MORALE METHODOLOGY GUIDEBOOK
Software Architecture Analysis Method (SAAM)

Gregory D. Abowd
College of Computing
Georgia Institute of Technology
abowd@cc.gatech.edu

Len Bass, Paul Clements and Rick Kazman
Software Engineering Institute
Carnegie Mellon University
firstname.lastname@sei.cmu.edu

1. Introduction

1.1 Purpose

Because of the importance of architectural decisions, it is fitting that they receive close scrutiny. In particular, few will argue that it is more cost-effective to evaluate software quality as early as possible in the life cycle. If problems are found early in the software life cycle, they are trivial to correct—a change to a requirement, or specification, or design is all that is necessary. Software quality cannot be appended late in a project; it must be inherent from the beginning, built in by design. It is in the project's best interest for prospective design candidates to be evaluated (and rejected, if necessary) during the design phase, before long-term institutionalization occurs. Therefore, an effective technique to assess a candidate architecture—*before* it becomes the project's accepted blueprint—is of great economic value. Furthermore, when *acquiring* or *evolving* a large software system that will have a long lifetime within the acquiring organization, it is important that an organization develop an understanding of the underlying architecture of the candidates. This allows an assessment of the suitability of the candidates with respect to qualities of importance.

On the other hand, it should be noted that an architecture cannot guarantee the functionality or quality required of a system. Poor downstream design, implementation, testing, or management decisions can always undermine an acceptable architectural framework. Decisions at all stages of the life cycle—from high level design to coding and implementation—will affect quality. Therefore, quality cannot be completely assessed on the basis of an architectural description. An architecture-based quality assessment provides only one dimension of a system's quality characteristics and is a necessary, but not sufficient, component of evaluating the overall quality of a system. It assesses the ability of the architecture to facilitate the desired qualities. Refinements of the architecture into implementation that preserve the qualities are necessary for the final product to actually achieve those qualities.

We present a method, the Software Architecture Analysis Method (SAAM), for predicting system level quality attributes based upon software architectural evaluation. SAAM makes use of scenarios derived from the different roles involved in a computer system (administrator, user, maintainer, etc.) in order to evaluate competing architectures. We illustrate SAAM in the context of a simple but classic software architecture example, a key-word-in-context system. We then demonstrate this method in a larger case study. We present an architectural analysis of three well-known Internet information systems—WWW, WAIS and Harvest

1.2 The overall MORALE architecture

(common across guidebooks)

1.3 Overview of method

Inputs to and outputs from the evaluation:

If the architecture being evaluated is not yet instantiated in a system—as ideally, it should not yet be—then what is the basis for the analysis? Obviously a description or specification of the architecture is required. Since, as we have seen, architectures may be described in many levels of detail and from many views, what kind of description or specification is needed for evaluation? The answer is whatever is needed to answer the questions imposed by the evaluation technique. If the evaluation will delve into issues of performance or parallelism, than a description of the architecture's tasking and communication structure will probably be required. If modifiability is an issue, then the architecture's decomposition into modules or work assignments will be needed. And so forth. SAAM starts by using whatever rendition of the architecture is available, and proceeds to make whatever evaluation it can based on that. If other views or finer-grained depictions of the architecture are needed, SAAM will make their need clear as it progresses.

What does an evaluation produce? As we shall see in the next chapter, results vary across methods. SAAM produces a relative ranking. That is, if two or more candidate architectures are being compared to see which one satisfies its quality requirements more fully, SAAM will produce a ranking of the candidates. If a single architecture is being evaluated, SAAM will point out places where that architecture fails to meet its quality requirements, and will in some cases show obvious alternative designs that would fare better. The results are coarse and broad; this reflects the variance inherent in performing architectural analysis at a very coarse grain.

In no case are absolute numbers on some mythical scale of "architecture excellence" produced. SAAM is predicated on the principle that there is no such scale; an architecture's suitability depends entirely on the specifics of the context in which it is being developed.

The coarse analysis provided by SAAM is justified by its low cost and the ability to achieve it very early in the software development life cycle.

Analyzing Qualities with Scenarios:

People often want to analyze software architectures with respect to quality attributes expressed using words like *maintainability, security, performance, reliability*, and so forth. These words provide convenient ways for describing and communicating a host of common, recurring problems in software. There are even IEEE standards that define many of them. However, most software quality attributes are too complex and amorphous to be evaluated on a simple scale, in spite of our persistence in describing them that way. Consider the following examples:

Suppose a system can change the background color of the user interface merely by changing a data file, but that same system requires a manual change to dozens of programs in order to accommodate a new data storage layout. Would you say that this system is or is not *modifiable*?

Suppose the user interface to a system is carefully thought out so that a novice user can exercise the system with a minimum of training, but the experienced user finds it so tedious as to be inhibiting. Would you say that this system is *usable* or not?

The point, of course, is that quality attributes do not exist in isolation, but rather only have meaning within a context. A system is modifiable (or not) with respect to certain classes of changes, secure (or not) with respect to specific threats, usable (or not) with respect to specific user classes, efficient (or not) with respect to its utilization of specific resources, and so forth. Statements such as "This system is highly maintainable" turn out to be entirely without operational meaning.

