different commercial providers of platform as a service, and design of the methodology for the detection and resolution to interoperability problems. First, two use cases were defined. These use cases are examined to determine technical and semantic interoperability problems among APIs of different providers of platform as a service and to test methodologies and tools used to detect and resolve interoperability problems. In the first use case, data will be migrated between different providers of platform as a service. Successful execution of more complex interoperability scenarios cannot be imagined without being able to move data from one PaaS vendor to another. The majority of vendors' API operations deal with data manipulation and management, so the first use case is also important to learn more about the mentioned APIs in practical problems. The result of the first use case is an architecture for data migration among PaaS providers that uses data ontology (OWL is intermediate data format) and data type mappings stored as individuals in PaaS ontology. The validation of the first use case and the data migration architecture was done by migrating a more complex set of data (concretely, data of open-source content management system) and manually checking all of the migrated data elements. In the second use case, current user information from one PaaS offer are added

to the application hosted on another PaaS offer. The main aim is to investigate interoperability problems on service layer when using APIs from different providers. The ontology driven data mediation are used and tested in second use case. Web operations and their inputs/outputs were semantically annotated, and SAWSDL and XSLT were used to define service type mappings.

Next, the PaaS ontology for resources and operations and the ontology of interoperability problems were developed. For this purpose, the Ontology Development 101 methodology was selected, 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. The representation of resources and operations in APIs of platform as a service is determined as the domain of the ontology of PaaS resources, remote operations, and data types. It provides information about the most important PaaS resources, it classifies providers' remote API operations, and supports mappings of data types among the heterogeneous APIs and cloud storages. The domain of the second ontology is the representation of the technical and semantic interoperability problems of commercial platform as a service offers. This ontology is used in the methodology for detecting interoperability problems among providers of platform as a service as a comprehensive list of possible interoperability issues. Developed ontologies were evaluated. There is no consensus on the best ontology evaluation approach, but evaluating the ontologies systematically certainly raise its quality. Due to a lack of gold standards and corpus of data, the evaluation by humans and application-based evaluation was chosen.

The ontology of PaaS resources, remote operations, and data types developed in the previous step is used to create semantic web services that represent remote functions (APIs) of platform as a service offers. 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. SAWSDL lightweight annotation was used to define semantic web services, and XSLT was used to define needed input/output transformations. Web services, their inputs and outputs were semantically annotated, and service data type mappings and needed transformations were defined.

For AI planning process, a JSHOP2 planner was used. The inputs of JSHOP2 are a planning domain and a planning problem. Problem description file is composed of logical atoms

showing the initial state and a task list. 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 (e.g., some interoperability action such as ones described in two use cases) that need to be completed are generated and saved in JSHOP2 problem description file. The initial state (a set of logical atoms) is also created programmatically. Based on the chosen method representing the chosen interoperability action (task list to be executed), SAWSDL and/or PaaS ontology files are parsed to generate logical atoms. The domain description file was defined manually. 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. During execution of web service compositions, the needed transformations between inputs and outputs should be performed. If there is no suitable plan returned by 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 algorithm for this purpose was developed and presented in this work.

The final contribution of this dissertation is the creation of a methodology for determining the relevant interoperability issues among two or more PaaS providers. Currently, there is still no methodology that aims at identification and resolution of interoperability problems; neither among APIs of commercial platforms as a service nor among cloud offers in general. The proposed methodology uses iterative approach, because PaaS offers and their APIs evolve and change very often. The user's interoperability requirements also change during time and new interoperability problems could arise. The proposed methodology has five main steps: requirements identification, interoperability analysis, solution design, solution implementation, and evaluation. In the first step, the most important interoperability needs of users should be listed. Interoperability analysis deals with identifying levels of interoperability problems and reasoning on possible interoperability problems between different commercial providers of platform as a service. Solution design includes activities such as the development of the ontology of resources, remote operations and data types, definition of the semantic web service, needed mappings and transformations, and defining AI planning domain. Solution implementation deals with approach implementation and execution

of the defined use cases. Evaluation step evaluates the successful execution of use cases and correct identification of possible interoperability problems.

Keywords

Cloud interoperability, cloud data portability, platform as a service, AI planning, semantic web services, ontology, cloud APIs

Prošireni sažetak

Mnogobrojne razlike između pružatelja usluga rezultiraju time da je interoperabilnost uslužnog računarstva složen istraživački i praktični problem. Uslužno računarstvo je danas popularna paradigma za pružanje računalne infrastrukture, ali su ujedno poznati i određeni nedostaci ovog modela, od kojih je jedan od najznačajnijih ovisnost o pružatelju usluge. Ovaj problem manifestira se vremenski zahtjevnom i skupom migracijom podataka i aplikacija na alternativno rješenje u oblaku, nemogućnošću ili ograničenom mogućnošću korištenja računalnih resursa, aplikacija ili podataka izvan odabranog rješenja, te ovisnošću o korištenju samo onih programskih jezika koje odabrani pružatelj usluge podržava. Ova disertacija rješava spomenuti problem u modelu platforme kao servisa korištenjem semantičkih web servisa i AI planiranja kako bi se otkrili i pokušali riješiti problemi interoperabilnosti.

Osnovni koraci ovog istraživanja su: dizajn i implementacija slučajeva korištenja, razvoj ontologije platforme kao usluge, definicija i razvoj semantičkih web servisa, identifikacija problema interoperabilnosti između različitih komercijalnih pružatelja platforme kao usluge, te dizajn metodike za otkrivanje i rješavanje problema interoperabilnosti. Najprije su definirana dva slučaja korištenja. Njihov je cilj odrediti tehničke i semantičke probleme interoperabilnosti između API-a različitih pružatelja platforme kao usluge, te testirati razvijenu metodiku i korištene alate. U prvom slučaju korištenja, podaci se migriraju između različitih pružatelja usluga. Uspješno izvođenje kompleksnijih scenarija interoperabilnosti ne može se zamisliti bez postojanja mogućnosti migriranja podataka od jednog do drugog pružatelja platforme kao usluge. Osim toga, većina udaljenih operacija pružatelja usluga na neki način manipulira ili upravlja podacima, pa je prvi slučaj korištenja koristan i za detaljno izučavanje tih API-a. Rezultat prvog slučaja korištenja je arhitektura za migraciju podataka između različitih pružatelja platforme kao usluge koja koristi podatkovnu ontologiju (OWL je posrednički podatkovni format) i mapiranja tipova podataka koji su implementirani kao instance u PaaS ontologiji. Validacija prvog slučaja korištenja i arhitekture za migraciju podataka napravljena je migriranjem kompleksnijeg skupa podataka (konkretno, podataka jednog besplatnog sustava za upravljanje sadržajem) i ručnom provjerom svih migriranih podatkovnih elemenata. Drugi slučaj korištenja opisuje dodavanje korisničkih informacija iz jedne PaaS usluge na aplikaciju koja koristi resurse drugog pružatelja usluge. Glavni cilj je

istražiti probleme interoperabilnosti na razini servisa kada se koriste API-i više različitih pružatelja usluga. U drugom slučaju korištenja prikazuje se i testira posredovanje podacima korištenjem ontologija. Web operacije i njihovi ulazi i izlazi su semantički anotirani, a SAWSDL i XSLT definiraju mapiranja između različitih tipova podataka u servisima.

Nakon toga razvijene su dvije ontologije: ontologija resursa i operacija platforme kao usluge i ontologija problema interoperabilnosti. Za tu svrhu, odabrana je metodika Ontology Development 101, jer je jedna od najjednostavnijih i jer je fokusirana na sam rezultat, tj. na brzo stvaranje početne verzije ontologije koja se s vremenom razvija i mijenja u skladu sa zahtjevima. Domena ove ontologije je prikaz resursa i operacija API-a različitih ponuda platforme kao usluge. Ontologija popisuje najvažnije resurse platforme kao usluge, klasificira udaljene API operacije različitih pružatelja usluga, te podržava mapiranja tipova podataka između različitih API-a i spremišta podataka na oblacima. Domena druge ontologije je prikaz tehničkih i semantičkih problema interoperabilnosti komercijalnih rješenja platforme kao usluge. Ova se ontologija koristi u predloženoj metodici za otkrivanje i rješavanje problema interoperabilnosti i služi kao sveobuhvatna lista mogućih problema interoperabilnosti. Razvijene ontologije su evaluirane. U literaturi nema konsenzusa oko najboljeg pristupa za evaluaciju ontologije, ali sistematska evaluacija ontologija sigurno povećava njihovu kvalitetu. Zbog nedostatka prihvaćenih standarda i podataka, izabrana je evaluacija korištenjem eksperata i evaluacija bazirana na primjeni ontologije u aplikacijama.

Ontologija resursa, udaljenih operacija i tipova podataka koja je prethodno razvijena, koristi se za kreiranje semantičkih web servisa koji prikazuju udaljene funkcije API-a različitih platformi kao usluga. Razvijeni su web servisi koji učahuruju udaljene operacije API-a od tri komercijalna pružatelja usluga (Google, Microsoft i Salesforce) s ciljem pristupa tim servisima na jedinstven način. Jednostavne anotacije SAWSDL-a su korištene za definiranje semantičkih web servisa, a XSLT se koristi za definiranje potrebnih transformacija ulaza i izlaza. Web servisi, njihovi ulazi i izlazi, kao i mapiranja tipova podataka servisa i eventualnih transformacija su također ovdje definirani.

Za AI planiranje koristio se planer JSHOP2. Ulazi u taj alat su domena planiranja i problem planiranja. Datoteka opisa problema sastoji se od logičkih atoma koji prikazuju početno stanje i listu zadataka. Ovi elementi se kreiraju programski, prilikom izvršavanja prototipa, kada

korisnik preko klijentske web aplikacije pokrene željenu akciju interoperabilnosti. Ovisno o odabiru korisnika, zadaci koji se moraju izvršiti (na primjer, određene akcije interoperabilnosti poput onih opisanih u slučajevima korištenja) se generiraju i spremaju u JSHOP2 datoteku za opis problema. Početno stanje (skup logičkih atoma) također se kreira programski. Ovisno o odabranoj metodi, parsiraju se SAWSDL datoteke i ontologija kako bi se generiralo početno stanje. Datoteka s opisom domene se kreira ručno. Ukoliko planer JSHOP2 pronađe plan, on se ispisuje na klijentskoj strani web aplikacije, te se korisniku prikazuje opcija izvršenja plana, tj. u krajnjem slučaju, pozivanje relevantnih web servisa. U tom slučaju sve je prošlo u redu, to jest u toj fazi nisu nađeni problemi interoperabilnosti. Prilikom izvršenja kompozicije web servisa, trebaju biti napravljene potrebne definirane transformacije između ulaza i izlaza. Ako planer ne vrati pogodan plan, web aplikacija na strani klijenta ispisuje poruku o greški. U tom su slučaju nađeni određeni problemi interoperabilnosti, te stoga treba utvrditi razlog greške. Za tu svrhu razvijen je algoritam koji je opisan u samoj disertaciji.

Zadnji znanstveni doprinos ove disertacije je kreiranje metodike za određivanje relevantnih problema interoperabilnosti između dva ili više pružatelja platforme kao usluge. Trenutno još ne postoji metodika za identifikaciju i rješavanje problema interoperabilnosti bilo između API-a različitih platformi kao usluga, bilo između uslužnog računarstva općenito. Predložena metodika koristi iterativni pristup, jer se rješenja platforme kao usluge i njihovi API-i često mijenjaju. Tokom vremena mogu se promijeniti i zahtjevi korisnika vezani uz interoperabilnost. Predložena metodika sastoji se od pet glavnih koraka: identifikacije zahtjeva, analize interoperabilnosti, dizajna rješenja, implementacije rješenja i evaluacije. U prvom se koraku trebaju izlistati najvažnije korisničke potrebe za interoperabilnošću. Analiza interoperabilnosti identificira razine problema interoperabilnosti i razmatra koji su sve problemi interoperabilnosti mogući između različitih pružatelja platforme kao usluge. Dizajn rješenja uključuje aktivnosti poput razvoja ontologije resursa, udaljenih operacija i tipova podataka, definiciju semantičkih web servisa, potrebnih mapiranja i transformacija, kao i definiranja AI domene. Implementacija rješenja sastoji se od same implementacije i izvršavanja definiranih slučajeva korištenja. Korak evaluacije provjerava valjanost i uspješnost izvršavanja slučajeva korištenja i ispravnu identifikaciju problema interoperabilnosti.

Ključne riječi

Interoperabilnost oblaka, prenosivost podataka na oblacima, platforma kao usluga, AI planiranje, semantički web servisi, ontologija, aplikacijska programska sučelja oblaka

TABLE OF CONTENTS

LIST OF TABLES **IV**

LIST OF FIGURES **V**

ABBREVIATIONS **VI**

1. INTRODUCTION **1**

1.1 MOTIVATION 1
1.2 RESEARCH QUESTIONS AND HYPOTHESES 2
1.3 RESEARCH OBJECTIVES 2
1.4 CONTRIBUTIONS 3
1.5 RESEARCH METHODOLOGY 3
1.6 DISSERTATION OUTLINE 5

2. CORE CONCEPTS AND BACKGROUND **6**

2.1 CLOUD COMPUTING 6
2.2 PLATFORM AS A SERVICE 7
2.3 INTEROPERABILITY 9
2.4 CLOUD COMPUTING STANDARDS 10
2.5 SEMANTIC WEB AND ONTOLOGIES 12
2.5.1 Semantic Web and its standards *12*
2.5.2 OWL *13*
2.5.3 Ontology *13*
2.5.4 Methodologies for ontology development *14*
2.6 SEMANTIC SERVICE ORIENTED ARCHITECTURE 15
2.6.1 SOA and web services *15*
2.6.2 Semantic web services *16*
2.7 AI PLANNING METHODS 18
2.7.1 AI planning and automated service composition *18*
2.7.2 Hierarchical Task Network (HTN) planning *19*

3. RELATED WORK **20**

3.1 REVIEW OF INTEROPERABILITY 20
3.1.1 Interoperability problems, issues and conflicts *20*
3.1.2 Cloud computing interoperability *24*
3.1.3 Research projects on the cloud interoperability *27*
3.1.4 Cloud computing interoperability use cases *30*
3.1.5 Interoperability methodologies *34*
3.1.6 Data mediation between the services *35*
3.2 ONTOLOGIES 36
3.2.1 Cloud ontologies *36*

3.2.2 Ontology anomalies and ontology evaluation *37*
3.3 AI PLANNING 39
3.3.1 AI planning methods and cloud computing *39*
3.3.2 Gaps in planning domains *39*
3.4 SUMMARY OF THE EXISTING WORK 40
3.4.1 Systematic mapping study on cloud interoperability *40*
3.4.1.1 Research scope of the systematic mapping 41
3.4.1.2 Conduct search 41
3.4.1.3 Screening of papers 41
3.4.1.4 Classification scheme 44
3.4.1.5 Data extraction and mapping 45
3.4.2 Identified gaps *49*
4. USE CASES **50**
4.1 PRELIMINARIES 50
4.1.1 Chosen PaaS offers *50*
4.1.2 PaaS data and application models *51*
4.1.3 Working with external PaaS data *53*
4.1.4 Web services support in PaaS offers *54*
4.2 USE CASE 1: MIGRATION OF DATA BETWEEN PAAS PROVIDERS 55
4.2.1 Requirements and use case description *55*
4.2.2 Export data structures and data from PaaS providers *57*
4.2.3 Transformation of data structures and data to ontology *58*
4.2.4 Data type mappings *60*
4.2.5 Architecture for data migration among PaaS providers *63*
4.2.6 Validation and assessment *64*
4.2.7 Lessons learned from use case 1 *68*
4.3 USE CASE 2: ADD EXISTING USER TO ANOTHER PAAS 69
4.3.1 Requirements and use case 2 description *69*
4.3.2 Ontology driven data mediation *71*
4.3.3 Validation *72*
4.3.4 Lessons learned from use case 2 *73*
5. DEVELOPMENT AND EVALUATION OF ONTOLOGIES **73**
5.1 SELECTED ONTOLOGY DEVELOPMENT METHODOLOGY, TOOL AND LANGUAGE 73
5.2 ONTOLOGY OF PAAS RESOURCES, REMOTE OPERATIONS, AND DATA TYPES 75
5.2.1 Domain and scope of the ontology *75*
5.2.2 Reusing the existing ontologies *75*
5.2.3 Important terms for the ontology *75*
5.2.4 Classes and their hierarchy *77*
5.2.5 Properties of classes *83*

5.2.6 Creating instances *85*
5.3 ONTOLOGY OF PLATFORM AS A SERVICE INTEROPERABILITY PROBLEMS 85
5.3.1 Domain and scope *85*
5.3.2 Reused concepts from other ontologies *86*
5.3.3 Enumerate important terms *86*
5.3.4 Definition of the class hierarchy *88*
5.3.5 Define the properties of classes *92*
5.3.6 Creation of the facets and instances *93*
5.4 EVALUATION OF THE ONTOLOGIES 93
5.4.1 Evaluation by tools *94*
5.4.2 Evaluation by humans *94*
5.4.3 Application-based evaluation *96*
6. PROPOSED SOLUTION AND METHODOLOGY **98**
6.1 SEMANTIC PAAS WEB SERVICES 98
6.2 IMPLEMENTATION OF AI PLANNING 102
6.2.1 JSHOP2 planner *102*
6.2.2 JSHOP2 problem description *103*
6.2.3 JSHOP2 domain description *105*
6.3 PLAN EXECUTION AND SERVICE COMPOSITION 107
6.4 FINDING INTEROPERABILITY PROBLEMS 108
6.5 METHODOLOGY FOR DETECTION OF INTEROPERABILITY PROBLEMS 111
6.5.1 Methodology justification *111*
6.5.2 Steps of the methodology *112*
6.5.3 Applying the methodology *115*
7. CONCLUSIONS **116**
7.1 SUMMARY OF CONTRIBUTIONS 116
7.1.1 Creation of detailed ontologies *116*
7.1.2 Development of a methodology *116*
7.1.3 Solving interoperability problems *117*
7.2 ANSWERS TO RESEARCH QUESTIONS 117
7.3 HYPOTHESES REVISITED 118
7.4 LIMITATIONS OF RESEARCH 119
7.5 OPEN ISSUES AND FUTURE WORK 119
BIBLIOGRAPHY **121**
CURRICULUM VITAE **149**

LIST OF TABLES

Table 1 Summary of the existing use cases and scenarios 32
Table 2 Distribution of the found publications 41
Table 3 Full list of identified relevant papers 42
Table 4 Data form for sample paper 44
Table 5 Classification dimensions and their categories 45
Table 6 Obligatory standard fields of custom objects (194) 52
Table 7 Description of data migration use case 56
Table 8 Mappings from PaaS storages' to OWL data types 61
Table 9 Mappings from OWL to PaaS storages' data types 62
Table 10 Manual evaluation of Vosao's data migration to Salesforce 66
Table 11 Description of use case 2 69
Table 12 List of identified terms for PaaS ontology 76
Table 13 List of all classes of the PaaS ontology 78
Table 14 Object properties defined in PaaS ontology 83
Table 15 Data properties of PaaS ontology 84
Table 16 Reused concepts from Naudet et al. (17) 86
Table 17 List of important terms for PaaS interoperability ontology 87
Table 18 List of classes in the PaaS interoperability ontology 89
Table 19 Object properties of the interoperability problems ontology 93
Table 20 Summary of ontology evaluation by experts 95
Table 21 Example of operation's annotations and transformations 99
Table 22 Example of input/output transformations 100
Table 23 Possible logical atoms in the initial state 104
Table 24 Methods defined in JSHOP2 domain file 105
Table 25 Operators and their preconditions and postconditions 106
Table 26 Testing examples of finding interoperability problems 110
Table 27 Steps and activities of the proposed methodology 114

LIST OF FIGURES

Figure 1 Paper distribution per cloud model 46
Figure 2 Distribution per paper types 46
Figure 3 Applied solutions 47
Figure 4 Investigated interoperability problems 48
Figure 5 Visualization of a systemic map using a bubble chart 48
Figure 6 Architecture for data migration between PaaS providers 64
Figure 7 API operations executed in use case 2 69
Figure 8 Example of service input/output transformations 72
Figure 9 Top level classes of PaaS ontology 78
Figure 10 Top level classes of interoperability problems ontology 89

ABBREVIATIONS

AI	Artificial Intelligence
AIF	ATHENA Interoperability Framework
API	Application Programming Interface
CDMI	Cloud Data Management Interface
CIMI	Cloud Infrastructure Management Interface
CMS	Content Management System
CSV	Comma-separated values
DMTF	Distributed Management Task Force
EIF	European Interoperability Framework
GAE	Google App Engine
HTN	Hierarchical Task Network
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a service
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OCCI	Open Cloud Computing Interface
OVF	Open Virtualization Format
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
PaaS	Platform as a service
PDDL	Planning Domain Definition Language
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational State Transfer
SAWSDL	Semantic Annotations for WSDL and XML Schema
SaaS	Software as a service
SDK	Software Development Kit
SHOP	Simple Hierarchical Task Network

SLA	Service-Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TOSCA	Topology and Orchestration Specification for Cloud Applications
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

1. INTRODUCTION

1.1 Motivation

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. Currently, each cloud vendor offers its own tools, remote application programming interfaces (APIs), and some even create new programming languages and frameworks. If clouds are not interoperable, it is difficult or even impossible to achieve collaboration among computing resources of different cloud service providers, and possible migration to another provider is a complex and expensive task.

The numerous heterogeneities among different vendors make cloud interoperability an interesting and complex research and practical problem. Because of the different models of cloud computing (infrastructure as a service, platform as a service, software as a service, etc.) and the complexity of the technologies used, it is impossible to cover the interoperability of all cloud computing models in one study. This dissertation is focused on platform as a service (PaaS) model. Interoperability of platform as a service model is chosen because it is not well investigated in the current literature (for example, interoperability of infrastructure as a service model is dealt with more in the existing literature and cloud standards), there are no accepted standards, and its vendor lock-in problem is very significant due to heterogeneities of PaaS offers on many levels. This dissertation has tackled vendor lock-in problem in platform as a services offers by using Semantic Web services and AI planning to detect and try to solve the identified interoperability problems.

1.2 Research questions and hypotheses

The following research questions are identified:
(1) How to semantically describe resources and operations of commercial platform as a service APIs?
(2) What are key indicators of the existence of interoperability problems among the available remote functions of providers of platform as a service?
(3) What are the possible solutions to known interoperability problems?

The two hypotheses are:

H1 Developed ontology will determine the differences among remote application programming interfaces (APIs) of commercial platform as a service providers and improve understanding of platform as a service resources and operations.

H2 Based on the concepts identified in the ontology (resources, operations and interoperability problems), the methodology for determining semantic interoperability problems among the various commercial platforms as a service providers and their resolution using the available APIs will be developed.

1.3 Research objectives

The general goal of the dissertation is to contribute to the resolution of a problem of platform as a service interoperability. More particularly, this dissertation aims at:
• Identification of resources and operations from APIs of relevant commercial platform as a service providers and development of the ontology.
• Abstraction of platform as a service APIs in the form of Semantic Web services using the aforementioned ontology.
• Development of a methodology for the detection of semantic interoperability problems and conflicts among the APIs of two or more selected providers of platform as a service.

• Determination of whether found interoperability problems can be solved using the available vendors' APIs.

1.4 Contributions

The main contribution of the dissertation is a study of interoperability problems among different commercial providers of platform as a service and finding some solutions to achieve interoperability among them. More specific contributions include:

1. Creation of a detailed ontology of resources and operations from APIs of commercial providers of platform as a service and ontology of common interoperability problems between different PaaS' APIs
2. Development of a methodology for the detection of interoperability problems among various commercial platform as a service providers
3. Determining whether it is possible to solve interoperability problems found using the available API functions provided by commercial vendors of platform as a service.

During the dissertation work, it became evident that the majority of the functions of remote providers' APIs deal with the underlying storage and its metadata. So, the majority of interoperability problems that can be solved by using providers' API are data interoperability problems. The additional contribution of the dissertation is therefore:

4. Design of the architecture for automated data migration among different providers of platform as a service

1.5 Research methodology

The basic steps in this research include: design and implementation of use cases, development of the ontology of platform as a service, definition and development of semantic web services, identification of interoperability problems among different commercial providers of platform as a service, and design of the methodology for the detection and resolution to interoperability problems. 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 platform as a service and to test methodologies and tools used to detect and

resolve interoperability problems. Initial use cases will be gradually evolved into more complex ones while research progresses.

The second step of this research is the development of 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. Additionally, the ontology 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. Developed ontologies have been adequately evaluated.

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. In the next phase of this work, the technical and semantic interoperability problems of commercial platform as a service APIs are identified. The remote functions of commercial cloud providers are mostly in the form of SOAP or REST web services. In the context of service-oriented architecture, semantic interoperability means the ability to interact and collaborate among software services, and the subject of this dissertation is to determine interoperability problems among the available remote functions from APIs of relevant commercial platform as a service providers. The final contribution of this dissertation is the creation of a methodology for determining the relevant interoperability issues among two or more providers. It is used to determine the existing interoperability problems among selected commercial solutions of platform as a service 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 platform as a service providers. For this purpose, the AI planning methods (1) were used.

1.6 Dissertation outline

The remainder of this dissertation is composed of six additional chapters. Chapter 2 presents the main core concepts and background (cloud computing, platform as a service, interoperability, cloud computing standards, Semantic Web and ontologies, semantic service oriented architecture and AI planning methods). In Chapter 3, the most relevant existing work is listed. Chapter 4 concentrates on practical part of the dissertation, including use cases aimed at detecting interoperability problems and the evaluation of the proposed theoretical solution and methodology developed as part of this work. Chapter 5 is dedicated to the development of two ontologies: the ontology of platform as a service resources, remote operations, and data type mappings; and the ontology of platform as a service interoperability. Chapter 6 deals with finding and solving interoperability problems among different platform as a service offers and it elaborates on the methodology developed for the detection of interoperability problems. The conclusion of the dissertation is given in the last chapter, together with the summary of contributions, open issues, and ideas for future research.

2. CORE CONCEPTS AND BACKGROUND

2.1 Cloud computing

Cloud computing is a business and computing paradigm whose main benefits are flexibility, pay-per-use model and significant cost reduction. National Institute of Standards and Technology (NIST) provided the most accepted definition of cloud computing: *"Cloud computing is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources"* (2). Erl defines cloud computing as *"a specialized form of distributed computing that introduces utilization models for remotely provisioning scalable and measured resources"* (3). The Gartner Group defines cloud computing as *"a style of computing in which massively scalable IT-related capabilities are provided "as a service" using Internet technologies to multiple external customers"* (4).

Armbrust et al. (5) conclude that there are three new aspects in cloud computing from a hardware point of view: the illusion of infinite resources available on demand, the elimination of users' up-front commitment and the ability to pay for use when specific cloud resources are needed. The same authors presented the ten biggest obstacles and opportunities for cloud computing. Their opinion is that the most significant obstacles are: service availability, data interoperability problems, data confidentiality, bottlenecks caused by data transfer, variations in performance, legal liability, and new means of software licensing. Each obstacle has associated opportunities, e.g. data lock-in is the obstacle, but its associated opportunities are standardization of APIs and development and execution of compatible software on multiple clouds. Wang et al. (6) considered the functional aspects of cloud computing and its differences from other computing paradigms. According to Wang et al. (6), types of services are: hardware as a service, software as a service, data as a service, and infrastructure as a service. Cloud computing possesses customer-oriented interface and offers the guaranteed quality of service, scalability and flexibility. Enabling technologies behind cloud computing are (6): systems for distributed data storage, cloud programming models, virtualization, service-oriented architecture and web 2.0.

Most researchers distinguish three main types of cloud services. These main service models of cloud computing are (2): software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). In software as a service model, the consumer directly uses the provider's applications running on their cloud infrastructures from various client devices (2). In regard to management and control of cloud infrastructure, the consumer can typically modify only limited user-specific application configuration settings (2). The second model of cloud computing is platform as a service. Using this type of cloud service, the client can deploy their own or acquired applications supported by vendor's platform together with the supported programming languages, libraries and tools (2). The client can control deployed applications and configure application environment. In infrastructure as a service model, the consumer controls operating systems, storages, deployed applications and some selected networking components (2). Infrastructure as a service provides the capability to provision fundamental computing resources to run operating systems, web, email and application servers and applications.

Mell and Grance (2) distinguish four main deployment models of cloud computing: private cloud, community cloud, public cloud and hybrid cloud. Private cloud's resources are used exclusively by a single organization with multiple users (2). Community clouds are used by specific communities from organizations (2). The general public can use infrastructures provisioned by public cloud (2). The hybrid cloud's infrastructure consists of two or more distinct types of clouds (private, community, or public) bound by standardized or proprietary technology (2).

2.2 Platform as a service

If a cloud vendor supplies the software platform on which systems run, instead of providing a virtualized infrastructure, one talks about platform as a service (PaaS) model (7). The NIST definition of the platform as a service is elaborated in the previous chapter (2.1). Boniface et al. (8) define platform as a service as *"the provision of a development platform and environment providing services and storage, hosted in the cloud"* (8). Multiple applications use this single platform and its predefined services. The platform itself is built on some offers of infrastructure as a service. The promise of PaaS is that one only needs to code the application, and cloud vendor will handle everything else, from infrastructure and network to

their operations (9). In theory, PaaS consumers will get better security and business continuity accompanied by a much lower price (9). Walton (10) claims that PaaS solutions represent web-based development platforms with predefined and vendor-controlled infrastructure for application deployment. PaaS users build and deploy their applications with the providers' tools and application environments. Platform as a service vendors offer virtual platforms to their users to develop and run applications. Data is mostly stored within vendor's infrastructure. Erl (3) also agreed that platform as a service relies on the usage of ready-made environment to support the complete lifecycle of web applications. The cloud consumer has a lower level of control over the underlying infrastructure compared to infrastructure as a service model. At the same time, he can focus more on core aspects of his job (application development) and minimize time spent on configuration and system engineering.

Platform as a service model of cloud computing has the following benefits (10): increasing programmer productivity, companies can release products more quickly, and development costs are reduced. Apart from these benefits, Lawton (10) also listed some limitations, concerns, and drawbacks of platform as a service model: strong provider lock-in, security and privacy problems, it only delivers a subset of functions that are standard in classical development platform, heavy-weight management or governance services are not provided. Emison (9) listed the following possible trade-offs regarding platform as a service: limitation of control over many aspects of application development, low-level performance tuning is not available or is very limited, providers support only a limited set of software versions, a limited set of configuration options, dependence on vendor-created metrics of application performance and vendor lock-in.

Vendor lock-in problem is mentioned in both lists of PaaS drawbacks. Specific PaaS implementations are less portable than virtual machines (9), so there is a greater risk of occurrence of the mentioned problem in PaaS than in IaaS environment. Furthermore, some PaaS offers like Force.com, Rollbase and WorkXpress use proprietary computer languages and application environments (9).

Emison (9) distinguishes three main categories of platform as a service providers:

- Comprehensive PaaS vendors support more languages and/or environments to address many scenarios and support as many different applications as possible (9). Google

App Engine, Microsoft Azure, and Red Hat Openshift can be listed as representatives of the first category.

- Specific-stack PaaS targets customers that already use standard stacks for applications and enables simpler deployment of applications to PaaS environment without the need to dramatically change applications developed in the specific enterprise-based stack (9). The representative of the second category of PaaS providers is IBM.
- Proprietary PaaS providers offer robust platform with many useful features in exchange for using their proprietary languages, tools and configuration (9). An example of the third category is Salesforce's platform as a service offer.

2.3 Interoperability

Interoperability can be defined in several ways. One of the simplest definitions is credited to IEEE that defines this term as *"the ability of two or more systems or components to exchange information and to use the information that has been exchanged"* (11). Brownsword et al. (12) provided the following working definition of interoperability: *"The ability of a collection of communicating entities to (a) share specified information and (b) operate on that information according to an agreed operational semantics"* (12). Pokraev et al. (13) claim that *"interoperability implies that systems are able to interact (i.e., exchange messages), read and understand each other's messages and share the same expectations about the effect of the message exchange"* (13). From this definition, three main aspects of interoperability arise (13): syntactic interoperability (compatible formats), semantic interoperability (meaning of the information), and pragmatic interoperability (effect of the exchanged information). Vernadat similarly defines the term interoperability as "*the ability for a system to communicate with another system and to use the functionality of the other system"* (14).

Park and Ram (15) think that interoperability is the most critical issue for businesses that use data from different information systems. Two types of interoperability are distinguished in their work: semantic interoperability and syntactic interoperability. For them, semantic interoperability exists at the knowledge-level and it is used to bridge semantic conflicts due to differences in meanings, perspectives, and assumptions (15). On the other hand, they define syntactic interoperability as the interoperability at application level that aims at cooperation

among different software components with different implementation languages and development platforms (15).

Interoperability is a multidimensional concept that can be looked at from multiple perspectives. Therefore, frameworks for interoperability which specify a set of common elements such as vocabulary, concepts, principles, guidelines and recommendations were developed and can be identified in the literature. Some of the most important frameworks are (16): ATHENA Interoperability Framework (AIF), IDEAS Interoperability Framework, LISI Reference Model, Enterprise Interoperability Framework, and GridWise Interoperability Context-Setting Framework.

