International Journal of Grid Distribution Computing
Vol.7, No.6 (2014), pp.211-220
http://dx.doi.org/10.14257/ijgdc.2014.7.6.17

Semantic Discovery of Cloud Service Catalog Published Over Resource Description Framework

Magesh Vasudevan, Haleema P. K. and N. Ch. S. N. Iyengar

School of Computing Science and Engineering, Vellore Institute of Technology University, Vellore-632014, Tamil Nadu, India
mageshv88@gmail.com, haleema@vit.ac.in, nchsniyr@vit.ac.in

Abstract

Cloud computing is a model to provide pool of services on demand which is shared among the consumers and metered. Cloud deployment should support efficient mechanism to publish its services in a right method such that the end users identify them. Services has to be discovered according to user's requirement dynamically. The proposed work discusses about introducing semantics in the cloud services description, such that it projects itself apart from other providers and capably handle the commercial demand for their services. The semantics are introduced at the service catalog level and the CSP publish their list of services in the form of RDF, whose semantics are defined in a provider specific ontology. The system was evaluated with keyword & frequency matching discovery and proved to be efficient with semantic discovery over RDF data.

Keywords: *Cloud Computing, Cloud Catalog, Cloud Service Discovery, Semantic web, RDF, Ontology, SPARQL*

1. Introduction

Cloud computing is viewed as utility computing where physical and abstract entities are provided as services [1]. One perception of the service is that it should provide elasticity and 'pay as usage'. [2] defines cloud computing with five essential characteristics, which are on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service. Today, CSPs provide both horizontal and vertical scalability for the services provided by them. The services are fully automated where they are provided on demand to the consumers, where pool of resources shared among the consumers, with atomic level of configuration provision, geographically distributed and charged as per capacity or metered[1]. Cloud is delivered as three types of service models and taxonomically viewed as Infrastructure, Platform and Software as service [2]. The Infrastructure is semantically categorized as physical entity and whereas platform and software is categorized into abstract entity. Difference in the perception of the services provided by the CSPs has lead a way to flexibly add semantics and publish the catalogs to the consumers. The Service catalogs provide a way for the CSPs to publish the list of services they provide to the consumers, who choose suitable services and demand for resources.

In this proposed work we provide a mechanism to efficiently query the service catalogs provided by CSPs that is semantically enhanced by providing an ontology for their service. The aim of the work is to list out the set of services by different providers that matches to the consumer's query in a natural language. The rest of the paper is organized as follows, Section

2 analyses the surviving techniques. Section 3 gives the outline of the proposed methodology. Section 4 evaluates the performance of the system. Section 5 concludes with the future work.

2. Surviving Techniques

The need for Service Catalog is discussed by [3], it carefully evaluates the elements that should be present in an Service Catalog. We identified the prominent entities for a service catalog is

1. List of cloud services.
2. Pricing of services.
3. The atomic level of features and characteristics that can be configured.
4. Service Level Agreement.
5. Should be capable of managing demand by directing or incenting consumers.

This specific requirement of the service catalogs has made a way to enrich the catalog with semantics by the service providers to absorb the consumers towards their services. There must be a platform for consumers to search for services according to their requirements. The current techniques focus on generic ontology for publishing the catalog. CSPs operate with different semantics of the cloud services provided by them, so there must be flexibility in defining ontology for the service catalog. Let's get into detail of the current approaches addressing the problem of defining the service catalog and the centralized access of the catalogs, and their drawbacks.

Cloud service discovery mechanism enhanced by ontology was explained in [4]. System is of cloud service reasoning agent which reasons over the services like similarity, equivalence and numerical using a proposed cloud ontology. The data about the services for reasoning using the proposed agent, is acquired through filtering agent which filters web pages using heuristics method like finding evident phrases and frequency of those phrases and nearness of the keywords of those phrases. The semantics did not play a role for defining the service, the plain text is used to identify the services and reasoning agent performs the operations semantically using the proposed cloud ontology. This did not address the issue of publishing the services by semantics directly.

A multi agent system which cooperates through a flexible ontology based matching was proposed in [5]. Yet, the semantics are utilized to match the results and rank but not utilized to publish the service catalog.

