Analyzing differences between Internet information system software architectures

Gregory D. Abowd, James E. Pitkow

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{abowd,pitkow}@cc.gatech.edu

Rick Kazman

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1
rnkazman@watcgl.uwaterloo.ca

Abstract: While the growth and variety of Internet information systems has been dramatic over the past five years, the methodical consideration of the differences between systems has not been emphasized. We present a particular scenario-based method for analyzing different systems in this domain based on their software architecture. We demonstrate this method, the Software Architecture Analysis Method (SAAM), by applying it to three well-known Internet information systems—WWW, WAIS and Harvest.

Keywords: Internet Information Systems, Software Architecture, Scenario-based analysis, WWW, WAIS, Harvest.

1 Introduction

The development of Internet information systems (IISs) within the past five years has been tremendous. While this growth has produced significant tools that facilitate information dissemination and retrieval, it has increased the difficulty of identifying exactly how these systems differ. In such a rapidly evolving software domain, it would be very useful to provide methodical ways to understand how various systems differ for two reasons. First, such a method would help determine which among a collection of existing systems is right for what types of information requirements. Second, a designer would have a way to predict the value of an alternative design and communicate that rationale to a larger community.

In this paper, we will demonstrate a method for analyzing the difference between a set of existing IISs by examining their software architectures. In doing this, we will provide an extensible technique for the Internet development community to communicate alternative design decisions and predict their value for users of an IIS. Though we will not provide a broad analysis of IISs (we will look at only three example systems) we will provide a structure and depth of analysis that allows for meaningful comparisons across systems.

1.1 Architectural analysis

The software architecture of a system is a high-level organization of computational elements (called components) and the interactions (called connections) between those elements. Software architecture has emerged as an important sub-field within software engineering in the past few years ([13],[19]), and our particular concern has been in the use of scenarios to analyze architectural descriptions ([9],[15]). High

level system descriptions are useful for communicating to a large audience the general behavior of a system, so it is important that these architectural descriptions be understandable and informative. In addition, designers make claims about qualities that their systems possess (e.g., scalability, portability, modifiability), but these claims are frequently vacuous because there is no way to establish their validity.

We have developed a method—the Software Architecture Analysis Method (SAAM)—which uses scenarios, or concrete descriptions of expected use of a system, as benchmarks to compare and contrast different candidate architectures. The purpose of SAAM is to provide a way to validate claims about system quality at the architectural level of description and not through vacuous appeal to abstract terms ("Our system is scalable"). SAAM relies on a description of all candidate architectures that identifies the important components and connections and the overall coordinated behavior of the system. These descriptions are then measured against an agreed set of scenarios to determine the extent to which the candidate architecture supports each scenario or must be modified in order to support it. Candidate architectures are then compared against each other by noting how they "perform" for similar scenarios.

1.2 Surveys of IISs

IISs are large network-based distributed software systems which are increasingly important in both academic and commercial computing. Some examples of these systems are the World-Wide Web, Gopher, WAIS, and Archie. Significant amounts of resources are being devoted to creating information repositories hosted by servers and clients to browse the repositories. Given the importance and virtual ubiquity of these systems, it is natural to want to compare and contrast them. Some researchers have provided overviews and comparisons of the various IISs. We will briefly review some of that previous work.

Schwartz et al. [20] provided an early taxonomy of resource discovery tools. They provided a textual description of nine different IISs and then defined four different characteristics (granularity, distribution, interconnection topology and data integration scheme) that are used to classify how a given IIS handles both data and metadata. This survey provides a good set of scenarios (browsing, searching and organizing) from the perspective of both the consumer and provider of information. The descriptions of each system is fairly informal and the level of detail provided varies between. There is also no clear connection between these descriptions and the taxonomic classification provided at the end of the paper. Our approach is not taxonomic; we do not intend to provide a description of the whole space of possible IIS designs. We do, however, want to provide a means of comparing systems. Schwartz et al. do the comparison by means of how systems differ on the general characteristics for data and metadata. Our comparison will be based on much less general characteristics, but will be performed on more consistent and formal system descriptions.

Obraczka, Danzig and Li [18] provide another taxonomic approach to describing 11 different IISs across 8 different characteristics. Their taxonomy extends that of Schwartz et al. with the addition of four yes/no characteristics (query, browse, organize, directory of services) that are used to determine if an IIS provides some capability or service. These last four characteristics are similar in flavor to the scenarios that we propose in this paper, except that we want to go further to explain how a given architecture must be modified in order to provide some capability or service. They explicitly describe the architecture of a number of these systems using the familiar box-and-line diagram approach. A brief glance at the different architectural diagrams will show that there is no common interpretation for the diagrams and it is therefore very difficult to do any analysis based on the architecture. This is more a reflection on the immaturity of software architectural description languages than a poor reflection on designers.

There are also plenty of technical reports that provide details on any single IIS, some of which we make use of in this paper. But the overriding concern with all of those papers is that they either provide too much implementation detail to be communicated effectively to a more general design audience or they do not provide enough detail (vague box-and-line diagrams) in order that a reader could establish the suitability of the design for its intended purposes. Another purpose of this paper is to provide an example of suitably high level yet sufficiently detailed architectural descriptions that can be the basis of analysis across a rapidly evolving domain such as IIS.

