First International Workshop on Architectures for Software Systems, Seattle, April 1995

An Architectural Analysis Case Study: Internet Information Systems[1]

Rick Kazman[1], Len Bass[2], Gregory Abowd[3], Paul Clements[2]

[1]Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

[2]Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA
U.S.A. 15213

[3]College of Computing
Georgia Institute of Technology
Atlanta, GA
U.S.A. 30332-0280

Abstract: This paper presents a method for analyzing systems for nonfunctional qualities from the perspective of their software architecture and applies this method to the field of Internet information systems (IISs). Since nonfunctional qualities tend to be too abstract for comparing systems at the level of their software architecture, the method employs task scenarios as a way of comparing and contrasting the capabilities of different architectures with respect to specific aspects of the quality attributes of interest. Three issues are explored in detail in this paper: generality of the method, repeatability of the method, and the distinction between direct and indirect task scenarios with respect to architectural ramifications.

1 Introduction

Increased attention is being directed toward the description and analysis of software architecture as an artifact of design. Our goal is to provide a method for analyzing architectural descriptions to determine how nonfunctional software qualities are supported. Our previous work [4] was motivated by frequent claims in the literature that some system possessed particular nonfunctional qualities (modifiability, scalability, security, etc.) without any validation. Our concern was that the claims were seemingly unsupported; further, there seemed to be no clear way to demonstrate the validity of such claims.

We have focussed on defining a scenario-based method for evaluation of software quality for an architectural description. The previous example in [4] examined the domain of user- interface development tools with respect to the software quality of *modifiability*. We claim that such a quality label is generally too abstract to be useful; such claims need to be made more concrete. That is, there are a great many modifications that might be made to a given system; any particular design may make some of these modifications easy and others more difficult. Thus, concrete scenarios specific enough to characterize what is meant by modifiability in the domain of user- interface software development were provided to enable support of a claim of modifiability.

1. This work partially supported by the U.S. Department of Defense and the Natural Sciences and Engineering Research Council of Canada.

We restrict our analysis method to examining architectural descriptions of the system. An architectural description contains information about what functions or services the system provides and how that functionality is allocated onto a structure of interacting components. The evaluation demonstrates how the function allocated onto structure supports the activity of a particular scenario.

The work reported here has two objectives: to discuss the evolution of our method, and to provide another interesting example as a case study. There were several valid criticisms of our initial effort:

·We could not demonstrate the generality of our method, since we applied it to only a single application domain, user interface software, and looked at only a single quality factor, modifiability.

We did not demonstrate the repeatability of our method. That is, if three different software engineers applied the method, we could not say with any certainty whether they would produce one, two, or three different analyses among them.

*There was an unclear relationship between the scenarios chosen and the description of important domain functions.

We will try to address all three of those criticisms in the context of another case study. In this case study, we will examine various Internet information systems—to see how they support qualities as varied as scalability, maintainability, integrability, and portability. Though we are reporting now on work in progress, we are encouraged that this new case study has helped us to clarify our method and begin to address the above criticisms.

The paper begins by introducing the domain and the scenarios. We then briefly describe the meaning of some of the terms we use with respect to architectures, introduce our method for analysis, describe three different Internet information systems, and apply the method to these systems. We conclude by providing a status report of what is possible with the current state of the method and what extensions need to be made to it.

2 Introduction to the Case Study and Scenarios

Internet information systems (IISs) are large software systems which are increasingly important in both academic and commercial computing. Some examples of these systems are the World Wide Web, Gopher, Wide Area Information Server, and Archie/Prospero. Significant amounts of resources are being devoted to creating information repositories, seekers, and servers. Given the importance and ubiquity of these systems, it is natural to attempt to analyze them with respect to their support for various quality attributes and to ask the questions: how well does this system scale up,, how portable is this system, or how good is this system's support for integrability?

