Helping Architects In Retrieving Architecture Documents: A Semantic Based Approach

Rambabu Duddukuri[1], Prabhakar T.V[2]

[1] Member, Technical Staff, Oracle India Private Ltd,
Bangalore, India -560029.
Rambabu.Duddukuri@oracle.com

[2] Professor, Department Of Computer Science and Engineering,
Indian Institute of Technology Kanpur, Kanpur India - 208016
tvp@iitk.ac.in

Abstract. Large organizations have a need and challenge of archiving the architecture work done on software projects. Knowledge management in such organizations depends on how well the company preserves the knowledge acquired on completed projects and how well the company provides facilities to retrieve that architectural knowledge. Architecture properties such as styles, patterns, tactics and quality requirements play a major role while architecting the systems. Annotating the documents with such properties helps the architect in searching for architecture documents at a later date. There exist a large number of relationships between these architecture properties. A huge knowledge base is required to know about the best practices and the existing relationships between them. In this paper, we present an ontology for these architecture properties. We describe how this ontology can be used in various applications like semantic based search, academic purpose and building of new systems using the best practices.

Keywords: Software Architecture, Ontologies, Semantic search, Architecture styles, Patterns, Tactics, Archiving, Annotation.

1. Introduction

The architecture phase has become an integral part in the design process of large and complex systems. Architecting a system, deals with modeling high level structures of the system in terms of views, architecture styles and patterns. Documenting the architecture of a system is essential in understanding the design decisions taken during this process thereby serving as a medium for the stakeholders and project developers to communicate. Several frameworks [1, 2, 3, 4, and 5] exist for designing the architecture of the system. All the frameworks specify representing the system in several viewpoints in accordance to the stockholder's concerns. The typical job of an architect includes meeting the stakeholder concerns, making a collection of architectural requirements in various forms, turning them into quality scenarios and then architecting the system through various architectural styles, patterns and available architectural knowledge such that the constraints are met. The most important deliverable of the process of designing the architecture is the architecture document describing the structure of the system through its various

views. This document is used to communicate to the customer, for analysis of various quality attributes, development and future maintenance. In essence they form the pivot around which all further activity about the software revolves.

In Software organizations where a large number of projects are documented daily, preserving the knowledge of the architectures for further reference plays a major role. Architectural level re-use is another activity that is impacted by documentation. Our previous work addresses these problems [11] and provides a new way of archiving the architecture documents and extracting these documents with a comprehensive search. In that approach, we have tried to annotate the architecture documents with architectural properties like styles, patterns, tactics, domain, technology components, quality requirements, and other framework standards. We represented this metadata as an XML file and search is performed on this metadata as an XML tag search.

Several relationships exist between the patterns and styles, which in turn relate to the problem domain of the system. For instance, Real time systems mainly deal with concurrency patterns, resource patterns, and safety and reliability patterns. Similarly financial and accounting systems have their own best practices defined. The distributed and layered architecture styles use certain pre-defined design patterns to solve the problems that might occur with regards to distributed communication, preserving data integrity, structuring the application logic etc. Suppose that, we try to find some architecture documents using layered architecture style, we might generally look for architecture documents using the patterns like structuring application logic, domain logic, application logic etc. These relationships and the associated vocabulary form a huge knowledge database. A taxonomy or ontology of this knowledge base helps the architect in searching and retrieving architecture documents in a semantic way. Figure 2 depicts the conceptual overview of the Ontology we suggest for use in searching. The major concentration is on the architectural properties like problem domain, architecture styles, patterns, architecture tactics, quality requirements and the relationships that exist among them.

