Research Article

Semantic-Based Requirements Content Management for Cloud Software

Jianqiang Hu,^{1,2} Songhao Jia,³ and Keshou Wu¹

¹School of Computer and Information Engineering, Xiamen University of Technology, Xiamen 361024, China

²Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China

³School of Computer and Information Technology, Nanyang Normal University, Nanyang 473061, China

Correspondence should be addressed to Jianqiang Hu; jqhucn@gmail.com and Songhao Jia; jiasonghao2000@163.com

Received 25 October 2014; Revised 1 March 2015; Accepted 2 March 2015

Academic Editor: Chih-Cheng Hung

Copyright © 2015 Jianqiang Hu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Cloud Software is a software complex system whose topology and behavior can evolve dynamically in Cloud-computing environments. Given the unpredictable, dynamic, elasticity, and on-demand nature of the Cloud, it would be unrealistic to assume that traditional software engineering can “cleanly” satisfy the behavioral requirements of Cloud Software. In particular, the majority of traditional requirements managements take document-centric approaches, which have low degree of automation, coarse-grained management, and limited support for requirements modeling activities. Facing the challenges, based on metamodeling frame called RGPS (Role-Goal-Process-Service) international standard, this paper firstly presents a hierarchical framework of semantic-based requirements content management for Cloud Software. And then, it focuses on some of the important management techniques in this framework, such as the native storage scheme, an ordered index with keywords, requirements instances classification based linear conditional random fields (CRFs), and breadth-first search algorithm for associated instances. Finally, a prototype tool called RGPS-RM for semantic-based requirements content management is implemented to provide supporting services for open requirements process of Cloud Software. The proposed framework applied to the Cloud Software development is demonstrated to show the validity and applicability. RGPS-RM also displays effect of fine-grained retrieval and breadth-first search algorithm for associated instance in visualization.

1. Introduction

Cloud computing is a collection of web-accessible resources, provisioned under service-level agreements established via negotiation, that should be dynamically composed and virtualized based on consumers’ needs on an on-demand basis. Cloud Software is a software complex system whose topology and behavior can evolve dynamically in Cloud-computing environments. Cloud Software tends to evolve at a rapid pace to meet the continuous growing requirements. Due to this, Cloud Software can support (i) coordination of independent and self-interested parties, for example, Cloud consumers and Cloud service providers, (ii) efficient reconfiguration of existent and permanent Cloud service compositions, given constantly changing Cloud consumer requirements, (iii) dealing with incomplete knowledge about the existence of Cloud participants and the services they provide, due to

the distributed nature of Cloud-computing environments, and (iv) dynamic and automated composition of distributed and parallel Cloud services [1, 2]. The adaptability of Cloud Software to Cloud-computing environments and requirement variations becomes critical. Given the unpredictable, dynamic, elastic, and on-demand nature of the cloud, it would be unrealistic to assume that traditional software engineering can “cleanly” satisfy the behavioral requirements of Cloud Software [3]. Hence, a unified requirements metamodeling frame called RGPS (Role-Goal-Process-Service, International Standard ISO/IEC19763) [4] is proposed by merging functional and nonfunctional (context and trustworthy) requirements. This trend is an underlying driving force for “X” as-a-service (XAAS).

A phenomenon worth noting is that vast stacks of requirements of Cloud Software are deposited in the open interconnected environment. The majority of traditional

requirements management tools (such as QSS's DOORS, TBI's Caliber-RM, and IBM Rational RequisitePro) store requirements specifications and related information in multiuser database. This problem can fall into low degree of automation, coarse-grained management, and limited support for requirements modeling activities. Software requirements are insufficient in semantic information and have no unified metamodel. Moreover, traditional requirements management was concerned with change control, version management, and requirements tracking management, but it cannot provide perfect support for requirements elicitation, requirements analysis and requirements V&V, and other activities. Requirements are low reuse and unsuitability for complex software systems including Cloud Software.

