European Journal of Scientific Research
ISSN 1450-216X Vol.62 No.3 (2011), pp. 311-320
© EuroJournals Publishing, Inc. 2011
http://www.eurojournals.com/ejsr.htm

Distributed Knowledge Acquisition System for Software Design Problems

Nedhal A. Al-Saiyd
Comp. Sc. Department, Faculty of Information Technology
Applied Science University, Amman-Jordan
E-mail: nedhal_alsaiyd@asu.edu.jo

Adel H. Mohammad
CIS Department, Faculty of Information Technology
Applied Science University, Amman-Jordan
E-mail: a_hamdan@asu.edu.jo

Intisar A. Al-Sayed
Software Engineering Department
Faculty of Computer Science Information Technology
Al-Isra University Amman-Jordan
E-mail: Intisar@ipu.edu.jo

Muna F. Al-Sammarai
MIS Department, Faculty of Economics and Administrative Sciences
Al-Zaytoonah University, Amman, Jordan
E-mail: Faik_muna@yahoo.com

Abstract

Knowledge plays an important role in designing intelligent systems especially in expert systems and knowledge-based systems in different application domains. Its efficiency and effectiveness depends on the Knowledge Acquisition (KA) phase. Acquiring experience knowledge and transferring it into a knowledge-based system is complex and involves a range of diverse activities. There are two key problems related to KA-the identification of domain knowledge expertise and the heuristics that are useful in problem solving; and the identification of people who can provide this knowledge. In this paper, an approach for Web-based knowledge acquisition system is presented for acquiring expertise from software designers. This knowledge is used in software design decision making and knowledge reasoning. The process and decision laddering techniques are involved with the concept mapping to create and evaluate the knowledge as efficient and effective as possible.

Keywords: Distributed Knowledge Acquisition, Intelligent Systems, Knowledge Models, Conceptual Graph, Software Design.

1. Introduction

In artificial intelligence, the cognitive scientists try to find methods that map expert knowledge from the real world into symbols and operations that computers can process. They found that there is a strong relationship between the structure of an expert's knowledge and the processes used for problem-solving [1], [2]. In general, experts possess highly organized knowledge structures which can be rapidly and efficiently accessed during problem-solving. In addition, a problem's representation affects the knowledge structures and problem-solving processes [3].

Knowledge acquisition (KA) is the process of obtaining knowledge from a domain expert as knowledge source and is used to solve artificial intelligent problems; when and/or where the experts are not available. Acquiring adequate and high-quality knowledge is the most costly, time-consuming and difficult part of knowledge engineering. Experts have large amounts of knowledge and a lot of tacit knowledge that is hard to describe, so KA requires intensive efforts [4], [5]. Knowledge acquisition includes the elicitation, collection, analysis, modeling and validation of knowledge for knowledge engineering and knowledge management projects [6].

Software designers develop the design by conducting various cognitive and physical activities. Designers often fail to express what's in their mind. Generally at the cognitive level, the design process consists of a series of design activities. They are described as analysis-synthesis-evaluation. These activities take place in an iterative manner. Design is an iterative process where schemes are recognized, explored, revised and enhanced until a solution is identified. Design issues are general categories for sorting design information into manageable chunks to support efficient decision making. Each design issue will not evoke only one specific solution. Multiple design solutions are to be generated and one of them will eventually be chosen for the skeleton of final solution.

Recently, the software architecture research community has faced the need to consider architecture design decisions (ADD) as first class entities that should be stored and documented, embodied with the standard architecture documentation. This shift is mentioned by Bosch [7], who states that: '*The key difference from traditional approaches is that we do not view software architecture as a set of components and connectors, but rather as the composition of a set of ADD*'. Bass *et al.* [8] justify this as: '*software architecture is a coherent, justified collection of system's earliest set of design decisions. These decisions will affect much of what the system will become*'.

In practice the standard documentation of software architectural design lacks explicit description of the decisions made and their underlying rationale, which often leads to knowledge loss. This fact strongly affects the understanding of the changes performed in the design and the maintenance activities when we need to spend additional effort and time to re-apply the decisions made. Hence, codifying this architectural knowledge is a challenging task that requires adequate tool support. In 2010 Capilla and etal test the capabilities of Architecture Design Decision Support System (ADDSS), a web-based tool for supporting the creation, maintenance, use, and documentation of architectural design decisions (ADD) with their architectures.[9].

