A Software Redocumentation Process Using Ontology Based Approach in Software Maintenance

Sugumaran Nallusamy, Suhaimi Ibrahim, and Mohd Naz'ri Mahrin

Abstract—**This paper describes a framework to redocumenting legacy source codes and other related software components to support the software maintenance. The framework generates the ontology to enhance the program understanding in semantic levels by defining knowledge which is relevant to the domain of interest. The redocumentation process is done semi automatically which was initiated by the Software Work Products (SWP) and maps out to the metadata and generates it as ontology. Finally, the generated ontology will be mapped to standard software documentation such as the MIL-STD-498 based on the request of the maintainer by using the reasoning tools.**

Index Terms—**Software documentation, ontology, reverse engineering, software maintenance, program understanding.**

I. Introduction

Software redocumentation is one of the approaches used as an aid for program understanding to support the maintenance and evolution. According to Elliot, J.C. et. al. "Redocumentation is a creation or revision of a semantically equivalent representation within the same relative abstraction level"[1]. In other words, redocumenting a code is transformation of one existing (and other documents and stakeholder knowledge) into a new or updated documented code. It therefore becomes an aid for the recovery and recording in software comprehension. Since program comprehension is the most expensive part of the software maintenance, redocumentation is the key to software maintainability. The main goals of the software redocumentation process are threefold. It is firstly used to create alternative views of the system so as to enhance the understanding. For example the generation of a hierarchical data flow[2] or control flow diagram from the source code. Secondly, it is also used to improve the current documentation. Ideally, such documentation should have been produced during the development of the system and updated as the system changes. This, unfortunately, is not usually the case. Thirdly, it also generates documentation for a newly modified program[3]. This is aimed at facilitating future maintenance work on the system which can act as the preventive maintenance. Therefore, understanding of the software system by the system maintainer is important for the software evolution. However, the people involved in software maintenance normally do not participate in the development of the system. The maintainer does not have the domain knowledge of the software system and probably this has an effect on the effectiveness and efficiency on the software maintenance. At presents most solutions is only emphasize the redocumentation on the source code or design level rather than the domain level.

Manuscript received August 31, 2011; revised September 14, 2011. This work was supported by the Ministry of Higher Education (MOHE), Malaysia and University Technology Malaysia (UTM) under the Fundamental Research Grant Scheme (FRGS) vote number 4F061.

Sugumaran Nallusamy is with the University of Tunku Abdul Rahman (UTAR) , Kuala Lumpur, Malaysia. (e-mail: sugumaran@ utar.edu.my).

Suhaimi Ibrahim is with the Centre of Advanced Software Engineering (CASE), University Technology Malaysia, Kuala Lumpur, Malaysia. (e-mail: suhaimiibrahim@utm.my).

Mohd. Nazr'ri Mahrin T. C. Author is with the Centre of Advanced Software Engineering (CASE), University Technology Malaysia, Kuala Lumpur, Malaysia. (e-mail: mnazrim@utm.my).

In this scope of the research, there is a need to explore the redocumentation process to produce a standard software documentation to describe the context of the software system or in terms specific concepts of the domain. The models and techniques used should be able to produce the documentation from the SWP. It needs to capture the artifacts from the latest version of the software system by establishing a reverse engineering environment. This will involve the development of the tool to extract the components from the source code and translate them to the component which can present a high level of abstraction in the system of a standard documentation. The new work should be able to view the documentation as a result of the reasoning tool which supports the search and select mechanisms to extract the required knowledge. Software engineers or maintainers should consider these results as an aid in understanding the program for the maintenance task. The remainder of this paper is organized as follows. Section 2 describes the basic model of redocumentation. The ontology based approach used to redocument the source code and other sources while in section 3. The partial implementation scenario is in section 4. It followed by section 5 investigates the related work. The final part of this paper provides the conclusion and suggestions for future work in this research.

II. Software Redocumentation Process

Basically, the redocumentation process uses the reverse engineering architecture to produce the documentation. The basic model for reverse engineering is provided by Robert Arnold of the Software Productivity Symposium[1]. In addition, the model has been restructured to show a detailed knowledge based structure. Mainly, the redocumentation process is viewed as a knowledge rescue process[4] as shown in Fig. 1. The process consists of five main components namely the software work product, the parser, the system knowledge base, the view composer and the software documentation.