While these sound like important analyses to make, particularly considering the tremendous growth of IISs,[2] these questions cannot be answered as posed. Quality attributes are too abstract to provide meaning-

2. According to a recent survey [1], the number of World Wide Web servers appears to be growing an order of magnitude per year (over the past two years).

ful starting points for analysis because they have multifaceted meanings. This paper argues for a different approach to architectural analysis based upon concise *task scenarios*, rather than quality attributes. This is a subtle departure from our original work in which we stated that one begins with a software quality and then refines it through representation as concrete scenarios. When we used that older approach in this new case study with several abstract quality factors, we spent more time worrying about how to categorize a particular scenario than we did actually generating the scenarios of significance.We decided it was more important to know which task scenarios were being supported, so we have de-emphasized the earlier top-down approach based on qualities. Task scenarios realize abstract quality attributes in a sufficiently concrete way that they alone provide a reasonable starting point for meaningful analyses. It might be useful to determine later which abstract qualities the concrete scenarios represented, but not initially.

More specifically, rather than saying integrability is important for an IIS and then suggesting an instance of integrability (e.g., the task scenario of introducing a new media type in an existing system), we just begin with this important scenario. We can then add to this scenario description by distinguishing between introducing a single new media type and many new media types. This latter scenario represents both integrability and scalability (and also, one could argue, maintainability).

The use of scenarios to define the purpose and scope of a system is a common practice in workflow, task, and requirements analysis. In those areas, the scenarios tend to be fairly rich descriptions of end-to-end behavior. We do not have a use for scenario descriptions of such depth and context. Our scenarios are fairly concise descriptions of tasks that are carried out between some role player and the system. They indicate the intention of the various role players only.

There are a number of roles whose concerns influence the design of a particular system. These include the end user, the system administrator, and the system builder. The end user is only incidentally affected by the scalability or portability of a system. In our presentation of the scenarios, we characterize them partially based on which role is interested in each scenario.

3 Architectural Models

Since the discipline of software architecture is still an emerging one, terms have a tendency to be used in a variety of fashions by different authors. In this section, we give our perspective on some of the terms that are important to our work. We differentiate between the *functional* and the *structural* perspective on software architecture. The functional perspective indicates the functions or services provided by the system and the structural perspective describes the implementation in terms of independent and interacting components. Our evaluation technique depends on being able to draw conclusions based on how the functions were allocated over components in the structural descriptions.

3.1 Functional Partitionings

A functional partitioning specifies the distinct functions or services that a system must provide, but not the actual structure of the software that computes those functions (although the functional partitioning may suggest a structure). Thus, one system may combine functions *a, b*, and *c* into a single software component, whereas another system may distribute each of functions *a, b*, and *c* over several components.

The functional partitioning is a means for understanding what is common between all applications in a given problem domain. Though this description might sound similar to a domain analysis, there are some important distinctions. We will use the task scenarios to develop a functional partitioning. Since these scenarios are not intended to cover the complete usage of the system, the functional partitioning that results is not guaranteed to cover all aspects of any system developed in this domain. We are interested only in a description of the functions that serve the scenarios of interest. We are also not concerned with defining dependencies among the functions. Those dependencies will be reflected somewhat by the structural description of a system.

3.2 Structure

A system's software structure reveals how it is constructed from smaller connected pieces. The structure is described in terms of the following parts:

- acollection of components that represent computational entities (e.g., a process) or persistent data repositories (e.g., a file)
- a representation of the connections between the components, that is, the communication and control relationships among the components

4 Analysis Method

In this section we present our analysis method. In the broadest sense, it has three stages:

1. Define a collection of scenarios that represent important usages of a system in the domain, including the points of views of all of the roles involved in the system.
2. Use the scenarios to generate a functional partitioning of the domain and a coupling of the scenarios with the various functions or services in the partitioning
3. Use the functional partitioning with reference back to the scenarios to perform the analysis of the proposed architectures.

The analysis will provide, for each scenario, a ranking of the architectures being evaluated (the so-called *candidate* architectures). It is the responsibility of the evaluator to determine the weighting of the scenarios with respect to each other and, hence, an overall rating of the candidates.

