Journal of mathematics and computer science 11 (2014), 93-104

Contents list available at JMCS

Journal of Mathematics and Computer Science

Journal Homepage: www.tjmcs.com

An Ontology-Based Approach For Software Architectural Knowledge Management

Narges Choobdaran [1], Sayed Mehran Sharfi[2] , Mohamad Reza Khayyambashi[3]

[1]*Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Esfahan, Iran*
[2]*Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Esfahan, Iran*
[3]*Computer Department, Faculty of Computer Engineering, University of Esfahan, Esfahan, Iran*
choobdaran.narges@yahoo.com, sharafi@iaun.ac.ir, m.r.khayyambashi@eng.ui.ac.ir

Article history:
Received April 2014
Accepted May 2014
Available online June 2014

Abstract:

Over the past few years, a large number of models, ontologies and tools have been proposed to capture, share and the management of architectural knowledge (AK) and particularly architectural design decisions (ADD) as an important part of AK of a software-intensive system. However, the growing tendency in Globalization of Software Development sets the stage for new challenges in the management of AK in a geographically distributed context in which it seems the existing AK models and tools are no longer sufficient for such setting. In this paper we develop an ontology-based approach to manage AK in order to partly mitigate the deficiencies of existing AK approaches in a distributed software devotement.

Keywords: Software Architecture, Architectural Knowledge Management, Architectural Design Decision, Ontology

1. Introduction

The software architecture society has had a remarkable tendency towards the architectural knowledge concept in recent years. Architectural knowledge management (AKM) is of special importance for global software development (GSD); software systems, which require the cooperation and coordination of, team members distributed across geographically. Architectural knowledge is a knowledge that is produced along with software architecture during architectural production process. Architectural knowledge is considered an essential element for architectural production process in a way that it can enhance the quality of that process and, consequently, the quality of architecture itself. Some define architecture knowledge as: AK= design decisions +

architectural design [3]. It seems that most people agree that at least one part of architecture includes rationale, assumptions and decisions that lead to a specific design. Ordinary methods to record architectural knowledge include documenting knowledge in files or using tools. However, the following challenges are seen in the knowledge management of GSD systems [17]:

- Evolution: The requirement for change and development is seen along the life of a software program [10]. Software architecture can be considered a decision making process throughout which the software architect ought to make the right decisions at the right times. People do not usually work on a single system during their lifetime. The different dialects of stakeholders in GSD and growing complexity due to the expansion of global software systems have led to some improper AKM produced along the production phases of software architecture. [4] For example, a system designer might leave the project when the system is completed or is being implemented. If the systems designer does not document their knowledge and information system, that precious knowledge may be lost due to the systems designer leaving. If the system requires changes in the future, either inexperienced decisions will have to be taken or a lot of time be spent on the retrieval and reconstruction of the lost knowledge failing to record architectural knowledge and/or design decisions in GSD leading to higher costs of software systems maintenance and completion.
- Knowledge coordination and consistency: In distributive systems, each system activity is done in a certain section of the organization. An important challenge in such systems is to coordinate and adapt with the knowledge that exists along architectural phases. For example, with a requirement changing, the decisions and products related to that requirement will also change. The information traceability and adaption capability in those systems will enhance our understanding of architecture.
- Communication: Proper interaction among stakeholders means that both the sender and receiver have a common understanding of the subject in question. Geographical distances in GSD deprive the stakeholders of the opportunity to have face-to-face communication despite the fact that stakeholder interaction is very important in GSD systems.
- Control: Focusing on the management of the tasks done along architectural phases and recording it are important for understanding and completing software systems, but challenging in GSD systems.

In order to solve these challenges, alternatives are required that develop communication in a distributive medium among individuals and have an array of characteristics based on knowledge coordination and communication strategies. That knowledge may be exchanged among stakeholders informally through simultaneous communication such as chatting or through non-simultaneous communications that consists of recording knowledge in weblog or wiki email. Wikis are lightweight documentations with a shared structure suitable to keep distributive knowledge. In fact, wikis use proper function to store and keep the history of the activities done, thus, removing knowledge-sharing problem and determining stakeholder's roles [6, 7]. Another advantage of wikis is easy access to pages and the possibility to make changes to them by connecting to the internet in addition to using a simple browser. A shortcoming of wiki pages is their inability to understand semantic relationships between entities. Semantic wiki is a developed wiki medium based on semantic technology that supports such semantic concepts as semantic annotation and semantic investigation. Semantic wiki can be used in different software engineering fields such as architectural design and architectural knowledge reuse [8].

