A Reference Architecture for Web Servers

Ahmed E. Hassan and Richard C. Holt
Software Architecture Group (SWAG)
Dept. of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
CANADA
+1 (519) 888-4567 x 4671
{aeehassa, holt}@plg.uwaterloo.ca

ABSTRACT

A reference software architecture for a domain defines the fundamental components of the domain and the relations between them. Research has shown the benefits of having a reference architecture for product development, software reuse, and maintenance. Many mature domains, such as compilers and operating systems, have well-known reference architectures.

In this paper, we present a process to derive a reference architecture for a domain. We used this process to derive a reference architecture for web servers, which is a relatively new domain. The paper presents the mapping of this reference architecture to the architectures of three open source web servers: Apache (80KLOC), AOL-Server (164KLOC), and Jigsaw (106KLOC).

Keywords

Software architecture, reference architecture, domain architecture, web server.

1. INTRODUCTION

Research has shown the importance of having an architecture document during the development of a software system [15,17]. Such a document improves developers' system understanding. It provides a building plan for the system and reduces its maintenance cost. It provides an overview description of the system. It permits the developer to view the major subsystems in the software system and the relations between them. Unfortunately, many software systems do not have an architecture document. The cost of manually developing this document increases with the size and the complexity of the software system. Recently, a number of tools have been developed to decrease this cost by helping to extract the architecture of a software system [7, 16, 20, 21]. Using these tools, reverse engineering researchers have developed semi-automated processes to extract the product's architecture from available artifacts such as the product's source code and any available documentation.

The *reference architecture* [4] for a domain is an architecture template for all the software systems in the domain. It defines the fundamental components of the domain and the relations between these components. The architecture for a particular product is an instance of the reference architecture. During product development, the product designer refines and extends the reference architecture, based on the product's requirements and constraints. Recently, there has been some work on the derivation of reference architectures for specific domains [6]. For mature domains, the major components and the relations between them have been studied extensively and are well understood [12]. For example, an operating system is understood to have certain major subsystems such as a file system, a memory manager, a process scheduler, a network interface, and an inter-process communication subsystem [19]. Similarly, a compiler is understood to have a scanner, parser, semantic analyzer and a code generator subsystem [17]. Many of the relations between such subsystems are known and are expected to exist in the architecture of products in the same domain. For example in the architecture of a compiler, the scanner is expected to pass tokens to the parser.

Research has shown that a reference architecture enables software reuse and reduces the development efforts [9]. The different components of the reference architecture provide a template for design and code reuse. Software architecture analysis methods such as SAAM [11] use a reference architecture to evaluate alternative architectures. As new products are developed in new domains, the designers develop new sets of concepts and names. The comparison of architecture alternatives in new domains is hampered by the lack of consistency among concepts and terminology. A reference architecture provides a common nomenclature across all software systems in the same domain, this allows the architectures of a set of products to be described uniformly. Such uniformity establishes a common level of understanding and assists in comparing the different architectures. The existence of a reference architecture is useful in the reverse engineering of a software system in the domain [18], by providing a set of suggested subsystems and relations between them that can be expected to exist in the investigated system.

The web server domain is an emerging domain. The architecture of different web servers has not been studied extensively. Little has been published about the reference architecture for web servers. Luckily, three implementations of a web server are available online under an open source license: Apache [3], AOLServer [2], and Jigsaw [10]. Developed by three different organizations with distinct requirements, built using different development techniques, and with their source code available online, these web servers are prime candidates to study in the derivation of a reference architecture. Using the architecture of these three servers and a modest amount of web server domain knowledge, we developed a reference architecture for web server. We will present this reference architecture and will show how it is mapped to the architecture of each of the three web servers, to validate the derivation process.

2. THE WEB BROWSER DOMAIN

Web servers provide access to many features for users such as daily news, email service, etc. A user needs a web browser to access these features. For example, when a user using his or her browser wants to check the daily news, he or she enters the Uniform Resource Locator (URL) for the document that contains the daily news, such as *http://www.cnn.com/index.html*. Using the HyperText Transfer Protocol (HTTP), the browser in turn requests the daily news from the CNN web server. The CNN web server locates the resource that contains the daily news and sends the daily news back to the browser, which displays it to the user.

