World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:1, 2010

Mapping Knowledge Model Onto Java Codes

B.A.Gobin , R.K.Subramanian

***Abstract*—**This paper gives an overview of the mapping mechanism of SEAM-a methodology for the automatic generation of knowledge models and its mapping onto Java codes. It discusses the rules that will be used to map the different components in the knowledge model automatically onto Java classes, properties and methods. The aim of developing this mechanism is to help in the creation of a prototype which will be used to validate the knowledge model which has been generated automatically. It will also help to link the modeling phase with the implementation phase as existing knowledge engineering methodologies do not provide for proper guidelines for the transition from the knowledge modeling phase to development phase. This will decrease the development overheads associated to the development of Knowledge Based Systems.

***Keywords*—**KBS, OWL, ontology, knowledge models

Open Science Index, Computer and Information Engineering Vol:4, No:1, 2010 publications.waset.org/10085.pdf

I. Introduction

THE area of KBS development has matured over the years. It started with first-generation expert systems with a single flat knowledge base and general reasoning engine, typically built in a rapid-prototyping approach [1]. It was essentially based on the process of knowledge transfer [2]. Maintenance of such systems was very difficult. Hence the approach changed to a methodological approach which was similar to that of software engineering with knowledge as its main focus. Knowledge Engineering is no longer simply a means of mining the knowledge from the expert. It now encompasses *methods and techniques for knowledge acquisition, modelling, representation and use of knowledge* [3].

However KBS development still remains complex and has not gained success as compared to application developed in the software engineering field. Methodologies like CommonKADS [3], Protégé[4], MIKE [5], and MOKA [6] and knowledge base development environments like IBROW3 [7] and EXPECT[8] all face criticisms.

However they face some common criticisms which are as follows:

- Knowledge modelling is tedious because of it requires a good understanding of AI concepts and also because of knowledge acquisition problems. [9]
- Development overheads due to complexity of certain methodologies and development environment. [9][10]
- No Knowledge modelling standards – modelling is done in an ad-hoc way based on the experience of the knowledge engineer. [11]
- Concept of reuse through PSM is difficult to implement. [10]
- Lack of guidelines for the proper transition from knowledge modelling to implementation phase in the methodology. [13][10][11]

These criticisms are associated to the knowledge modelling process which is considered to be one of the most important activities for the development of KBS. To decrease some of the problems associated with the development of KBS, we are currently working on a methodology called SEAM that will help in the modelling through the semi-automatic generation of knowledge models and the mapping of these models onto Java codes so as to create a prototype which will be used for the validation of the knowledge model. The generation of the Java codes will also help to decrease the complexities associated with the transition from the knowledge modelling to the implementation phase as once the prototype has been developed it can be used as to develop the full version of the KBS.

The aim of this paper is to explain the rules for mapping of the different components of the knowledge model on the respective Java codes. It first gives a small overview of the SEAM methodology and the ontology representing the knowledge model before going into details about the various mapping rules.

II. SEAM - A Methodology for the Knowledge Modelling

SEAM is a 4-step methodology that used for the semi-automatic generation for knowledge models. The aims of the methodology are to:

- decrease complexities associated with learning AI concepts. The complexity of AI concepts is one of the main reasons why development of KBS still remains difficult. Since the generation of the knowledge model and the mapping will be automated much of the learning and development overheads will be decreased. There is no need to learn AI concept in depth.
- have simple steps so that inexperienced knowledge engineers or even domain experts can go through the knowledge modeling process. We attempt to ease the knowledge modelling process so that it no longer stays an activity reserved only to the more privileged.

B. A. Gobin (phone: 230-4037893 e-mail: b.gobin@uom.ac.mu) and R. K. Subramanian (phone: 230-4541041 fax: 230-4657144 e-mail: rks@uom.ac.mu) are with the Department of Computer Science and Engineering of the University of Mauritius.