This notion of context-based evaluation of quality attributes has led us to adopt *scenarios* as the descriptive means of specifying and evaluating quality attributes. SAAM is a scenario-based method for analyzing architectures; it provides a means 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 that tax resources), etc.

A particular scenario actually serves as a representative for an entire class of scenarios. This class consists of all scenarios for which the system under consideration responds in exactly the same way. For example, the scenario "change the background color on all windows to blue" is under most system designs essentially equivalent to the scenario "change the window border decorations on all windows". SAAM measures, among other things, how many evaluation scenarios cause changes to the same architectural component; this is called *clustering*. As a consequence, judgment needs to be exercised as to whether the clustered scenarios represent variations on a similar theme or whether they are substantially different. In the first case, the clustering is a good thing; in the second, it is bad. In other words, if a group of scenarios are similar, and they all affect the same component or components in an architecture (i.e., they cluster), that is a good thing, because it means that the system's functionality has been modularized in a way that properly reflects the modification tasks. If, on the other hand, a group of similar scenarios affect many different components throughout an architecture, that is bad. We return to this theme after SAAM has been presented in more detail.

As an aid to creating and organizing scenarios, we appeal to the concept of *roles* related to the system. Examples of roles include: the person responsible for upgrading the software—the end user; the person responsible for managing the data repositories used by the system—the system administrator; the person responsible for modifying the run-time functions of the system—the developer; the person responsible for approving new requirements for the system, etc. The concept of roles matches the difference between run-time qualities and developmental qualities, i.e., those qualities that are a function of the system's execution (such as performance), and those that reflect operations performed off-line in a development environment (such as modifiability). Scenarios of the former variety would be performed by roles such as end user; the latter by developers or maintainers.

In summary, SAAM 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.

Overview of SAAM:

In this section, we define the steps of a SAAM evaluation, and discuss the interactions among them. Not every SAAM evaluation need go through all of these steps. In fact, projects can benefit from performing almost any subset, as some of the case studies will show.

Develop scenarios. Develop 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.

Describe candidate architecture(s). 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. SAAM evaluations have tended to use very simplistic architectural primitives. A typical representation will distinguish between 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 of the overall behavior or some other more formal and structured specification.

Classify scenarios. 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 architecture 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. A particular scenario is only direct or indirect with respect to a particular architecture. One candidate architecture may execute a scenario directly while another may require modification before it can be executed. When using SAAM in an acquisition effort, whether scenarios are direct or indirect is often of great interest and often unknown to the customer before the analysis begins. Obviously, an architecture for which the scenarios of interest are directly supported is more capable than one that requires modification to support those same scenarios, and will score higher under SAAM.

Perform scenario evaluations. For each indirect 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 that 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. A weighting of the difficultly must also accompany this stage. Usually, this weighting is coarse grained, based on the understanding of the architecture. 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.

Reveal scenario interaction. When two or more indirect task scenarios require changes to a single component of a system, they are said to *interact* in that component. Scenario interaction is important to highlight because it exposes the allocation of functionality to the product's design. The interaction of semantically unrelated scenarios explicitly shows which system modules are computing semantically unrelated functions. Areas of high scenario interaction reveal potentially poor separation of concerns in a system component. Thus, areas of scenario interaction indicate where to focus the designer's subsequent attention. The amount of scenario interaction is related to metrics such as structural complexity [7], coupling, and cohesion [8]. Therefore, it is likely to be strongly correlated with number of defects in the final product.

Overall evaluation. Finally, assign a weight to each scenario and the scenario interactions in terms of their relative importance and use that weighting to determine an overall ranking. The purpose of assigning weights is to resolve the situation where the first candidate architecture scores well on half the scenarios, and the second candidate architecture scores better on the other half. By understanding which scenarios are more important, ties can be broken. Assigning weights is a subjective process, involving all of the stake-holders in the system. Rather than offering a single architectural metric, SAAM produces a collection of small metrics, a set of 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 out-score each other on different scenarios. It has also been the case that the process of performing a SAAM analysis has been used to gain a higher level understanding of the competing architectures; this higher level understanding, rather than just a scenario based table, is useful to perform comparative analysis. The case study of Section 4 emphasizes this aspect.

Crafting the scenarios and representing the architecture(s) are inter-dependent steps. 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 scenario dealing with the number of lines of source code). 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 also reflected in the architecture.

Figure 8 illustrates the steps in a SAAM evaluation.