Design decision storage and maintenance and the logic beyond that as part of the knowledge plays an important role in effective interaction between stakeholders along architectural phases and also in architectural knowledge completion and maintenance [5]. The existing models and anthologies have paid less attention to offering a proper ontology in order to complete the distributive software system and using the semantic wiki for stakeholder interaction, knowledge reasonability and architectural knowledge marketing considering decisions as a primary entity. This article aims at offering a proper ontology to complete the distributive software system and using a semantic wiki called data wiki in order to remove these challenges and shortcomings.

2. Review of Literature

As said above, the recording and management of architectural knowledge in a decisive manner through a systematic method will improve an organization's architectural capabilities, stakeholders' interaction, quality and traceability of architectural design, and software systems maintenance, and completion. By using such a method, knowledge vaporization can be avoided to a great extent. Therefore, researchers and experts have made great efforts in the past few years to develop tools, models and ontologies that can explicitly record and share architectural decisions. These efforts have resulted in many different models emerging to record architectural decisions. One of the initial models in the field of ADD was proposed by Tyree and Akerman [20,22] which was used to model ADD as a text template. This template is used to record the ADD by capturing design issue, assumptions and constraints of resulting system, arguments for making decisions, its implications and its relationships with other decisions and artifacts. Neil et al. have claimed that they can reduce the effort made to record design decision by using patterns. They compared pattern with Tyree's decision template and noticed that a lot of characteristics of patterns match the entities of Tyree's decision template. But architectural patterns cannot alleviate software architect from all responsibility from documenting the ADDs. For instance, the architect should make the documentation personally for application-specific decisions [21].

Kruchten suggests an ontology to model architectural design decision in software- intensive and complicated systems. [9] in his ontology, each architectural design decision may be placed in one of the following categories: existence decisions, behavior decisions, property decisions. In Kruchten's ontology, each design decision may have the following characteristics: rationale, state, class. This type of classification can be useful for investigations into a variety of design decisions belonging to a specific subject matter or qualitative specification. In addition to the characteristics and specifications that each design decision can have, a design decision may have attachments or relationships with other design decisions. The tool offered for this ontology is capable of listing decision and relations, visualizing design structure and temporal view of design decisions. This model and its tool neither do consider requirements relationships and their effects nor the aspects of knowledge sharing and stakeholders interactions.

Barbar et al. [11] offered a data model for software architectural knowledge. Their model identifies and defines architectural constructs and their interrelationships that constitute architectural and design knowledge. By using this model, one can support architectural process activities such as architectural evaluation. This data model includes twelve design decision elements. Some of the most significant elements include design decision, architectural scientifically requirement and rationale. DAMSAK connects architectural decisions with architectural scenarios and important architectural requirements and can be used in architectural evaluation methods such as Architecture Tradeoff Analysis Method (ATAM). PAKME is a web-based AKM tool aimed at supporting AKM in architectural production processes that support the data model DAMSAK. This tool offers its services in four modules: Data search service, Architectural knowledge maintenance service, Knowledge recording service, and Architectural knowledge display service. This tool has been suggested to support knowledge in architectural evaluation but does not focus on architectural completion. It uses web 1 technology for stakeholder's interactions.

Capilla [10] offered a model for design decisions that can, not only be used in architectural production process but also in providing better support for completion of architectural design decisions. This model is composed of three main parts: project model, architectural model, and decisions model. In the decision model, the specifications of design decision are divided into two categories: mandatory attributes and optional attributes. ADDSS offers a web- based tool following this model with capabilities such as architectural knowledge tracing, design decisions recording and temporal view of architectural knowledge. The sharing process and shareholders' interaction follow the technology in web 1. ADDSS has been the only model for completion of software architecture using design decisions knowledge that has disadvantages such as inability to seek knowledge, not considering other alternatives in the model and not considering sensitivity points, risk and non- risk. Using web 1 technology to implement this model as a tool to store software architectural knowledge in data search, data organization and data access and maintenance entails some challenges.

