World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:4, 2009

An Owl Ontology for Commonkads Template Knowledge Models

B. A. Gobin, and R. K. Subramanian

***Abstract*—**This paper gives an overview of how an OWL ontology has been created to represent template knowledge models defined in CML that are provided by CommonKADS. CommonKADS is a mature knowledge engineering methodology which proposes the use of template knowledge model for knowledge modelling. The aim of developing this ontology is to present the template knowledge model in a knowledge representation language that can be easily understood and shared in the knowledge engineering community. Hence OWL is used as it has become a standard for ontology and also it already has user friendly tools for viewing and editing.

***Keywords*—**Ontology, OWL, Template Knowledge Models, CommonKADS

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]. Several methodologies and frameworks have been developed over years e.g. CommonKADS [3], Protégé[4], MIKE [5], and MOKA[6].

CommonKADS [3] is one of the most mature second generation knowledge engineering methodologies. KBS development is based on the construction of a number of separate models that capture the desired features of the system and its environment. It has knowledge modelling as one of its main activity. The knowledge modelling activity consists of a selection and refinement of available model templates. CML, a frame-based language [7], is used for building the knowledge models. The template knowledge models also are defined using CML also.

One of the main criticisms associated with existing knowledge engineering methodology is its steep learning curve due to complexities associated with the methodology and the language used for knowledge representation. In an attempt to help in decrease complexity associated with knowledge representation language we propose the use of OWL[8] instead of CML for the template knowledge model. OWL has become the standard language for ontologies and is understood by many in the knowledge engineering community. Ontologies are used by all methodologies for representing the domain knowledge. Some methodologies and development environment e.g. Protégé and IBROW[9] use ontology to represent the reasoning component from a generic perspective which can then be used each time a knowledge model needs to be developed. Protégé e.g. has a library of PSM developed in UPML[10].

CommonKADS used ontologies only for the representation of the domain knowledge. We create an ontology that will contain template models proposed by CommonKADS so that can be easily shared and used and this work explains how the ontology was created. It first gives an overview of knowledge modelling in CommonKADS, followed by an overview of OWL. The ontology is then explained.

II. Knowledge Modelling in CommonKADS

A. Knowledge Model

In CommonKADS the knowledge model has three parts: *domain knowledge, task knowledge and inference knowledge*.

- *Domain Knowledge*

The domain knowledge specifies the domain specific knowledge and the information types that are needed in the application. It is basically a description of the knowledge that will be found in the system. The description can be categorized into two major groups: 1) domain schemas and 2) the knowledge base. The domain schema is a schematic description of the domain-specific knowledge and information through a number of type definitions. From a software engineering point of view the domain schema resembles the data model. The domain schema uses a set of modelling construct for domain knowledge specifications. There are three main modelling constructs are: CONCEPTS, RELATION and RULE TYPE. The knowledge base is the instantiation of the domain schema.

- *Inference Knowledge*

The inference knowledge describes how domain knowledge can be used to carry out reasoning process. The inference

B. A. Gobin (phone: 230-7494911 e-mail: baby_gobin@hotmail.com) 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.

knowledge is the inferences which describe the lowest level of functional decomposition. It consists of the knowledge roles needed for the inference and a small specification about the inference. The knowledge roles describe the input and the output. Another component of the inference knowledge are the transfer functions which describe the function that

- *Task Knowledge*

The task knowledge describes the goals that need to be achieved by the system and the strategies that will be used to achieve them. It consists of the task which defines the complex reasoning function and the task methods which describe how the task is realised through the decomposition into other subfunctions e.g. subtasks or inference or a transfer function. It also defines a control structure which describes in what order the subfunctions need to be carried out.

B. Template Knowledge Model

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 Fig. 1. Knowledge engineer can use these templates to build a system with respect to the task that need to be accomplished instead of starting everything from scratch. The advantages of reuse are as follows:

- It prevents from "re-inventing the wheel"
- It is cost/time efficient
- It decreases complexity
- It provides for quality-assurance

Hence a catalogue of task templates is provided for the above tasks. The task templates consist of the task definition and the task methods. They are reusable combination of model elements that have an inference structure, a typical control structure and a typical domain schema from task point-of-view.