Figure 1 shows the interaction between the web server software and the rest of the environment. A web server is responsible for providing access to resources that are under control of the operating system. The most prominent web servers include Apache, StrongHold, Netscape's iPlanet, and Microsoft's IIS web server [13]. The web server provides access to resources that range from static documents, such as HTML or text, to more dynamic resources, which are created by executing programs on the web server's machine. Java servlets and Common Gateway Interface (CGI) are some of the types of programs that are executed by web servers. The clients accessing the web server are called browsers. Browsers' capabilities range from simple text oriented browser to graphically capable browsers. The most prominent browsers include Netscape Navigator, Internet Explorer, and Lynx.

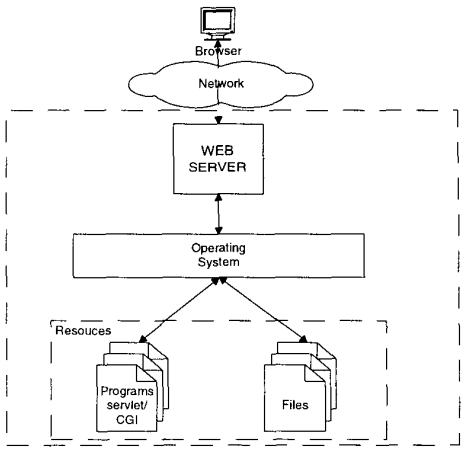

Figure 1: Web server in the network environment.

A web server can be thought of as the operating system's portal to the web. It provides a façade to the operating systems resources. It encapsulates the operating system and provides the requested resources to the browser using the functionality of the local operating system. Web servers have similar functionality, for example, all web servers can serve simple text files. But each web server may have extra features based on its design goals, for example, not all servers can serve Java servlets. The existence of a common set of features leads to the existence of a common reference architecture for web servers.

3. DERIVING A REFERENCE ARCHITECTURE

Starting from the source code of three different web servers and no architecture documentation, we derived a reference architecture for the web server domain. We are not web server domain experts, and we were not able to interview any of the developers of the systems. Instead, we used the defining artifact of these systems: their source code.

Kazman [11] describes a method to derive a reference architecture for use in conducting the SAAM analysis. Using a set of scenarios that represent the important usages of the system, a domain expert traces the implementation of the scenarios and recovers a subset of the reference architecture. Kazman [1] acknowledges that the derived architecture is an incomplete reference architecture, but it is sufficient for conducting the SAAM analysis. The derived reference architecture is limited by the quality and the quantity of the chosen scenarios.

We will now present a process for deriving a reference architecture by a non-domain expert. Given a set of implementations (such as Apache, AOLServer, and Jigsaw), some documentation for each implementation and some domain knowledge, our process for deriving the reference architecture consists of these four steps:

Step 1: Derive a conceptual (as-designed) architecture (see Figures 4, 6 & 8) for each implementation, as follows:

> ***Step 1a:*** Propose a conceptual architecture for each implementation, using domain knowledge and available documentation.
>
> ***Step 1b:*** Refine the conceptual architecture using the concrete (as implemented) architecture.

Step 2: Derive a reference architecture (see Figure 3) using the conceptual architectures derived in step 1 as follows:

> ***Step 2a:*** Propose a reference architecture based on domain knowledge and the common structure between the conceptual architectures.
>
> ***Step 2b:*** Refine the reference architecture using the conceptual architectures from step 1.

Figure 2 depicts the process used for deriving the reference architecture for web servers. Step 1 was performed for all three servers, creating their conceptual architecture.

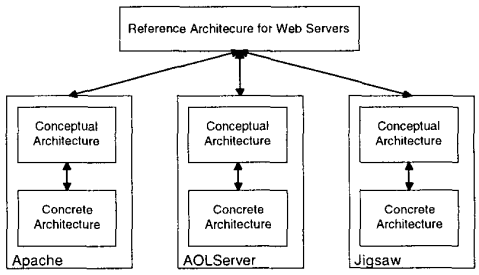

Figure 2: Reference architecture derivation process.