Open Science Index, Computer and Information Engineering Vol:4, No:1, 2010 publications.waset.org/10085.pdf

- standardize knowledge modeling process. There is a lack of standard for modeling, lack of formalism for domain knowledge, and lack of standards to represent rules. We thus aim at bringing standards to the knowledge modeling process through the use of semantics. The automatic generation of the knowledge models will also contribute to achieve this. Thus the modeling process will be done in an ad-hoc way and will not depend on the experience of the knowledge engineer.

- link knowledge model phase to implementation phase through the mapping of the knowledge model onto Java codes so that a quick prototype can be build which can be used to validate the knowledge model

There are four distinct steps involved when using SEAM for the modeling of KBS and its mapping onto Java codes. The steps are as follow (Figure 1):

- *Select the task that the KBS needs to do*
  The domain expert/knowledge engineer has to first select the task for the application that needs to be developed. The generation of the knowledge model is based on the template knowledge models that are provided. We have developed an ontology in OWL which is a library containing the template knowledge models proposed by CommonKADS. CommonKADS supports the partial reuse of knowledge models to support the knowledge modelling process. As compared to software engineering, knowledge intensive task are limited and can be categorised as shown in Figure 1. The template knowledge models are generic knowledge models which are representing different task which are then adapted to the domain of application. It is therefore important that the selected task is found in the template knowledge model. Hence a task selection mechanism has been developed so that the template knowledge model can be extracted from the ontology "templateknowledgemodel.owl" [14].

- *Extract the template knowledge model*
  On selection of the task the generic knowledge model for the particular task is extracted and generated. The application ontology which is basically an empty rdf file, "applicationknowledgemodel.owl", is first generated. Classes, properties and instances representing the knowledge model for the selected task is then extracted from "templateknowledgemodel.owl" and added to "applicationknowledgemodel.owl".

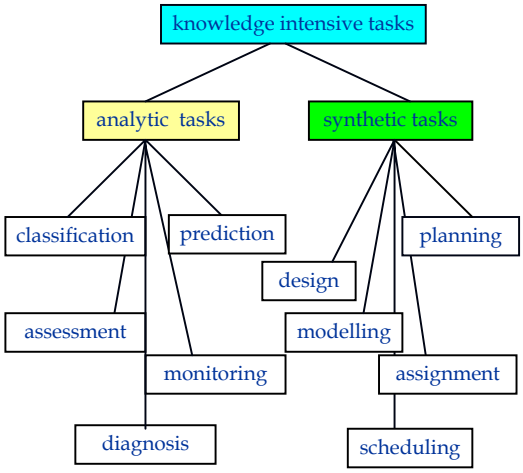

Fig. 1 Hierarchy of knowledge-intensive task types on the type of problem being solved

- *Adapt the knowledge model to the domain of application*
  The template knowledge model contains a generic domain schema which is independent from the application domain knowledge schema. The generic domain knowledge contains information about the domain schema and the 'ruletype' that need to be created for the knowledge model of a particular task. The domain knowledge can be added to the application knowledge model either manually or automatically using existing ontologies representing the domain of application. The aim of using domain ontologies is to allow the knowledge engineer to reuse existing ontologies. Based on these and input from the knowledge engineer/domain expert the domain schema and the rules for the application will be generated using the mechanism for the semi-automatic generation of domain schema and the mechanism for the semi-automatic generation of rules. The application knowledge model "applicationknowledgemodel.owl" is thus updated with the domain knowledge through these mechanisms.

- *Map the knowledge model onto Java codes*
  The aim of this mechanism is to ease the transition from the analysis to the implementation phase. The application ontology is mapped onto Java codes that will be used to implement the knowledge based system. The implementation team will have to go through the finetuning of the codes and the development of user interfaces of the KBS for input and output purposes.

III. Application Knowledge Model in OWL

The application knowledge is therefore obtained from the template model ontology has been developed in OWL[15]. It consists of the task, domain and inference knowledge.