A. Software work product (SWP)

The SWP can be the source code, the configuration files, build scripts or auxiliary artifacts. An auxiliary artifact can be a data gathering process, a manual, a job control and graphic user interface which helps to understand the source code. The author in [5], explains the approach for the redocumentation process and emphasizes on the importance of the software work product to produce a documentation for the different types of information.

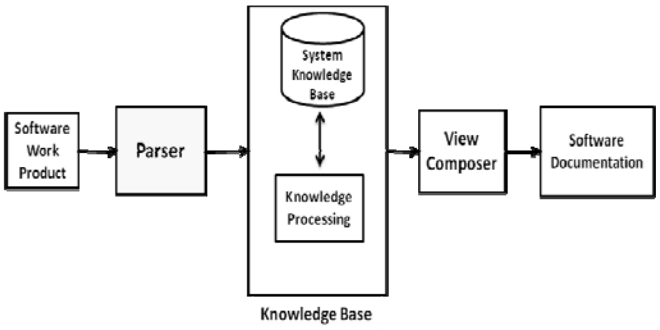


Fig. 1. Redocumentation process[1]

B. Parser

A parser is used to extract the necessary information from the SWP and store it into the repository or system knowledge base. The importance of the parser is to return the relevant information such as the parser used in [6] to produce the documentation. The parser is mostly used for extracting information from a specific language source code.

C. System Knowledge Based

According to [7], the knowledge base is a collection of simple facts and general rules representing some universal discourse. The purpose of this component is to store the extracted information from the SWP in order to describe the context of the information. This component becomes the heart of the system to allow the tools to access the required information. In other words, all parts of the model are able to access and organize the information from the knowledge base. Previous researches only focus on the knowledge base to make sure the knowledge retrieved mostly supports the most for program understanding. The knowledge base works closely with the knowledge processing to analyses the source code. This is important in extracting the required artifacts documenting the various relationships implicit in the recovering and revealing process of SWP[4]. The processed knowledge is represented by the different types of form such as the data modeling and the procedure or the function. The basic knowledge processing applied in many reverse engineering tools such as Rigi uses the RSF file as a repository and provides the knowledge presentation in a procedural form [8].

D. View Composer

Knowledge produced from the knowledge base is used by the composer to search and select the knowledge that is not presented explicitly. Generally, the view composer is used as a management tool to identify the needed components to produce a new software work product.

E. Software Documentation

Finally the processed knowledge is presented in various forms of documentation to the user (developer, maintainer, software engineer or end user). These range from the directed graph, the annotation, the visualization, and metrics or in documentation. The software components extracted include the modules, the procedures, the classes, the subclasses, the interface, the control flow, the composition and the enslavement. Software documentation can be categorized into the Textual and the Graphics. A textual documentation ranges from the inline style which is written in an informal mode to the personalized views dynamically generated from a document database[9]. HTML or XML are considered as the more flexible forms of the textual documentation which allows to automate the indexing and creates a hyperlink between the document partitions or sections [10, 11].The least mature type of the graphical documentation is a static image, which may use non-standard representations of the software artifacts and relationships[12]. The most advanced graphical documents are editable by the user, which enable them to create customized representations of the subject system. A software visualization technique is used to present the graphical documentation which helps the maintainer to understand the process.

III. Ontology Based Approach

The Ontology Based Approach is to produce a schema from the legacy system to describe the context of the software system or in terms of the domain specific concepts. The schema should able to capture the artifacts from the latest version of the software system by establishing a reverse engineering environment. The main objectives of this proposed approach are to:

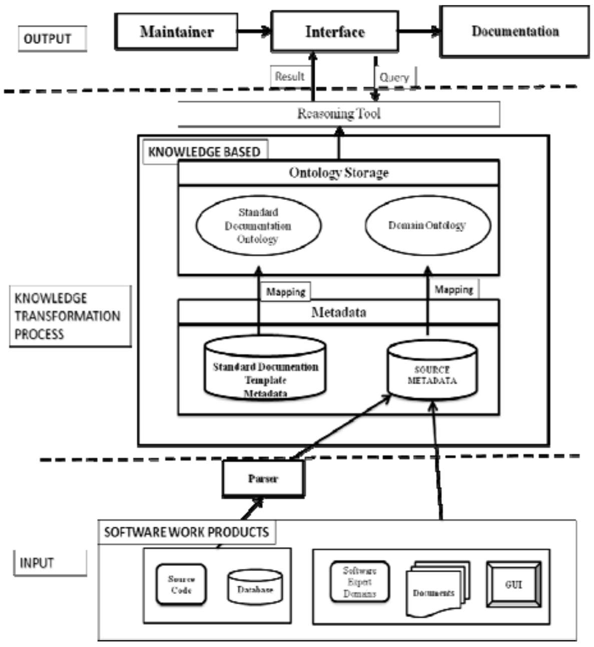


Fig. 2. Ontology based redocumentation framework

- present the domain components and describe the relevant concepts to establish the semantic relationship among the components.
- integrate the standard document templates and the

domain components to enhance better communication with the maintainers on the software system.
- enable the software maintainer to search for relevant information within a reasonable time frame.

