On Archiving Architecture Documents

Rambabu Duddukuri, Prabhakar T.V
Dept of Computer Science and Engg
Indian Institute of Technology, Kanpur
Kanpur, India 208016
{rambabu, tvp}@cse.iitk.ac.in

Abstract

This paper presents a novel perspective on archiving architecture documents in large organizations. Designing and architecting a system deals with modeling the high level structure of a system in terms of views, architectural patterns and styles. Aspects such as knowledge management and archiving of architecture work done on artifacts of software projects challenge the large organizations. Knowledge management in such organizations depends on how well the company preserves the knowledge acquired on projects and how well the company provides the facilities to retrieve that architectural knowledge. The main issue in identification of the architecture documents is how we annotate these documents while storing in the repository. We have identified an approach on how these architecture documents can be annotated with architecture properties, and provided a comprehensive search design on the architecture documents already stored in the database. We describe how this perspective offers a good solution on archiving architecture documents and outlines various issues worthy of further exploration.

1. Introduction

Architecting a software system deals with modeling a system in a common high level abstraction in terms of architectural structures, views, architectural patterns, styles etc. and embodiment of earliest set of design decisions about a system. There are several schools in designing the architecture of a software system. In Kruchten 4+1 view architecture [1], four different views of software architecture are defined each view supporting architectural framework. The Siemens' viewset [2] looks at the process slightly differently and uses a different set of views. Clements et al [4, 5] reconcile these very elegantly. They point out that any architecture description will be made up of views, which can be categorized to be belonging to three basic types: Module viewtype, Component and Connector viewtype and Allocation viewtype. In the IEEE 1471 standard [6] of describing architectures, a conceptual framework of architectural descriptions is established. In this framework a view conforms to a viewpoint and these viewpoints are defined with specific stakeholder concerns in mind. Architectural styles and patterns catalogued by Buschmann et al. [7] also play a significant role in architecting a software system. The technical process of designing a system includes making a collection of architectural requirements in various forms, turning them into quality scenarios, and then architecting the system through various architectural styles and available architectural knowledge such that the constraints are met. The diagram below illustrates the same.

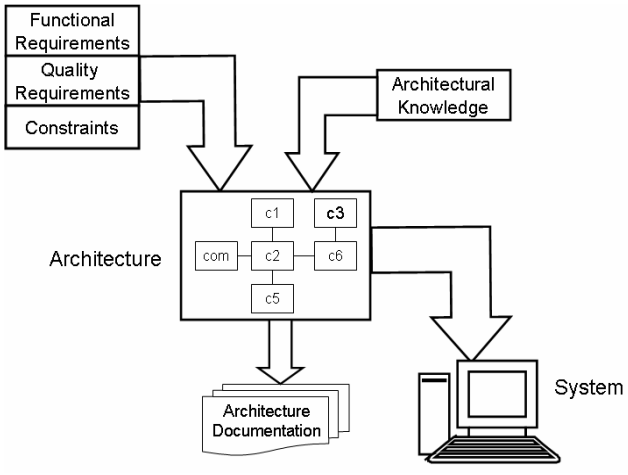

Figure 1: Technical process of System design

The most important deliverable of the process of designing the architecture is the architecture document

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

Architectural level re-use is another activity that is impacted by documentation. Large organizations where knowledge management is a major issue need to pay lot of attention to documentation issues.

Our approach addresses these problems and provides a new way of archiving the architecture documents and extracting these documents with a comprehensive search. The rest of the paper is organized as follows. In the following section, we briefly outline the Objective and scope of this approach. Section 3 describes the motivation for our problem. Section 4 illustrates the architecture properties that are used to annotate architecture documents while archiving them. In section 5, we sketch the search design on architecture documents. In section 6, we elaborate the method with a case study. Finally, in section 7 we conclude the paper by giving a brief outlook on future work.