Compared to traditional requirements management, requirements management of Cloud Software shows new characteristics: requirements management meets Cloud participants to the greatest extent; requirements of Cloud Software are not only personalized and uncontrolled, but also managed to throughout ensure the lifecycle of requirements, from induction period-growth period-maturity period and finally to the decline period; requirements changes are more complex and difficult to be controlled; requirements reuse is needed to be more refined to achieve the finer granularity.

Accordingly, we argue that automatic construction of Cloud Software is very important for Cloud services that depend upon requirement capabilities. The motivation of our work is to provide a semantic-based requirements content management for Cloud Software that is based on RGPS frame and provides effective support for open requirements process. Meanwhile, we make efforts to build prototype tool called RGPS-RM and explore its function and performance.

2. Background

2.1. Metamodeling Frame RGPS. The ultimate goal of Cloud Software is to realize mass customization at a low cost in a short time. There are many challenges in requirements engineering of Cloud Software as scarcity of standard description language (like WSDL and WADL) for Cloud service [5], without unified domain knowledge for mass requirements custom, and no unified description method for personalized preference. Furthermore, due to uncertain requirements of Cloud Software, current requirements modeling method is difficult to deal with personalized, mass, and diversified requirements of Cloud Software.

Wuhan University and Tsinghua University joined the ISO/IEC/JTC1/SC32/WG2 workgroup, which made International Standard ISO/IEC19763-Metamodel Framework for Interoperability (MFI) complex software systems including Cloud Software [4]. Metamodel frame RGPS (Role-Goal-Process-Service) is the most important standard. In Figure 1, metamodel frame RGPS is a hierarchical and cooperative frame, which sums up multilevel and multi-granularity requirements as Role layer-Goal layer-Process layer-Service layer and the relationship between these four layers. In detail, the Role layer characterizes the organization, role, and actors and describes the interaction and

cooperation in problem domain; the Goal layer depicts the decomposition of goals and determines the constraint relationship among goals; the Process layer distinguishes atomic processes and composite processes and defines the inputs/outputs/preconditions/effects of processes; the Service layer guides the construction of service aggregation of service resources.

In fact, requirement modeling process, based on metamodeling frame RGPS, begins from the analysis of organizational structure of requirement-solved problem space, through mapping and transformation between Role layer-Goal layer-Process Layer-Service layer, and generates Cloud service-based requirements specifications ultimately. The normative formal definition of metamodeling frame RGPS is OWL language.

2.2. Traditional Requirements Management System. Schwaber and Sterpe provide a definition of requirements management: "The storage of requirements, the tracking of relationships among requirements, and the control of changes to individual requirements and groups of requirements" [6]. Hence, traditional requirements management is the process of documenting, analyzing, tracing, prioritizing, and agreeing on requirements and then controlling change and communicating to relevant stakeholders. For supporting an efficient requirements management process, firms have developed some specific tools, which can assist organizations in defining and documenting requirements by allowing them to store requirements in a central location.

2.2.1. DOORS. DOORS is a sophisticated product that can manage requirements on large products. It treats individual requirements as objects but presents them in a visual format that resembles a structured, hierarchical requirements document. The requirements display also shows attribute values, indicators of links to other requirements, and colored bars that indicate a requirement's change status. Defining requirement links through the link matrix is clumsy, but DOORS also provides several other link definition mechanisms.

2.2.2. Caliber-RM. Caliber-RM takes a database-centric approach to requirements management. Caliber-RM has fairly loose integration with Word, but it has highly flexible import capabilities. Caliber-RM provides a Windows Explorer-like workplace for manipulating the hierarchical requirements tree, with requirement details accessible through a tabbed dialog on the right side of the screen. It can also manage traceability relationships and attributes through a grid display.