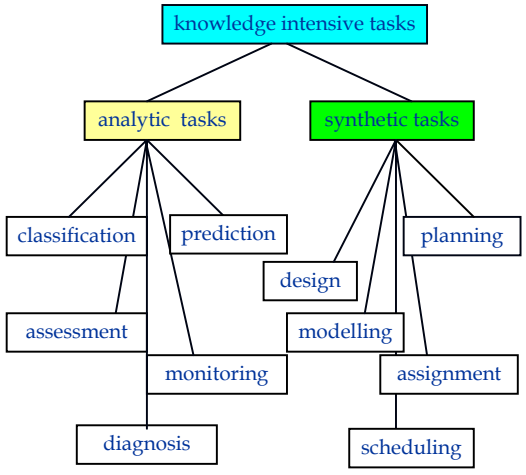

Fig. 1 Hierarchy of knowledge-intensive task types on the type of problem being solved

III. An Ontology for Template Knowledge Models

Ontology is a term borrowed from philosophy where ontology means a doctrine about existence in which general foundations, principles of existence, its structure and laws are studied. Gruber [11] defines 'ontology' as 'a formal, explicit specification of a shared conceptualization', and definitions in Gruberian spirit have been and still are accepted by most ontological engineers. This definition is based on the idea of conceptualization i.e. a simplified version of the real world that we want to represent. They provide a shared and common understanding of a domain that can be communicated across people and application systems. 'Conceptualisation' refers to the understanding of concepts and relationships that may exists or do exist between them. A representation of a shared knowledge in a specific domain that has been commonly agreed to refers to the 'specification' of conceptualisation [9]. An ontology should be: 1) representing knowledge specific to a domain, 2) shared, 3) used.

Our aim is thus to build an ontology of the template knowledge models so that it can be shared throughout the knowledge engineering community. However instead of using CML or UPML for knowledge representation we use OWL. OWL is a Semantic Web [13] Language for the following reasons:

1. *It provides more features than frame based knowledge representation languages.* OWL has more expressive power as compared to frame based languages. It provides for a series of OWL primitives and allows constraints checking. Also rules can be represented in an OWL document using SWRL.
2. *It is the standard language for knowledge representation.* OWL has been adopted by many as the language for ontologies. Hence is understood by many knowledge engineers and domain expert.
3. *User-friendly tools are available for the creation and manipulation of the OWL documents.* Protégé 2000[14] has proved to be a very mature and easy to use ontology editor. This tool will be used in our framework to cater for the manual changes that need to be made to the generate knowledge models. The OWL classes, instances and properties can be viewed and modified so as the rules using the SWRL tab.
4. *APIs are available for the manipulations of the OWL documents.* The Jena API can be used to extract or created classes and properties. Hence once the ontology of the template knowledge models have been created, the template required for a specific task can be extracted for generation of a knowledge model.

Due to these advantages we make use of Semantic Web Technologies for representing the knowledge model. We also believe that defining the ontology in this format will enable us to develop an easy mechanism for the automatic generation of the knowledge model since an OWL file is basically an RDF document. They can be manipulated using Jena API. Thus the template knowledge model ontology, domain ontologies and the application knowledge model are all defined in OWL. OWL Web Ontology Language is a language for defining Web ontologies. OWL is mainly based on OIL and DAML+OIL[15], therefore the main features of OWL are

very similar to the languages introduced above. OWL document consists of three main components:

- Sequence of axioms and facts plus reference to other ontologies
- Axioms used to associate class and property IDs with either partial or complete specifications of their characteristics and to give other logical information about classes and properties
- Fact which state information about particular individuals in the form of a class that the individual belongs to plus properties and values

It is these components that are used to create an ontology. In the next section we explain the different classes, properties and instances that we created to represent the template knowledge models defined by CommonKADS.

IV. Representing the Template Knowledge in Owl

A. Representing task knowledge

Each knowledge intensive task as per CommonKADS catalogue is represented as subclass of the main class task as shown in Fig. 2. The subclasses of the main class "***task***" are:

assessment ⊆ ***task*** , *diagnosis* ⊆ ***task*** , *classification* ⊆ ***task***, *monitoring* ⊆ ***task***, *prediction* ⊆ ***task***, *design* ⊆ ***task***, *modelling* ⊆ ***task***, *planning* ⊆ ***task***, *scheduling* ⊆ ***task***, *assignment* ⊆ ***task***

We chose to create it as a subclass rather than an instance as it becomes easier to extract related classes and properties with respect to this task from the template knowledge model ontology.Subtasks of the knowledge intensive task are then instances of the subclass created. E.g. for the knowledge intensive task "*assessment*" a subclass "*assessment*" is created which has as instance "*abstract_case*" and "*match_case*" which are the subtask of the task "*assessment*".