The Ontology generates the vocabulary for the domain application in the form of a machine readable artifact. The propose framework is as shown Fig. 2 consisting of three levels which are the input, the knowledge transformation process and the outputs. Fundamentally, this approach focuses on the knowledge transformation process and output levels. The main sources which are needed are explained as follows.

A. Input

The input level consists of two main components namely: the source and the parser. The source consists of the source code, database, user manual and the software expert domains. Each component is described as follows:

- Source Code - is one of the most reliable components to understand the software system. However it is also required to be adaptable to changes in the program to make sure retrieved knowledge is the latest. Although most medium and large companies practice keeping a record on changes carried out on the source code, but the information is not stored in the repository or machine learning tool.
- Database – is another important source which helps to retrieve the related knowledge for the application domains. There are three major resources in the database that can be used for extracting and these are the data dictionary, the data and also the relationship between relations. The query on data and the system table allows the locating of all the three resources above.
- Documents - Documentations such as the system documentation and user manual are the available components which help to extract high level components. Software documents follow the proper document structure which organizes the chapter, sections and sub sections standard format that is used to extract the knowledge.
- Software Expert Domains - such as the maintainer or developer will be able to give input for a higher level of abstraction such as software specifications and requirements. In most organizations, the knowledge software experts keep their valuable knowledge related to the application domain in their mind. Surveys and interview techniques can be used to retrieve this valuable knowledge. Collected data will be transformed into the repository for analysis.

The next component in the input level is the parser. The parser is developed for the extraction of the components from the source code. The existing parser, the Island Grammar Approach is used to reduce the development time for the parser and the return of the relevant information. The Syntax Definition Formalism (SDF) can be used to define the Island Grammar. Using the SDF has a few advantages such as allowing for concise and natural definition of syntax and promoting the modularization of a syntax definition. The SDF extracts the related components and relationship from the source code. Manual analysis using the data dictionary provided by the relational database will be used to extract related data from the relational database.

B. Knowledge Transformation Process

There are three components in the knowledge transformation process which are the metadata, the ontology storage and the reasoning tool. The main concern on this level is to transform the knowledge stored in the schema to ontology. Next to support the user through reasoning tool to generate a better structure of semantic schema and provide efficient knowledge representation. The following describes each of these components in detail.

- Metadata – to change the metadata to ontology automatically, the data must be in a structured manner. Therefore the unstructured data retrieved from the source, must be transformed as the structured data. In the metadata there are two major components namely: the standard documentation template metadata and the source metadata. The standard documentation template needs to be defined by the user manually according to the standard document available in software development such as the MIL-STD-498. Each sub section in the standard document will be defined in the metadata. The user will be provided with the tools to allow the user flexibly in defining the metadata. The source metadata consists of the retrieved data from the different sources. The database and source code will be handled by the parser to extract the relevant components. However sources such as the developer, the user manual or the graphical user interface will be defined manually by the user in the source metadata using the interface module given. The main task of the source metadata is to save and manage the extracted components. The most flexible metadata used is the XML schema to wrap all the source and standard documentation in a structured data.
- Ontology - this is the heart of the framework in mapping the XML schema onto the ontology. Based on the ontology methodologies, the purpose will be defined as the first step. The purpose will define the knowledge which is relevant to the domain of interest. The conceptualization step needs to identify the basic attributes and competency questions. It structures the domain knowledge as meaningful models at the knowledge level either from scratch or by reusing the existing models. The competency questions identify the requirement that the ontology must address. This step leads to a preliminary ontology containing the description of kind, relations and properties. The next step will be the formalization which will transform the metadata to the formal or semi-computable model. A formal description language will be used to describe the ontology and will be integrated with the interface model. The implementation step builds the computable model in an ontology language. The final step is to validate and refactor the created ontology to make sure the quality and usefulness of the concept for a particular domain is achieved. The Web Ontology Language (OWL) will be used to define, describe and express the ontology. The steps described above will be implemented in creating the ontology for a standard documentation and the domain ontology. Even

though the ontology for the standard documentation and domain ontology are different, it is possible to surpass from one to another by using the domain components as bridge. The related components in the domain ontology are mapped into the section of the standard document. The maintainer will be given features to query and create the intersection between both the ontologies using the ontology reasoner.