Some mature domains (such as databases, compilers, user interfaces) will already have a functional partitioning that is available for use. In such cases, this functional partitioning may be used instead of generating one from the scenarios. It is also possible that the previously available functional partitioning may not reflect all of the scenarios and may need to be further refined. Such distinctions represent refinements of the method we present here. The presentation here does not assume an existing functional partitioning. The steps of the method are

1. Develop task scenarios that illustrate the kinds of activities that the system must support. These will reflect the nonfunctional qualities of interest, but, as discussed in Section 2, the explicit connection between scenario and qualities is not necessary. These scenarios will also present interactions from different roles, such as end user, system administrator, maintainer, and devel-

oper.

2. Develop a functional partitioning that manifests the ramifications of the task scenarios. The various functions or services introduced in the partitioning will be introduced as a result of one or more scenarios. Maintain a coupling between the elements of the functional partitioning and the scenario(s) that each supports.
3. Classify the task scenarios as *direct* or *indirect* by ascertaining whether they are executable by the functions and services determined in step 2 (direct scenarios) or are manifestations of the structure (indirect scenarios). This is not always an obvious decision and involves an evaluation of the types of functions that should be included in systems in the domains. In general, a function should be added whenever the contained functionality is contained in some system in the domain. For example, in the IIS domain, when adding new data to an existing data base, a system may or may not automatically update the index files. When the functional partitioning is developed, a decision must be made as to whether there is a function to automatically update index files when adding new data. If such a function exists, this is a direct task; if it does not and updating the index files is a manual operation of some sort, then this is an indirect task.
4. Express the candidate architectures in a common syntactic architectural notation.
5. Map the functional partitioning onto each candidate architecture.
6. For each direct task scenario, determine whether the target system supports this task (by feature inspection). That is, a direct task can be executed by the functions in the functional partitioning. This check is to see if the task can be executed by the functions in the candidate architecture. The evaluation is binary; a candidate architecture will either support the task scenario and receive "+" on this scenario or it will not and receive a "-" on this scenario.
7. Identify the functions and services coupled with the indirect task scenarios. For each function or service, inspect the structural elements to which it is allocated in the candidate to determine whether any other function or service is computed within the same structural element. If so, then give the candidate a "-" for the scenarios coupled with the function or service. If not, then give the candidate a "+" for those scenarios.
8. Finally, develop an overall ranking of the candidate architectures by weighting each scenario and using that weighting to interpret the ratings on the individual scenarios.

Some of these steps may be iterative. For example, the choice of scenarios will be influenced by the systems in the domain to be analyzed. The choice of whether a task is direct or indirect may also be influenced by the candidates to be examined. All of these choices can be made by reiterating through the appropriate steps of the method.

5 Task Scenarios

In this section we perform step 1 of the method for the domain of IIS: define a set of task scenarios according to the roles that they affect. Above, when we were speaking generically, we mentioned the roles of end user, system administrator, and system builder. Within the IIS domain, the roles of interest can be more specifically named: information consumer (a person who uses an IIS to search for information); 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 builder (a person who modifies part of the infrastructure of the IIS).

Information-Consumer Scenarios

- Perform a directed search for information. The user has a relatively clear goal of the desired information and attempts to use the IIS to locate that information.
- Browse for information. This is an undirected search. The user has no particular goal to find certain information and wishes to look through whatever is encountered.

Information-Provider Scenarios

- Add information to an existing data repository. We assume that data exist in various sites or repositories located on the Internet. In this scenario, an information provider wants to add more data to an existing repository. We assume that the dataare in an acceptable form for the repository.
- Add a data repository (of an existing type) at a new location. In this scenario, a provider wants to introduce a whole new repository at a location on the Internet that was not previously accessed by the IIS. The form of the information is similar to information in existing repositories, so we can assume that the infrastructure of the IIS need not be modified to access the data.
- Add a new data repository (of an existing type) at an existing location. Similar to the previous scenario, except that the repository will reside at an Internet location that already hosts another repository accessed by the IIS.
- Add a new view of preexisting information. Most IISs allow you a particular view of the information that they access, whether by means of a hierarchical menu of documents (as in Gopher) or a straight listing of files that match some search criteria (as in WAIS). This scenario refers to the effort needed to provide a different view of the information in an existing repository.
- Move information (for reorganization purposes). It might be desirable to change the physical location of a repository, or a piece of information within a repository. This scenario refers to making that change in such a way that the information consumer is not affected.