Apart from interoperability frameworks, some comprehensive interoperability models are presented in the current literature. For example, Naudet et al. (17) developed a general ontology of interoperability. This ontology describes the ontological metamodel system, problems and solutions and can be used to diagnose and resolve interoperability problems. The above mentioned authors conclude that there are only two alternative technical solutions to interoperability problems: bridging and homogenization. Bridging uses an intermediate system (often called an adapter) between systems having interoperability problems. The intermediate system relies on the translation protocol (for example, using mappings) to achieve interoperability between interacting systems (17). Homogenization implies the unified model and acts directly on models or their representations (17). It requires either syntactic or semantic transformations that used the defined unified model.

2.4 Cloud computing standards

For now, there are not any adopted cloud computing standards (18) among different commercial cloud providers. Each commercial service provider has its own specific APIs and different technological solutions, which is not conducive to their mutual interoperability. So, the initiatives for standardization in this area are very important. Pahl, Zhang and Fowley listed the most promising initiatives in their two papers - (19) and (20).

The scope of *The Open Cloud Computing Interface (OCCI)* (20) is high-level functionalities for life cycle management of virtual machines running on virtualization technologies. OCCI is RESTful API for remote management including deployment, autonomic scaling, and

monitoring (21). First it was developed for infrastructure as a service model, but current version is capable to serve all three main models of cloud computing (IaaS, PaaS, and SaaS). Similarly, *The Cloud Infrastructure Management Interface (CIMI)* (22) defines a model for the management of resources of infrastructure as a service. It addresses deploying and managing virtual machines, volumes, network and other IaaS artifacts. *The Open Virtualization Format (OVF)* (23) is the DMTF's standard that describes the open format for the virtual machines. It is optimized for the distribution of single or multiple virtual machines; it is vendor and platform independent, extensible and localizable. It provides the complete specification of a virtual machine.

OASIS's *Topology and Orchestration Specification for Cloud Applications* (TOSCA) aims to enhance the portability of application layer services across alternative clouds (24). TOSCA can be used to provide description of service components, their relationships and service management procedures (24). TOSCA's core concept is the ability to move services and applications between public and private cloud infrastructures, but the most prominent providers of infrastructure as a service have not yet joined this OASIS consortium (20). *Cloud Data Management Interface (CDMI)* specifies the interface for cloud storages and their successful management (25). It enables cloud programmers to discover the capabilities of the chosen cloud storage, to manage containers and their associated data, and to use metadata for containers and/or data objects. CDMI provides standardized interface by means of RESTful web services that can be used to create, retrieve, update and delete data objects. CDMI is now accepted as ISO standard in ISO/IEC 17826:2012.

Lewis's technical report (26) explores the role of standards regarding cloud interoperability. Her opinion is that cloud standards probably do not make sense beyond infrastructure as a service layer, because value-added features provided by PaaS and SaaS vendors automatically correspond to greater differences between them. Lewis (26) thinks that cloud standardization will take some time, similar to the development of web service standards in the past. Petcu (27) listed several barriers in cloud standardization. These include barriers to exit which many vendors put into their cloud offers, differentiated services of various commercial vendors, standards take years to mature, and different standards are needed for each of the three main models of cloud computing.

Demands for cloud standards are growing, but there is not a central body to lead the standardization, although many try to become such body. Cloud landscape is still in innovation phase, vendors often change their services, and many major market players do not support the existing standard initiatives and are not involved in the new ones. Many cloud standards are not mature yet. At this moment, the existing standards are not yet able to port applications and data from one vendor of platform as a service to another or find and solve interoperability problems between different APIs, so use cases shown in this dissertation cannot be solved by using one or more of the existing cloud standards.

2.5 Semantic Web and ontologies

2.5.1 Semantic Web and its standards

Most of the content on the Web is designed for people, and not for computer programs and agents. Programs can parse this content, but it is complex to process the semantics. The solution for the mentioned problem is the use of Semantic Web technologies. The ultimate goal of Semantic Web is to create structured and meaningful web pages that can be used by software agents capable of carrying out sophisticated users' tasks automatically (28). For now, there are many prototypes and proof-of-the-concept solutions, but the Semantic Web has not yet become the mainstream in the industry. The main idea of the Semantic Web is to provide coherent data model that is a part of the web infrastructure (29). One data item can point to another using standard links. The fundamental concepts of Semantic Web are (29): the AAA slogan (anyone can say anything about any topic), open world (it is assumed that there is always more information than known), and non-unique naming (the same entity can have more names).

Semantic modeling usually starts with the definition of the competency questions to determine what questions the model should answer (29). Semantic model should anticipate its possible usage by someone other than its designer in the future, and should be flexible regarding the ability to upgrade and merge with other semantic models. The meaning of classes and properties in Semantic Web differs from their meaning in object-oriented modeling and programming. Properties in Semantic Web exist independently of any class, they can be used to describe any individual, regardless of which classes it belongs to (29). Membership of individuals in multiple classes is also possible.

Semantic Web consists of a number of modeling languages that are organized in layers (29). The basis of Semantic Web is the Resource Description Framework (RDF) used for representing information about things (resources) that can be identified on the web by using web identifiers (URIs) (30). It can be represented as a graph of nodes and arcs depicting the resources, their properties and values. This modeling language uses a particular terminology for various parts of statements: the subject (the thing that the statement is about), the predicate (the property of the subject), and the object (the value of the property) (30).

The main elements of Web Ontology Language (OWL 2) are classes, properties, individuals, and data values (31). It is a semantic language designed to represent rich and complex knowledge. OWL 2 ontologies can be used together with the information written in RDF. The OWL 2 is chosen to design ontologies in this dissertation, since this modeling language contains all the concepts required to describe the functionalities of cloud resources, API operations, and interoperability problems of a platform as a service offers. It is also frequently used in related papers. OWL is described in more detail in the next subchapter (2.5.2).

2.5.2 OWL

An ontology in OWL is a set of precise statements about the domain of interest (31). Axioms in OWL are the basic statements of the OWL ontology (31). Entities include all types of elements used to refer to real-world objects to abstract categories (classes in OWL), relations (object properties, datatype properties, and annotation properties in OWL), and objects (individuals). The expressions are combinations of entities to represent complex descriptions (for example, the atomic classes “man” and “pilot” could be combined to new class of male pilots). The most important tools when working with OWL are ontology editors used to create and edit ontologies, and reasoners to infer logical consequences (31).

2.5.3 Ontology

The most cited definition of ontology is: *“An ontology is an explicit specification of a conceptualization“* (32). The ontology defines basic concepts and their relationships in a

specified domain of interest. Noy and McGuinness (33) define ontology as *"formal explicit description of concepts in a domain of discourse"* (33), together with their properties and restrictions. The ontologies are most often developed to share common understanding, reuse, separate, and analyze the existing domain knowledge, and make domain assumptions explicit (33).

An ontology consists of axioms that are stated in an ontology language (34). An ontology language lists the available language constructs and the formal semantics, and today there are many different ontology languages in use. Web ontologies are subtypes of ontologies designed by using one of the semantic web ontology languages described in the previous chapter. In his doctoral dissertation, Vrandenčić (34) lists the main elements of a web ontology: axioms (class axiom, property axiom), facts, annotation, ontology entity, individual, class and property.

2.5.4 Methodologies for ontology development

Bergam (35) reviewed the existing ontology development methodologies. Many ontology development methodologies were proposed in the existing literature. However, the pace of new methodology development has recently waned, but still there is no methodology that is dominantly used by most researchers. Some of the leading methodologies are (35): ONIONS (Ontologic Integration of Naive Sources), COINS (Context Interchange System), METHONTOLOGY, OTK (On-To-Knowledge), Cyc, TOVE (Toronto Virtual Enterprise), IDEF5 (Integrated Definition for Ontology Description Capture Method), UPON (United Process for Ontologies) and Ontology Development 101. Most methodologies share general logical steps from assessment to deployment, testing and improvement. For the purpose of this research, the Ontology Development 101 methodology (33) was chosen. The methodology itself and reasons why it was selected as the methodology for developing these ontologies is described in Chapter 5.1. In the next paragraph, other relevant methodologies will be elaborated.

Corcho et al. (36) reviewed the methodologies, tools and languages for building ontologies and relationships among them. Cyc (36) consists of manual codification of knowledge and it acquires new knowledge using natural language or some machine learning tools.

METHONTOLOGY (37) distinguishes the following ontology development states: specification, conceptualization, formalization, implementation, and maintenance. It promotes the evolving prototypes approach as a life cycle for developing ontologies. The specification phase produces specifications written in natural language or using competency questions. In the conceptualization activity, the obtained domain knowledge is structured into a conceptual model. METHONTOLOGY recommends the reuse of the existing ontologies when this is possible. The On-To-Knowledge methodology (36) is based on the analysis of ontology's usage scenarios and consists of the following phases: feasibility study, kick-off (ontology requirements, competency questions, and draft version of the ontology), refinement (mature ontology is produced), evaluation and ontology maintenance. In 2005, De Nicola et al. (38) proposed the UPON methodology which is inspired by Unified Software Development Process and uses Unified Modeling Language (UML) for the preparation of the ontology development project. UPON is a use-case driven and iterative process that has cycles. Each cycle has four phases (inception, elaboration, construction, and transition) and output of each cycle represents a new version of the ontology. Each mentioned phase is divided into iterations consisting of requirements, analysis, design, implementation, and test workflows. In 2013, Iqbal et al. (39) conducted a literature review on ontology engineering methodologies and concluded that none of the methodologies are fully mature, and recommended the use of METHONTOLOGY, UPON and Ontology Development 101 because they all follow an evolving prototype model and provide some details of used techniques and activities for ontology development.

2.6 Semantic service oriented architecture

2.6.1 SOA and web services

The main aim of service-oriented architecture (SOA) is to enable loosely coupled and protocol independent distributed computing (40). The main elements of SOA are services usually defined as self-contained software modules that are independent of other services (40). SOA is independent of any specific technology and it assumes that service can be dynamically located, invoked and combined. Each service consists of interface and service implementation. The preferred implementation technology for SOA is web services. W3C describes the term of web service as *"a software system identified by a URI, whose public interfaces and bindings are defined and described using XML"* (41).

Generally, there are two types of web service architecture common in practice: WS-* stack and RESTful web services. WS-* stack uses SOAP for the definition of message architecture and formats, WSDL to define interfaces syntactically, and a large set of WS-* specification to enable security, quality of service, and service interoperability (42). REST is based on four principles (42): resource identification through URI, uniform interface, self-descriptive messages and stateful interactions through hyperlinks. REST services are much simpler than SOAP stack, and they are based on standard HTTP methods. In the REST architecture, everything is a resource which can be located using URIs. Due to its simplicity, a very large number of service providers are switching to REST (43).

2.6.2 Semantic web services

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.

To implement semantic web service, new languages should be used. OWL-S (Semantic Markup for Web Services) is the ontology of services that enables users and/or software agents to discover, invoke, and compose web services (44). This ontology is defined by using the OWL language. It has three main parts: the service profile for specifying the intended purpose and functionality of the service; the process model for describing the operation of the service, and the grounding containing details on how to use a service. Next initiative, the Web Service Modeling Ontology (WSMO) is used for describing various aspects related to Semantic Web services (45). It is an extension of the Web Service Modeling Framework (WSMF). WSMF itself consists of four different elements: ontologies, goals, web services descriptions and mediators. WSMO refines and extends this framework by developing the ontology for the core elements of Semantic Web services and the description language that consists of non-functional, functional and behavioral aspects of web services.

WSMO and OWL-S are heavyweight solutions for semantic web services; they introduce new languages founded on expressive formalisms and promote the semantics-first modeling

approach (46). The heavyweight solutions are perceived as complex in terms of modeling and computational complexity (47). Lightweight approaches can be used to reduce the complexity and enhance the existing SOA capabilities with intelligent and automated integration on top of the existing service descriptions (48). Lightweight service ontologies use the bottom-up modeling. The most known lightweight approaches include WSMO-Lite, SAWSDL, MicroWSMO, hRESTS, SA-REST. Furthermore, lightweight service annotation models are surprisingly cost-effective, because work on the semantic annotation is faster.

To semantically annotate web services, SAWSDL will be used in this dissertation. It is chosen because SOAP web services described by WSDL were developed, because it is simple, it possesses a rich ontology-based data mediation mechanism for mapping inputs to outputs of web services, and there exists a tool that can be easily integrated into the Java application. SAWSDL consists of extension attributes for WSDL and XML Schema to add semantics to their components (49). It enables the usage of the semantic annotation by specifying references to semantic models such as OWL ontologies. The concept from the semantic models can be referenced from WSDL or XML schema. SAWSDL uses the following extension attributes (49):

• "modelReference" defines semantics of the inputs or outputs of WSDL operations.

• "liftingSchemaMapping" and "loweringSchemaMapping" are used to specify mappings between semantic data and mapping language

A model reference can be used with every WSDL element, but its meaning is defined in SAWSDL only for interface, operation, fault, xs:element, xs:complexType, xs:simpleType and xs:attribute (49). The same annotation on a WSDL operation or fault gives semantic information about the annotated operation or fault, and it provides a classification of the interface on a WSDL interface.

XML Schema simple types can be annotated by using *modelReference* attribute (49). Furthermore, complex types can be annotated using two techniques: bottom level (annotation of the member element or attribute) and top level (annotation of complex type itself) (31). A "modelReference" attribute can be used to annotate semantic mapping for the data type, but detailed mappings must be specified for the actual invocation by using "liftingSchemaMapping" and "loweringSchemaMapping" (50). For this purpose, SAWSDL

allows using any mapping language, and its documentation gives examples written in XQuery, XSLT, and SPARQL.

2.7 AI planning methods

Humans plan when addressing new or complex situations, when there is high risk from environment, or when they work with large numbers of coworkers on a project. Planning is *"an abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes"* (51). It aims at achieving the predefined objectives. Planning is a complex, time-consuming and costly process (51). Planning can come in different forms (51): path and motion planning (path from a starting point to a goal), perception planning (to sense actions for gathering information), navigation planning, manipulation planning (to handle objects), and communication planning. AI planning is mostly interested in domain-independent approaches where the input that the planner takes is the problem specifications and domain knowledge (51). Classical AI planning problem consists of a set of all the possible states, the initial state, the planning goal, and a set of actions together with their preconditions and effects (1). AI planning for automated service composition is described in more detail in the next chapter (2.7.1).

2.7.1 AI planning and automated service composition

In the current literature, automated composition of web services was performed using numerous methods, such as: Event Calculus (52), Petri Nets (53), Colored Petri Nets - (54) and (55), Linear Logic theorem proving (56), AI planning, logic programming, Markov process, States Machines, etc. AI planning is one of the most promising techniques to solve a problem of automated web service composition. Some of the most prominent papers will now be briefly listed. McDermott (57) showed how to compose simple web services using the extension of PDDL (Planning Domain Definition Language). Sirin et al. (58) described how to use AI planner software SHOP2 (Simple Hierarchical Task Network) to compose web services. Bertoli et al. (59) showed that the tools for automated service composition can be implemented by using and upgrading an AI planning techniques. They described the framework for automated service composition and algorithms to solve the service composition. At the end of their paper, they showed an implementation of the approach and

experimental results. Hatzi et al. (60) showed an integrated approach to automated composition of web services using AI planning techniques. The descriptions of web services in OWL-S were transformed into the AI planning problem using PDDL (the Planning Domain Definition Language), while the semantic information was used to improve the process of composition and to evaluate the optimal composition of services. The implementation of this approach was made by the integration of the two software systems.

2.7.2 Hierarchical Task Network (HTN) planning

Hierarchical Task Network (HTN planning) is the AI planning technique that is most widely used for practical applications. This is partly because it provides a convenient way to write problem-solving "recipes" that is similar to human domain in which an expert thinks about the ways of solving the problems of planning. An HTN planner uses domain knowledge and formulates the plan by recursively decomposing the tasks until it reaches primitive tasks that can be executed directly (61). As an illustration, Anshul Goyal (61) explored how HTN planning algorithm can be used to perform real-time planning in a stealth-based game. HTN planning is suitable for service composition because it encourages modularity, it can scale up to a large number of services, and it has means to deal with failures and costs (62). Sirin et al. (58) proved the semantic correspondences between the SHOP2 planner and OWL-S, and they showed how one can use SHOP2 planner to compose web services (62).

3. RELATED WORK

3.1 Review of interoperability

3.1.1 Interoperability problems, issues and conflicts

Interoperability is a multidimensional concept where interoperability problems, issues, and conflicts can occur on multiple levels. The best description of these levels is given in some interoperability frameworks. European Interoperability Framework (EIF) (63) aims at interoperability of European public services and identifies four levels of interoperability: legal (due to incompatibilities of legislation in different EU countries), organizational (business process incompatibilities), semantic (it is caused by conflicts of the meaning of data elements and format of the exchanged information), and technical interoperability. IDEAS interoperability framework (64) distinguishes the following interoperability levels: business, knowledge, ICT system level, and semantic level. The ATHENA interoperability framework (65) lists four interoperability levels: enterprise/business (organizational and operational ability of an organization to work with other organizations), process (level of business processes), service (flexible execution and composition of services) and information/data (management and exchange of information). Enterprise interoperability framework (64) identifies three categories of interoperability barriers: conceptual barriers (syntactic and semantic data conflicts), technological barriers (incompatibilities between different architectures, platforms and infrastructures), and organizational barriers (organizational responsibility, authority and structures). GridWise interoperability framework (66) lists three interoperability aspects: technical aspects (basic connectivity, network interoperability, syntactic interoperability), informational aspects (semantic understanding and knowledge to apply these semantics in process workflows), and organizational aspects (alignment between business processes, shared business objectives, and economic/regulatory policy).

Legal and organizational interoperability issues will be observed first. Rosati and Lamar (67) listed privacy, security, Stark Law, non-profit task, antitrust, intellectual property, medical malpractice and state law issues as the most important legal issues when dealing with interoperable electronic health records. The results of a case study executed by Hellman (68) showed ten barriers to organizational interoperability: competency gaps, missing indicators for measuring organizational interoperability, funding the interoperability projects, national

joint efforts, disconnected small projects, different legislation, anemic arenas, under investigated best practice, people and their managers and ubiquitous heterogeneity. Rauffet at al. (69) conducted two case studies and discovered the following issues regarding organizational interoperability between two examined organizations: heterogeneities in functional practices and processes, communication between different actors, and a missing means to manage heterogeneous and complex structures. Rana and Ion (70) claim that many legal issues arise when two organizations work together and give an example of antitrust law. Vernadat (71) lists several possible organizational issues: different human behaviors, various organizational structures, heterogeneities in business process organization and management, different value creation networks and business goals. The main objective of an organizational interoperability (71) is to coordinate business processes, enable collaboration between the involved organizations, and address the requirements of users.

In this dissertation, the focus is on the technical and semantic interoperability issues among commercial providers of platform as a service. For this reason, the next paragraphs will elaborate on the mentioned types of interoperability problems in more detail. Sheth and Kashyap (72) classified and defined the most important interoperability conflicts among multiple independent database systems. They listed the following main categories of incompatibilities (72): 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 corresponds to schema elements in another). For each incompatibility category, Sheth and Kashyap listed possible concrete conflicts.

Parent and Spaccapietra (73) listed the most relevant issues and the approaches to tackle data interoperability problem when integrating databases. They distinguished seven categories (73):

- Heterogeneity conflicts: different data models
- Generalization/specialization conflicts: different generalization/specialization hierarchies and different classification abstractions
- Description conflicts: types have different properties and/or their properties are described differently (73)
- Structural conflicts: different structures of related types

- Fragmentation conflicts: the same object is depicted by decomposition into different elements (73)
- Metadata conflicts
- Data conflicts: data instances have different values for the same properties.

Park and Ram (15) 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 (15): naming conflicts, entity-identifier problems, schema-isomorphisms, conflicts of generalization, aggregation conflicts, and schematic discrepancies.

Haslhofer and Klas (74) dealt with metadata interoperability and provided a classification of heterogeneities impending interoperability from a model-centric perspective. They distinguish two classes of heterogeneities: structural heterogeneity and semantic heterogeneity. Structural heterogeneities occur at the model level in the form of:
- Naming conflicts – different names of model elements that represent the same real object (74).
- Identification conflicts – model elements are identifiable by their name or by identifier
- Constraints conflicts – different definition of constraints in different models (74)
- Abstraction level incompatibilities – different generalization of aggregation of the same real-world object (74)
- Multilateral correspondences – an element from one model corresponds to multiple models in another model (74)
- Meta-level discrepancy – The same elements in one model could be modeled differently in another model (74).
- Domain coverage – Real-world concepts described in one model are missing from the other model (74).

Semantic heterogeneities occur because of the differences in the semantics of models:
- Domain conflicts – incompatible or overlapping domains (74).
- Terminological mismatches – synonyms or homonyms

- Scaling/unit conflicts
- Representation conflicts – different encoding schemes for content values (74).

Ponnekanti and Fox (75) examined if it is feasible to substitute one vendor service for another when using SOAP web services. They classified interoperation incompatibilities into: structural (a mismatch in the structure of ingoing and outgoing messages), value (occurrence of unexpected filled-in values in ingoing or outgoing messages), encoding and semantic (vendors' extensions with the same structure and value, but different meaning). In their paper, Ponnekanti and Fox (75) focused on structural and value incompatibilities and defined the following classes of incompatibilities: missing methods, extra fields, missing fields, facet mismatches (different types for input or output fields), and cardinality mismatches (different cardinality requirements for the field). Zhu et al. (76) addressed the problem of large scale data integration in the healthcare domain and described the following heterogeneities among different data sources:

- Naming – synonyms and homonyms
- Relational structure varies
- Value – different representations of values in different databases
- Semantic – differences in meaning or the context in different databases
- Data model differences and transformations
- Timing – changes in the structure of the database, attribute representations and values over time
- Syntax – query languages may be different
- Different transaction mechanisms in different databases
- Different security mechanisms and policies in different systems

Nagarajan et al. (77) classified the types of heterogeneities that can occur between web services and presented a possible solution for data interoperability using semantic descriptions and schema/data mappings. They used pre-defined mappings to enable data mediation in web services environment. Message or data heterogeneities exist when the data elements sent between the two services are incompatible (77). There are no syntactic heterogeneities, because the XML resolves them. The main classes of heterogeneities in web services are (77): attribute level incompatibilities (different descriptions are used to model similar attributes), entity definition incompatibilities (different descriptions are used to model similar entities),

and abstraction level incompatibilities - different levels of abstraction (77). In their approach to solve the aforementioned interoperability problems, Nagarajan et al. (77) used SAWSDL (49) to annotate web services with semantic concepts from an external ontology. Data mediation between services can be achieved by means of the manual mappings, but these mappings have to be changed every time the services are modified. An alternative solution is more flexible, and it maps inputs and outputs to a conceptual model (77). Through ontologies, web services can resolve their message level heterogeneities. A similar approach will be used in this dissertation to solve interoperability problems among API operations of different PaaS vendors. The support for data mediation in SAWSDL is provided by using the 'liftingSchemaMapping' and 'loweringSchemaMapping' attributes on web service message input and output elements to create mappings with the ontology concept with which input or output is associated with (77).

3.1.2 Cloud computing interoperability

Basically, cloud users want to be able to transfer data or applications among multiple cloud environments and connect each other across various clouds (78). Petcu (27) listed various definitions of cloud interoperability as the ability to model the programmatic differences, translate between different abstractions of clouds, move applications from one cloud to another, enable applications to run on multiple clouds, port data between cloud providers, use unified management tools for multiple clouds. The development of a common interface for accessing a variety of clouds in a unique way is shown by Tao et al. (79). This paper demonstrates functions of a service request and the graphical interface to display information about cloud computing services that are available to the user. Rodero-Merino et al. (80) propose a new abstraction layer for infrastructure as a service. This layer is closer to the service lifecycle and it provides automatic deployment, definition and management of services. Ranabahu and Maximilien (81) describe their own Altocumulus middleware to homogenize different cloud solutions and the associated cloud best practice model. Bernstein and Vij (82) present their InterCloud Directories and Exchanges mediator to enable connectivity and collaboration among cloud vendors. They define their ontology of cloud computing resources by means of the Resource Description Framework (RDF). Merzky, Stamou and Jha (83) demonstrate a proof-of-concept of application-level interoperability among different clouds and grids by means of the SAGA-based implementation of MapReduce. They developed a range of cloud adaptors for SAGA. MapReduce is a Google's

programming framework used to simplify data processing of massive data, and SAGA is a programming interface for developing distributed applications in an infrastructure independent way.

Ranabahu and Sheth (84) 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). Buyya et al. (85) present the vision, challenges and architecture of a utility-oriented federation of cloud computing environments. In their paper, Buyya et al. advocate the creation of a federated cloud. The reference architecture for semantically interoperable clouds (RASIC) was proposed by Loutas et al. (86). The main aim of the architecture is to enable the design, deployment and execution of new services using semantic descriptions of different cloud computing offerings.

Demchenko et al. (87) presented their inter-cloud architecture which they plan to use as a basis for building framework for cloud service integration. This is work in progress, and only the initial abstract model was defined. In his master's thesis, Fazai (88) proposed a three-dimensional space to assess the cloud provider's interoperability level. He argues that before choosing any provider, clients need to answer questions of vendor's interoperability level. The first dimension of Fazai's model is technology. It represents the interoperability level of the technology used by the specified cloud provider and it evaluates whether users can move their applications, data and virtual machines without significant effort. Management dimension includes vendor's management's tools and their level to support interoperability. The third dimension is concerned with provider's constraints or regulations.

Miranda et al. (89) used software adaptation techniques to tackle cloud interoperability and migration. Software adaptation techniques are aimed at developing mediator elements, called adaptors (89). They identified three important interoperability problems of cloud service based applications: communication is conditioned by the technology supported by each vendor, invoking third-party services is limited by the supported invocation mechanisms, portability problems occur due to vendor-specific technologies. The variability among

different providers' APIs and service specifications can be defined by using formal methods and by generating the required mappings and adaptation components. Bhukya et al. (90) showed how to use web services to connect Google App Engine and Windows Azure. Bastiao Silva et al. (91) developed a unified API for delivering services using cloud resources of multiple vendors with abstract layer for cloud blob stores, cloud columnar data (e.g. *Azure Table*), and *Publish/Subscribe* mechanism (*Channel API* of *Google App Engine* and *Azure Queue*).

Ma et al. (92) introduced service mediators that mediate the collaboration of services. Their idea is to specify service-oriented applications that involve yet unknown component services. Their mediator consists of local components and yet unknown services and it specifies the flow of data in and out of services. They argue that a service description should comprise three parts: a functional description of inputs and outputs, pre- and post-conditions; a category of the service operation; and a quality of service (QoS) attributes. Khalfallah et al. (93) proposed the use of a two-phase semantic data mediation model and a cloud-based platform to achieve interoperability for collaborative product development in the aerospace industry. They converted the proprietary data models into OWL ontology and mapped it to the reference OWL ontology that contains concepts from data exchange standards in the aerospace industry. They also used other conversion rules for data transformations. Their mapping ontology describes the concepts to map classes, object properties and data properties.

Guillen et al. (94) proposed a framework for cloud agnostic software development. An application is converted into sets of cloud artifacts that contain predefined structure, source code, adapters and interoperability elements. A deployment plan of a software project contains cloud artifacts and their configuration parameters, services to achieve interoperability, and adapters for cloud integration. The core component of the framework is Cloud Variability Model (94) that contains information (service catalogue, technological restrictions, templates, and configuration parameters) about all supported cloud platforms.

There are several cloud APIs and frameworks that act as intermediaries between different clouds. Apache Libcloud (95) 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 (96) 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 (97) 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 transfer of data between cloud application such as Salesforce, Tumblr, Nimble, Pipedrive, SugarCRM, and Zoho CRM. Import2 is currently focused on CRM, helpdesk and blog migration of cloud data. The mentioned two offers are focused on SaaS data.

3.1.3 Research projects on the cloud interoperability

Cloud computing interoperability is a very active research topic and several European research projects used to be or are currently concerned with it. The main objective of the FP7-funded Cloud4SOA project (16) was to open up the cloud market to small and medium European providers of platform as a service and solve the vendor lock-in problem. Researchers involved in the mentioned cloud project planned to semantically interconnect heterogeneous platform as a service (PaaS) solutions that share the same technology (programming language and frameworks). The main research objectives were: design of semantic interoperability framework, introduction of reference architecture to interconnect different clouds and development of Cloud4SOA system. This project is dealing with semantic interoperability at platform level (98). Cloud4SOA interoperability framework is described in the deliverable D1.2 (16). Cloud Semantic Interoperability Framework has the following core dimensions (86): fundamental entities (e.g. system, offering, API, cloud application), types of semantics (e.g. functional, non-functional, execution), and semantic conflict levels (information model and data). Loutas et al. (86) claim that a semantic conflict arises when semantic descriptions of the aforementioned fundamental PaaS entities are incompatible. The core capabilities of Cloud4SOA are (99):

- Semantic matchmaking – It lists the offerings of platform as a service that satisfies defined user requirements
- Management capability – It supports the deployment and management of applications on PaaS offers.
- Migration capability – It enables migrating applications from one supported PaaS offer to another.
- Monitoring capability – It monitors the performance of application hosted on clouds.

Cloud4SOA uses repository layer to store semantic and syntactic data (99). Semantic data includes RDF triples of developer's profiles and PaaS providers' capabilities. Harmonized API component is a unified PaaS API that contains a number of operations for the management of the cloud applications. The adapter that translates operations of this unified API to vendors' native APIs is also developed. Cloud4SOA API includes methods for working with instances of platform as a service, for deployment of applications, for migration of the application, for monitoring, for discovering offering of platform as a service, for recommendation of PaaS offerings, and for user management.

Another FP7 project, mOSAIC (100), aims at creating, promoting and exploiting an open-source Cloud API and a platform targeted for developing multi-cloud applications. The existence of standard API could simplify the development process, increase the adoption of cloud services and enable the interoperability of data and services of different cloud providers. Petcu et al. (98) presented an integrated overview of the mOSAIC's architecture and its various usage scenarios.

The FP7-funded Contrail project (101) is aimed at designing an open source system for cloud federations. Cloud federation in Contrail implies (102) integrating platform as a service and infrastructure as a service offers, integrating resources from other clouds with private infrastructure, and allowing live application migration across clouds. The Contrail project key objectives are: to support the pay-per-use model, to enable users to specify the quality of service requirements in the Service Level Agreements (SLAs) and to integrate elastic resource provisioning capabilities to the deployed applications. Contrail is developing a software stack that enables (103): federation (combines services from different cloud providers), identity management (federated identity management to use all services from different cloud vendors), federated service level agreements (user defines them and the system translates them into

requirements), cloud file system, and interoperability layer that eases the management of infrastructure and deployment of the application.

Vision Cloud project (104) 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. Federated cloud assumes a formal vendors' agreement, while the term multi-cloud (105) denotes the usage of multiple independent clouds. Five areas of innovation in the VISION Cloud platform (104) 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. Vision Cloud builds storage of tens of data centers, and it can serve millions of clients with billions of data objects that contain data of arbitrary type and size and corresponding metadata. 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 another. Vision Cloud's approach uses a cloud storage container as the basic unit of federation. Vision Cloud offers a RESTful API to manage data federation. The *Federator-Direct* component provides a unified view of a data container distributed over the new and old cloud (106). The *FederatorJobsExecutor* is responsible for moving the data and their corresponding metadata. *Multi-Cloud Adapter* implements multiple existing cloud data APIs and converts metadata formats.

Mohagheghi and Saether (107) presented the achievements of REMICS whose main aim was the development of the methodology and tools for model-driven migration of legacy applications to software as a service solutions. The primary goal of the mentioned project is to transform legacy systems into UML models, and to manipulate these models to migrate applications to clouds. REMICS extracts the architecture of the legacy application, analyzes it and finds out how to modernize it. This information is converted into models that represent the start of the migration activity. Researchers working on REMICS project defined a methodology (108) for the migration of legacy systems to clouds. Their methodology consists of the following activities: requirements and feasibility (to gather the migration requirements),

recover (to get the application model of the legacy application), migrate (to migrate to the cloud), validation (to define testing strategy), supervise (to control the performance of the system), interoperability (to solve interoperability problems), and withdrawal (to stop the service). They developed the PIM4Cloud (109) metamodel and UML profile, an extension to the existing modeling standards that supports specification of deployment to cloud platforms from an application designer perspective. For now, PIM4Cloud does not provide elements to abstract PaaS in application models due to the high degree of heterogeneities of PaaS solutions (109).

MODAClouds (110) plans to provide methods for deployment on multiple clouds and for data synchronization among multiple clouds by using model-driven techniques. This project is in initial phase; it started in October 2012 and will last until September 2015. They plan to develop MODAClouds IDE to support a cloud-agnostic design of software.

3.1.4 Cloud computing interoperability use cases

Several cloud computing interoperability use cases have already been described in the current literature. The FP7 project Cloud4SOA (111) defines the following usage scenarios: deploying a service-based application on the Cloud4SOA platform, and migration to/deployment on a different platform as a service provider. In the other deliverable of the same project, four semantic interoperability use cases were defined (16):

• Deployment of an application on a PaaS offering
• Migration of an application deployed on one PaaS solution to a different PaaS offering
• Hybrid clouds: PaaS systems/offering interoperation
• Integration between applications deployed on different PaaS offerings

Another FP7 project, Contrail (112), describes four use cases that represent a diverse set of requirements:

• Distributed provision of geo-referenced data which is an implementation of a 3D Virtual Tourist Guide (VTG service)
• Multimedia processing service marketplace that will exploit Contrail federated cloud to develop a marketplace offering multimedia services to end-users
• Scientific data analysis that will archive climate model output data and the neutron scattering

• Electronic drug discovery use case plans to use modern bioinformatics tools/applications on a federated cloud system

The research project mOSAIC (113) covers three basic business scenarios for using multiple Clouds:

• Switch the cloud – Application developers or their clients should easily change the cloud provider

• Service brokerage – Finding the best cloud services for a certain application

• Development of cloud applications

There are several application scenarios (113) that will be deployed by project partners:

• Document manager – Receives and classifies documents and offers dedicated services for searching them.