1.3 Overview

In the remainder of this paper we will define the SAAM method for architectural analysis of the IIS domain and present a summary of a case study of a SAAM analysis of three IIS candidates—WAIS, WWW and Harvest. In Section 2, we will define the steps involved in a SAAM analysis (scenario identification, candidate architecture description, per scenario evaluation, scenario interaction and overall evaluation). The full SAAM evaluation is too long for this paper, so we will include a representative sample of the whole analysis in the following sections. In Section 3, we will describe a subset of the scenarios used as benchmarks for the analysis. In Section 4, we will provide the architectural descriptions of the three candidate systems. In Section 5, we will show how the per scenario evaluation proceeds and what concrete rationale behind the architectures emerges. We will not show the scenario interaction for this case study, but in Section 6, we will discuss the conclusions on this case study before discussing in Section 7 our conclusions concerning the importance of this method for analyzing architectural variants in the IIS domain in general.

2 Overview of SAMM

Our previous work ([15], [6], [16]) was motivated by frequent claims in the literature of how a particular system satisfied non-functional qualities (modifiability, scalability, security, etc.) without any validation of those claims. Our concern was not only that the claims were unsupported but also that there seemed to be no clear way to demonstrate their validity. One reason for this is that most software quality attributes are too complex and amorphous to be useful for evaluation because they lack proper consideration of the context of use of a system.

This desire for context-based analysis has led us to adopt scenarios as the descriptive means of specifying and evaluating quality attributes. SAAM provides a framework to characterize how well a particular architectural design responds to the demands placed on it by a particular set of scenarios, where a scenario is a specified sequence of steps involving the use or modification of the system. It is thus easy to imagine a set of scenarios that would test what we normally call modifiability (by proposing a set of specific changes to be made to the system), security (by proposing a specific set of threat actions), performance (by proposing a specific set of usage profiles), etc.

In this section, we define the steps of a SAAM evaluation.

1. **Develop scenarios.** Develop task scenarios that illustrate the kinds of activities the system must support and the kinds of changes which it is anticipated will be made to the system over time. In developing these scenarios, it is important to capture all important uses of a system. Thus scenarios will represent tasks relevant to different roles such as: end user/customer, marketing, system administrator, maintainer, and developer. There is an important distinction between scenario types that we introduce at this point. Recall that a scenario is a brief description of some anticipated or desired use of a system. It may be the case that the system directly supports that scenario, meaning that anticipated use requires no modification to the system in order to be performed. This would usually be determined by demonstrating how the existing archi-

tecture would behave in performing the scenario (rather like a simulation of the system at the architectural level). If a scenario is not directly supported, that means that there must be some change to the system that we could represent architecturally. This change could be a change to how one or more components perform an assigned activity, the addition of a component to perform some activity, the addition of a connection between existing components, or a combination of the above. We refer to the first class of scenarios as *direct* scenarios and the second class as *indirect* scenarios. The interplay between these first two stages of the SAAM analysis is greatly aided by the identification of direct scenarios that lead to better understanding of both static and dynamic architectural descriptions.

2. **Describe candidate architecture.** The candidate architecture or architectures should be described in a syntactic architectural notation that is well-understood by the parties involved in the analysis. These architectural descriptions need to indicate the system's computation and data components, as well as all relevant connections. For our purposes, we have tended to use very simplistic architectural primitives. A typical representation will distinguish between components that are active (transform data) and passive (store data) and also depict data (passing information between components) and control (one component enabling another component to perform its function) connections. This simple lexicon provides a reasonable static representation of the architecture. Accompanying this static representation of the architecture is a description of how the system behaves over time, or a more dynamic representation of the architecture. This can take the form of a natural language specification (as in this paper) of the overall behavior or some other more formal and structured specification (as in some of our other current case studies).
3. **Perform scenario evaluations.** For each indirect task scenario, list the changes to the architecture that are necessary for it to support the scenario and estimate the cost of performing the change. A modification to the architecture means that either a new component or connection is introduced or an existing component or connection requires a change in its specification. By the end of this stage, there should be a summary table which lists all scenarios (direct and indirect). For each indirect scenarios the impact, or set of changes, that scenario has on the architecture should be described. In our experience, it is sufficient to list the existing components that must be altered and the new components and connections that must be introduced. A tabular summary is especially useful when comparing alternative architectural candidates because it provides an easy way to determine which architecture better supports a collection of scenarios.
4. **Reveal scenario interaction.** Different scenarios may necessitate changes to the same components or connections. Determining scenario interaction is a process of identifying scenarios that affect a common set of components. Scenario interaction measures the extent to which the architecture supports an appropriate separation of concerns. For each scenario determine the components or connections affected by that scenario. SAAM favors the architecture with the fewest such scenario contentions.
5. **Overall evaluation.** Finally, weight each scenario and the scenario interactions in terms of their relative importance and use that weighting to determine an overall ranking. This is a subjective process, involving all of the stake-holders in the system.

Steps 1 and 2 are highly interdependent. Deciding the appropriate level of granularity for an architecture will depend on the kinds of scenarios you wish to evaluate (though not all scenarios are appropriate, such as a code size scenario). Determining a reasonable set of scenarios also depends on the kinds of activities you expect the system to be able to perform, and that is reflected in the architecture.

Rather than offering a single architectural metric, SAAM produces a collection of small metrics (per-scenario analyses). Given this set of mini-metrics, SAAM can be used (and in fact was developed with the intent to) compare competing architectures on a per-scenario basis. It is left to the users of SAAM to determine which scenarios are most important to them, in order to resolve cases where the candidates outscore each other on different scenarios. Overall evaluation can only be derived in the context of organizational requirements.

