2010 34th Annual IEEE Computer Software and Applications Conference

An Ontology-Based Approach to Reengineering Enterprise Software for Cloud Computing

Hong Zhou and Hongji Yang
Software Technology Research Laboratory
De Montfort University
Leicester, UK
hongzhou, hyang @ dmu.ac.uk

Andrew Hugill
Institute Of Creative Technologies
De Montfort University
Leicester, UK
ahu @dmu.ac.uk

Abstract—**Cloud computing is the future trend for enterprise software solutions, which means a lot of legacy systems will need to be either adapted to fit the requirement of cloud computing or to be purged and redesigned from scratch. However, enterprise software is far too complex for any human being to understand as a whole. This paper proposes a novel approach to reengineering enterprise software for cloud computing by building an ontology for enterprise software and then partitioning the enterprise software ontology to decompose enterprise software into potential service candidates. Ontology development process includes three steps, namely, building ontologies for source code, data, and application framework respectively, integrating captured ontologies and deploying the final produced ontology. Firstly, the ontology development process is supported by the reverse engineering and model transformation techniques. Secondly, the ontology integration is based on ontology engineering research. Thirdly, the deployment of enterprise software ontology is done through the software reengineering activities. Once the ontology is built, there will be a link between ontology and enterprise software. By analysing the concepts and relations in ontology, the enterprise software will be understood and decomposed as different service candidates.**

Cloud Computing; Enterprise Software; Service; Ontology; Ontology Development Process; Ontology Integration; Ontology Partitioning; Model Transformation; Class Diagram; Hibernate ORM Framework

I. Introduction

Cloud computing is one of the future trends of software engineering research, which implies a service-oriented architecture (SOA) aiming to reduce IT overheads by providing more flexible and more economic usage for software end users. In essence, cloud computing provides a set of IT services covering from the software applications to hardware devices, which are transparent to the end users. End users do not need to own any IT resources, but consume resources as services and pay for the resources they use. It could involve many related research areas such as distributed computing, grid computing, utility computing, web services, software as a service (SaaS), platform as a service (PaaS) etc. For software reengineering research, the research question in relation to cloud computing will be how to decompose the legacy system into potential service candidates that will be fit for a cloud computing environment, specifically, how to understand and detect the loosely coupled reusable components of legacy system and then to make these components work as services in the cloud. This paper is going to explore the first part of this question, i.e., understanding legacy software and decomposing it into potential service candidates that could meet the requirements of cloud computing

This paper proposes an ontology-based approach to reengineering enterprise software system for the cloud computing environment. Enterprise software is known as a suite of programs that are intended to solve an enterprise problem and is always complex and large-scale. Therefore, reengineering enterprise software has never been an easy job in terms of comprehension and decomposition. An ontology-based reengineering methodology is proposed in this study to understand and decompose the enterprise software for shifting to cloud computing.

The remainder of the paper is organised as follows: In Section II, background and previous studies will be introduced with related work. In Section III, proposed approach is presented with a framework and other detailed techniques. In Section IV, a case study is demonstrated to evaluate the proposed approach. In Section V, the paper is concluded and the possible future work is suggested.

II. Background and Related Work

As an emerging and promising computing paradigm, cloud computing has attracted increasing interest from software engineering researchers. On one hand, researchers [18] are trying to build a layered classification, in which cloud computing research could be divided into five layers in a top-down manner, namely, cloud application layer (e.g. Software as a Service (SaaS)), cloud software environment layer (e.g. Platform as a Service (PaaS)), cloud software infrastructure layer (e.g., Virtualisation, Infrastructure as a Service (Iaas), Data-Storage as a Service (DaaS) and Communication as a Service (CaaS)), software kernel layer (e.g., Hardware as Service (HaaS)). On the other, it is also important to compare cloud computing with other existing computing paradigms. Mei et al. [12] have done a qualitative comparison among cloud computing, service computing and pervasive computing from different aspects, from which they discovered three notable similarities among those computing paradigms, namely: I/O similarity between cloud computing and service computing; storage similarity between cloud computing and pervasive computing; and calculation similarity among all three paradigms.