System-Builder Scenarios

- Add a new information access type. Allow for the introduction of new types of query - formulation mechanisms such as query by example, map-based queries, or queries formulated by intelligent agents.

- Add a new type of data repository. A large part of the appeal of an IIS is its ability to deliver data of varying types (text, video, audio, etc.) with minimal effort by the consumer. Much of the task of delivering a certain type of data is taken care of by the infrastructure of the IIS, which will recognize a type and know what protocol to use to deliver it to the consumer. This scenario is concerned with the effort involved to modify the infrastructure of an IIS so that it can handle a new type of data.

6 Functional Partitioning

In this section we exemplify step 2. That is, we use the scenarios to derive a collection of functions and services (and a data repository).

6.1 Internet Information System Functions

We begin the process for defining functional decomposition by considering a subset of the task scenarios. The first set of task scenarios that we will consider is

- search for information
- browse for information

Given this set of tasks, we can infer that we need a data repository, a way to view data, and a way to formulate queries to retrieve data. These needs translate into three functions or services:

UI: user interface

IA: information access engine

DR: data repository

In addition, if we are supporting searching, it should be done efficiently, so we include an additional function to provide an inverted file for the data.

IX: index

Now we consider the effects of adding additional task scenarios on the functional decomposition. In each case, when we add a new function, we separate it from all other functions (recall that this is a specification of functions, not structure—a given system may decide to combine several functions into a single structural element, or distribute a single function over multiple structural elements). The next set of tasks that we consider are

- add a new type of data repository
- add a data repository (of an existing type) at a new location
- add a new view of preexisting information

In order to satisfy these task scenarios, we need to posit four new functions;

DW: database wrapper

LD: logical database

NR: name resolver

CM: communication manager

The database wrapper is a function that makes heterogeneous data repositories appear uniform to the information consumer or to a program that accesses those databases. A logical database (or *view*) is a grouping of preexisting information, possibly collated from disparate sources. [A communication manager allows us to communicate with other computational entities. For example, the communication manager might manage the use of TCP/IP to communicate with other Internet sites.} A name resolver allows users to access resources by name across the Internet without knowing precisely where those databases are located or how to define a path from the user to the resource. As an example, most electronic mail is addressed symbolically and delivered to a specific Internet address with the aid of a name resolver. Such a function is crucial to systems that need to address resources at unpredictable locations.

Now we consider the scenario that moves information from one location to another. There are two functions involved in this scenario: the name resolver identified previously, and

MV: a move utility

that automatically updates the tables used by the name resolver.

The other scenarios:

- adding a new information access type
- adding information to an existing repository
- adding a new repository of an existing type

cause the additions of the following functions:

VIA: virtual information access

IAU: information adder utility

RAU: repository adder utility

The VIA provides a virtual interface to the information-access engine and buffers changes to that mechanism based on the query-formulation mechanism; the utilities are responsible for automatically updating whatever tables are associated with adding data (such as the inverted files) or adding a new repository (such as the WAIS repository of repositories).

To our knowledge, there is no IIS that has the function of the RAU. On the other hand, if this is an important scenario, then it is useful to determine whether a particular system supports it. Thus, it is a function in our functional partitioning. In general, the functional partitioning derived using our technique will be a superset of the functions present in any particular IIS.

6.2 Task Scenario Classification

Now that we have developed our functional partitioning, we perform step 3 of the method: to determine, for each of the task scenarios, whether that scenario is direct or indirect. We perform this evaluation

against the theoretical IIS that we have defined via the functional partitioning. We also maintain the coupling between the indirect tasks and the portions of the functional partitioning associated with those indirect tasks.