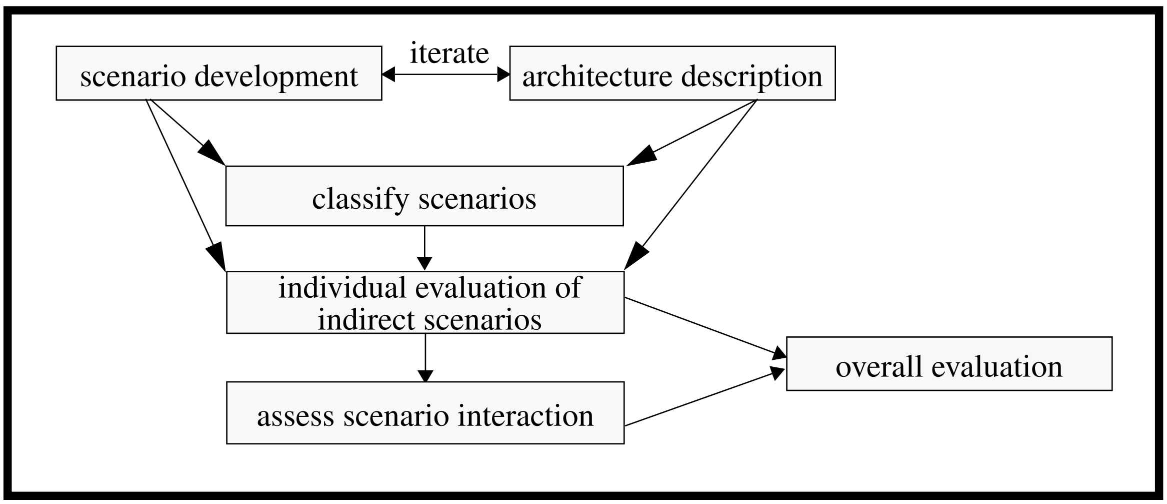

Figure 8: Activities and dependencies in SAAM analysis

1.4 A small example: KWIC

In this section, we apply SAAM to a small, decidedly academic application to provide an easily-understood example of each of the steps of SAAM. The application is the construction of a keyword-in-context (KWIC) index generator. It is the example that Parnas used in his 1972 landmark paper that introduced the concept of information-hiding [[REF]]. The application is small, easily understandable, and familiar. Thus, it makes a good example to illustrate the method. This example has also been used by others to illustrate points about software engineering and software architecture. In this example, the KWIC system has been completely specified beforehand. This would not always be the case when performing an architectural analysis.

The message of Parnas' paper in 1972 was the virtue of information-hiding as the design principle around which to organize a system's functionality. Parnas compared two designs for KWIC (one "traditional" and one developed using the information-hiding principle) and showed the advantages of the latter over the former. SAAM takes many of the ideas that are implicit in that paper and makes them explicit by virtue of its methodical approach to evaluation.

The behavioral requirement for the KWIC program is to take as input a list of sentences, where each sentence is an ordered set of words. The program outputs, in alphabetical order, a listing of all circular shifts of all sentences. So, for example, if the following sentences were given as input:

predicting software quality
architecture level evaluation

the KWIC program would produce, as output:

architecture level evaluation
evaluation architecture level
level evaluation architecture
predicting software quality

quality predicting software
software quality predicting

Defining Roles and Scenarios:

The KWIC example has two roles of interest: the *end user* who wishes to produce the KWIC index and the *developer* who produces or modifies the code that produces the KWIC index.

Two of the scenarios we use that have been discussed by various authors involve the end user:

1. Modify the KWIC program to operate in an incremental rather than a batch fashion. This version of the program would work by accepting one sentence at a time, and producing an alphabetical list of all permutations of all sentences input to date.

2. Modify the KWIC program to eliminate entries beginning with “noise” words (articles, prepositions, pronouns, conjunctions, etc.).

The other two scenarios we will use involve the developer but are not apparent to the end user:

3. Change the internal representation of the sentences (e.g., compressed or uncompressed).

4. Change the internal representation of intermediate data structures (e.g., either store the shifted sentences directly or store an index to the shifted words).

Describing Candidate Architectures:

The next step of the method is to represent the candidate architectures that are to be evaluated, using a common notation. This is done in order to ease the evaluation process so that the evaluator knows exactly what a box or arrow means in the architectural diagram.

Shared memory solution: I

n the first candidate architecture, there is a global storage area, called *Sentences*, where all of the input sentences are stored. The input sentences are: read by the *Input* routine; stored in *Sentences*; shifted by the *Circular Shift* routine (which builds a *Shift Index* of the starting locations of each of the shifted permutations of the original input sentence, rather than storing the sentences themselves in shifted form); alphabetized by the *Alphabetize* routine (which stores an alphabetized version of the *Shift Index*); and then output. *Master Control* passes control to the various routines as required.

Figure 9 is annotated with the evaluations from the scenarios (the numbers in the various computational components). In this step we can simply ignore the annotations.