We focus on using design decisions knowledge to offer an ontology to complete software systems along previous works and using semantic wiki to share architectural knowledge and reusing architectural knowledge to facilitate system completion and maintenance. To better understand this, the problem can be broken up as follows:

- Establishing an ontology to use design decisions in order to complete and keep software architecture: What concepts should be considered in that ontology so it can support the architectural completion process? How can the relationships between entities and attributes in question support that process?
- Data retrieval, knowledge tracing and their compatibility play a key role in architectural completion process in addition to maintenance software architectural knowledge. How could knowledge inference capabilities including knowledge search, tracing and data compatibility be developed by using proper tools and ontology?
- A current problem is the inability to share knowledge among stakeholders. How can a proper strategy be used to establish the right interaction among stakeholders?
- What are the costs and advantages of developing that ontology and using semantic wikis? Do that tool and model have better efficiency compared to older methods such as ADDSS?

This article aims at studying the first three problems. The costs and advantages of this alternative will be studied in future works.

3. Proposed Model

An ontology is a set of formal definitions for the concepts in a certain area and the relations between them. An ontology along with the instances defined for its classes makes up a knowledge base for the area in question in order to share knowledge. An ontology may be expressed by RDF or OWL languages. Some of the advantages of using RDF include the ontology decipherability for machines, enhanced comprehensibility for humans, and the ability to investigate and infer knowledge. Ontology, RDF and OWL are part of the semantic web aiming at better management of architectural knowledge in web- based systems. The framework offered for modeling design decisions knowledge is shown using UML in the following figure 1. This ontology is defined using the relations between entities and their attributes. The main entities in this ontology include stakeholder, concern, decisions and architecture. Below, the details of this ontology are examined. As we all know, software architectural design starts with a series of problems. Transparency and completeness of this part constitute a principle for AKM. A software architect examines the concerns and contents of the architecture to find and define most important architectural requirements. Two main entities for problem expression include the stakeholder and the concerns related to them. These two entities are defined as follows:

- Stakeholder entity: The concepts are expressed according to standard IEEE1471-2000[12] and indicate persons that participate either directly or indirectly in different phases of software system design.
- Concern: each stakeholder has some concerns and the system must be answerable to remove those concerns and arrive at the stakeholder's purposes. [12]

The next main entity is design decisions that are defined as a bridge to connect the requirements and architectural design and it can be said that decisions are the main element to describe software architecture in this ontology. [16] The attributes aimed at for design decisions for system completion are as follows:

- Author/ Responsible: Along a decision making process, some of the decisions are local where a stakeholder plays a part. However, some other decisions are inclusive, meaning that, some stakeholders are related thereto. Knowledge of the person(s) that make the decision and its responsibility can be important in project completion and other individuals' follow- up.
- Relate- to Dec: This attribute designates decisions that are related to a decision. A decision affects other decisions. By using this option, the list of the decisions related to this decision is saved. This attribute emphasizes system completion because following the change of a decision, all other decisions related to that decision are identified and the tracing and establishing compatibility between those requirements and decisions are facilitated.
- Type- impact- component: In the previous attribute, the pieces related to a decision were introduced. In this attribute, the type of the relationship between this decision and developed pieces is suggested. This relationship consists of development, deletion and change.

- Type decision: According to Kruchten’s ontology, each architectural design decision may be placed in one of the following categories: entity decisions, attributive decisions or executive decisions. [14]
- Status: Each decision has different states during its lifetime: rejected, approved, obsolesced, decided. These states are suggested based on Kruchten’s ontology. [14]
- By using history entity, the history of decisions can be kept which, in fact, designates the completion of a design decision during a time span. This entity is used in order to display different versions of a single decision. For that, the attributes state, version and date are used.

The decisions rationale has been selected as a justification for decision. This rationale must be explicitly documented with design rules and design constraints so that stakeholders will be able to comprehend it. While the system is being completed, it is important for decision selection method and rationale to exist between the existing alternatives. In this model, decision-making rationale has been suggested as an entity and has the following attributes.

- Assumption: Indicates the hypotheses that must be considered by selecting this decision.
- Constrain- design: By choosing each decision, a series of limits might be exerted onto the system. This attribute indicates the limitations of these decisions.
- Measure- Evaluation: Each decision is selected based on the specific evaluation with an eye to the stakeholder’s requirement. This attribute indicates the evaluation criterion for selecting this decision.
- Sensitivity Point: Architectural decisions have specific effects on one or more architectural qualities. Also, an architectural decision might result in an undesirable effect due to said qualitative attributes. A non- risk is an architectural decision that seems safe according to the analysis. Identified risks may be a basis for diminution of architectural risk.