With regards to both the classification of cloud computing research and the comparison among different computing paradigms, the position of this research will be mainly on the top layer of the five-layer cloud computing classification, aiming to reengineer legacy enterprise software for delivery as services in the cloud computing environment. Due to the similarities it shares with service computing and grid computing, this study will consider service-oriented software reengineering (SOSR) as related research.

Most traditional approaches in SOSR endeavour to expose the whole legacy system as Web services, which is summarised in [10]. Black-box wrapping technique is used in the research carried out by Canfora et al. [7] to apply a new Web service interface to interactive form-based legacy systems. Grey-box reengineering approach based on clustering is proposed by Zhang and Yang [19] in order to extract services from legacy systems. White-box reengineering approach developed by Sneed [14] is also used, which requires code analysis and manipulation for obtaining the parts of the legacy application to be exposed as Web services.

A knowledge-based system can be a possible approach to help with the situation. Several studies have been carried out to utilise ontology and related techniques to facilitate program comprehension and reverse engineering. Yang et al. [17] suggest that ontologies have a great potential for legacy software understanding and re-engineering. Ontology-based PlaTform-specIfic software Migration Approach (OPTIMA) is proposed in [21], in which ontology is used to represent the features of operating system to assist platform specific software migration. The research in [9] is an extension of OPTIMA approach, which focusing on using Real Time Operating System (RTOS) ontology to aid Virtual RTOS (VRTOS) design. In addition, an ontology-based domain specific program comprehension approach is discussed in [20, 22], with respect to the knowledge intensive features of both software system and program understanding process. Chen et al. [8] propose an ontology-based approach to identifying service in service-oriented software reengineering research.

Based on these studies, the proposed approach in this paper is trying to build a comprehensive ontology for enterprise software and then to partition this ontology in order to detect potential service candidates for cloud computing environment.

III. An Ontology-Based Approach to Reengineering Enterprise Software for Delivery as Services in Cloud Computing

Enterprise software is the software program which performs a set of functions or processes to meet the general requirements for many different organisations and industries. Generally speaking, most enterprise software will have the following features: a.) using object-oriented design patterns, b.) using relational database management systems for data storage, c.) using reusable libraries/application frameworks, d.) hiding implementations in the back of well-defined interfaces, and e.) providing commonly needed functions and services, e.g., ERP, CRM and so on.

The proposed approach is to utilise ontology and its related techniques to explore the concepts and relations in enterprise software and therefore to enhance enterprise software comprehension. Enterprise software ontology is developed by integrating three other ontologies: code ontology, database ontology and Hibernate ORM framework [1] ontology. By analysing and modularising strongly related concepts in enterprise software ontology, it will enable decomposing legacy software into loosely coupled modules that are considered to be potential service candidates in cloud computing environment.