Open Science Index, Computer and Information Engineering Vol:4, No:1, 2010 publications.waset.org/10085.pdf

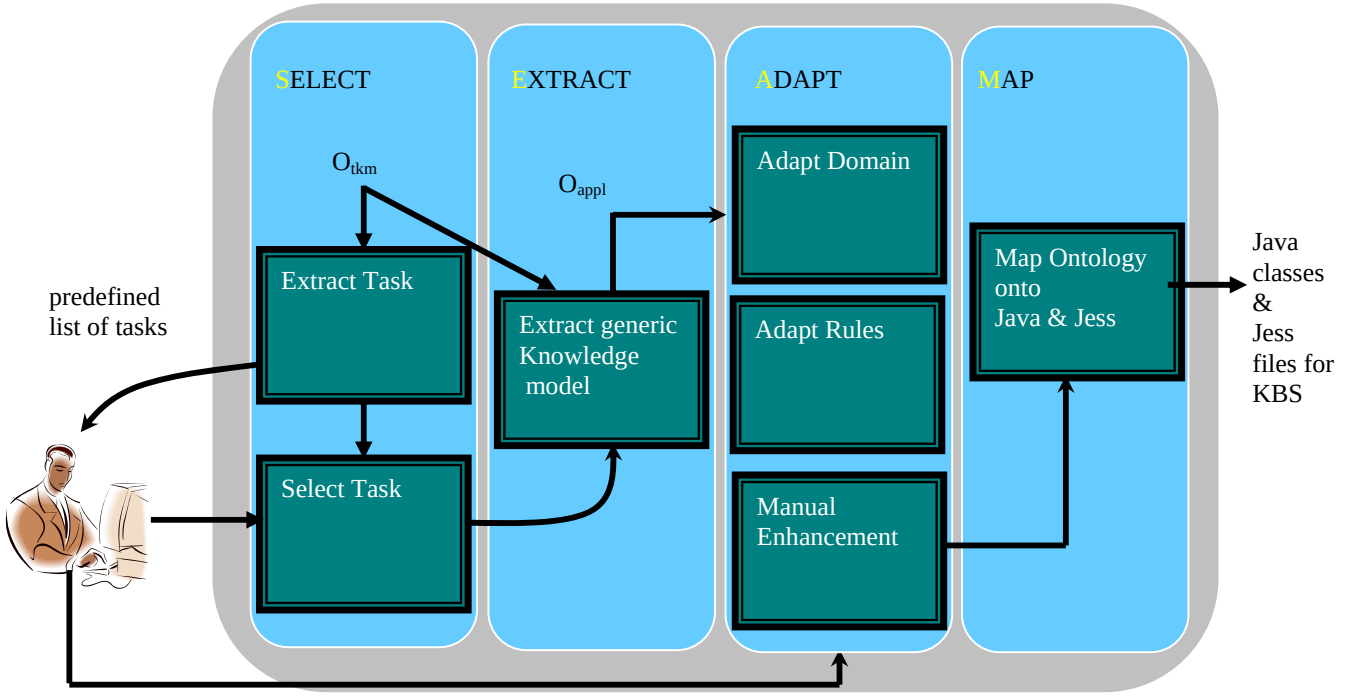

Fig. 2 SEAM Methodology

This section gives an overview of the application knowledge model generated for the task "assessment".

A. Representing domain knowledge

Subclasses for the class "*concepts*" are as follows:
casedescription ***concepts***
casecriterion ***concepts***
casedecision ***concepts***
assessment_rule_type ***rule_type***

The instances of assessment_concepts contain all the concepts that are required for the "assessment" task. Table 1 contains the different classes that are used to represent the domain knowledge, the properties of the classes and some example of the instances.