• Cloud bursting – In order to face a peak of requests, the provider buys additional resources from other cloud providers and resells them to its final users.

• A port of document transformation and information extraction algorithms into the cloud environment.

• Structural dynamics application that is used by civil engineers to study the behavior of a structure when subjected to some action.

• Earth observation

• A railway company – A project for the maintenance of devices, early diagnosis of faults and real-time monitoring.

The real world scenarios that drive Vision Cloud FP7 project (114) are:

• SAP – Business intelligence on-demand – Vision Cloud will be used for storage, data mobility and data federation

• Telco use case – telecommunication operators want to offer data-intensive applications with high quality of service

• Media use cases – videos in clouds

• Healthcare use case – personalized healthcare applications based on patient health records

Badger et al. from NIST (115) listed 25 cloud computing use cases, and some of them are directly related to cloud interoperability:

• Copy data object between cloud providers

• Cloud burst from data center to cloud

• Dynamic operation dispatch to infrastructure as a service (IaaS)

• Migrate a queuing-based application

Some use case scenarios from the Open Cloud Manifesto (116) are also related to interoperability:

• Cloud applications deployed on the public cloud interoperate with partner applications

• Switching cloud vendors or working with additional ones

• Hybrid clouds – multiple different clouds should be able to work together to federate data and applications

Microsoft established the IEC Council in June 2006 as a means of regularly interacting with customers to solve their technology interoperability challenges. Their white paper (117) describes ten of the most common cloud computing use cases from a practical point of view based on customer experience:

• Move three-tier application from own servers to cloud

• Move three-tier application deployed on cloud to another cloud vendor

• Move part of application to cloud to create hybrid applications

• Hybrid application with shared user identity

• Move hybrid application to another similar cloud

• Hybrid cloud application using platform services

• Port cloud application using platform services to another cloud vendor

• Develop cloud application for multiple clouds

• Cloud application workload requires use of multiple clouds (cloudburst)

• Users can "shop around" for cloud services

Even from the first use case, application and data portability is a key requirement (117). A raw listing of use cases and scenarios from different sources can be summarized, so use case and scenarios are here divided into five categories described in Table 1.

Table 1 Summary of the existing use cases and scenarios

Category	Description	List of use cases and scenarios
Cloud deployment and migration	Development and deployment on cloud, migration of data and application from on-premise to cloud or from one cloud to another	- Application deployment on a PaaS solution (16) - Migration of an application to a different PaaS offering (16) - Changing the cloud (113) - Development of cloud applications (113) - Copy data object between cloud providers (115)

		- Migration of a queuing-based application (115) - Move three-tier application to cloud (117) - Move three-tier cloud application to another cloud vendor (117) - Move hybrid application to another similar cloud (117) - Port cloud application that uses platform services to another cloud (117) - Move part of application to cloud to create hybrid applications (117) - Switch cloud providers or work with additional providers (116)
Cloud application cooperation	Cooperation among two or more applications on different clouds, or cooperation among components of one application where components are deployed on multiple clouds	- Integration between applications on different PaaS offerings (111) - Cloud applications deployed on the public cloud interoperate with partner applications (116) - Hybrid application with shared user identity (117) - Hybrid cloud application that uses platform services (117) - Create cloud application with components that run on multiple clouds (117)
Federated cloud	Data and/or applications use federated cloud	- Distributed provision of geo referenced data (112) - Multimedia processing service marketplace (112) - Scientific data analysis on federated cloud (112) - Electronic drug discovery on a federated cloud system (112) - Business intelligence on-demand for storage, data mobility and data federation (114) - Telco use case to offer data-intensive applications with high quality of service (114) - Media use cases with videos in clouds (114) - Personalized healthcare applications based on patient health (114) - Hybrid clouds where multiple clouds work together (116)
Cloudburst	Cloud application requires	- Cloud burst from data center to cloud (115)

	use of multiple clouds	- Dynamic operation dispatch to infrastructure as a service (115) - Cloud application requires use of multiple clouds (117)
Brokerage of cloud services	Finding appropriate cloud services among services of different providers	- Finding the best cloud services for a certain application (113) - Users can “shop around” for cloud services (117)

3.1.5 Interoperability methodologies

There are some interoperability methodologies in the existing literatures that are mostly concerned with enterprise interoperability. The ATHENA Interoperability Methodology (AIM) (118) is an extension of the Unified Software Development Process (UP) (119) which introduces a group of interoperability activities. AIM is used to identify interoperability issues and select the adequate ATHENA solutions. Chen and Daclin (120) proposed four main phases of methodology for interoperability:

- Definition of interoperability objectives and needs
- Analysis of the existing system to identify interoperability barriers and measure current interoperability level
- Select and combine solutions
- Implementation and testing

Sanati et al. (121) presented their E-service Integration Methodology (E-SIM) to solve complex interoperability problems and configure service workflow. The tasks in the mentioned methodology include specification of life-event requirements of the user of the service, specification of interoperability requirements at business process, data, and interface levels, detailed design of e-government services, design and implementation of Semantic Web specifications.

European Interoperability Framework (EIF) (63) addresses interoperability of European public services at four identified interoperability levels: legal, organizational, semantic, and technical. The involved public organizations should make interoperability agreements for each level, such as agreements on transposition of European directives to national legislation,

SLAs, reference taxonomies, code lists, data dictionaries, interface specifications, data formats etc. Interoperability agreements specify one or more interoperability solutions that are implemented by one or more interoperability solution instances. Due to environment changes, interoperability of European public services is a continuous task.

3.1.6 Data mediation between the services

When dealing with the composition of web services, a dominant interoperability problem is how to map the inputs and outputs of the involved services, and in most cases, data mediation is required to achieve interoperability among web services (122). Many works in the existing literature address the mentioned problem. Nagarajan et al. (122) proposed a data mediation architecture that uses WSDL-S for mapping from inputs and outputs to common ontology and vice versa. The web services should be semantically annotated by using WSDL-S, and mapping engine was used to transform SOAP messages according to defined XSLT or XQuery mappings. WSDL-S later became the main input for W3C recommendation SAWSDL that provides similar data mediation mechanism. The main contributors of SAWSDL standard were members of METEOR-S research project and IBM (123). Sheth et al. (123) claim that key SAWSDL's benefit is systematic data mediation where XSLT is used to map a service schema to the ontology (lifting schema mapping) and vice versa (lowering schema mapping). Klímek and Necaský (124) introduced a model-driven method to automatically generate XSLT for lifting and lowering schema mappings and its prototype implementation.

Li et al. (125) presented an approach to reconciliate semantic conflicts in the composition of web services. They used COIN ontology, SAWSDL and mapping algorithms to handle complex differences by using minimal numbers of predefined transformations. The method to automatically analyze data flows of BPEL process and automatically determine possible semantic differences is also shown in the same paper. Stollberg et al. (126) proposed mediation model for Semantic web services using WSMO mediators at data, functional, and process level.

3.2 Ontologies

3.2.1 Cloud ontologies

There are several existing studies involving cloud computing ontologies. One of the first attempts was introduced in Youseff at al. (127). They presented an ontology which differentiates five main layers of cloud computing (applications, software environments, software infrastructure, software kernel and hardware). Weinhardt et al. (128) 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. (129) introduced a formal catalog of cloud computing services modeled by means of ontological representation. Takahashi, Kadobayashi and Fujiwara (130) applied the ontology for cyber security to cloud computing. Adrian Martinez et al. (131) 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 (132) 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 (82) developed a mediator to enable collaboration among different cloud vendors. They defined the ontology of cloud computing resources using RDF.

Han and Sim (133) 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. Kang and Sim (134) proposed a 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. Dastjerdi et al. (135) presented an ontology-based discovery architecture providing QoS-aware deployment of virtual appliances on infrastructure as a service. Ma et al. (136) 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.

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 ontology, but it is focused on infrastructure as a service model and SLA. The ontologies presented in this dissertation are focused on remote operations of PaaS providers' APIs and interoperability problems among different platform as a service offers. There are not any similar ontologies in the existing literature.

3.2.2 Ontology anomalies and ontology evaluation

There are some ontology anomalies and pitfalls that can arise during ontology modeling. Poveda-Villalón et al. (137) manually inspected pitfalls in ontologies of 26 students. They have identified 24 pitfalls and classified them into (137): 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 (138), the same authors presented a web based tool called OOPS! that can detect the mentioned anomalies in OWL ontology. Baumeister and Seipel (139) 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 (34). Gomez-Perez (140) divides ontology evaluation into ontology verification and ontology validation. Lovrenčić and Čubrilo (141) also recognized the fact that the most important parts of an ontology evaluation are verification and validation. Ontology verification evaluates the correctness of the ontology building process. It finds errors such as circular class hierarchies, redundant axioms, and inconsistent naming schemes. Ontology validation evaluates whether the meanings of ontology elements really match the specified conceptualization.

Vrandečić (34) 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 (34). These questions can be formalized in a semantic query language.

Brank et al. (142) 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 Brank et al. (142) 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. Amirhosseini and Salim (143) 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).

3.3 AI planning

3.3.1 AI planning methods and cloud computing

Some initial works on using AI planning methods in cloud computing were published. Weber et al. (144) used an AI planner to discover the appropriate sequence of the available Amazon API operations in order to rollback to checkpoint. They implemented a proof-of-concept prototype by choosing AWS as the domain and the PDDL as the planning formalism. They formalized part of Amazon AWS APIs in a planning domain model, and used planner to create undo sequences for rollback. Zou et al. (145) proposed a framework for web service composition in multi-cloud environments. Their proposed method is based on AI planning and combinatorial optimization to minimize the number of clouds involved in a service composition sequence. They tried to upgrade traditional web service selection and composition methods to address new possible requirements where web services can reside on multiple clouds. Different cloud platforms have different functionalities in terms of adaptivity, scalability, and load scheduling, so different algorithms are needed for the selection and composition of web services deployed on various clouds (145). They modeled web services in multiple clouds as trees, defined an approximation algorithm to select services, and used AI planning for service composition.

3.3.2 Gaps in planning domains

Goebelbecker et al. (146) addressed the problem when AI planners are unable to come up with a plan. They presented an algorithm to find excuses for not being able to find a plan. Planning task can be changed so it is possible to generate a solution and find out why planning failed in the first place (146). In their paper, they concentrated on the changes to the initial state. Excuses enable users not only to realize that something went wrong, but also to decipher what went wrong. They defined an excuse as a change in the initial state without adding fluent values that contribute to the plan's goal (146). Goebelbecker et al. (146) transformed the problem of finding excuses into planning a problem by adding new operators that change the candidates of excuses. Kungas and Matskin (147) showed how to apply partial deduction for finding possible missing web services and identifying possible inconsistencies

in the descriptions of semantic web services. Yan et al. (148) proposed repair techniques instead of recomposition when available web services and requirements change and compositions become broken. They used graph planning to complete this task. Vukovic and Robinson (149) presented framework that reformulates failed goals into new AI planning problems and to show partial satisfaction of a goal. Friedrich et al. (150) proposed a self-healing model-based approach to dynamically create repair plans for faulty activities in web service compositions.

3.4 Summary of the existing work

To present the summary of the existing work, the method of systematic mapping study was chosen. The main aim of these studies is to give an overview of a research field. Petersen et al. (151) listed five essential steps to perform a systematic mapping study in software engineering:

1. Definition of research questions – Research questions are specified to determine the research scope of the systematic mapping study.
2. Conduct search – Studies are found by executing a search string derived from research questions on scientific databases.
3. Screening of papers – Irrelevant papers are excluded based on the defined inclusion and exclusion criteria.
4. Keywording using abstracts – In this step, researchers need to read abstracts, look for keywords (main concepts) and build classification scheme.
5. Data extraction and mapping of studies – The relevant studies are presented and summarized in the form of a systematic map.

3.4.1 Systematic mapping study on cloud interoperability

Based on the steps proposed by Peterson et al. and described in the previous chapter, the systematic mapping study on cloud interoperability was performed in July 2014. The main aim of this study was to get an overview of the existing work on cloud interoperability, determine which interoperability of which cloud model (IaaS, PaaS, or SaaS) is most investigated, and recognize the main existing methods and tools used to achieve cloud

interoperability. These goals are reflected in the defined research questions for the mapping study.

3.4.1.1 Research scope of the systematic mapping

RQ1: Which model of cloud computing is best investigated in the existing literature regarding cloud interoperability?

RQ2: What types of papers are published in the cloud interoperability area?

RQ3: What are the most frequently applied methods and techniques to achieve cloud interoperability?

RQ4: Which journals include papers on cloud interoperability?

RQ5: Which types of interoperability problems are most investigated?

3.4.1.2 Conduct search

The studies were identified by using a search string ("cloud interoperability" OR "cloud provider lock-in" OR "cloud vendor lock-in") on the following databases: IEEE Xplore, Scopus, INSPEC, Science Direct, Springer Link, Web of Science, and Google Scholar. The full text search was performed on 15th July 2014. A total of 1182 publications were identified and their distribution per scientific database is shown in Table 2.

Table 2 Distribution of the found publications

Source	Number of publications
IEEE Xplore	146
Scopus	108
INSPEC	30
Science Direct	35
Springer Link	91
Google Scholar	772

3.4.1.3 Screening of papers

Irrelevant studies (publications that are not relevant to answer the stated research questions of the systematic mapping) were excluded based on the analysis of their titles, abstracts and keywords. Book chapters, scientific conferences and journal papers on cloud interoperability and cloud provider lock-in were included. Duplicate studies and papers not written in English

were excluded. If several papers reported the same findings, only the newest work was included. If abstract was not good enough to determine whether the focus of the work is on cloud interoperability, introduction and conclusion was read to determine whether to include this article or not. Review papers were excluded; only papers that describe solutions to some cloud interoperability problems were included. Finally, the list of all 41 studies considered to be relevant is shown in Table 3.

Table 3 Full list of identified relevant papers

Id	Authors	Paper title
P1	Dowell et al. (152)	Cloud to Cloud Interoperability
P2	Di Martino et al. (153)	Semantic and Agnostic Representation of Cloud Patterns for Cloud Interoperability and Portability
P3	Petcu et al. (154)	Building an interoperability API for Sky computing
P4	Hill and Humphrey (155)	CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications
P5	Loutas et al. (156)	A Semantic Interoperability Framework for Cloud Platform as a Service
P6	Mindruta and Fortis (157)	A Semantic Registry for Cloud Services
P7	Thabet and Boufaida (158)	An Agent-Based Architecture and a Two-Phase Protocol for the Data Portability in Clouds
P8	Emeakaroha at al. (159)	Analysis of Data Interchange Formats for Interoperable and Efficient Data Communication in Clouds
P9	Miranda et al. (160)	Assisting Cloud Service Migration Using Software Adaptation Techniques
P10	Boob et al. (161)	Automated Instantiation of Heterogeneous FastFlow CPU/GPU Parallel Pattern Applications in Clouds
P11	de Morais et al. (162)	Cloud-Aware Middleware
P12	Nguyen et al. (163)	Development and Deployment of Cloud Services via Abstraction Layer
P13	Maheshwari et al. (164)	Evaluating Cloud Computing Techniques for Smart Power Grid Design Using Parallel Scripting
P14	Oprescu et al. (165)	ICOMF: Towards a Multi-Cloud Ecosystem for Dynamic Resource Composition and Scaling
P15	Demchenko et al. (166)	Intercloud Architecture Framework for Heterogeneous Cloud Based Infrastructure Services Provisioning On-Demand

P16	Li et al. (167)	Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines
P17	Abdul-Rahman and Aida (168)	Multi-Layered Architecture for the Management of Virtualized Application Environments within Inter-Cloud Platforms
P18	Michon et al. (169)	Porting Grid Applications to the Cloud with Schlouder
P19	Miceli et al. (170)	Programming Abstractions for Data Intensive Computing on Clouds and Grids
P20	Kotecha et al. (171)	Query Translation for Cloud Databases
P21	da Silva and Lucrédio (172)	Towards a Model-Driven Approach for Promoting Cloud PaaS Portability
P22	Strijkers et al. (173)	Towards an Operating System for Intercloud
P23	Aversa et al. (174)	Cloud Agency: A Guide through the Clouds
P24	Steinbauer et al. (175)	Challenges in the Management of Federated Heterogeneous Scientific Clouds
P25	Ciuffoletti (176)	A Simple and Generic Interface for a Cloud Monitoring Service
P26	Amato and Venticinque (177)	A Distributed Agent-Based Decision Support for Cloud Brokering
P27	Lordan et al. (178)	ServiceSs: An Interoperable Programming Framework for the Cloud
P28	Di Martino and Cretella (179)	Semantic Technology for Supporting Software Portability and Interoperability in the Cloud-Contributions from the MOSAIC Project
P29	Sotiriadis et al. (180)	Meta-Scheduling Algorithms for Managing Inter-Cloud Interoperability
P30	Bastião Silva et al. (91)	A Common API for Delivering Services over Multi-Vendor Cloud Resources
P31	Zeginis et al. (181)	A User-Centric Multi-PaaS Application Management Solution for Hybrid Multi-Cloud Scenarios
P32	Andročec and Vrček (182)	Platform as a Service API Ontology
P33	Amin et al. (183)	Intercloud Message Exchange Middleware
P34	Kostoska et al. (184)	A New Cloud Services Portability Platform
P35	Rezaei et al. (78)	A Semantic Interoperability Framework for Software as a Service Systems in Cloud Computing Environments
P36	Guillén et al. (94)	A Service-Oriented Framework for Developing

		Cross Cloud Migratable Software
P37	Amato et al. (185)	Vendor Agents for IAAS Cloud Interoperability
P38	Wright et al. (186)	A Constraints-Based Resource Discovery Model for Multi-Provider Cloud Environments
P39	Zhang et al. (187)	A Survey on Cloud Interoperability: Taxonomies, Standards, and Practice
P40	Kamateri et al. (99)	Cloud4SOA: A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability
P41	Woo and Mirkovic (188)	Optimal Application Allocation on Multiple Public Clouds

3.4.1.4 Classification scheme

The next step proposed by Petersen et al. (151) is to read abstracts of the selected primary studies and write out relevant keywords and concepts to understand the contributions of each study. This helps to define a set of categories. Using this technique, data aimed at answering five research questions of this systematic mapping study (cloud model; type of paper; applied methods, techniques and tools; journal name; and types of interoperability problems being investigated) was collected. Table 4 shows an example of data form of one relevant paper. For each of 41 relevant studies, this form was filled in Excel file. In this work, publications are classified into categories in four different dimensions. Dimensions and their corresponding categories are presented in Table 5.

Table 4 Data form for sample paper

Data header	Value
Identifier	P9
Retrieved from	IEEE Xplore
Authors	Javier Miranda, Joaquın Guillen, Juan Manuel Murillo and Carlos Canal
Year	2013
Paper title	Assisting Cloud Service Migration Using Software Adaptation Techniques
URL	http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6676742
Abstract	Paper abstract is copied here
Keywords and key	cloud service migration, software adaptation, model-driven

phrases	
Cloud model	PaaS
Type of paper	conference paper
Applied methods, techniques and tools	proposal of software adaptation techniques with small case study
Journal	-
Investigated interoperability problems	cloud application migration issues

Table 5 Classification dimensions and their categories

Dimension	Categories
cloud model	infrastructure as a service, platform as a service, software as a service, mobile cloud computing
type of paper	book chapter, conference paper, journal article
applied methods, techniques and tools	model/framework, ontology, broker, adapter, unified management/standardized API, use of cloud standards
types of interoperability problems	cloud storage interoperability issues, cloud application/platform interoperability problems, management/API interoperability problems, infrastructure interoperability problems

3.4.1.5 Data extraction and mapping

Finally, the relevant papers are sorted into the established classification schema (151). One paper can be mapped to multiple categories, so the total numbers on sides of the map may not be equivalent. The frequencies of each category show what kind of research was prevalent in the past and then gaps and new research possibilities can be identified. In this work, the results will be shown as answers to research questions stated in the first step of systematic mapping study process.

RQ1: Which model of cloud computing is best investigated in the existing literature regarding cloud interoperability?

The question is answered by cloud model dimension of the chosen classification scheme. There are papers mapped to more than one category, the most common example of combination is IaaS/PaaS, i.e. the solution that addresses the infrastructure and platform

models of cloud computing. The results are shown in Figure 1. The most investigated cloud model is infrastructure as a service.

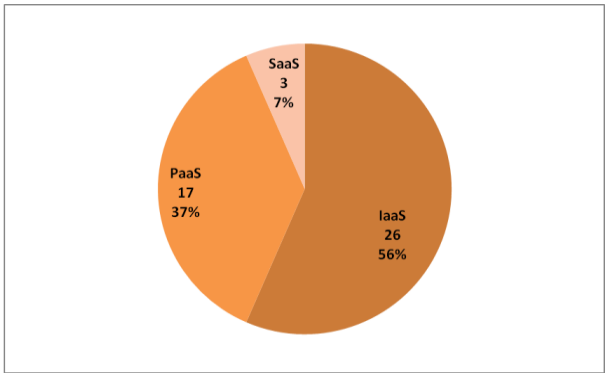

Figure 1 Paper distribution per cloud model

RQ2: What types of papers are published in the cloud interoperability area?

The majority of the published types of papers in the cloud interoperability area are conference papers. The distribution is depicted by Figure 2. These conference papers often present work in progress or a proposal of solution with or without simple prototype.

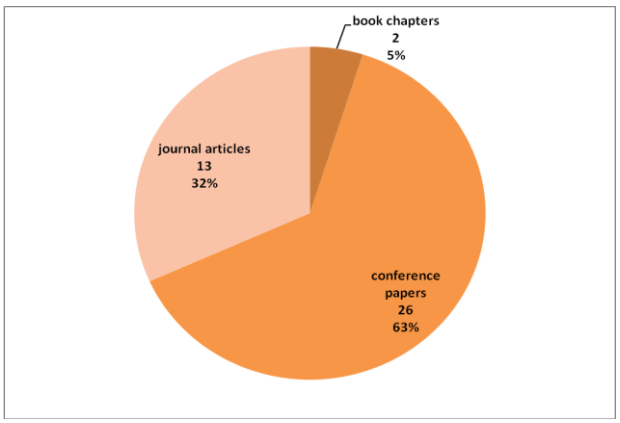

Figure 2 Distribution per paper types

RQ3: What are the most frequently applied methods and techniques to achieve cloud interoperability?

The results are shown in Figure 3. The proposal of model and frameworks is most frequently used, and a runner-up is unified management/standardized API. The majority of the solutions are not mature enough to present the solution to cloud interoperability problems in industrial cases or more realistic scenario rather than simple prototypes.

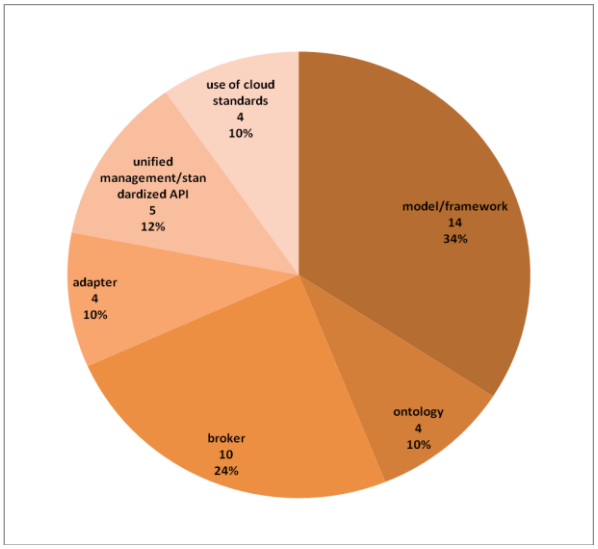

Figure 3 Applied solutions

RQ4: Which journals include papers on cloud interoperability?

Two journals include two papers on cloud interoperability (Scalable Computing and Journal of Systems and Software), whereas one paper on cloud interoperability was published in the following journals: Mondo Digitale, Journal of Integrated Design and Process Science, Journal of Grid Computing, Advances in Parallel Computing, International Journal of High Performance Computing and Networking, Expert Systems with Applications, Journal of Cloud Computing, ACM SIGMETRICS Performance Evaluation Review and Computer Networks. A total of 13 articles relevant to this study were published in journals. Only one journal has the term "cloud computing" in its name, the rest of the journals are on distributed computing, grids, systems and software and expert systems. Interest for cloud interoperability issues exists, and the quality research can be published in journals.

RQ5: Which types of interoperability problems are most investigated?

Distribution per interoperability issues are presented in Figure 4. Some papers investigated more than one category. Interoperability problems connected to infrastructures are the most investigated ones. Cloud storage and API interoperability problems are less investigated, so this work is trying to cover these issues.

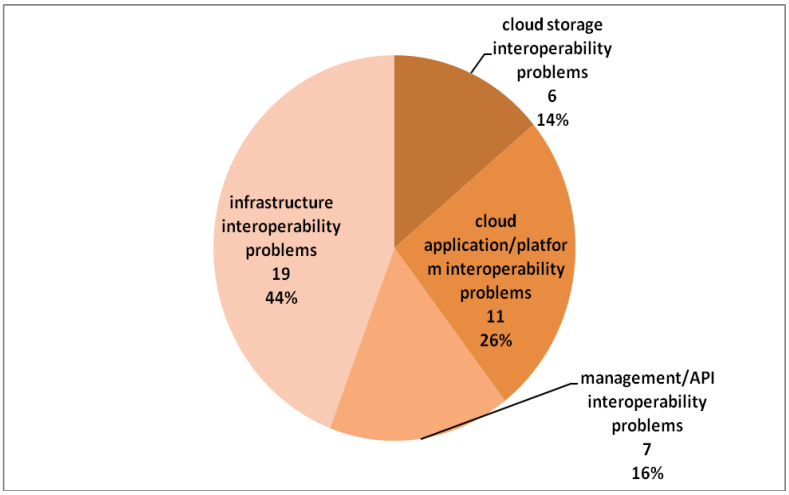

Figure 4 Investigated interoperability problems

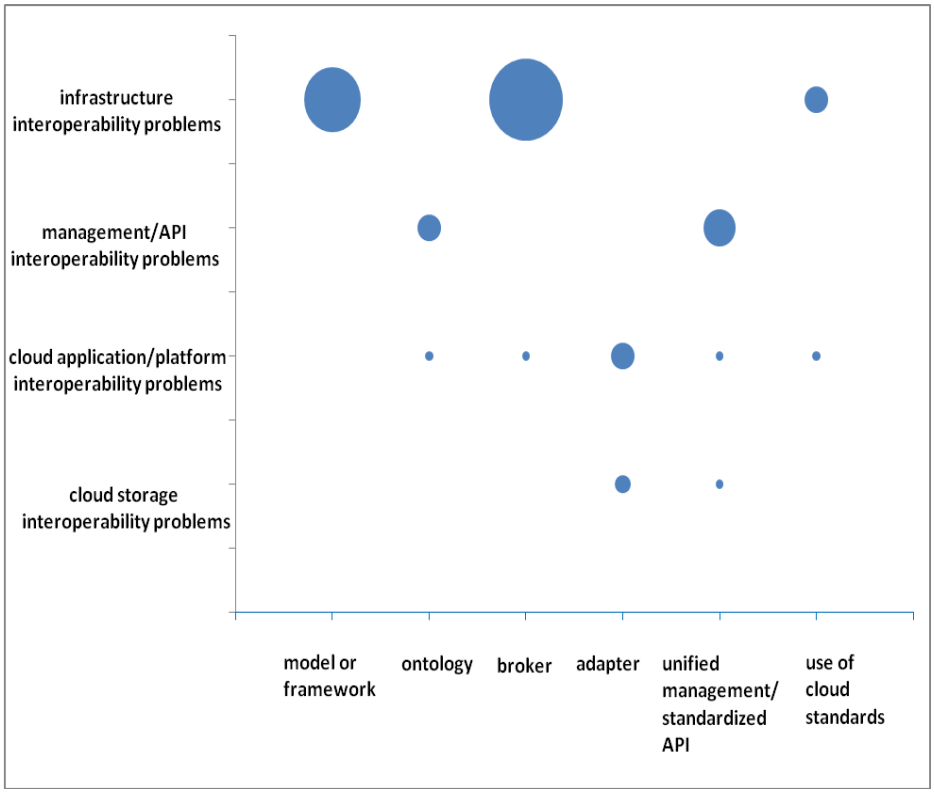

Figure 5 Visualization of a systemic map using a bubble chart

Finally, a systemic map in the form of an Excel bubble chart was created. The connection between types of interoperability problems and proposed solutions is visualized in Figure 5. It shows that there is a lack of papers on cloud application/platform and cloud storage

interoperability problems. The majority of papers deal with infrastructure interoperability problems where they propose some kind of broker architecture or model/framework.

3.4.2 Identified gaps

Platform as a service model was chosen as a focus of this dissertation, because it has significant interoperability problems and it is not investigated, as well as interoperability at infrastructure level. The problem of interoperability among different commercial providers of platform as a service is far from being resolved. The main market players often upgrade their services, and for now they did not adopt cloud standards. Many cloud standards are still not mature enough, and some authors even argue that standardization is reasonable only for infrastructure as a service model. The vendor lock-in is omnipresent in platform as a service offers, and many clients have postponed their investment because they fear the significant costs if they decide to migrate to another provider.

The work in this dissertation is built upon the foundations described in the existing literature. The gaps in the existing literature include lack of data portability among different cloud vendors. There was some work regarding data migration in Cloud4SOA project, but this problem is far from being fully resolved. Furthermore, there is no existing work that solves the problem of data type mappings among different platform as a service offers. Based on their systematic review on cloud migration research, Jamshidi et al. (189) conclude that there is a lack of cloud migration tools from legacy systems to clouds. A detailed ontology of platform as a service and operations from PaaS providers' APIs is not yet available. Also, the methodology for detecting and solving interoperability problems among commercial platform as a service offers is not yet defined. These identified gaps are a motivation for the work presented in this dissertation.

4. USE CASES

The final goal of use cases is to create applications that can evaluate, test and demonstrate an approach to find and solve interoperability problems presented in this dissertation. The use cases were chosen to represent a diverse set of interoperability problems among PaaS offers.

4.1 Preliminaries

4.1.1 Chosen PaaS offers

There are many providers of platform as a service. The following three prominent providers of platform as a service (Microsoft, Google, and Salesforce) will now be examined, as well as their offers and the most important functionalities. The mentioned PaaS offers will be used throughout the use cases and as an important source for terms in the presented ontologies. These offers were chosen in the first place because they are currently among the leading offers in platform as a service market with many current users. For example, in his magic quadrant for enterprise application platform as a service published in January 2014, Gartner (190) 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 storages that can possibly identify more data interoperability problems in comparison to moving data only among the cloud storages of the same types.

Google App Engine supports the following programming languages: Java, Python, and Go. Google App Engine does not support the entire Java EE specification (191): e.g., EJB, JAX-RPC, JDBC, JCA, etc. are all the Java EE APIs that are currently not supported in this platform. Therewithal, Google App Engine can run most of the Python web frameworks. Google's platform as a service runs on Google's custom Linux distributions. It supports the following types of data stores: relational *Google Cloud SQL*, blob storage named *Google Cloud Storage*, and non-relational *High Replication Datastore*. Google App Engine runs on its own web and application servers. App Engine offers thick client and RESTful APIs. Google App Engine provides its own *Memcache* service, full text search, logging service, monitoring service, email, *Google Talk*, *Channel* service, and queuing service.

Microsoft Windows Azure supports SDKs for the following programming languages: C#, Java, PHP, Javascript, and Python. It can be configured to run any framework that can run on Windows Server operating systems. The platform is supported in *Visual Studio* and *Eclipse* development environments. There are three main storage offerings on the Azure platform: *Local Storage*, *Windows Azure Storage*, and *SQL Database*. All applications run on IIS for Windows Azure web and application server. Windows Azure is compatible with *Memcache*, users can configure *Solr/Lucene* search services, and it supports logging services and Azure queuing service. Windows Azure APIs are exposed through HTTP/REST. For some languages, additional libraries are available (as an illustration, Windows Azure Libraries for Java offer Java classes for CRUD operations on Azure Blobs, Tables and Queues, helper classes, and support for logging, authentication and error management).

Force.com is Salesforce.com's platform as a service offer. Force.com development is performed by using Salesforce's tools and a proprietary computer language called Apex (192). The Apex is a pseudo-combination of Java and SQL. SOAP and REST Salesforce's web services APIs can be used for integration with other systems. The biggest benefit of Salesforce's platform as a service offer is time saving for developers (they can easily use the existing common objects, forms and workflows, and they can use only one predefined framework). The biggest obstacle is a significant vendor lock-in, because no other vendor supports Salesforce's programming language, tools and framework. Salesforce also offers many tools for CRM software integration and customization.

4.1.2 PaaS data and application models

First, this dissertation will attempt to determine the differences in data and application models between chosen commercial vendors of platform as a service. For this purpose, the documentation of three chosen PaaS offers was examined in detail. Additionally, the dissertation tries to model simple data structure and sample web application and deploy it to three chosen providers, to see whether some other differences exist among the chosen platforms and available tools of various commercial PaaS vendors. Next paragraphs will describe the main characteristics of each of the chosen PaaS offers (Salesforce, Google App Engine, and Microsoft Azure).

On the Force.com platform, data objects are called custom objects (similar as tables in databases). In Salesforce (193), 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 (Table 6). Every custom object's name on Salesforce must finish with postfix __c (e.g. *Customer__c*).