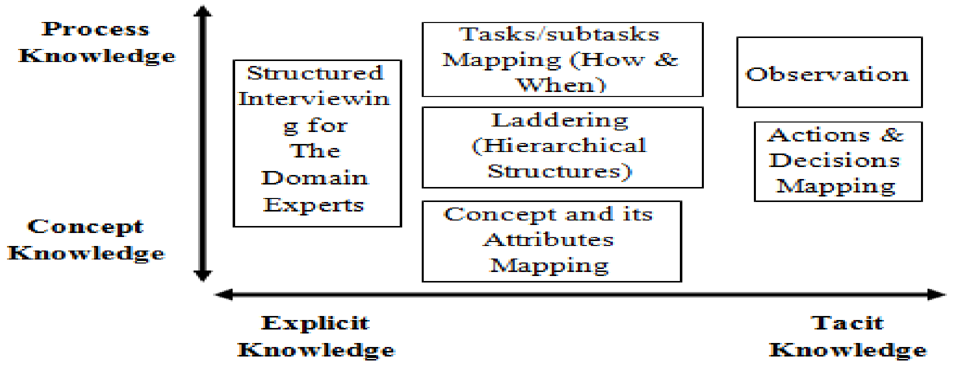
In this paper, we need to identify the most important knowledge that is valuable in software design phase as a knowledge domain. To overcome this, the knowledge engineer has to review the goals and purposes of the elicited knowledge. The question "why do we need this knowledge?" may lead to analyze and evaluate software design methodologies, techniques, and models. The elicitation process is then expected to meet the knowledge that supports these objectives and strategies.We attempt to use a hybrid method that integrates the knowledge acquisition method called Delphi method and a semi-automated method to support knowledge engineer to elicit and evaluate knowledge from different experts. In the Delphi method [10], the selection of domain experts is made according to the knowledge field. They are given set of questions and then categorize their answers on the topic of problems. The contribution of each expert is made available to other experts and the results of the initial set of questions are used to derive another set of questions. In addition to the hybrid KA method, the hierarchy-generation techniques (laddering) is used to create, review and modify hierarchical knowledge. Laddering technique is used to build taxonomies, concepts, process roadmap and a hierarchical goal structures. The use of these is particularly important in capturing the "what", "how", "when", "who" and "why" of tasks and events.

The rest of the paper is organized as follows: Section 2 briefly introduces the problems with acquiring expertise and skills of software designs and describes the considerations that had an impact on the choice of the overall approach to distributed KA from software designers. Section 3 provides distributed knowledge acquisition approach and a description of the approach that is used in knowledge acquiring, weighting and conceptual modeling. Section 4 contains the conclusions and proposes lines of future work.

2. Knowledge Acquisition Techniques

The knowledge engineer focuses on the essential knowledge and uses suitable techniques to capture various types of knowledge, such as goals, decisions, relationships and attributes from different experts [11]. Figure 1 shows a the main KA techniques, it shows the main tacit and explicit knowledge types, which are mainly intended at knowledge elicitation. The vertical axis on the figure represents the dimension from concept (object) knowledge to process knowledge, and the horizontal axis represents the dimension from explicit knowledge to tacit knowledge [12]. One of these techniques that we use is a hierarchy-generation technique (laddering). It is used to builds hierarchical structures for taxonomies and decisions. Various forms of ladder can be used:

Figure 1: Main KA Techniques


Process Knowledge
Concept Knowledge
Structured Interviewing for The Domain Experts
Tasks/subtasks Mapping (How & When)
Laddering (Hierarchical Structures)
Concept and its Attributes Mapping
Observation
Actions & Decisions Mapping
Explicit Knowledge
Tacit Knowledge