A method to discover the cloud services by predefined concept for the ontology, where each CSP publish their list of services to an inter cloud registry by adding semantics to the services by the concepts or specifications of the registry was explained in [6]. Further the user query is matched onto the ontology in database provided by the CSP to find the suitable services for the consumer. This approach does not provide the flexibility to the CSP to design the semantics on their perception; expressiveness of the services provided are restricted based on the rules provided by the inter cloud registry.

Further it discusses the scenario for semantics in service catalog, but not to discover appropriate services for the consumers based on the semantically enabled service catalogs. In Section 3 we discuss in detail how we overcome the problems in the current techniques to discover the appropriate services for the end users.

3. SCSD Architecture overview

3.1. SCSD Architecture

Semantic cloud service discovery aims at CSD to publish their data semantically in the form of Resource Description Framework (RDF) [16]. The semantics of the RDF should be enhanced by Resource Description Framework Schema (RDFS) [17] or by a Web Ontology Language (OWL) [18]. OWL has more expressive power than the RDFS, due to the addition of Description Logics. The RDF for list of services provided by the CSP is maintained at the SCSD server. When a consumer queries over the SCSD for appropriate services matching for their requirement, SCSD extracts the Classes and attributes from the user request and queries over existing CSP endpoints in a federated fashion to retrieve the matching services; Figure1 depicts the entire process of service discovery mechanism. The subsystems of the SCSD are Concept Extraction, Target Ontology Mapper, generating SPARQL [19] query and ranking of the results. The natural language and semantic web are closely related to extract the information and evaluate [7].

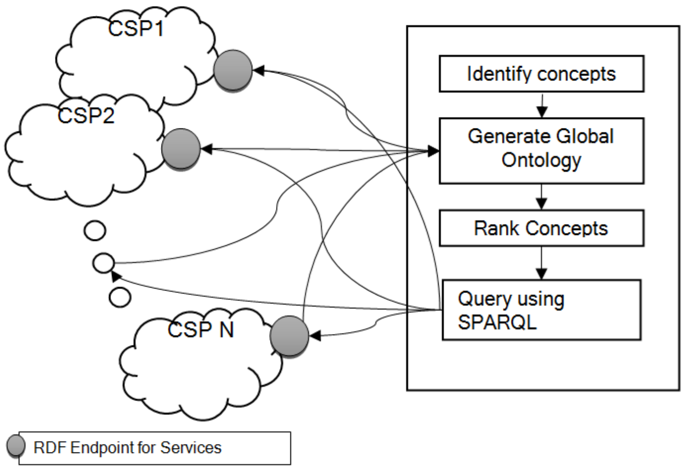


Figure 1. SCSD Architecture

3.2. Concept Extraction

When a user present their query in the form of natural language text, there must be method to extract concepts and their attributes for mapping into the semantics of the CSP. Here we use ReVerb library that extracts the relationship between a concept and the attribute value. For example let’s look into a simplest possible query assumed “Ubuntu having dram memory 2GB”. The reverb splits it into Subject, Predicate and Object as like “Ubuntu”, “having dram memory”, “2GB”. ReVerb does extract the terms through lexical analysis.

3.3. Ontology Merge

The Concepts from different CSPs are merged into single ontology for the purpose of enabling fast query generation and ranking, when concepts are extracted the it is annotated with the CSP id for the purpose of identifying the ontology to generate SPARQL. The Architecture to merge ontology and generating matching concepts is shown in Figure 2. The merging of source and target ontologies are beyond the scope of this paper;

[8, 9, 10, 11] are few effective methods proposed in literature. Concept subclass and superclass are mapped using WordNet *hyponym* and *hypermym* [12, 13], with attribute mapping. Further we add triple to identify similar concept by *owl:sameAs* predicate. While retrieving the Range of the RDF resource the properties related to the relation in an object property resource are also annotated to the Domain of the original resource to enable identification of related data property.