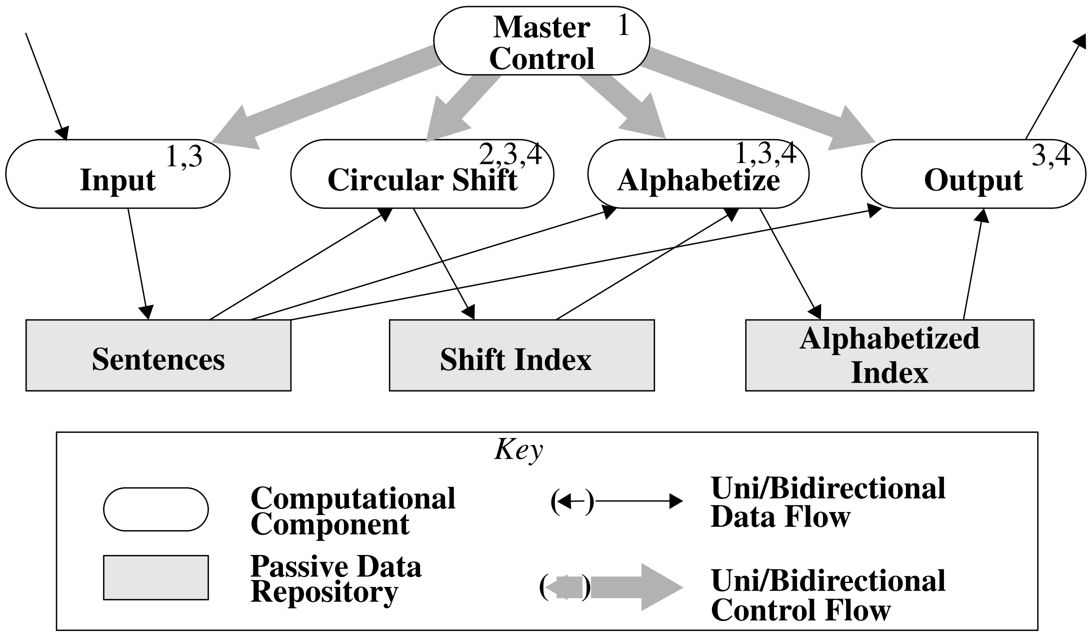

Figure 9: KWIC—shared memory solution

Abstract data type solution:

The second architecture, represented in Figure 10, uses an abstract data type (ADT) where each function hides and protects its internal data representation and provides specialized access functions as the only way to store, retrieve, and query the data. The *Sentence* ADT has access functions *set* and *getNext* to add and retrieve a sentence. The *Shifted Sentences* ADT provides the *setup* access function to set up its circular shifts of the sentences,[1] and the *getNext* function to retrieve the shifted sentences.

The *Shifted Sentences* ADT makes use of the *getNext* function from the *Sentence* ADT to retrieve stored input sentences. The *Alphabetized Sentences* ADT provides a *setup* function and an *i-th* function.The *setup* function calls *Shifted Sentences*'

1. As both Parnas [24] and Garlan and Shaw [17] point out, the ADT solution admits of other possibilities. For example, the *setup* function in the *Shifted Sentences* ADT may in fact do nothing when it is called, and all shifts are done "on demand." That is, whenever the *Shifted Sentence getNext* function is called, the ADT gets another sentence from the *Sentences* ADT and shifts it. However, for the purposes of architectural analysis, some assumptions must be made about how the functions will work together. SAAM assumes that each function is accompanied by a specification, so that when doing an architectural analysis, these decisions are known.

getNext function repeatedly to retrieve all the lines to be sorted and sorts them. The *i-th* function, called specifying some integer parameter *i*, returns the i^{th} sentence from the sorted list of *Shifted Sentences*.