A. Ontology-Based Enterprise Software Comprehension Framework

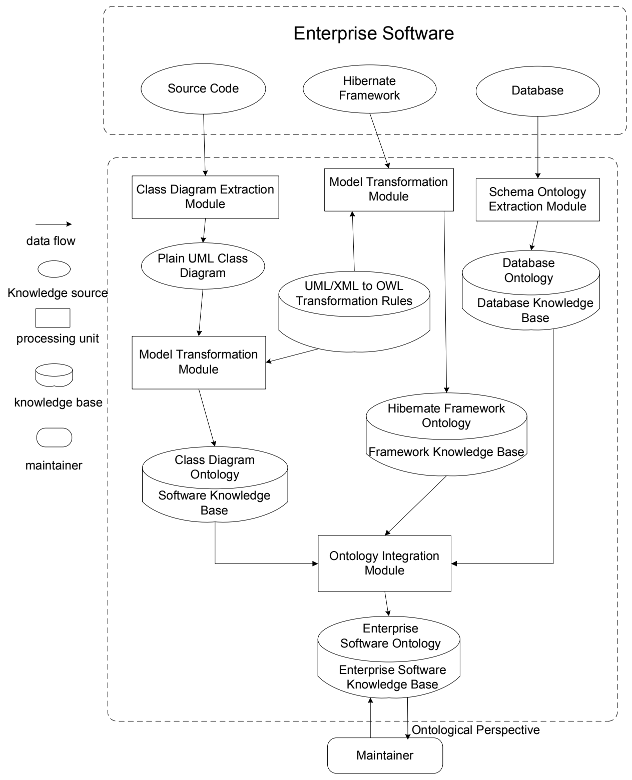

Figure 1. Ontology-Based Enterprise Software Comprehension Framework

Figure 1 illustrates a framework for proposed approach. The inputs of the framework are source code, Hibernate framework files and database. The output of the system is enterprise software ontology, with which software maintainer can have a better understanding of the enterprise software.

B. *Enterprise Software Ontology Development Process*

The enterprise software ontology development process is one of the important parts in the proposed approach. The ontology development process was first introduced by Uschold and Gruninger [16] and will be adopted and slightly modified in this study. The modified process is:

1)**Preparation** This aims to identify the purpose of building the ontology and its intended uses in software reengineering.

2)**Capture** As an essential process in building software ontology, capture seeks generic and reusable representations of software system knowledge that can be reused across a variety of software reengineering activities. Key concepts and relations in software system will be identified during this process.

3)**Coding** As a formal description of concepts and relationships, ontology forms a shared terminology for the objects of interest in software reengineering. Through the coding process, software system ontology will be (semi) automatically generated and be written in a representation language. Coding and capture are simply merged into one step in this study, since they are always closely connected.

4)**Integration** As a unified platform, ontology provides the methodologies to integrate different knowledge sources. Ontologies that represent different representation perspectives of software system can be integrated into one, which is used to assist software reengineering.

5)**Deployment** As a knowledge sharing technology, deployment is always the last step. In this study software system ontology will be deployed in software reengineering activities such as program comprehension.

C. *Source Code Ontology*

At the moment, this study is mainly focusing on object-oriented enterprise software. The source code ontology generation process is supported by both reverse engineering techniques and model transformation techniques, and it can be divided into two parts, i.e., UML class diagram extraction and UML to OWL model transformation. Current reverse engineering techniques can provide an automatic extraction of UML class diagram from a legacy system. ATL model transformation [2] is used in this study to support transforming UML to OWL. A set of rules will be written to guide the transformation process based on ATL transformation language. Source code ontology generation is therefore an automatic process which extracting class diagram first and then transforming class diagram into ontology.

D. *Database Ontology*

When reengineering database dominant systems, database becomes inevitable to be analysed because of the complicated connection between source code and database. In addition, database somehow stores explicit knowledge about the system, which could be used to assist program comprehension.

The process of eliciting knowledge from a database is defined as data understanding in this research. At this stage, only database schema is being studied, because things like the semantic meanings of the tables, columns, relationships among different tables and columns will directly affect software maintainers' decisions when performing a modification, while the things like the actual individual record stored in particular table will not. Ontology is used to describe the semantics of the information sources and to make the contents explicit. Specifically, with respect to the comprehension of data, they can be used for the identification and association of semantically corresponding information concepts. On the one hand, extracted database ontology could be used to associate with the source code ontology, which can enhance the understandability of the code ontology. On the other, extracted database ontology could be used to improve data interoperability for the further requirements, e.g., data integration.

E. *Hibernate ORM Framework Ontology*

An Application (software) framework is a pre-defined program structure or a set of reusable common code hidden behind well-defined APIs, which provide generic functionalities. As one of the general features, application framework is widely used in enterprise software. To narrow down the research scope, this study will mainly focus on a database related software framework Hibernate ORM Framework. Hibernate uses XML mapping documents to store the information about persistent classes such as how the classes map to tables or columns in database. With those XML mapping documents, Hibernate will be able to generate SQL at runtime and free programmers from writing SQL statements in the code. The proposed approach is focusing on those XML mapping documents that associate code to database.