3 Scenarios for IIS

This section defines a set of task scenarios according to the roles that they affect. These scenarios attempt to unify other scenario descriptions [Mealling, Daniels & NCSA repository requirements & Library Group at IITA] as well as contribute new tasks and clarify existing roles. Within the IIS domain, the roles of interest can be specifically named: **information consumer**, a person who uses an IIS to search for or browse information from one or a collection of data repositories; **information provider**, a person who wants to make some information available on the Internet either as a stand-alone repository or as an addition to an existing repository; and **system administrator**, a person who modifies or monitors the infrastructure of the IIS. Here we provide a name and a brief description of the scenario that will be the basis of later evaluation. Several more scenarios were done in the actual case study, but we will only do the analysis here for a subset.

3.1 Information consumer scenarios

C.1. Resource location
The user knows where a document is located and wants to be able to retrieve it, e.g. the user knows the document is located on the W3C's ftp server in /pub/www/arena/README.

C.2. Resource discovery
The user wants a particular kind of information, but does not know the information's location, e.g. a user might want to find a verified copy of Xmosaic_2.6a, or to find online publications by a certain author within the past year.

C.3. Browsing
Browsing, or serendipitous discovery ([7]), involves undirected behavior. For example, a user might want to look to see what information is available across various sites.

3.2 Information provider scenarios

P.1. Adding information of a supported type
The provider wants to include information of a supported type into an existing repository.

P.2. Adding lots of information of a supported type
An important generalization of the previous scenario is to ask how difficult it is to support many instances of the above task.

P.3. Creating new information of a type not currently supported
The provider wants to create new information of a type (FrameMaker, PhotoShop, etc.) not currently supported by this IIS.

3.3 System administrator scenarios[1]

A.1. Security from consumers

The administrator wants to prevent information consumers from corrupting the repository, i.e. causing accidental or intentional changes to the repository that compromise the integrity of the information from an external consumer's perspective.

A.2. Security from providers

The administrator wants to prevent information providers from corrupting the repository in the same manner as describe above.

4 Candidate architectures

We will now present an architectural description of three IISs, the WWW, WAIS, and Harvest. The architectural description consists of a modified data flow diagram of the system, indicating gross-level computational and data components[2], and the control and data connections between them, together with a textual description of the overall behavior of the system. Though this is not a fully formal definition, care was taken to provide an accurate representation of each system at a similar level of detail. In creating these descriptions, the authors referred to recent literature, inspected code, and consulted IIS domain experts.

The descriptions distinguish between data and control connections, even though these are frequently implemented by the same mechanism, e.g. a procedure call. A data connection between two components indicates that tokens of data are passed between these components. A control connection between two components indicates that one component causes the other to perform some of its main computation. For example, when one executes a procedure call in a standard procedural language, one typically passes data tokens as the arguments and return value of the procedure call and a "life" token that enables the called procedure to become active, with the calling procedure being suspended until the called procedure returns the "life" token. Some connection types involve only a single type of connection. For example, TCP/IP connections are pure data: neither procedure can force the other procedure to become active or inactive by sending or not sending data. On the other hand, a Unix "virtual fork", is mainly a control connection.

4.1 WWW

The Web is a distributed hypermedia system organized as a loosely connected set of clients and servers that share a common set of communication protocols and markup languages. Servers make Internet resources available to a community of clients that speak a common protocol. In addition, WWW clients typically understand a number of other protocols ([2],[3],[4],[12]). Figure 1 depicts the WWW software architecture. In this example, we only consider the connection between a WWW client and WWW server, even though it is possible for a WWW client to natively speak to other IISs (Gopher, FTP, etc.).[3]

WWW clients provide a User Interface Manager.[4] This manager can be graphical or character based. The User Interface Manager captures user's request for information retrieval in the form of a Uniform Re-

1. These scenarios do not refer to encryption, traffic analysis, or any other information content based notion of security. Rather, the security scenarios listed deal directly with the architectures vulnerability to various roles of intruder.
2. We also use boxes in the diagrams to delineate important system boundaries, such as the client-server distinction.
3. The use of the Common Gateway Interface (CGI) provides another way for a WWW server to simulate other access protocols while still communicating with the WWW client using HTTP. This ability does not reduce the overall functional capability of WWW system presented.