Table 6 Obligatory standard fields of custom objects (194)

Standard field	*Description*
Created By	*Creator of the record.*
Currency	*Currency of the record.*
Division	*Division to which the custom object record belongs.*
Last Modified By	*User who last modified the record.*
Name	*Identifier for the custom object record.*
Owner	*Owner of the custom object record.*

Custom data objects are created by using a web administration application provided by Salesforce or programmatically by using the Salesforce Metadata API. To build a web user interface, one must use Visualforce and Salesforce's proprietary programming language Apex which is similar to Java. Visualforce (195) is a framework for building custom user interfaces on the Force.com platform. It includes tag-based markup language and implements Model-View-Controller (MVC) design pattern (196) to separate the view and its styling from the data and logic.

Next, the features of Google App Engine were examined. The mentioned PaaS offer has three options for data storage: *App Engine Datastore*, *Google Cloud SQL* and *Google Cloud Storage*. The *App Engine Datastore* (197) 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 (198) 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 Google App Engine plugins for Eclipse and Netbeans, therefore it is possible to develop and deploy Java application on Google App Engine using any of these IDEs.

Finally, the third platform as a service offer (*Microsoft Azure*) was examined. There are three main storage offerings on the Azure platform (199): *Local Storage*, *Windows Azure Storage*, and *SQL Database*. *Local Storage* provides a 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 storages, 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 storages 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, as done here. The supported programming languages are any of the languages of .NET family. C# was chosen here, due to its similarities with Java programming language.

4.1.3 Working with external PaaS data

Next, the options to use external cloud storage in each of the three chosen PaaS offers were examined. Briefly, Google App Engine, Microsoft Azure, and Salesforce do not allow applications deployed on their cloud to directly access external databases (other than their predefined ones that are part of their platform as a service offer or one of their other cloud storage options). The external data can only be accessed using REST or SOAP web services. These web services need to use vendor's remote APIs to access and manipulate the corresponding cloud data. There is no accepted standard for remote APIs among commercial vendors, so each vendor defines its own set of data functions that vary in name, input and

output parameters, and behavior. Another big issue lies in different query languages used by vendors and often required as input parameters in remote APIs for cloud storage manipulation.

The simplest way to connect external databases to a web application stored on Google AppEngine is by exposing them via RESTful web services that will layer upon the database or cloud storage. The commands can be sent and data can be received from external cloud storage by communicating over HTTP using *UrlFetch* class provided by *Google AppEngine*. Using *UrlFetch* call, the author of this dissertation managed to print some data containers from the other two providers (Microsoft Azure and Salesforce) on the page of web application deployed on Google App Engine. The same approach was used in Microsoft Azure. The author connected to REST web services of this project containing web services to access cloud data, here using the Microsoft libraries to do HTTP calls in C# programming language. Data from other chosen PaaS offers in web application published on Azure was successfully fetched. In Salesforce, before users can access external servers using Apex, the remote site must be added to a list of authorized sites in the Salesforce user interface (*Setup* | *Security Controls* | *Remote Site Settings*). HTTP calls in Apex were done using Salesforce's libraries.

4.1.4 Web services support in PaaS offers

Interoperability between two applications hosted on two different clouds can be achieved using principles of service-oriented architecture. The most common way is to design REST or SOAP web services that can work together. All three chosen PaaS providers support the creation of SOAP and REST services by using different methods that are investigated in this subchapter to examine their differences.

A RESTful service as part of the application deployed on Google App Engine can be implemented by means of App Engine Endpoints. App Engine Endpoints (200) are still in experimental release, so the API can be easily changed drastically in the future and it is not covered in SLA. Therefore, a classic Java solution was opted for: Jersey REST framework. Musial-Bright (201) listed all the steps needed to create REST service using the mentioned framework, and to deploy it on Google's infrastructure: Jersey REST framework must be downloaded, Jersey libraries must be added to Java project intended to be deployed on App Engine, and Jersey servlet must be properly configured. When all mentioned preconditions

are satisfied, a simple REST service can be coded in a web application deployed on Google App Engine. REST services can be developed for application published on Microsoft Azure using several techniques depending on the framework used for creating web application. A sample MVC3 web application was created, deployed on Microsoft Azure, and the steps proposed by Schwartzenberger (202) were used to code sample REST service. API Area is created within the application, routes for RESTful URLs were configured, and the controller that represents REST service was written. It is also possible to create your own REST-based web services using Apex (203) and deploy it to Salesforce. A programmer must set up a custom REST Apex endpoint and develop a class that can have different methods for HTTP GET, POST, PUT or DELETE requests. To access REST web service from Saleforce's platform as a service an authentication such as OAuth with the help of cURL tool must be used.

SOAP server and SOAP client can be built and deployed on Google App Engine, but the needed procedure (204) is far more complex than REST alternative. On Microsoft Azure, it is pretty simple to run and deploy SOAP web service (205). The third-party SOAP service can also easily be consumed by an application deployed on Azure using the same code as the one for an on-premise server. Apex, the programming language used for applications stored on Salesforce, supports the ability to expose methods as web services (206) and to invoke an external SOAP service. Both approaches (SOAP and REST) are supported by PaaS vendors, so an application on different clouds can use any of the mentioned approaches to interoperate. There must only be an agreement on the chosen approach and on the message format (e.g., REST service can output the result as a simple text, JSON or XML).

4.2 Use case 1: Migration of data between PaaS providers

4.2.1 Requirements and use case description

In the first use case, data will be migrated between different providers of platform as a service. Two main requirements are defined. First, the user must be able to port all data from one PaaS provider to another. Secondly, the user may move only one chosen data container (for example, table, custom object, or entity) from one PaaS offer to another. Additionally, the migration method should be flexible and use the ontologies and AI planning method described later in this dissertation. This use case will check if the ontology can be used to semantically annotate relevant API operations and whether data type mappings work. Successful execution

of more complex interoperability scenarios cannot be imagined without being able to move data from one vendor to another. First use case will also help to identify new interoperability problems or confirm the known interoperability problems together with the associated indicators. The focus of this dissertation is on using vendors' APIs to find and solve interoperability problems. The majority of vendors' API operations deal with data manipulation and management, so the first use case is also important to learn more about the mentioned APIs in practical problems. The use case is described in Table 7.

Table 7 Description of data migration use case

Use Case ID:	UC-1		
Use Case Name:	Migration of data between PaaS providers		
Created By:	Darko Andročec	**Last Updated By:**	Darko Andročec
Date Created:	August 2013	**Last Revision Date:**	September 2014
Actors:	PaaS user, PaaS provider 1, PaaS provider 2		
Description:	This use case shows how to migrate data from one PaaS provider to another. User can choose to move all data or only one data container (table, entity, or custom object) from source PaaS vendor to target PaaS provider.		
Trigger:	This use case is initiated by the cloud user when he chooses to move data stored in current PaaS offer to another one.		
Preconditions:	1. PaaS user must have the existing data stored on one PaaS offer 2. PaaS user must register to another PaaS offer and be able to put data on it		
Post conditions:	1. Chosen data is moved from one PaaS offer to another		
Normal Flow:	1. PaaS user chooses whether he wants to move all data or specific data container (table, entity, or custom object) 2. PaaS user selects the source and target PaaS offer from the available ones 3. PaaS user initiates the data migration		
Alternative Flows:	-		
Exceptions:	1. If there is a problem with connection to chosen source or target PaaS offers, the exception is raised and error message is shown 2. If the system finds the interoperability problem during data migration, data migration is stopped and found interoperability problem is shown to the PaaS user		
Includes:	No other use case is included by this use case.		

Special Requirements:	Data migration should be flexible and use PaaS ontology and data type mappings defined in them.
Assumptions:	PaaS user understands the English language.
Notes and Issues:	-

4.2.2 Export data structures and data from PaaS providers

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 container. All three chosen PaaS providers enable CSV export, and these files can be used to obtain the names of data containers. First, observe how to do this in Salesforce. There is an option in Salesforce's administration web interface to schedule data exports (*Setup - Administration Setup - Data Management - Data Export*). A system sends the compressed (.zip) file with CSV files of the chosen Salesforce data objects to the user's e-mail. A CSV file stores tabular data with headers in a plain-text format.

Google provides the bulk loader tool in Python SDK that provides functionalities to upload and download data to and from your application's Google App Engine Datastore. However, the data export process is not as simple as expected, and as seen in the other two providers (Salesforce and Microsoft). Python and Google App Engine SDK for Python need to be installed. The configuration of the deployed application also must be changed by adding *RemoteApiServlet* to the configuration file named *web.xml* and redeploy the application. After completing this step, Google's bulk loader tool can be used to access application's data. Next, a connector for every kind of entity needs to be specified in bulkloader.yaml file. Afterwards, the additional commands should be executed to store data into CSV files.

Microsoft offers free *MS SQL Server 2012 Express Management Studio* that contains, inter alia, a tool *Import and Export Data* that can export data from *Azure SQL to CSV* files. When connection string is properly configured, working with Azure SQL database is the same as with local or remote regular *SQL Server* instance. One can directly connect to Azure database and export the associated data.

When parsing CSV files, the basic structures of data objects and their attributes can be obtained, but one cannot get data types and primary or foreign keys (or their synonyms: other

ways to mark identifiers and relationships with other tables/entities/data objects). 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 storages. *DatastoreService* interface (207) from *Google App Engine API* can be used to get data structures, data and data types from Google's storage. It is a schema-less data storage system and its fundamental unit of data is called the entity (207). The entity has key and zero or more mutable properties. Basically, the key of each entity from CSV file was extracted and thereafter API functions were used to retrieve all entities by their keys. The entity's kind, keys, properties and their data types were identified using the aforementioned method.

The similar technique was also used for Salesforce's data. First, a list of important objects and fields was extracted (for each application, Salesforce also automatically stores its own standard objects, so only custom objects that have postfix __c as part of their name were listed). Then describeSObject() Salesforce API call was used to obtain metadata for a given object, and query() calls were executed to retrieve all data from each of the retrieved object. Data structures, data and data types were obtained from Azure SQL database after writing the code for various database operations using JDBC SQL Azure driver. Thereafter, queries were constructed that enable a retrieval of metadata about all tables.

4.2.3 Transformation of data structures and data to ontology

The 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. This architecture is inspired by direct mapping approach (208) proposed by the *RDB2RDF Working Group*. 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. (209) proposed an approach to automatic transformation of relational databases to ontologies. They listed the mapping rules (209) which inspired the rules presented later in this dissertation. Inevitably, some of the semantics captured in a relational database will be lost when transforming the relational database to the ontology (209), the same situation will certainly also happen when dealing with PaaS storages.

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:

A) From Microsoft Azure SQL

1. A table is mapped to an OWL class.
2. A column is mapped to a data type property.
3. A row is mapped to an instance.
4. A primary key is mapped to a value of data property *identifier* in an instance.
5. A foreign key is mapped into object property *hasLinkToObject* with the appropriate domain and range in an instance.

B) From Google App Engine Datastore

1. An entity kind is mapped to an OWL class.
2. A property is mapped to a data type property.
3. An entity is mapped to an instance (OWL individual).
4. An identifier from an entity key is mapped to a value of data property *identifier* in an instance.
5. A relationship between two entities is mapped into object property *hasLinkToObject* with the appropriate domain and range in an instance.

C) From Salesforce

1. An object is mapped to an OWL class.
2. A field is mapped to a data type property.
3. A record is mapped to an instance.
4. An identifier of an object (recognized as a field of Salesforce's ***ID*** data type) is mapped to a value of data property *identifier* in an instance.
5. A relationship between two objects (recognized as a field of Salesforce's ***reference*** data type) is mapped into object property *hasLinkToObject* with the appropriate domain and range in an instance.

The mappings in other direction (from OWL ontology to cloud storage) could also be defined, so representing these data models by means of OWL ontology can provide a common layer for information exchange:

A) To Microsoft Azure SQL

1. An OWL class is mapped to a table.

2. A data type property is mapped to a column.

3. An instance is mapped to a row.

4. A value of data property *identifier* in an instance is mapped to a primary key.

5. Object property *hasLinkToObject* with the appropriate domain and range in an instance is mapped into foreign key.

B) To Google App Engine Datastore

1. An OWL class is mapped to an entity kind.

2. A data type property is mapped to a property.

3. An instance is mapped to an entity.

4. A value of data property *identifier* in an instance is mapped to an identifier from an entity key.

5. Object property *hasLinkToObject* with the appropriate domain and range in an instance is mapped into a relationship between two entities.

C) To Salesforce

1. An OWL class is mapped to an object.

2. A data type property is mapped to a field.

3. An instance is mapped to a record.

4. When inserting new record, Salesforce automatically assigns its identifier that is unique within the organization's data. So, the identifier cannot be manually set. If it is important to save the identifier when migrating from different storage, this value (of data property *identifier*) can be stored in a new custom field such as *identifier__c*.

5. Object property *hasLinkToObject* with the appropriate domain and range in an instance is mapped into a relationship between two objects (a field of Salesforce's ***reference*** data type).

The web services for reading and writing OWL data ontologies were created using the above specified transformation rules and the Apache Jena framework (210) 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.

4.2.4 Data type mappings

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 storages - Google App Engine Datastore (211), Microsoft Azure SQL Database (212), Salesforce (213) - are mapped to XSD (because OWL uses Schema Data Types - (214) and (215)), more specifically to an OWL data property's range of data model

ontology. The summary of the mentioned mappings is shown in Table 8. The mapping in the other direction (from OWL data types to data type of platform as a service storage) is also specified below (see
Table 9).

In these two tables representing mappings, OWL (XSD) data types are chosen as a baseline system.

Table 8 Mappings from PaaS storages' to OWL data types

Azure SQL	*Salesforce*	*GAE DataStore*	*OWL (XSD)*
char, varchar, text, nchar, nvarchar, ntext	string, combobox, email, encryptedstring, multipicklist, phone, textarea, URL	java.lang.String, com.google.appengine.api.datastore.Text, com.google.appengine.api.datastore.GeoPt, com.google.appengine.api.datastore.PostalAddress, com.google.appengine.api.datastore.PhoneNumber, com.google.appengine.api.datastore.Email, com.google.appengine.api.users.User, com.google.appengine.api.datastore.IMHandle, com.google.appengine.api.datastore.Link, com.google.appengine.api.datastore.Category, com.google.appengine.api.datastore.Key, com.google.appengine.api.datastore.EmbeddedEntity	**xsd:string**
bit	boolean	boolean, java.lang.Boolean	**xsd:boolean**
decimal, money, numeric, smallmoney			**xsd:decimal**
real		float, java.lang.Float	**xsd:float**
float	double, currency, percent	double, java.lang.Double	**xsd:double**
		java.lang.Integer, com.google.appengine.api.datastore.Rating	**xsd:integer**
bigint		long, java.lang.Long	**xsd:long**
int	int	int	**xsd:int**
smallint		short, java.lang.Short	**xsd:short**
	byte		**xsd:byte**
tinyint			**xsd:unsignedByte**
datetime, datetime2, datetimeoffset, smalldatetime	dateTime	java.util.Date	**xsd:dateTime**
time	time		**xsd:time**
date	date		**xsd:date**
		com.google.appengine.api.datastore.ShortBlob, com.google.appengine.api.datastore.Blob, com.google.appengine.api.blobstore.BlobKey	**xsd:hexBinary**
binary, varbinary, image, timestamp	base64		**xsd:base64Binary**
			xsd:anyURI

geography, geometry, cursor, hierarchyid, sql_variant, table, uniqueidentifier, xml	anyType, calculated, DataCategoryGroupReference, ID, masterrecord, picklist, reference	**- (unsupported mapping to OWL)**

Table 9 Mappings from OWL to PaaS storages' data types

OWL (XSD)	*Azure SQL*	*Salesforce*	*GAE DataStore*
xsd:string	varchar	string	java.lang.String
xsd:normalizedString	varchar	string	java.lang.String
xsd:token	varchar	string	java.lang.String
xsd:language	varchar	string	java.lang.String
xsd:NMTOKEN	varchar	string	java.lang.String
xsd:Name	varchar	string	java.lang.String
xsd:NCName	varchar	string	java.lang.String
xsd:boolean	bit	boolean	java.lang.Boolean
xsd:decimal	decimal	double	java.lang.Double
xsd:float	real	double	java.lang.Double
xsd:double	float	double	java.lang.Double
xsd:integer	int	int	java.lang.Integer
xsd:positiveInteger	int	int	java.lang.Integer
xsd:nonPositiveInteger	int	int	java.lang.Integer
xsd:negativeInteger	int	int	java.lang.Integer
xsd:nonNegativeInteger	int	int	java.lang.Integer
xsd:long	bigint	int	java.lang.Long
xsd:int	int	int	java.lang.Integer
xsd:short	smallint	int	java.lang.Short
xsd:byte	tinyint	byte	java.lang.Short
xsd:unsignedLong	bigint	int	java.lang.Long
xsd:unsignedInt	int	int	java.lang.Integer
xsd:unsignedShort	smallint	int	java.lang.Short
xsd:unsignedByte	tinyint	byte	java.lang.Short
xsd:dateTime	datetime	dateTime	java.util.Date
xsd:time	time	time	java.util.Date
xsd:date	date	date	java.util.Date
xsd:gYearMonth	varchar	string	java.lang.String
xsd:gYear	varchar	string	java.lang.String
xsd:gMonthDay	varchar	string	java.lang.String
xsd:gDay	varchar	string	java.lang.String
xsd:gMonth	varchar	string	java.lang.String
xsd:hexBinary	varbinary	-	com.google.appengine.api.datastore.Blob
xsd:base64Binary	varbinary	base64	-
xsd:anyURI	varchar	string	java.lang.String

Data type mappings were implemented by means of PaaS ontology that will be elaborated in detail in Chapter 5. Two classes deal with data types mappings between different PaaS storages: *DataType* and *DataTypeMapper*. 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 from the tables above 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. For instance, SalesforceToOwl_2 is an instance of the *DataTypeMapper hasSource SalesforceBoolean* and *hasDestination XsdBoolean*, so it shows that the Salesforce *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. For now, *DataTypeMapper* has approximately 150 instances (mappings). The mapping instances are created based on the mappings presented in Table 8 and Table 9. 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 a 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 mapping of data types supported by different PaaS providers. Table 8 and Table 9 show that some data types have unsupported mappings (for example, Salesforce's anyType has not mappings to any OWL type). In these cases, data migration will stop and error will be shown to the user suggesting that there is interoperability problem connected to data types of different PaaS storages.

4.2.5 Architecture for data migration among PaaS providers

User starts data migration using client web application (Figure 6). The CSV files are parsed and the 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 data ontology, perform mappings, create data and data structures and move them into target PaaS storage. If 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 other remaining data during migration.

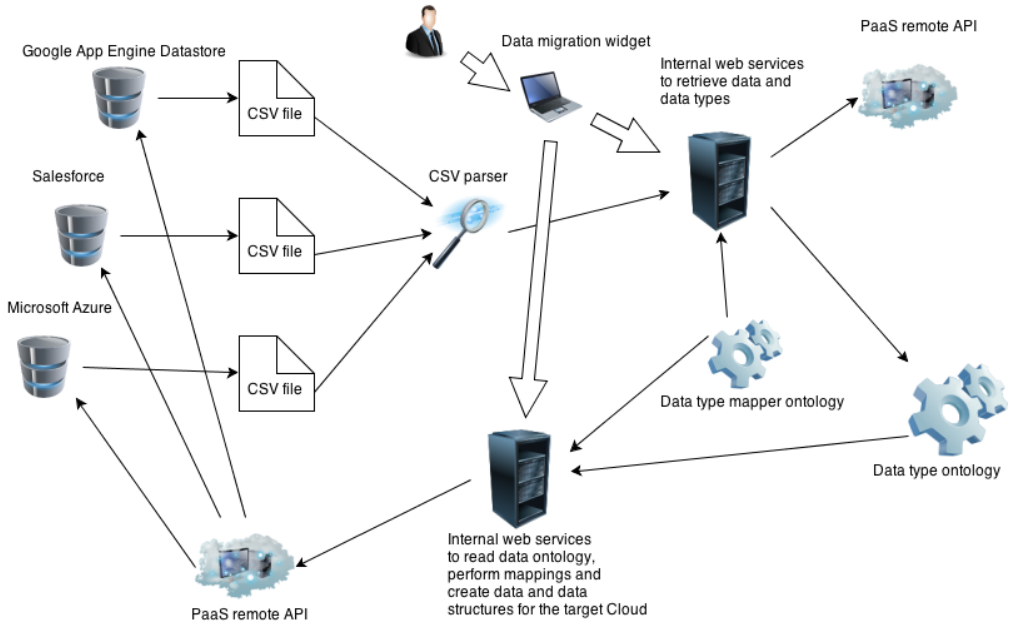

Figure 6 Architecture for data migration between PaaS providers

4.2.6 Validation and assessment

The validation of the first use case and 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, an open-source content management system (CMS) Vosao (216) was chosen. It is coded in Java and it aims to be deployed on Google App Engine. On Vosao web site there is a list of 16 active web sites that use this software as their content management system. The author of this dissertation downloaded the source code of Vosao CMS from December 2013, opened and compiled the source and successfully deployed it on Google App Engine using his Google account. The installation of Vosao CMS is publicly available on http://quiet-surface-517.appspot.com/.

Vosao CMS uses Google App Engine Datastore and initially consists of 19 entities. There will be an attempt to migrate its data to other two chosen PaaS offers (Microsoft Azure and Salesforce) to check whether the methodology and developed prototype migration tool works smoothly when there is a real cloud application with more data containers and data objects than shown in the previously considered simple examples. Furthermore, Vosao CMS default installation already contains some pages with contents, so data store is filled with the initial data.

According to this migration approach, first Vosao's data need to be exported into CSV file. Google's bulk loader tool will be used in its Python SDK to accomplish the mentioned task. To be able to remotely access the data, web.xml configuration file must be changed to include *RemoteApiServlet*. Then, Google's bulk loader tool can be used to export data. Next, a connector for every kind of entities needs to be specified in bulkloader.yaml file to store data into CSV files.

The underlying data and data structures of Vosao are represented by the unified data model ontology using transformations/mapping rules presented in Chapter 4.2.3 of this work. For each Google's data store entity, attributes, identifiers and number of instances were checked and the conclusion was that the transformation was successful. The data ontology contains all the entities and data from Vosao's data stored in Google Datastore.

Next, the developed client web application, data type mappings logic, web services and AI planning will be used to actually migrate data to Salesforce and Microsoft Azure. Using this tool ontology, and AI planning techniques, Vosao's data from Google App Engine platform was successfully migrated to Salesforce platform. In Salesforce, data objects are custom objects (with suffix __c) and their attributes are custom fields. Custom objects and custom fields can be created using Salesforce Metadata API. Furthermore, Vosao's data from Google App Engine platform was successfully migrated to Microsoft Azure platform and its Azure SQL Server database. In it, data objects are tables and their attributes are tables' columns.

The verification of migration of Vosao's data was done manually in Excel. The example for the used form for Salesforce destination is shown in Table 10. All data containers, their

names, names and number of their attributes and number of records were listed there. Additionally, all the data for randomly chosen entities was also checked.

Table 10 Manual evaluation of Vosao's data migration to Salesforce

Data entity	Number of attributes	Attributes	Number of data records
ConfigEntity__c	25	attributesJSON__c, commentsEmail__c, commentsTemplate__c, createDate__c, createUserEmail__c, defaultLanguage__c, defaultTimezone__c, editExt__c, enableCkeditor__c, enablePicasa__c, enableRecaptcha__c, googleAnalyticsId__c, modDate__c, modUserEmail__c, picasaPassword__c, picasaUser__c, recaptchaPrivateKey__c, recaptchaPublicKey__c, sessionKey__c, site404Url__c, siteDomain__c, siteEmail__c, siteUserLoginUrl__c, version__c, identifier__c	1
ContentEntity__c	9	content__c, createDate__c, createUserEmail__c, languageCode__c, modDate__c, modUserEmail__c, parentClass__c , parentKey__c, identifier__c	23
ContentPermissionEntity__c	10	allLanguages__c, createDate__c, createUserEmail__c, groupId__c, languages__c, modDate__c, modUserEmail__c, permission__c, url__c, identifier__c	1
FieldEntity__c	17	createDate__c, createUserEmail__c, defaultValue__c, fieldType__c, formId__c, height__c, index__c, mandatory__c, modDate__c, modUserEmail__c, name__c, regex__c, regexMessage__c, title__c, values__c, width__c, identifier__c	3
FileChunkEntity__c	8	content__c, createDate__c, createUserEmail__c, field__c, index__c, modDate_c, modUserEmail__c, identifier__c	43
FileEntity__c	11	createDate__c, createUserEmail__c, filename__c, folderId__c, lastModifiedTime__c, mimeType__c, modDate__c, modUserEmail__c, size__c, title__c, identifier__c	43
FolderEntity__c	8	createDate__c, createUserEmail__c, modDate__c, modUserEmail__c, name__c, parentId__c, title__c, identifier__c	15
FolderPermissionEntity__c	8	createDate__c, createUserEmail__c, folderId__c, groupId__c, modDate__c, modUserEmail__c, permission__c, identifier__c	3
FormConfigEntity__c	7	createDate__c, createUserEmail__c, formTemplate__c, letterTemplate__c, modDate__c, modUserEmail__c, identifier__c	1
FormEntity__c	14	createDate__c, createUserEmail__c, email__c, enableCaptcha__c, enableSave__c, letterSubject__c, modDate__c, modUserEmail__c, name__c, resetButtonTitle__c, sendButtonTitle__c, showResetButton__c, title__c, identifier__c	1
GroupEntity__c	6	createDate__c, createUserEmail__c, modDate__c, modUserEmail__c, name__c, identifier__c	1
LanguageEntity__c	7	code__c, createDate__c, createUserEmail__c, modDate__c, modUserEmail__c, title__c, identifier__c	1
PageEntity__c	31	attributes__c, cached__c, commentsEnabled__c, contentType__c, createDate__c, createUserEmail__c, description__c, enableCkeditor__c, endPublishDate__c, friendlyURL__c, headHtml__c, keywords__c, modDate__c, modUserEmail__c, pageType__c, parentUrl__c, publishDate__c, restful__c, searchable__c, skipPostProcessing__c, sortIndex__c, state__c, structureId__c, structureTemplateId__c, template__c, title__c, velocityProcessing__c, version__c, versionTitle__c, wikiProcessing__c, identifier__c	26
PageTagEntity__c	7	createDate__c, createUserEmail__c, modDate__c, modUserEmail__c,	13

		pageURL__c, tags__c, identifier__c	
StructureEntity__c	7	content__c, createDate__c, createUserEmail__c, modDate__c, modUserEmail__c, title__c, identifier__c	1
StructureTemplateEntity__c	11	content__c, createDate__c, createUserEmail__c, headContent__c, modDate__c, modUserEmail__c, name__c, structureId__c, title__c, type__c, identifier__c	2
TagEntity__c	9	createDate__c, createUserEmail__c, modDate__c, modUserEmail__c, name__c, pages__c, parent__c, title__c, identifier__c	5
TemplateEntity__c	8	content__c, createDate__c, createUserEmail__c , modDate__c, modUserEmail__c, title__c, url__c, identifier__c	2
UserEntity__c	12	createDate__c, createUserEmail__c, disabled__c, email__c, forgotPasswordKey__c, modDate__c, modUserEmail__c, name__c, password__c, role__c, timezone__c, identifier__c	1

First, the data migrated to Salesforce was checked. Some errors were initially found and bugs in the programs were fixed until migration was properly done. In Salesforce, custom object 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 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. In the end, some entities were randomly chosen and all the data and mappings of data types were checked.

Next, the data migrated from Google App Engine to Microsoft Azure was checked in the similar way. Using the same manual technique and Excel as in the migration to Salesforce, 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 and methodology to a new instance of Google App Engine (datafromazure.appspot.com and datafromsalesforce.appspot.com) and then this data was compared to original Vosao's data in its original instance of Google App Engine and its underlying datastore. Number and names of entities, properties and identifiers were manually checked. When migrating from Salesforce, the only difference in data is 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.

4.2.7 Lessons learned from use case 1

Use case 1 illustrates that there are many data migration/interoperability problems among PaaS providers. The first identified problem is the difference between data storage models of various commercial providers of platform as a service. As an illustration, it is difficult or even impossible to move data without losing important information from an SQL model of one provider to a NoSQL model of another platform as a service provider. Even if the same models were chosen (e.g. SQL) in two various offers, these models will still have significant differences due to provider's design and used technology. For example, each provider supports their own set of data types. Data types differ in name, value space, permitted range of values, precision of data etc. Some offers also have predefined standard objects or tables, e.g. Salesforce lists standard objects in its documentation (some object/table names are reserved) and it also adds some standard fields to any new custom object (object created by user). Data import or export is often complicated. Most providers offer only basic CSV or XML exports (list of columns and row data), so users cannot determine data types, identifiers, possible relationships between tables (e.g. foreign keys) etc. Users must use remote APIs of cloud providers to get that information. APIs are not standardized, so users need to cope with different functions, input and output parameters and different means to access remote API functionalities by using libraries for programming languages and/or SOAP or REST web services. Various platform as a service providers also use their own versions of data query languages. For instance, Salesforce demands that the Salesforce Object Query Language (SOQL) and Salesforce Object Search Language (SOSL) be used to query data in its PaaS storage. Google Query Language (GQL) is a language for retrieving entities or keys from the Google App Engine datastore, and its syntax is similar to that of SQL. SQL Azure uses T-SQL as its query language.

To minimize the possible data interoperability problems in PaaS domain, users should carefully choose PaaS offer, underlying PaaS storages, and features. It is best to avoid using vendors' specific features that are not supported in any other PaaS 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 PaaS 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.

4.3 Use case 2: Add existing user to another PaaS

4.3.1 Requirements and use case 2 description

In the second use case, current user information from one PaaS offer will be added to the application hosted on another PaaS offer. The main aim is to investigate interoperability problems on service layer when using APIs from different providers. To solve possible interoperability issues like the one described by Nagarajan et al. (77), the ontology driven data mediation will be used and tested in this use case. Web operations and their inputs/outputs will be semantically annotated, and SAWSDL and XSLT will be used to define service type mappings. The use case is described in Table 11. The flow of API operation with operation names as defined in the ontology is shown in Figure 7.

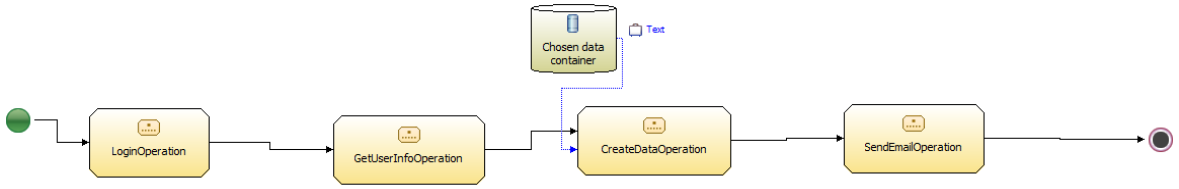

Figure 7 API operations executed in use case 2

Table 11 Description of use case 2

Use Case ID:	UC-2		
Use Case Name:	Add existing user to another PaaS providers		
Created By:	Darko Andročec	**Last Updated By:**	Darko Andročec
Date Created:	September 2014	**Last Revision Date:**	September 2014
Actors:	PaaS application administrator, PaaS provider 1, PaaS provider 2		

Description:	This use case shows how to add user from one PaaS offer to an application hosted on another PaaS. PaaS administrator specifies data container of target PaaS where user information will be stored. Adequate schema mapping files should also be created. In the end, an e-mail is sent to PaaS application administrator that new user is added.
Trigger:	This use case is initiated by the PaaS application administrator when he decides that he wants to add the existing user information (from other PaaS offer) to PaaS application that he manages.
Preconditions:	1. User that is required to migrate must be logged-in on source PaaS offer 2. PaaS application administrator must be able to put data into data container with user information on target PaaS offer
Post conditions:	1. The existing user from source PaaS offer is added to the application hosted on target PaaS offer, an e-mail is sent to PaaS application administrator
Normal Flow:	1. PaaS application administrator selects the source and the connections of target PaaS offer, and specifies the name of data container where user information is stored for target application 2. PaaS application administrator initiates the user migration 3. Input/output mappings are performed, appropriate web services are called, user is added for application stored on target PaaS offer, and email on new user is sent to administrator
Alternative Flows:	-
Exceptions:	1. If there is a problem with connection to chosen source or target PaaS offers, the exception is raised and error message is shown 2. If the system finds the interoperability problem during the planning phase or service execution, the action is stopped and found interoperability problems are shown in user interface
Includes:	No other use case is included by this use case.
Special Requirements:	This use case should validate API service level interoperability, using ontology based data mediation and lifting and lowering schema as defined in SAWSDL.
Assumptions:	PaaS user understands the English language.
Notes and Issues:	-

4.3.2 Ontology driven data mediation

To solve message-level heterogeneities of PaaS APIs, the author will use the approach similar to the one presented by Nagarajan et al. (122). Their approach was used to enable automatic or semi-automatic composition of semantic web services and it can resolve data heterogeneities between different web services by means of the ontology. In this work, semantic web services are used to abstract PaaS providers' APIs. The operations are semantically annotated using cross-PaaS concepts from the ontology of PaaS resources, remote operations, and data types defined in Chapter 5.2. These cross-PaaS concepts are actually subclasses of the *ComplexServiceDataType* and *SimpleServiceDataType*. For example, *UserInfoType* is a cross-PaaS concept for complex type giving the basic information about currently logged user, and it consists of three data properties: *userInfoEmail*, *userInfoName*, and *userInfoUserName*.