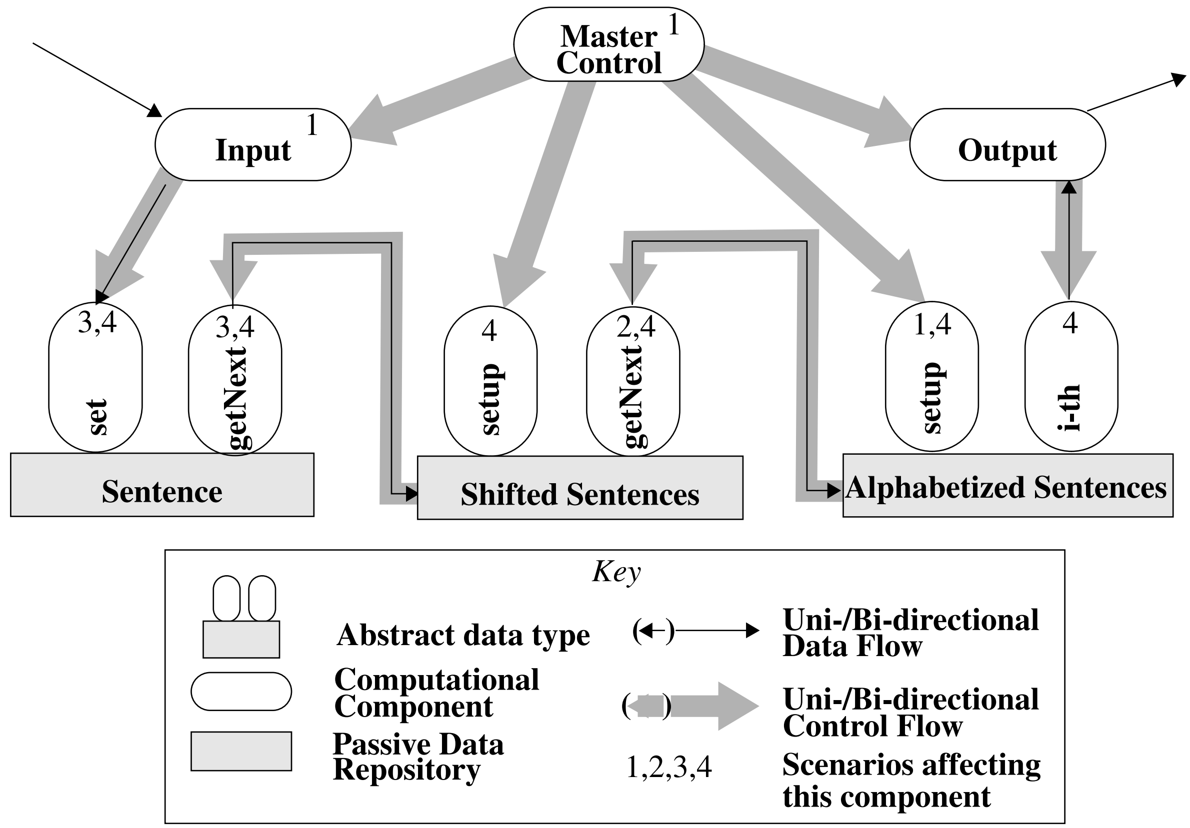

Figure 10: KWIC—ADT solution

Classifying the Scenarios :

Now that we have specified the candidate architectures in a common notation, the next step in the method is to classify the scenarios as direct or indirect against each architecture. In this case, each of the scenarios which we have chosen for evaluation of the KWIC architectures is indirect. None can be directly executed by either candidate architecture so that all of the evaluation is dependent upon some modification to the architecture.

Evaluating the Candidate Architectures:

The next step in the method is to evaluate the architectures against the scenarios. The evaluation begins by considering each of the scenarios in turn.

Scenario 1:

The first scenario is to move from a batch mode to an incremental mode. That is, operate on one sentence at a time rather than inputting all of the sentences and operating on the total input.

For the shared memory solution, this requires a modification of the *Input* routine, so that it yields control after each sentence. *Master Control* must be modified because the sequencing among the subordinate routines is going to be repetitive rather than calling each routine only once. The *Alphabetizer* will need to be modified because an incremental algorithm, such as insertion sort, must be used rather than one that sorts all of the sentences at once. Presumably, the *Circular Shift* routine operates on one sentence at a time and the output function outputs whatever is available whenever it is called.

Note the assumptions we are making about the specification of the routines in the shared memory solution. In general, the accuracy of the judgments are dependent on knowledge of the internal workings of the various computational components. That is why we expect the evaluation to be performed by individuals either with general knowledge of the computational components or with access to the specification of the components.

For the ADT solution, shown in Figure 10, the *Input* function needs to be changed so that only one line at a time is input each time it is called. The *Sentence* abstraction is assumed to be implemented to relinquish control when it has stored its input. Since it receives input one sentence at a time, this need not be changed. The *Shifted Sentences* abstraction is also assumed to be implemented to request and shift all available sentences when it is invoked. Thus, this routine also does not need to be modified. The *Alphabetized Sentences* abstraction must be modified as in the shared memory example.

For this scenario, since both candidate architectures have been affected equally, we rate the two architectures as a tie.

Scenario 2:

The second scenario is the elimination of noise words. This change is most easily performed by modifying the shifter function in both architectures (*Circular Shift* in the shared memory architecture and *Shifted Sentences* in the ADT architecture) to eliminate those shifted sentences beginning with noise words. Since the shifter function is localized in both architectures, and since noise word elimination does not affect the internal representation of the sentences, the difficulty of making this modification is equivalent for the two architectures.

Scenario 3:

The third scenario is the modification of the internal representation of an input sentence, e.g., moving from an uncompressed representation to a compressed representation.

In the shared memory architecture all functions share a common representation, and so all of the functions are affected by this scenario, except for the *Master Control*. In the ADT architecture, the input sentence internal representation is buffered by the *Sentence* abstraction. Thus, the ADT architecture is better than the shared memory architecture for this scenario.

Scenario 4:

In this scenario—changing the representation of the intermediate data structures—we observe that the shifting and alphabetizing functions maintain indices to the input sentences and only one copy of the input sentence exists.

For the shared memory example, this requires a modification of the *Circular Shift*, the *Alphabetizer* and the *Output* routines.

For the ADT solution, this requires a modification of the *Shifted Sentences* and the *Alphabetized Sentences* ADTs. Thus the ADT solution affects fewer components and is superior for this example.

Scenario interaction analysis:

Both figures have been annotated to reflect the impact of each scenario. For example, the “1” inside the *Master Control* component in Figure 9 reflects the change to that component required to support Scenario 1. Examining the candidate architectures to determine which one has more contention of scenarios (components with more than one scenario annotation) we see that the ADT is superior in this regard. While both architectures show scenario contention in four components, in the shared memory solution the contention is among three of the scenarios in two components (*Circular Shift* and *Alphabetize*), whereas no component in the ADT solution has contention from more than two tasks.

0.1 Summary of evaluation

Table 22 summarizes the results of the SAAM evaluation. A “0” indicates that neither architecture is superior with respect to this scenario. In a real evaluation, the evaluation would conclude by weighting the scenarios to represent the organization’s priorities. For example, if the addition of functions was the primary concern (as represented by scenario 2) then the two architectures would be ranked closely, since they do not differ with respect to this task.

If, on the other hand, a change in the internal representation of sentences (as represented by scenario 3) was of primary concern to the organization, then the ADT solution would be superior.