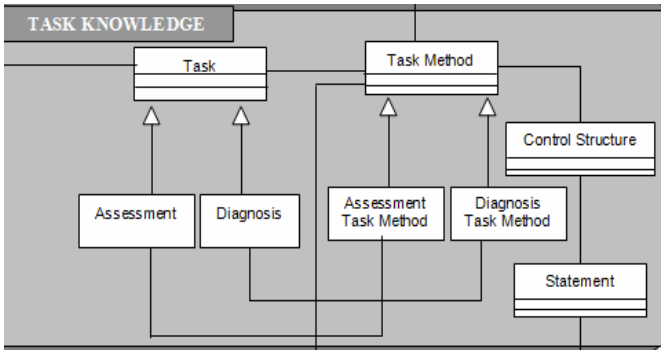

Fig. 2 Task Knowledge represented in UML

The same applies for all other components of the knowledge model except for inferences e.g. to represent task methods we have a class "***task_method***" which has a subclass "*assessment_method*" to represent the task method for the task "*assessment*" and instances "*abstact_method*" and "*match_method*". The different subclasses of the main class "***task_method***" are as follows:

assessment_method ⊆ ***task_method*** , *diagnosis_method* ⊆ ***task_method*** , *classification_method* ⊆ ***task_method*** , *monitoring_method* ⊆ ***task_method*** , *prediction_method* ⊆ ***task_method*** , *design* ⊆ ***task_method*** , *modelling_method* ⊆ ***task_method*** , *planning_method* ⊆ ***task_method*** , *scheduling_method* ⊆ ***task_method*** , *assignment_method* ⊆ ***task_method***

Subclasses of the main classes "***control_structure***" and "***statement***" are as follows:

assessment_cs ⊆ ***control_structure*** , *diagnosis_cs* ⊆ ***control_structure*** , *classification_cs* ⊆ ***control_structure***, *monitoring_cs* ⊆ ***control_structure***, *prediction_cs* ⊆ ***control_structure***, *design_cs* ⊆ ***control_structure***, *modelling_cs* ⊆ ***control_structure***, *planning_cs* ⊆ ***control_structure***, *scheduling_cs* ⊆ ***control_structure***, *assignment_cs* ⊆ ***control_structure***

assessment_statement ⊆ ***statement*** , *diagnosis_statement* ⊆ ***statement***, *classification_statement* ⊆***statement***, *monitoring_statement* ⊆ ***statement***, *prediction_statement* ⊆ ***statement***, *design_statement* ⊆ ***statement***, *modelling_statement* ⊆***statement***, *planning_statement* ⊆ ***statement***, *scheduling_statement* ⊆ ***statement***, *assignment_statement* ⊆ ***statement***

Fig. 3 gives a snapshot of the classes and the instances of the class "*assessment*" which is the subclass of the class "*task*", as seen in Protégé 2000. The two instances are "*abstract_case*" and "*match_case*". Table I gives the properties defined for the classes and some instances created for each class used to represent the task knowledge.