2.2.3. RequisitePro. RequisitePro takes a document-centric approach to requirements management, exhibiting the tightest integration with Word. It can mark selected blocks of text to include in the database as discrete requirements. It is easy to access the requirement details, including its revision history, attributes, traceability, hierarchy, and discussions. The mechanisms it uses for synchronizing the requirements in the database with the contents of the requirements specification (SRS) are a bit clumsy [7, 8].

FIGURE 1: Metamodeling frame RGPS (Role-Goal-Process-Service).

2.3. Requirements Management of Cloud Software. In addition to the above features, requirements management of Cloud Software provides supporting services for other requirements activities including requirements elicitation, requirements analysis, and requirements validation. The majority of traditional requirements managements take document-centric approaches because neither software requirements have a unified metamodel nor computer needs to understand contents of requirements specifications. Links between requirements are constructed by manual way or information retrieval technologies. Correspondingly, requirements management of Cloud Software has metamodel frame RGPS and needs semantic content management with RGPS requirements (Cloud Software requirements). Keyword-based matching is not enough to reuse and needs to improve the granularity and accuracy, in which information retrieval technologies may be inapplicable to build the relationship among RGPS requirements.

3. Semantic-Based Requirements Management Framework

Facing the new characteristic and challenges of semantic-based requirements management for Cloud Software, we propose a hierarchical framework for semantic-based requirements management of Cloud Software, including semantic requirements storage layer, semantic content management layer, fine-grained semantic indexing layer, requirements organization, and classification layer and common APIs layer, as shown in Figure 2.

3.1. Semantic Requirements Storage Layer. When customizing personalized requirements of Cloud Software, a requirements description language called Service-Oriented Requirements Language (SORL) [9] is to elicit users' requirements precisely. Based on metamodel Frame RGPS, user's requirements in SORL can be customized into goal model and process

FIGURE 2: A hierarchical framework for semantic-based requirements management of Cloud Software.

chain step by step. Finally, these models expressed with OWL can be saved as the corresponding Cloud service-based requirements specifications. In order to get rid of the document-centric management mode of traditional requirements management, this layer focuses on solving the storage mechanisms of Cloud Software requirements and meeting the manageability of massive Cloud Software requirements under the complex network environment. This layer should be necessarily scalable and efficient. Based on the native storage scheme, it takes an ontology hypergraph representation as data mode for its persistent storage, which effectively avoids the costs of data model transformation when accessing Cloud Software requirements data.

3.2. Fine-Grained Semantic Indexing Layer. Besides the basic hypergraph traversal, Cloud Software requirements indices mechanism is designed to accelerate the access to the persistent storage, including tree triple indices. The approach is based on the so-called triple indices which are B-tree index structures built on tree triple sets. In the practical implementation, an ordered index with keywords is an important basis to achieve fine-grained granularity retrieve requirements and to improve the flexibility of requirements reuse.

3.3. Requirements Organization and Classification Layer. In order to achieve an order cache and improve the efficiency of hierarchical retrieval Cloud Software requirements, requirements classification algorithm is a necessary component in typical classification tasks. Some typical algorithms are (1) kernel-based algorithms, such as support vector machines (SVMs) [10], which maximize the margin of confidence of the classifier, and are the method of choice for many such tasks and (2) probabilistic graphical models; they represent correlations between labels by exploiting problem structure. For example, conditional random fields (CRFs) [11]

are a probabilistic framework for labeling and segmenting structured data, such as sequences, trees, and lattices.

3.4. Hierarchical Retrieval Layer. In order to solve single and unitary form in traditional retrieval, this layer constructs concepts, properties, and instances indices to achieve composite retrieval. In addition, association retrieval of Cloud Software requirements is designed to find indirect relevance of requirements instances, to support trace analysis, change analysis, and postimplementation (i.e., free customization of Cloud service-based requirements specifications).

3.5. Common Interfaces Layer. This layer provides well-defined and rich APIs for integration and invocation services of requirements management when it is needed by requirements activities. For example, requirements query service gives four main interfaces, that is, SimpleQuery(), AdvancedQuery(), HierarchicalQuery(), and RelationQuery(). These interfaces shield complex implementation of services and guarantee transparency when requirements activities access semantic-based requirements management system of Cloud Software.