2. Objective and Scope

Proper understanding of functionality, design and architecture of existing systems is essential before providing any constructive efforts in the system design. Knowledge management and acquisition plays a crucial role in achieving this. Extensive knowledge migration from existing projects also helps in reverse engineering projects where the system has to satisfy the functionality of existing system. It also helps in the architectural analysis of the developed project with the existing architectural design in the repository.

This approach can be applied in large organizations for searching the design documents where huge numbers of software architecture documents are maintained. This can also be used in educational organizations for analyzing the existing software designs and also for evaluating software architectures which are already stored in the repository.

3. Motivation

The main motivation in designing this comprehensive search facility on software architecture repositories are the following factors: 1) It promotes large scale architectural reuse. 2) It greatly reduces time of a software architect. 3) It helps in guiding certain tasks such as software system restructuring, code reuse and adding new functionalities to the system. Some software products generally share a basic set of building artifacts. Architects can either extend such software products or can modify them for different configurations promoting a large scale architecture reuse. Many questions arise while starting the design process of an application such as 1) Is it required to know how to use a particular component? 2) Does the experience with a development platform on the past projects is good or bad? 3) How to design the solution to a problem regarding issues like performance, component reuse, availability etc.

Most of the architecture documents are concerned with the components and connectors represented through some graphical UML tools [12], architecture description languages (ADL) like xADL [11, 15]. These approaches mainly concentrate on the identification of architecture components and the notation to be used. Quality attributes such as performance, reliability etc, Tactics [3] used such as concurrency, caching policy, voting etc., while making architectural design decisions are equally important while searching the documents for architecture reuse. We define these issues as architecture properties and provide a way to annotate the design documents with these architectural properties which further helps in searching architecture documents stored in the repository.

4. Architectural Descriptions

Architecture description mainly focuses on describing different structures of the system. The main uses of architecture descriptions are listed below

- It facilitates communication and allows getting a common understanding between different stakeholders. It also helps in understanding of existing systems.
- It promotes reusing of existing components at a high level.
- It guides the system development during its construction and evolution.

Describing architecture with architecture properties helps in achieving these functionalities. We now describe the architecture properties that can be used in annotating the architecture documents. Our description regarding architectural properties is necessarily brief and has several open issues; we hope to convince that the approach is indeed worthy of further in-depth study.

Problem domain: Understanding the problem domain is the key for successful building of any application. Lack of domain knowledge in the development team can result into the chaos of the

software product. A clear explanation of the problem domain by the domain expert in the architecture documents helps the search facility in the repository. If the architecture document is annotated with problem domain, the queries such as 1) for what problem the architecture document is? 2) Retrieve the design documents with similar problem domain? can be easily addressed while searching the repository. The search is based on keyword similarity.

Scenarios handled: Scenarios provide an intuitive way for eliciting requirements from project stakeholders. Scenario based techniques proved to be an effective way in capturing requirements, describing test cases and evaluating software architectures [9]. A scenario is nothing but an actual or desired execution trace of a software system. Scenarios may be represented in UML sequence diagrams [14]. Scenarios are also used for the specification of quality requirements. According to Bass et.al [3], quality attribute scenarios guide the specification of quality attribute requirements. When they are applied to a specific system, they are called as concrete quality attribute scenarios. According to them, general scenario generation tables can be used to construct general scenarios for each attribute, from which concrete scenarios are instantiated. Concrete scenarios specify system dependent quality requirements.

Annotating the architecture documents with concrete scenarios helps the user in searching for the documents based on these scenarios.

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 then mapped to technology components and how they are related to each other in terms of interfaces they provide. The questions such as 1) retrieve the architecture documents that used *Mysql* database on a Linux platform? 2) Search for architecture documents which used *Apache* webserver and *ODS* gateway with *CORBA* middleware and *Oracle* backend? Can be easily answered if the architecture documents are annotated with all the technology components used in that system design.