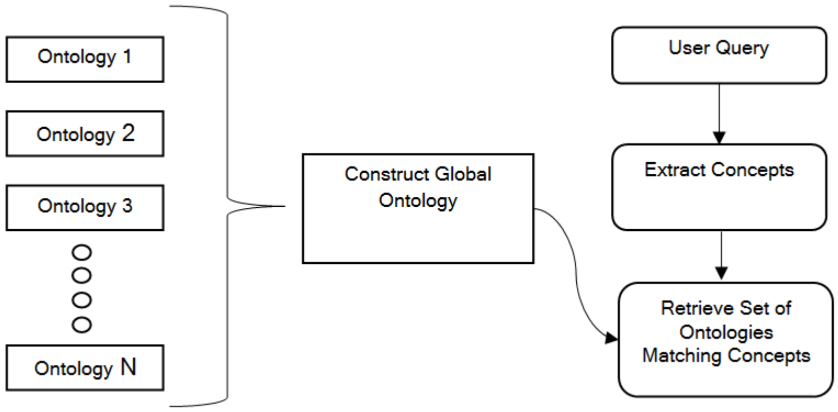

Ontology 1
Ontology 2
Ontology 3
Ontology N
Construct Global Ontology
User Query
Extract Concepts
Retrieve Set of Ontologies Matching Concepts

Figure 2. Construction of Global Ontology

Mapping the semantics to the target ontology enables the SCSD to generate the SPARQL Query for the given user's natural language query. The merged ontology had *64* Entities and *128* properties for illustrative purpose part of ontology is displayed in OntoGraf is show in Figure 3.

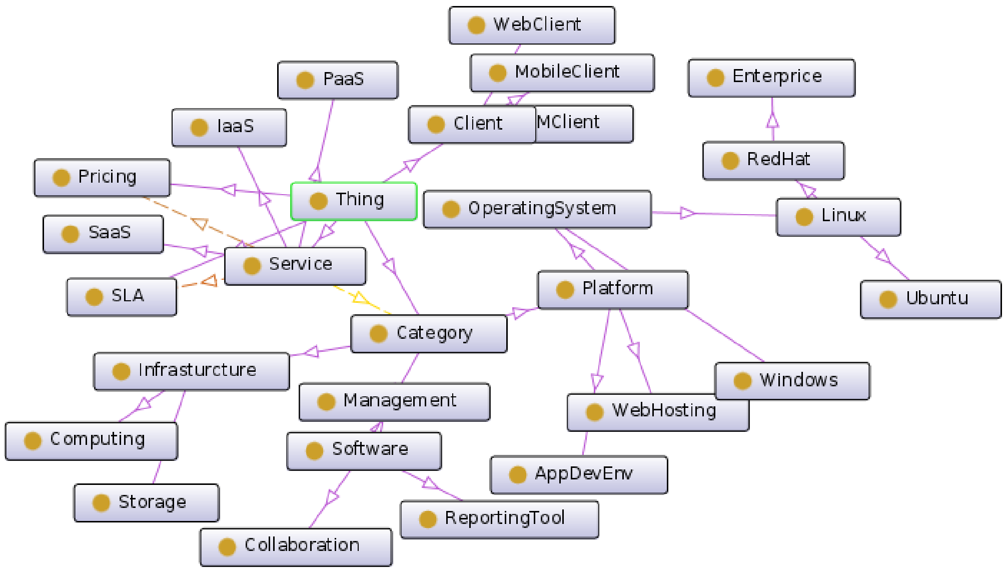


Figure 3. Global Ontology

3.4. Ranking of the results

Three cases are considered while ranking the semantics of the concepts: String similarity, semantic similarity and user constraint on property. Idea behind matching concepts in information retrieval is explained [14]. If concepts are mapped with the attributes specified by the end user, then it has higher rank. The set of concepts from the global ontology is C and set of properties from global ontology is P. The concepts extracted from user query is A, and set of properties from user query is B. String similarity is generated by levenshtein distance and concept similarity is conducted by LCH (Least Common Hypernym) similarity, most of the cases where LCH plays importance is in SaaS, where the words are not restricted to certain domain. Product of these two similarities is made a constant to multiply with the levenshtein and LCH of the constraint specified in case of String, or a constant 1 in case of numerical relatedness. The Algorithm 1 explains the above method to obtain the rank of the concepts. The function compute_levenshtein returns the distance inverted in range from 0 to 1, the max distance is restricted to 15, more than 15 it returns 0.