In order to arrive at a requirement, several alternatives may be proposed. Recording the positive and negative points of the alternatives and the reason why they are rejected are important for system completion. While the system is being evolutes, the architect can be informed of the reasons why the decision are not approved by reviewing them and, if a decision requirements changing, one can have quick access to other alternatives proposed before and avoid waste of time and money. In addition to the above mentioned attributes for design decisions and their part in completing decisions, the relationships between entities are also important. The combination of these two elements indicates the relation between them. The relationship between requirements and architectural design has been made possible through the existence of design decisions. These relationships and their consideration play a key role in architectural analysis, architectural knowledge reuse and completion. First, the relations between requirements are studied: [19]

- Change to: A requirement may result in another requirement changing if a new version of the requirement is proposed.
- Refined to: A requirement may be a result of correcting and refining several requirements, which makes a hierarchical construct. Some of those requirements are related to other requirements as part of a whole.
- Conflict: A requirement might interfere with another requirement. Meeting a requirement might overshadow the effect of another requirement, as in security and effectiveness. This

case includes states where both requirements can be met or when these requirements affect each other negatively and the trade- off between them requirements recording.

- Require: The completion of a requirement depends on the completion of another requirement.

Each decision may affect other decisions. The relationships include constrains, forbids, conflict with, is an alternative to, is bound to, is made of. [14] The knowledge management process in the ontology offered is as follows: Each stakeholder has a set of concerns identified as either functional requirements or non- functional requirements and affecting each other. The correct identification of those requirements and their effects on each other affect the solution space, too. Also, considering the changes of those requirements in the future plays an important role for system completion. [15] Qualitative scenarios are employed to identify and describe the qualitative attributes of software. These scenarios offer an explicit expression of qualitative attributes. Many concerns may be related to one decision making group and therefore can be examined as a single group. The grouping of concerns has been done for the purpose of their quick retrieval and their classification for the purposes of the facilitation of reuse and classification of data. After the analysis of the requirements related to a concern, different alternatives are suggested for solution in the decision making space. Considering the criteria in stakeholders' minds, one of those alternatives is selected as the final decision. Each decision includes a rationale to take that decision and a history entity that is useful for maintenance of the system and the individuals who work on system completion.