4. Key Technologies of Semantic-Based Requirements Management Framework

4.1. Native Storage Scheme. Data storage modes can be categorized into three groups.

4.1.1. Memory-Based Storage Model. It loads all data into memory with queries fast response, memory overhead significantly.

4.1.2. Relational Object Database-Based Storage Model. It organizes, operates, and manages data by using mature database technologies. However, the performance bottleneck is also inevitable because of the extra cost for data model transformation when accessing Cloud Software requirements data within a non-RDF persistent storage.

4.1.3. Native Storage Schema. It enhances storage and data access performance, as the Cloud Software needs large amount of Cloud Software requirements, as well as the relational model and ontology graph model in the data model conversion prone to impedance mismatch effect. OWL specification defines a mapping OWL ontology O into an RDF graph $G(O)$. In fact, a Cloud Software requirement is defined as a set of triples, a formal definition is given as follows.

Definition 1. Suppose that $G = (V, E)$ is an RDF graph, where $V = \{v_x \mid x \in U \cup B \cup L\}$, and $E = \{(v_s, v_p, v_o) \mid (s, p, o) \in G\}$, U stands for an infinite set of URI references, B for an infinite set of blank nodes, and L for an infinite set of RGPS literals.

We design a concise storage scheme as the class diagram shown in Figure 3, which is implemented on Berkeley DB. *GraphVertex* represents a vertex in V and has three fields:

FIGURE 3: Class diagram of hypergraph persistent storage.

oid, *label*, and *type*. *Oid* has the type of integer with the length of 64 bits and is used to uniquely identify a vertex. *Label* is a variable length string used to record the value of URI reference or literal. *Type* is also a 64-bit integer designed to encode specific semantic information of the vertex in a bitmap manner. *GraphEdge* represents an edge, where *srcVertex*, *typeVertex*, and *tarVertex* correspond to the *oids* of three elements of a triple [12].

The native persistent storage takes the hypergraph representation for Cloud Software requirement as the data model, which effectively avoids the costs of data model transformation, and thereafter achieves an efficient and scalable access to the RDF data.

4.2. Cloud Software Requirements Indices Schema. The data indexing helps to ease the search of and access to data at any given time. Most of the approaches maintain a set of six indices covering all possible access schemes an RDF query may require. These indices are PSO, POS, SPO, SOP, OPS, and OSP (*P* stands for property, *O* for object, and *S* for subject) [13]. These indices materialize all possible orders of precedence of the three RDF elements. This representation allows fast retrieval of all triple access patterns. Otherwise, it is oriented to towards simple statement-based queries and has limitations for efficient processing of more complex queries.

Besides the above set of six indices, there exists another approach to facilitating complex queries. The approach is based on the so-called an ordered index with keywords (see Figure 4) which are structures built on Lexion and placement table (postings). Lexion takes charge of keywords related to postings. Placement table is designed as two-dimensional pointer list: one includes pointers related to software requirements documents; another includes pointers pointing to the same requirements documents where keyword-included triples and records statistics of its frequency exist.

The first dimension pointer can easily obtain network software requirements document related to the keywords. The second dimension pointer can determine the exact triple where the keyword appears and also returns relevant statistical information. Thus, requirements indices schema can ensure fine-grained retrieval and fast keyword search.

4.3. Cloud Software Requirements Instances Classification. Based on metamodel frame RGPS, classification algorithms (such as SVM, linear CRFs) are used to achieve the effective organization of Cloud Software requirements and automatic

classification. The No Free Lunch (NFL) [14] theorem suggests that a more useful strategy is to gain an understanding of the dataset characteristics that enable different classification algorithms perform well. Figure 5 shows the general process of automatic classification of Cloud Software requirements. Obviously, classification algorithm is crucial to automatic classification of Cloud Software requirements. An algorithm's performance is measured on both the percentage of correct classifications and computational complexity.