It is believed that obtaining the mapping knowledge from Hibernate framework can help building the link between code knowledge and database knowledge. As a result, it could simplify the ontology integration process. Furthermore, it will assure the software modification being carried out in a more secured way.

F. *Enterprise Ontology Integration Algorithm*

Only one of the above ontologies will not be sufficient to represent the entire software system. In other words, to understand software system will need to obtain knowledge from different perspectives [20]. Comprehensive software ontology should contain several different knowledge perspectives. Integration of different ontology is therefore performed. The integration algorithm for enterprise software ontology is given below:

1) Load three ontologies Oc, Od and Of, in which Oc is the code ontology, Od is the database ontology and Of is the framework ontology.
2) Scan the concepts names (Linguistic similarity can be determined in many different ways, such as by

synonymy, shared substrings, common prefixes, or common suffixes).

3) Define a triple pattern (Cf1, C f2, Rfm) in Of, which represents concepts Cf1 and C f2 and the mapping relation Rfm between them in framework ontology Of. It is believed that for any concept Cf stored in Of, there is always a concept Cc from Oc or Cd from Od that will match with Cf in terms of linguistic similarity.
4) For each triple pattern (Cf1, C f2, Rfm) in Of. Assume that Cf1 represents the mapping concept in code and C f2 represents the mapping concept in database. Retrieve the initial list of suggestions. Find the best choice Cc that matches with Cf1 from the suggestions in Oc and Cd that matches with Cf2 from the suggestions in Od.
5) Introduce a new relation Rfm between Cc and Cd, and then create a new triple pattern (Cc, Cd, Rfm) which build the connection between code ontology Oc and database ontology Od.
6) Delete the old triple pattern (Cf1, C f2, Rfm) in Of.
7) Repeat 4 and 5 until all the triple patterns (Cf1, C f2, Rfm) are deleted.
8) Generate enterprise software ontology Oe.
9) Stop.

Based on this algorithm, the code ontology, database ontology and Hibernate framework ontology will be integrated, and the enterprise software ontology is populated.

G. *Toolset Support*

To apply the proposed approach to the selected case studies, a prototype toolset is designed. This toolset could be divided into following modules:

1)***UML Class Diagram Extraction Unit*** For the external open source toolset unit that produces UML class diagram, there are currently bunch of choices, covering most object-oriented language such as Java, C ++ and C #. Topcased [3] is one of Eclipse plug-ins that are widely used in software development project. It is more stable for large scale source code and will be used to perform UML class diagram generation.

2)***Database Ontology Extraction Unit*** For the external open source toolset unit that produces database schema ontology, there are also a few options available. DataGenie [4] and DataMaster [5] are Protege plug-in which allow Protege to import schema structure and data from relational database. In another word, these plug-in are capable of converting relational database into ontology. DataMaster is used in this study for database schema ontology generation.

3)***ATL Model Transformation Unit*** The generation of class diagram ontology and framework ontology will be implemented by ATL [2] model transformation. ATL is a model transformation language and toolkit, which provides ways to produce a set of target models from a set of source models in terms of Model-Driven Engineering (MDE). In this case, both UML model and XML model will be transformed into OWL model.

4)***Toolset Implementation*** To support the proposed approach, a prototype toolkit is implemented as an Eclipse plug-in development based on Topcased modeling tools, ATL model transformation and Protege-OWL API [15]. Figure 2 illustrates the architecture for this toolset. This toolset includes two parts: ontology generation module and ontology integration module. To generate code ontology, firstly a UML class diagram will be extracted from the source code with the reverse engineering function of Topcased modeling tools; then the UML class diagram will be transformed to ontology by ATL model transformation. Hibernate framework ontology is also populated by ATL model transformation. Database ontology is built by Protege plug-in DataMaster. The output of ontology generation module will be three OWL files which store the knowledge elicited from enterprise software. The ontology integration module is responsible for integrating all these three OWL files together. Ontology integration algorithm is designed to integrate the different ontologies. The manipulation of ontology in this module is implemented by Protege-OWL API. The integrated ontology will be in OWL format.