Algorithm 1. Ranking Concepts

```
Input   Set A (Concept from user query),B(Properties from user query),C(Concept form
        ontology),P(Property from Ontology), N(Constrain for each $b_i$ from B), M( value
        from instance of C with P in global ontology)
Output  Ranked Concepts <R,score>
Step
1       For each $a_i$ in A
2           For each $c_i$ in C
3               lvd_c.add(Compute_levenshtein($a_i$,$c_i$))
4               lch_c.add(Compute_lch($a_i$,$c_i$))
5               maxlvdc := max(lvd_c)
6               maxlchc := max(lch_c)
7               cscore := (maxlvdc*maxlchc)
8               $r_i$ := csore
9           End For
10      End For
11      For each $b_i$,$n_i$ in B,N
12          For each $p_i$,$m_i$ in P,M
13              lvd_p.add(Compute_levenshtein($a_i$,$c_i$))
14              lch_p.add(Compute_lch($a_i$,$c_i$))
15              maxlvdp := max(lvd_p)
16              maxlchp := max(lch_p)
17              If $n_i$ isNumerical then pscore=1
18              Else
19                  lvd_cs.add(compute_lenvenshtein($n_i$,$m_i$))
20                  lch.cs.add(compute_lch($n_i$,$m_i$))
21                  maxlvdcs := max(lvd_cs)
22                  maxlchcs := max(lch_cs)
                    pscore := maxlvdcs*maxlchcs
23              End if
24              $r_i$ := $r_i$ * pscore
25          End for
26      End for
27      Return <R,score>
```

The general mathematical model for constructing rank of Concept $C_i \in$ C with r concepts and property $p_i \in P$ with s properties. The System checks for Match from the concept where match M a scalar representation is defined as,

$$M = |C_i| \sum_{j=1}^{N} |P_j| \quad where\ i = 1 \ldots, r\ and\ J = 1 \ldots, s$$

The above summation is performed if it satisfies c_i in C,

$$C_{i \Rightarrow} P_1 \cap P_2 \ldots . \cap P_s$$

3.5. Generating SPARQL query

SPARQL is a query language to query over RDF data. It has federated query option by specifying the endpoints directly over the query. Using the mapped concepts onto the ontology set of identified semantics are constructed to query over the multiple endpoints i.e the RDF endpoints provided by the CSPs to match the instances. The query is expanded as using the lexical knowledge as presented by [15]. The Ranked concepts from previous step is used to identify the potential concepts to be queried, so the annotated ontologies of the concept generated at the phase of ontology merge (*section 3.3*) is utilized to generated the designated SPARQL Query.

4. Evaluation

4.1. Experimental Setup

The system is simulated with the help of Fuseki RDF server on OpenStack node with Jena interface to query and reason. The Fuseki server was loaded with *32* ontologies representing one CSP with named graph for each. The ontologies were designed carefully differentiated in representation but leading to similar semantics to generate a global ontology as represented in *section 3.3*. As there is no standard data set available for semantic cloud service discovery we carefully generated 165 queries with all possible combinations of logical operations and service category. The query consisted of words with more generalization like "*collaboration*" and more specified like "*instant messenger*". The number of queries for SaaS were more due to the categories of the software range is high. The numerical functions for more, less, average, minimum and maximum were generated. More complex queries were constructed like "*fetch me providers with linux virtual machine with pay per hour cost less than .2 dollars else java web hosting platform with minimum pay per resource*".