Architectural patterns and Antipatterns: Successful architecture of a product is based on application of best practices and not on the contemporary and latest practices. Architectural patterns are defined as "expressing a fundamental structural organization schema, with set of predefined subsystems, their responsibilities and a set of guidelines for organizing relationships between them [7]". Several architectural patterns such as pipe and filter, MVC pattern [13], Gough patterns [10] such as strategic, singleton patterns provide a reusable solution, describing a group of components, how the components interact and responsibilities of each component. Annotating the architecture documents with these architecture patterns provides a comprehensive search mechanism. The questions such as 1) provide the solution architecture that uses MVC pattern or strategic pattern? can be easily answered with this process.

Antipatterns define a common and formally defined toolkit for categorizing, identifying and mitigating the typical problems in software development [8]. The antipattern solution generates negative consequences to management and software development. Thus, annotating documents with antipatterns used in development process provides a means for managers and software developers to identify typical problems.

Structural properties of Architectural diagrams: Architectural structures are foundational representations of architectural information representing the base artifacts that we design. The corresponding architectural description should reflect the overall modular organization of the system. Representing the system as a set of views namely module view, CnC view, allocation view and the corresponding styles such as pipe/filter, layered, client server, publish-subscribe helps the architect in understanding the modular organization of the system. These canonical structures can be classified as below depending on the context of the development.

1. Module Viewtype: Module view represent system as collection of code units Elements in the viewtype are modules such as class, package, function, procedure etc. Relationship between them is code based such as part-of, depends on, calls generalization-specialization etc. Styles such as decomposition style, uses style are applicable in this viewtype
2. CnC Viewtype: CnC view represents system as a collection of runtime entities called components. Here the component has an identity in executing system. For examples components can be objects, processes, dll etc. Connectors provide means of interaction between components e.g., pipes, shared memory, sockets etc. Styles such as pipe-filter, shared-data, and client-server are applicable in this viewtype.
3. Allocation Viewtype: Allocation view mainly focuses on how software units are allocated to resources like hardware, filesystem, people etc. It exposes structural properties like which process runs on which processor, while file resides where etc. Styles such as Work assignment, deployment style are applicable in this viewtype.

Annotating documents with the viewtypes and the corresponding styles depending upon the development

stage provides an easy way to search for architecture documents.

Conceptual properties: These properties include definition of the problem domain specific concepts and terms, and their relationship. The concepts of a project can be documented in a glossary like form. Here each of the important terms of the problem domain is defined. These concepts of the problem can be visualized by using UML class diagram. The relationships between these concepts are modeled by generalization, aggregation and association relationships. Annotating the architecture documents with conceptual terms helps the user in answering the queries such as 1) Search for the documents which used the concept Mobile Base station? 2) Retrieve the architecture documents that used the concept patter Matching?

Architectural Tactics: Tactics are well known design practices that architects use in order to meet quality requirements. The quality attribute model defines a set of parameters that are evaluated in order to calculate the quality attribute response. An architectural tactic is a transformation on a software architecture that affects one of the parameters defined by the quality attribute model. Example tactics for performance are change the scheduling policy, synchronization policy, use an intermediary etc. Annotating the architecture documents with architectural tactics used while making architectural decisions helps to answer queries such as 1) Did we use this tactics before and what was the result?

Information Exchange between components: Information exchange between components impact the software design. The information exchange between components can be either through memory or through a physical medium. Information exchange methods can be,

- Memory
    1. Parameter passing
    2. Shared memory
    3. Global variables
- Physical medium
    1. Http, Tcp/Ip ports
    2. Remote procedure call
    3. Pipe

If the information exchange is done through memory then components can run on a single CPU except in the case of shared memory, where the components can run on different CPU's. Shared memory information exchange mechanism is useful in exchange of persistent data, which has multiple accessors. The role of components in shared memory can be producer-consumer. The consumer knows data available in the memory either through blackboard mechanism where the store informs the consumer or the repository where the consumer is responsible for it. Connectors can be data reading and data writing. Information exchange through shared memory requires synchronization mechanisms such as semaphores, monitors in case of concurrent processes. Information exchange through physical medium is done through sockets, RPC or event buses. Components in this mechanism can be either client/server or publisher/subscriber which plays different roles for the exchange of information. Components in publish/subscribe model interact via announced events. Components subscribe to a set of events and P-S runtime ensures that each published event is delivered to all subscribers. Annotating the documents with these information exchange mechanisms helps in answering the queries such as 1) Retrieve the documents with solution architectures using MPI shared memory? 2) Search for the documents with solutions using Java RMI?