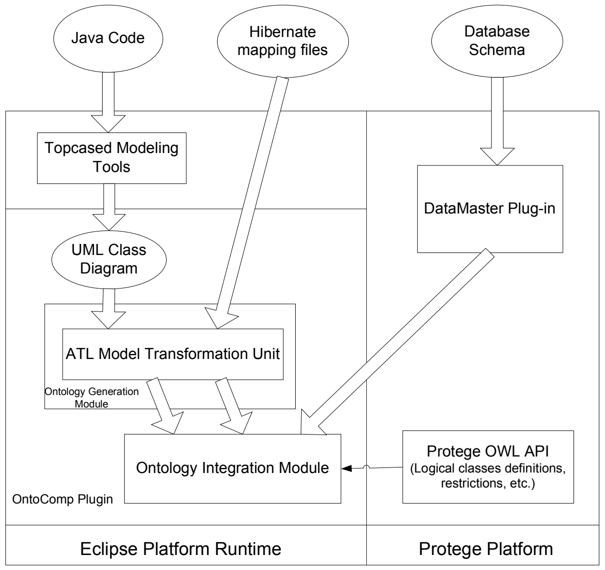

Figure 2. Toolset Architecture

H. *Partitioning Ontology to Identify Potential Service Candidates for Cloud Computing*

The goal of developing enterprise software ontology is to achieve a more comprehensive representation form of software system and then to deploy this new form in software reengineering projects that identify potential service candidates of legacy system for cloud computing. The identification of cloud computing service candidates requires a decomposition of software system with respects to the following two principles, loosely coupled components and reusable functionalities. Therefore, the components and their interrelationships in the software system need to be analysed, and the strongly related components need to be staying together while loosely coupled ones can be apart. In this research, the decomposition of enterprise software is accomplished by modularising enterprise software ontology, which is also known as ontology partitioning.

Currently, there are several studies on ontology partitioning [11, 13], this paper will adopt the structure-based partitioning algorithm proposed by Schlicht and Stuckenschmidt [13]. Their partitioning algorithm is based on the structural dependencies between concepts in ontology, which are represented through a weighted dependency graph. Then the strength of the dependencies between the concepts is calculated and the proportional strength network is obtained to detect sets of strongly related concepts. As a result, the concepts which are stronger related will be modularised and the original ontology will be divided into loosely coupled partitions.

After applying structure-based partitioning algorithm, the enterprise software ontology can be decomposed into a few modules. For each module, all the concepts are strongly related, and also organised around specific functions and domain concepts. Furthermore, all the modules are loose coupled as well. Hence, they can be considered to be potential service candidates in relation to cloud computing environment.

IV. Case Study – Plazma Business Solution System

A. *Plazma Business Solution System*

Plazma [6] business solution system is open-source ERP+CRM application for middle business. It contains seven enterprise functionalities including accounts management, contacts management, sales management, tasks management, campaigns management, products management and analytical reports. The database server supports Oracle, PostgreSQL, MySQL, Firebird and HSQL. And it is compatible with Windows, Linux and MacOS. It is no doubt to be a very powerful and robust business solution system. But the end user will need more IT resources to support it, which indicates that it could become a burden for the end user to some extent. However, all the above features also show that this enterprise software could be a potential cloud computing application in terms of scalability and flexibility. The proposed approach is applied to reengineer Plazma business software for shifting to cloud computing.

B. *Ontology Generation*

Topcased is used to perform class diagram extraction. It supports reversing Java from both a project and a JAR. A UML file containing all the classes and associations has been generated by this reverse engineering tool. With ATL UML to OWL model transformation, this UML file has been transformed to an OWL file containing concepts and relations. 575 classes have been extracted from the source code and transformed to ontology concepts, while 684 associations have been extracted and restored as relations in ontology.

DataMaster is used to generate database ontology from MySQL database. Only the database schema is being considered. 644 concepts as been extracted from the Plazma database and saved into an OWL file.