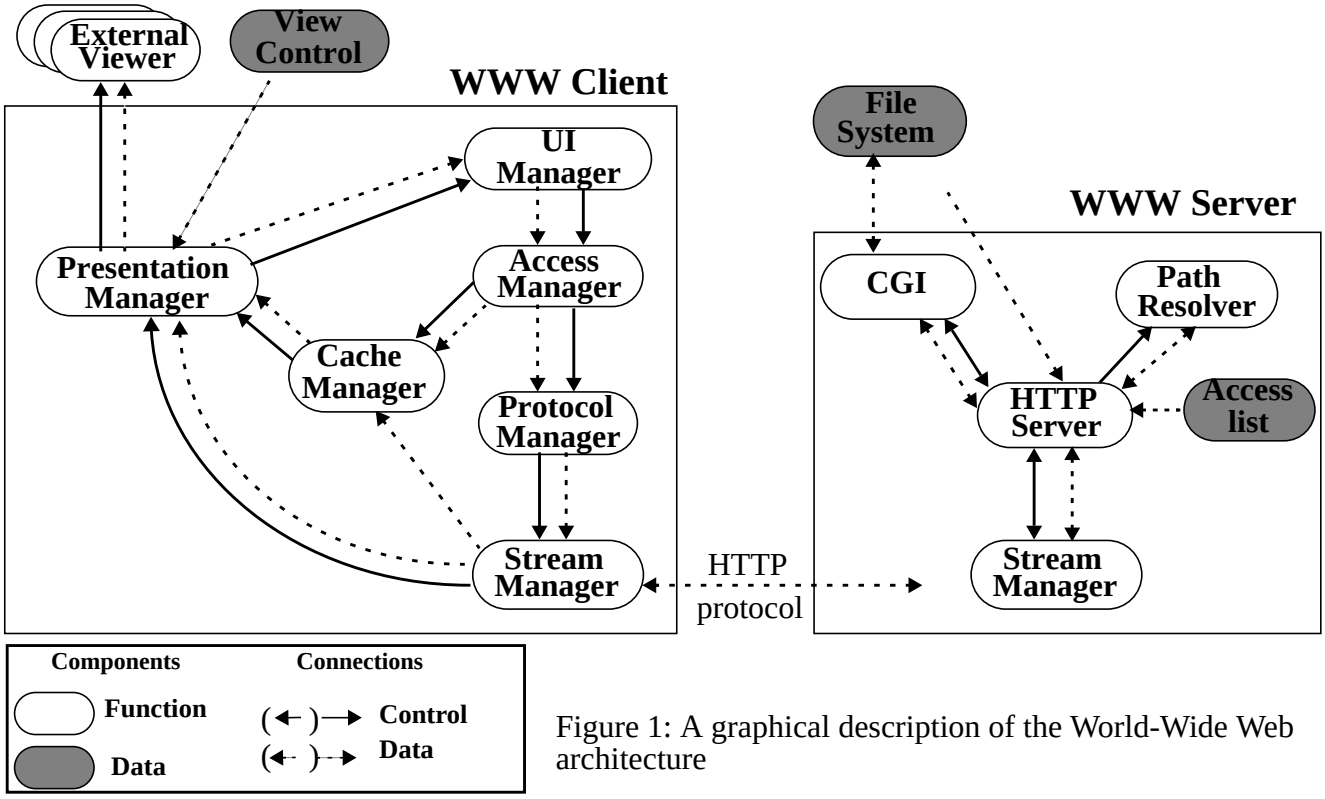

Figure 1: A graphical description of the World-Wide Web architecture

source Locator (URL) and passes the information to the Access Manager. The Access Manager determines if the requested URL exists in cache and also interprets history-based navigation, e.g. ‘back’. If the file is cached, it is retrieved from the Cache Manager and passed to the Presentation Manager for display to either the User Interface or an external viewer. If the file is not cached, the Protocol Manager determines the type of request and invokes the appropriate protocol suite to service the request. This protocol is used by the client Stream Manager for communicating the request to the server. Once the client Stream Manager receives a response from the server in the form of a document, this information is passed to the Presentation Manager for appropriate display. The Presentation Manager consults a static View Control configuration file (mimerc, mailcap, etc.) that aids in the mapping of document types to external viewers.

Currently, the WWW servers available implement a subset of defined HTTP requests. This subset allows for: the retrieval of documents; the retrieval of document meta-information; and server-side program execution via the Common Gateway Interface (CGI).

When a request is received by the server Stream Manager, the type of request is determined and the path of the URL is resolved via the Path Resolver. The HTTP Server consults an Access List to determine if the requesting client is authorized to access the data pointed to by the URL. The HTTP Server might initiate a password authentication session with the client to permit access to secured data. Assuming authentication succeeds, the HTTP Server accesses the File System (which is outside the WWW Server boundary) and writes the requested information to stream. If a program is to be executed, a process is made available (either new or polled) through the CGI and the program is executed, with the output written by the server Stream Manager back to the WWW Client.

4. In this description and later descriptions of WAIS and Harvest, we use capitalization to refer to components in the architectural description.