TABLE I PROPERTIES AND INSTANCES OF CLASSES REPRESENTING THE DOMAIN KNOWLEDGE

Class	concepts	rule_type
Properties	☐ specification (DP)	☐ specification (DP) ☐ has_concept1 (OP) ☐ has_concept2(OP)
Instances	e.g. instances of class "assessment_concepts" : ☐ applicant ☐ case_criterion ☐ case_decision	e.g. instances of class "assessment_relation" : ☐ application

B. Representing the inference knowledge

The class "*inference*" has as subclasses the different inferences that are found in the catalogue provided by CommonKADS. The subclasses are defined based on general inferences. Several task methods can call inferences bearing the same name e.g. the inference "*select*" is called in the task method for "*assessment*" and "*diagnosis*". Therefore in our ontology representing the application knowledge model, we will have class "*select*" which is a subclass of the class "*inference*" which h"*assessmentselect*" which is the inference that will be called when modelling for the task "*assessment*". The same applies for other inferences which are called in different task methods e.g. inference "*specify*. Subclasses for the classes "*inference*", "*role*" and "*statement*" are as follows:

abstract ***inference***
match ***inference***
assessment_role ***role***
assessment_statement ***statement***

Table 2 contains the different classes that are used to represent the inference knowledge, the properties of the classes and some example of the instances.

C. Task Knowledge represented in OWL

Each knowledge intensive task as per CommonKADS catalogue is represented as subclass of the main class task. The subclass of the main class "***task***" in the application knowledge modele is: *assessment* ***task***. Subtasks of the knowledge intensive task are then instances of the subclass created. E.g. for the knowledge intensive task *assessment* has two subtask : *abstractcase and matchcase.* Therefore A subclass "*assessment*" is created which has as instance "*abstractcase*" and "*matchcase*". The same applies for all

Open Science Index, Computer and Information Engineering Vol:4, No:1, 2010 publications.waset.org/10085.pdf

other components of the knowledge model except for inferences e.g. to represent task methods we have a class *"taskmethod"* which has a subclass *"assessmenttaskmethod"* and instances *"abstactmethod"* and *"matchmethod"*.

Table 3 contains the different classes that are used to represent the task knowledge, the properties of the classes and some example of the instances.

TABLE II PROPERTIES AND INSTANCES OF CLASSES REPRESENTING THE INFERENCE KNOWLEDGE

Class	***Inference***	***role***
Properties	☐ has_input_role(OP) ☐ has_output_role(OP) ☐ has_static_role(OP) ☐ specifications(DP)	☐ type (OP) ☐ domain_mapping(OP)
Instances	e.g. instances of class "abstract" : ☐ assessment_ abstract	e.g. instances of class "assessment_role" : ☐ abstracted_case ☐ abstraction_knowledge ☐ case_description

IV. RULES FOR MAPPING KNOWLEDGE MODEL ONTO JAVA CODES

In this section we discuss the different rules for the mapping of the different components of the knowledge model onto Java codes. We take as case study the "housing assessment" example found in [3].

A. Rules for Creating Java Package

Rule 1: The main classes in the O_{appl} are used to create equivalent java packages

The different java packages for the application will be built based on the names of the main classes found in the application knowledge model. Different directories bearing the name of the classes are created i.e. "task", "taskmethod", "inference", "role", "controlstructure", "concepts", "relation", "ruletype". Not all them will be however required e.g. controlstructure, relation and ruletype will not be required. But since we do not want to hardcode which package are needed Rule 1 applies to all main OWL class. All unnecessary packages directories can be deleted afterwards.

Rule 2: A package is created to run the Jess Engine

This package is necessary as it will contain all the classes that will be required to run the Jess Engine for different inferences. This is not defined at modeling level. It is to be noted that irrespective of what type of task the application knowledge model is based on, the creation process of packages will always be same.