Quality Requirements: Several architectures for different systems can have the same set of functional requirements or scenarios. The degree of quality achieved may vary from system to system. The quality requirements which may vary from system to system are performance, availability, reliability etc. Some systems may compromise with some quality requirements considering the cost factors. Reliability and availability can be achieved through fault tolerance mechanisms such as using redundant components. The redundant components here may be hardware or software components. Performance is an important architectural attribute used to understand and document the resources that are required to meet the objectives. Performance specified in the corresponding service level agreement (SLA) should be measured and quantified. Some measurable criteria of performance include throughput, response time, and data accuracy. Annotating the documents with performance attributes and their values helps in improving the developing system. The process includes the following steps.

1. Get a proper set of architecture documents
2. Identify similar performance scenarios
3. Identify problem areas/bottlenecks
4. Address design and architectural performance issues including hardware and software system configuration
5. Check the tuned system whether it is feasible/not feasible.

The above process can be repeated until the desired performance levels are achieved. The queries such as 1) Retrieve the documents with throughput of the scenario between t1 and t2? 2) Search for solution with response time not less than 100ms of the required component? can be easily answered if the architecture

documents are annotated with the required quality requirements.

Runtime Issues: Software design documents do not describe the runtime behavior of a system. Architectural solution for a project sometimes can be of execution architecture with enough simulation of runtime behavior. Time criteria play a major role in the design of real time systems. The time requirements may vary for different systems such as hard/soft real time systems. Also for real time systems correctness and failure mechanisms play a crucial role. Moreover describing the interactions between the runtime elements that occur at execution time is a difficult process. The runtime elements can be processes, threads, tasks etc. Annotating the architectural documents with this runtime issues such as timelines, fail/operational cases or runtime execution components such as number of processes/threads helps the user in answering the queries such as - Search for architectural solutions of a real time system with correctness not less than t?

Miscellaneous Issues: Several other properties may also help the architect in retrieving the documents. The documents can also be annotated with success stories of the product, number of versions released, impact of product variations in the market, standards applied to the product. These properties may substantially influence the architecture of the software system.

Several queries such as

1. Search for the documents with high success factor?
2. Retrieve the solution architectures that applied IEEE international standard.
3. Search for the products with versions less than N?

Can be easily addressed by annotating the documents with such non functional properties.

5. Search for documents: Our Proposal

In this section we briefly explain our process of annotating the documents as an XML file and search for the documents based on the architecture properties. Current search facilities in the document repository are based on the title, author, and Year of the work done. User may be interested in searching for the documents with architectural properties as mentioned above. Our approach for this kind of search is as follows.

1. Annotate the documents with architectural properties as XML file as shown below.
2. Search in the XML files with architecture properties taken as input from the user.
3. Extract the title and author of the documents satisfying the Architectural properties given by the user.
4. Retrieve the documents from the repository indexed with title, author, year, etc.