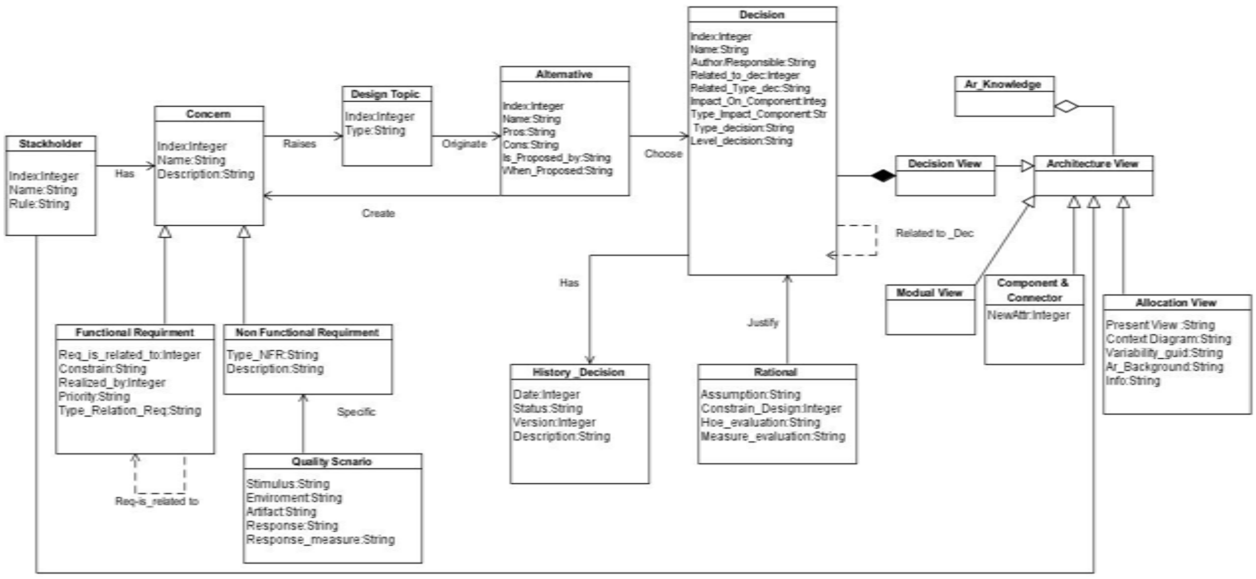


Figure 1: Ontology Model For Knowledge Management

4. Using semantic wiki tool to support conceptual model.

The semantic web aims at making the data on the web comprehensible and enhancing inter-personal cooperation. Therefore it can be used to share architectural knowledge and data solidification. Making comprehensible aims at providing the ability to infer the new knowledge using response to users' investigations into the existing knowledge.

Data- Wiki is a semantic media wiki with attributes of semantic nature which can be used in software engineering domains such as sharing knowledge among stakeholders, reuse and tracing of knowledge. The semantic features in this tool can be helpful for better data search and data retrieval compared to other tools. Here, we focus on the sharing, tracing and semantic search attributes of the architectural design, decisions, requirement recording. The executive details of Data wiki are offered below.

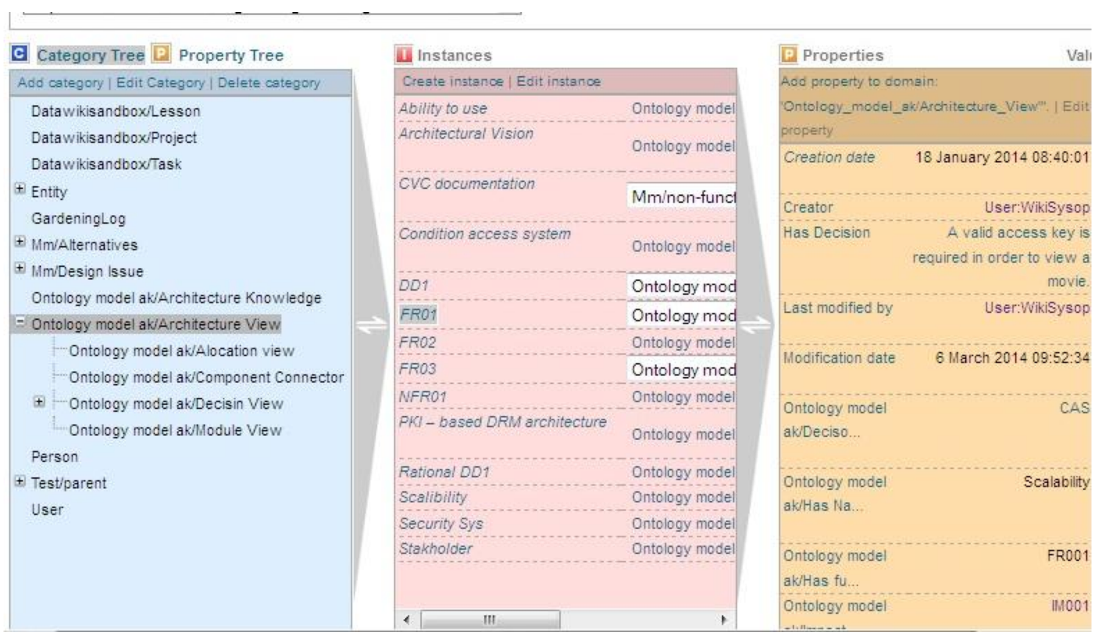
- Ontology support: Semantic wikis are based on ontology and are used to explain wiki pages and existing data on wiki pages. The feature Data Explorer makes it possible for the user to use the ontology in this medium including instances, group and attributes and with which knowledge models are expressed. The ontology offered in the previous stage is composed of four main parts shown in the following table. Figure 2 depict ontology model that import in Data-Wiki.

Table 1: Ontology Construct In Data Wiki

Example Datawiki	construct Data Wiki	ontology construct
[Category:Requirement]]	Category	Class
[[req id:FR-001]]	Property	Class properties
[[is proposed by:Stakeholder A]]	Property that link to the instance of other Class	Class Relationship
Functional Requirement, specify [[Category:Requirement]]	Category Sub categorization	Subclass of

- Semantic annotation: Data wiki supports semantic annotation on wiki pages. This very simple task for annotating requirements, design decisions and architectural products is done by adding [[Category: Concept Name]] in the existing editing box based on the entered ontology.
- Semantic tracing: This feature indicates semantic tracing through semantic annotation. In ordinary wikis, tracing is done as relation between wiki pages without semantic concepts while in the semantic wiki, the meanings of these links are determined and distinguished through the existing relations in the entered ontology.
- Semantic query: A user link for query makes it possible to develop semantic queries using semantic concepts and data. These queries are used for search in semantic data using the semantic search language SPARQL.