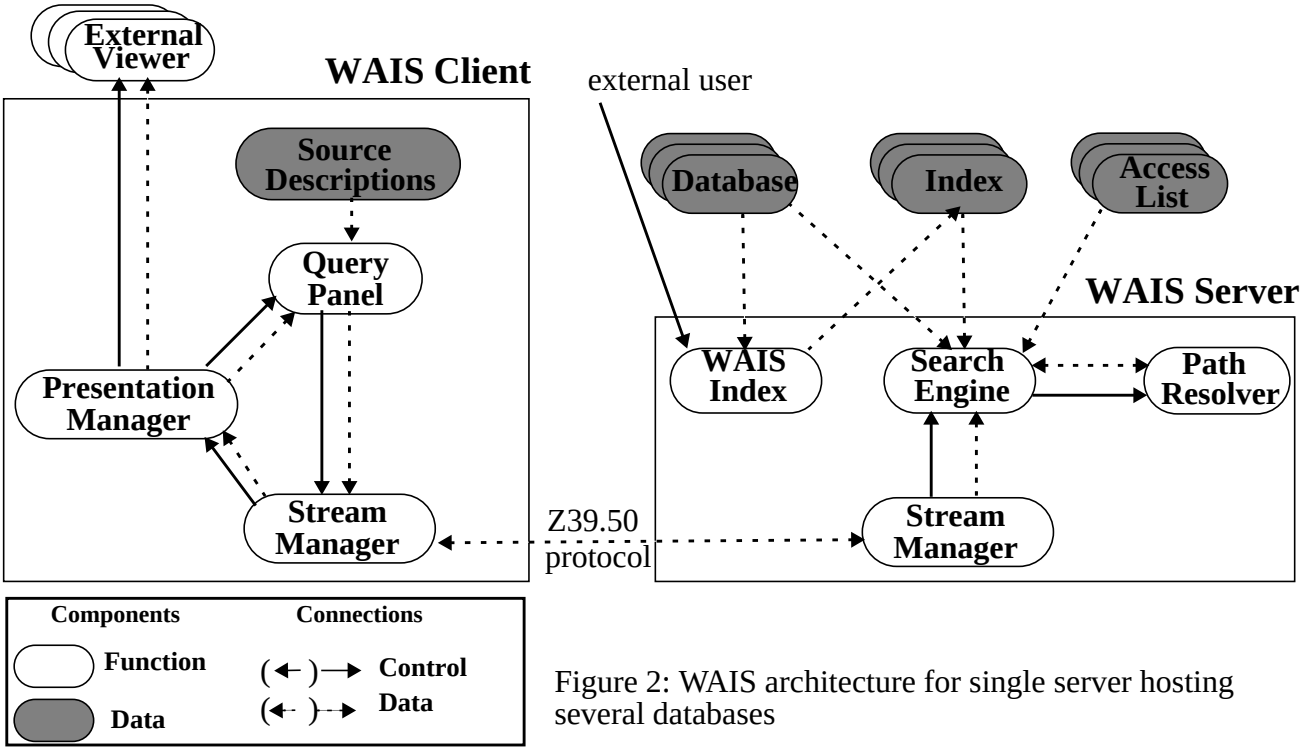

Figure 2: WAIS architecture for single server hosting several databases

4.2 WAIS

Wide Area Information Server (WAIS)[5] is a network publishing and retrieval system designed to help users find distributed networked information using a mixture of natural language and boolean queries ([8], [21],[22]). Communication between client and server adheres to the NISO standard protocol Z39.50. Searching for information via the WAIS client-server model is a two step process. The directory of services—a centralized index of all registered WAIS servers—is queried to locate potential databases that contain information on a user-specified topic. The user then selects a subset of these potential databases and formulates a query to locate specific relevant documents. WAIS ranks all returned hits for a query, indicating which documents most closely meet the user's query. WAIS allows for an iterative search strategy in which a user can indicate that subsequent searches on a database should look for files that most nearly match some previously identified file. Once the user finds a document of interest, a request can be made to retrieve that document from its database. The architecture of WAIS is depicted in Figure 2.

The WAIS client provides the user interface as a Query Panel that is used to formulate a query and indicate the data sources (WAIS databases, including the Directory of Services) that the user wishes to search. The Query Panel also displays the ranked results of queries. Each Database provides descriptive information including its location, name and summary of contents which is accessible through the consumer-browsable Source Descriptions file. This information is used to establish the connection from the client to the server. After the user formulates a query, it is passed to the client's Stream Manager, which handles communica-

5. WAIS and Wide Area Information Servers are both registered trademarks of WAIS Inc. There are public domain versions of WAIS, called freeWAIS and ZDist. For the purposes of this paper, the differences between these systems is not significant. Most of the technical documentation for freeWAIS has been consulted in our analysis, but we will still refer to the system as WAIS.

tion with the WAIS Server. Information returned to the client's Stream Manager is then passed to the Presentation Manager, which either sends the query result to the Query Panel or directs the document information to an External Viewer, thus enabling the viewing of various document formats.

The WAIS Server services the requests from the WAIS Client. Two types of requests are serviced: queries and document retrievals. Each server can host more than one database. Each database has an associated inverted keyword Index and Access List. This Index facilitates the matching of queries to documents. The Access List identifies those clients with permission to access the databases. For a query, the Search Engine determines which documents are relevant for the search using the Index and Access List. It then retrieves the documents from the Database itself and ranks the documents according to some relevance criteria based on the user's query.[6] The Search Engine then writes to stream an ordered list of documents with header information (including relevance to the original query) to the client for display to the user. For document retrieval, the client sends a request to the server, the Search Engine checks access privileges and retrieves the file if access by the client is allowed, compresses the data and writes it to stream.

4.3 Harvest

Harvest is a resource discovery tool which leverages off of much of the functionality of the Web [6]. Unlike the Web, it attempts to provide mechanisms to facilitate global resource discovery amongst heterogenous Internet based services like FTP, Gopher, UseNet News, & the WWW. While Harvest does not introduce any new client technologies (WWW Browsers are the default Harvest client) and many components speak existing protocols (HTTP) and generate existing markup (HTML), Harvest does provide an architecture for global search and retrieval across existing network-based, information providers.

The architecture of Harvest (shown in Figure 3) which extends WWW is based up the notion of information "Gatherers" and "Brokers." A Gatherer resides either in an existing repository server (e.g., a WWW server or WAIS server) or interacts remotely via a network protocol (currently HTTP) and collects summaries of metadata for various Brokers to store. This collection is done by the Typer recognizing types of information in the repository and then the Summarizer extracting useful content metadata. Gatherers and Brokers use a specialized protocol (SOIF) for efficient communication and storage of metadata. Information from one Broker to another is replicated in a hierarchical manner, this organization being maintained by a Replication Manager (not shown). Periodically, the Collector component in the Broker will request updates of metadata from each Gatherer or Broker it services. Any updates are passed to the Registry, which records unique object identifiers and time-to-live values for the data. Identifiers are provided to the IR/Search Engine and the Store Manager is requested to archive a copy of summary metadata in the Metadata DB on the file system for later retrieval by the IR/Search Engine.

The Broker responds to queries from clients (currently only WWW clients) by returning the location (URLs) of the items of interest. The query is accepted by the Query Manager, which then translates the request into the Harvest Query Language (HQL) and passes it onto the IR/Search Engine. The IR/Search Engine then provides a response to the query by accessing the Metadata DB according to its own searching strategy.