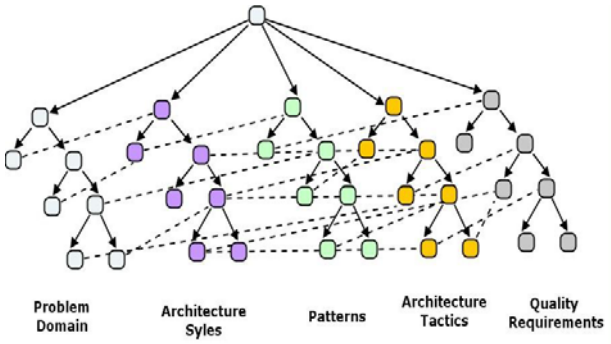

Figure 1: Conceptual overview Of Our Ontology

The rest of the paper is organized as follows. In the following section we provide an outline of the related work done in this area and briefly describe the motivation behind our work. Section 3 explains several architecture domain vocabularies that are

used in this ontology and how these architectural properties help in retrieving the architecture documents. In section 4, we tabulate the relationships used between the terms in our ontology and briefly describe about how this ontology along with some applications of this Ontology. Finally in section 5, we conclude the paper by giving a brief outlook on future work.

2. Related Work

In Grady Booch's *Handbook of Software Architecture* [6], a large number of patterns are classified which allow comparisons across domains and architecture styles. However, he does not describe the relationships between architecture tactics and quality requirements, mapped to the real life problem domains, which can also be used for searching architecture documents. According to [8], a knowledge base is developed for representation and reuse of software patterns facilitating the semantic related search for the reuse of patterns. There is no such effort of mapping these patterns to other architecture properties, which will prove to be an efficient searching technique for architecture documents. Our approach to software architecture ontology is to provide a mapping between several architectural properties like patterns, styles, and tactics and problem domains. Figure 2 depicts several possible terms of the OntoSoftArch Ontology and the relationships between them as a concept map [9]

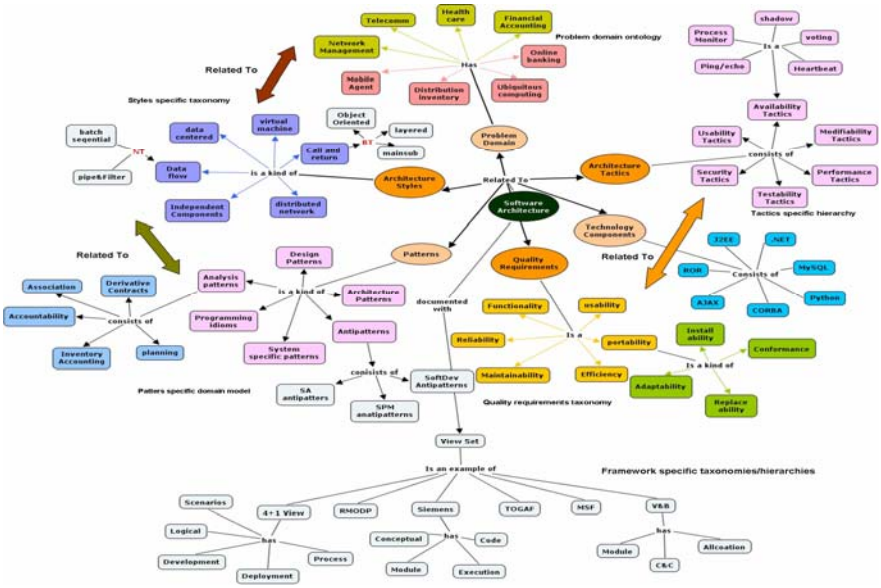

Figure 2: A fragment of Our Ontology as a Concept Map

3. Architectural Descriptions

We now describe the architecture properties that can be used in annotating the architecture documents. Our description regarding these architecture properties is necessarily brief and mainly concentrate on the search criteria of the documents.