The architecture properties represented as an XML DTD is shown below. Quality requirements in our DTD file are necessarily brief.

```
<!ELEMENT ArchitectureDescription(title,author,
        ArchitectureProperties)>
<!ELEMENT ArchitectureProperties (ProblemDomain,
        Scenarios,Technology,InformationExchange,
        ArchitecturePattern, Style, Tactics,
        QualityRequirements, MISC)>
<!ELEMENT Scenarios (ModScenario*,
        PerfScenario*,  SecScenario*,
        TestScenario*,  UseScenario*)>
<!ELEMENT Technology(Component+)>
<!ELEMENT InformationExchange (Memory+,
    PhysicalMedium+)>
<!ELEMENT ArchitecturePattern (pattern*,
        Antipattern*)>
<!ELEMENT Styles (CnCView+, ModuleView+,
        AllocationView+)>
<!ELEMENT Tactics (FaultDetection*,
        FaultRecovery*, LocalizeChanges*,
        ResistingAttack*, BindingTime*,
        ManageI/O*>
<!ELEMENT QualityRequirements(
        Performance*,
        Availability*,Modifiability*,Usability*)>
<!ELEMENT MISC(Standard*,Version*,Success*)>
<!ELEMENT ModScenario (#PCDATA)>
<!ELEMENT PerfScenario (#PCDATA)>
<!ELEMENT SecScenario (#PCDATA)>
<!ELEMENT TestScenario (#PCDATA)>
<!ELEMENT UseScenario (#PCDATA)>
<!ELEMENT Component (#PCDATA)>
<!ELEMENT Memory (#PCDATA)>
<!ELEMENT PhysicalMedium (#PCDATA)>
<!ELEMENT Pattern (#PCDATA)>
<!ELEMENT AntiPattern (#PCDATA)>
<!ELEMENT CnCView (#PCDATA)>
<!ELEMENT ModuleView (#PCDATA)>
<!ELEMENT AllocationView (#PCDATA)>
<!ELEMENT FaultDetection (#PCDATA)>
<!ELEMENT FaultRecovery (#PCDATA)>
<!ELEMENT LocalizeChanges (#PCDATA)>
```

```
<!ELEMENT ResistingAttack (#PCDATA)>
<!ELEMENT BindingTime (#PCDATA)>
<!ELEMENT ManageI/O (#PCDATA)>
<!ELEMENT Standard (#PCDATA)>
<!ELEMENT Version (#PCDATA)>
<!ELEMENT Success (#PCDATA)>
<!ELEMENT Conceptual Term(#PCDATA)>
<!ATTLIST Performance
        ResponseTime CDATA #Required
        Throughput   CDATA #Required
        Latency      CDATA #Required>
<!ATTLIST Availability
        FaultRate    CDATA #Required
        FailureRate  CDATA #Required>
<!ATTLIST Modifiability
        BindingTime  CDATA #Required
        ChangeCost   CDATA #Required
        RippleEffect CDATA #Required>
<!ATTLIST Usability
        TaskTime     CDATA #Required
        UserSatisfy  CDATA #Required>
```

Advantages with this approach are

- Allowing the user to search for documents with the technical details involved.
- Refining the search to the lowest level by allowing the user to search based on the multiple architectural properties at the same time.
- Annotating the documents with an XML file typically reduces the problem to an XML-tag search.

Search based on architectural properties open up several interesting issues for further exploration. We now talk about these issues.

- Analysis of user queries for relevant keywords and those keywords are to be compared with the ones associated in the XML file. Can this problem be represented as an Information retrieval problem?
- In many situations, one needs to take into account, the presence of keywords present in the XML file, synonymous or related to the query keywords. Can the semantic meaning be embedded in the XML file so that search can be performed based on semantics rather than just a simple keyword based search?

6. Online Cricket Broadcasting System: A Case Study

The ideas in this paper will be illustrated using a case study. In this section we briefly go through the online cricket broadcasting system case study to enable better understanding of the system represented in Architectural properties. The illustration of this case study can be found at http://www.cse.iitk.ac.in/~soft_arch/ocbs

The objective is to design an online cricket broadcasting system enabling flawless integration with TV, mobile, WWW. The other basic requirements include, the system should meet the streaming media needs with enriched broadcast quality. It should also support superior image stability, greater audio clarity, delivery of the messages or content to the target audience in real time. Based on the basic requirements, scenarios are divided into functional and non functional requirements.