---

6. Though the WAIS documentation suggests that a WAIS query can span across several Databases at different sites, we do not see in the documentation of this system any indication that the relevance ratings span across sites, that is, the individual WAIS Servers all produce relevance criteria for information in their databases and no attempt is made in the WAIS Client to do further ranking.

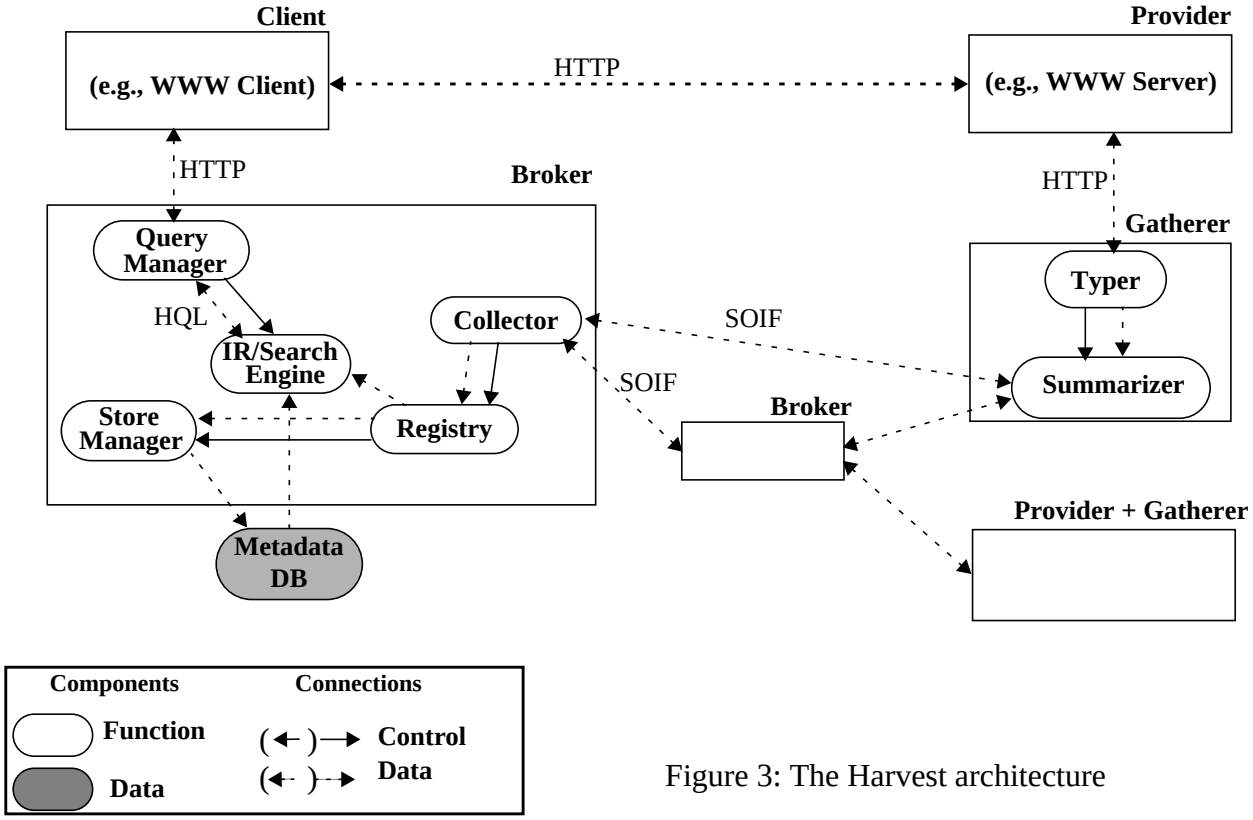

Figure 3: The Harvest architecture

5 Scenario evaluations

In this section, we will evaluate the three candidate scenarios against each of the consumer and provider scenarios defined in Section 3. For reasons of space, we will not discuss the administrator scenarios here.

5.1 Information consumer scenarios

C.1. Resource location

WWW: The user can explicitly communicate the known URL via the User Interface to the Access Manager. This scenario, therefore, is directly supported.

WAIS: The notion of a universal name does not exist with WAIS, therefore the user would have to know the Database where the information resides as well as the handle to the file that contained the information. This scenario, therefore is only indirectly supported by the architecture. In order to support this scenario, the Query Panel would have to be modified to allow sending explicit document retrieval directives without first having to perform a search.

Harvest: Since Harvest is currently built on top of WWW clients, this scenario is directly supported.

C.2. Resource discovery

WWW: The current Web implementation does not directly enable users to find information on a topic. It is possible for WWW to provide other protocol services, including WAIS, either via CGI or the common library of protocols.

WAIS: WAIS supports a two-phase search strategy which allows a narrowing of the search to a set of potential databases and then an actual search of those databases. The effectiveness of this two-pass strategy depends on the accuracy of meta-information stored in the directory of services. We conclude that WAIS directly supports this scenario.

Harvest: The arrangement of topic-specific Gatherers and Brokers allows Harvest to directly support this scenario.

C.3. Browsing

WWW: Browsing is enabled to the extent that the User Interface component makes available locations of information (ar URLs) at various places on the Web. WWW supports this scenario directly, though there is no support for organized browsing.

WAIS: A limited form of browsing is possible in WAIS. The Source Descriptions file is accessible from the Query Panel. But this information is restricted to a particular site's repository. It is also possible to view a catalog on each server, but this is still more restrictive than the scenario suggests and cannot be explicitly requested by the Query Panel. Any form of browsing to be supported by WAIS would have to modify the Query Panel and the Search Engine.

Harvest: This scenario is direct for the same reasons as WWW. We note that it is not currently possible to browse metadata in the Brokers, but the scenario was not referring to that capability.