Problem domain: A good architecture document depends on how well the domain model is identified and how well the commonality and the architect depicts variations among different instantiations of the system. Lack of domain knowledge in the architect can result into chaos during the project development. Problem frames [12], help in decomposing the problem into several sub problems but not help the architect in deriving the architecture of the system. Often the major concerns of the software architect vary from domain to domain. For example, for Telecommunication systems, that are distributed systems, solving the problems with distributed communication, configuration, session data storage, preserving data integrity etc. are the main concerns. Patterns such as cache proxy, Broker, Remote proxy, Client session state, Fine grained locking etc are used to solve such problems. There exist many similar types of relationships between problem domain and the patterns used. Similarly in the domain such as Aerospace and defense, quality requirements such as performance and reliability are the main concerns. We have identified such problem domains and the relationships with other architecture properties in our ontology. Often the architects are faced with queries like - can we retrieve the design documents with similar problem domain that is being worked upon? Such queries can be easily addressed while searching the repository if the architecture documents are annotated within this architecture property. The search will also be performed with the architecture properties related to this domain as explained above.

Technology components: The architecture includes hardware and software components that are not directly part of the actual design process. The design process helps in identifying various subsystems and the way they interact. These subsystems are thn mapped to technology components and they are related to each other in terms of interfaces they provide. The questions such as 1) can the architecture documents that use *MySql* database be retrieved on a Linux platform? 2) Can search be performed for the architecture documents that use the *Apache* webserver and *ODS* gateway with *CORBA* middleware and *Oracle* as the backend? Such queries can be easily answered if the architecture documents are annotated with all the technology components used in that system design. Often these technologies are related to best practices, for example the Yahoo UI library [10] is related to best practices like drag and drop, color picker, Image viewer. Similarly J2EE technology has some patterns such as Web Service Broker, Application Control, and Composite entity that are applicable in that domain. The same is the case with AJAX based applications. Many such relations are captured in our ontology; enabling search for architecture documents based on these technological issues as well as related best practices to these technologies.

Architecture Styles The architecture styles depicted in [13] are classified based on the characteristics such as data flow between components, call/return systems, data centered systems etc. Architecture styles in [27] are classified based on the views in which stakeholder is concerned. For example in an allocation view deployment and work assignment, styles are used. Several relationships exist between the architecture styles and the patterns used. For instance in layered architecture style, structuring the presentation logic, domain logic and the application logic are the main concerns. Several patterns such as Model view controller [26], packed abstraction controller,

transaction script, operation script, virtual proxy are used to resolve such problems. Similarly several patterns classified under Message routing, message transformations, message channels and message end points used in information exchange mechanisms are used in several styles like event based systems, black board styles etc. Our explanation regarding these styles and the relationships to other architecture properties are necessarily brief. We have gathered different architecture styles and relationships with other architecture properties in our ontology. Queries such as can the architecture documents be retrieved with Interpreter and Rule based system styles? Can search be performed for architecture documents, which use Blackboard and Hypertext systems? Again such queries can be easily answered by annotating the documents with the corresponding styles used in the system.

Patterns: Application of best practices comes in the form of patterns. Several new patterns are evolving every year depending on the technologies developed. Gamma et.al [16] classified their patterns into three groups, Creational, Structural and behavioral. Tichy [17] gives a catalogue of over 100 patterns and arranged them under the categories like decoupling, state handling, virtual machine etc. Zimmer [18] analyzed the relationships between the patterns by Gamma. He introduced three kinds of relationships between patterns - X uses Y, X is similar to Y, X can be combined with Y. Architecture patterns catalogued by Buschman [19] also play a significant role in architecting the system. Also, a catalogue of 72 analysis patterns [20, 21] classified under Accountability, Association, Inventory and Accounting. There are also patterns associated with technologies such as CORBA, J2EE, AJAX [23, 24, 25] etc. Often the architects are left with the questions like - what is the consequence of using common interface and then wrapping it up to integrate it. Also, for example in Application Integration, how different applications should be integrated is the major concern. Patterns such as File transfer, Messaging, Remote procedure call, and shared database provide solutions to such design problems and annotating the documents with these patterns used in the system.