The conceptual architecture shows the system's subsystems and the inter subsystem relations that are meaningful to the system's developers. As we were not able to interview the developers, we built the conceptual architecture using available documentation and domain knowledge that we acquired from using web servers/browsers and installing the Apache server. We used the Portable BookShelf (PBS) tool [13] to visualize and validate each conceptual architecture. The PBS tool recovers the concrete architecture of the system from the source code of the software system. The concrete architecture shows the actual relations between the different subsystems according to the system's source code. As pointed by Bowman [5], the concrete architecture is likely to have more dependencies than the dependencies in the conceptual architecture. We examined the unexpected dependency relations and revised our conceptual architectures where appropriate. For example, if the dependency relation was due to developer's laziness (from code comments) and was not essential, it was not added to our conceptual architecture. We added relations that we missed due to features that were not documented. We did not add relations when we were not able to justify their existence. Using the revised conceptual architecture and the knowledge we gained about the web server domain, we compared the different conceptual architectures to find common components and common relations between components. Based on the commonality analysis, we inferred a reference architecture. Later we validated that each of the architectures of each web servers had a direct mapping back to our reference architecture for web servers.

4. WEB SERVER REFERENCE ARCHITECTURE

Following the process presented in the previous section, we derived the web server reference architecture, shown in Figure 3. The reference architecture follows a pipe and filter architecture style, as described by Shaw and

Garlan [17]. This reference architecture specifies the data flow and the dependency between the different subsystems. The reference architecture is composed of seven major subsystems that are divided between two layers: a *server* layer and a *support* layer. The server layer contains five subsystems that are responsible for implementing the functionality of the web server. We will now discuss the server subsystems:

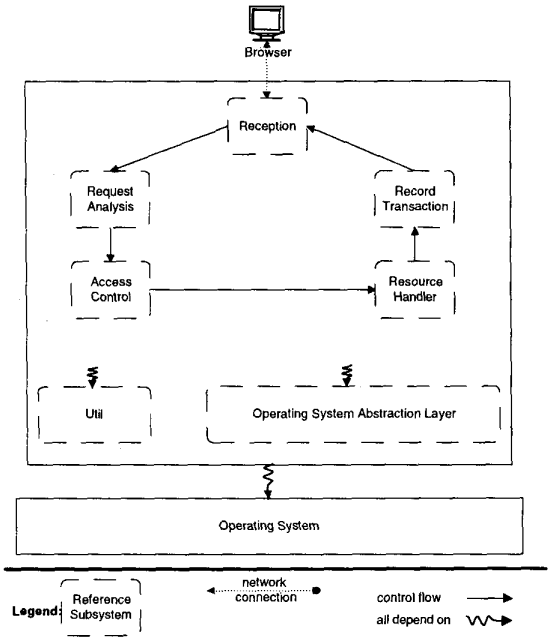

Figure 3: Web server reference architecture.

1. The *Reception* subsystem interprets the resource request protocol, such as the HTTP protocol. It is responsible for waiting for browser requests that arrive through the network, parsing the requests, and building an internal representation of the request so the other subsystems could operate on the request without any knowledge of the HTTP protocol. In addition, it determines the capabilities of the browser (such as simple text browser or graphically capable browser) and adjusts the request's response to match these capabilities. This subsystem contains the logic and the data structures needed to handle multiple browser requests simultaneously.
2. The *Request Analyzer* subsystem operates on the internal representation of the request, built by the Reception subsystem. This subsystem translates the location of the resource from a network location to local file name. For example, a request for resource *~/index.html* could be transformed to local file */usr/httpd/pub/webfiles/index.html*. Also, this subsystem may correct the spelling of the requested resource, if it cannot find an appropriate resource. For example, if the user mistyped *index.html* as *indAx.html*, the Request Analyzer subsystem could correct the typing error.
3. The *Access Control* subsystem enforces the access rules employed by the server. It authenticates the browsers and authorizes their access to the requested resources. This is the subsystem that requests a username and password to access the required resources, if needed.
4. The *Resource Handler* subsystem determines the type of the resource requested by the browser, executes it and generates the response. For example, the Resource Handler subsystem must determine if the requested resource is a static file that can be sent back directly to the user or if it is a program that must be executed to generate the response.
5. The *Transaction Log* subsystem records all the requests and their result.