- Ontology Reasoner - the maintainer will be given a tool with the ontology inference mechanism capability to search and query the conceptual and semantic levels to access the knowledge that is not presented explicitly. This is an important feature for the proposed approach compared to keyword-based search. The proposed approach makes use of the existing capability of the ontology using the inference rules to implement the intelligent retrieval and improves the rationality and usability of the results. Formalization of such query allows the maintainer to integrate both ontologies to produce the standard documentation containing the view of the software domain model.

C. *Output*

The output consists of the interface that is used to refine raw or semantically enriched standard documentation. Based on the domain ontology produced and query from the reasoning tool, the standard documentation can be produced such as the MIL-Std-498 standard system documentation. The results from the ontology can be mapped to predefine the components in a standard document structure such as the component description, using the case diagram and descriptions, possible requirement specification, data flow graph for specific use case, relationship among components using the semantic relationship from the domain point of view. The interface module will be equipped with functions such as the metadata management module, the ontology creator and the browser, the query editor and the documentation viewer. The Table I shows the description for each interface module.

TABLE I: USER INTERFACE DESCRIPTION

No	Interface	Description
1.	Metadata Management Module	Provides semi-automatic mechanism to create the metadata summarizing the artifacts received from the parser and the other sources. However other sources will be input manually as a metadata.
2.	Ontology Creator and Browser	The ontology creator is accomplished with the generator engine that reads the metadata in order to generate the ontology for the application domain.
3.	Query Editor	Provides the ontology based search or the inference mechanism to allow the user to process the required knowledge source.
4.	Documentation Management Module	Integrates with the query editor to extract the knowledge source from the domain ontology and the document it with the standard documentation.

IV. CASE STUDY

The implementation of this framework is still in the initial stage. A partial implementation of this framework only involves the components with the shaded box as shown in Fig. 3. The partial implementation emphasizes on developing only the domain ontology and generates the documentation for it. To have a clearer understanding on the partial implementation of the shaded box from the Fig. 3 which is extracted and shown in Fig. 4. The whole process is done semi automatically. The detail process of the implementation is described in the following sub sections.

A. *Input*

As shown in Fig. 4, the source chosen at this stage is only the source code. As we know the source code is more reliable compared to the other sources in the SWP. To evaluate the proposed framework a simple project is conducted by using the Visual Basic 6.0 (VB6). The project developed is for the management of a swimming training centre. Fig. 5 shows the sample modules and classes applied in this project. The lines of codes in this project are approximately 6000 lines. The sample VB6 code is given as an input to the parser termed as the Source Code Extractor and the Metadata Generator (SCEMG) through the Graphical User Interface (GUI) provided in the Fig.6. The SCEMG is divided to 2 main components:

- Source Code Extraction
- Metadata Generator

The Source Code Extraction basically reads and extracts the data required from the source folder. The source folder consists of the Visual Basic Project file, Forms, Modules and other related files. The extracted data is stored in database for metadata generation. Metadata Generator uses the data to generate the metadata as a form of strings in the text file. During this process, the data will be classified as the class hierarchy. This is one of the important steps towards generating ontology.

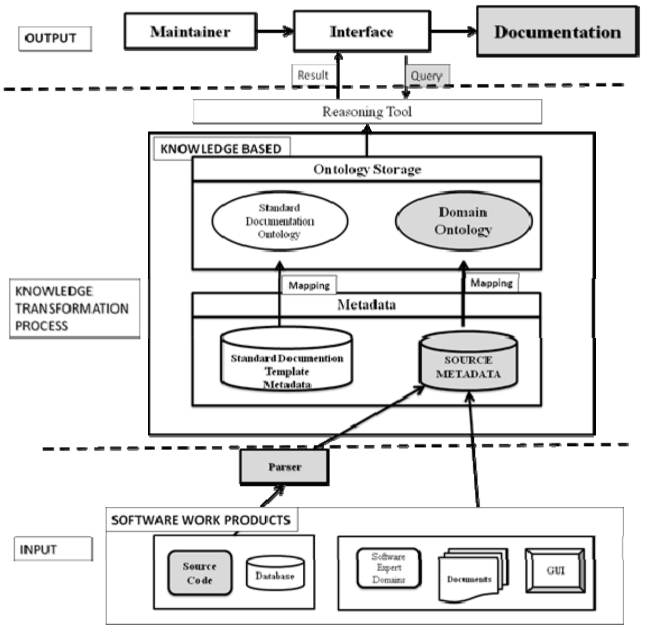


Fig. 3. Ontology based redocumentation framework – partial implementation (shaded box)