Architecture Tactics: An architecture tactic is a transformation of the system from one state to other that affects one of the parameters defined by quality attributes [15]. A large number of tactics have been identified and catalogued in Bass et al [15, 26]. The classified tactics are based on the quality attribute addressed. Their classification of patterns and tactics are based on the following relationship, Quality attributes → Tactics → Patterns. Consider an example of performance related tactics and patterns. Two patterns Flyweight and Thread pool pattern uses the same tactic, reduce computational overhead thereby meeting the quality requirement Such relationships between quality attributes, patterns and tactics are depicted in our ontology. Annotating the architecture documents with architectural tactics used while making architectural decisions helps to answer queries such as 1) did we use these tactics before and what was the result? The search will also be performed on the documents handling the quality attribute related with that tactic.

Quality Requirements: Quality requirements are the architecture drivers for any successful development of the system. The degree of quality achieved may vary from system to system. The Extended ISO model [28] depicts several quality attributes.

These are classified mainly based on reliability, usability, portability, efficiency and maintainability. Quality requirements and their attributes are defined in quality attribute theory [14, 28]. The purpose of the quality attribute theory is to enable the interpretation of software architecture in terms that are meaningful to quality attributes. Several relationships exist between quality attributes and tactics as described in the previous section. Also several relationships exist between patterns and quality requirements such as system performance patterns, and patterns for performance and reliability such as Fail over cluster, Load balancing cluster, Server cluster etc. We have defined such relations between quality attributes, patterns, tactics and problem domain in our ontology. Annotating the documents with the quality attributes handled in those systems helps the architect in full filling his queries like - can we retrieve the documents with throughput of the scenario between t1 and t2?

4. Ontology Relationships

Name	Description
Used-to	Denotes a means/ mechanism
Related-To	Denotes an association relationship
Is-A	Denotes a super, subclass relationship
Is – part - of / Has	Denotes an aggregation relationship
Is-Similar-To	Denotes an equivalence relationship
Requires	Denotes an association relationship

Table 1: Relationships between the Terms

Initially we gathered all the terms to software architecture from several architecture books and research papers. We also gathered some of the important index terms from some of the major architecture books [1, 15, 18, 19, and 21]. We manually went through the terms and refined the vocabulary. We found several relationships between the terms within the domain of each architecture property as well as the relationships between the architecture properties. We also included the relationships that are already between the terms like architecture tactics and patterns [15].

Currently our ontology consists of 1470 terms from all the architecture properties like problem domain, styles, patterns, tactics, technology components and several architecture frameworks. Identification of relationships between the terms is an ongoing effort, and we are augmenting and refining the relationships. More explanation of the relationship along with the Ontology is available to download from the URL http://www.cse.iitk.ac.in/~soft_arch/ontosoftarch. The viewer support tool for viewing this ontology is also available for download. We are trying to convert this ontology into an OWL based ontology, which allow users to load into any ontology editors like Protégé.

This ontology helps the architect in understanding the existing relationships between best practices thereby enabling him to construct a new system with existing best practices. Consider a scenario where an architect wants to develop a system for Financial and Accounting domain. This ontology helps him in understanding all the analysis accountability patterns related to that domain and the quality requirements to be considered for this system.

The ontology is useful for pedagogical purposes. This ontology helps the students to clearly understand all the terms and concepts in software architecture and the existing relationships between them.

A third application would be allowing the user to semantic search for the architecture documents which are annotated based on the above said architecture properties. Our previous work [29] talks about annotating the architecture documents with the architecture properties as an XML file, enabling the user to search for architecture documents based on the XML tags. In this search, the user will be giving one architecture property based on which search should be performed. The search will also be continued on the architecture properties related to the given architecture property. If the user gives the query like "Search for architecture documents, which used Client Server Architecture style", the search will also be performed on the architecture documents using the message exchange patterns in this architecture style. The related terms will be extracted from our ontology and searched in the repository.

5. Conclusions

An ontology of software architecture helps the architect in understanding the best practices used for documenting software architectures. Ontologies can also help in semantic annotations of architecture documents. Currently our ontology is populated with patterns, styles, tactics, domain concepts and different frameworks. The knowledge base contains the terms that are normally used in software architecture and relates them semantically, allowing effective searches and reuse of best practices. We plan to extend this ontology by adding more terms and more associations. This can make more inferences among the terms and a full-fledged ontology can be developed. Next step is the construction of CASE tool to allow semantic search for the architecture documents stored in the repository based on the semantic annotations.