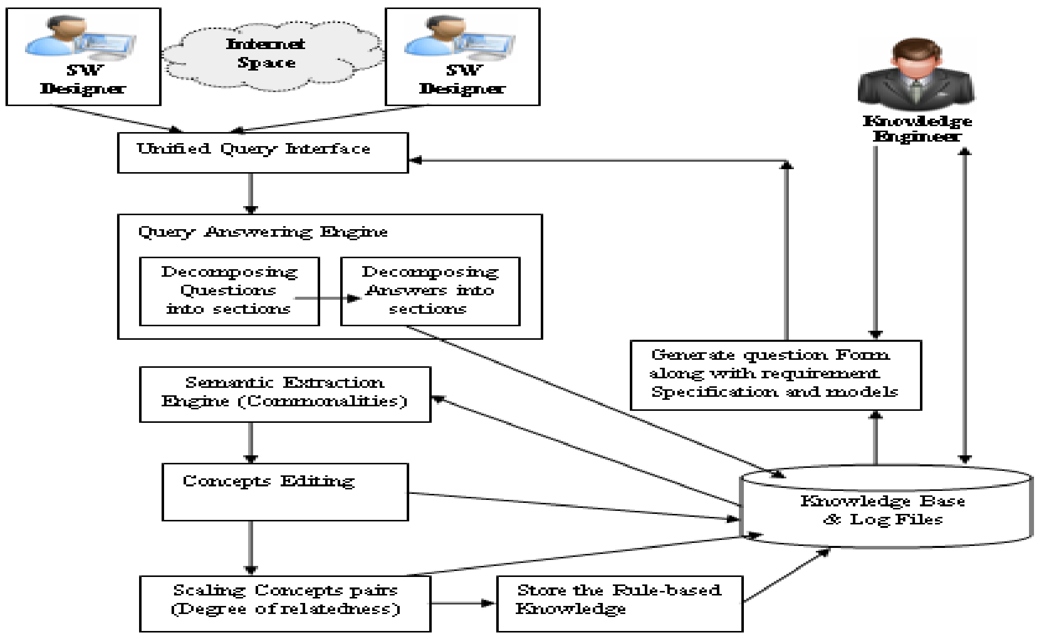
- Concept ladder: shows the way an expert classifies knowledge objects (concepts) and instances into classes and sub-classes to which they belong. Hence, it is called taxonomy. It is an important key to understand the way the domain knowledge is conceptualized.
- Composition ladder: shows the complex objects that is composed from its constituent parts (i.e. piece of knowledge) in addition to their connected relations 'is-composed-of'.
- Attribute ladder: shows the attributes and attribute values of the concepts. By reviewing and appending such a ladder, the knowledge engineer can validate and help elicit knowledge of the properties of concepts.
- Process ladder: shows processes and sub-processes, all the relations are of the 'is-part-of' sort. A process not only shows 'what experts do', but also 'what people produce' [13].
- Network diagrams: shows connected nodes. Concept maps are concentrated on labeled relations that connected concept nodes and are similar to semantic nets. Concepts connected with other words to form a meaningful statement. Process maps show the inputs, outputs, resources, roles, and decisions associated with each process or task. It shows how and when processes, tasks, and activities are performed. It shows how input is transformed into output process map, the primary processes and the parallel processes.

The validation of the knowledge represented in a ladder with another expert is often very quick and efficient.

3. Framework of Distributed Ka for Software Architecture Design

In this paper, we present a new distributed and incremental methodology to acquire knowledge from the Web to build semi-automatically knowledge base and ontology of concepts. This is done (through the composition of automatically obtained taxonomies) for a given "software architectural design" domain according to the designer's interests, using a flexible, dynamic and slightly supervised agent-based components. This can be considered as an improvement over the classical way of elicits knowledge expertise. The main framework components are shown in figure 2. We have to isolate the broader characteristics of KA tasks concerned in knowledge engineering for those involved in eliciting software design knowledge from software designers, in order to gain in-depth closer decision to the solutions of specific indicated development problems.

Figure 2: The framework of distributed knowledge acquisition methodology from software design experts


SW Designer
Internet Space
SW Designer
Knowledge Engineer
Unified Query Interface
Query Answering Engine
Decomposing Questions into sections
Decomposing Answers into sections
Generate question Form along with requirement Specification and models
Semantic Extraction Engine (Commonalities)
Concepts Editing
Knowledge Base & Log Files
Scaling Concepts pairs (Degree of relatedness)
Store the Rule-based Knowledge

We proposed a methodology that consists of a set of activities that covers distributed and incremental KA that help us to extract the expertise of software designers. These interrelated activities are illustrated in figure 3. The acquired knowledge structure are retrieved and categorized according to the specific topic covered.