In document summarization area, many classification algorithms consider the summarization task as two classes and classify each sentence individually without leveraging the relationship among sentences. One is in a discriminative way with well-known algorithm such as support vector machine (SVM). SVM is to find the decision surface that maximizes the margin between the data points of the two classes. Yet, as Cloud Software follows metamodel frame RGPS, its feature space is so large and dependent that makes SVM algorithm cannot fully exploit the potential useful features.

Conditional random fields (CRFs) are a probabilistic framework for labeling and segmenting structured data, such as sequences, trees, and lattices. Taken concept instance, attribute, and attribute value of Cloud Software requirement as three entities, if there is a relationship among the concept instance, attribute, and attribute value, it is marked 1; otherwise it is marked 0. Input sequences of CRFs are attribute value and edge. Suppose the attribute value sequence $X = (x_1, x_2, \dots, x_n)$ and corresponding edge sequence $Y = (y_1, y_2, \dots, y_n)$, the probability of Y conditioned on X defined in CRFs, $P(Y | X)$, is as follows:

$$P(Y | X) = Z_X^{-1} \exp \left(\sum_{i,k} \lambda_k f_k(y_{i-1}, y_i, X) + \sum_{i,l} u_l g_l(y_i, X) \right), \quad (1)$$

where Z_X is the normalization constant that makes the probability of all state sequences sum to one. $f_k(y_{i-1}, y_i, X)$ is an arbitrary feature function over entire observation sequence while $g_l(y_i, X)$ is a feature function of state at position i and the observation sequence; λ_k, u_l are the weights learned for the feature functions f_k and u_l , reflecting the confidence of feature function. The feature functions can measure any aspect of a transition for $y_{i-1} \rightarrow y_i$, and the global characteristics of X .

Given the conditional probability of the state sequence defined by a CRF in (2) and the parameters Λ , the most probable sequence can be obtained as

$$Y^* = \arg \max_Y P_{\Lambda}(Y | X) \quad (2)$$

which can be efficiently calculated by the Viterbi algorithm. Obviously, CRF algorithms consider classification problem with leveraging the relationship among RGPS features.

4.4. Hierarchical Retrieval. Hierarchical retrieval serves for output visualization of semantic requirements management and avoids unitary form in traditional retrieval. In addition, association retrieval of RGPS requirements is designed to find potential relevance of requirements instances.

FIGURE 4: Diagram of an ordered index with keywords.

FIGURE 5: Process of automatic classification of Cloud Software requirements.

4.4.1. Compound Retrieval. One of the Cloud Software requirements is viewed as ontology data in nature. It is necessary to construct all kinds of composite indices, including keywords, attribute, and concept instance, to support arbitrarily complex retrieval. On basis of keyword indexing, we designs three indexing structures: $\langle k, \text{URI}(c), \text{frequency} \rangle$, $\langle k, \text{URI}(p), \text{frequency} \rangle$, $\langle k, \text{URI}(i), \text{URI}(p), \text{frequency} \rangle$, where k is the keyword, $\text{URI}(c)$ is the concept, $\text{URI}(p)$ is the attribute, $\text{URI}(i)$ is an instance URI, and frequency represents occurrences of keyword k . And it in consequence supports compound retrieval with SPARQL and SeRQL languages.

4.4.2. Association Retrieval. In order to find out the potential relationship among Cloud Software requirements, it provides support for requirement traceability of Cloud Software. Requirements traceability is concerned with documenting the life of a requirement. Traditionally, it is difficult to construct the relationship among Cloud Software requirements by manual way because it is not applied to fine-grained changes of Cloud Software requirements. Through ontology-based relation mining among Cloud Software requirements,

including machine automatic analysis, cluster analysis, and statistical analysis, semantic relationship and measurable relationship are found out to support requirements change analysis and requirements version management.