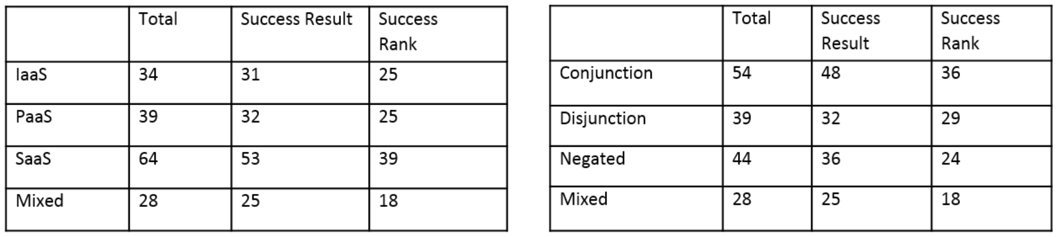
4.2. Evaluating the System

The resulted 165 queries were categorized two ways one is based on type of service retrieval IaaS, PaaS and SaaS, other category is based on logical operation on concept and property with Conjunction , Disjunction and Negation; rest of the queries were tagged under mixed section where it contains all type of constraints. Table 1 represents the numbers of the queries categorized. The reason for categorizing the concepts based on the logical operations is to understand the efficiency of system in handling the concepts with logical operations for similarity measures chosen to rank the results.

The precision and recall measures are used to evaluate the systems performance. The Success result is number queries returned service listings and the success rank are the number of queries that were ranked correctly. The analysis showed that the precision of the system is *.76* and recall *0.85*. Figure 4 and Figure 5 shows the performance of the system.

Table 1. Categorized count of queries

	Conjunction	Disjunction	Negated	Mixed	
IaaS	11	9	9	6	34
PaaS	14	8	11	6	39
SaaS	21	15	17	11	64
Mixed	8	7	7	5	28
	54	39	44	28	165



	Total	Success Result	Success Rank
IaaS	34	31	25
PaaS	39	32	25
SaaS	64	53	39
Mixed	28	25	18

(a) Service Type

	Total	Success Result	Success Rank
Conjunction	54	48	36
Disjunction	39	32	29
Negated	44	36	24
Mixed	28	25	18

(b) Logical Operation

Figure 4. Success rate for result and rank

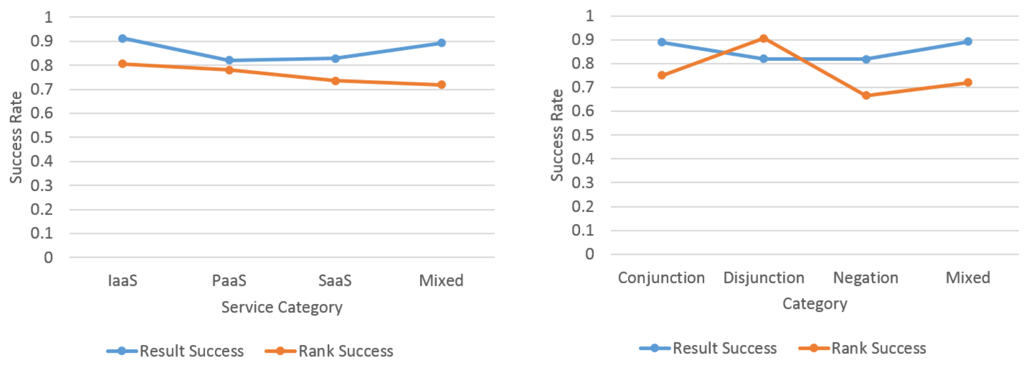


Figure 5. Success rate compared with categories

Further the performance was compared with successfully ranked using semantics and without semantics. The key words from user query were mapped with higher frequency of key word which results in higher rank; this ranking methodology was compared with the results of SCSD is show in Figure 6.

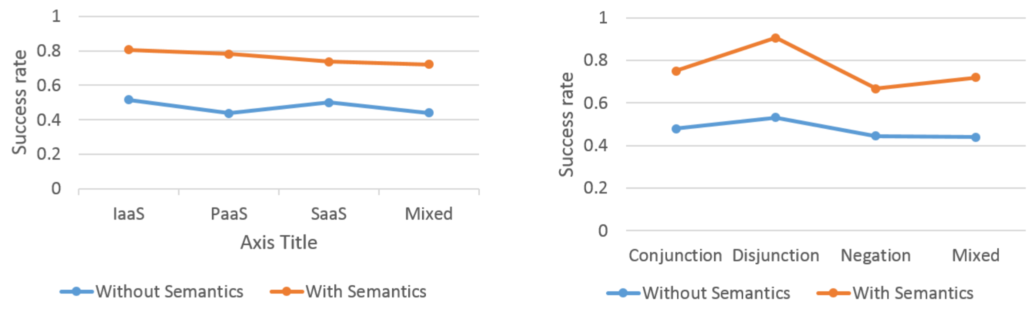


Figure 6. Success rate of SCSD

The response time of the query is compared with the key word & frequency match based discovery is shown in Figure 7. The response time was calculated for the logical operation difference and the number of constraints present in the query.