As we mentioned above, there is a connection between the functions defined and whether a particular scenario is direct or indirect. Our method calls for adding a function whenever a scenario demands it. However, scenarios are considered direct whenever any system in the domain supports the functionality directly. Thus, other systems within the domain being evaluated will be ranked lower if they do not support that function directly. If *none* of the systems being evaluated has the indicated functionality, they will all be ranked lower equally and the directness or indirectness of the scenario doesn't affect their relative rankings. Of course, the weight given the scenario is determined by the evaluator.

For example, when one wants to move information (for reorganization purposes) in an IIS, this task might be direct or indirect. Furthermore, if one has an index built on top of the information, re-creating the index after the information has moved might be direct or indirect. If at least one IIS supports these tasks directly, then the task scenario itself is labelled as being direct. Other IISs might support the scenario indirectly (for example, by manually moving the files or rebuilding the indices), and those IISs will be ranked lower than an IIS that provides this functionality directly.

Our classification of the tasks given in Section 5 and the coupling with the functions defined is shown in Table 1. We indicate the functional coupling only for indirect tasks, and we use this information in our analysis of the architectural support for each task, shown next.

Task Scenario	Classification	Functional coupling
search for information	direct	
browse for information	direct	
add information to an existing repository	direct	
add repository (of an existing type) at a new location	direct	
add new repository (of an existing type) at an existing location	direct	
add new view of pre-existing information	direct	
move information (for reorganization purposes)	direct	
add new type of data repository	indirect	DW, DR
add new information access engine	indirect	UI, IA, DW

Table 1: Task Scenario Classifications

7 Systems to Be Analyzed

We will now examine three well-known IISs—the World Wide Web, Prospero, and WAIS (Wide Area Information Server)—using the method defined in Section 4, and in light of the task scenarios defined in Section 5. The steps are to represent the architecture of each candidate in a common notation, map the

functional partitioning to the architecture, see if the direct scenarios can be directly executed by the candidate, and evaluate the indirect scenarios based on the functions coupled to them. Not all systems will support all of the direct and indirect scenarios. It is by linking these scenarios to the architecture of the various systems (represented in a common notation) that we can make comparisons between them and understand the tradeoffs involved in supporting or not supporting a particular task.

To analyze and compare a number of independently developed systems, it is helpful to first represent these systems in a common structural notation, as suggested in [4].[3] This is step 4 of the method. The notation that we will be using is shown in Figure 1.

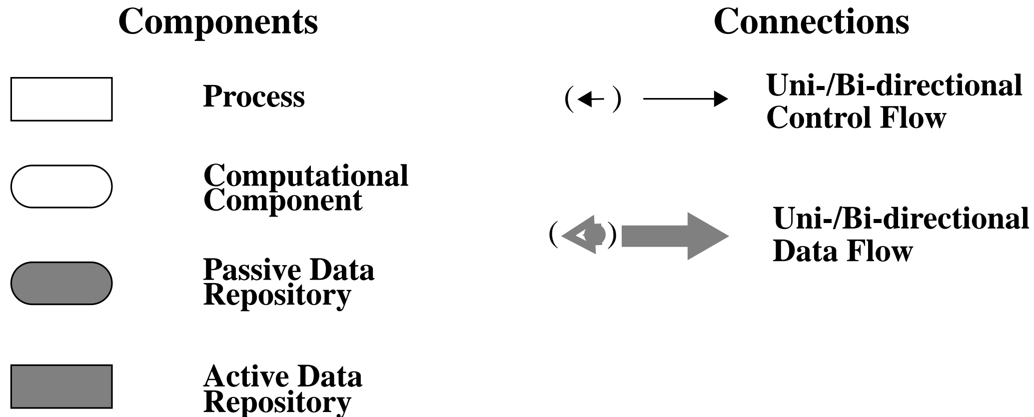

Figure 1: Structural Notation

Given this notation, we can now represent each of the three IISs and examine their suitability with respect to each of our task scenarios. For each system, we perform step 5 of the method: we map the functional partitioning derived in Section 6.1 onto each of the architectures. We will indicate the allocation of functions to each structural component by annotating the structural component with the two-letter acronym introduced in Section 6.1, presented in italics: *UI* for User Interface, *IA* for Information Access, and so forth.