Definition 2 (association relation). There are four association relation modes in Cloud Software requirements as follows:

- (i) $R_{\text{direct}}(c_1, c_2) = r(c_1, c_2)$ represents direct association relation from the concept c_1 to the concept c_2 .
- (ii) $R_{\text{direct}}(i_1, i_2) = r(i_1, i_2)$ represents direct association relation from the instance i_1 to the instance i_2 .
- (iii) $R_{\text{indirect}}(c_1, c_2) = r(c_1, c_k)r(c_k, c_a) \cdots r(c_b, c_2)$ represents indirect association relation from the concept c_1 to the concept c_2 , where $c_1, c_k, c_a, \dots, c_b, c_2$ is sequence of concepts.
- (iv) $R_{\text{indirect}}(i_1, i_2) = r(i_1, i_k)r(i_k, i_a) \cdots r(i_b, i_2)$ represents indirect association relation from the instance i_1 to the instance i_2 , where $i_1, i_k, i_a, \dots, i_b, i_2$ is sequence of instances.

The distinction between the direction mode and indirect mode is very important because it determines whether data mining is needed for requirements data.

Fortunately, metamodeling frame RGPS has been clearly defined relationship among the concepts existed in Cloud Software requirements. We design breadth-first search algorithm for associated instances under the help of metamodeling frame RGPS, without consideration of indirect association.

The advantage of this algorithm is to fully utilize metamodeling frame RGPS to guide node expansion. However, this RGPS-dependent relation is relatively strong and a large number of nodes need to be expanded especially in complexity of nested subclass. Notice that queue instead of stack is used to control the instance expansion (see Algorithm 1).

```

Input: instance  $i_1, i_2$ 
Output: association path between  $i_1, i_2$ 
Algorithm Description:
(1) Flag = 0;
    /* Flag is a flag for existing association path */
(2) Q = null;
    /* Queue Q is an abstract data in which items are entered at one end and remove for the other end (FIFO),
    inque, deque are names used for insertion operation and deletion operation respectively. */
(3)  $c_1 = \text{class}(i_1), c_2 = \text{class}(i_2)$ ;
    /* Get concept class  $c_1, c_2$  from instance  $i_1, i_2$ , i.e.,  $i_1 \in c_1, i_2 \in c_2$ ; */
(4) Get indirect concept path from meta-modeling frame,  $R_{\text{indirect}}(c_1, c_2) = r(c_1, c_k)r(c_k, c_a) \cdots r(c_b, c_2)$ 
    find corresponding association path from  $i_1$  to  $i_2$ 
    where  $i_k \in c_k, i_a \in c_a, \dots, i_2 \in c_2$ 
(5) Q.inque( $i_1$ );
(6) While (!Flag || Q  $\neq \Phi$ )
    {  $i = \text{Q.deque}()$ ;
      expand( $i$ );
      /* Expansion rule:  $\forall i_j \in c_j$ , if concept  $c_j, c_m$  has association relation  $r(c_j, c_m)$ ,
      then  $i_j$  can be expanded as:  $\text{expand}(i_j) = \{i_0 \mid r(i_j, i_0), i_0 \in c_m\}$  */
      if ( $i_2 \in \text{expand}(i)$ )
        {Flag = 1; return indirect association path.}
      else if {
        foreach( $i' \in \text{expand}(i)$ )
          Q.inque( $i'$ ); // add  $i'$  to queue Q
        }
      }
(7) Return null.

```

ALGORITHM 1: Breadth-first search algorithm for association relation.

5. Experiments and Results

5.1. An On-Demand Q/A Case in Traveling Domain. In this section, experiments to test the semantic-based requirements management framework are provided. Through an on-demand Q/A system case in traveling domain, the corresponding instance of RGPS requirement is construed.