B. *Knowledge Transformation Process*

In this phase, the generated metadata will be transformed to the ontology. The strings are passed to the Protégé to store the metadata as an OWL ontology. A Protégé is the software development environment for the OWL ontology. The generated ontology can be refined to add axioms, object and data properties. The sample source code ontology is generated as shown in Fig.7 which is defined in the RDF/XML schema. However, to extract only the needed information the reasoning tool is used to query the ontology. Sesame Java API is used as a reasoning tool to query the ontology and retrieve the only needed information in the form of graphs and is represented in a triple pattern.

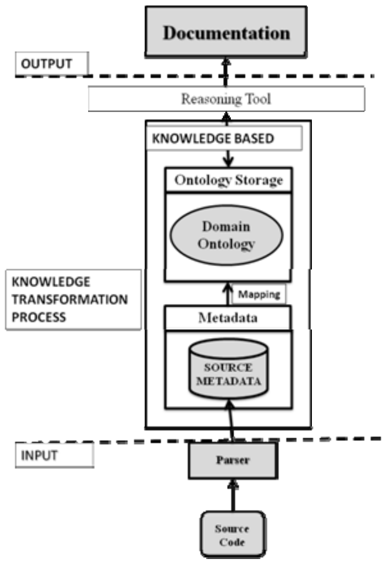


Fig. 4. A Partial Implementation of the Ontology Based Redocumentation Framework

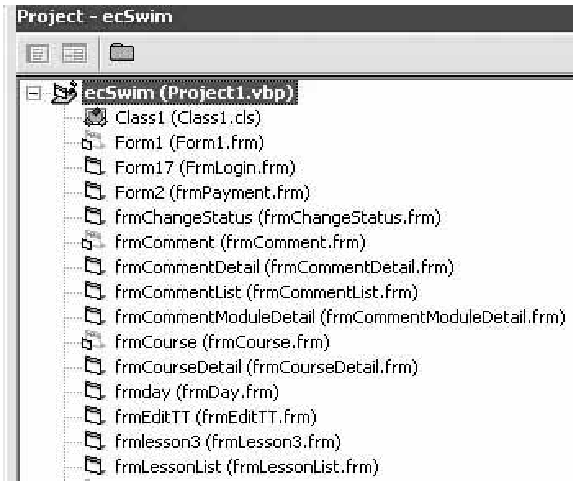
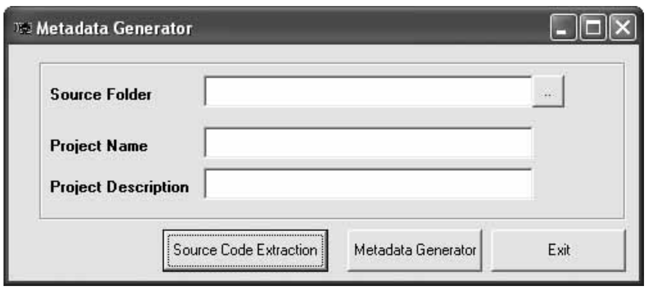

```
Project - ecSwim
ecSwim (Project1.vbp)
    Class1 (Class1.cls)
    Form1 (Form1.frm)
    Form17 (FrmLogin.frm)
    Form2 (frmPayment.frm)
    frmChangeStatus (frmChangeStatus.frm)
    frmComment (frmComment.frm)
    frmCommentDetail (frmCommentDetail.frm)
    frmCommentList (frmCommentList.frm)
    frmCommentModuleDetail (frmCommentModuleDetail.frm)
    frmCourse (frmCourse.frm)
    frmCourseDetail (frmCourseDetail.frm)
    frmday (frmDay.frm)
    frmEditTT (frmEditTT.frm)
    frmlesson3 (frmLesson3.frm)
    frmLessonList (frmLessonList.frm)
```