The support layer contains two subsystems that provide functions that are used by the subsystems in the upper server layer:

- The *Utility* subsystem contains functions that are used by all other subsystems. It has functions for manipulating strings or URLs and many commonly used functions.
- The *Operating System Abstraction Layer* (OSAL) encapsulates the operating system specific functionality to facilitate the porting of the server to different platforms. This layer will not exist in a server that is designed to run on only one platform.

In a later section, we will map the conceptual architecture of the each of the three web servers to our derived reference architecture.

5. FLEXIBILITY OF THE REFERENCE ARCHITECTURE

To be useful a reference architecture must be flexible enough to encompass many product architectures. We will now list some of the ways in which our reference architecture is flexible.

Resource Mapping Flexibility
A web server controls access to resources that are available through the local operating system. To perform its tasks, it must map resources, which are on the operation system to resources on the web. A designer of a web server can choose different mapping methods, such as online or offline mapping. For online mapping, the mapping rules are applied to the resource as it is being served to the client. For off-line mapping, the rules are applied ahead of time and the results of the mappings are stored in a server cache. The online mapping is efficient for resources that change frequently, but this design has a higher overhead for serving resources, as the mapping operation must be performed every time the resource is requested. By specializing the design of the Resource Handle and Resource Analysis subsystems, the designer can create a web server to support these two mapping alternatives. The Apache server uses an online mapping and the Jigsaw server uses an offline mapping.

Security Flexibility
Web server designers have many choices of security models. The security model could range from simple username/password to more sophisticated models based on signed certificates. The reference architecture specifies only that an access control scheme exists but the details of the Access Control subsystem is left to the designer's discretion.

Concurrency Flexibility
For good response, web servers need to handle multiple clients simultaneously. The reference architecture does not specify how to implement this concurrency. The product designer can choose from a number of designs to achieve this concurrency using the reference architecture, by specializing the design of the Reception subsystem. A multi-threaded model (Jigsaw and AOL-Server) or a multi process model (Apache) can be used.

6. MAPPING THE CONCEPTUAL ARCHITECTURES TO THE REFERENCE ARCHITECTURE

The presented reference architecture is based on the common features and functionalities in the three examined web servers. In this section, for each of the three web servers, we provide a brief background about the serve, a conceptual architecture diagram and a conceptual to reference architecture mapping diagram. The mapping diagram shows how the architecture for the software system can be viewed as an instance of the reference architecture. In the mapping diagram, a *rounded-dotted* box is a subsystem in the reference architecture and a *square* box is a subsystem in the conceptual architecture.

6.1. Apache

The Apache server (80KLOC) is the most used web server in the Internet [13]. Apache's first release was on April 1995. This server is being developed on the Internet as an open source project. Developers are encouraged to contribute to the development of the server, but a *Core* group of developers controls the architecture of the server and the features introduced in each release. Apache's main architect is Robert Thau. The top-level architecture of the server has not changed for the past five years. Apache's development documentation indicates that no subsystems where added or removed since 1995. The main design goals for the server followed by the Core group are: speed, simplicity, support for multiple platforms and ease of distributed development. We examined the source code of release 1.3.4. All the source code of the Apache is written in C.