Take Xiamen traveling planning as an example, Mr. Li chooses a suitable planning considering factors as weather conditions, traffic information, and distance between Xiamen University and corresponding destination. RGPS guides dynamic requirements modeling by describing personalized requirements from multirole, multiobjective, multiprocess, and multiservices. A requirements description language named SORL is to elicit users' requirements precisely, and requirement of Cloud Software that can be expressed with OWL can be saved as the corresponding Cloud service-based requirements specifications (Cloud service ontology) in the end. A prototype tool called RGPS-RM for semantic-based requirements content management is implemented to provide supporting services for open requirements process of Cloud Software.

Figure 6 demonstrates the process of automatic construction of Cloud Software. Figure 7 gives part of traffic services categorization ontology. Cloud providers/services of traveling domain must be searched according to their geographical location, offered services, deployment model, security mechanisms, pricing, and so forth. When one Cloud provider goes out of business again, a compliant Cloud provider according

to the requirements must be chosen by searching method (e.g., ontology-based generic search method [15] and hidden Markov model-based service discovery method [16]) and it must be ensured that Cloud services are interoperable.

5.2. RGPS-RM. In order to evaluate the effectiveness of the proposed framework, a prototype tool called RGPS-RM is built, whose backend database is Berkeley DB to ensure reliable and high performance. Combined with an on-demand Q/A system case in traveling domain, Figure 8 shows an overall view of RGPS-RM, which is very convenient for observation and navigation. Different colors dots denote Role layer (red)-Goal layer (yellow)-Process Layer (blue)-Service layer (green). RGPS-RM provides four main interfaces, that is, SimpleQuery(), AdvancedQuery(), HierarchicalQuery(), and RelationQuery(), to achieve arbitrary granularity retrieval. Figure 8 shows center all kinds of #TripleItinerary's association relations which are searched by RelationQuery() using breadth-first search algorithm for associated instances.

5.2.1. Performance Evaluation. In order to evaluate the performance and native storage scheme, we compared RGPS-RM with Sesame v2.7.14 against Lehigh University Benchmark (LUBM) [17]. Sesame was configured to store RDF data in MySQL v5.6.22. All the experiments were conducted on a server with two Dual-Core Intel Xeon CPUs (2.8 GHz), 3 GB DDR 333 RAM, and two 300 GB hard disks.

FIGURE 8: RGPS-RM with relation query.

FIGURE 9: Query response time comparison.

TABLE 1: Result of SVM and CRF in RGPS classification.

Algorithm	Precision	Recall	F_1
SVM	88.6%	86.2%	0.874
CRF	95.6%	91.3%	0.934

5.2.2. Classification Result. We use precision, recall, and F_1 -measure [18] to evaluate the results, which are widely used in information retrieval.

We choose 500 RGPS requirements stored in RGPS-RM. Evaluation was performed through SVM and CRFs to classify RGPS requirements. Table 1 shows that CRFs better than SVM in both the average value of each category and as whole.

6. Conclusions

Cloud Software brings forward many new challenges to traditional requirements management, particularly its low degree of automation, coarse-grained management, and limited support for requirements modeling activities.

Based on metamodeling frame called RGPS (Role-Goal-Process-Service) international standard, this paper presents a hierarchical framework of semantic requirements content management for Cloud Software, which has the following key technologies: introducing native storage scheme to avoid data model transformation; taking an ordered index with keywords to support Cloud Software requirements in arbitrary granularity retrieval; employing linear CRFs to achieve the effective organization of Cloud Software requirements and automatic classification; and using breadth-first search instance to find out the potential association relation among Cloud Software requirements. In addition to the above, we also construct prototype system called RGPS-RM, which demonstrates perfect performance and exhibits effect of breadth-first search algorithm for associated instance in visualization.

The future work includes providing better supporting services for requirements elicitation, requirements analysis and requirement V&V, and other activities and completing prototype system RGPS-RM and being applied in some areas including Cloud-based e-business and Cloud-based mobile augmentation.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the reviewers for their detailed reviews and constructive comments, which have helped improve the quality of this paper. Special thanks are due to Juanzi Li and Keqing He for their sound insight and professional assistance. This work was supported by Xiamen Science and Technology Program of University Innovation Project under Grants nos. 3502Z20143031 and 3502Z20133033.