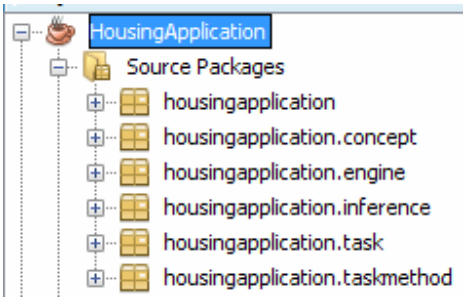

Figure 3: Packages created

Fig. 1 shows all the java packages that will be created

B. Rules for Mapping Domain Knowledge

Rule 3: Create Java classes for all classes and instances of the OWL class "concept" found in O_{appl}

Java classes will be created for subclasses and instances of the OWL class "concept". For the "assessment" example, the following Java classes are created for the OWL class: "casedescription", "casecriteria" and "casedecision" i.e. the following java files will be created "casedescription.java", "casecriteria.java" and "casedecision.java".

Java classes will also be created for the instances of the classes. E.g. rentfitsincome which is an instance of the class "residencecriterion" in O_{appl} will be mapped on the Java class "RentFitsIncome.java" and will be a subclass of "ResidenceCriterion.java" . Figure 4 gives the list of all the classes created based on the domain knowledge for the "housing

public class **RentFitsIncome** extends **ResidenceCriterion** {}

TABLE III PROPERTIES AND INSTANCES OF CLASSES REPRESENTING TASK KNOWLEDGE

Class	***Task***	***TaskMethod***	***Control Structure***	***Statement***
Properties	☐ goal (DP) ☐ has_input_role (OP) ☐ has_output_role (OP) ☐ has_task_method (OP) ☐ has_order	☐ has_inference(OP) ☐ has_control_structure (OP) ☐ has_intermediate_role (OP)	☐ has_statement (OP)	☐ has_action(DP) ☐ has_statement_order (OP) ☐ has_condition_ inference(OP) ☐ has_control_condition (OP) ☐ has_action_inference (OP) ☐ has_control_structure (OP) ☐ has_control_loop(OP)
Instances	e.g. instances of class "assessment" : ☐ abstract_case ☐ match_case	e.g. instances of class "assessment_method" : ☐ abstract_case ☐ match_case	e.g. instances of class "assessment_cs" : ☐ abstract_cs ☐ match_cs	e.g. instances of class "abstract" abstracted_case ☐ assessment_ statement1

Open Science Index, Computer and Information Engineering Vol:4, No:1, 2010 publications.waset.org/10085.pdf

Figure 4 shows all the java classes that will be created to represent concepts.

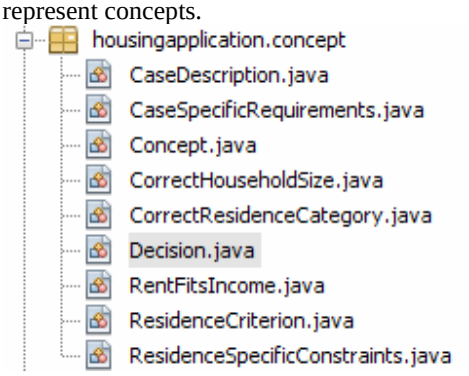

Fig. IV Java classes in "concept" package

Rule 4: Create Java classes for all classes and instances of the OWL class "concept" found in O_{appl}

The methods to "get" and "set" properties associated to a concept will also be created based on the properties in the OWL knowledge model.

Rule 5: SRWL rules are mapped onto Jess rules

The SWRL rules will be mapped into Jess rules in ".clp" files. These ".clp" files will be called by the Jess engine.

C. Rules for Mapping Inference Knowledge

Rule 6: Instances of the class "inference" in O_{appl} are mapped onto Java classes and the values of the properties "has_input_role", "has_output_role", "has_static_role" are mapped as input and output parameters of the methods of the class.