5.2 Information provider scenarios

P.1. Adding information of a supported type

WWW: The Web servers attempt to be a MIME Content Type File System, so any existing MIME typed digital information can be made accessible. Permission to write new information to the File System is the only other provider requirement. Once that information is in the File System, it can instantly be accessed. This scenario is directly supported by WWW.

WAIS: The repositories are the Databases that the Server accesses. These are typed, allowing WAIS to perform its indexing, query response and text retrieval more efficiently. The entries in the databases are stored as files in the underlying file system, so there is no real difficulty introducing a new piece of information (again, assuming the provider has correct file system privileges). Even though the Index must be updated to make the new data available to the consumer, this scenario is directly supported.

Harvest: The information can be added to the repository, just as with WWW. The Gatherer must then detect the presence of the new information (either by automatic polling or explicit invocation) to extract the SOIF summary. When the Broker next requests updates from the Gatherer, this new information will be passed along and made available to a consumer. This scenario is direct, though we now have two points of delay in making information available to the user.new information has to be gathered and then forwarded on upon request by an interested Broker.[7]

7. Both WAIS and Harvest introduce delays in making new information available to the consumer. Though immediate availability of information is an important feature of an IIS, it is not a matter considered by the P.1 scenario, and so we ignore it in our overall ranking of the candidates for this scenario.

P.2. Adding lots of information of a supported type to a single repository

WWW: The architecture indirectly supports this scenario. Efficient support of this scenario hinges on how well the underlying operating system supports adding numerous new files. Modifications would be required to the File System.

WAIS: Adding lots of information to the Database is similar to the WWW case, depending on the operating system for efficiency. The Index allows for incremental updating, meaning that the whole Index does not have to be recomputed after every addition. What is not clear from the technical documentation on the Index is how efficient the incremental indexing is over time. We therefore consider this scenario indirectly supported with potential bottlenecks at the Database and Index.

Harvest: Similar to WWW with added bottlenecks at the Gatherer and Broker. If Gatherer is local to repository, then efficient automated gathering can occur. If Gatherer is remote, then there is a network bottleneck. Broker-Gatherer communication depends on performance of high-level SOIF protocol, which is communicated in compressed form to speed transmission. There is an additional step involved here which is equivalent to the WAIS step for updating the Metadata DB for use by the IR/Search Engine. The efficiency of that step depends on the IR/Search Engine in place (which could be WAIS Search Engine).

P.3. Creating new data repository of a type not currently supported

WWW: Since WWW servers attempt to use MIME Content Types, the addition of an unregistered type would result in no header information being passed to the Client, unless the HTTP Server were modified. If the client was unable to type the file, it would default to ‘application/octet-stream’. If the HTTP Server adds a new MIME Content Type that the Client does not understand, the behavior of the client is undefined, unless the View Control is modified and an External Viewer application is provided.

WAIS: Modifying WAIS to support this task would require changing the Index because it currently needs to have the format of new types hard coded as a parsing routine.

Harvest: Harvest relies on having a SOIF format for data in order to operate effectively. For new types of information, the provider can use the Essence system [14] to change how the Typer recognizes data types and how the Summarizer extracts metadata. Since Essence is used at the core of defining the behavior of the Typer and Summarizer, we consider this scenario directly supported.

6 Overall evaluation

The results of the scenario evaluations are given in Table 1. The final three columns indicate the extent of

Table 1: Summary of evaluation of WWW and WAIS

Scenario	**Rank**	**Changes/additions necessary**		
		WWW	**WAIS**	**Harvest**
Information consumer				
C1. Resource location	{H,W},A	direct	Query panel	direct
C2. Resource discovery	{A,H},W	Protocol Manager CGI	direct	direct
C3. Browsing	{H,W},A	direct	Query Panel Search Engine	direct
Information provider				
P1. Adding information of supported type	{A,H,W}	direct	direct	direct
P2. Adding lots of information of supported type	W,A,H	File System	Database WAIS Index	Provider repository Gatherer Broker-Gatherer SOIF Broker
P3. Creating new data of a type currently unsupported	H,A,W	HTTP Server View Control External Viewer	Database WAIS Index	direct

architectural modification that is required for indirect scenarios. The second column provides a relative ranking for that scenario (curly brackets {} indicate a tie, W=WWW, A=WAIS, H=Harvest).This table can be used to reveal the scenario interactions discussed as Step 4 in Section 2, but we will not discuss that here. We also do not consider weighting the scenarios in this paper, as that depends on the intended use of the IIS, which can vary based on consumer, provider and administrator preferences. Given a relative ranking of scenarios, however, it should be clear how one could use the summary analysis in Table 1 to justify choosing one IIS over another.

7 Conclusions

We have described a Software Architecture Analysis Method (SAAM) that provides a scenario-based analysis technique for comparing different software architectures within the same domain, and we demonstrated this by comparing three different IISs—WWW, WAIS and Harvest. Whereas some of the conclusions of our analysis might be debatable, we would like to focus attention on the value of the process behind the evaluation, because that offers the most promise to the growing community of IIS users and developers. We have provided a procedure and first attempt toward understanding the differences between systems in this rapidly evolving domain. The results of SAAM can be used to choose between various IISs (for the consumer). to establish rationale for proposed changes to the general IIS infrastructure and to document the rapid evolution of software architecture for IISs.