The ontology offered in the previous section is implemented using the software protégé, which is for establishing a knowledge base and developing semantic concepts between entities. Further we will show how the developed ontology in this software can be implemented by using semantic concepts in the software Data- wiki and we will explain knowledge sharing and saving abilities [13].


Category Tree
Property Tree
Add category | Edit Category | Delete category
Instances
Create instance | Edit instance
Properties
Val
Add property to domain: "Ontology_model_ak/Architecture_View". | Edit property

Datawikisandbox/Lesson
Datawikisandbox/Project
Datawikisandbox/Task
Entity
GardeningLog
Mm/Alternatives
Mm/Design Issue
Ontology model ak/Architecture Knowledge
Ontology model ak/Architecture View
Ontology model ak/Alocation view
Ontology model ak/Component Connector
Ontology model ak/Decisin View
Ontology model ak/Module View
Person
Test/parent
User

Ability to use	Ontology model
Architectural Vision	Ontology model
CVC documentation	Mm/non-funct
Condition access system	Ontology model
DD1	Ontology mod
FR01	Ontology mod
FR02	Ontology model
FR03	Ontology mod
NFR01	Ontology model
PKI – based DRM architecture	Ontology model
Rational DD1	Ontology model
Scalibility	Ontology model
Security Sys	Ontology model
Stakholder	Ontology model

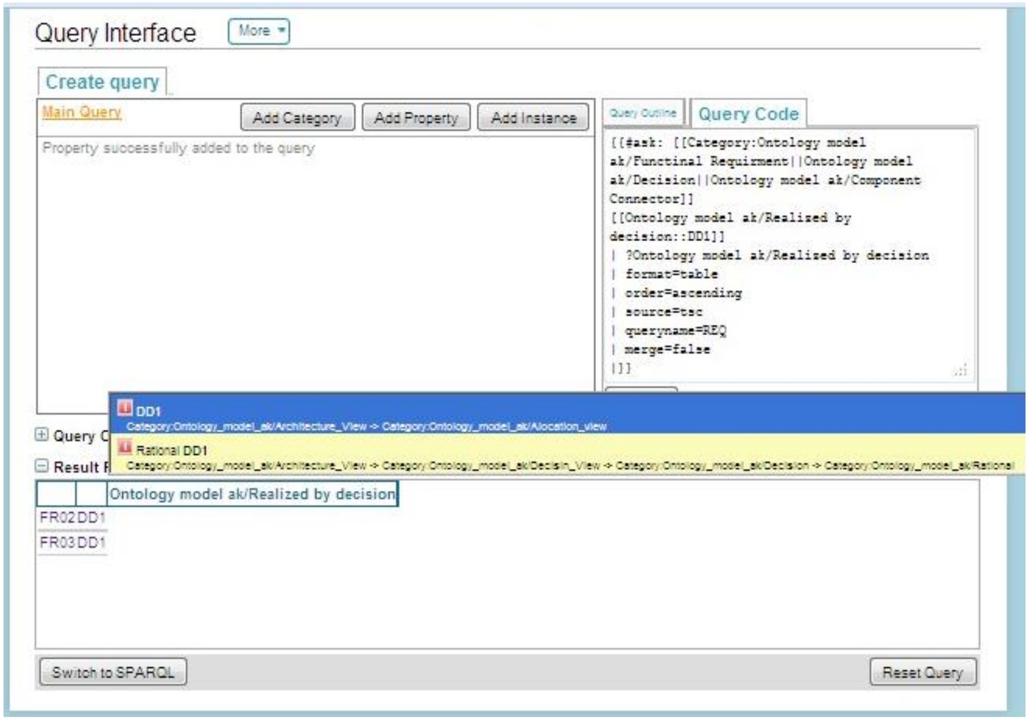
Creation date	18 January 2014 08:40:01
Creator	User:WikiSysop
Has Decision	A valid access key is required in order to view a movie.
Last modified by	User:WikiSysop
Modification date	6 March 2014 09:52:34
Ontology model ak/Deciso...	CAS
Ontology model ak/Has Na...	Scalability
Ontology model ak/Has fu...	FR001
Ontology model	IM001