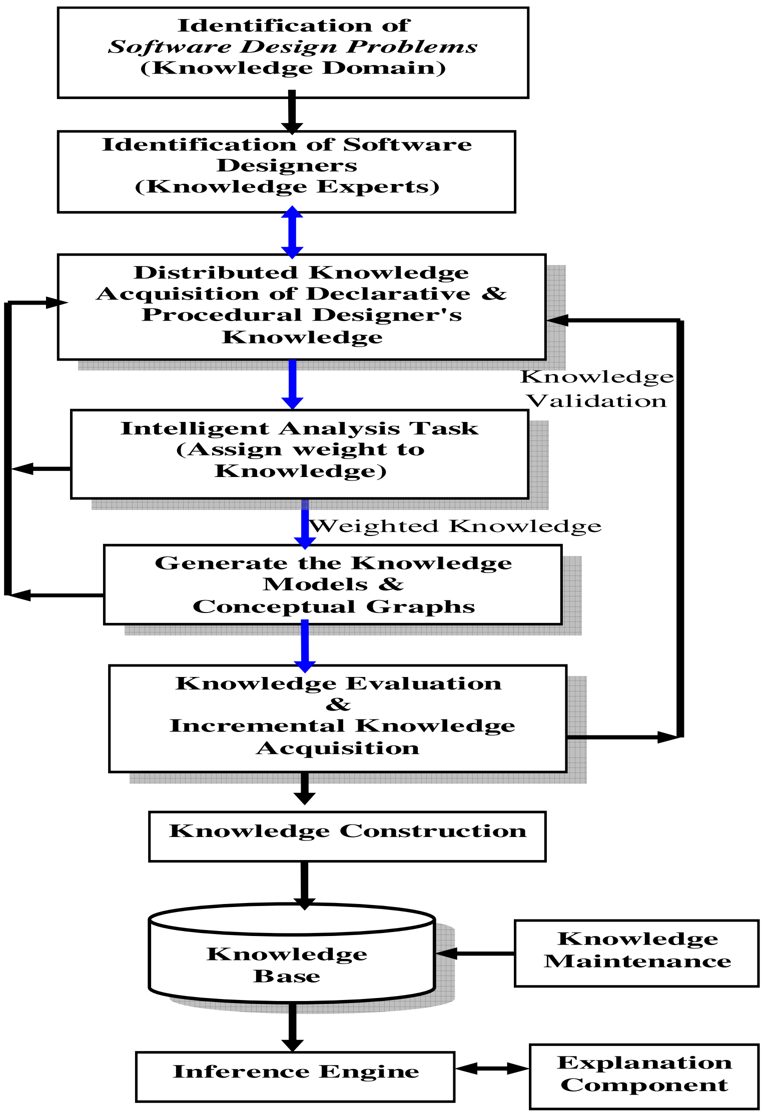
3.1. Knowledge Acquisition Requirements Phase

The knowledge elicitation process requires the identification of:

1) The Problem Domain: The knowledge engineer identifies the "software design" as the knowledge domain, which is the process of problem-solving and planning for a software solution. The "software design" domain can be divided into stable sub-domains.
   The knowledge designer plans to accumulate specialized knowledge of each sub-domain regarding the software design task because each expert (software designer) has specific tasks within software design team and learn more from experience when working on his own tasks.
2) The Knowledge Engineer: reads many sources of information and discuss many issues with different persons to (a) determine what knowledge is to be acquired, (b) decide what is the purpose the knowledge, (c) find out some understanding of key terminology. Once knowledge

engineer feels familiar with the problem domain, he prepares for computer-based interviewing with one or more experts to acquire the knowledge.