The details of semantic annotations, service data type mappings, AI problem generation and concrete service composition and inputs/outputs transformations are presented in Chapter 6. The most relevant concepts will be explained here. Besides semantically anotating operation and input/output types, the needed transformations between data types should be provided. For this purpose, standard mechanism provided by SAWSDL, lifting and lowering schema mappings will be used. The SAWSDL's *liftingSchemaMapping* specifies a mapping and/or transformation from XML element in XSD schema of service description to the concept from an ontology (77). On the other side, *loweringSchemaMapping* specifies a mapping and/or transformation in the reverse direction (122). SAWSDL enables the use of any ontology and mapping language, and the most used ones were chosen: OWL for the ontology and XSLT for XML transformations (217). When there is a need for new transformation, users need to manually construct valid XSLT files and add lifting or lowering schema mappings to these files. Observe one of the examples. Service annotated with *GetUserInformation* has output *UserInfoType*. This operation provides basic information on the user that is logged in specific PaaS offer, and its output is used by other two operations (*CreateDataOperation* and *SendEmailOperation*) to create data object in other PaaS offer, and send an e-mail to application administrator that new user is added. *CreateDataOperation* has input of

NoSqlDataObjectType, and *SendEmailOperation* has input of *EmailMessageType* (see Figure 8). Minimally three mappings should be defined, from output of the *GetUserInformation* to the concept in the ontology, and from the concept in the ontology to both inputs of *CreateDataOperation* and *SendEmailOperation.*

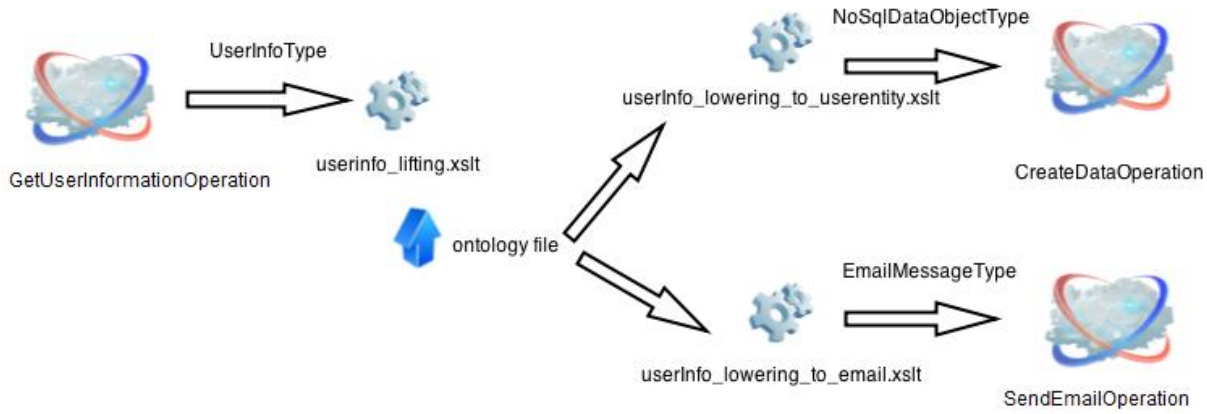

Figure 8 Example of service input/output transformations

The next problem is the actual dynamic execution of these web services and on the fly transformations of inputs/outputs. Apache CXF (218) framework was used in which all message transformations are done by means of interceptor classes. Custom interceptor was implemented to adequately transform input and output message based on XSLT files obtained from AI planning files, and to use CXF features to dynamically call web services. XSLT processor Xalan was used to parse XSLT files, and standard Java classes for XML parsing were used to parse transformed XML files. The procedure is described in more detail in Chapter 6.

4.3.3 Validation

Testing and validation was performed on a case where current Salesforce's user is added to data container in Vosao CMS application mentioned in use case 1. The name of Google's entity where Vosao's user data is stored is called UserEntity. When choosing the action of use case 2, the client web application enables a user to write to which data container in target PaaS the basic user information will be stored. Salesforce's and Google App Engine's web services and their inputs and outputs were semantically annotated, and lifting and lowering schema mappings were created and incorporated in adequate SAWSDL files. In the end, an AI planner has successfully found the plan, CXF interceptor class and service data mapping

and transformation were successfully finished, and web services defined in composition were successfully invoked. UserEntity was successfully created with the appropriate properties for username, name and email filled-in. Finally, the email message was sent to test e-mail representing mail of application administrator. Also, SAWSDL files (semantic annotations) and XSLT files were changed, to test whether relevant interoperability problem was listed.

4.3.4 Lessons learned from use case 2

Nagarajan et al. (77) claim that structural and semantic differences of messages exchanged by web services represent the most complex interoperability challenges regarding the interoperability on a service level. The same is true for this case. API operations of different PaaS vendors have different types, most often these types are complex (they consist of more simple types and/or other complex types). To achieve interoperability, mappings and transformations between inputs and outputs need to be defined. SAWSDL provides its lifting and lowering schema mapping features to map XML elements to the ontology and back. Use of cross-PaaS concept for data types in the ontology simplifies mappings, and enables the creation of new mappings and possible transformations, when new PaaS offer is used, or when specific API is changed. This is a more flexible approach than direct mapping and transformation approach used in web service composition languages like BPEL. The most critical part of this approach is the requirement for user/administrator to create valid and meaningful mappings and transformations.

5. DEVELOPMENT AND EVALUATION OF ONTOLOGIES

5.1 Selected ontology development methodology, tool and language

For the purpose of this research, the Ontology Development 101 (33) methodology was selected. The various ontology development methodologies are briefly presented in Chapter 2.5.4. 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. Noy and McGuinness (33) 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 (33):

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, Noy and McGuinness (33) 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.

5.2 Ontology of PaaS resources, remote operations, and data types

5.2.1 Domain and scope of the ontology

In the first step of Ontology Development 101 (33) guide, the domain and scope of the model should be limited. 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.

5.2.2 Reusing the existing ontologies

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 Chapter 2.4.4. There is no ontology that is focused on remote operations providers of commercial platform as a service and data type mappings among them, but some concepts from mOSAIC ontology (132) and Deng et al. (129) were used as important terms for development of this ontology. These important terms are listed in the next step (Table 12).

5.2.3 Important terms for the ontology

A list of all the relevant terms was identified in this step, without worrying about the overlap between the concepts or considering whether the concepts were OWL classes or properties. 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 (109) metamodel from REMICS project, OASIS Reference Ontology for Semantic Service Oriented Architecture (219), relevant related works from literature (127), 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 12. The list of terms was incrementally updated during the whole research.

Table 12 List of identified terms for PaaS ontology

Source	Important terms
Deng et al. (129)	service offering, composite offering
mOSAIC ontology - Moscato et al. (132)	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 (44)	service, variable, parameter, input, output, result, precondition
WSMO (45)	web service, precondition, assumption, postcondition, effect
OCCI (20)	entity, resource, kind, action
TOSCA (24)	properties, capabilities, interfaces, operation, requirements
CDMI (25)	container, data object, queue object
Salesforce's APIs - (213)**,** (220) **- 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 - (211)**,** (221) **– 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 (222) **–**	set table service properties, get table service properties,

list of remote operations	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 (109)	PaaS resource, communication resource

5.2.4 Classes and their hierarchy

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 (31). 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 (see Figure 9). All classes are systematically specified in Table 13.

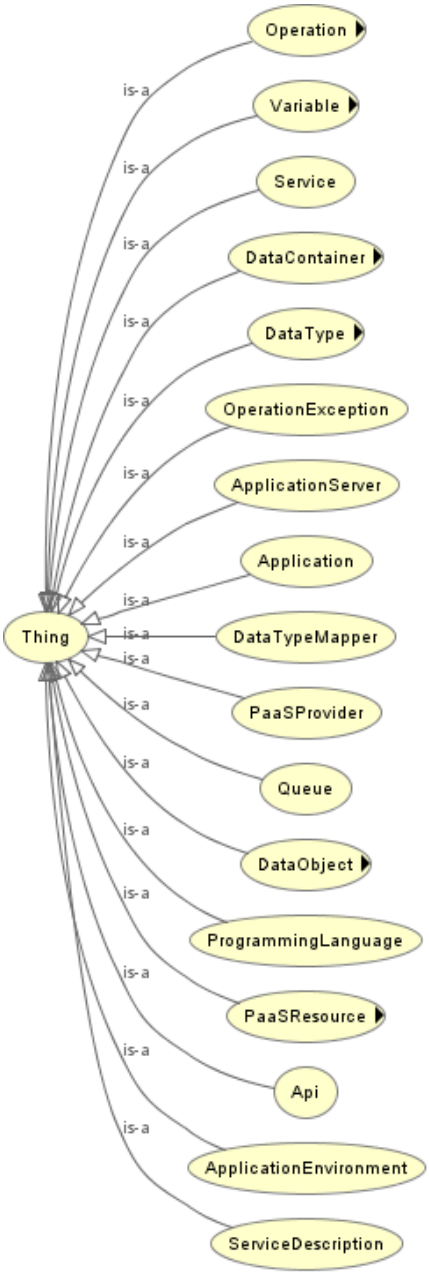

Figure 9 Top level classes of PaaS ontology

Table 13 List of all classes of the PaaS ontology

Class	Super class	Description
Api	Thing	It represents vendors' Application Programming Interfaces (APIs).
Application	Thing	It contains all instances of applications that are deployed to a PaaS offer and run in the *ApplicationEnvironment*.
ApplicationEnvironment	Thing	PaaS application environment such as Google App Engine Java runtime environment.

ApplicationServer	Thing	Application server dedicated to efficient execution of cloud applications on vendor's servers.
DataContainer	Thing	This class is an abstraction of containers of data objects, e.g. tables, entities, objects, files directories.
DirectoryContainer	DataContainer	A data container in the form of a directory.
EntityContainer	DataContainer	A data container for key-value cloud storage.
ObjectContainer	DataContainer	A data container for object of object-like cloud storage.
TableContainer	DataContainer	A container for tables in relational-database cloud storage.
DataObject	Thing	This class includes instances of data objects of various storage options such as NoSQL, relational database, object database and cloud file systems.
EntityProperty	DataObject	An instance of data objects in key-value cloud datastores.
ObjectRecord	DataObject	It describes a particular occurrence of an object.
TableRow	DataObject	A row of a table in relational cloud storage.
DataType	Thing	Data types in cloud storages or cloud services.
CloudStorageDataType	DataType	Data types in cloud storages.
AzureDataType	CloudStorageDataType	Microsoft Azure's data types.
GoogleDataType	CloudStorageDataType	Data types in Google App Engine.
OWLDataType	CloudStorageDataType	Standard OWL (XML schema) data types.
SalesforceDataType	CloudStorageDataType	Data types in Salesforce's cloud storage.
ServiceDataType	DataType	Data types of inputs and outputs of remote APIs in form of web services.
ComplexServiceDataType	ServiceDataType	Cross-PaaS complex types of inputs and outputs of remote APIs in the form of web services. They consist of simple or other complex types.
EmailMessageType	ComplexServiceDataType	Cross-PaaS complex type for email message that contains the most important fields of email header (from, to, subject) and email text.
NoSqlDataObjectType	ComplexServiceDataType	Cross-PaaS complex type that represents NoSQL data object, for example, *Entity* in Google App Engine Datastore.
NoSqlKeyType	ComplexServiceDataType	Cross-PaaS complex type representing a key of NoSQL data object, for example, *Key* in Google App Engine Datastore.
NoSqlPropertyType	ComplexServiceDataType	Cross-PaaS complex type for a property of NoSQL data objects, for example, *Property* in Google App Engine Datastore.
UserInfoType	ComplexServiceDataType	Cross-PaaS concept for complex type giving basic information about currently logged user.
SimpleServiceDataType	ServiceDataType	Cross-PaaS concepts for simple service input/output data types of PaaS vendors.
CurrencyType	SimpleServiceDataType	Cross-PaaS simple service data type for currency.
EmailAddressType	SimpleServiceDataType	Cross-PaaS simple service data type for email address.

EncryptedStringType	SimpleServiceDataType	Cross-PaaS simple service data type for encrypted text fields (strings).
GeographicLocationType	SimpleServiceDataType	Cross-PaaS simple service data type for geographic locations.
PercentType	SimpleServiceDataType	Cross-PaaS simple service data type for percentage values.
PostalAddressType	SimpleServiceDataType	Cross-PaaS simple service data type for postal addresses.
RatingType	SimpleServiceDataType	Cross-PaaS simple service data type for a user-provided integer rating.
TelephoneNumberType	SimpleServiceDataType	Cross-PaaS simple service data type for telephone numbers.
UrlLinkType	SimpleServiceDataType	Cross-PaaS simple service data type for URL links.
DataTypeMapper	Thing	Its instances are used for data type mappings between different storages of different PaaS vendors.
Operation	Thing	It represents all instances of remote operations defined in various vendors' APIs.
MonitoringOperation	Operation	Operations for monitoring PaaS resources.
ResourceUsageOperation	MonitoringOperation	It returns information on PaaS resource usage.
BillingOperation	MonitoringOperation	Operation that returns current cost and other billing information.
UpdateAlertRuleOperation	MonitoringOperation	It updates the specified alert rule.
ListAlertRulesOperation	MonitoringOperation	It retrieves information about all of the alert rules.
GetAlertRuleOperation	MonitoringOperation	It retrieves information about the specified alert rule.
DeleteAlertRuleOperation	MonitoringOperation	It deletes the specified alert rule.
CreateAlertRuleOperation	MonitoringOperation	It creates a new alert rule.
AuthenticationOperation	Operation	Operations for authentication, access control and security
AddServiceCertificateOperation	AuthenticationOperation	It adds a certificate to a cloud service.
DeleteServiceCertificateOperation	AuthenticationOperation	It deletes an existing certificate of a cloud service.
ChangePasswordOperation	AuthenticationOperation	It is used to change the password.
GetDataAccessInformationOperation	AuthenticationOperation	It gets information about access policies for specified data.
GetPublicCertificatesForAppOperation	AuthenticationOperation	It returns a list of public certificates.
GetUserInfoOperation	AuthenticationOperation	It gets information about the specified or current user.
LoginOperation	AuthenticationOperation	It logs in to be able to use PaaS service.
LogoutOperation	AuthenticationOperation	It logs out from PaaS offer.
SetDataAccessInformationOperation	AuthenticationOperation	It sets information about access policies for specified data.
SetPasswordOperation	AuthenticationOperation	It sets the password to the specified value.
CustomCompositeOperation	Operation	These operations are implemented by external developers and are not part of vendors' APIs, they are built upon remote APIs, and compose multiple API operations to perform some composite task such as creation of data model described in the use case of data migration presented earlier in this dissertation.

CreateDataElementsFromOntologyOperation	CustomCompositeOperation	It creates data elements (data objects and containers) from data model ontology.
CreateDataModelOntologyOperation	CustomCompositeOperation	It creates data model ontology for migrating data between different providers.
FindKeyOperation	CustomCompositeOperation	It finds all keys for all data containers in specific cloud data storage.
DataOperation	Operation	Operations for cloud data manipulation and management
BlobDataOperation	DataOperation	Operations for manipulation and management of binary data (blobs)
GetBlobCreationDateOperation	BlobDataOperation	It returns the time and date the blob was uploaded.
GetBlobFilenameOperation	BlobDataOperation	It returns the file included in the *Content-Disposition* HTTP header during upload of this blob.
GetBlobMd5Operation	BlobDataOperation	It returns the md5Hash of the blob.
GetBlobSizeOperation	BlobDataOperation	It returns the size in bytes of the blob.
GetContentTypeOperation	BlobDataOperation	It returns the MIME content-type.
GetMaxSizeBlobOperation	BlobDataOperation	It sets the maximum size in bytes for the total upload.
BeginTransactionOperation	DataOperation	It begins a transaction.
CommitTransactionOperation	DataOperation	It commits a transaction.
CopyDataOperation	DataOperation	It copies one data object to another.
CreateDataOperation	DataOperation	It adds one or more new data objects/containers.
DeleteDataOperation	DataOperation	It deletes one or more data objects/containers.
EmptyRecycleBinOperation	DataOperation	It empties the recycle bin (the temporary limited storage of deleted data).
GetDeletedDataOperation	DataOperation	It retrieves a list of data objects deleted since the specified time.
GetUpdatedDataOperation	DataOperation	It retrieves a list of data objects updated since the specified time.
MergeDataOperation	DataOperation	It merges data objects.
QueryDataOperation	DataOperation	It executes query and returns data that matches the specified criteria.
RetrieveDataOperation	DataOperation	It retrieves data object specified by identifier.
RollbackTransactionOperation	DataOperation	It rollbacks the transaction.
SearchDataOperation	DataOperation	It performs text search in your data.
UndeleteFromRecycleBinDataOperation	DataOperation	It recovers data from recycle bin.
UpdateDataOperation	DataOperation	It updates the data object.
UpsertDataOperation	DataOperation	It updates an existing data object or inserts a new data object if it does not exist in the data container.
MetadataOperation	Operation	Operations to retrieve, deploy, create, update or delete metadata and for managing customizations.
CreateMetadataOperation	MetadataOperation	It creates new metadata component/s.
DeleteMetadataOperation	MetadataOperation	It deletes the metadata component.
DescribeApplicationGuiOperation	MetadataOperation	It describes the GUI of the application, e.g. layout.
GetQueuePropertiesOperation	MetadataOperation	It gets the properties of the queue.
GetStoragePropertiesOperation	MetadataOperation	It gets the properties of the storage service.
ListAvailableDataContainersOperation	MetadataOperation	It lists and describes the available data

		containers (e.g. objects, entities, tables).
ListMetadataOperation	MetadataOperation	It lists metadata.
RetrieveMetadataForDataContainerOperation	MetadataOperation	It retrieves metadata for the specified data container.
SetQueuePropertiesOperation	MetadataOperation	It sets the properties for the queue.
SetStoragePropertiesOperation	MetadataOperation	It sets the properties of the storage.
UpdateMetadataOperation	MetadataOperation	It updates metadata components.
QueueOperation	Operation	Operations that work with queues in platform as a service offers.
CreateQueueOperation	QueueOperation	It creates new queue.
DeleteElementFromQueueOperation	QueueOperation	It deletes an element from the queue.
DeleteQueueOperation	QueueOperation	It deletes the queue.
EmptyQueueOperation	QueueOperation	It clears all the elements from the queue.
GetElementFromQueueOperation	QueueOperation	It retrieves an element from the queue.
ListQueueOperation	QueueOperation	It lists all available queues.
PutQueueElementOperation	QueueOperation	It adds a new element to the queue.
UtilityOperation	Operation	Operations for environment configuration, registration, log manipulation, sending and receiving emails, figures manipulations and transformations.
RegistrationOperation	Operation	Operation registers new user.
CheckServiceAvailabilityOperation	UtilityOperation	It checks whether the specified service is available.
EmailOperation	UtilityOperation	Operations dealing with email messages.
GetAttachmentOperation	EmailOperation	It gets the content of the attachment.
GetEmailHeaderOperation	EmailOperation	It gets email header.
ReceiveMailOperation	EmailOperation	It receives incoming e-mails.
SendEmailOperation	EmailOperation	It sends an email message.
SendEmailToAdminsOperation	EmailOperation	It sends an email alert to all administrators.
EnvironmentOperation	UtilityOperation	Operations that work with application environment.
GetBackendAddressOperation	EnvironmentOperation	It gets the address of a specific backend.
GetCurrentBackendOperation	EnvironmentOperation	It gets a name of the current backend.
GetCurrentInstanceOperation	EnvironmentOperation	It gets an instance.
GetMaintenanceDateOperation	EnvironmentOperation	It returns the scheduled date of maintenance.
GetServerTimestampOperation	EnvironmentOperation	It retrieves the current system timestamp.
GetSystemPropertyValueOperation	EnvironmentOperation	It gets a system property.
SetSystemPropertyValueOperation	EnvironmentOperation	It sets a system property.
FigureOperation	UtilityOperation	Operations for image manipulations.
TransformFigureOperation	FigureOperation	It applies the chosen transformations (resize, rotate, flip, or crop) to images.
InvalidateSessionOperation	UtilityOperation	It ends one or more sessions.
LoggingOperation	UtilityOperation	Operations for logging.
GetLogDataOperation	LoggingOperation	It gets logs.
OperationException	Thing	It includes all instances of possible exceptions thrown by remote operations defined in vendors' APIs.
PaaSProvider	Thing	It includes instances of commercial vendors who offer platform as a service.
PaaSResource	Thing	A generic resource provided by PaaS vendor.
CommunicationResource	PaaSResource	It represents PaaS communication resource.
DataStorage	PaaSResource	Different types of data storages in PaaS
FileStorage	DataStorage	A storage working with files.

KeyValueStorage	DataStorage	NoSQL key-value storage
ObjectStorage	DataStorage	It stores data in form of objects.
RelationalDatabaseStorage	DataStorage	A PaaS storage with typical relational database' functionalities.
ProgrammingLanguage	Thing	It contains instances of computer languages used for developing applications in vendor's environment.
Queue	Thing	It covers all instances of FIFO queues supported by commercial providers of platform as a service.
Service	Thing	It includes all kinds of services provided by commercial vendors of platform as a service.
ServiceDescription	Thing	A description of the functionality provided by service
Variable	Thing	Its subclasses include input, output, and results of APIs' web services.
Parameter	Variable	A parameter
Input	Parameter	An input of web service
Output	Parameter	An output of web service
Result	Parameter	A result of the invocation of web service

5.2.5 Properties of classes

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 (33). 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 set of defined object properties, along with their corresponding domains, ranges and other characteristics is shown in Table 14. A total of 34 object properties were defined.

Table 14 Object properties defined in PaaS ontology

Object property	Domain	Range	Other characteristics
isOfferedByPaaSProvider	DataStorage	PaaSProvider	Inverse property: offersStorage
configuresEnvironment	EnvironmentOperation	ApplicationEnvironment	
containsDataObject	DataContainer	DataObject	Inverse property: isInContainer
definesOperation	Api	Operation	Inverse property: isDefinedIn
describes	ServiceDescription	Service	Asymmetric
facilitatesDevelopment	ApplicationEnvironment	Application	Asymmetric
hasContainer	DataStorage	DataContainer	Asymmetric
hasDataType	DataObject	DataType	Asymmetric

hasDestination	DataTypeMapper	DataType	Functional
hasInput	Operation	Input	Asymmetric
hasNoSqlDataObjectKey	NoSqlDataObjectType	NoSqlKeyType	Asymmetric
hasNoSqlDataObjectParent	NoSqlDataObjectType	NoSqlDataObjectType	
hasNoSqlDataObjectProperty	NoSqlDataObjectType	NoSqlPropertyType	
hasNoSqlPropertyValue	NoSqlPropertyType	SimpleServiceDataType	
hasOutput	Operation	Output	Asymmetric
hasParameter	Operation	Parameter	Asymmetric
hasResult	Operation	Result	
hasSource	DataTypeMapper	DataType	Functional
isDefinedIn	Operation	Api	Inverse property: definesOperation
isDeployedOn	Application	ApplicationServer	Asymmetric
isDescribedBy	Service	ServiceDescription	
isDevelopedFor	Api	Service	
isInContainer	DataObject	DataContainer	Inverse property: containsDataObject
isProvidedBy	Service	PaaSProvider	Inverse property: providesService
isSupportedInService	ProgrammingLanguage	Service	Inverse property: supportsLanguage
isThrown	OperationException	Operation	
isTypeFor	DataType	DataObject	Asymmetric
managesDataStorage	DataOperation	DataStorage	
offersStorage	PaaSProvider	DataStorage	Inverse property: isOfferedByPaaS-Provider
providesService	PaaSProvider	Service	isProvidedBy
runsInEnvironment	Application	ApplicationEnvironment	Asymmetric
supportsLanguage	Service	ProgrammingLanguage	Inverse property: isSupportedIn-Service
worksWithQueue	QueueOperation	Queue	

Additionally, instances can be described by data values. For this purpose, OWL provides data type properties (31) that relate instances to data values (instead of relating them to other instances). A total of 30 data properties were defined and are listed alphabetically in Table 15.

Table 15 Data properties of PaaS ontology

Data property	Domain	Range (XSD data type)
appId	Application	string
appURL	Application	string
appVersion	Application	string
assumption	ServiceDescription	string
capacity	DataStorage	string
communicationBandwidth	CommunicationResource	string
communicationType	CommunicationResource	string
dataContainerKey	DataContainer	string
dataContainerName	DataContainer	string

dataObjectValue	DataObject	-
effect	ServiceDescription	string
emailFromField	EmailMessageType	string
emailSubjectField	EmailMessageType	string
emailTextField	EmailMessageType	string
emailToField	EmailMessageType	string
hasName	PaaSProvider	string
noSqlDataObjectKind	NoSqlDataObjectType	string
noSqlKeyId	NoSqlDataObjectType	long
noSqlKeyName	NoSqlKeyType	string
noSqlKeyNamespace	NoSqlKeyType	string
noSqlPropertyName	NoSqlPropertyType	string
postcondition	ServiceDescription	string
precondition	ServiceDescription	string
serviceName	Service	string
serviceUrl	Service	string
typeName	DataType	string
userInfoEmail	UserInfoType	string
userInfoName	UserInfoType	string
userInfoUserName	UserInfoType	string

5.2.6 Creating instances

The last step in the methodology devised by Noy and McGuinnes (33) 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.

5.3 Ontology of platform as a service interoperability problems

The second ontology was also developed using Ontology Development 101 methodology (33), OWL and Protégé tool.

5.3.1 Domain and scope

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?

5.3.2 Reused concepts from other ontologies

Naudet et al. (17) developed a general ontology of interoperability that can be used as a starting point for this ontology of platform as a 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 (17). 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 this ontology. The complete list of reused concepts is listed in Table 16, and more details can be found in Chapter 5.3.4 and Chapter 5.3.5 in which classes and properties are described.

Table 16 Reused concepts from Naudet et al. (17)

Reused classes	Reused properties
InteroperabilitySolution, Indicator, InteroperabilityProblem, InteroperabilityExistenceCondition, Model, ConformancePoint, AntiPattern, InteroperabilitySolution, AprioriInteroperabilitySolution, AposterioriInteroperabilitySolution, Incompatibility , Misalignment , Heterogeneity	actsOnApi, actsOnModel, actsOnRepresentation, canInduceNewProblem, concernsApi, concernsModel, concernsRepresentation , definesCondition, existsIf, solvesProblem

5.3.3 Enumerate important terms

According to the instructions in Ontology Development 101 (33), the main activity in this step is to list all the relevant terms, without worrying about the overlap between the concepts or considering whether the concepts were OWL classes or properties. 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 (17), interoperability problems

between different databases listed in the literature - (15), (72), (73), (76), metadata interoperability problems (74), interoperability problems of web services - (75) and (77), the ATHENA Interoperability Framework (65) and problems identified by the author of this dissertation when working on use cases. Terms obtained from these sources are listed in Table 17. Interoperability problems, issues and conflicts from the existing literature are described in more details in Chapter 3.1.1.

Table 17 List of important terms for PaaS interoperability ontology

Source	Important terms
Naudet et al. (17)	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 (15)	DataLevelConflict, DataValueConflict, DataRepresentationConflict, DataUnitConflict, DataPrecisionConflict, SchemaLevelConflict, NamingConflict, EntityIdentifierConflict, SchemaIsomorphismConflict, GeneralizationConflict, AggregationConflict, SchematicDiscrepancies
Cloud4SOA (16)	different data models, different APIs, different query languages
Haslhofer and Klas (74)	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 (73)	generalization/specialization conflicts, description conflicts, structural conflicts, fragmentation conflicts, metadata conflicts, data conflicts
Sheth and Kashyap (72)	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 (75)	structural, value, encoding and semantic incompatibilities, missing methods, extra fields, missing fields, facet mismatches, cardinality mismatches
Zhu et al. (76)	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 (65)	interoperability at enterprise/business level, interoperability of processes, interoperability of services, interoperability of information/data

5.3.4 Definition of the class hierarchy

Again, from the list created in the previous step, the terms that describe independent objects were selected, because they present classes in the ontology. The top level of the ontology of platform as a service interoperability is shown in Figure 10. A total of 78 classes were defined. All classes are systematically specified in Table 18.

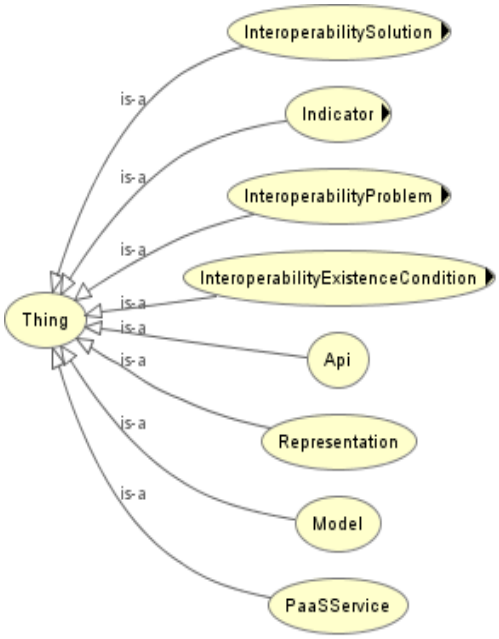

Figure 10 Top level classes of interoperability problems ontology

Table 18 List of classes in the PaaS interoperability ontology

Class	Super class	Description
Api	Thing	It represents remote APIs of platform as a service offers.
Indicator	Thing	Indicators detect the occurrence of potential conflicts (17).
AntiPattern	Indicator	It is a formalization of the known problems, the conditions in which the problems appear and possible solutions (17).
ConformancePoint	Indicator	It describes checking points that must be verified to test the actual operation of the system (17).
InteroperabilityExistenceCondition	Thing	An existence condition for interoperability problems (17).
Incompatibility	InteroperabilityExistenceCondition	It represents incompatibility (17).
Heterogeneity	Incompatibility	Heterogeneous interfaces (e.g. PaaS remote APIs) constitute the most commonly considered interoperability problems.
ApisHeterogeneity	Heterogeneity	PaaS providers offer different Application Programming Interfaces (APIs).
PaaSDataModelHeterogeneity	Heterogeneity	Each PaaS provider supports different types of underlying data models.
ProgrammingModelHeterogeneity	Heterogeneity	Different PaaS providers offer different programming models.
QueryLanguagesHeterogeneity	Heterogeneity	Query languages differ among various PaaS providers.
SupportedDataTypesHeterogeneity	Heterogeneity	Different data types are supported in different PaaS offers.
SupportedProgrammingLanguages-Heterogeneity	Heterogeneity	Different PaaS offers support different programming languages.
TypesOfPaaSServicesHeterogeneity	Heterogeneity	There are heterogeneities among PaaS services types.

Misalignment	Incompatibility	Misalignment can occur when a system constrains the building, structure or behavior of other system (17).
InteroperabilityProblem	Thing	It represents an interoperability problem.
PaaSApiInteroperabilityProblem	InteroperabilityProblem	It represents an interoperability problem that occurs because of different vendors' APIs.
AbstractionLevelProblem	PaaSApiInteroperabilityProblem	It lists interoperability problems that arise because two semantically similar API operations or their parameters are represented at different level of abstraction (77).
ApiOperationAggregationProblem	AbstractionLevelProblem	Two semantically similar API operations where one is represented as an aggregate of another API operation (77).
ApiOperationGeneralizationProblem	AbstractionLevelProblem	Semantically similar API operations are represented at different levels of generalization (77).
ApiOperationParameterConflictProblem	AbstractionLevelProblem	Semantically similar entities are modeled as a parameter in one PaaS offer and API operation in another PaaS offer (77).
ApiOperationLevelProblem	PaaSApiInteroperabilityProblem	Interoperability problems between API operations.
ApiOperationNamingProblem	ApiOperationLevelProblem	Problems that occur because of different naming.
ApiOperationHomonymProblem	ApiOperationNamingProblem	Semantically unrelated API operations might have the same name in different PaaS offers (homonyms) (77).
ApiOperationSynonymProblem	ApiOperationNamingProblem	Semantically alike API operations might be named differently in different PaaS offers (synonyms) (77).
ApiOperationSchemaIsomorphismProblem	ApiOperationLevelProblem	Semantically similar API operations may have different numbers of parameters (77).
MissingApiOperationProblem	PaaSApiInteroperabilityProblem	Some needed API operation is missing from vendor's remote API (75).
ParameterLevelProblem	PaaSApiInteroperabilityProblem	Differences that exist due to different descriptions for semantically similar parameters (77).
ParameterDataTypeProblem	ParameterLevelProblem	Two semantically similar parameters might have different data types (77).
ParameterNamingProblem	ParameterLevelProblem	Problems arise due to different parameters' naming.
ParameterHomonymProblem	ParameterNamingProblem	Two semantically unrelated parameters might have the same names (77).
ParameterSynonymProblem	ParameterNamingProblem	Two semantically alike parameters might have different names (77).
ParameterScalingProblem	ParameterLevelProblem	Two semantically similar parameters might be represented using different precisions (77).
PaaSApplicationInteroperabilityProblem	InteroperabilityProblem	Interoperability problems that arise when PaaS applications need to cooperate.
ApplicationComputerLanguageNot-SupportedProblem	PaaSApplicationInteroperability-Problem	Programming language in which the specific application is written may not be supported by specific PaaS vendor.
LibraryNotSupportedProblem	PaaSApplicationInteroperability-Problem	Some PaaS vendor may forbid some standard libraries that the application uses (for example, some standard libraries used in J2EE are not supported by some providers).