The conceptual architecture of Apache, shown in Figure 4, has eight major subsystems. Execution of a request starts in the Core subsystem. The browser issues its request using the HTTP protocol to the machine running Apache. The Core subsystem is always waiting for incoming requests on the machine. The Core maintains a pool of processes to support answering multiple requests for different clients. Once the request is received by the Core subsystem, a *Request_rec* structure is built. This structure stores the information needed to process the request by the other subsystems. This structure is passed to the next subsystem, the Translation subsystem. The Translation subsystem determines the local location of the requested resource. It spell checks the request and corrects it, if necessary. Next, the Authentication subsystem determines if the client requesting the resource needs to be authenticated. For example, the Authentication subsystem may ask the user for a username and password. The *Request_rec* structure is then transferred to the Authorization subsystem, which checks if the client is authorized to access the requested resource. Next, the MIME type subsystem determines the type of the requested resource. The response for the request is generated in the Response subsystem. Finally, the Logging subsystem records the

request and the Core subsystem sends the response back to the browser. During the processing of the request, if any of the subsystems encounter an error, the error is recorded in the *Request_rec* structure and the structure continues to be passed from subsystem to subsystem.

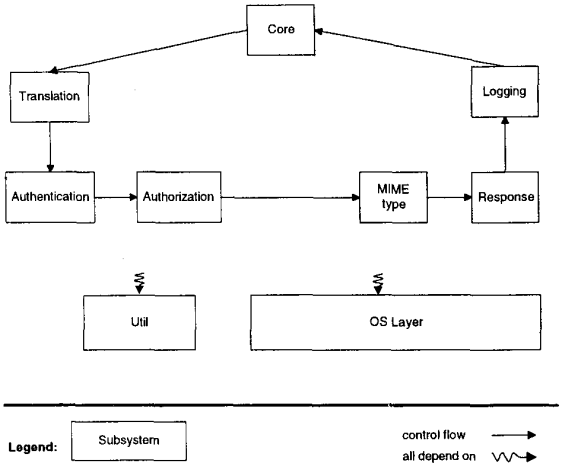

Figure 4: Conceptual architecture of Apache.

The Util subsystem contains a regular expression engine, and URL and string manipulation libraries. The OS Layer abstracts many functionalities that are operating system dependent. The OS Layer has facilitated the porting of Apache to a multitude of platforms that range from mainframes to personal computers.

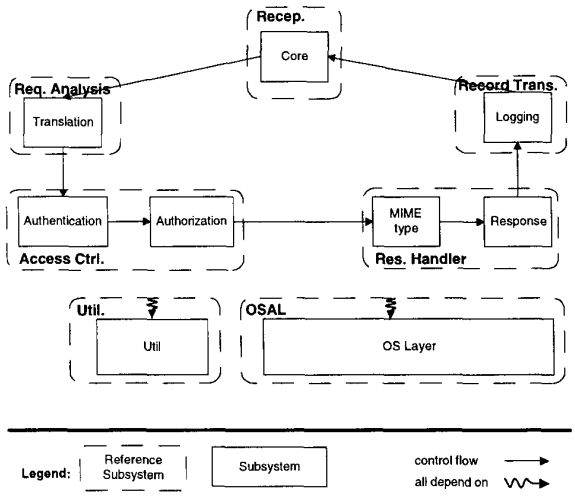

Figure 5: Conceptual to reference architecture mapping for Apache.

Figure 5 shows the conceptual to reference architecture mapping for Apache. This mapping shows a good fit between the levels of architecture. The conceptual architecture of Apache has two more major subsystems than the reference architecture. For example, the functionality of the reference Access Control subsystem is divided between two subsystems in Apache: the Authentication and Authorization subsystems. This design decision may be due to Apache's need to support distributed development. By providing finer detailed subsystems, the Apache Core group can manage the large number of developers working on the system.

6.2. AOLServer

The AOLServer (164KLOC) is a commercial web server developed by AOL. Originally, the server was developed by NaviSoft, which was bought by AOL. The NS prefix in the server's subsystem names is an abbreviation for NaviSoft. For example, NSPerm subsystem stands for NaviSoft Permission subsystem. AOLServer's first release was early 1995. The source code of the server was open-sourced in mid-June 1999. We examined the first open source release; release 3.0, which did not contain any contributions from outside sources. The architect of the server isn't known as the development of the server was closed until recently. The main design goals for the server are to provide powerful support for sites that use databases extensively, and to provide extensibility using a maintainable and safe extension language. The AOLServer uses the Tool Command Language (TCL) as the extension language. We examined the source code of release 3.0. All the source code of AOLServer is written in C, except 4 KLOC of TCL. We did not extract the TCL part of the server.