Fig. 5. Partial Project Modules and Classes in Visual Basic 6


Metadata Generator

Source Folder

Project Name

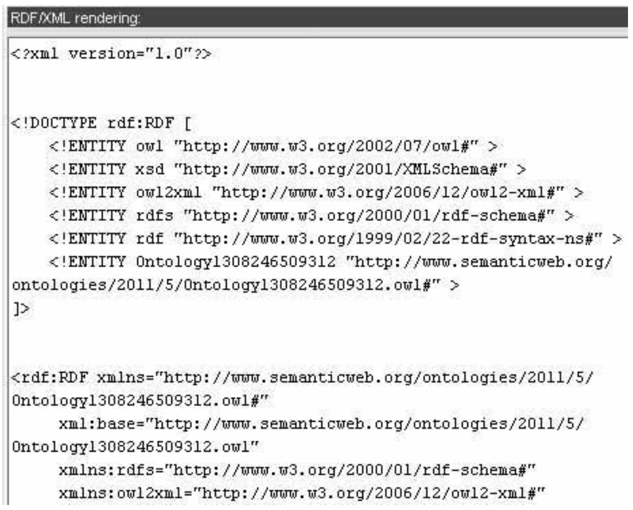
Project Description

Source Code Extraction | Metadata Generator | Exit

Fig. 6. GUI for SCEMG

C. Output

In this phase, the initial implementation only generates the basic documentation as shown in Fig.8 which is generated by using the OWLDoc. The OWLDoc is implemented for the Protégé-OWL to generate the documentation for the OWL ontologies. The generated documentation represents the Abstract Syntax of the classes, properties and individuals.

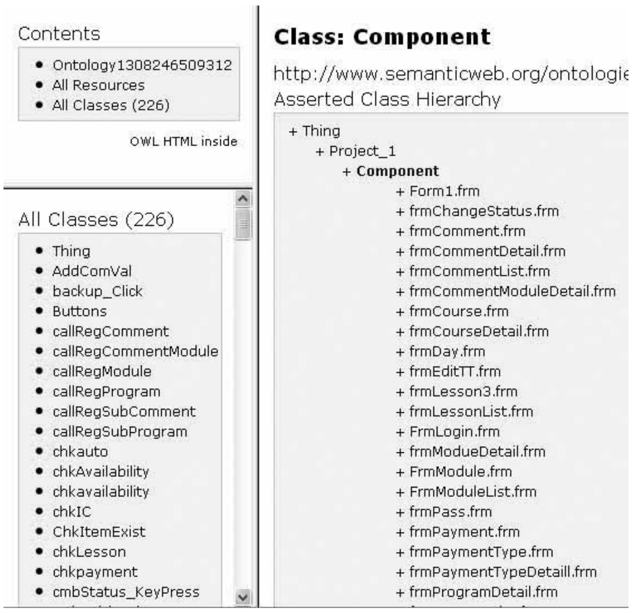

```
RDF/XML rendering:

<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY Ontology1308246509312 "http://www.semanticweb.org/
ontologies/2011/5/Ontology1308246509312.owl#" >
]>


<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2011/5/
Ontology1308246509312.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2011/5/
Ontology1308246509312.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
```

Fig. 7. Sample VB6 Source Code Ontology


```
Contents
 • Ontology1308246509312
 • All Resources
 • All Classes (226)
                      OWL HTML inside

All Classes (226)
 • Thing
 • AddComVal
 • backup_Click
 • Buttons
 • callRegComment
 • callRegCommentModule
 • callRegModule
 • callRegProgram
 • callRegSubComment
 • callRegSubProgram
 • chkauto
 • chkAvailability
 • chkavailability
 • chkIC
 • ChkItemExist
 • chkLesson
 • chkpayment
 • cmbStatus_KeyPress

Class: Component
http://www.semanticweb.org/ontologie
Asserted Class Hierarchy
+ Thing
    + Project_1
        + Component
            + Form1.frm
            + frmChangeStatus.frm
            + frmComment.frm
            + frmCommentDetail.frm
            + frmCommentList.frm
            + frmCommentModuleDetail.frm
            + frmCourse.frm
            + frmCourseDetail.frm
            + frmDay.frm
            + frmEditTT.frm
            + frmLesson3.frm
            + frmLessonList.frm
            + FrmLogin.frm
            + frmModueDetail.frm
            + FrmModule.frm
            + FrmModuleList.frm
            + frmPass.frm
            + frmPayment.frm
            + frmPaymentType.frm
            + frmPaymentTypeDetaill.frm
            + frmProgramDetail.frm
```