PaaSLegalInteroperabilityProblem	InteroperabilityProblem	Interoperability problems that arise due to different legislature.
DataPrivacyLegislationInteroperability-Problems	PaaSLegalInteroperabilityProblem	Different countries have different data privacy laws.
DataSovereigntyInteroperabilityProblem	PaaSLegalInteroperabilityProblem	Is the data subject to the jurisdiction where it is physically stored or hosted on servers?
OwnershipOfDataInteroperabilityProblem	PaaSLegalInteroperabilityProblem	Agreements on temporary or permanent transfer of certain data rights to the service provider by the end-user in exchange for using the cloud services.
PaaSOrganizationalInteroperabilityProblem	InteroperabilityProblem	Interoperability problems that arise at interoperability level.
PaaSStorageInteroperabilityProblem	InteroperabilityProblem	Interoperability problems that arise because of heterogeneities of cloud storages.
DataAggregationProblem	PaaSStorageInteroperabilityProblem	An aggregation is used in one cloud storage to represent a set of entities in another cloud storage (72).
DataAttributeEntityProblem	PaaSStorageInteroperabilityProblem	The same entity is being modeled as an attribute in one cloud storage and a data container in another storage (72).
DataAttributeIntegrityConstraintProblem	PaaSStorageInteroperabilityProblem	Two semantically similar attributes are restricted by some inconsistent constraints (72).
DataContainerGeneralizationProblem	PaaSStorageInteroperabilityProblem	Two entities are represented at different levels of generalization in various cloud storages (72).
DataContainerIdentifierProblem	PaaSStorageInteroperabilityProblem	Two data containers modeling the same entity have semantically different identifiers (72).
DataContainerNamingProblem	PaaSStorageInteroperabilityProblem	Conflicts that arise due to naming of data containers.
DataContainerHomonymProblem	DataContainerNamingProblem	Semantically unrelated entities might have the same name in different cloud storages (homonyms) (72).
DataContainerNamingRestrictionProblem	DataContainerNamingProblem	Some names are reserved and forbidden, and some types of names can be required (e.g. Salesforce requires that you name your custom object with postfix __c).
DataContainerSynonymProblem	DataContainerNamingProblem	Semantically similar entities are named differently in different PaaS storages (synonyms) (72).
DataContainerRelationshipProblem	PaaSStorageInteroperabilityProblem	Different means to define relationships between two data containers (e.g. foreign key, no relationship between data containers etc.), or maybe some cloud storage does not have any means to connect two data containers.
DataContainerUnionCompatibilityProblem	PaaSStorageInteroperabilityProblem	Two entities are union incompatible when a one-one mapping is not possible between the two sets of attributes (72).
DataDefaultValueProblem	PaaSStorageInteroperabilityProblem	Two attributes might have different default values in different cloud storages (72).
DataDifferentSupportedDataTypesProblem	PaaSStorageInteroperabilityProblem	Different cloud storages support different data types.
DataModelDifferencesProblem	PaaSStorageInteroperabilityProblem	Differences between data models.
DataObjectNamingProblem	PaaSStorageInteroperabilityProblem	Different naming of data objects can also be the cause for interoperability problems.
DataObjectHomonymProblem	DataObjectNamingProblem	Two data objects that are semantically unrelated might have the same name (homonyms) (72).

DataObjectNamingRestrictionProblem	DataObjectNamingProblem	Some names are reserved and forbidden, and some types of names can be required (e.g. Salesforce requires that you name your custom fields with postfix__c).
DataObjectSynonymProblem	DataObjectNamingProblem	Two data objects that are semantically alike might have different names (synonyms) (72).
DataPrecisionProblem	PaaSStorageInteroperabilityProblem	Data object in different cloud storages have different precisions.
DataRepresentationProblem	PaaSStorageInteroperabilityProblem	Different data types or representations of two semantically similar attributes (72).
DataScalingProblem	PaaSStorageInteroperabilityProblem	Data has different units and measures (15).
DataSchemaIsomorphismProblem	PaaSStorageInteroperabilityProblem	Semantically alike entities have different numbers of attributes (72).
DataValueAttributeProblem	PaaSStorageInteroperabilityProblem	The value of an attribute in one cloud storage corresponds to an attribute in another cloud storage (72).
DataValueEntityProblem	PaaSStorageInteroperabilityProblem	This conflict arises when the value of an attribute in one cloud storage corresponds to a data container in another data storage.
DifferentQueryLanguageProblem	PaaSStorageInteroperabilityProblem	The query languages of different PaaS providers are different.
DifferentTransactionMechanismProblem	PaaSStorageInteroperabilityProblem	Transactions mechanism may be different in various PaaS offers (76).
MissingDataItemProblem	PaaSStorageInteroperabilityProblem	One of the semantically similar entities has a missing attribute (72).
InteroperabilitySolution	Thing	A solution to some interoperability problem.
AposterioriInteroperabilitySolution	InteroperabilitySolution	This is a solution that corrects problems after they occurred (17)
BridgingSolution	AposterioriInteroperabilitySolution	It is an intermediate system, often called adapter (17).
AprioriInteroperabilitySolution	InteroperabilitySolution	This is a solution that corrects problems by anticipation (17).
HomogenisationSolution	AprioriInteroperabilitySolution	It uses a unified model of several kinds: a unified language, a unified metamodel, or a unified interface such as API (17).
Model	Thing	A simplified representation of a concrete or abstract reality (17).
PaaSService	Thing	PaaSService is concrete platform as a service offer (e.g. Google App Engine).
Representation	Thing	It is the aggregation of symbols used to materialize a model (17).

5.3.5 Define the properties of classes

A set of defined object properties, along with their corresponding domains, ranges and other characteristics is shown in Table 19. A total of 14 object properties were defined. For now, the ontology does not contain any data properties.

Table 19 Object properties of the interoperability problems ontology

Object property	Domain	Range	Other characteristics
actsOnApi (17)	BridgingSolution	Api	Asymmetric
actsOnModel (17)	HomogenizationSolution	Model	Asymmetric
actsOnRepresentation (17)	HomogenizationSolution	Representation	Asymmetric
canInduceNewProblem (17)	InteroperabilitySolution	InteroperabilityProblem	
concernsApi (17)	Heterogeneity	Api	Inverse property: hasHeterogeneity
concernsModel (17)	Heterogeneity	Model	
concernsRepresentation (17)	Heterogeneity	Representation	
definesCondition (17)	Indicator	InteroperabilityExistenceCondition	Inverse property: isDefinedByIndicator
existsIf (17)	InteroperabilityProblem	InteroperabilityExistenceCondition	Inverse property: causes
solvesProblem (17)	InteroperabilitySolution	InteroperabilityProblem	Asymmetric
isSolvedUsing	InteroperabilityProblem	InteroperabilitySolution	
hasHeterogeneity	Api	Heterogeneity	Inverse property: concernsApi
isDefinedByIndicator	InteroperabilityExistenceCondition	Indicator	Inverse property: definesCondition
causes	InteroperabilityExistenceCondition	InteroperabilityProblem	Inverse property: existsIf

5.3.6 Creation of the facets and instances

The last step in the methodology devised by Noy and McGuinnes (33) is filling in the values for individuals. In Protégé, the class needs to be selected and individuals of the chosen class can then be created. A total of 15 individuals were created.

5.4 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 (223). 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 (223), 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 Chapter 3.2.2 of this dissertation. Neuhaus et al. (223) 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 subchapters, the evaluation process of developed ontologies will be shown.

5.4.1 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, 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!) (138) 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.

5.4.2 Evaluation by humans

Ontology was also evaluated by four human experts working in the field of cloud computing interoperability and related science projects Contrail (102) and mOSAIC (132). The

questionnaire was sent to ten researchers, and four answers were obtained. 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 20.

Table 20 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. x509) 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 and use cases were used to better describe where the ontologies will be used
- 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

5.4.3 Application-based evaluation

To perform application-based evaluation of the ontologies, the use cases where ontologies are extensively used were performed. The use cases are described in Chapter 4. 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.

6. PROPOSED SOLUTION AND METHODOLOGY

6.1 Semantic PaaS 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 data migration between PaaS storages). Web services and all other parts of the author's prototype were implemented in Java.

SAWSDL (W3C's Semantic Annotations for WSDL) (123) lightweight annotation was used to define semantic web services. As already stated in Chapter 2.6.2, 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. A source code of the SOWER tool which can be used to semantically annotate web services using SAWSDL standard was downloaded, installed and adjusted, and included into the author's client web application. The SOWER tool was developed as part of the SOA4All FP7 project (224). The aforementioned tool is an editor to facilitate the manual annotation of WSDL service descriptions with the semantic information (224). SOWER saves SAWSDL files to *iServe* repository (remote repository of semantic web services of the SOA4All FP7 project), but the code was changed to save the files to a folder that can be accessed by Glassfish application server on which other parts of the prototype are deployed. 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. In SOWER, the platform as a service ontology is opened. Also, WSDLs of desired web services can be opened, such as *AzureServices* that represent the remote API's operations of Windows Azure platform as a service. The ontology class can be dragged and dropped to WSDL area, and the tool will automatically annotate the service operation. For instance, the Azure's *createTable* web service operation can be referenced to *CreateDataOperation* class of the OWL ontology.

Similarly, input and output parameters of web services can be semantically annotated. Data types on inputs and outputs are annotated using cross-PaaS concepts of simple and complex service data types from PaaS ontology. More detailed mappings and needed transformations can be specified in SAWSDL by using "liftingSchemaMapping" and "loweringSchemaMapping" annotations. The SOWER tool supports addition of the mentioned semantic annotations, so this standard was used to map outputs of one operation to inputs of another operation. For this purpose, SAWSDL allows the usage of any mapping language and its specification contains examples in XQuery, XSLT, and SPARQL. In this work, XSLT (217) was used for XML transformations.

An example of semantic annotation of web services will now be discussed. The service annotated with *GetUserInformation* has output *UserInfoType* that provides information on the user, and *SendEmailOperation* has input of *EmailMessageType*. Some user information can be sent to a predefined email account. Mappings and transformations need to be defined. Semantic annotations are defined in SAWSDL files, and an example from Salesforce's SAWSDL file, together with relevant annotated elements is shown in Table 21.

Table 21 Example of operation's annotations and transformations

Element name	Type	Annotations
getUserInfo	operation	<operation name="getUserInfo" *sawsdl:modelReference* = "**http://localhost:8080/PaaSOntologyv4.owl#GetUserInfoOperation**" >
tns:getUserInfoResponse	output	<output message="tns:getUserInfoResponse" wsam:Action="http://services.api.salesforce.foi.org.hr/SalesForceServices/getUserInfoResponse" *sawsdl:modelReference* = "**http://localhost:8080/PaaSOntologyv4.owl#UserInfoType**">
getUserInfoResult	complex type	<xs:complexType name="getUserInfoResult" *sawsdl:liftingSchemaMapping* = "**http://localhost:8091/SowerWeb/xslt/userInfo_lifting.xslt**" *sawsdl:modelReference* = "**http://localhost:8080/PaaSOntologyv4.owl#UserInfoType**">
sendEmail	operation	<operation name="sendEmail" sawsdl:modelReference = "**http://localhost:8080/PaaSOntologyv4.owl#SendEmailOperation**">
tns:sendEmail	input	<input message="tns:sendEmail" wsam:Action="http://services.api.salesforce.foi.org.hr/SalesForceServices/sendEmailRequest" *sawsdl:modelReference* = "**http://localhost:8080/PaaSOntologyv4.owl#EmailMessageType**">

singleEmailMessage	complex type	<xs:complexType name="singleEmailMessage" *sawsdl:loweringSchemaMapping* = "**http://localhost:8091/SowerWeb/xslt/userInfo_lowering_to_email.xslt**" *sawsdl:modelReference* = "**http://localhost:8080/PaaSOntologyv4.owl#EmailMessageType**">

In the example shown in the table above, operations, input and output elements and types are linked with the appropriate cross-PaaS concepts from PaaS ontology described in Chapter 5.2. Two schema mappings are defined: lifting schema that maps from WSDL to an ontology element, and lowering schema which transforms the known ontology element to input of *SendEmailOperation*. During the service execution, the prototype performs needed transformations. XSTL files for lifting and lowering schemas need to be manually specified before semantic annotations and successful service composition. For the above mentioned simple scenarios, XML transformations are shown in Table 22.

Table 22 Example of input/output transformations

XML description	XML content
1. SOAP result obtained after sample execution of *getUserInfo()* operation	<?xml version="1.0"?> <return> <accessibilityMode>false</accessibilityMode> <currencySymbol>$</currencySymbol> <orgAttachmentFileSizeLimit>5242880</orgAttachmentFileSizeLimit> <orgDefaultCurrencyIsoCode>USD</orgDefaultCurrencyIsoCode> <orgDisallowHtmlAttachments>false</orgDisallowHtmlAttachments> <orgHasPersonAccounts>false</orgHasPersonAccounts> <organizationId>00DA0000000ZdWQMA0</organizationId> <organizationMultiCurrency>false</organizationMultiCurrency> <organizationName>FOI</organizationName> <profileId>00eA0000000ssupIAA</profileId> <sessionSecondsValid>7200</sessionSecondsValid> <userEmail>darkoandr@yahoo.com</userEmail> <userFullName>Darko Androcec</userFullName> <userId>005A0000000p3ZeIAI</userId> <userLanguage>en_US</userLanguage> <userLocale>en_US</userLocale> <userName>darkoandr@yahoo.com</userName> <userTimeZone>America/Los_Angeles</userTimeZone> <userType>Standard</userType> <userUiSkin>Theme3</userUiSkin> </return>
2. Lifting schema: *userInfo_lifting.xslt*	<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

	xmlns:n1="http://localhost:8080/PaaSOntologyv3.owl#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:service="http://localhost:8080/SalesForceServices.sawsdl" version="1.1"> <xsl:output encoding="iso-8859-1" indent="yes" method="xml" version="1.0"/> <xsl:template match="/"> <rdf:RDF> <n1:UserInfoDataType> <n1:userInfoName> <xsl:value-of select="/return/userFullName"/> </n1:userInfoName> <n1:userInfoEmailAddress> <xsl:value-of select="/return/userEmail"/> </n1:userInfoEmailAddress> <n1:userInfoUserName> <xsl:value-of select="/return/userName"/> </n1:userInfoUserName> </n1:UserInfoDataType> </rdf:RDF> </xsl:template> </xsl:transform>
3. Transformed output (after XSTL transformation)	<?xml version="1.0" encoding="iso-8859-1"?><rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:service="http://localhost:8080/SalesForceServices.sawsdl" xmlns:n1="http://localhost:8080/PaaSOntologyv3.owl#"> <n1:UserInfoDataType> <n1:userInfoName>Darko Androcec</n1:userInfoName> <n1:userInfoEmailAddress>darkoandr@yahoo.com</n1:userInfoEmailAddress> <n1:userInfoUserName>darkoandr@yahoo.com</n1:userInfoUserName> </n1:UserInfoDataType> </rdf:RDF>
4. Lowering schema: *userInfo_lowering_to_email.xslt*	<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:n1="http://localhost:8080/PaaSOntologyv3.owl#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:service="http://localhost:8080/SalesForceServices.sawsdl" version="1.1"> <xsl:output encoding="iso-8859-1" indent="yes" method="xml" version="1.0"/>

	<xsl:template match="/"> User name: <xsl:value-of select="rdf:RDF/n1:UserInfoDataType/n1:userInfoUserName"/> Full name: <xsl:value-of select="rdf:RDF/n1:UserInfoDataType/n1:userInfoName"/> </xsl:template> </xsl:transform>
5. Input after XSLT transformation	<?xml version="1.0" encoding="ISO-8859-1"?> <SingleEmailMessage xmlns:n1="http://localhost:8080/PaaSOntologyv3.owl#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:service="http://localhost:8080/SalesForceServices.sawsdl"> <toAddresses> dandrocec@foi.hr </toAddresses> <subject> New user is automatically added to Vosao </subject> <plainTextBody> New user is automatically added to Vosao. Password needs to be generated. User name: darkoandr@yahoo.com Full name: Darko Androcec</plainTextBody> </SingleEmailMessage>

6.2 Implementation of AI planning

6.2.1 JSHOP2 planner

For AI planning process, a JSHOP2 planner was used in this dissertation. 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. 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 execution (225). The objective of JSHOP2 and other HTN planners is to accomplish a set of tasks where each task can be decomposed, until primitive tasks (226) 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 precondition (must be satisfied to execute the action), delete list (set of

properties that will be removed), and add list (set of properties that will be added) (225). 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. Next subchapters deal with definitions of domain and problem description files in the context of executing compositions and finding interoperability problems introduced earlier in this dissertation when the two use cases were described.

6.2.2 JSHOP2 problem description

Problem description file is composed of logical atoms showing the initial state and a task list (225). 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, when the user chooses "*add existing user to another PaaS*", he must also select a source and target PaaS and data container on target PaaS where user information will be stored. One sample of the mentioned task list could be:

```
((addUserToAnotherPaaS SalesForce GoogleAppEngine UserEntity))
```

If the user chooses another interoperability action, then other task list to be executed by planner will be generated. For example, if the user selects the data migration between Salesforce's and Google App Engine's PaaS storages, 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. This file is described in the next subchapter that defines which operators need to be executed to carry out some interoperability actions.

The initial state (a set of logical atoms) is also created programmatically. Based on the chosen method representing the chosen interoperability action (task list to be executed), SAWSDL and/or PaaS ontology files are parsed to generate logical atoms. For example, if the chosen interoperability action is not to migrate data, there is no need to parse PaaS ontology to obtain all data types of PaaS storages and their mappings. This enables users to always have relatively small problem definition and faster execution of the planning process. A SAWSDL parser was developed in Java by using EasyWSDL open-source library and its extension EasySAWSDL. The class for parsing OWL ontology was implemented by using Apache Jena

library. Based on these two files, various logical atoms could be generated to represent the initial state. All possible logical atoms, together with their definition and description of their creation are systematically listed in Table 23.

Table 23 Possible logical atoms in the initial state

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 semantic annotation by means of *sawsdl:modelReference* on a service operation, it then generates *hasApiOperation* logical atom in JSHOP2 problem description file
ServiceIOType `(ServiceIOType NoSqlDataObjectType)`	- it shows that specific cross-PaaS type is used in input or output of some operations - SAWSDL files are parsed to find out annotations (*sawsdl:modelReference*) on simple and complex types used by inputs and outputs of the operations
operationHasInput `(operationHasInput GoogleAppEngine CreateDataOperation NoSqlDataObjectType)`	- it describes the input of the operation (PaaS offer, cross-PaaS operation name, and its cross-PaaS concept for type) - SAWSDL files are parsed to find out annotations (*sawsdl:modelReference*) on inputs of the operations and on simple and complex types used by inputs
operationHasOutput `(operationHasOutput SalesForce GetUserInfoOperation UserInfoType)`	- it describes the output of the operation (PaaS offer, cross-PaaS operation name, and its cross-PaaS concept for type) - SAWSDL files are parsed to find out annotations (*sawsdl:modelReference*) on outputs of the operations and on simple and complex types used by outputs
TypeHasLiftingSchema `(TypeHasLiftingSchema SalesForce UserInfoType userInfo_lifting)`	- it shows which type defined in a specific PaaS offer has lifting schema mapping associated with it - SAWSDL files are parsed to determine which types are annotated by using *sawsdl:liftingSchemaMapping*
TypeHasLoweringSchema `(TypeHasLoweringSchema GoogleAppEngine`	- it shows which type defined in a specific PaaS offer has lowering schema mapping associated with it

NoSqlDataObjectType userInfo_lowering_to_email)	- SAWSDL files are parsed to determine which types are annotated by using *sawsdl:loweringSchemaMapping*
typeInCurrentData (typeInCurrentData salesforcecurrency)	- it is used when the user chooses data migration interoperability action - it shows which type is present in storages of the chosen PaaS offers - present data types in PaaS storages are obtained calling remote APIs of PaaS providers
dataTypeMappingExists (dataTypeMappingExists azuresmallmoney xsddecimal)	- it specifies data type mapping between data types of different PaaS storages - the PaaS ontology is parsed to obtain all instances of DataTypeMapper OWL class that represent data type mappings between PaaS storages – a more detailed description is presented in Chapter 4.2.4 of this dissertation

6.2.3 JSHOP2 domain description

The domain description file consists of operators, methods and axioms (225). The preconditions of operators and methods are described using logical expressions (226). An operator is a primitive task and it consists of logical preconditions, delete list (negative postconditions), add list (positive postconditions), and optionally cost (225). A method consists of logical precondition and a task list (225), and it defines how composite tasks are decomposed. The domain description file is defined manually. Two methods which show how to get plans for two interoperability actions presented in use case 1 and use case 2 were defined and are listed in Table 24. These methods are decomposed into operators (see Table 25) which are shown together with their preconditions and positive postconditions.

Table 24 Methods defined in JSHOP2 domain file

Method	JSHOP2 source	Description
migrateData	(:method (migrateData ?from ?to) () ((!checkDataTypeMappings ?from)(!createDataModelOntology ?from)	- method showing which operators should be called to migrate data from one PaaS storage to another - first, the existence of needed data type mappings are checked, then data model ontology is created, and

	(!createDataElementsFromOntology ?to)))	finally data is migrated to target PaaS storage
addUserToAnotherPaaS	(:method (addUserToAnotherPaaS ?from ?to ?containerName) () ((!checkAddUserServiceDataTypeMappings ?from ?to)(!login ?from) (!getUserInfo ?from) (!createData ?to ?containerName)(!sendEmail ?from)))	- this method shows which operators to call to add current user to another PaaS - needed service input/output data type mappings are checked, and then the existence of appropriate services for login, user information, data creation, and email sending are checked

Table 25 Operators and their preconditions and postconditions

Operator	Preconditions	Positive postconditions
checkDataTypeMappings	((forall (?p) (typeInCurrentData ?p) (and (dataTypeMappingExists ?p ?x)))) - Precondition checks whether all data types from data to be migrated have appropriate data type mappings defined in JSHOP2 problem file	((hasAllDataTypeMappings ?from))
createDataModelOntology	(hasApiOperation ?from CreateDataModelOntologyOperation)	((haveDataOntology ?from))
createDataElementsFromOntology	(hasApiOperation ?to CreateDataElementsFromOntologyOperation)	((dataMigrationSuccessfulTo ?to))
checkAddUserServiceDataTypeMappings	(operationHasOutput ?from GetUserInfoOperation ?type1) (operationHasInput ?to CreateDataOperation ?type2) (operationHasInput ?from SendEmailOperation ?type3) (TypeHasLiftingSchema ?from UserInfoType ?lifting) (TypeHasLoweringSchema ?to NoSqlDataObjectType ?lowering1) (TypeHasLoweringSchema ?from EmailMessageType ?lowering2)) - If appropriate lifting and lowering schemas exist, and are defined in the planning problem, then it is assumed that there are no input/output message problems	((TransformationDuringExecution GetUserInfoOperation ?from UserInfoType ?lifting) (TransformationDuringExecution CreateDataOperation ?toNoSqlDataObjectType ?lowering1) (TransformationDuringExecution SendEmailOperation ?from EmailMessageType ?lowering2)) - this will be used during execution to see which XSLT transformations need to be performed
login	(hasApiOperation ?from	((userIsLoggedIn ?from))

	LoginOperation)	
getUserInfo	(hasApiOperation ?from GetUserInfoOperation)	((userInfoIsObtained ?from))
createData	(hasApiOperation ?to CreateDataOperation)	((dataObjectIsCreated ?to))
sendEmail	(hasApiOperation ?from SendEmailOperation)	((EmailIsSent ?from))

6.3 Plan execution and service composition

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 (218) was used to dynamically invoke web service. This framework enables a dynamic creation of web service clients, and invokes web services with their inputs. It works fine, when operation inputs and outputs are simple types, and the class which takes care of inputs was implemented.

But there are operations that have different complex types, and to be able to actually execute web services, the transformations between inputs and outputs should be performed. The transformations are defined in SAWSDL and accompanying lifting and lowering schema mappings in form of XSLT. Furthermore, they are also defined as postconditions of the *checkAddUserServiceDataTypeMappings* operator in JSHOP2. After the plan execution, the state can be obtained from JSHOP2 planner, and users can then parse which transformations should be performed. This was done in the prototype: during execution, the program looks at the current state after a plan is found, and it searches for *TransformationDuringExecution* to get all lifting/lowering transformations that need to be performed. In Apache CXF, all message transformations are done by means of interceptor classes. Interceptor classes are the fundamental unit of Apache CXF that can read, transform, process the headers of messages, and validate messages both at client and server side. The interceptors can be added

programmatically during execution, if transformations are needed. Custom interceptor was implemented to adequately transform input and output message based on XSLT files obtained from *TransformationDuringExecution* postconditions from the planner's state, and then CXF features were used to dynamically call web services with appropriately transformed SOAP inputs. Open-source XSLT processor Xalan was used to parse XSLT files, and standard Java classes for XML parsing were used to parse intermediate XML files. An example of intermediate XML files is shown in Chapter 6.1 in Table 22.

6.4 Finding interoperability problems

If there is no suitable plan returned by 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 planning domains. The most relevant methods are listed in Chapter 3.3.2 of this dissertation. This approach is similar to the one proposed by Goebelbecker and Keller (146). 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 excuse where they replan with new initial states to find out whether they found the cause why the plan is not found.

This approach differs from the one proposed by Goebelbecker and Keller (146), because it does not need replanning that is an expensive and time-consuming task. This algorithm consists of four main steps:

1. Find problematic operator or method

The domain description file of JSHOP2 is simple and it is described in Chapter 6.2.3. Every interoperability action is represented by one method that describes a set of operators that need to be executed. When the user chooses an interoperability action, source PaaS offer, target PaaS offer and other parameters, an AI goal is formed that calls the appropriate JSHOP2 method with parameters. There is only one way to successfully get a plan (for now, there is no interoperability action defined where more possible solution paths were introduced) – 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 operator or method were found where first

BACKTRACKING action occurs. This action occurs when some preconditions of the operator or method are not satisfied, and then 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 problematic atom (problematic method or operator where interoperability problem had occurred).

2. Parse concrete preconditions

The next step is to parse preconditions of a 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.

3. Check whether the preconditions are satisfied in the end state

The end state (the last state after 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.

4. List interoperability problems

Chapter 6.2.2 describes how logical atoms in the initial state are programmatically created. Each logical atom that is used in states and preconditions has some meaning (for example, *hasApiOperation* describes that some PaaS offer has a particular API operation annotated with 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.

Here are some examples. Everything started with the scenario introduced in the Chapter on use case 2, and some intentional errors were made in SAWSDL annotations and PaaS ontology to test the problem finding technique and the software tool. These tests, together with found problematic operator, found problematic precondition, and results obtained from the client web application are shown in Table 26. In the first three test scenarios, the author selected to add the existing user in his client web application from Salesforce instance to Vosao CMS deployed on Google App Engine instance. In the last scenario the choice was to migrate all the data from Salesforce to Google App Engine PaaS offer.

Table 26 Testing examples of finding interoperability problems

Test scenario	Problematic operator with all preconditions	Problematic preconditions	Message on web client application
- In Salesforce's SAWSDL file an annotation on operation sendEmail to cross-PaaS operation concept from PaaS ontology is removed	`(:operator (!sendEmail ?from) ((hasApiOperation ?from SendEmailOperation))`	`(hasApiOperation ?from SendEmailOperation )`	MissingApiOperationProblem => Operation sendEmail is missing in Salesforce! Check service annotations (SAWSDL file) or whether this operation is supported by PaaS vendor!
- In Salesforce's SAWSDL file an annotation of lowering schema mapping on complex type *EmailMessageType* is removed	`(:operator (!checkAddUserServiceDataTypeMappings ?from ?to) ((operationHasOutput ?from GetUserInfoOperation ?type1)(operationHasInput ?to CreateDataOperation ?type2)(operationHasInput ?from SendEmailOperation ?type3)(TypeHasLiftingSchema ?from UserInfoType ?lifting)(TypeHasLoweringSchema ?to NoSqlDataObjectType ?lowering1)(TypeHasLoweringSchema ?from EmailMessageType ?lowering2))`	`(TypeHasLoweringSchema ?from EmailMessageType ?lowering2)`	Missing lowering schema => TypeHasLoweringSchema salesforce EmailMessageType ?lowering2! Check service annotations (SAWSDL file) and add adequate lowering schema!

- In Salesforce's SAWSDL file an annotation on input of *sendMail* operation is intentionally removed together with the link to lowering schema mapping on complex type *EmailMessage*	`(:operator (!checkAddUserServiceDataTypeMappings ?from ?to) ((operationHasOutput ?from GetUserInfoOperation ?type1)(operationHasInput ?to CreateDataOperation ?type2)(operationHasInput ?from SendEmailOperation ?type3)(TypeHasLiftingSchema ?from UserInfoType ?lifting)(TypeHasLoweringSchema ?to NoSqlDataObjectType ?lowering1)(TypeHasLoweringSchema ?from EmailMessageType ?lowering2))`	`(operationHasInput ?from SendEmailOperation ?type3)`
		
		`(TypeHasLoweringSchema ?from EmailMessageType ?lowering2)`
			Missing annotation on operation input => operationHasInput salesforce SendEmailOperation ! Check service annotations (SAWSDL file)!
			
			Missing lowering schema => TypeHasLoweringSchema salesforce EmailMessageType ?lowering2! Check service annotations (SAWSDL file) and add adequate lowering schema!

- All the Salesforce's PaaS storage data type mappings were removed from the PaaS ontology	`(:operator (!checkDataTypeMappings ?from) ((forall (?p) (typeInCurrentData ?p) (and (dataTypeMappingExists ?p ?x))))`	`(typeInCurrentData ?p) (and (dataTypeMappingExists ?p ?x)`	DataRepresentationProblem : Source PaaS offering storage includes data types that cannot be mapped to destination PaaS's storage -> Missing or impossible data type mapping!

Most of the problems can be identified using this method. However, some problems can occur only in service composition execution phase. For example, some PaaS API could be temporary unavailable. Lifting and lowering schema can also have some errors, leading to runtime errors due to input mismatch. In these cases, the client web application will show the exception thrown. For the most common exceptions, a user-friendly description is added to list possible causes of the error to an end user. For example, if exception contains *org.apache.cxf.interceptor.Fault*, then the following message is printed: "There is a problem with input for the operation *operation_name PaaS_offer*! Please check lifting and lowering schema mappings"! *Operation_name* and *PaaS_offer* variables are substituted with concrete values during execution.

6.5 Methodology for detection of interoperability problems

6.5.1 Methodology justification

Interoperability problems between cloud providers are one of the most serious issues of this new computing paradigm. A methodology is needed to systematically and effectively find and solve interoperability problems. Currently, there is still no methodology that aims at identification and resolution of interoperability problems; neither among APIs of commercial platforms as a service nor among cloud offers in general. The most relevant similar interoperability methodologies are explained in Chapter 3.1.5. The only existing methodology that takes into consideration cloud interoperability problems is methodology developed by REMICS consortium (108) but its main purpose is to provide model-driven approach to migrate legacy application on software as a service. The part of methodology that addresses interoperability deals with finding possible interoperability problems for the future migrated system, and with building interoperability components in migrated software when it is needed. It does not consider interoperability problems between different cloud providers. In REMICS's methodology, interoperability is modeled as one of five technical practices with five tasks: identification of interoperability problems/scenarios, definition of interoperability requirements, performing interoperability analysis, implementation of interoperability components, and interoperability monitoring (108).

For this reasons, a new methodology with detailed steps to find and solve interoperability problems is here proposed. This new methodology is focused and implemented on platform as a service, but it can be used in any of the three main models of cloud computing. The

methodology uses iterative approach, because PaaS offers and their APIs evolve and change very often. The user's interoperability requirements also change during time and new interoperability problems could arise. This dissertation focuses on using remote PaaS APIs to solve interoperability problems on technical, PaaS storage and services level. Other levels of interoperability (for example, legal and organizational level) cannot be solved using remote APIs, and are not subject of this work and proposed methodology. In the next subchapter, the steps of the methodology will be described.

6.5.2 Steps of the methodology

The proposed methodology has five main steps:

- Requirements identification
- Interoperability analysis
- Solution design
- Solution implementation
- Evaluation

In the first step, the most important interoperability needs of users should be listed, i.e. interoperability actions such as migration of data from one PaaS offer to another cloud storage, working with external cloud data in PaaS applications, communication between two applications deployed on different PaaS offerings, composition of two or more API operations of different providers, etc. These actions can be derived from the available use cases presented in technical and research papers, deliverables of related projects, and proposals for cloud standards where authors already did some research on user's interoperability requirements. Based on the identification of relevant interoperability actions, adequate use cases should be defined and described.

Interoperability analysis deals with identifying levels of interoperability problems and reasoning on possible interoperability problems between different commercial providers of platform as a service. This step starts with studying the existing literature with an aim to find the most important known interoperability problems for a given context. The systematic mapping study or systematic review methods can be used to perform the mentioned review. The final result of the review will be identification of levels of interoperability problems and specific problems on each level. In the platform as a service context, the following levels of interoperability problems were determined: legal, organizational, service level, application

level, and storage level. Next, the ontology of the interoperability problems should be developed using the chosen ontology development methodology such as Ontology Development 101 (33).

Solution design prepares the whole architecture. It includes activities such as the development of the ontology of resources, remote operations and data types, definition of the semantic web service, needed mappings and transformations, and defining AI planning domain. The remote operations of commercial platform as a service, their data types and mappings are modeled by means of the ontology of resources, remote operations, and data type mappings. Current state is described in the ontology presented in Chapter 5.2. However, the landscape of cloud APIs is changing constantly, and the ontology should be upgraded during time. The refinement of the ontology is mandatory when users detect important changes in APIs of included providers and when they want to add a new cloud provider with its new remote functions, data types and new mappings. Next, the language for semantic web services is selected, and after that semantic web services are created by annotating operation, inputs and outputs, data types and needed mappings and transformations. In the end, an AI planner is chosen, and planning domain is created taking into account interoperability actions chosen in previous steps.