Java classes are created for each instance of the "inference" class. To carry out the "assessment" task there are five inferences (Table 2). Therefore five java classes will be created namely : "abstract.java", "evaluate.java", "select.java", "specify.java" and "match.java" as shown in Figure 5. Each inference has input, output and static roles. These roles will be set as input and output parameters of the methods associated to the class. E.g. for the "abstract" inference.

```
public class Abstract1 {
  public CaseDescription Abstract1(CaseDescription
  casedescription, CaseDescription abstractedcase, String
  staticknowledge ) }
```

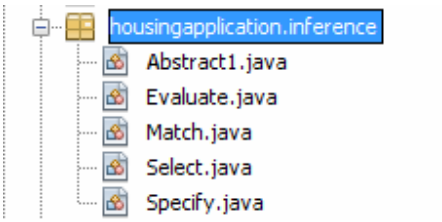

Figure 4: Classes created in "inference" package

D. Rules for Mapping Task Knowledge

Rule 7: Java classes are created for subclasses and instances of the following classes: "task", "taskmethod"

Java classes are created for the subclasses and instances of the class "task" and "taskmethod" found in O_{appl}. E.g. for "assessment" task the Java classes will be created as shown in Figure 4.

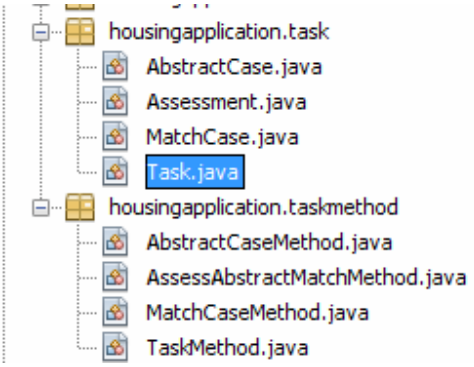

Fig. 5 Classes created in "task" and "taskmethod" packages

Rule 8: Each "task" will call a "taskmethod"

The instance class "task" in the OWL file has an object property "has_method" which has as value the name of the corresponding task method.

Rule 9: TaskMethods will either call a subtask or will execute one or a series of inferences

Task methods either call a particular subtask or a series of inferences depending of whether they are represented as a subclass or an instance in O_{appl}. If it is a subclass then, it will call subtasks, which are the instances of the class "task" in O_{appl}. The different subtasks are called based on the order which can be obtained from "has_order" property of the class "task" in O_{appl}.

On the other hand if the java file represents instances of the class "taskmethod" in O_{appl}, then the java file will contain all the statements of the control structure associated to it. The control structure associated to it is obtained from the value of the property "has_control_stucture. The statements in each control structure will be mapped onto Java codes. Hence

inferences will be called in the task methods as defined in the statements.

V. DISCUSSION AND FUTURE WORKS

Before coming to the mapping rules, a test KBS was built in Java as well as the knowledge model was created in OWL for "housing assessment" task, i.e. assessing whether someone is eligible to get a house based on certain predefined criteria. We then proceeded with establishing the rules for the mapping from OWL to Java to see to what extent complete automation is possible based on the rules that we have conceptualized. Therefore the rules for the mapping mechanism are based on the "assessment" task. We believe that the mapping rules will remain the same for all other types of knowledge intensive task, as long as the knowledge model follow the format required for mapping. However we shall test the mapping with other tasks once all the rules have been implemented. Also since we are still at the conceptualization phase, some rules may be subject to slight changes as we proceed with the development mechanism.

Another interesting aspect of this mapping approach is that reuse is possible. Since the mapping is modular, and the application knowledge model is generic to a great extent – only the domain knowledge changes, the generated Java codes can be reused. If a knowledge engineer has already built a "Housing Assessment" application for a particular situation and now needs to build another "assessment" application, changes can normally be done in the domain package only. It thus decreases development overheads and cost and saves time.

We have already started the mapping process and Rule 1 and Rule 2. We are now going ahead the implementation of the other rules. We believe that the difficulty will lie mainly in the implementation of the control structures and the Jess files. Since we want to maximize automation we want to develop a completely generic approach for the mapping of the control structures too. To what extent the rules we are putting forward will work can only be known when we start working on the rules for mapping the task knowledge.