Figure 3: Main activities of Knowledge Engineering for Distributed Incremental Knowledge Acquisition


Identification of *Software Design Problems* (Knowledge Domain)
Identification of Software Designers (Knowledge Experts)
Distributed Knowledge Acquisition of Declarative & Procedural Designer's Knowledge
Knowledge Validation
Intelligent Analysis Task (Assign weight to Knowledge)
Weighted Knowledge
Generate the Knowledge Models & Conceptual Graphs
Knowledge Evaluation & Incremental Knowledge Acquisition
Knowledge Construction
Knowledge Base
Knowledge Maintenance
Inference Engine
Explanation Component

3) The Knowledge Experts: It is needed to determine the people who are experts in solving specific type software design problems and identify their strength relation to the software design domain through task performance over a long period of time. Experts view problems differently than novices do because of their higher ability to recognize design components, patterns and models, conclude relationships, ignore irrelevant information, recall similar design problems from past experience and retrieve probable solutions, when matched to a given problem's characteristics.
    - Each one has his own skills and expertise. This needs focusing on related information about current experts' roles and responsibilities. The different knowledge sources' types lead to different types of knowledge and different ways of acquiring knowledge.
    - Since software design is considered as problem-solving and is determined by the way the problem is conceptualized and represented, the knowledge engineer sends at the same time a problem definition of proposed application software along with its complete requirement specification. It is considered as the input to the software

design. This will help to ensure a consistent understanding of software problem environment and its meaning.

4) Software Design Task and subtasks: it is helpful for the knowledge engineer to identify *what* information is absolutely necessary for problem-solving. It required the expert to recognize patterns, features and concept commonalities and variabilities of software design process.
5) The Question Form: The question form is divided into sections. Each section is concerned with one task (knowledge component) of software design, and the relationships among these knowledge components. The sequence and the importance of the questions within each section are studied thoroughly. The knowledge engineer asks questions regarding different duties and operations of the expert that seem more subtle.
   We took into our consideration the questions that are concentrated on:
   - ***Know-how*****:** refers to the software designer expertise and the software design standards. It is important to extract the procedural knowledge, i.e. how to design.
   - ***Know-what*****:** refers to the consistent set of design basics and their meaning, classification, design specifications, design tasks, design attributes. It is useful to extract the declarative knowledge, i.e. knowledge of facts.
   - ***Know-why*****:** refers to why a particular effort and process is related to one activity, it is useful in justification.

The relationship among questions of one section includes a description of task and the domain elements to which it belongs in order to better comprehend the connection between software design concepts. Once the questions of all the sections are identified, the constraints are considered and the description of constraint needs to be incorporated in question's formulation. A constraint may be attributed to a single question, to multiple questions at once, or as a global constraint applicable across all sections and all questions (i.e. constraints on design phase).

3.2. Incremental Knowledge Acquisition Phase

Within this phase, the emphasis is placed on eliciting as much information about the domain as can be found. The experts will receive the question forms on their emails and they answer them when they are out of tense of their work and when they have adequate time to do it, instead of take an appointment that may not presents the appropriate circumstances to perform the elicitation task.

- Our distributed knowledge acquisition system has a *unified query interface* that is used to send the same question's forms to different software design experts who have different viewpoints. This may lead to various interpretations of the same domain and will help to capture as much information as possible. The experts also give a linear sequence of justifications when they are explaining how they reach a conclusion. The expert of software design will send back his/her answers to the knowledge engineer. The answers of each section in the question form should be analyzed in the same order to which they were elicited.
- The answers of each question are decomposed to find the relevant declarative, descriptive, and the justification knowledge, classifying them into log files, finding commonalities and differences, accuracy degree, and heuristics. The knowledge engineer will examine the elicited information to determine their relevance to questions of each subtask (i.e. a section within question form) of software design work. After knowledge is collected, it must be interpreted and analyzed. Firstly, a transcript of the knowledge acquisition session is produced. The transcript is indexed by the session date, expert ID, design topic, and design sub-task. It helps to facilitate the search for the source knowledge if the knowledge engineer needs to maintain the key pieces of knowledge.
- The transcript is used as a basis in reviewing and analyzing the key pieces of knowledge and their relationships. It requires the involvement of experts to refine and approve the clarified knowledge. The knowledge engineer assigns *weights* to the knowledge depending on their roles and performance in solving problems, rather than depending on what experts rely heavily on.