Table 22: Summary of evaluation

Architecture	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Contention
Shared Memory	0	0	-	-	-
Abstract Data Types	0	0	+	+	+

2. A Larger Example (Internet information systems)

Internet Information Systems (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. [27] 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 [25] 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 report 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.

2.1 Scenarios for IIS

This section defines a set of 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.

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 ([10]), involves undirected behavior. For example, a user might want to look to see what information is available across various sites.

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.

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 described above.

2.2 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

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.

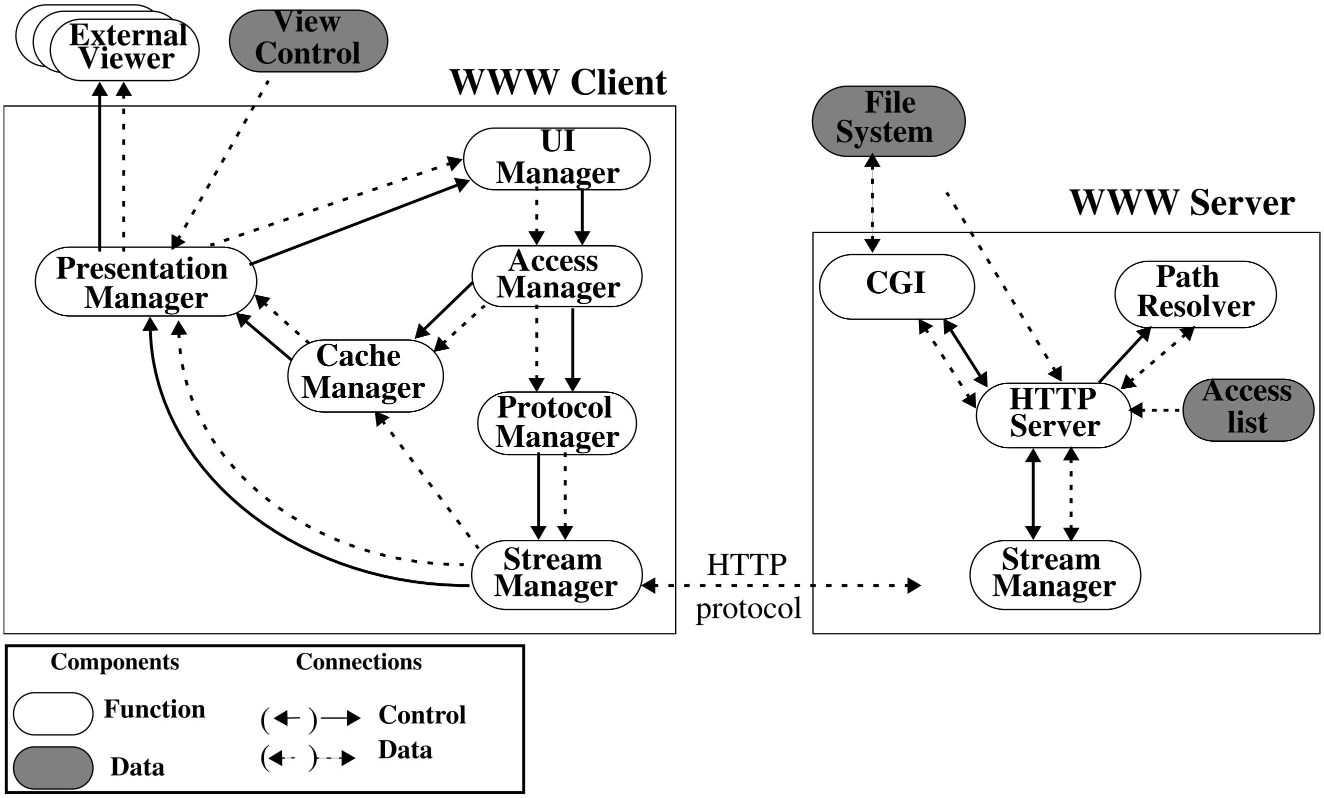

Figure 11: WWW architecture

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.

WWW/Mosaic:

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 ([4],[5],[6],[16]). Figure 11 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.).[1]

WWW clients provide a User Interface Manager.[2] 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 Resource 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

1.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.