REFERENCES

[1] P. Speel, A.T Schreiber., W Van Joolingen., Van Heijstg, and J.Beijer G.,. "onceptual Modelling For Knowledge Based Systems" Encyclopedia Of Computer Science And Technology, Marcel Dekker Inc. New York, 2001.

[2] R. Studer,, V.R. Benjamins, and D. Fensel,, "Knowledge Engineering:Principles And Method", Data & Knowledge Engineering, 1998, pp. 161-197.

[3] A. Th. Schreiber, J. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. Van De Velde and B. Wielinga, Knowledge Engineering and Management: The Commonkads Methodology, Mit Press, 2000.

[4] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy and Tu, S. W., "The Evolution Of Protege: An Environment For Knowledge-Based Systems Development", International Journal Of Human Computer Studies, 58(1), 2003, pp. 89-123.

[5] Angele J., Fensel D., Landes D. And Studer R. 1998. "Developing Knowledge Based Systems with Mike", Journal Of Automated Software Engineering, 5(4), 1998, pp.389-418.

[6] M. Callot, Methodology And Tools Oriented To Knowledge Engineering Applications, Moka Public Report No.2, 1999 [Online]. Available From: Http://Www.Kbe.Conventry.Ac.Uk/Moka [Accessed 19 October 2009].

[7] D. Fensel, E. Motta, V. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso , F. Van Harmelen, M. Musen, E. Plaza, G. Schreiber, R. Studer, A. Ten and B. Wielinga,"An Intelligent Brokering Service for Knowledge Component Reuse on the World-Wide Web", In: The 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW98), Banff, Canada, 1998.

[8] Y. Gil, J. Blythe, J. Kim And Ramachandran S., "Acquiring Procedural Knowledge in EXPECT". In: Proceedings of AAAI 2000 Fall Symposium on Learning How to Do Things, North Falmouth, Massachusetts, AAAI, 3-5 November 2000.

[9] H. Knublauch, An Agile Development Methodology For Knowledge-Based Systems Including A Java Framework For Knowledge Modeling And Appropriate Tool Support, Dissertation (Phd Thesis), University Of Ulm, 2002.

[10] CORSAR, D. and SLEEMAN, D. "KBS development through ontology mapping and ontology driven acquisition" In Proceedings of the 4th international Conference on Knowledge Capture, 28 – 31 October 2007 Whistler, BC, Canada, 2007, 3-30.

[11] M.S. Abdullah, A. Evans, I. Benest, R Paige and C. Kimble, "Modelling Knowledge Based Systems Using the eXecutable Modelling Framework(XMF)", In: Proceedings of the 2004 IEEE, Conference on Cybernetics and Intelligent Systems,1-3 December Singapore, IEEE, 2004,pp 1055-1060.

[12] D. Sleeman, T. Runchie and P. Gray, "Reuse: Revisiting Sisyphus-VT". In Staab, S and Svatek, V, Eds. Proceedings EKAW 2006 Conference Podebrady, Czech Republic, 2006, pp 59-66.

[13] R. Benjamins, D. Fensel, C. Pierret-Golbreich, E. Motta, R. Studer, B. Wielinga, M. Rousset. "Making knowledge engineering technology work". In Proc. of the 9th Int. Conf. on oSoftware Engineering and Knowledge Engineering (SEKE-97), Madrid, Spain, 1997.

[14] B.A.Gobin, R.K.Subramanian, "An OWL Ontology for CommonKADS Template Knowledge Model". In Proc. of the International Conference on Knowledge Systems Engineering (ICKSE), Rome, Italy, 2009.

[15] G. Antoniou, F and Van Harmelen, "Web Ontology Language: Owl", In Handbook On Ontologies In Information Systems, 2003, pp 67—92.

Open Science Index, Computer and Information Engineering Vol:4, No:1, 2010 publications.waset.org/10085.pdf