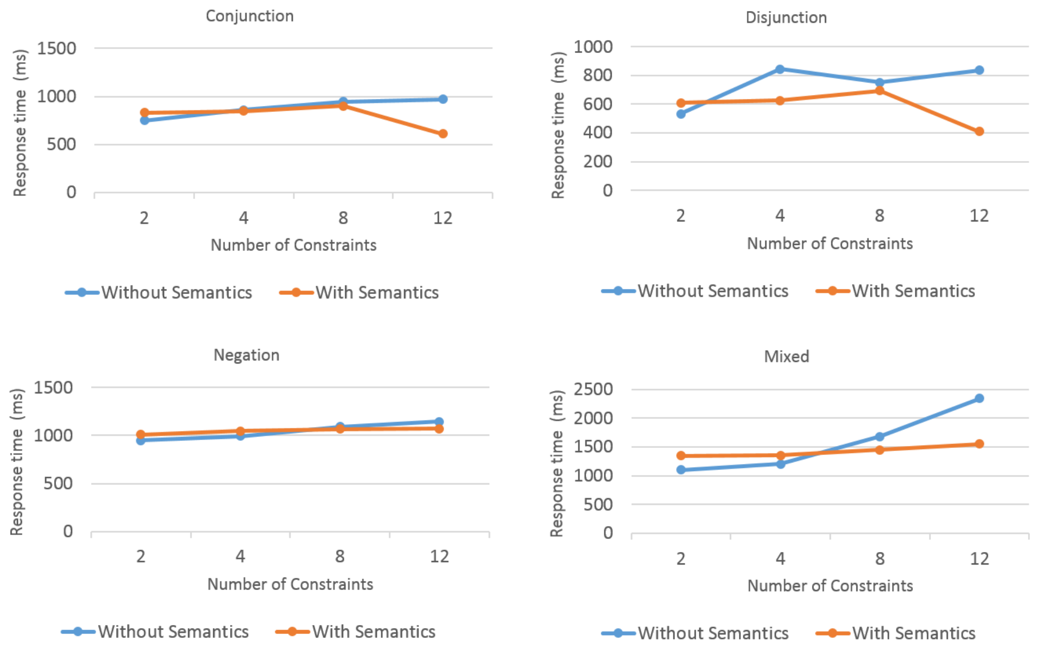


Figure 7. Response time of SCSD

5. Conclusion and future work

The need for cloud service discovery has motivated in designing the SCSD. Though there are many ontology based service retrieval it doesn’t concentrate of publishing the cloud catalog in the form RDF. The SCSD has successfully achieved its aim of publish the cloud catalog with semantics and retrieval of the catalogs in a global ontology method. The global ontology constructed from the CSP structured semantically considering the representation of individual CSP.

Though the response time in few cases were better than keyword based search, most of the cases had low response time; this can be accepted by trading of between the precision and response time. The SCSD aimed at semantically retrieving services that resulted in good precision.

SCSD can be extended with automatic generation of RDF instances from raw catalogs provided by the CSP. The response time of the system can be increased by creating dictionary of Cloud Entities and similar meaning representation based on user feedback. This could highly reduce the computing of similarity measures in retrieving concepts.

References

[1] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond and M. Morrow, "Blueprint for the intercloud-protocols and formats for cloud computing interoperability", Fourth International Conference on Internet and Web Applications and Services, ICIW'09, IEEE, **(2009)**, May, pp. 328-336.

[2] P. Mell and T. Grance, “The NIST Definition of Cloud Computing”, csrc.**nist**.gov/publications/**nist**pubs/800-145/SP800-145.pdf, **(2011)**.

[3] Cisco, “The Need for Service Catalog Design in Cloud Services Development,” Cisco Public Inf., **(2011)**, pp. 1–12.