Figure 2: Import Ontology Model in Data Wiki

5. Case Study

The documentation related to the software architecture of a case study system is entered into the Data wiki environment, and a semantic relationship is established among entities using semantic annotation. Since the existing architectural knowledge and requirements are linked to one another using the semantic annotation in this tool through ontology, semantic search can be used to extract the existing knowledge for better understanding the saved knowledge.

A WYSIWYG editor in Data Wiki, with image upload functionality, was implemented to allow users to copy software documentation content in popular text editors and paste it. In the scenario offered by the architect, changing a functional requirement requires tracing the effect of requirement change on other requirements and its effect on the decisions and products related to it. Tracing requirements, the decisions and products related to those requirements will provide the ability to establish compatibility between requirements and products. For example, the architect requirements to list qualitative and non- qualitative requirements to change a requirement which are related to DD1 decision and want to comprehend the effect of that decision on requirements. Using semantic search in Data wiki, the architect extracts requirements that are related to decision DD1. That way, the architect can discover requirements that lead to making the decisions DD1 in the shortest possible period. Part A of Figure 3 deals with query design. Below that, the results are shown as a table. The results show that the decision DD1 was taken in line with requirements FR03, FR02. If a decision changed during completion period and has different versions, the requirements of each version and their products can be easily extracted and the changes be examined with a semantic search.



Property successfully added to the query

```
[[#ask: [[Category:Ontology model
ak/Functinal Requirment||Ontology model
ak/Decision||Ontology model ak/Component
Connector]]
[[Ontology model ak/Realized by
decision::DD1]]
| ?Ontology model ak/Realized by decision
| format=table
| order=ascending
| source=tsc
| queryname=REQ
| merge=false
|]]
```

DD1
Category:Ontology_model_ak/Architecture_View -> Category:Ontology_model_ak/Alocation_view

Rational DD1
Category:Ontology_model_ak/Architecture_View -> Category:Ontology_model_ak/Decisin_View -> Category:Ontology_model_ak/Decision -> Category:Ontology_model_ak/Rational

		Ontology model ak/Realized by decision
FR02	DD1	
FR03	DD1	

Figure 3: Semantic Search in Data Wiki

6. Conclusion

The main idea of this article is to develop an ontology using design decisions knowledge to complete software system and using semantic wiki to keep, retrieve and share knowledge with various stakeholders in distributive media. The tool used to implement this model was Data wiki. The ontology offered and applied tool complement each other. The abilities of this ontology and offered tool are as follows:

- Using semantic annotation in software architectural knowledge and decisions: The user can easily select part of the text and annotate it. Using annotation provides a better understanding of the existing documentation according to the defined ontology. In addition, it plays an important role in semantic searches.
- Architectural knowledge tracing: Semantic annotation is developed based on the existing relations in the entered ontology. This annotation also includes the interactions between stakeholders. For example, a system includes many requirements suggested by other stakeholders. This relationship can be traced through semantic annotation.
- Architectural query: Semantic annotation has the possibility of data search by using the search language SPARQL. For example, searching all the decisions that were related to requirement XX and studied by stakeholder Y.
- Checking decision compatibility: By using data tracing attribute, the user will be able to study incompatibility between architectural design and requirement using the inference attribute. For example, when a requirement changes upon system completion, the entities related to that requirement can be extracted, the related documentation updated, and knowledge incompatibility avoided.
- Requirements defects: By using the existing relations between entities, one can find out if the suggested requirement was followed up to the implementation stage or it was dealt with defectively.

- Developing a proper medium for knowledge sharing: Suing this tool along with a proper ontology makes a non- simultaneous shared medium for the stakeholder where the data existing in the system can be easily traced if a new stakeholder, such as an architect, is added and can be employed as an instructional aspect for new stakeholders.

The next steps in research would be the comparison and evaluation of the suggested ontology to determine levels of productivity in architectural knowledge sharing and its effects on architectural completion compared to existing file- based methods in addition to developing a model to keep architectural knowledge and use it in industries and transform existing implicit knowledge to formal knowledge to use it for inference and logic.