For each candidate IIS, we will exemplify the final steps in the method by considering the ramifications of the indirect tasks on the candidate's software architecture. In each case, we appeal back to our functional partitioning to determine the ramifications of a task scenario. We consider the effects of a single instance of each scenario, and of multiple instances.

7.1 World Wide Web

The World Wide Web (WWW) is a distributed hypermedia system organized as a loosely connected set of clients and servers that share a common set of communication and markup protocols. Servers make internet resources available to a community of clients that speak HTTP (the hypertext transfer protocol). In

3. The notation used here is a slight variation of that used in [4].

addition, WWW clients typically understand a number of other protocols [1, 2]. Figure 2 shows an example of a typical WWW client and server.

The minimal WWW server must understand HTTP and respond to requests for resources by sending them, appropriately formatted, to the requesting client. Thus, a server has some set of files for which it serves requests, an HTTP server which "speaks" at least this one protocol, a path resolver that allows it to determine which file an HTTP message is requesting, and a stream manager which manages the raw communication with the network. In addition, if a server is making resources available that do not speak HTTP, then it must provide a common gateway interface (CGI) to translate from the native protocol into HTTP.

WWW clients must provide a user interface manager to display the results of user requests, an access manager to formulate user requests in terms of the underlying protocols, a protocol manager to map between the various protocols that WWW supports (HTTP, NNTP, WAIS, Gopher, FTP) and the access manager. The stream manager handles network communications, and a cache manager keeps a local cache of retrieved information, so that subsequent requests for the same information may be handled locally.