References

- [1] I. Neamtui and T. Dumitras, "Cloud software upgrades: challenges and opportunities," in *Proceedings of the International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA '11)*, pp. 1–10, Williamsburg, Va, USA, September 2011.
- [2] J. O. Gutierrez-Garcia and K. M. Sim, "Agent-based cloud service composition," *Applied Intelligence*, vol. 38, no. 3, pp. 436–464, 2013.
- [3] R. Bahsoon, I. Mistrík, N. Ali, T. S. Mohan, and N. Medvidović, "The future of software engineering in and for the cloud," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2221–2224, 2013.
- [4] ISO/IEC JTC1 SC32 WG2 Development/Maintenance, "ISO/IEC 19763, Information Technology-Metamodel Framework for Interoperability (MFI)," 2014, <http://metadata-stds.org/19763/>.
- [5] T. H. Noor, Q. Z. Sheng, A. Alfazi, A. H. H. Ngu, and J. Law, "CSCE: a crawler engine for cloud services discovery on the

- world wide web,” in *Proceedings of the IEEE 20th International Conference on Web Services (ICWS '13)*, pp. 443–450, July 2013.
- [6] M. G. Violante and E. Vezzetti, “A methodology for supporting requirement management tools (RMt) design in the PLM scenario: an user-based strategy,” *Computers in Industry*, vol. 65, pp. 1065–1075, 2014.
 - [7] K. E. Wieggers, “Automating Requirements Management,” http://www.processimpact.com/articles/rm_tools.pdf.
 - [8] B. J. M. Abma, *Evaluation of Requirements Management Tools with Support for Traceability-Based Change Impact Analysis*, University of Twente in Netherlands, 2009.
 - [9] J. Tian, K. He, C. Wang et al., “An approach to generation of process-oriented requirements specification,” *Journal of Software Engineering and Applications*, vol. 2, no. 1, pp. 13–19, 2009.
 - [10] M. Pontil and A. Verri, “Properties of support vector machines,” *Neural Computation*, vol. 10, no. 4, pp. 955–974, 1998.
 - [11] V. Deufemia, M. Risi, and G. Tortora, “Sketched symbol recognition using Latent-Dynamic Conditional Random Fields and distance-based clustering,” *Pattern Recognition*, vol. 47, no. 3, pp. 1159–1171, 2014.
 - [12] G. Wu, J.-Z. Li, J.-Q. Hu, and K.-H. Wang, “System II: a native RDF repository based on the hypergraph representation for RDF data model,” *Journal of Computer Science and Technology*, vol. 24, no. 4, pp. 652–664, 2009.
 - [13] D. C. Faye, O. Cure, and G. Blin, “A Survey of RDF storage approaches,” *ARIMA Journal*, vol. 15, pp. 11–35, 2012.
 - [14] D. H. Wolpert and W. G. Macready, “No free lunch theorem for search,” Tech. Rep. SFI-TR-05-010, Santa Fe Institute, Santa Fe, NM, USA, 1995.
 - [15] V. S. K. Nagireddi and S. Mishra, “A generic search based cloud service discovery mechanism,” in *Proceedings of the Science and Information Conference*, pp. 915–923, London, UK, October 2013.
 - [16] Q. Wu, M. Zhang, R. Zheng, Y. Lou, and W. Wei, “A QoS-satisfied prediction model for cloud-service composition based on a hidden markov model,” *Mathematical Problems in Engineering*, vol. 2013, Article ID 387083, 7 pages, 2013.
 - [17] SWAT Projects-the Lehigh University Benchmark (LUBM) [EB/OL], <http://swat.cse.lehigh.edu/projects/lubm/>.
 - [18] D. M. W. Powers, “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness & correlation,” *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.