8. References

[1] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice, 2nd edition, SEI Series in Software Engineering, Addison-Wesley Pearson Education, 2003.

[2] Kruchten, P., P. Lago, and H. van Vliet. Building up and Reasoning about Architectural Knowledge. In Second International Conference on the Quality of Software Architectures (QoSA 2006), volume 4214 of Lecture Notes in Computer Science, pages43–58, V¨aster°as, Sweden, 2006. Springer Berlin / Heidelberg. Cited on page

[3] D. Falessi, G. Cantone, and M. Becker. Documenting design decision rationale to improve individual and team design decision making: an experimental evaluation. In Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering (ISESE '06), pages 134–143, New York,NY, USA, 2006. ACM Press

[4] Holmstr¨om, H., E. ´O Conch´uir, P. J. °Agerfalk, and B. Fitzgerald. Global Software Development Challenges: A Case Study on Temporal, Geographical, and Socio-Cultural Distance. In IEEE International Conference on Global Software Engineering
(ICGSE'06), pages 3–11, Florianopolis, Brazil, 2006. IEEE Computer Society.
Cited on pages

[5] Clerc, V., P. Lago, and H. van Vliet. Assessing a Multi-Site Development Organization for Architectural Compliance. In 6th Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), Mumbai, India, 2007a. IEEE Computer Society. Cited

[6]B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth. Wiki- Based Stakeholder Participation in Requirements Engineering.
IEEE Software, 24(2):28–35, 2007.

[7] C. Silveira, J. Faria, A. Aguiar, and R. Vidal.Wiki Based Requirements Documentation of Generic Software Products.
In Proceedings of the 10th Australian Workshop on Requirements Engineering (AWRE), pages 42–51, 2005.

[8] S. Schaffert, F. Bry, J. Baumeister, and M. Kiesel. Semantic Wikis. IEEE Software, 25(4):8–11, 200

[9] Kruchten, P., An Ontology of Architectural Design Decisions in Software-Intensive Systems, Proceedings 2nd Groningen Workshop on Software Variability Management, Groningen, pages 109-119, 2004.

[10] Capilla, R., Naval, F., and Dueñas, J.C., Modeling and Documenting the Evolution of Architectural Design Decisions, Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK/ADI), 2007.

[11] Babar, M.A., Gorton, I., and Kitchenham, B., A Framework for Supporting Architecture Knowledge and Rationale Management, In Rationale Management in Software Engineering, A.H. Dutoit, et al., eds, Springer, pages 237-254, 2006.

[12] I. S. 1471-2000, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," 2000.
[13] http://diqa-pm.com/en/DataWiki
[14] Kruchten, P., An Ontology of Architectural Design Decisions in Software-Intensive Systems, Proceedings 2nd Groningen Workshop on Software Variability Management, Groningen, pages 109-119, 2004.
[15] Bilal Saeed Raja, M. Ali Iqbal, and Imran Ihsan" Moving From Problem Space to Solution Space ",World Academy of Science, Engineering and Technology 11 2007
[16] Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla and Muhammad Ali Babar, "A Comparative Study of Architecture Knowledge Management Tools". Journal of Systems and Software, 83(3):352–370, 2010
[17] Agerfalk, P. J., B. Fitzgerald, H. Holmstr¨om, B. Lings, B. Lundell, and E. ´OConch´uir.A Framework for Considering Opportunities and Threats in Distributed Software Development.In International Workshop on Distributed Software Development, pages47–61, Paris, 2005. Austrian Computer Society.
[18] Noy NF, McGuinness DL (2001) Ontology development 101: A guide to creating your first ontology. Tech. rep., Stanford University
[19] He Zhang, Juan Li, Liming Zhu, Ross Je, Yan Liu ,Qing Wang, Mingshu Li, Investigating Dependencies in Software Requirements for Change Propagation Analysis
[20] Tyree, J., and Akerman, A., Architecture Decisions: Demystifying Architecture, IEEE Software, 22(2):19–27, 2005.
[21] Harrison, N.B., Avgeriou, P., and Zdun, U., Using Patterns to Capture Architectural Decisions, IEEE Software, 24(4):38-45, 2007.
[22] Mojtaba Shahin, Peng Liang, Mohammad-Reza Khayyambashi: Architectural design decision: Existing models and tools. WICSA/ECSA 2009: 293-296