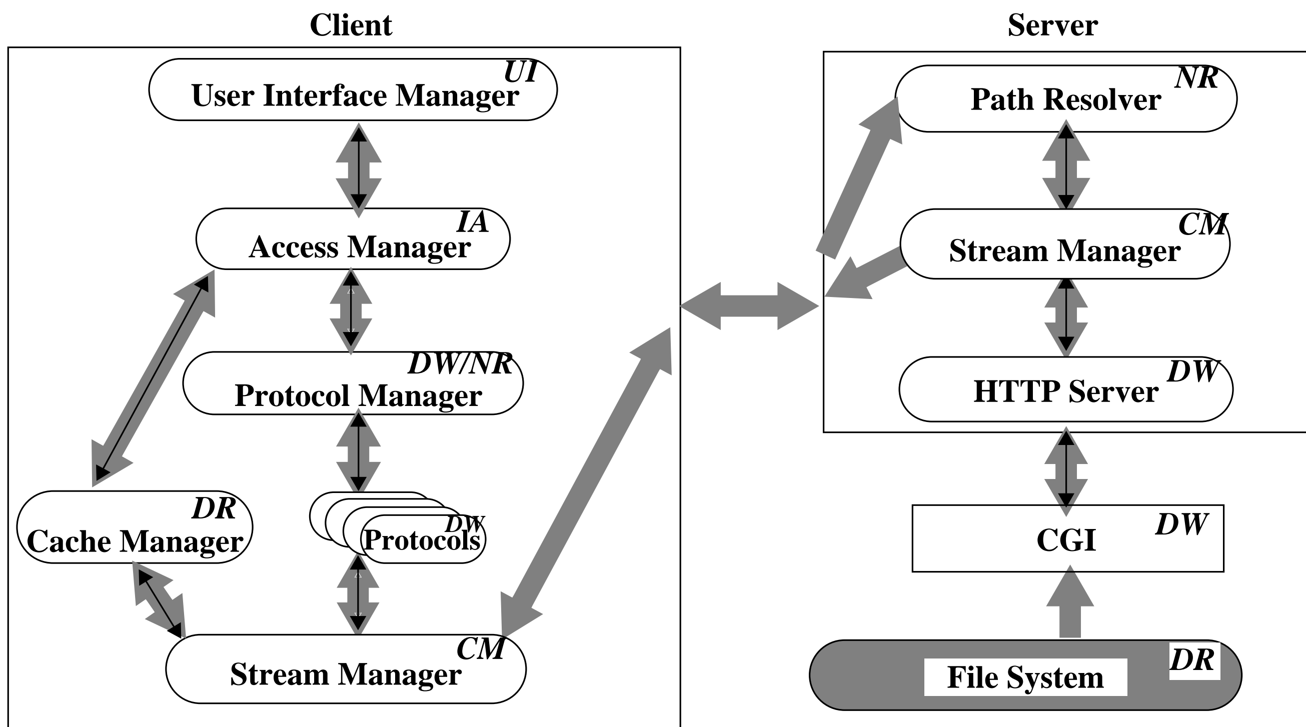

Figure 2: A Typical World Wide Web Client and Server

7.1.1 Add New Type of Data Repository

Referring to Table 1, we see that this indirect task affects the data repository (DR) and the data wrapper (DW) functions. Examining Figure 2, we see that the data repository function is mapped onto the file system and cache manager, and the data wrapper is mapped onto the CGI, HTTP server, cache manager, and protocol manager. We need to consider how adding a new type of data repository affects these structures.

If the new type of data repository can communicate using HTTP, then the change is isolated to replacing the file system. If the data repository does not communicate using HTTP (for example, to access an existing SQL database) then a CGI "gateway program" [5], a separate executable, must be created to translate the underlying file system into hyper text markup language (HTML).

For each instance of this task, a separate CGI must be created, which maps between the new type of data repository and the HTTP server. Thus, the task scales well because the file systems and servers are distributed, and because the WWW developers have created the CGI as an interface between native file systems and HTTP. There are no internal program limitations that dictate how many CGIs or file systems may be connected to WWW servers.

Finally, the cache manager and protocol manager are not affected by this change, because they manipulate documents in a well-known format, such as HTML, using a well-known protocol, such as HTTP or FTP. For this scenario, WWW merits a "+".

7.1.2 Add New Information-Access Engine

Once again referring to Table 1, we see that this indirect task affects the information access (IA), user interface (UI), and database wrapper (DW) functions. The information access function, if it assumes an underlying data model which already exists in the system, can be added without affecting any other component.

If, on the other hand, the new information-access engine requires a different data model (for example, if one wanted to add SQL-style searches to a hypertext browser, or if one wanted to be able to sketch visual queries in an interface to an image database), then the protocol manager would need to be modified to provide this view of the data, and either a new protocol would need to be added to the existing set or a new CGI program would need to be written to translate the existing protocol into HTTP. In addition, the user interface would need to be modified to provide the appropriate query and result-display mechanisms. This merits WWW a - for this scenario.

The response to the addition of large numbers of new information-access engines is similar. If these engines follow the existing data models, adding more of them amounts to the replacement of an isolated component. If we want to add large numbers of new information-access engines, each of which presupposes a different data model, then the protocol manager and user interface functions could easily become complex, posing a bottleneck for such scalability.

7.2 WAIS

Wide area information servers (WAIS)[4] is a network publishing system designed to help users find information distributed over the Internet using a mixture of natural language and boolean queries [8]. The querying capability (services for the information consumer) of WAIS is based on the client-server

4. WAIS and Wide Area Information Servers are both registered trademarks of WAIS Inc.

architecture, and is depicted in Figure 2.[5] WAIS also provides various services (not shown in Figure 2) to aid the information provider, and these are basically intended to modify or create the servers.