- Functional requirements
  1. Ability to show previous match clips
  2. Frequent score update system
  3. Visual analysis of raw data with several algorithms like pitchmap, wagon wheel, Man hattan etc.
  4. Third umpire decision system

Advertisement Section

- Non Functional requirements
  1. No glitches in transmission
  2. Synchronization of audio, video.
  3. Fast search on historic database.
  4. Fast generation of charts and visual rendered images.

The module view diagram of the system is as shown below

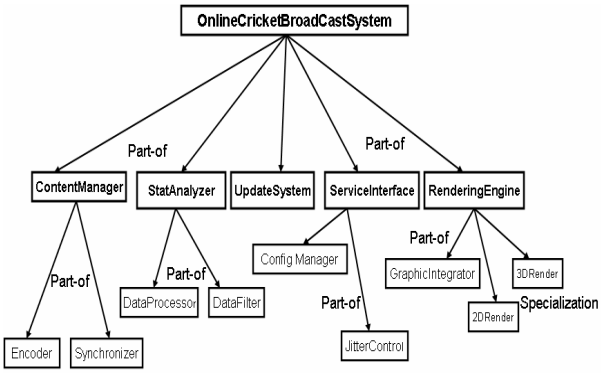

Figure 2: Module view of Cricket Broadcasting System

The main function of each module is explained below. *Update system* takes the sensor data and manual input and updates the database. The information is also passed to *statistical analyzer* which does some analysis on the score system. The *rendering engine* applies several visual analysis algorithms to generate some 3D images. The *content manager* combines the video and audio data with analysis data and images, and passes over to *Service interface* which delivers the packages to several clients.

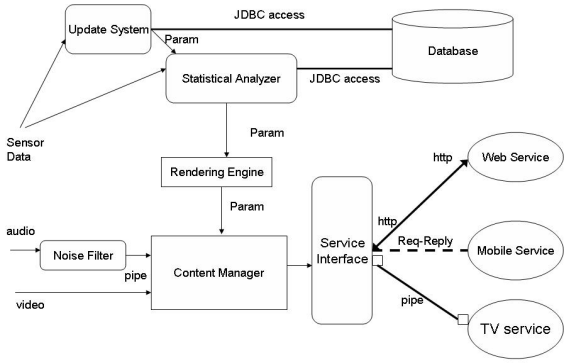

Figure 3: CnC view of Cricket Broadcasting system

The information exchange mechanisms used in this process typically include

- Parameter passing for the inner system
- Publish subscribe mechanism for the clients interacting with Service Interface.

Architectural patterns used in the system include:

1. MVC pattern for doing visual analysis.
2. Parallel pipeline pattern for streaming data.

The Architectural Tactics applied to the system are listed below.

1. Ping/echo, spare/shadow for update system in order to meet Fault detection and Recovery Quality Attribute.
2. Plug-in Module, Increase computing efficiency for statistical analyzer in order to meet modifiability and performance quality attributes.
3. Increase computing power for rendering engine in order to meet performance quality attribute.
4. Manage event rate, Bound execution time for service request handler.
5. Caching mechanism for faster data access and to get recent data in order to improve performance.
6. Abstract common service in service interface so that new service requestor can be easily integrated without many changes.

The technology components suggested for this case study include

1. Oracle database which provides rich support for triggers.
2. Data streaming management system for streaming data.
3. Webserver, DB server and App server services.
4. ODS gateway where middleware communicates with backend hosts.

The quality requirements mainly handled is the performance issue in terms of response time. The other quality attributes are availability and reliability of the system during peak times.