The ranking scale is between 0 and 4. '0' is assigned for irrelevant knowledge and '4' for the most significant. This is the conception of trust in the source of the knowledge that is an important component to any intelligent analysis task. Their results show that the incorporation of trust information improves the accuracy of assessments compared to similarity alone. It is considered as an important open issue in some of the methods for KA as it is mentioned in [14]. The inconsistent and uncertain knowledge that was stored in files from the first session; is resend later to the source of knowledge and to other experts to check them, clarify them, and to identify the reasoning of software design decision-making. The answers that are collected from one expert can be forwarded to another expert to review them and/or discussed them further. This will help in validating the collected knowledge. The irrelevant concepts or knowledge elements should be removed from the transcript. The apparent inconsistencies should be isolated along with any associated relationships and placed separately into a particular file with a description and expert ID for further inquiries. Constraints associated to the removed concept should be examined to determine whether it is still applied to other elements within the knowledge or need to be detached along with the related knowledge.

- After completing the second acquiring knowledge session, the knowledge engineer accumulates feedback knowledge from the experts and draws a tree-structure representation and sends them to the experts in the direction of appraisal. The experts add, delete, rename or re-classify nodes and/or relations as appropriate by having the expert's reasoning comments on each modified element. This will qualify the answers through evaluating the satisfaction criteria for the elicited knowledge. The expert can exchange knowledge among other expert members to explain their decisions by which members achieve specialized and tacit (procedural) knowledge that is associated with software design tasks.
- Finally, the knowledge engineer will collect the answers and organize them into knowledge representation models (KRM) to discuss, analyze and evaluate the sequence of answers and the correlation among them. KRM support experts and knowledge engineer in analyzing and emphasizes on the development of the conceptual model.

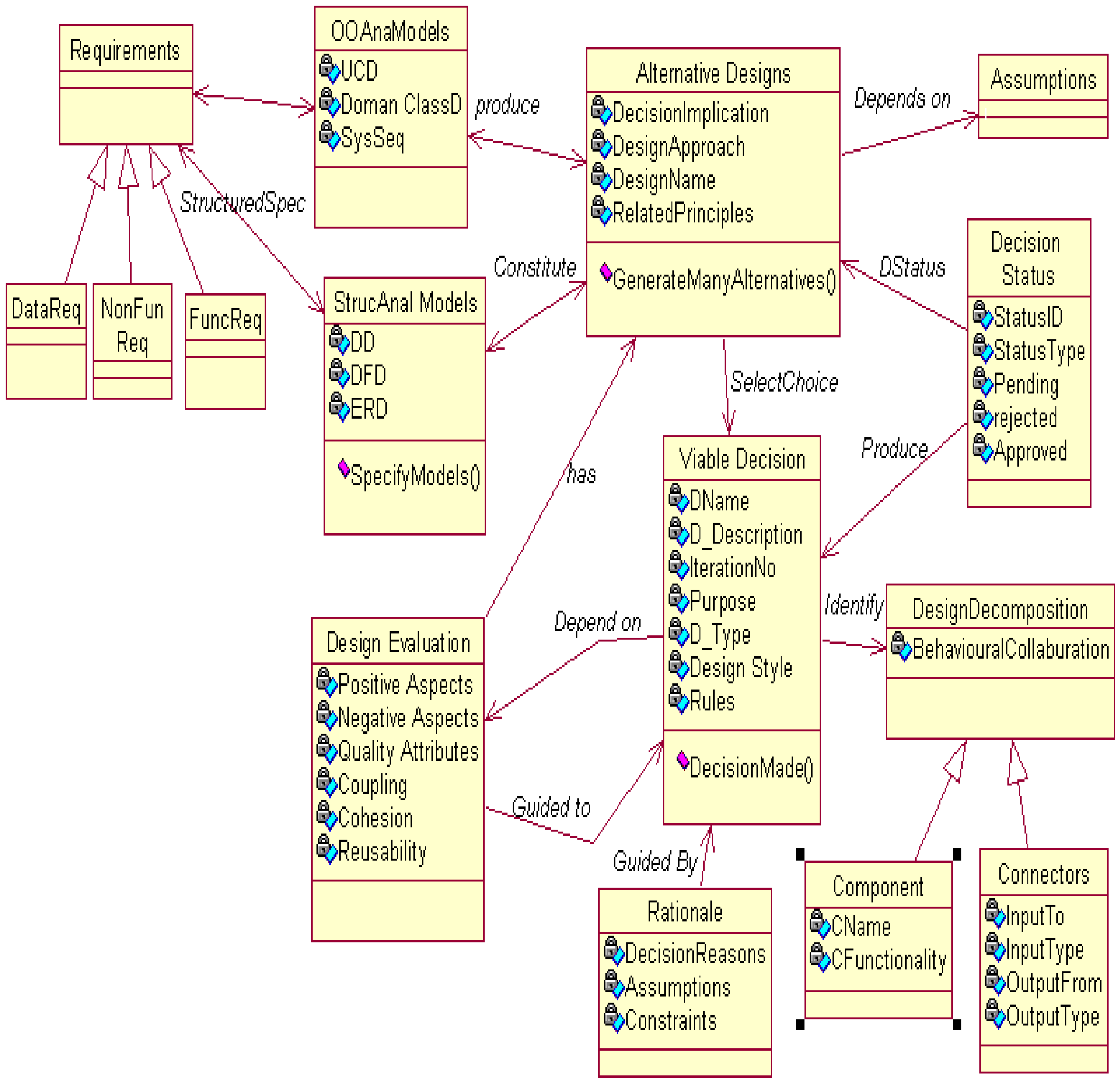
3.3. Generate Knowledge Representation Models