Fig. 8. Generated Sample Documentation for Source Code Ontology Using OWLDoc

V. Related Work

Some research related to the software redocumentation has been carried out to improve the understanding of the software system. Jochen et al., 2001 [13] used the XML as knowledge based to extract the artifacts from the source code and produced the documentation in line with to the users' needs. This approach has become a common solution for redocumentation because of the advantage of the XML which can easily transform to different medium. But the abstraction is in the medium level and it will be difficult to show semantically related knowledge of the same domain. Feng and Hongji [14] used Model Driven Engineering (MDE) approach for the extraction of software components in the higher level abstraction such as the class diagram and the activity diagram. However the output of this approach does not use any standard documentation to present the overall output. One of the common issues in maintaining the system is to record the changes requested by customer or user occurring in the source code. Rajlich [15] used the incremental redocumentation approach to solve this problem and recorded in the skeleton software documentation. But the abstractions are in the low level and not used by any standard documentation. Shahida Sulaiman [16] presents DocLike Modularized Graph (DMG) approach to visualize the document-like software and the re-documentation imparts the template of the software design documentation that enables users to directly update the description of each section in the Description Panel and its associated graph that is generated automatically. The nearest works to our approach is transforming the legacy schema to the ontology using the meta- framework approach. As described by a Manual Winner, this meta-framework can generate the ontology to improve structures and semantics compared to the original legacy schemas[17]. To produce semantic enrichment of the schemas, this method transforms the legacy scheme using heuristic and refactoring to improve the design and precision of the schemas. However, this method emphasis only on the knowledge base and output level and did not emphasize on the importance of the SWP that can help enrich the metadata to generate ontology and not emphasize on producing the standard documentation or ontology to describe the artifacts in meaningful way.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented an ontology based approach to redocument the source code and other related software components to a semantic level and mapped it to the software documentation. As an initial effort, the implementation of the framework only emphasizes on the source code. However the process for extracting other SWP components and also developing the documentation ontology (no shaded box in the Fig.3) are still the same process as developing the source code ontology and presentation to the HTML documentation. Our next task is to complete the implementation of this framework and apply the framework for other types of SWP such as the object oriented database. However the implementation will not be fully automated and needs some effort from the user to be involved especially in the input level and knowledge base.

REFERENCES

[1] Elliot, J.C. and H.C. James, II, *Reverse Engineering and Design Recovery: A Taxonomy.* IEEE Softw., 1990. **7**(1): p. 13-17,doi:http://dx.doi.org/10.1109/52.43044.

[2] Benedusi, P., A. Cimitile, and U. De Carlini. *A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance.* in *Software Maintenance, 1989., Proceedings., Conference on*. 1989.

[3] K.Lano, H.H., *Reverse Engineering and Software Maintenance : A Practical Approach*. 1994: Mc-Graw Hill, London.

[4] Inc., B.S. *Reverse Engineering.* 2010 [cited 2010 02/03/2010]; Available from: http://bus-software.com/re.htm.

[5] Shihong, H., et al. *Adoption-Centric Software Maintenance Process Improvement via Information Integration.* in *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*. 2005. Budapest

[6] Arie van, D. and K. Tobias. *Building Documentation Generators*. in *Proceedings of the IEEE International Conference on Software Maintenance*. 1999: IEEE Computer Society.

[7] Shirabad, J.S. *Supporting Software Maintenance by Mining Software Update Records*. in *17th IEEE International Conference on Software Maintenance (ICSM'01)*. 2003. Canada: University of Ottawa.

[8] Holger M. Kienle, H.A.M.u., *The Rigi Reverse Engineering Environment*, in *Proceedings of the International Workshop on Advanced Software Development Tools and Techniques*. 2008, ECOOP: Paphos, Cyprus.

[9] Ferenc, R., *Columbus – Reverse Engineering Tool and Schema for C++*. 2002

[10] Scott, T. and H. Shihong, *Documenting software systems with views III: towards a task-oriented classification of program visualization techniques*, in *Proceedings of the 20th annual international conference on Computer documentation*. 2002, ACM Press: Toronto, Ontario, Canada.

[11] Tadonki, C. *Universal Report: a generic reverse engineering tool*. in *12th IEEE International Workshop on Program Comprehension (IWPC'04)*. 2004.