The XML file below encodes this information conforming to the DTD given earlier.

```
<?xml version='1.0'?>
<!DOCTYPE OCBS SYSTEM "ocbs.dtd">
<ArchitectureDescription>
<title>CricketBroadcastingSystem</title>
<author>Rambabu D</author>
<ArchitectureProperties>
<ProblemDomain> Designing an online cricket broadcasting system enabling flawless integration with TV, Mobile, Internet </ProblemDomain>
<Scenarios>
    <PerfScenario>FasterSearch </PerfScenario>
    <PerfScenario>Rendering </PerfScenario>
    <ModScenario>NoGlitch</ModScenario>
    <SecScenario>AccessControl</SecScenario>
</Scenarios>
<Technology>
        <component>Oracle</component>
        <component>DSMS</component>
        <component>Webserver</component>
</Technology>
<InformationExchange>
        <Memory>parameter passing</Memory>
        <Physical>Stream via pipe</Physical>
</InformationExchange>
<ArchitecturePattern>
        <pattern>MVC</pattern>
        <pattern>ParallelPipeline</pattern>
</ArchitecturePattern>
<Styles>
<CnCView>PublishSubscribe</CnCView>
<CnCView>ClientServer</CnCView>
</Styles>
```

```
<ArchitectureTactics>
    <FaultDetection>PingEcho </FaultDetection>
    <FaultRecovery>Spare</FaultRecovery>
    <Performance>CompEffic</Performance>
    <Performance>Caching</Performance>
</ArchitectureTactics>
<QualityRequirements>
     <Performance ResponseTime="10ms"/>
</QualityRequirements>
<Concept>Cricket</Concept>
<Concept>Wagonwheel</Concept>
<Concept>PitchMap</Concept>
<Concept>Wagonwheel</Concept>
<Concept>MobileService</Concept>
</ArchitectureProperties>
</ArchitectureDescription>
```

7. Conclusions

Though a challenge, providing a comprehensive search facility to the architecture document repository is the need of the hour. Not reinventing the wheel and using the best practices and concepts like architecture patterns, styles, tactics and antipatterns in search for documents are the time savers for architects. Annotating the documents with these architecture properties play a crucial role in this archiving process. In this paper we have presented a novel perspective on archiving architecture documents in large organizations. Central to this perspective is the annotation of the documents with architecture properties. Based on our experience with a case study, we represented the architectural properties as an XML file which provides an easy way to search for the documents. Searching for the architecture documents using semantic web concepts to improve the search efficiency is left as the future extension. We are currently investigating what kind of tool support would be necessary to adopt some of the ideas described above in practise. For future work we would like to use this tool to search for documents in industrial case studies to empirically determine the effectiveness of our method On Archiving Architecture Documents for large organizations.

8. References

[1] P.B Kruchten. The 4+1 view model of architecture. *Software, IEEE*, 12:42–50, Nov 1995.

[2] Hoffmesiter, C, Nord, R., Soni, D., Applied Software Architecture, Addison-Wesley, 2000

[3] Len Bass, Paul Clements, Rick Kazman, and Ken Bass. *Software Architecture in Practice*. 2nd Edition, Addison-Wesley

[4] Clements P, Bachmann F, Len Bass, David Garlan, James Ivers, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*.

[5] Clements P, Bachmann P, Len Bass, David Garlan, James Ivers, Robert Nord, and Judith Stafford. *A practical method for documenting software architectures*. In ICSE, 2003.

[6] IEEE recommended practise for architectural description of software-intensive systems. *IEEE Standards*, pages 1–23, Sep 2000.

[7] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, and Michael Stal. Pattern-oriented software architecture.

[8] William J. Brown, Raphael C. Malveau, Hays W. ”Skip” McCormick (III), and Thomas J. Mowbray. Antipatterns: Refactoring software architectures and projects in crisis.

[9]Clements P, Kazman R, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns.

[11] Eric M. Dashofy, Andre van der Hoek, and Richard N. Taylor. A highly-extensible, xml-based architecture
description language. In *WICSA*, 2001.

[12] IBM Software Corporation. *Rational Rose Real Time*.

[13] Sun Microsystems. http://java.sun.com/blueprints/patterns/MVC-detailed.html, 2000.

[14] Object Management Group, *Unified Modeling Language Specification Version 2.0*, Jan 2004. http://www.omg.org.

[15]xADL2.0http://www.isr.uci.edu/projects/xarchuci