Hibernate ORM Framework ontology is created based on the Hibernate mapping files. ATL XML to OWL model transformation has been used to generate Hibernate framework ontology. 452 concepts have been extracted from model transformation, in which 226 concepts are mapping with code ontology concepts and 226 concepts are mapping with database ontology concepts.

C. *ATL Model Transformation*

ATL model transformation provides a mean to define the way that how source model and target model should be matched and transformed. The model transformation processes used in this study could be defined as following:

1) **UML to OWL Model Transformation** UML class diagram and ontology could be considered as two different models Mc and Mo. In order to achieve the transformation, two metamodels MMc and MMo, a metametamodel MMM and a transformation model Mc2Mo will be provided, in which Mc conforms to MMc, Mo conforms to MMo, Mc2Mo conforms to ATL, MMc conforms to MMM, MMo conforms to MMM and ATL conforms to MMM. Mc is the UML file, MMc, MMo, MMM have been defined in ATL Transformation knowledge base, i.e., UML.ecore, OWL.ecore and Ecore respectively. The actual transformation rules will be defined in Mc2Mo.atl file, which provides a guide for matching class diagram to ontology.

2) **XML to OWL Model Transformation** XML to OWL model transformation is similar to UML to OWL model transformation. XML model Mx conforms to metamodel MMx, ontology model Mo conforms to metamodel MMo. Both MMx and MMo conforms to metametamodel MMM, which is Ecore defined in ATL knowledge base. The transformation rules are defined in Mx2Mo.atl file, which are dealing with the details for each transformation.

D. *Plazma Enterprise Ontology Integration*

Based on the algorithm discussed in Section 4.6, the final produced comprehensive enterprise software ontology is generated by integrating code ontology, database ontology and Hibernate ORM Framework ontology. 870 concepts and 1215 relations are observed in this final ontology for Plazma business solution system. This final ontology will provide more knowledge perspectives for program comprehension by integrating knowledge that covers code, data and application framework.

E. Identification of Potential Service Candidates for Cloud Computing

After applying structure-based partitioning algorithm to Plazma enterprise software ontology, 6 partitions have been obtained initially, namely, finance partition, sale and purchase partition, product and inventory partition, project management partition, human resources and payroll partiotion, contacts management partition. Finance partition has 165 concepts including accounting, bank, cash, payment, tax, currency, etc. Sale and purchase partition has 145 concepts including sale order, sale plan, sale invoice, purchase order, purchase invoice, etc. Product and inventory partition has 133 concepts including inventory move, inventory writeoff, inventory outcome, product info, product price, product stock, manufacture, goods, etc. Project management partition has 98 concepts including task, task status, task priority, task type, etc. Human resources and payroll partition has 182 concepts including organisation, personality, employee, store, warehouse, person job, education type, employee rank, etc. contacts management partition has 147 concepts including email, phone, address, partner, partner group, industry, etc. Consequently, 6 potential service candidates for cloud computing environment are obtained, which are financial and accounting service, sale and purchase management service, product and inventory management service, project management service, human resources and payroll service and contacts management service. Hence, the Plazma business solution can be reengineered for cloud computing by providing these 6 services, which will still meet the requirements from the end user, but will require less IT resources to support. That is the goal of cloud computing – flexible, scalable and economic.

V. Conclusion and Future Work

This paper proposes an ontology-based approach to reengineering enterprise software for shifting to cloud computing, i.e., to identify potential service candidates from the legacy system. Following a five-step ontology development process the enterprise software ontology is created by generating and integrating code ontology, database ontology and Hibernate framework ontology. The deployment is performed by analysing and modularising strongly related concepts in enterprise software ontology, it will enable understanding and decomposing legacy software into loosely coupled modules that are considered to be potential service candidates in cloud computing environment. The case study is conducted on an open-source ERP+CRM system, which shows that the proposed approach is an efficient reengineering methodology for large-scale software system in terms of automating. 6 potential service candidates are successfully identified from the legacy software, which will then be reused in cloud computing. However, this approach is not fully automatic, human intervention is still needed. Hence, the future research will be focusing on providing a detailed instruction and prediction for human intervention.

References

[1] "Hibernate ORM Framework": https://www.hibernate.org/.