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

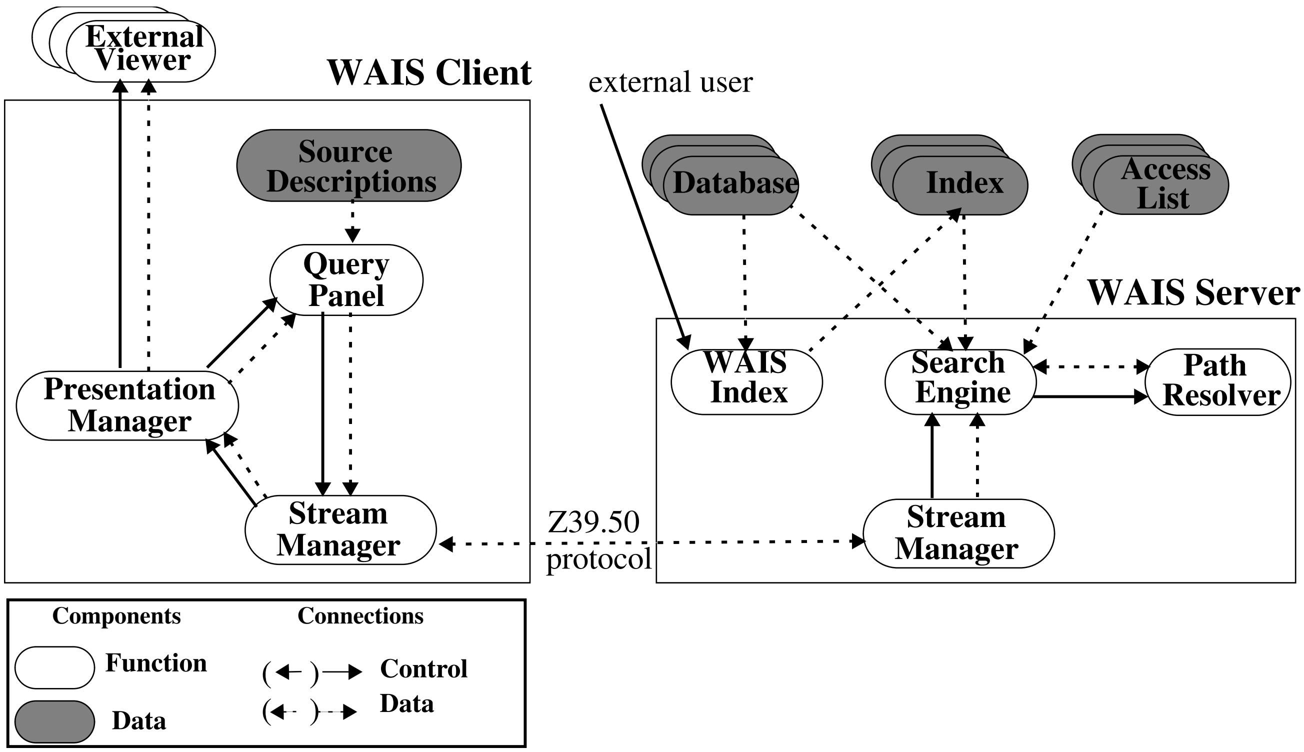

Figure 12: WAIS architecture

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.

WAIS:

Wide Area Information Server (WAIS)[1] is a network publishing and retrieval system designed to help users find distributed networked information using a mixture of natural language and boolean queries ([11], [29],[30]). 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 12.

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 sum-

1.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 report, 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.

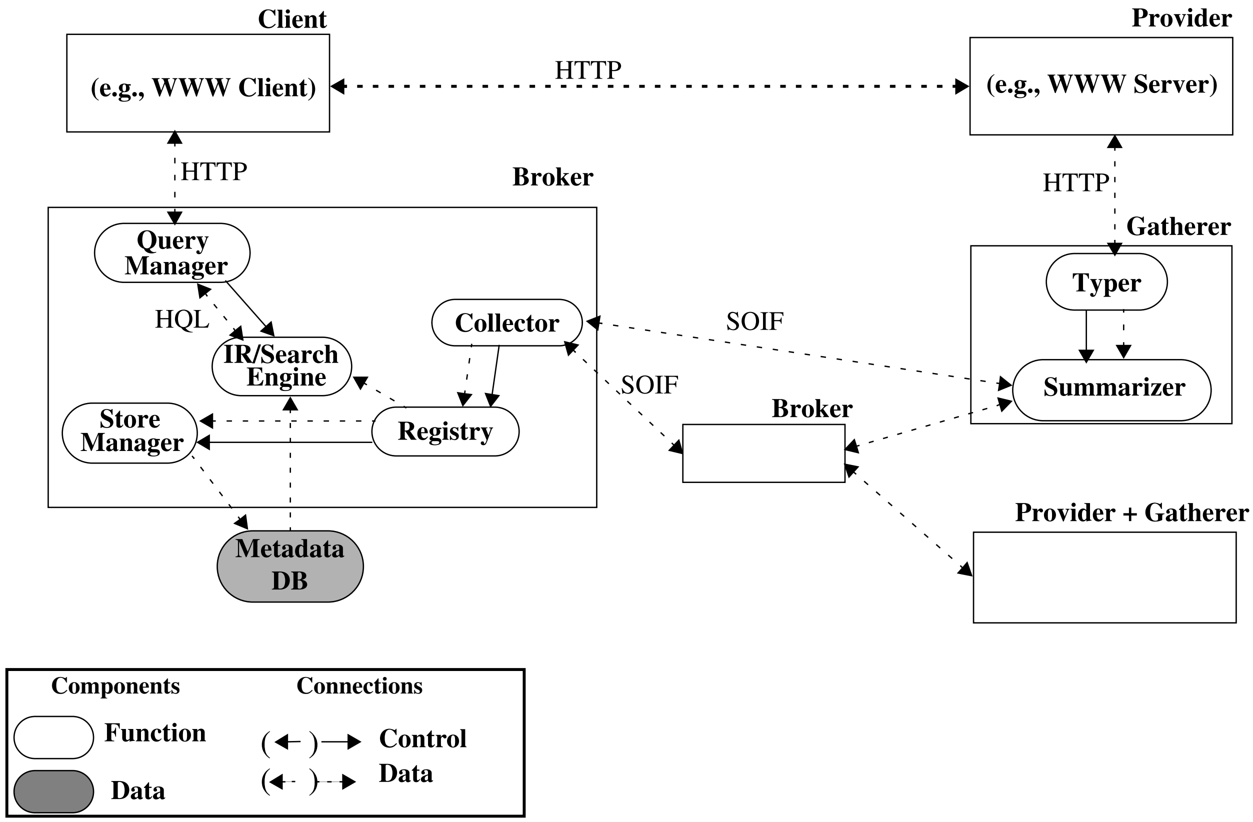

Figure 13: Harvest architecture

mary 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 communication 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.[1] 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.