Figure 6 shows the conceptual architecture of the AOLServer, which has ten subsystems. The server contains a TCL interpreter embedded in it. As the server is geared towards sites that use databases extensively, the server contains a Database Interface subsystem that provides a façade to different types of databases. The Database Interface subsystem is used by many of the subsystems in the server. For example, the NSPerm subsystem could store access control information in a database.

The Communication Driver provides an interface that is communication protocol independent. It supports multiple network protocols such as the Secure Socket Layer (SSL), TCP sockets, and Unix sockets. The Daemon-Core subsystem translates the client's request into an internal structure, called *Conn*, that is passed to

the other subsystems. The Daemon subsystem checks if the requested resource is available. Then the NSPerm subsystem checks the permissions on the requested resource and ensures authorized access to resources on the system. The URL Handle subsystem carries out the request and generates the response. Finally, the NSLog subsystem records the processing of the request.

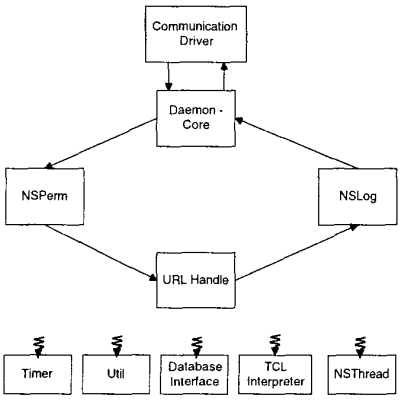

Figure 6: Conceptual architecture of AOLServer.

To facilitate the porting of AOLServer, the designers supply a thread library, NSThread, which is platform independent. AOLServer contains a Timer subsystem that permits the developer to schedule events that are executed at different time intervals. The Timer subsystem is used to timeout connections to database servers and to signal clean up for the cache structures used by AOLServer.

Some of the interesting points in the conceptual to reference architecture mapping for AOLServer, shown in Figure 7, are:

- The AOLServer design does not provide a clear separation between the Reception and Request Analysis subsystems.
- Support for multiple network protocols is important for the designers. This may be attributed to the commercial customer base of AOLServer and their needs for a server that can support multiple network protocold, such as the SSL protocol used for secure online commerce.
- The OSAL and Utility subsystems are much richer than their equivalent in Apache. OSAL provides a portable thread library implementation. Also, the Utility subsystem has a database Interface, a Timer and a full TCL language interpreter.

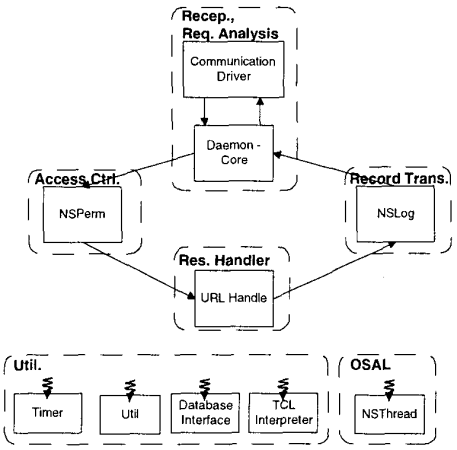

Figure 7: Conceptual to reference architecture mapping for AOLServer.

6.3. Jigsaw

The Jigsaw server (106KLOC) is an experimental server developed by the World Wide Web Consortium (W3C). Jigsaw's first release was in May 1996. The main architect of Jigsaw is Yves Lafon. The development documentation of the server indicates that the architecture of the server has not changed for the past two and a half years. The W3C uses the server for analyzing Internet protocols and standards. The server code is open sourced but its development is not as active as Apache's. We examined the source code of release 2.0.1. All the source code of Jigsaw is written in Java.