References

1. Clements P, Bachmann F, Len Bass, David Garlan, James Ivers, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*.
2. IEEE recommended practise for architectural description of software-intensive systems. *IEEE Standards*, pages 1–23, Sep 2000.
3. Hoffmesiter, C, Nord, R., Soni, D, Applied Software Architecture, Addison-Wesley, 2000 Kruchten, P
4. *Architectural Blueprints- The "4+1" View Model of Software Architecture*. Rational Software Corp., IEEE Software, November 1995.

5. RM-ODP Standards, ISO/IEC JTC1/SC21/WG7 Reference model of Open Distributed Processing available at http://archive.dstc.edu.au/AU/research_news/odp/refmodel/standards.html
6. Grady Booch's Handbook of Software Architecture
8. An Ontology-based Knowledge Base for the Representation and Reuse of Software Patterns Rosario Girardi and Alisson Neres Lindoso Federal University of Maranhão Campus do Bacanga, São Luís - MA, Brazi
9. Novak J.D., Cornell University *The Theory Underlying Concept Maps and How To Construct Them*, available at http://cmap.coginst.uwf.edu/info
10. http://developer.yahoo.com/yui/index.html.
11. Rambabu Duddukuri, Prabhakar T.V "On Archiving Architecture Documents'' In the Proceedings of APSEC 2005, Taipei, Taiwan.
12 Problem Frames: Analyzing and Structuring Software Development Problems by Michael Jackson
13 Shaw, M. "Making Choices: A Comparison of Styles for Software Architecture." *Carnegie Mellon University*, May 1994.
14 Barbacci, M.; Klein, M.; Longstaff, T.; & Weinstock, C. *Quality Attributes* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.
15 Len Bass, Paul Clements, Rick Kazman, and Ken Bass. *Software Architecture in Practice*. 2nd Edition, Addison-Wesley
16 Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides; Addison-Wesley, 1995
17 A catalogue of General-Purpose Software Design Patterns, Tichy, Walter F , University of Karlsruhe, Karlsruhe,Germany. http://wwwipd.ira.uka.de/~tichy/publications/Catalogue.doc
18 Relationships between Design Patterns by Zimmer, Walter, in Pattern Languages of Program Design, James O. Coplien, Douglas C. Schmidt, Editors, Addison-Wesley, 1995
19 Frank Buschmann, Regine Meunier, Hans Rohnert,Peter Sommerlad, Michael Stal, Peter Sommerlad, and Michael Stal. Pattern-oriented software architecture.
20. Fowler, M. *Analysis Patterns*. Reading, Massachusetts: Addison-Wesley, 1997.
21. Fowler, M. *Patterns of Enterprise Application Architecture*. Reading, Massachusetts: Addison-Wesley, 2003.
22 William J. Brown, Raphael C. Malveau, Hays W. Skip McCormick (III), and Thomas J. Mowbray. Antipatterns: Refactoring software architectures and projects in crisis.
23 Alur, D., Crupi, J., Malks, D. *Core J2EE Patterns, 2nd ed.* Upper Saddle River, New Jersey: Prentice Hall, 2005.
24 Gross, C. *Ajax Patterns And Best Practices*. New York, New York: Springer-Verlag, 2006
25 Mowbray, T. & Malveau, R. *CORBA Patterns*. N6w York, New York: Wiley, 1997.
26 Design Patterns, Quality Attributes and Software Architectural Tactics, Felix Bachmann, Len Bass, Mark Klein.
27 Sun Microsystems. http://java.sun.com/blueprints/patterns/MVC-detailed.html, 2000.
28 Bob van Zeist, Paul Hendriks, Robbert Paulussen, and Jos Trienekens. Quality of software products — Experiences with a quality model. http://www.serc.nl/quint-book/index.htm.