Harvest:

Harvest is a resource discovery tool which leverages off of much of the functionality of the Web [8]. 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 13) 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

1.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.

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.

0.2 Scenario evaluations

In this section, we will evaluate the three candidate scenarios against each of the consumer and provider scenarios defined in Section 2.1. For reasons of space, we will not discuss the administrator scenarios here.

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 (or 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.

0.2.1 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.[1]

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 [19] 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.

1.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.

2.3 Overall IIS evaluation

The results of the scenario evaluations are given in Table 23. The final three columns indicate the extent of architectural modification that is required for indirect scenarios, and rank the three architectures. Architectures which support the scenario directly are given a ranking of +. Architectures which require changes (i.e. those architectures for which the scenarios are indirect) are ranked according to the number of affected components.

Table 23: Summary of evaluation of WWW and WAIS

Scenario	**Changes/additions necessary**		
	WWW	**WAIS**	**Harvest**
Information consumer			
C1. Resource location	+	- Query panel	+
C2. Resource discovery	- Protocol Manager CGI	+	+
C3. Browsing	+	- Query Panel Search Engine	+
Information provider			
P1. Adding information of supported type	+	+	+
P2. Adding lots of information of supported type	+ File System	0 Database WAIS Index	- Provider repository Gatherer Broker-Gatherer SOIF Broker
P3. Creating new data of a type currently unsupported	- HTTP Server View Control External Viewer	0 Database WAIS Index	+

This table can also be used to reveal the scenario interactions discussed as Step 4 in Section , but we will not discuss that here. We also do not consider weighting the scenarios in this report, 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 23 to justify choosing one IIS over another.

3. Detailed Methodology Description

Already done.

3.1 Input formats

???

3.2 Output formats

???

3.3 Steps

Already done by example

3.4 Evaluation

???

4. Glossary

(part of this should be common across guidebooks)

5. Annotated References to the Literature

[1] Abowd, Gregory, Jonathan Engelsma, Luigi Guadagno and Okonon Okon. (Significant contribution from all authors, principal authorship by Engelsma and Abowd) Architectural Analysis of Object Request Brokers. To appear in Object Magazine special issue on distributed systems, March 1996, pp. 44-51.

[2] Abowd, G., Kazman, R. and Pitkow, J. (Significant contributions from Abowd and Kazman with Abowd as principal author) Analyzing Differences Between Internet Information System Software Architectures, Proceedings of ICC '96, (Dallas, TX), June 1996.

[3] Len Bass, Paul Clements, Linda Northrop, Amy Zaremski, Gregory D. Abowd and Rick Kazman (principal contributions from Zaremski and Abowd. Principal authorship by Abowd) Industrial best practice and recommendations for software architecture evaluation. Submitted to *IEEE Computer*. February, 1997.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Brown, A; Carney, D.; & Clements, P. "A Case Study in Assessing the Maintainability of a Large, Software-Intensive System." *Proceedings of the International Symposium on Software Engineering of Computer Based Systems*. Tucson. IEEE Computer Society, March 1995.

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

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

[12] Clements, P., Bass, L., Kazman, R. and Abowd, G. (Significant contributions by Clements and Bass with Clements as principal author) Predicting software quality by architecture-level evaluation. In the proceedings of *Fifth International Conference on Software Quality*. Austin, TX, October 1995.

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

[14] M. De Simone, R. Kazman, "Using SAAM: An Experience Report", *Proceedings of CASCON '95*, (Toronto, ON), November, 1995, 8 pages, to appear.

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

[16] 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.

[17] D. Garlan, G. Kaiser, D. Notkin, "Using Tool Abstraction to Compose Systems." *IEEE Computer 25*, 6 (June1992): 30-38.

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

[19] 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.

[20] 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.

[21] Kazman, R., Abowd, G., Bass, L. and Clements, P. (Significant contributions and authoring by all) Scenario-based analysis of software architecture, *IEEE Software*. **13**(6):47–56, November 1996.

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

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

[24] D. Parnas, “On the Criteria To Be Used in Decomposing Systems into Modules.” *Communications of the ACM 15*, 12 (1972): 1053-1058.

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

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

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

[28] M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, G. Zelesnik, “Abstractions for Software Architecture and Tools to Support Them.” *IEEE Transactions on Software Engineering 21*, 4 (April 1995).

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

[30] WAIS, Inc., “WAIS for UNIX”, On-line documentation, URL http://www.wais.com/company/Tech_TOC.html.(How this method relates to other methods, discriminators, links to further information)