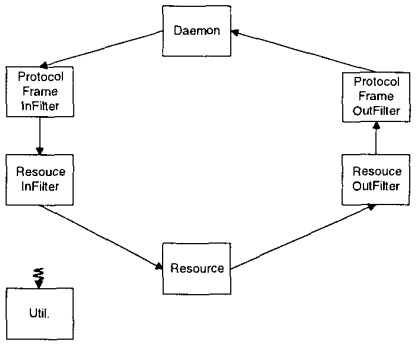

Figure 8: Conceptual architecture of Jigsaw.

The conceptual architecture of Jigsaw, shown in Figure 8, has seven subsystems. The Daemon subsystem sup-

ports various protocols used to request resources on the network. It also provides the thread pools needed to handle multiple requests concurrently. As can be seen in Figure 8, Jigsaw contains four types of filter subsystems. A browser request passes through two filter subsystems before the Resource subsystem processes the request and generates the response. The request passes through another pair of filters after the response has been generated. The choice of filters is based on the resource itself and the type of the protocol used to request the resource. Jigsaw provides different levels of filters, such as protocol and resource filters. This emphasis on filters may be due to the experimental nature of the server. These filter are useful for different types of benchmarking and for experimenting with new phases in the request processing. As with the other servers, there is a Utility subsystem that provides functionalities that are used throughout the server.

In the conceptual to reference architecture mapping for Jigsaw shown in Figure 9, we notice that an OSAL does not exist. The server is developed in the Java language, which has standard package that provides the needed platform independent layer.

Web Server	Main architect	Development type	Date of 1st release	Code size (KLOC)	Impl. language	Arch. stable for (years)	Number of conceptual subsystems
Apache	Robert Thau	Open source	April 1995	80	C	5	8
AOLServer	Unknown	Commercial	May 1995	164	C & TCL	-	10
Jigsaw	Yves Lafon	Experimental	May 1996	106	Java	2.5	7

Table 1: Statistics for the different web servers.

Reference Architecture	Apache	AOLServer	Jigsaw
Reception	Core	Communication Driver, Daemon-Core	Daemon
Request Analysis	Translation		
Access Control	Authentication, Authorization	NSPerm	Protocol Frame InFilter, Resource InFilter
Resource Handle	MIME type, Response	URL Handle	Resource
Record Transaction	Logging	NSLog	Protocol Frame OutFilter, Resource OutFilter
Util	Util	Util, DB Interface, TCL Interpreter, Timer	Util
OSAL	OS Layer	NS Thread	(Java support)

Table 2: Summary of the conceptual to reference architecture mapping.

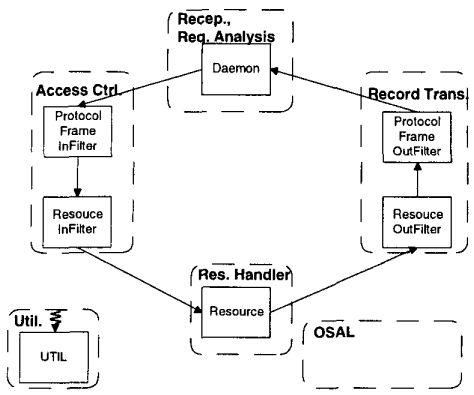

Figure 9: Conceptual to reference architecture mapping for Jigsaw.

6.4. Summary of mappings

Table 1 summarizes the characteristics of the presented web servers. They were designed and developed by separate organizations using various development techniques and languages. The diversity of the examined software systems indicates that our derived reference architecture is not biased to any development technique or organization. Examining Table 2 that summarizes the mappings from conceptual to reference architecture for each web server, we notice that the conceptual architecture of each web server fits well in the reference architecture for web servers. The main differences in structure between the reference and conceptual architectures are some splitting and merging of subsystems and differing numbers of support subsystems.

7. CONCLUSION

A reference architecture helps in system understanding. It provides a standard structure to compare different architectures in the same domain. It can be used as a framework to assist in improving system reuse, and it facilitates both forward and reverse engineering of products in the domain.