The majority of the benefit of SAAM is people-oriented. We are forcing an analysis across a variety of systems using a common architectural representation. Our experience in other case studies has shown that this helps to reinforce a common understanding within a larger community and is the first step toward developing canonical representations. That we are describing systems at the architectural level is important for highlighting major structural or behavioral differences. But we acknowledge that significant differences between systems exist at more detailed levels of description and these can be just as important as the high level decomposition.

SAAM is a scenario-based method for analyzing software architectures. We have demonstrated an evaluative use of scenarios on existing systems to understand their differences. It has long been recognized that scenarios are an effective way to capture requirements before proceeding to later stages of design. Scenarios can therefore be the basis of more generative design activities, as developers consider changes to the basic infrastructure or architecture of next-generation IISs. But whether the use of scenarios is generative or evaluative, we are here demonstrating the importance of establishing the important scenarios for IIS analysis. In this paper, we defined eight different possible scenarios from three different perspectives. This list is in no way complete, nor do we even claim that it represents the most important concerns of the IIS community. There has been activity in the IIS community to document scenarios (see, for example, the draft report on URC scenarios [10]), and that activity would be the best input for improving the relevance of this SAAM analysis.

Expertise in the domain of IIS (in our case provided by Pitkow and lots of publicly available documentation) was vital to the success of this case study, as was expertise in describing and analyzing software architecture. Without IIS expertise, many of the scenarios in this case study would have gone unanswered. The importance of SAAM is that it forces the IIS evaluator to answer many questions about the architectural design and potential changes to that design that would otherwise go undeclared.

8 Acknowledgments

The authors would like to thank Len Bass and Paul Clements of the Software Engineering Institute and Rob Allen of the Computer Science Department at Carnegie Mellon University and various anonymous referees for their comments on earlier drafts of this paper.

9 References

[1] G. Abowd and L. Bass, “Software Architecture: A tutorial introduction”, Tutorial presented at 1995 Motorola Software Engineering Symposium, Ft. Lauderdale, FL, June 28, 1995. Copies available from authors upon request.

[2] T. Berners-Lee, R. Cailliau, J.-F. Groff, “The World-Wide Web”, *Computer Networks and ISDN Systems*, **25**, North-Holland, 1992, pp. 454-459.

[3] T. Berners-Lee, R. Cailliau, J. Groff, B. Pollermann, “World-Wide Web: The Information Universe”, *Electronic Networking: Research, Applications and Policy*, 2(1), Meckler Publishing, 1992, pp. 52-58.

[4] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen, and A. Secret, “The World-Wide Web”, *Communications of the ACM*, 37(8), 1994, pp. 76-82.

[5] C. M. Bowman, P. Danzig, U. Manber, M. Schwartz, “Scalable Internet Resource Discovery: Research Problems and Approaches”, *Communications of the ACM*, 37(8), 1994, pp. 98-107.

[6] C. M. Bowman, P. B. Danzig, D.R. Hardy, U. Manber, M. F. Schwartz and D. P. Wessels, “Harvest: A scalable, customizable discovery and access system”, Department of Computer Science, University of Colorado, Technical Report

CU-CS-732-94. March, 1995 revision of original August 1994 draft.

[7] L. Catledge and J. Pitkow, “Characterizing Browsing Behaviors on the World-Wide Web”, *Computer Networks and ISDN Systems*, **27**, North-Holland, 1995 (in press).

[8] The Center for Networked Information Discovery and Retrieval, “freeWAIS Technical Documentation”, On-line documentation. URL ftp://ftp.cnidr.org/pub/NIDR.tools/freewais/.

[9] P. Clements, L. Bass, R. Kazman, G. Abowd, “Predicting Software Quality by Architecture-Level Evaluation”, In the proceedings of the *International Conference on Software Quality*, Austin, TX, October 1995. To appear.

[10] R. Daniel and M. Mealling, “URC scenarios and requirements”, IETF Internet Draft report, June 27, 1995. URL http://www.acl.lanl.gov/URI/Scenarios/.

[11] H. Frystyk, “WWW Library Internals—Overview”, On-line documentation, URL http://info.cern.ch/hypertext/WWW/Library/User/Guide/Overview.html.

[12] H. Frystyk, H. Lie, “Towards a Uniform Library of Common Code: A Presentation of the CERN World-Wide Web Library”, *Proceedings of the Second International World-Wide Web Conference*, 1994.

[13] D. Garlan and M. Shaw, An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, Volume I, World Scientific Publishing, 1993.

[14] D. R. Hardy and M. F. Schwartz, “Customized information extraction as a basis for resource discovery”, Department of Computer Science, University of Colorado, Technical Report CU-CS-707-94, March 1994.

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

[16] R. McCool, “The Common Gateway Interface”, On-line documentation, URL http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[17] B. Neuman, “The Virtual System Model: A Scalable Approach to Organizing Large Systems”, PhD Thesis, University of Washington, 1992.

[18] K. Obraczka, P. Danzig, and S. Li, “Internet Resource Discovery Services”, *IEEE Computer*, **26**(9), 1993, pp. 8-22.

[19] D. Perry and A. Wolf. “Foundations for the study of software architecture”, *SIGSOFT Software Engineering Notes*, **17**(4), October 1992, pp. 40-52.

[20] M. Schwartz, A. Emtage, B. Kahle, B. Neuman, “A Comparison of Internet Resource Discovery Approaches”, *Computing Systems*, **5**(4), 1992, pp. 461-493.

[21] WAIS, Inc., “WAIS Server, WAIS Workstation, WAIS Forwarder for Unix: Technical Description”, Technical Description, Release 1.1, 1993.

[22] WAIS, Inc., “WAIS for UNIX”, On-line documentation, URL http://www.wais.com/company/Tech_TOC.html.