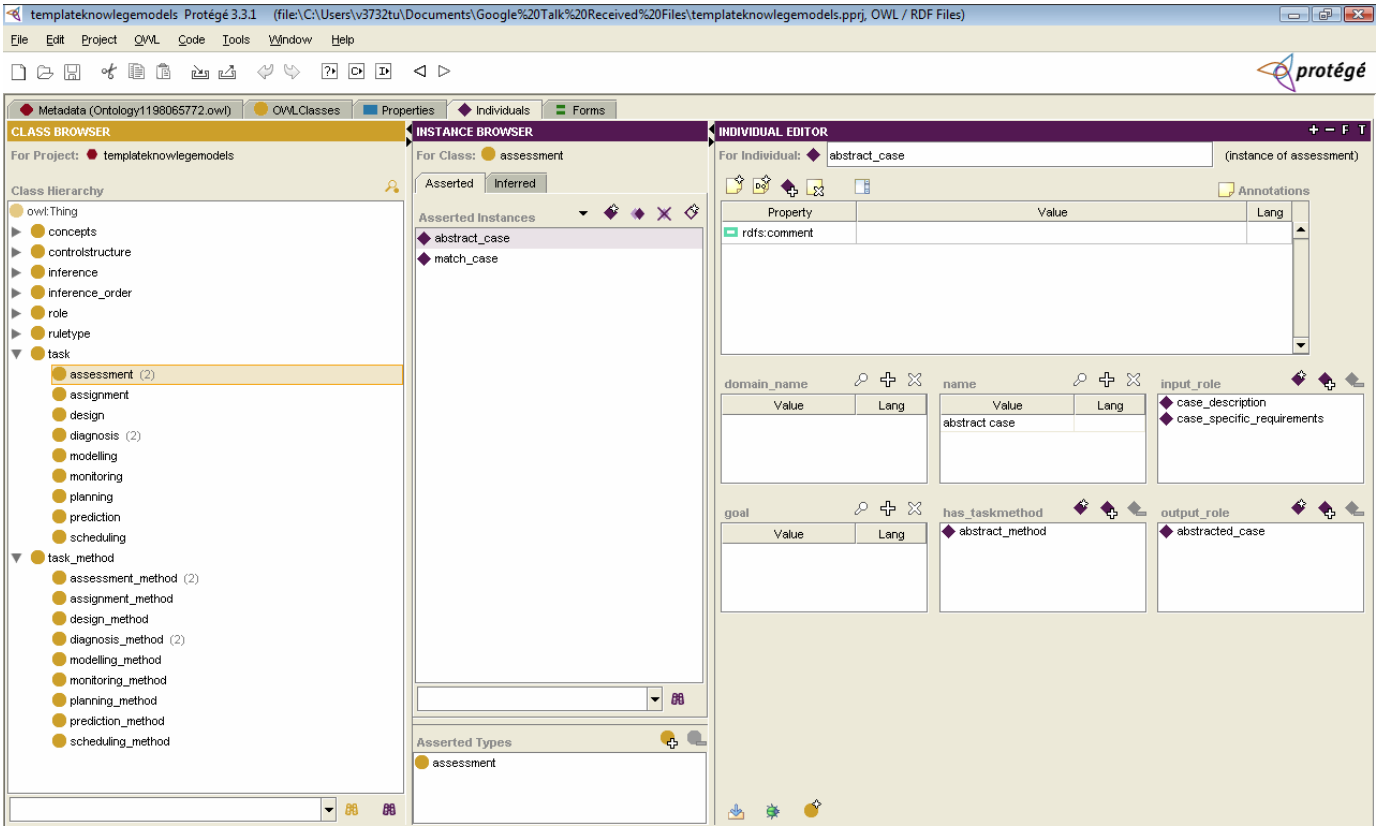

Fig. 3 Template knowledge model in Protégé 2000

TABLE I
PROPERTIES AND INSTANCES OF CLASSES REPRESENTING THE TASK KNOWLEDGE

Class	Properties	Instances
task	goal (DP)[a] has_input_role(OP) [a] has_output_role(OP) has_task_method(OP)	e.g. instances of class "assessment" : • abstract_case • match_case
task_method	has_inference(OP) has_control_structure (OP) has_intermediate_role (OP)	e.g. instances of class "assessment_method" : • abstract_case_method • match_case_method
control_structure	has_statement(OP)	e.g. instances of class "assessment_cs" : • abstract_cs • match_cs
statement	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)	e.g. instances of class "abstract" abstracted_case • assessment_statement1

[a] DP= Datatype Property OP = Object Property

B. *Representing the Inference Knowledge*

The class "*inference*" has as subclasses the different inferences that are found in the catalogue provided by CommonKADS (Fig. 4). The subclasses are defined based on general inferences and not on the task 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 template knowledge model, we will have class *"select"* which is a subclass of the class *"inference"* which has two instances one instance is for the task *"assessment"* called *"assessment_select"* and the second for the task *"diagnosis"* called *"diagnosis_select"*. The same applies for other inferences which are called in different task methods e.g. inference *"specify"*. Table II contains the properties for each class as well as some examples of instances which have been created.