[4] T. Han and K. M. Sim, "An ontology-enhanced cloud service discovery system", Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1, **(2010)**.

[5] J. Kang and K. M. Sim, "Towards agents and ontology for cloud service discovery", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), IEEE, **(2011)**.

[6] V. Nagireddi and S. Mishra, "A naive approach for cloud service discovery mechanism using ontology", 2013 National Conference on Parallel Computing Technologies (PARCOMPTECH), IEEE, **(2013)**.

[7] Y. Wilks and C. Brewster, "Natural language processing as a foundation of the semantic web", Foundations and Trends in Web Science 1.38211; vol. 4, **(2009)**, pp. 199-327.

[8] K. Makris, *et al.*, "Ontology mapping and SPARQL rewriting for querying federated RDF data sources", On the Move to Meaningful Internet Systems, OTM 2010, Springer Berlin Heidelberg, **(2010)**, pp. 1108-1117.

[9] S. Raunich and E. Rahm, "ATOM: Automatic target-driven ontology merging", 2011 IEEE 27th International Conference on Data Engineering (ICDE), IEEE, **(2011)**.

[10] S. Raunich and E. Rahm, "Towards a benchmark for ontology merging", On the Move to Meaningful Internet Systems: OTM 2012 Workshops. Springer Berlin Heidelberg, **(2012)**.

[11] Shvaiko, Pavel, and J. Euzenat, "Ontology matching: state of the art and future challenges", IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 1, **(2013)**, pp. 158-176.

[12] G. A. Miller, "WordNet: a lexical database for English", Communications of the ACM, vol. 38, no. 11, **(1995)**, pp. 39-41.

[13] R. Richardson, A. Smeaton and J. Murphy, "Using WordNet as a knowledge base for measuring semantic similarity between words", Technical Report Working Paper CA-1294, School of Computer Applications, Dublin City University, **(1994)**.

[14] C. D. Manning, P. Raghavan and H. Schütze, "Introduction to information retrieval", vol. 1, Cambridge university press,Cambridge, **(2008)**.

[15] M. -H. Hsu, M. -F. Tsai and H. -H. Chen, "Query expansion with conceptnet and wordnet: An intrinsic comparison", Information Retrieval Technology, Springer Berlin Heidelberg, **(2006)**, pp. 1-13.

[16] W3C, “RDF 1.1 Concepts and Abstract Syntax”, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[17] W3C, “RDF Schema 1.1”, http://www.w3.org/TR/rdf-schema/.

[18] W3C, “OWL2 Web Ontology Language Structural specification and Functional-Stlye Sysntax”, http://www.w3.org/TR/owl2-syntax/.

[19] W3C, “SPARQL 1.1 Query Language”, http://www.w3.org/TR/sparql11-query/.

Authors

Magesh Vasudevan is currently doing Masters in Computer Science and Engineering at VIT University, Vellore-632014, Tamil Nadu, India. He performed the role of Systems Engineer in Infosys under banking domain. His areas of Interest are Natural Language Processing, Machine Learning, Logic and Ontology Modelling in the Field of Artificial Intelligence.

Haleema (M.C.A., M.Phil,, M.Tech) is an Assistant Professor (Senior) in the School of Social Sciences and Languages and pursuing her Ph.D. research work in the School of Computing Science and Engineering. Her area of research is "Software Agent based computing".

Dr. N. Ch. S. N. Iyengar (b 1961) currently Senior Professor at the School of Computing Science and Engineering , VIT University, Vellore-632014, Tamil Nadu, India .He had 30 yrs of teaching experience. His research interests include Agent-Based Distributed secure Computing, Intelligent Computing, Network Security, Cloud Computing and Fluid Mechanics. He has authored several textbooks and had nearly 172 research publications in reputed peer reviewed International Journals. He delivered many keynote /invited lectures and served as PCM//TCM/reviewer for many International Conferences. He is Editor in Chief for International Journal of Software Engineering and Application (IJSEA) of AIRCC, Guest Editor for SI on Cloud Computing and Services of *Int'l J. of Communications, Network and System Sciences* and Editorial Board member for International Journals like **IJAST of SERSC, IJConvC of Inderscience** and many more.