[12] Müller, H.A. *Rigi User's Manual*. 1996 July 10, 1996 [cited 2007 July 18,2007]; Available from: http://www.rigi.csc.uvic.ca/ downloads/rigi/doc/user.html.

[13] Jochen, H., H. Shihong, and T. Scott. *Documenting software systems with views II: an integrated approach based on XML*. in *Proceedings of the 19th annual international conference on Computer documentation*. 2001. Sante Fe, New Mexico, USA: ACM Press.

[14]Feng, C. and Y. Hongji. *Model Oriented Evolutionary Redocumentation*. in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*. 2007.

[15] Rajlich, V., *Incremental redocumentation using the Web.* Software, IEEE, 2000. **17**(5): p. 102-106

[16] Shahida Sulaiman , N.B.I., and Shamsul Sahibuddin, *Enhancing Cognitive Aspects of Software Visualization Using DocLike Modularized Graph*, in *The International Arab Journal of Information Technology*. 2005.

[17] Wimmer, M. *A Meta-Framework for Generating Ontologies from Legacy Schemas*. in *Database and Expert Systems Application, 2009. DEXA '09. 20th International Workshop on*. 2009.

Sugumaran Nallusamy was born in Kuala Lumpur, Malaysia in 1977, educated at University Technology Malaysia (UTM), Kuala Lumpur, Malaysia (Bsc in computer science 2000 and MSc in Real-Time Software Engineering 2002). Currently is persuing his PHD in Software Engineering at UTM, Kuala Lumpur, Malaysia.

He is also a lecturer in the Faculty of Engineering and Science, Universiti Tunku Abdul Rahman(UTAR) in Kuala Lumpur, Malaysia since 2003. He has published a few papers in the field of software redocumentation namely : An Evaluation of Redocumentation Approaches and Tools Using Knowledge Representation Criteria (Bali, Indonesia, University of Indonesia, 2010) and An Evaluation on Software Redocumentation Approaches and Tools in Software Maintenance (Cairo, Egypt, IBIMA Publishing, 2011). His field of interest is reverse engineering and redocumentation for software maintenance.

Mr.Sugumaran is a member of IEEE since January 2011. He has been awarded with the Best Session Presentation at the International Conference on Advanced Computer Science and Information System, Indonesia in 2010.

Suhaimi Ibrahim was born in Kelantan, Malaysia in 1957, educated at the University of Strathclyde. Glassgow and received the Bachelor in Computer Science (1986), University Technology of Malaysia, Malaysia received Master in Computer Science (1990) and PhD in Computer Science (2006).

He is an Associate Professor attached to Department Off Software Engineering, Advanced Informatics School (AIS), Universiti Teknologi Malaysia International Campus, Kuala Lumpur. He currently holds the post of Deputy Dean of AIS. He has published many articles in international conferences and international journals such as the International Journal of Web Services Practices, Journal of Computer Science, International Journal of Computational Science, Journal of Systems and Software, and Journal of Information and Software Technology. His research interests include

software testing, requirements engineering, Web services, software process improvement, mobile and trusted computing.
Suhaimi Ibrahim is a member of the IEEE. He also an ISTQB certified tester and being appointed a board member of the Malaysian Software Testing Board (MSTB).

Mohd Naz'ri Mahrin was born in Kuala Lumpur, Malaysia in 1977, educated at the University Technology Malaysia (UTM), Kuala Lumpur, Malaysia (Bsc in Computer Science 1997 and MSc in Real-Time Software Engineering 2000). In 2010, he completed his PhD degree in Software Engineering from the University of Queensland, Australia.
He has been working with the Advanced Informatics School (formerly known as Centre for Advanced Software Engineering) since 2001. He teaches several postgraduate courses including software process and quality, requirements, design, testing, and secure software development. He also supervises master student projects. His research interests include software engineering process and quality, software measurement, usability evaluation, and information security management. His current research work (as a project leader) is on test coverage analysis and ontology-based software redocumentation. He is one of the auditors who are responsible to implement and improve the proposed framework during the period of pilot study with the Malaysian software industries. Currently he is an associate consultant with several certification bodies which are authorized to conduct a formal assessment based on the framework including TUV Rheinland Malaysia, SIRIM QAS International Sdn. Bhd, and SGS Malaysia.
Mohd Naz'ri Mahrin is a member of the IEEE, IEEE Computer Society, Malaysian Software Engineering Interest Group (MySEIG), and Information Security Professional Association of Malaysia (ISPA).