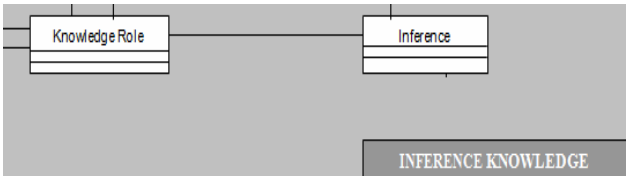

Fig. 4 Inference Knowledge Represented in UML

Subclasses for the classes "*inference*", "*role*" and "*statement*" are as follows:

abstract ⊆ ***inference*** , *cover* ⊆***inference***, *evaluate* ⊆***inference***, *generate* ⊆***inference***, *match* ⊆***inference***, *design* ⊆***inference***, *select* ⊆***inference***, *specify* ⊆***inference***, *verify* ⊆***inference***

TABLE II:
PROPERTIES AND INSTANCES OF CLASSES REPRESENTING THE INFERENCE KNOWLEDGE

Class	Properties	Instances
Inference	has_input_role(OP) [a] has_output_role(OP) has_static_role(OP) specifications(DP) [a]	e.g. instances of class "abstract" : assessment_abstract
Role	type (OP) domain_mapping(OP)	e.g. of instances of the class • casedescription • decision

[a] DP = Datatype Property OP = Object Property

assessment_role ⊆ ***role*** , *diagnosis_role*⊆ ***role***, *classification_role* ⊆***role***, *monitoring_role* ⊆***role***, *prediction_role* ⊆ ***role***, *design_role* ⊆ ***role***, *modelling_role* ⊆***role***, *planning_role* ⊆ ***role*** , *scheduling_role* ⊆ ***role***, *assignment_role* ⊆ ***role***

C. Representing Domain Knowledge

Fig. 5 shows the different classes used to represent the domain knowledge. Subclasses for the classes "*concepts*", "*rule_type*" and "*relations*" are as follows:

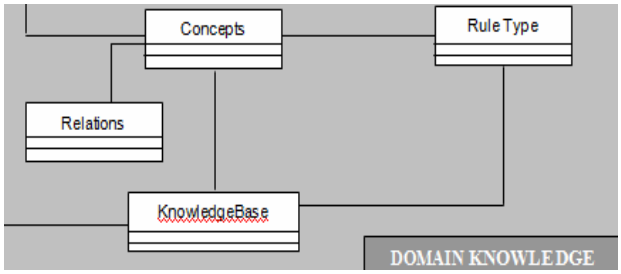

Fig. 5 Domain Knowledge Represented in UML

assessment_concepts ⊆ ***concepts*** , *diagnosis_ concepts* ⊆ ***concepts***, *classification_ concepts* ⊆ ***concepts***, *monitoring_ concepts* ⊆ ***concepts***, *prediction_ concepts* ⊆ ***concepts***, *design_ concepts* ⊆ ***concepts***, *modelling_ concepts* ⊆ ***concepts***, *planning_ concepts* ⊆ ***concepts***, *scheduling_ concepts* ⊆ ***concepts***, *assignment_ concepts* ⊆ ***concepts***

assessment_rule_type ⊆ ***rule_type*** , *diagnosis_rule_type*⊆ ***rule_type***, *classification_rule_type* ⊆ ***rule_type***, *monitoring_rule_type* ⊆ ***rule_type***, *prediction_rule_type* ⊆ ***rule_type***, *design_rule_type* ⊆ ***rule_type***, *modelling_rule_type* ⊆ ***rule_type***, *planning_rule_type* ⊆ ***rule_type***, *scheduling_rule_type* ⊆ ***rule_type***, *assignment_rule_type* ⊆ ***rule_type***

assessment_relations ⊆ ***relations*** , *diagnosis_relations*⊆ ***relations*** , *classification_relations* ⊆***relations***, *monitoring_relations* ⊆ ***relations***, *prediction_relations* ⊆ ***relations*** , *design_relations* ⊆ ***relations*** , *modelling_relations* ⊆ ***relations*** , *planning_relations* ⊆ ***relations*** , *scheduling_relations* ⊆ ***relations*** , *assignment_relations* ⊆ ***relations***

Table III contains the properties for each class as well as some examples of instances which have been created.

TABLE III
PROPERTIES AND INSTANCES OF CLASSES REPRESENTING THE DOMAIN KNOWLEDGE

Class	Properties	Instance
concepts	specification (DP) [a]	instances of class "assessment_concepts" : • case_criterion • case_decision
rule_type	specification (DP) has_concept1 (OP) [a] has_concept2(OP)	e.g. instances of class "assessment_relation" : application
relations	specification (DP)	e.g. instances of class "": • abstraction_rules • decision_rules • requirement_rules