Solution implementation deals with approach implementation and execution of the defined use cases. The initial state and goal for AI planner are generated programmatically based on the chosen interoperability action, semantic annotations, the ontology, and defined mappings and transformations. Interoperability tool is developed or upgraded; AI planner is executed to get a plan or list found interoperability problems. If there is a suitable plan, appropriate service compositions are executed, taking into account possible mappings and transformations of inputs and outputs of different services representing remote APIs or composite service consisting of more remote APIs with additional logic.

Evaluation step evaluates the successful execution of use cases and correct identification of possible interoperability problems. If some problems are found, the AI domain and problem definitions, interoperability tool, and semantic annotations should be inspected and errors should be eliminated. Additionally, it is useful to evaluate developed ontologies using known ontology evaluation techniques and methods. The steps of the proposed methodology and their main activities are listed in Table 27.

Table 27 Steps and activities of the proposed methodology

Step	Activities
1. Requirement identification	1.1 Choose cloud model 1.2 Study the existing use cases 1.3 Identification of relevant interoperability actions 1.4 Define use cases
2. Interoperability analysis	2.1 Review the existing literature on interoperability problems 2.2 Identify levels of interoperability problems 2.3 Identify specific interoperability issues at each level 2.4 Choose ontology development methodology 2.5 Create ontology of interoperability problems
3. Solution design	3.1 Create ontology of resources, remote operations, and data types 3.2 Choose language for semantic web services 3.3 Create mappings and transformations 3.4 Associate mappings and transformations to the appropriate elements of services 3.5 Choose AI planner 3.6 Define AI planning domain 3.7 Define algorithms for finding interoperability problems
4. Solution implementation	Use cases execution: 4.1 Implement needed web services to invoke remote APIs 4.2 Generate AI planning problem based on semantic annotations, the ontology and user choice 4.3 Develop or modify/upgrade interoperability tool 4.4 Get a suitable plan from AI planner or find interoperability problems 4.5 Execute service composition
5. Evaluation	5.1 Evaluation of the ontologies 5.2 Validation of execution of use cases

6.5.3 Applying the methodology

The methodology is the result of the work in this dissertation. All listed steps and activities were performed on platform as a service model and two use cases: migration of data between different PaaS storages and adding user to application deployed on other PaaS offers. These two use cases were constructed to illustrate how PaaS storage interoperability and service-level interoperability can be solved using this approach and remote APIs of PaaS providers. The plan for the future is to apply the proposed methodology on additional use cases regarding PaaS interoperability, using other AI planners (for example, some contingent planner to address the non-determinism of the domain), and try to apply it to other two models of cloud computing (IaaS and SaaS). Hopefully, the other researchers will find this methodology useful, and apply it in their research.

7. CONCLUSIONS

The main aim of this dissertation was to advance knowledge of interoperability problems among different commercial vendors of platform as a service, and develop ontologies and methodology to identify and solve interoperability problems among different API operations. In the following subchapters the scientific contribution of the dissertation and review of hypotheses and research questions are presented. The limitations of this study are also brought up, followed by directions for future research.

7.1 Summary of contributions

The main contributions of the dissertation proposal are fulfilled in this work:

7.1.1 Creation of detailed ontologies

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 two use cases to show their practical applicability.

7.1.2 Development of a methodology

Based on use cases, literature review and this research, the new methodology for the detection of interoperability problems among different providers of platform as a service was developed. This methodology uses semantic web annotations, semantic web services, ontology and AI planning method to detect and solve common interoperability problems. Remote PaaS API operations are used to execute interoperability actions.

7.1.3 Solving interoperability problems

In this study, AI planning method was used to identify and try to solve interoperability problems. Practical examples of solving interoperability problems are shown in this dissertation. These approaches were succesful in determining interoperability problems and showed how most common interoperability problems can be solved using semantic web services, cross-PaaS concepts defined in an ontology, and AI planning techniques.

7.2 Answers to research questions

How to semantically describe resources and operations of commercial platform as a service APIs?

The answer to the above question is presented in Chapter 4.1 and Chapter 5.2. The OWL2 ontology was used to semantically describe resources and operations of commercial platform as a service APIs. The aim of the ontology is to clearly describe and categorize the existing functionalities and features of commercial providers of platform as a service. This ontology is used to semantically annotate API operations of platform as a service offers. SAWSDL (W3C's Semantic Annotations for WSDL) lightweight annotation was chosen to define semantic web services.

Which are key indicators of the existence of interoperability problems among the available platform as a service APIs?

Key indicators can be found in the description of classes in Chapter 5.3.4 where classes of the ontology of interoperability problems of platform as a service are presented in Table 18. Classes representing interoperability problems are subclasses of *InteroperabilityProblem* OWL class.

What are the possible solutions to known interoperability problems?

The solutions were presented in earlier chapters on use cases, PaaS ontology, the proposed solution and methodology. Briefly, interoperability problem on PaaS storage level can be

solved by using exported CSV files, custom-built composite web services, PaaS remote APIs, mappings between data types of different PaaS storages defined as instances of the PaaS ontology, and transformations of data from PaaS storage to and from unified data model ontologies. Interoperability problems on service level can be handled by semantically annotating web services using SAWSDL and its lowering and lifting schema mappings coded in XSTL format.

7.3 Hypotheses revisited

H1 Developed ontology will determine the differences among remote application programming interfaces (APIs) of commercial platform as a service providers and improve understanding of platform as a service resources and operations.

Instances in the ontology show different categories of PaaS API operations, data types of input and outputs of the operations, data types supported in different PaaS storage options and data type mappings. The logical consistency of the ontology was checked, it was evaluated by four human experts, and it was succesfully used in two presented use cases. The ontology improves the understanding of PaaS offers, their operations and data type, and enables 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 and help solve known interoperability problems.

H2 Based on the concepts identified in the ontology (resources, operations and interoperability problems), the methodology for determining semantic interoperability problems among the various commercial platform as a service providers and their resolution using the available APIs will be developed.

The methodology for determination and resolution of PaaS interoperability problems was developed as part of this dissertation. This methodology extensively uses elements of ontology to find and solve interoperability problems and to enable data type mappings among PaaS storages and cross-PaaS concepts representing operations and input/output types. The developed ontology is the most important element of the methodology, because other steps

extensively use this ontology. Two use cases illustrate how the metholodogy can be applied to address PaaS interoperability problems.

7.4 Limitations of research

There are several limitations of this work that need to be considered. Real industrial case study proving that it is possible to solve certain interoperability problems by using cloud providers' API may improve validation of this methodology and overall approach. However, it is very difficult to find real (industrial) case studies using more than one PaaS offer or trying to migrate from one PaaS provider to another. Currently, in Croatia, cloud computing usage in general is at its beginning, and the search for the companies that use platform as a service and are willing to cooperate regarding this research was not successful. The small number of companies that use cloud computing paradigm in Croatia use only infrastructure as a service model as a substitution for on-premise solution or previous hosting provider.

Furthermore, AI planning components of this system do 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; 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. Three prominent commercial offers of platform as a service (Google App Engine, Salesforce and Microsoft Azure) were used in use cases presented in this dissertation. 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.

7.5 Open issues and future work

Some possible future research topics could arise by solving limitations of this study listed in the previous section. If the appropriate real (industrial) case study could be found, this approach and methodology to solve it could be applied. In addition to JSHOP2 planner that is used in this approach, this methodology could be upgraded to use some contingent planners to address the non-determinism of the domain. The presented ontology of PaaS resources, remote operations, and data type mappings can be extended including the other providers of

platform as a service. 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 try this approach to solve interoperability problems of other two main models of cloud computing (software as a service and infrastructure as a service). The author will work further on the tool for migration and solving interoperability problems. Generally, the interoperability of platform as a service and cloud computing are very complex and important issues, and hopefully, this dissertation will be a solid foundation for future research in this field.

BIBLIOGRAPHY

1. Chan KSM, Bishop J, Baresi L. Survey and Comparison of Planning Techniques for Web Services Composition [Internet]. Pretoria: Pretoria University; 2007 [cited 2013 Jul 9]. Available from: http://polelo.cs.up.ac.za/papers/Chan-Bishop-Baresi.pdf

2. Mell P, Grance T. The NIST Definition of Cloud Computing [Internet]. NIST; 2011 Sep [cited 2013 Jun 24] p. 7. Report No.: 800-145. Available from: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

3. Erl T. Cloud computing: concepts, technology, & architecture. Upper Saddle River, NJ: Prentice Hall; 2013. 487 p.

4. Velte AT. Cloud computing: a practical approach. New York: McGraw-Hill; 2010. 334 p.

5. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, et al. Above the Clouds: A Berkeley View of Cloud Computing [Internet]. Berkeley: EECS Department, University of California, Berkeley; 2009 Feb [cited 2013 Jun 4] p. 23. Report No.: EECS-2009-28. Available from: http://d1smfj0g31qzek.cloudfront.net/abovetheclouds.pdf

6. Wang L, von Laszewski G, Kunze M, Tao J. Cloud computing: A Perspective study. Proceedings of the Grid Computing Environments (GCE) workshop [Internet]. Austin, Texas; 2008 [cited 2013 Jun 24]. Available from: https://ritdml.rit.edu/bitstream/handle/1850/7821/LWangConfProc11-16-2008.pdf?sequence=1

7. Vaquero LM, Rodero-Merino L, Caceres J, Lindner M. A break in the clouds. ACM SIGCOMM Computer Communication Review. 2008 Dec 31;39(1):50.

8. Boniface M, Nasser B, Papay J, Phillips SC, Servin A, Yang X, et al. Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds. Fifth International Conference on Internet and Web Applications and Services (ICIW 2010)

[Internet]. Barcelona: IEEE; 2010 [cited 2013 Aug 6]. p. 155–60. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5476775

9. Emison JM. PaaS Buyer's Guide. InformationWeek; 2013 Mar.

10. Lawton G. Developing Software Online With Platform-as-a-Service Technology. Computer. 2008 Jun;41(6):13–5.

11. IEEE. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries [Internet]. New York; 1991 [cited 2013 Jun 27]. Report No.: 610-1991. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=182763

12. Brownsword LL, Carney DJ, Fisher D, Meyers C, Morris EJ, Place PRH, et al. Current Perspectives on Interoperability [Internet]. Pittsburgh: Carnegie Mellon University; 2004 Mar [cited 2014 Mar 25] p. 1–51. Report No.: CMU/SEI-2004-TR-009. Available from: http://resources.sei.cmu.edu/asset_files/TechnicalReport/2004_005_001_14390.pdf

13. Pokraev S, Quartel D, Steen MWA, Reichert M. Semantic Service Modeling: Enabling System Interoperability. In: Doumeingts G, Müller J, Morel G, Vallespir B, editors. Enterprise Interoperability [Internet]. London: Springer London; 2007 [cited 2013 Jun 27]. p. 221–30. Available from: http://www.springerlink.com/index/10.1007/978-1-84628-714-5_21

14. Vernadat F. Enterprise modeling and integration: principles and applications. London ; New York: Chapman & Hall; 1996. 513 p.

15. Park J, Ram S. Information systems interoperability. ACM Transactions on Information Systems. 2004 Oct 1;22(4):595–632.

16. Loutas N, Kamateri E, Tarabanis K, D'Andria F. D 1.2 Cloud4SOA Cloud Semantic Interoperability Framework [Internet]. 2011 [cited 2013 Jun 26]. Available from: http://www.cloud4soa.eu/sites/default/files/D1.2_Cloud4SOA%20Cloud%20Semantic%20Interoperability%20Framework.pdf

17. Naudet Y, Latour T, Guedria W, Chen D. Towards a systemic formalisation of interoperability. Computers in Industry. 2010 Feb;61(2):176–85.

18. Machado GS, Hausheer D, Stiller B. Considerations on the interoperability of and between cloud computing standards. Procs OGF27: G2C-Net [Internet]. Banff, Alberta, Canada; 12-15 October2009 [cited 2013 Jul 3]. Available from: http://www.csg.uzh.ch/publications/ogf27-g2cnet-discussion-cc-standards-finalversion.pdf

19. Pahl C, Zhang L, Fowley F. Interoperability Standards for Cloud Architecture. 3rd International Conference on Cloud Computing and Services Science, CLOSER 2013 [Internet]. Aachen, Germany; 2013 [cited 2013 Jul 3]. Available from: http://doras.dcu.ie/17824/1/closer13-sp.pdf

20. Pahl C, Zhang L, Fowley F. A look at cloud architecture interoperability through standards. The Fourth International Conference on Cloud Computing, Grids, and Virtualization [Internet]. Valenica, Spain; 2013 [cited 2013 Jul 3]. p. 7–12. Available from: http://doras.dcu.ie/18355/1/CloudComp13-Std.pdf

21. Nyren R, Edmonds A, Papaspyrou A, Metsch T. Open Cloud Computing Interface - Core [Internet]. Open Grid Forum; 2011 Jun [cited 2013 Jul 3]. Report No.: GFD-P-R.183. Available from: http://www.ogf.org/documents/GFD.183.pdf

22. DMTF. Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP based Protocol 6 - An Interface for Managing Cloud Infrastructure [Internet]. Distributed Manage ment Task Force, Inc. (DMTF); 2012 Sep [cited 2013 Jul 3]. Available from: http://www.dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf

23. DMTF. Open Virtualization Format Specification [Internet]. DMTF; 2012 Dec [cited 2013 Jul 3]. Report No.: DSP0243. Available from: http://dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0.pdf

24. OASIS. Topology and Orchestration Specification for Cloud Applications Version 1.0 [Internet]. OASIS; 2013 Mar [cited 2013 Jul 3]. Report No.: Committee Specification

01. Available from: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf

25. SNIA. Cloud Data Management Interface (CDMI™) Version 1.0.2 [Internet]. SNIA; 2012 Jun [cited 2013 Jul 3]. Available from: http://snia.org/sites/default/files/CDMI%20v1.0.2.pdf

26. Lewis GA. The Role of Standards in Cloud Computing Interoperability [Internet]. Carnegie Mellon University; 2012 Oct [cited 2013 Jul 5]. Report No.: Paper 682. Available from: http://www.sei.cmu.edu/reports/12tn012.pdf

27. Petcu D. Portability and Interoperability between Clouds: Challenges and Case Study. In: Abramowicz W, Llorente IM, Surridge M, Zisman A, Vayssière J, editors. Towards a Service-Based Internet [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011 [cited 2013 Nov 26]. p. 62–74. Available from: http://www.springerlink.com/index/10.1007/978-3-642-24755-2_6

28. Berners-Lee T, Hendler J, Lassila O. The Semantic Web. Scientific American. 2001 May;284(5):29–37.

29. Allemang D. Semantic Web for the working ontologist: effective modeling in RDFS and OWL. 2nd ed. Waltham, MA: Morgan Kaufmann/Elsevier; 2011. 354 p.

30. W3C. RDF Primer [Internet]. 2004 Feb [cited 2013 Jun 29]. Available from: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

31. W3C. OWL 2 Web Ontology Language Primer [Internet]. 2009 Oct [cited 2013 Jun 29]. Available from: http://www.w3.org/TR/2009/REC-owl2-primer-20091027/

32. Gruber TR. A translation approach to portable ontology specifications. Knowledge Acquisition. 1993 Jun;5(2):199–220.

33. Noy NF, McGuinness DL. Ontology Development 101: A Guide to Creating Your First Ontology [Internet]. Stanford University; 2001 [cited 2013 Jun 30]. Available from: http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf

34. Vrandečić D. Ontology Evaluation [Internet] [PhD thesis]. [Karlsruhe]: Karlsruher Instituts fuer Technologie (KIT); 2010 [cited 2013 Sep 19]. Available from: http://www.aifb.kit.edu/images/b/b5/OntologyEvaluation.pdf

35. Bergman M. A Brief Survey of Ontology Development Methodologies [Internet]. 2010 [cited 2013 Jun 29]. Available from: http://www.mkbergman.com/906/a-brief-survey-of-ontology-development-methodologies/

36. Corcho O, Fernández-López M, Gómez-Pérez A. Methodologies, tools and languages for building ontologies. Where is their meeting point? Data & Knowledge Engineering. 2003 Jul;46(1):41–64.

37. Fernandez M, Gomez-Perez A, Juristo N. METHONTOLOGY: From Ontological Art towards Ontological Engineering. Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering. Standford, USA; 1997. p. 33–40.

38. De Nicola A, Missikoff M, Navigli R. A proposal for a unified process for ontology building: UPON. DEXA'05 Proceedings of the 16th international conference on Database and Expert Systems Applications. Copenhagen, Denmark: Springer; 2005. p. 655–64.

39. Iqbal R, Murad MAA, Mustapha A, Sharef NMS. An Analysis of Ontology Engineering Methodologies: A Literature Review. Research Journal of Applied Sciences, Engineering and Technology. 2013;6(16):2993–3000.

40. Papazoglou MP, Heuvel W-J. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal. 2007 Mar 3;16(3):389–415.

41. Austin D, Barbir A, Ferris C, Garg S. Web Services Architecture Requirements [Internet]. W3C; 2004 Feb [cited 2014 Feb 10]. Available from: http://www.w3.org/TR/wsa-reqs/

42. Pautasso C, Zimmermann O, Leymann F. Restful web services vs. "big"' web services: making the right architectural decision. ACM Press; 2008 [cited 2014 Feb 8]. p. 805. Available from: http://portal.acm.org/citation.cfm?doid=1367497.1367606

43. Pautasso C. RESTful Web service composition with BPEL for REST. Data & Knowledge Engineering. 2009 Sep;68(9):851–66.

44. Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, et al. OWL-S: Semantic Markup for Web Services [Internet]. 2004 Nov [cited 2013 Jun 29]. Available from: http://www.w3.org/Submission/OWL-S/

45. Roman D, Lausen H, Keller U, de Bruijn J, Bussler C, Domingue J, et al. D2v1.4. Web Service Modeling Ontology (WSMO) [Internet]. 2007 Feb p. 29 June 2013. Available from: http://www.wsmo.org/TR/d2/v1.4/

46. Fensel D, Kopecky J, Komazec S. Leight-weight Annotations [Internet]. 2010 [cited 2013 Jun 29]. Available from: http://www.sti-innsbruck.at/sites/default/files/fileadmin/documents/SWS_SS11/SWS-Lecture11-handouts.pdf

47. Pedrinaci C. Lightweight Semantic Annotations for Services on the Web [Internet]. 2009 [cited 2013 Jun 29]. Available from: http://soa4all.eu/training/SOA4All-FirstExternalTutorialSlideSet.pdf

48. Vitvar T, Kopecky J, Viskova J, Mocan A, Kerrigan M, Fensel D. Chapter 5 Semantic Web Services Architecture with Lightweight Descriptions of Services. Advances in Computers [Internet]. Elsevier; 2009 [cited 2013 Jun 29]. p. 177–224. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0065245809010055

49. W3C. Semantic Annotations for WSDL and XML Schema [Internet]. W3C; 2007 Aug [cited 2013 Jun 29]. Available from: http://www.w3.org/TR/sawsdl/

50. Verma K. Configuration and adaptation of semantic web processes [Internet] [Doctoral thesis]. [Athens, Georgia]: University of Georgia; 2006 [cited 2014 Feb 5]. Available from: http://athenaeum.libs.uga.edu/bitstream/handle/10724/9083/verma_kunal_200608_phd.pdf?sequence=1

51. Ghallab M. Automated planning: theory and practice. Amsterdam ; Boston: Elsevier/Morgan Kaufmann; 2004. 635 p.

52. Okutan C, Cicekli NK. A monolithic approach to automated composition of semantic web services with the Event Calculus. Knowledge-Based Systems. 2010 Jul;23(5):440–54.

53. Yoo T, Jeong B, Cho H. A Petri Nets based functional validation for services composition. Expert Systems with Applications. 2010 May;37(5):3768–76.

54. Ni Y, Fan Y. Model transformation and formal verification for Semantic Web Services composition. Advances in Engineering Software. 2010 Jun;41(6):879–85.

55. Xu C, Qu W, Wang H, Wang Z, Ban X. A Petri Net-Based Method for Data Validation of Web Services Composition. IEEE 34th Annual Computer Software and Applications Conference (COMPSAC 2010) [Internet]. Seoul: IEEE; 2010 [cited 2013 Jul 10]. p. 468–76. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5676297

56. Rao J, Küngas P, Matskin M. Composition of Semantic Web services using Linear Logic theorem proving. Information Systems. 2006 Jun;31(4-5):340–60.

57. McDermott D. Estimated-Regression Planning for Interactions with Web Services. Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems [Internet]. Toulose, France: AAAI; 2002 [cited 2013 Jul 10]. p. 204–11. Available from: ftp://cs.yale.edu/pub/mcdermott/papers/aips02.pdf

58. Sirin E, Parsia B, Wu D, Hendler J, Nau D. HTN planning for Web Service composition using SHOP2. Web Semantics: Science, Services and Agents on the World Wide Web. 2004 Oct;1(4):377–96.

59. Bertoli P, Pistore M, Traverso P. Automated composition of Web services via planning in asynchronous domains. Artificial Intelligence. 2010 Mar;174(3-4):316–61.

60. Hatzi O, Vrakas D, Nikolaidou M, Bassiliades N, Anagnostopoulos D, Vlahavas I. An Integrated Approach to Automated Semantic Web Service Composition through Planning. IEEE Transactions on Services Computing. 2012;5(3):319–32.

61. Goyal A. Real-Time Planning Using HTN in Stealth Game [Internet]. 2010 [cited 2013 Jul 5]. Available from: http://www.anshulgo.com/download/AnshulGoyal_AIProjectPaper_April2010.pdf

62. Yang T-H, Lee W-P. A Service-Oriented Framework for the Development of Home Robots. International Journal of Advanced Robotic Systems. 2013;10(122):1–11.

63. European Commission. European Interoperability Framework (EIF) for European public services [Internet]. European Commission; 2010 [cited 2014 Jul 7]. Available from: http://ec.europa.eu/isa/documents/isa_annex_ii_eif_en.pdf

64. Chen D, Doumeingts G, Vernadat F. Architectures for enterprise integration and interoperability: Past, present and future. Computers in Industry. 2008 Sep;59(7):647–59.

65. Berre A-J, Elvesæter B, Figay N, Guglielmina C, Johnsen SG, Karlsen D, et al. The ATHENA Interoperability Framework. In: Gonçalves RJ, Müller JP, Mertins K, Zelm M, editors. Enterprise Interoperability II [Internet]. London: Springer London; [cited 2013 Aug 3]. p. 569–80. Available from: http://www.springerlink.com/index/10.1007/978-1-84628-858-6_62

66. The GridWise Architecture Council. GridWise Interoperability Context-Setting Framework [Internet]. The GridWise Architecture Council; 2008 [cited 2014 Jul 7]. Available from: http://www.gridwiseac.org/pdfs/interopframework_v1_1.pdf

67. Rosati K, Lamar M. The Quest for Interoperable Electronic Health Records: A Guide to Legal Issues in Establishing Health Information Networks [Internet]. American Helath Lawyers Association; 2005 [cited 2014 Jul 8]. Available from: http://www.crowell.com/pdf/2005-July_Quest_for_EHRs_Butler.pdf

68. Hellman R. Organizational Barriers to Interoperability: Norwegian Case Study. 2009 Proceedings of ongoing research, general development issues and projects of EGOV 09 8th International Conference. Linz, Austria; 2009. p. 182–9.

69. Rauffet P, Cunha CD, Bernard A. Designing and Managing Organizational Interoperability with Organizational Capabilities and Roadmaps. IESA ’09 International Conference on Interoperability for Enterprise Software and Applications

[Internet]. Beijing, China: IEEE; 2009 [cited 2014 Jul 8]. p. 120–6. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5260854

70. Rana J, Ion M. Challenges of Interoperability Issues for Enterprise Software and Applications [Internet]. OSCO Team, CREATE-NET; 2007 [cited 2014 Jul 8]. Available from: ftp://ftp.cordis.europa.eu/pub/ist/docs/ict-ent-net/eivp-create_en.pdf

71. Vernadat FB. Technical, semantic and organizational issues of enterprise interoperability and networking. Annual Reviews in Control. 2010 Apr;34(1):139–44.

72. Sheth AP, Kashyap V. So Far (Schematically) yet So Near (Semantically). Proceedings of the IFIP WG 26 Database Semantics Conference on Interoperable Database Systems. North-Holland Publishing Co.; 1993. p. 283–312.

73. Parent C, Spaccapietra S. Database Integration: the Key to Data Interoperability. Advances in Object-Oriented Data Modeling. MIT Press; 2000.

74. Haslhofer B, Klas W. A survey of techniques for achieving metadata interoperability. ACM Computing Surveys. 2010 Feb 1;42(2):1–37.

75. Ponnekanti SR, Fox A. Interoperability among independently evolving web services. Middleware ’04 Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware. Toronto, Canada: Springer; 2004. p. 331–51.

76. Zhu F, Turner M, Kotsiopoulos I, Bennett K, Russel M, Budgen D, et al. Dynamic Data Integration Using Web Services. ICWS ’04 Proceedings of the IEEE International Conference on Web Services. San Diego, USA: IEEE Computer Society; 2004. p. 262–72.

77. Nagarajan M, Verma K, Sheth AP, Miller JA. Ontology Driven Data Mediation in Web Services. International Journal of Web Services Research. 2007 34;4(4):104–26.

78. Rezaei R, Chiew TK, Lee SP, Aliee ZS. A Semantic Interoperability Framework for Software as a Service Systems in Cloud Computing Environments. Expert Systems with Applications. 2014 Mar;41(13):5751–70.

79. Tao J, Marten H, Kramer D, Karl W. An Intuitive Framework for Accessing Computing Clouds. Procedia Computer Science. 2011 Jan;4:2049–57.

80. Rodero-Merino L, Vaquero LM, Gil V, Galán F, Fontán J, Montero RS, et al. From infrastructure delivery to service management in clouds. Future Generation Computer Systems. 2010 Oct;26(8):1226–40.

81. Ranabahu A, Maximilien EM. A Best Practice Model for Cloud Middleware Systems. Proceedings of the Best Practices in Cloud Computing: Designing for the Cloud workshop in ACG SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). Orlando FL, USA; 2009. p. 41–51.

82. Bernstein D, Vij D. Intercloud Directory and Exchange Protocol Detail Using XMPP and RDF. 6th World Congress on Services (SERVICES-1 2010) [Internet]. Miami, Florida: IEEE; 2010 [cited 2013 Jun 28]. p. 431–8. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5577272

83. Merzky A, Stamou K, Jha S. Application Level Interoperability between Clouds and Grids. Workshops at the Grid and Pervasive Computing Conference (GPC '09) [Internet]. Geneva: IEEE; 2009 [cited 2013 Jun 28]. p. 143–50. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4976556

84. 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) [Internet]. Indianapolis: IEEE; 2010 [cited 2013 Jun 28]. p. 234–41. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708456

85. Buyya R, Ranjan R, Calheiros RN. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In: Hsu C-H, Yang LT, Park JH, Yeo S-S, editors. Algorithms and Architectures for Parallel Processing [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010 [cited 2013 Jun 28]. p. 13–31. Available from: http://www.springerlink.com/index/10.1007/978-3-642-13119-6_2

86. Loutas N, Peristeras V, Bouras T, Kamateri E, Zeginis D, Tarabanis K. Towards a Reference Architecture for Semantically Interoperable Clouds. IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)

[Internet]. Indianapolis: IEEE; 2010 [cited 2013 Jun 28]. p. 143–50. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708445

87. Demchenko Y, Makkes MX, Strijkers R, de Laat C. Intercloud Architecture for interoperability and integration. 4th International Conference on Cloud Computing TEchnology and Science (CloudCom 2012) [Internet]. Taipei: IEEE; 2012 [cited 2013 Jul 5]. p. 666–74. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6427607

88. Fazai S. Three-Dimensional Space to Assess Cloud Interoperability [Internet] [Master's thesis]. [Monterey, CA]: Naval Postgraduate School; 2013 [cited 2013 Jul 5]. Available from: http://calhoun.nps.edu/public/handle/10945/32818

89. Miranda J, Murillo JM, Guillén J, Canal C. Identifying adaptation needs to avoid the vendor lock-in effect in the deployment of cloud SBAs. Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12) [Internet]. Bertinoro, Italy: ACM Press; 2012 [cited 2013 Jul 5]. p. 12. Available from: http://dl.acm.org/citation.cfm?doid=2377836.2377841

90. Bhukya DP, Sony R, Muduganti G. On Web Services Based Cloud Interoperability. IJCSI. 2012 Sep;9(5):232–6.

91. Bastião Silva LA, Costa C, Oliveira JL. A common API for delivering services over multi-vendor cloud resources. Journal of Systems and Software. 2013 Apr;86(9):2309–17.

92. Ma H, Schewe K-D, Thalheim B, Wang Q. A formal model for the interoperability of service clouds. Service Oriented Computing and Applications. 2012 Jan 18;6(3):189–205.

93. Khalfallah M, Barhamgi M, Figay N, Ghodous P. A Novel Approach to Ensure Interoperability Based on a Cloud Infrastructure. In: Stjepandić J, Rock G, Bil C, editors. Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment [Internet]. London: Springer London; 2013 [cited

2013 Nov 26]. p. 1143–54. Available from: http://www.springerlink.com/index/10.1007/978-1-4471-4426-7_96

94. Guillén J, Miranda J, Murillo JM, Canal C. A service-oriented framework for developing cross cloud migratable software. Journal of Systems and Software. 2013 Sep;86(9):2294–308.

95. The Apache Software Foundation. Welcome to Apache Libcloud's documentation! [Internet]. 2013 [cited 2013 Oct 23]. Available from: https://ci.apache.org/projects/libcloud/docs/#main

96. Apache. About Deltacloud [Internet]. 2013 [cited 2013 Oct 23]. Available from: http://deltacloud.apache.org/about.html

97. Apache. What is Apache jClouds? [Internet]. 2013 [cited 2013 Oct 23]. Available from: http://jclouds.incubator.apache.org/documentation/gettingstarted/what-is-jclouds/

98. Petcu D, Di Martino B, Venticinque S, Rak M, Máhr T, Esnal Lopez G, et al. Experiences in building a mOSAIC of clouds. Journal of Cloud Computing: Advances, Systems and Applications. 2013;2(1):12.

99. Kamateri E, Loutas N, Zeginis D, Ahtes J, D'Andria F, Bocconi S, et al. Cloud4SOA: A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability. In: Lau K-K, Lamersdorf W, Pimentel E, editors. Service-Oriented and Cloud Computing [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013 [cited 2013 Sep 18]. p. 64–78. Available from: http://link.springer.com/10.1007/978-3-642-40651-5_6

100. Petcu D, Macariu G, Panica S, Crăciun C. Portable Cloud applications—From theory to practice. Future Generation Computer Systems. 2013 Aug;29(6):1417–30.

101. Yegou Y, Artac M, Temporale C, Cascella R. First Specification of the System Architecture D10.1 [Internet]. INRIA; 2011 Aug [cited 2013 Jun 28]. Report No.: D10.1. Available from: http://contrail-project.eu/documents/18553/136157/D10.1.pdf

102. Carlini E, Coppola M, Dazzi P, Ricci L, Righetti G. Cloud Federations in Contrail. In: Alexander M, D'Ambra P, Belloum A, Bosilca G, Cannataro M, Danelutto M, et al.,

editors. Euro-Par 2011: Parallel Processing Workshops [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012 [cited 2013 Aug 9]. p. 159–68. Available from: http://www.springerlink.com/index/10.1007/978-3-642-29737-3_19

103. Contrail consortium. Contrail White Paper - Overview of the Contrail system, components and usage [Internet]. Contrail consortium; 2012 Nov [cited 2013 Aug 10]. Available from: http://contrail-project.eu/documents/18553/341152/WhitePaper.pdf

104. Gogouvitis SV, Kousiouris G, Vafiadis G, Kolodner EK, Kyriazis D. OPTIMIS and VISION Cloud: How to Manage Data in Clouds. In: Alexander M, D'Ambra P, Belloum A, Bosilca G, Cannataro M, Danelutto M, et al., editors. Euro-Par 2011: Parallel Processing Workshops [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012 [cited 2013 Jun 26]. p. 35–44. Available from: http://www.springerlink.com/index/10.1007/978-3-642-29737-3_5

105. Grozev N, Buyya R. Inter-Cloud architectures and application brokering: taxonomy and survey. Software: Practice and Experience. 2012 Dec;44(3):369–90.

106. Vernik G, Shulman-Peleg A, Dippl S, Formisano C, Jaeger MC, Kolodner EK, et al. Data On-Boarding in Federated Storage Clouds. IEEE Sixth International Conference on Cloud Computing (CLOUD 2013) [Internet]. Santa Clara, California: IEEE; 2013 [cited 2014 Jan 28]. p. 244–51. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6676701

107. Mohagheghi P, Sæther T. Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project. IEEE World Congress on Services (SERVICES 2011) [Internet]. Washington, DC: IEEE; 2011 [cited 2013 Jun 28]. p. 507–14. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6012736

108. Benguria G, Mohagheghi P, Gomez Y, Hein C, Morin B. Deliverable D.2.2 REMICS Methodology, Interim Release [Internet]. REMICS Consortium; 2011 [cited 2013 Sep 18]. Available from: http://www.remics.eu/system/files/REMICS_D2.2_V1.0.pdf

109. SOFTEAM, SINTEF, Tecnalia. REMICS Deliverable D4.1 PIM4Cloud [Internet]. REMICS Consortium; 2012 Mar [cited 2013 Nov 19] p. 1–98. Available from: http://www.remics.eu/system/files/REMICS_D4.1_V2.0_LowResolution.pdf

110. Ardagna D, Di Nitto E, Casale G, Petcu D, Mohagheghi P, Mosser S, et al. MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. Procs MISE 2012. Zurich, Switzerland: IEEE; 2012. p. 50–6.