The graphical technique defines structure of knowledge is used to show the pieces of knowledge and the interrelationships among them. The knowledge engineer creates set of knowledge models to summarize understanding of acquired knowledge. The concept ladder, attribute ladder, process ladder, concept map, and process maps are produced to visualize them by the experts. They are easier for experts to view knowledge conceptually because transcript models are "String maps" illustrate either poor understanding of the material or an inadequate restructuring of the map. Using laddering also involve a set of predefined investigate questions, such as "Could you tell me some types and sub-types of Software Design?", "Could you tell me, how can you tell that the problem-solving approach is Object-Oriented Design?", "Why would you prefer using design patterns to Y module?" and "Why would you prefer Object-Oriented Design to Z-subsystem?".

The semantic mapping between the acquired knowledge and ontology that represents a domain expertise mainly in hierarchies of concepts and relationships will facilitate semantic reasoning. The conceptual model can be represented using the object-oriented approach. By conceptualization, we mean a stable and formal representation of concepts, relationships and objects; for the structural and behavioral knowledge. Clearly there is an interaction between these levels during the KA process. The conceptual model for concepts of software design is shown in figure 4. It shows the declarative and procedural knowledge.

The conceptual diagrams can also be used for evaluation. They should be revised and concepts should be positioned in ways that lead to clarify a final diagram. They usually go through two to three revisions and the request of the expert’s opinion is helpful to guide and limit when specific and non widely-used concepts may appear. The creation of meaningful diagrams as a form of knowledge elicitation nowadays has wide application in human factors studies [15], [16].

Figure 4: The conceptual graph of software design and rationale for design decision making



3.4. The Examination and Evaluation Phase

Resend the forms with inconsistencies to the source of knowledge and to other experts to resolve the incompleteness and/or inconsistencies.

When negotiating the different interpretations of inconsistent knowledge is resolved, a new knowledge is incrementally added to its related knowledge in a knowledge base. As time passes, the knowledge base is growing.

The experts verify the inconsistent/incomplete knowledge and sent the verification knowledge to the knowledge engineer who in turn adds new knowledge or replace some old knowledge from knowledge base. Similarly, if the experts may add further knowledge, therefore, the knowledge engineer can add it after knowledge representation to the knowledge base.

KA is a structured method so the new knowledge is added having one or more relations with existing knowledge. Knowledge growth is dependent on knowledge transfer methods and on the evaluation technique. The evaluation function **Evaluation (K, Refinement)** can be defined recursively as follows:

If 'Refinement' is the corner stone refinement passed to *K* and *K* is a knowledge base, which is of the form:

```
A ← L1,L2, ……, Lm then Evaluation(K, Refinement) = A.
and
If K = Correlation(Rule1,Rule2),
Let:
change1 = Evaluation(Rule1)
change2 = Evaluation(Rule2)
if change1 is 'ok' and change2 is 'not_ok' then
        Evaluation(K, Refinement) = No_Change,
elseif change1 is 'not_ok' and change2 is 'ok' then
        Evaluation(K, Refinement) =change1,
elseif Evaluation(K, Refinement) =change2
```

If change1 is 'ok' and change2 is 'not_ok' then Evaluation(K, Refinement) = No_Change, means that: information initially added does not need to be removed.