[a] DP = Datatype Property OP = Object Property

V. Future Works

The main difficulty faced during the creation of the ontology was the representation of the control structure. We introduced a new concept called statement to represent each statement in the control structure. It contains all the constructs defined by CommonKADS that can be found in a control structure. They are represented as the properties of the concept. Each statement is of the control structure is represented as an instance of the class concept. Up to now we have seen this as the best solution. As future work we shall continue our investigation on how to best represent the control structure. We have not opted for OWL-S because we want all our templates to be represented in only one OWL document. The main reason behind this is also because we want to use this ontology for the semi automatic generation of knowledge models. Research in the automatic generation of knowledge model can help to bring in solutions to issues related to the knowledge modelling process. Therefore in view to 1) decrease learning and development overheads, 2) standardise knowledge modelling process, 3) implement reuse, 4) link knowledge model phase to implementation phase, we have conceptualised a framework for semi-automatic of knowledge model. Though we would have like to provide for full automation, we are of the opinion that full automation is not feasible since interactions with knowledge engineers/domain experts are necessary. We use OWL and SWRL in our framework to build our knowledge model so that it can be represented in an easy format which can be understood not only by knowledge engineers having expertise in AI but also by domain experts also. This in turn can help decrease the communication gap between these two experts, which is one of the reasons for knowledge acquisition bottlenecks. The components of the knowledge model are generated from the the ontology for template knowledge models proposed that we have created. The generic application knowledge model is then adapted to the domain of application based on the domain ontologies which are in OWL and rules that are input by the knowledge engineer. The relevant concepts and properties are extracted based on the knowledge about the domain schema in the generic application knowledge model. As for the rules, they are input as "*if-then*" statements, which are automatically converted into SWRL and added to the adapted knowledge model. Also the framework will allow the mapping of the knowledge models on Java classes, which will act as a bridge between the modelling stage and implementation stage, hence providing for smooth transition between these two stages.

References

[1] P. Speel, A.T. Schreiber, W. Van Joolingen, J. van Heijstg and G. Beijer, "Conceptual 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, vol 25. 1998, pp. 161-197.

[3] A. Th Schreiber, J. Akkermans, A. Anjewierden, R de Hoog, N. Shadbolt, W. van de Velde , 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 S. W. Tu, "The evolution of protege: an environment for knowledge-based systems development", International Journal of Human Computer, 2003, 58(1), pp 89-123

[5] J. Angele, D. Fensel, D. Landes and R. Studer, "Developing knowledge based systems with MIKE", Journal of Automated Software Engineering, 1998, 5(4), 389-418.

[6] M. Callot 1999, "Methodology and tools oriented to knowledge engineering applications, MOKA public report No.2" [Online]. Available : http://www.kbe.conventry.ac.uk/MOKA

[7] H. Knublauch, "An agile development methodology for knowledge-based systems including a java framework for knowledge modelling and appropriate tool support", Ph.D . Dissertation, University of Ulm,2002.

[8] G. Antoniou, F. van Harmelen, "Web ontology language: OWL", in: Handbook on Ontologies in Information Systems, 2003, pg 67--92

[9] D. Fensel, E. Motta, F. van Harmelen, V. R Benjamins., M Crubezy., S. Decker, M Gaspari., R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R Studer. and B. Wielinga, "The unified problem-solving method development language UPML", Knowledge and Information Systems", 1999,5(1), 83-131.

[10] Fensel D., Motta E., Benjamins V., Decker S.,Gaspari M., Groenboom R., Grosso W. , F. van Harmelen, M. Musen , E. Plaza, G. Schreiber, R. Studer, A. Ten, 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.

[11] T. R.Gruber, "A translation approach to portable ontologies", Knowledge Acquisition, vol 5 pp 199-220,1993.

[12] T. Dillon, E.Chang, M. Hadzic, Wongthongtham P. , "Differentiating conceptual modelling from data modelling, knowledge modelling and ontology modelling and a notation for ontology modelling", in 2008 Proc of the fifth Asia-Pacific conference on conceptual modelling - Volume 79.

[13] S. Decker, F.van Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein, and S. Melnik, "The semantic web - on the respective roles of XML and RDF", IEEE Internet Computing, vol 4,2000.

[14] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, A practical guide to building owl ontologies using the prot´eg´e-owl plugin and co-ode tools edition 1.0.,2004.

[15] Sinuhe A., Ying D., Ruben L., Stollberg M. AND Fensel D., 2004. "Semantic web languages. strengths and weakness." Presented at the Int. Conf. in Applied computing (IADIS04), Lisbon Portugal,23-26 March2004.