[2] "ATL Model Transformation": http://www.eclipse.org/m2m/atl/.

[3] "Topcased": http://www.topcased.org/.

[4] "DataGenie": http://protege.cim3.net/cgi-bin/wiki.pl?DataGenie.

[5] "DataMaster": http://protegewiki.stanford.edu/index.php/DataMaster.

[6] "Plazma Business Solutions": http://plazma.sourceforge.net.

[7] G. Canfora, A. R. Fasolino, G. Frattolillo and P. Tramontana, "Migrating Interactive Legacy Systems to Web Services", *10th European Conference on Software Maintenance and Reengineering (CSMR'06)*, Bari, Italy, Mar. 2006, pp. 24-36.

[8] F. Chen, Z. Zhang, J. Li, J. Kang and H. Yang, "Service Identification via Ontology Mapping", *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, Seattle, USA, Jul. 2009, pp. 486-491.

[9] F. Chen, H. Zhou, J. Li, R. Liu, H. Yang, H. Li, H. Guo and Y. Wang, "An Ontology-based Approach to Portable Embedded System Development", *21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009)*, Boston, USA, Jul. 2009, pp. 569-574.

[10] D. Kuebler and W. Eibach, "Adapting Legacy Applications as Web Services", http://www.ibm.com/developerworks/library/ws-legacy/ Jan 2002.

[11] Z. Lei, S. Xia, Z. Xia and Z. Yong, "Study on Ontology Partition Based on Ant Colony Algorithm", *9th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD' 08)*, Thailand, Aug. 2008, pp. 73-78.

[12] L. Mei, W. K. Chan and T. H. Tse, "A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues", *IEEE Asia-Pacific Services Computing Conference (APSCC'08)*, Yilan, Taiwan, Dec. 2008, pp. 464-469.

[13] A. Schlicht and H. Stuckenschmidt, "A Flexible Partitioning Tool for Large Ontologies", *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '08)*, Sydney, Australia, Dec. 2008, pp. 482-488.

[14] H. M. Sneed, "Encapsulation of Legacy Software: A Technique for Reusing Legacy Software Components", *Annals of Software Engineering,* vol. 9, pp. 293-313, 2000.

[15] Stanford, "Protégé Programming Development Kit (PDK)": http://protege.stanford.edu/doc/dev.html.

[16] M. Uschold and M. Gruninger, "Ontologies: Principles, Methods and Applications", *Knowledge Engineering Review,* vol. 11, pp. 93--136, 1996.

[17] H. Yang, Z. Cui and P. O'Brien, "Extracting Ontologies from Legacy Systems for Understanding and Re-Engineering", *23rd Annual International Computer Software and Applications Conference (COMPSAC'99)*, Phoenix, AZ, Oct. 1999, pp. 21-26.

[18] L. Youseff, M. Butrico and D. D. Silva, "Toward a Unified Ontology of Cloud Computing", *Grid Computing Environments Workshop (GCE'08)*, Nov. 2008, pp. 1-10.

[19] Z. Zhang and H. Yang, "Incubating Services in Legacy Systems for Architectural Migration", *11th Asia-Pacific Software Engineering Conference (APSEC'04)*, Busan, Korea, Nov. 2004, pp. 196 - 203.

[20] H. Zhou, F. Chen and H. Yang, "Developing Application Specific Ontology for Program Comprehension by Combining Domain Ontology with Code Ontology", *8th International Conference on Quality Software (QSIC'08)*, Oxford, UK, Aug. 2008, pp. 225-234.

[21] H. Zhou, J. Kang, F. Chen and H. Yang, "OPTIMA: an Ontology-based PlaTform-specIfic software Migration Approach", *7th International Conference on Quality Software (QSIC'07)*, Portland, Oregon, USA, Oct. 2007, pp. 143-152.

[22] H. Zhou, "COSS: Comprehension by Ontologising Software System", *24th IEEE International Conference on Software Maintenance (ICSM'08)*, Beijing, China, Sep. 2008, pp. 432-435.