111. Loutas N, Kamateri E, Zotou M, Zeginis D, Tarabanis K, Bocconi S, et al. D1.1 Requirements Analysis Report [Internet]. 2011 Feb [cited 2013 Jul 1]. Available from: http://www.cloud4soa.eu/sites/default/files/Cloud4SOA%20D1.1%20Requirements%20Analysis.pdf

112. CONTRAIL. Description of applications, use cases and requirements D12.1 [Internet]. STFC; 2011 Jun [cited 2013 Jul 1]. Available from: http://contrail-project.eu/documents/18553/136157/D12.1.pdf

113. Petcu D, Sandru C, Di Martino B, Venticinque S, Rak M, Aversa R, et al. D 1.1 - Architectural Design of the mOSAIC's API and Platform [Internet]. 2011 Dec. Available from: http://www.mosaic-cloud.eu/index.php?option=com_chronocontact&Itemid=186

114. Allalouf M, Averbuch A, Bonelli L, Brand P, Chevalier G, Dao M, et al. Deliverable D10.2 - High Level Architectural Specification - Release 1.0 [Internet]. 2011 Jul [cited 2013 Jul 1]. Available from: http://visioncloud.eu/resource.php?resourceID=193

115. Badger L, Bohn R, Chandramouli R, Grance T, Karygiannis T, Patt-Corner R, et al. Cloud Computing Use Cases [Internet]. NIST; 2010 [cited 2013 Jul 1]. Available from: http://www.nist.gov/itl/cloud/use-cases.cfm

116. Ahronovitz M, Amrhein D, Anderson P, de Andrade A, Armstrong J, Arasan E, et al. Cloud Computing Use Cases White Paper Version 4.0 [Internet]. 2010 Jul [cited 2013 Jul 1]. Available from: http://opencloudmanifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf

117. Microsoft IEC. Cloud Computing Use Cases [Internet]. Microsoft; 2011 Aug [cited 2013 Jul 1]. Available from: http://www.google.hr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CEAQFjAA&url=http%3A%2F%2Fdownload.microsoft.com%2Fdownload%2FA%2FD%2FD%2FADDE6C9B-FD11-4E8A-959F-DF796FAD0749%2FIEC_Cloud_Computing_Use_Cases-Aug_2011.pdf&ei=MjfRUd-sEdHHswbC54DwAg&usg=AFQjCNFcM8FQvyd_2dN3mdk9hjZQ4W0pPA&sig2=lWzxOlPPbGZu1ctmMzj-DQ&bvm=bv.48572450,d.Yms&cad=rja

118. ATHENA Consortium. Interoperability methodology [Internet]. ATHENA Consortium; 2006 [cited 2013 Oct 10]. Available from: http://athena.modelbased.net/methodology/index.pdf

119. Jacobson I, Booch G, Rumbaugh J. The Unified Software Development Process. Reading, Massachusetts: Addison-Wesley; 1999. 512 p.

120. Chen D, Daclin N. Barriers Driven Methodology for Enterprise Interoperability. In: Camarinha-Matos LM, Afsarmanesh H, Novais P, Analide C, editors. Establishing the Foundation of Collaborative Networks [Internet]. Boston, MA: Springer US; 2007 [cited 2013 Oct 10]. p. 453–60. Available from: http://link.springer.com/10.1007/978-0-387-73798-0_48

121. Sanati F, Lu J, Zeng X. A methodological Framework for E-government Service Delivery Integration. eGovernment Interoperability Campus [Internet]. Paris; 2007 [cited 2013 Oct 10]. Available from: http://80.14.185.155/egovinterop/egov07cd/eGov07-CDROM/pages /papers/ T1C.pdf

122. Nagarajan M, Verma K, Sheth A, Miller J, Lathem J. Semantic Interoperability of Web Services - Challenges and Experiences. IEEE; 2006 [cited 2014 Sep 1]. p. 373–82. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4032048

123. Sheth AP, Gomadam K, Ranabahu A. Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST. IEEE Data Eng Bull. 2008;31(3):8–12.

124. Klímek J, Necaský M. Generating Lowering and Lifting Schema Mappings for Semantic Web Services. IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA 2011) [Internet]. IEEE; 2011 [cited 2014 Sep 2]. p. 29–34. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5763433

125. Li X, Madnick S, Zhu H, Fan Y. Reconciling Semantic Heterogeneity in Web Services Composition [Internet]. Massachusetts Institute of Technology; 2009 Sep [cited 2014 Sep 2] p. 1–17. Report No.: CISL#2009-08. Available from: http://dspace.mit.edu/bitstream/handle/1721.1/66542/SSRN-id1478025.pdf?sequence=1

126. Stollberg M, Cimpian E, Mocan A, Fensel D. A Semantic Web Mediation Architecture. In: Koné MT, Lemire D, editors. Canadian Semantic Web [Internet]. Springer US; 2006 [cited 2014 Sep 2]. p. 3–22. Available from: http://www.springerlink.com/index/10.1007/978-0-387-34347-1

127. Youseff L, Butrico M, Da Silva D. Toward a Unified Ontology of Cloud Computing. GCE ’08 Grid Computing Environments Workshop [Internet]. Austin, Texas: IEEE; 2008 [cited 2013 Jul 9]. p. 1–10. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4738443

128. Weinhardt C, Anandasivam A, Blau B, Stosser J. Business Models in the Service World. IT Professional. 2009 Mar;11(2):28–33.

129. Deng Y, Head MR, Kochut A, Munson J, Sailer A, Shaikh H. Introducing Semantics to Cloud Services Catalogs. IEEE International Conference on Services Computing (SCC 2011) [Internet]. Washington, DC: IEEE; 2011 [cited 2013 Jul 9]. p. 24–31. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6009240

130. Takahashi T, Kadobayashi Y, Fujiwara H. Ontological approach toward cybersecurity in cloud computing. Proceedings of the 3rd international conference on security of information and networks (SIN ’10) [Internet]. Rostov-on-Don, Russian Federation: ACM Press; 2010 [cited 2013 Jul 9]. p. 100. Available from: http://portal.acm.org/citation.cfm?doid=1854099.1854121

131. Adrian Martinez C, Isaza Echeverri G, Castillo Sanz AG. Malware detection based on Cloud Computing integrating Intrusion Ontology representation. IEEE Latin-American Conference on Communications (LATINCOM 2010) [Internet]. Bogota: IEEE; 2010 [cited 2013 Jul 9]. p. 1–6. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5641013

132. Moscato F, Aversa R, Di Martino B, Fortis T-F, Munteanu V. An Analysis of mOSAIC ontology for Cloud Resources annotation. Proceedings of the Federated Conference on Computer Science and Information Systems. Szczecin; 2011. p. 973–80.

133. Han T, Sim KM. An Ontology-enhanced Cloud Service Discovery System. Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 [Internet]. Hong Kong; 2010 [cited 2013 Jul 9]. p. 644–9. Available from: http://www.iaeng.org/publication/IMECS2010/IMECS2010_pp644-649.pdf

134. Kang J, Sim KM. Ontology and search engine for cloud computing system. International Conference on System Science and Engineering (ICSSE 2011) [Internet]. Macao: IEEE; 2011 [cited 2013 Jul 9]. p. 276–81. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5961913

135. Dastjerdi AV, Tabatabaei SGH, Buyya R. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010) [Internet]. Melbourne, Australia: IEEE; 2010 [cited 2013 Jul 9]. p. 104–12. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5493487

136. Ma H, Schewe K-D, Wang Q. An abstract model for service provision, search and composition. IEEE Asia-Pacific Services Computing Conference (APSCC 2009) [Internet]. Singapore: IEEE; 2009 [cited 2013 Jul 9]. p. 95–102. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5394133

137. Poveda-Villalón M, Suárez-Figueroa MC, Gómez-Pérez A. A Double Classification of Common Pitfalls in Ontologies. Proceedings of Workshop on Ontology Quality (OntoQual 2010), Co-located with EKAW 2010. Lisbon, Portugal; 2010.

138. Poveda-Villalón M, Suárez-Figueroa MC, Gomez-Perez, Asuncion A. Validating ontologies with OOPS! EKAW'12 Proceedings of the 18th international conference on Knowledge Engineering and Knowledge Management. Galway City, Ireland: Springer-Verlag; 2012. p. 267–81.

139. Baumeister J, Seipel D. Anomalies in ontologies with rules. Web Semantics: Science, Services and Agents on the World Wide Web. 2010 Mar;8(1):55–68.

140. Gómez-Pérez A. Ontology Evaluation. In: Staab S, Studer R, editors. Handbook on Ontologies [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004 [cited 2013 Sep 20]. p. 251–73. Available from: http://link.springer.com/10.1007/978-3-540-24750-0_13

141. Lovrenčić S, Čubrilo M. Ontology Evaluation – Comprising Verification and Validation. Proceedings of the 19th Central European Conference on Information and Intelligent Systems. Varaždin: Faculty of organization and informatics; 2008. p. 657–63.

142. Brank J, Grobelnik M, Mladenić D. A survey of ontology evaluation techniques. Proceedings of the Conference on Data Mining and Data Warehouses SiKDD 2005 [Internet]. Ljubljana, Slovenia; 2005. Available from: http://ailab.ijs.si/dunja/sikdd2005/Papers/BrankEvaluationSiKDD2005.pdf

143. Amirhosseini M, Salim J. OntoAbsolute as a ontology evaluation methodology in analysis of the structural domains in upper, middle and lower level ontologies. International Conference on Semantic Technology and Information Retrieval (STAIR 2011) [Internet]. Putrajaya: IEEE; 2011 [cited 2014 Jul 11]. p. 26–33. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5995760

144. Weber I, Wada H, Fekete A, Liu A, Bass L. Automatic Undo for Cloud Management via AI Planning. Eight Workshop on Hot Topics in System Dependability: HotDep '12 [Internet]. Hollywood, CA; 2012 [cited 2013 Jul 5]. Available from: https://www.usenix.org/system/files/conference/hotdep12/hotdep12-final4.pdf

145. Zou G, Chen Y, Xiang Y, Huang R, Xu Y. AI Planning and Combinatorial Optimization for Web Service Composition in Cloud Computing. CCV Conference

[Internet]. Singapore; 2010 [cited 2013 Jul 5]. Available from: http://www.cse.wustl.edu/~ychen/public/AI%20Planning%20and%20Combinatorial%20Optimization%20for%20Web%20Service%20Composition%20in%20Cloud%20Computing.pdf

146. 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) [Internet]. Toronto, Canada: The AAAI Press; 2010 [cited 2013 Aug 27]. p. 81–8. Available from: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1453/1532

147. Kungas P, Matskin M. Detection of Missing Web Services: The Partial Deduction Approach. Proceedings on the International Conference on Next Generation Web Services Practices (NWeSP 2005) [Internet]. Seoul, Korea: IEEE; 2005 [cited 2014 Aug 4]. p. 339–44. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1592450

148. Yan Y, Poizat P, Zhao L. Repair vs. Recomposition for Broken Service Compositions. Proceedings of the 8th International Conference ICSOC 2010. San Francisco, USA: Springer Berlin Heidelberg; 2010. p. 152–66.

149. Vukovic M, Robinson P. GoalMorph: Partial Goal Satisfaction for Flexible Service Composition. International Conference on Next Generation Web Services Practices (NWeSP 2005) [Internet]. Seoul, Korea: IEEE; 2005 [cited 2014 Aug 4]. p. 149–54. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1592421

150. Friedrich G, Fugini M, Mussi E, Pernici B, Tagni G. Exception Handling for Repair in Service-Based Processes. IEEE Transactions on Software Engineering. 2010 Mar;36(2):198–215.

151. Petersen K, Feldt R, Mujtaba S, Mattsson M. Systematic mapping studies in software engineering. Proceedings of the 12th international conference on Evaluation and Assessment in Software (EASE ’08). Bari, Italy; 2008.

152. Dowell S, Barreto A, Michael JB, Shing M-T. Cloud to cloud interoperability. 6th International Conference on System of Systems Engineering (SoSE 2011) [Internet]. Albuquerque: IEEE; 2011 [cited 2014 Jul 18]. p. 258–63. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5966607

153. Martino BD, Cretella G, Esposito A. Semantic and Agnostic Representation of Cloud Patterns for Cloud Interoperability and Portability. IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013) [Internet]. Bristol: IEEE; 2013 [cited 2014 Jul 18]. p. 182–7. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6735416

154. Petcu D, Craciun C, Neagul M, Lazcanotegui I, Rak M. Building an interoperability API for Sky computing. International Conference on High Performance Computing and Simulation (HPCS 2011) [Internet]. Istanbul: IEEE; 2011 [cited 2014 Jul 18]. p. 405–11. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5999853

155. Hill Z, Humphrey M. CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications. IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom 2010) [Internet]. Indianapolis: IEEE; 2010 [cited 2014 Jul 18]. p. 504–11. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708493

156. Loutas N, Kamateri E, Tarabanis K. A Semantic Interoperability Framework for Cloud Platform as a Service. IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011) [Internet]. Athens: IEEE; 2011 [cited 2014 Jul 18]. p. 280–7. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6133154

157. Mindruta C, Fortis T-F. A Semantic Registry for Cloud Services. 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA 2013) [Internet]. Barcelona: IEEE; 2013 [cited 2014 Jul 18]. p. 1247–52. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6550566

158. Thabet M, Boufaida M. An Agent-Based Architecture and a Two-Phase Protocol for the Data Portability in Clouds. 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA 2013) [Internet]. Barcelona: IEEE; 2013 [cited 2014 Jul 18]. p. 785–90. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6550491

159. Emeakaroha VC, Healy P, Fatema K, Morrison JP. Analysis of Data Interchange Formats for Interoperable and Efficient Data Communication in Clouds. IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC 2013) [Internet]. Dresden: IEEE; 2013 [cited 2014 Jul 18]. p. 393–8. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6809438

160. Miranda J, Guillen J, Murillo JM, Canal C. Assisting Cloud Service Migration Using Software Adaptation Techniques. IEEE Sixth International Conference on Cloud Computing (CLOUD 2013) [Internet]. Santa Clara, California: IEEE; 2013 [cited 2014 Jul 18]. p. 573–80. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6676742

161. Boob S, Gonzalez-Velez H, Popescu AM. Automated Instantiation of Heterogeneous Fast Flow CPU/GPU Parallel Pattern Applications in Clouds. 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2014) [Internet]. Torino: IEEE; 2014 [cited 2014 Jul 18]. p. 162–9. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6787267

162. De Morais T, Liberalquino D, Rosa N. Cloud-Aware Middleware. IEEE 27th International Conference on Advanced Information Networking and Applications (AINA 2013) [Internet]. Barcelona: IEEE; 2013 [cited 2014 Jul 18]. p. 780–7. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6531833

163. Nguyen BM, Tran V, Hluchy L. Development and deployment of cloud services via abstraction layer. International Conference on Computing, Management and Telecommunications (ComManTel 2013) [Internet]. Ho Chi Minh City, Vietnam: IEEE; 2013 [cited 2014 Jul 18]. p. 246–51. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6482399

164. Maheshwari K, Birman K, Wozniak J, Zandt DV. Evaluating Cloud Computing Techniques for Smart Power Grid Design Using Parallel Scripting. 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013) [Internet]. Delft: IEEE; 2013 [cited 2014 Jul 18]. p. 319–26. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6546108

165. Oprescu A-M, Antonescu A-F, Demchenko Y, Laat C de. ICOMF: Towards a Multi-cloud Ecosystem for Dynamic Resource Composition and Scaling. IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013) [Internet]. Bristol: IEEE; 2013 [cited 2014 Jul 18]. p. 49–55. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6753777

166. Demchenko Y, Ngo C, de Laat C, Garcia-Espin JA, Figuerola S, Rodriguez J, et al. Intercloud Architecture Framework for Heterogeneous Cloud Based Infrastructure Services Provisioning On-Demand. 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA 2013) [Internet]. Barcelona: IEEE; 2013 [cited 2014 Jul 18]. p. 777–84. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6550490

167. Li W, Tordsson J, Elmroth E. Modeling for Dynamic Cloud Scheduling Via Migration of Virtual Machines. IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011) [Internet]. Athens: IEEE; 2011 [cited 2014 Jul 18]. p. 163–71. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6133140

168. Abdul-Rahman O, Aida K. Multi-layered Architecture for the Management of Virtualized Application Environments within Inter-cloud Platforms. IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013) [Internet]. Bristol: IEEE; 2013 [cited 2014 Jul 18]. p. 238–43. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6735427

169. Michon E, Gossa J, Genaud S, Frincu M, Burel A. Porting Grid Applications to the Cloud with Schlouder. IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013) [Internet]. Bristol: IEEE; 2013 [cited 2014

Jul 18]. p. 505–12. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6753839

170. Miceli C, Miceli M, Jha S, Kaiser H, Merzky A. Programming Abstractions for Data Intensive Computing on Clouds and Grids. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid [Internet]. Shanghai: IEEE; 2009 [cited 2014 Jul 18]. p. 478–83. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5071908

171. Kotecha S, Bhise M, Chaudhary S. Query translation for cloud databases. International Conference on Engineering (NUiCONE 2011) [Internet]. Ahmedabad, Gujarat: IEEE; 2011 [cited 2014 Jul 18]. p. 1–4. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6153249

172. Da Silva EAN, da Silva VG, Lucredio D, de Mattos Fortes RP. Towards a model-driven approach for promoting cloud PaaS portability. Latin American Computing Conference (CLEI 2013) [Internet]. Naiguata: IEEE; 2013 [cited 2014 Jul 18]. p. 1–9. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6670667

173. Strijkers R, Cushing R, Makkes MX, Meulenhoff P, Belloum A, Laat C de, et al. Towards an Operating System for Intercloud. IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013) [Internet]. Bristol: IEEE; 2013 [cited 2014 Jul 18]. p. 63–8. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6735397

174. Aversa R, Tasquier L, Venticinque S. Cloud agency: A guide through the clouds. Mondo Digitale. 2014;13(49).

175. Steinbauer M, Khalil I, Kotsis G. Challenges in the Management of Federated Heterogeneous Scientific Clouds. Journal of Integrated Design and Process Science. 2014;(1):53–67.

176. Ciuffoletti A. A simple and generic interface for a cloud monitoring service. CLOSER 2014 - Proceedings of the 4th International Conference on Cloud Computing and Services Science. Barcelona, Spain: SciTePress; 2014. p. 143–50.

177. Amato A, Venticinque S. A Distributed Agent-based Decision Support for Cloud Brokering. Scalable Computing: Practice and Experience [Internet]. 2014 Apr 30 [cited 2014 Jul 18];15(1). Available from: http://www.scpe.org/index.php/scpe/article/view/966

178. Lordan F, Tejedor E, Ejarque J, Rafanell R, Álvarez J, Marozzo F, et al. ServiceSs: An Interoperable Programming Framework for the Cloud. Journal of Grid Computing. 2014 Mar;12(1):67–91.

179. Di Martino B, Cretella G. Semantic Technology for Supporting Software Portability and Interoperability in the Cloud – Contributions from the mOSAIC Project. Advances in Parallel Computing. 2013;66–78.

180. Sotiriadis S, Bessis N, Antonopoulos N, Hill R. Meta-scheduling algorithms for managing inter-cloud interoperability. International Journal of High Performance Computing and Networking. 2013;7(3):156.

181. Zeginis D, D'Andria F, Bocconi S, Gorronogoitia Cruz J, Collell Martin O, Gouvas P, et al. A user-centric multi-PaaS application management solution for hybrid multi-Cloud scenarios. Scalable Computing: Practice and Experience [Internet]. 2013 Apr 16 [cited 2014 Jul 18];14(1). Available from: http://www.scpe.org/index.php/scpe/article/view/824

182. Andročec D, Vrček N. Platform as a Service API Ontology. Proceedings of the 12th European Conference on eGovernment. Barcelona, Spain: Academic Publishing International Limited; 2012. p. 47–54.

183. Amin MB, Khan WA, Awan AA, Lee S. Intercloud message exchange middleware. Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC'12 [Internet]. Kuala Lumpur; Malaysia: ACM Press; 2012 [cited 2014 Jul 18]. p. 1. Available from: http://dl.acm.org/citation.cfm?doid=2184751.2184845

184. Kostoska M, Gusev M, Ristov S. A New Cloud Services Portability Platform. Procedia Engineering. 2014;69:1268–75.

185. Amato A, Tasquier L, Copie A. Vendor Agents for IAAS Cloud Interoperability. Intelligent Distributed Computing VI. Springer Berlin Heidelberg; 2013. p. 271–80.

186. Wright P, Sun Y, Harmer T, Keenan A, Stewart A, Perrott R. A constraints-based resource discovery model for multi-provider cloud environments. Journal of Cloud Computing: Advances, Systems and Applications. 2012;1(1):6.

187. Zhang Z, Wu C, Cheung DWL. A survey on cloud interoperability: taxonomies, standards, and practice. ACM SIGMETRICS Performance Evaluation Review. 2013 Apr 29;40(4):13.

188. Woo SS, Mirkovic J. Optimal application allocation on multiple public clouds. Computer Networks. 2014 Aug;68:138–48.

189. Jamshidi P, Ahmad A, Pahl C. Cloud Migration Research: A Systematic Review. IEEE Transactions on Cloud Computing. 2013;1(2):142–57.

190. Natis Y, Pezzini M, Driver M, Smith DM, Iijima K, Altman R. Magic Quadrant for Enterprise Application Platform as a Service [Internet]. Magic Quadrant for Enterprise Application Platform as a Service. 2014 [cited 2014 Jul 4]. Available from: http://www.gartner.com/technology/reprints.do?id=1-1P502BX&ct=140108&st=sb

191. Google. Will it play in Java [Internet]. Google; 2012 [cited 2013 Aug 8]. Available from: http://code.google.com/p/googleappengine/wiki/WillItPlayInJava

192. Schaeffer C. Salesforce.com Review—An Independent Assessment [Internet]. 2011 [cited 2013 Aug 8]. Available from: http://www.crmsearch.com/salesforce-review.php

193. Salesforce. Database.com Workbook [Internet]. Salesforce; 2013 [cited 2013 Jul 1]. Available from: http://www.salesforce.com/us/developer/docs/workbook_database/workbook_database.pdf

194. Salesforce. Defining Custom Object Fields [Internet]. [cited 2013 Jul 1]. Available from: http://help.salesforce.com/help/doc/en/dev_objectfields.htm

195. Salesforce. Visualforce Developer's Guide [Internet]. 2013 [cited 2013 Jul 1]. Available from: http://www.salesforce.com/us/developer/docs/pages/index_Left.htm

196. Pattern-oriented software architecture: a system of patterns. Chichester ; New York: Wiley; 1996. 457 p.

197. Google. Google App Engine - Storing Data [Internet]. Google; 2013 [cited 2013 Jul 1]. Available from: https://developers.google.com/appengine/docs/java/datastore/

198. Google. Google Cloud SQL [Internet]. Google; 2013 [cited 2013 Jul 1]. Available from: https://developers.google.com/cloud-sql/

199. Franks L. Data Storage Offerings on the Windows Azure Platform [Internet]. 2010 [cited 2013 Jul 1]. Available from: http://social.technet.microsoft.com/wiki/contents/articles/1674.data-storage-offerings-on-the-windows-azure-platform.aspx

200. Google. Overview of Google Cloud Endpoints [Internet]. Google; 2013 [cited 2013 Jul 12]. Available from: https://developers.google.com/appengine/docs/java/endpoints/

201. Musial-Bright A. Deploy Java RESTful Application on the Google App Engine [Internet]. O'Reilly Answers. 2011 [cited 2013 Jul 12]. Available from: http://answers.oreilly.com/topic/2727-deploy-java-restful-application-on-the-google-app-engine/

202. Schwartzenberger J. Build a RESTful API architecture within an ASP.NET MVC 3 application [Internet]. I Want My MVC. 2011 [cited 2013 Jul 12]. Available from: http://www.iwantmymvc.com/rest-service-mvc3

203. Salesforce. Creating REST APIs using Apex REST [Internet]. Salesforce; [cited 2013 Jul 13]. Available from: http://wiki.developerforce.com/page/Creating_REST_APIs_using_Apex_REST

204. Rudominer M. HOW TO: Build a SOAP Server and a SOAP Client on Google App Engine [Internet]. 2011 [cited 2013 Jul 13]. Available from: https://developers.google.com/appengine/articles/soap

205. Microsoft. Code Quick Start: Create and deploy a WCF service in Windows Azure [Internet]. Microsoft; 2011 [cited 2013 Jul 13]. Available from: http://msdn.microsoft.com/en-us/library/windowsazure/gg651130.aspx

206. Albert A. Apex Web Services and Callouts [Internet]. Salesforce; 2013 [cited 2013 Jul 13]. Available from: http://wiki.developerforce.com/page/Apex_Web_Services_and_Callouts

207. Google. Interface DatastoreService [Internet]. 2013 [cited 2013 Jul 1]. Available from: https://developers.google.com/appengine/docs/java/javadoc/com/google/appengine/api/datastore/DatastoreService

208. Auer S, Feigenbaum L, Miranker D, Fogarolli A, Sequeda J. Use Cases and Requirements for Mapping Relational Databases to RDF [Internet]. 2010 Jun [cited 2013 Jul 1]. Available from: http://www.w3.org/TR/rdb2rdf-ucr/

209. Astrova I, Korda N, Kalja A. Rule-Based Transformation of SQL Relational Databases to OWL Ontologies. Proceedings of the 2nd International Conference on Metadata & Semantics Research [Internet]. 2007 [cited 2013 Jul 1]. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.8189

210. Apache. Apache Jena [Internet]. 2013 [cited 2013 Jul 1]. Available from: http://jena.apache.org/index.html

211. Google. Entities, Properties, and Keys [Internet]. 2013 [cited 2013 Aug 26]. Available from: https://developers.google.com/appengine/docs/java/datastore/entities

212. Microsoft. Data Types (Windows Azure SQL Database) [Internet]. 2013 [cited 2013 Jul 1]. Available from: http://msdn.microsoft.com/en-us/library/windowsazure/ee336233.aspx

213. Salesforce. SOAP API Developer's Guide Version 28.0 [Internet]. 2013 [cited 2013 Jul 1]. Available from: http://www.salesforce.com/us/developer/docs/api/

214. Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider P, et al. OWL Web Ontology Language Reference [Internet]. W3C; 2004 Feb [cited 2013 Jul 1]. Available from: http://www.w3.org/TR/owl-ref/

215. W3C. XML Schema Part 2: Datatypes Second Edition [Internet]. W3C; 2004 Oct [cited 2013 Jul 1]. Available from: http://www.w3.org/TR/xmlschema-2/

216. Husted T. Vosao - Google App Engine CMS [Internet]. 2013 [cited 2014 Apr 3]. Available from: https://code.google.com/p/vosao/

217. XSL Transformations (XSLT) Version 1.0 [Internet]. W3C; 1999 Nov [cited 2014 Sep 19]. Available from: http://www.w3.org/TR/xslt

218. Apache. Apache CXF Dynamic Clients [Internet]. The Apache Software Foundation; 2013 [cited 2013 Oct 1]. Available from: http://cxf.apache.org/docs/dynamic-clients.html

219. Domingue J, Zaremba M, Norton B, Kerrigan M, Mocan A, Carenini A, et al. Reference Ontology for Semantic Service Oriented Architectures [Internet]. OASIS; 2008 Nov [cited 2013 Oct 22] p. 1–35. Available from: http://docs.oasis-open.org/semantic-ex/ro-soa/v1.0/pr01/see-rosoa-v1.0-pr01.pdf

220. Salesforce. Metadata API Developer's Guide [Internet]. Salesforce; 2013 [cited 2013 Oct 14]. Available from: http://www.salesforce.com/us/developer/docs/api_meta/

221. Google. Google App Engine Java API [Internet]. Google; [cited 2013 Oct 14]. Available from: https://developers.google.com/appengine/docs/java/javadoc/

222. Microsoft. Windows Azure Storage Services REST API Reference [Internet]. Microsoft; 2011 [cited 2013 Oct 14]. Available from: http://msdn.microsoft.com/en-us/library/windowsazure/dd179355.aspx

223. Neuhaus F, Vizedom A, Baclawski K, Bennett M, Dean M, Denny M, et al. Towards ontology evaluation across the life cycle. Applied Ontology. 2013;(3):179–94.

224. SOA4All Consortium. SOWER [Internet]. SOA4All Consortium; 2010 [cited 2013 Sep 30]. Available from: http://technologies.kmi.open.ac.uk/soa4all-studio/provisioning-platform/sower/

225. Ilghami O. Documentation for JSHOP2 [Internet]. 2006 [cited 2013 Jul 5]. Available from: http://sourceforge.net/projects/shop/files/JSHOP2/

226. Ilghami O, Nau DS. A General Approach to Synthesize Problem-Specific Planners [Internet]. College Park, Maryland: University of Maryland; 2003 Oct [cited 2014 Sep

23]. Report No.: CS-TR-4597. Available from: http://www.cs.umd.edu/~nau/papers/ilghami2003general.pdf

Curriculum Vitae

Darko Andročec was born on October 11th 1981 in Čakovec, where he received high school diploma from *Gimnazija Čakovec*. He graduated at Faculty of Organization and Informatics in 2004. Since 2009 he works as teaching assistant at Faculty of Organization and Informatics, University of Zagreb, where he teaches computer labs for the following courses: Information system development, E-business, Business processes in organizations, and Information systems of small and medium-sized enterprises. He is member of Department for information system development. Before joining faculty, he was a computer security incident handler at CARNet (Croatian Academic and Research Network) and a Java developer of banking information systems at private company Abit Ltd. He was involved in many software development projects, including development of complete core banking information system for Croatian small banks, content management system for CARNet CERT (computer emergency response team), and many other complex web applications. His practical skills include Java EE, web programming (HTML/Javascript/CSS/PHP/JSF), developing applications for platform as a service offers (mainly Google App Engine, Salesforce, and Microsoft Azure), and Semantic Web (RDF, OWL and SAWSDL).

His research interests are in interoperability, cloud computing, e-services, web technology, Semantic Web, AI planning, Internet of things, and programming. He published 9 scientific papers and 19 popular articles. Recently, he was participating on a Croatian national scientific project "Information Infrastructure and Interoperability" (016-0161199-1715). He is a reviewer for two scientific conferences - ECEG (European Conference on eGovernment) and ICCSM (International Conference on Cloud Security and Management), and one scientific journal - IET Software. He lives in Prelog, he is married and has three children.

Published scientific papers

1. Andročec, Darko.
Data Portability Among Providers of Platform as a Service. // Research papers Faculty of Materials Science and Technology Slovak University of Technology in Trnava. 21 (2013); 17-22

2. Pihir, Igor; Tomičić-Pupek, Katarina; Andročec, Darko.
Governmental Incentives for the Application of the Developed e-Services // Proceedings of the 13th European Conference on eGovernment, Volume 2 / Ferrari, Elena ; Castelanovo, Walter (ur.).
Varese, Italy : University of Insubria, Como, Italy ; Academic Conferences and Publishing International Limited , UK, 2013. 398-405

3. Andročec, Darko; Dobrović, Željko.
Creating Hybrid Software Engineering Methods by Means of Metamodels // Proceedings of the ITI 2012 / Luzar-Stiffler, Vesna ; Jarec, Iva ; Bekic, Zoran (ur.).
Zagreb : University Computing Centre, 2012. 481-486

4. Andročec, Darko; Kermek Dragutin.
Useful Patterns for BPEL Developers // Central European Conference on Information and Intelligent Systems - 23rd International Conference 2012 / Hunjak, T. ; Lovrenčić S. ; Tomičić I.

5. Andročec, Darko; Vrček, Neven.
Platform as a Service API Ontology // Proceedings of the 12th European Conference on eGovernment / Gasco, Mila (ur.).
Barcelona : Academic Publishing International Limited, 2012. 47-54

6. Andročec, Darko; Vrček, Neven; Ševa, Jurica.
Cloud Computing Ontologies: A Systematic Review // MOPAS 2012 - The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services / Dini, Petre (ur.).
Chamonix : IARIA, 2012. 9-14

7. Andročec, Darko.
Research Challenges for Cloud Computing Economics // Proceedings of the 22nd Central European Conference on Information and Intelligent Systems / Hunjak, Tihomir ; Lovrenčić, Sandra ; Tomičić, Igor (ur.).
Varaždin : Faculty of Organization and Informatics, 2011. 175-180

8. Andročec, Darko; Vrček, Neven.
E-service Cost-Benefit Analysis in the Public Sector // Proceedings of the ITI 2011 33rd International Conference on Information Technology Interfaces / Luzar-Stiffler, Vesna ; Jarec, Iva ; Bekić, Zoran (ur.).
Zagreb : University Computing Centre, University of Zagreb, 2011. 359-364

9. Andročec, Darko.
Simulating BPMN Models with Prolog // Proceedings of the 21st Central European Conference on Information and Intelligent Systems / Aurer, Boris ; Bača, Miroslav ; Schatten, Markus (ur.).
Varaždin : Faculty of Organization and Informatics, 2010. 363-369