4. Conclusion

In this paper a new methodology for distributed knowledge elicitation, evaluation and management for *software architecture design process* was presented. This method integrates both the descriptive and perspective software design knowledge. We have developed methods for performing deductive and inductive reasoning. This methodology has several advantages over the existing ones, including ease of elicitation, analysis, and evaluation knowledge. The design process performed by the expert designers or guided by the system, novice designers will be greatly assisted to learn how to approach a certain class of design. The system would also be useful for learning design process.

5. Acknowledgment

The authors are grateful to the applied science private university, Amman-Jordan, for the partial financial support granted to cover the publication fee of this research article.

References

[1] Mehdi Sagheb-Tehrani, A Conceptual Model of Knowledge Elicitation, Proc CONISAR 2009, v2, pp. 1-7,EDSIG. Also available on: http://proc.conisar.org/2009/1542/CONISAR.2009.Sagheb-Tehrani.pdf

[2] Laudon, K. C and Laudon J. P (2007). Management Information Systems: Managing the Digital Firms. 10th ed. Prentice-Hall.

[3] Wayne A, Nelson, Artificial Intelligence Knowledge Acquisition Techniques for Instructional Development, Instructional Development, Volume 37, Number 3, pp. 81-94, Sept. 1989. Also available from: http://www.springerlink.com/content/t124762278p4q73l/

[4] Cao, T., Martin, E. & Compton, P., On the convergence of incremental knowledge base construction, in Proceedings of 7th International Conference of Discovery Science‟, pp. 207–218, 2004.

[5] S. S. Patil, B. V. Dhandra, U. B. Angadi, A. G. Shankar, and Neena J., Web based Expert System for Diagnosis of Micro Nutrients Deficiencies in Crops, Proceedings of the World Congress on Engineering and Computer Science, Vol I, WCECS 2009, October 20-22, 2009, San Francisco, USA.

[6] Hamdan A., and Al Saiyd N., A Framework for Expert Knowledge Acquisition, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.11, November 2010 http://epistemics.co.uk/Notes/63-0-0.htm.

[7] Bosch J. Software architecture: The next step. Proceedings of the 1st European Workshop on Software Architecture, (EWSA2004) (Lecture Notes in Computer Science, vol. 3047, pp. 194–199 Springer: Berlin, 2004.

[8] Bass L, Clements P, Kazman R. Software Architecture in Practice, 2nd ed., Addison-Wesley Professional: U.S.A., 2003.

[9] Capilla R., Due˜nas J. C., Nava F., Viability for codifying and documenting architectural design decisions with tool support, Journal Of Software Maintenance And Evolution: Research And Practice J. Softw. Maint. Evol.: Res. Pract. 2010; 22 PP. 81–119. Also available from: http://web.ebscohost.com/ehost/pdfviewer/pdfviewer?sid=1b1195c9-77fc-4256-99aa-992f7de70aa2%40sessionmgr14&vid=9&hid=7.

[10] Mironela PÎRNĂU, Methods for Knowledge Acquisition, Metalurgia International, vol. XIII, no.9, pp. 99-102, 2008.

[11] Botha, A. P., Knowledge: Living and Working with It, Juta & Co, South Africa, 2007, Also available from: http://epistemics.co.uk/Notes/90-0-0.htm.

[12] http://epistemics.co.uk/Notes/90-0-0.htm

[13] http://www.ajrhem.com/AJRhem-KAF%20Demo/Knowledge_Models.cfm.htm.

[14] L. Holder, Z. Markov and I. Russell, Advances in Knowledge Acquisition and Representation, International Journal on Artificial Intelligence Tools Vol. XX, No. X (2006), pp. 1–8, World Scientific Publishing Company Also available from: http://www.cs.ccsu.edu/~markov/papers/ijait06.pdf.

[15] Robert R. Hoffman, Human Factors Contributions to Knowledge Elicitation, Institute for Human and Machine Cognition, Pensacola, Florida, 2008.

[16] Joseph D. Novak , The Theory Underlying Concept Maps and How to Construct Them, Cornell University, http://www.ajrhem.com/AJRhem-KAF%20Demo/Knowledge_Models.cfm.htm