This paper has presented a process for deriving a reference architecture of a domain, by a non-domain expert. Using multiple software systems from the domain, and some domain knowledge, we derived a reference architecture for web servers. We validated the presented reference architecture using three open source systems: Apache, AOLServer, and Jigsaw. Clearly, more validation of the derived reference architecture would be beneficial and is needed. We encourage web server developers to examine our reference architecture and validate it against their servers. Finally, we hope that our presented reference and conceptual architecture for web servers will be helpful in providing a better understanding for the web server domain.

ACKNOWLEDGEMENTS

The concrete and conceptual architecture of the Apache web server is based on early work done by: Octavian Andrei Dragoi, Richard Gregory, Eric Lee, Thomas Parry, Jean Preston, Mark Scott, and Ladan Tahvildari, as part of CS 746G, a graduate computer science course offered at the University of Waterloo. The authors would like to thank Thomas Parry for his help in developing the conceptual and concrete architecture for the Jigsaw web server, and Ivan Bowman for providing a robust Java extractor.

References

[1] G. Abowd, J. Pitkow, R. Kazman, "Analyzing Differences Between Internet Information System Software Architectures", *Proceedings of ICC '96*, Dallas, TX, June 1996.

[2] The AOLServer server homepage. Available online at http://www.aolserver.com

[3] The Apache server homepage. Available online at http://www.apache.org/httpd.html

[4] J. Bergey, G. Campbell, P. Clements, S. Cohen, L. Jones, R. Krut, L. Northrop, and D. Smith. Second DoD Product Line Practice Workshop Report. Technical Report CMU/SEI-99-TR-015, Carnegie Mellon University, October 1999.

[5] Ivan T. Bowman, R. C. Holt, and Neil V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In *Proceedings of ICSE'99,* Los Angeles, May 1999.

[6] W. Eixelsberger, M. Ogris, H. Gall, and B. Bellay. Software architecture recovery of a program family. In *Proceedings of ICSE '98,* Kyoto, Japan, Apr. 1998.

[7] P. J. Finnigan, R. C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. A. Muller, J. Mylopoulos, S.

G. Perelgut, M. Stanley, and K. Wong. The software bookshelf. *IBM Systems Journal*, 36(4):564–593, October 1997.

[8] C. Gacek. Exploiting Domain Architectures in Software Reuse. in Proceedings of the ACM-SIGSOFT Symposium on Software Reusability (SSR'95), ACM Press, Seattle, WA, 28-30 April 1995, pp. 229-232.

[9] Honeywell Corporation. What are the Benefits of Using a DSSA? Available online at http://www.src.honeywell.com/projects/dssa/dssa_benefits.html

[10] The Jigsaw server homepage. Available online at http://www.w3.org/Jigsaw/

[11] R. Kazman, L. Bass, G. Abowd, M. Webb, "SAAM: A Method for Analyzing the Properties Software Architectures", *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, 81-90.

[12] R. Kazman, J. Carrière. View extraction and view fusion in architectural understanding. *Proceedings of ICSR'98,* Victoria, BX, Canada, June 1998.

[13] The Netcraft web server survey. Available online at http://www.netcraft.com/survey/.

[14] The Portable BookShelf tool. Available online at http://www.turing.cs.toronto.edu/pbs

[15] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, October 1992.

[16] M. Shaw, R DeLine, D. V. Klein, T. L. Toss and D. M. Young. Abstraction for software architecture and tools to support them. *IEEE Transactions on Software Engineering,* 21(4):314-355, Apr. 1995.

[17] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall Press, April 1996.

[18] M. Tanaun, "Software Architecture in the Business Software Domain: The Descartes Experience", In *Proceedings of ISA W3,* Orlando, 1998.

[19] S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 1992.

[20] V. Tzerpos, R. C. Holt. A hybrid process for recovering software architecture. In *Proceedings of CASCON 1996,* Toronto, Canada, Nov. 1996.

[21] K. Wong, S. R. Tilley, H. A. Müller, and M. A. D. Storey. Structural redocumentation: A case study. *IEEE Software*, 11(6): 501–520, January 1995.