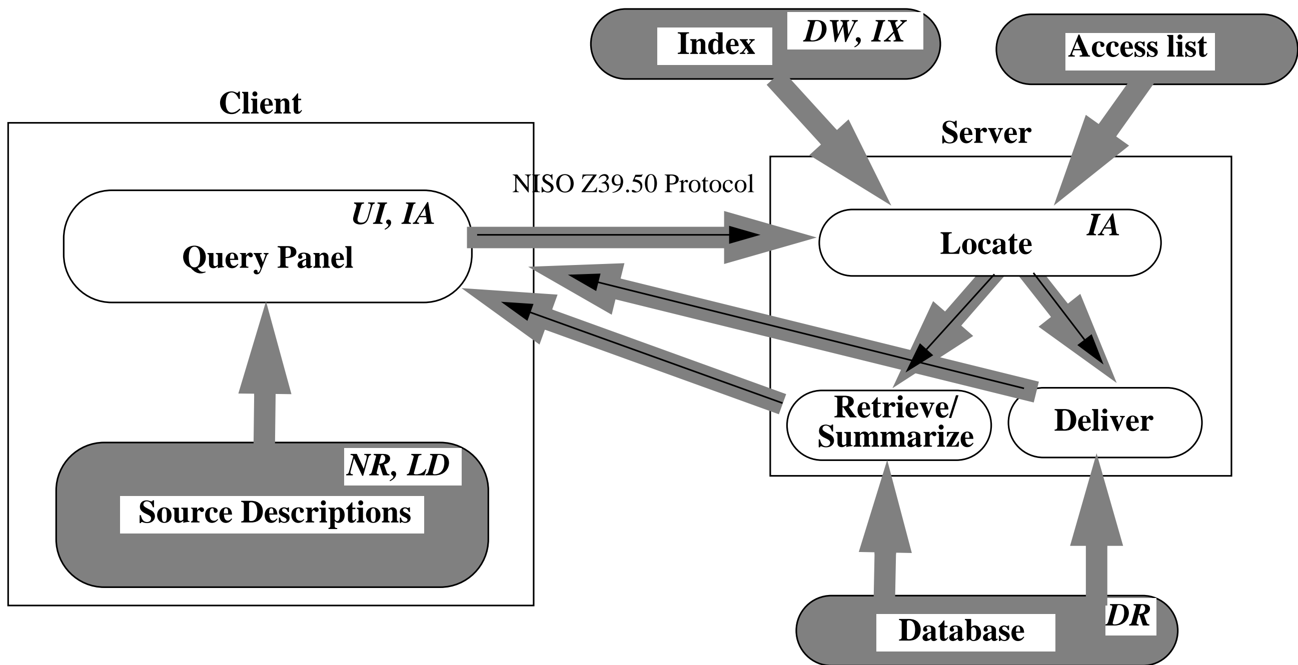

Figure 3: WAIS architecture for single server hosting a single database

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) that the user wishes to search. The client also provides logical database and name resolution functions through a collection of source descriptions providing relevant information about the available sources, such as physical location of the server hosting the database on the Internet, the name of the database on the server, and a short description of the contents of the database.

The server services the query request from the client. One client request is to search a database (or a collection of databases) that it hosts to find the best matches for a user query. Another client request is to retrieve a particular piece of data from the database. The database is a collection of documents of the same type. The document types range from textual to graphical to some combination of both. Each database has an index that contains an inverted file to link all words in the database to the documents that contain them. This index facilitates searching for keyword matches suggested by the user queries. Each database also has an access list that identifies those clients with permission to access the database. Once the server determines which documents are relevant for the search using the index and access list, it retrieves the documents from the database itself and ranks the documents according to some relevance criteria based on the user's query. The server then returns an ordered list of documents with header information to the client for

5. At the time of this writing, we are only clear about the architecture of WAIS for single server accesses. We are not sure how this picture changes to reflect searches that span several servers, though we know this is possible.

display to the user. The user can then either reiterate on the query to refine the output or request that a particular document be retrieved from the server.

The connection between client and server adheres to the NISO standard protocol Z39.50.

7.2.1 Add New Type of Data Repository

Referring to Table 1, we can see that this indirect task affects the data repository (DR) and the data wrapper (DW) functions. Examining Figure 2, we can see that the data-repository function is mapped onto the raw database, and the data-wrapper function is provided by the database index. We now consider how adding a new type of data repository affects those structures.

We mentioned earlier that WAIS provides several utilities for the information provider that are not depicted in Figure 2. Two of those utilities, an indexer and a parser, are relevant in this scenario. When an information provider wishes to publish a database, the information in that database must be parsed into documents containing header information and a collection of words that forms the document contents. Then an index is formed to provide quick access into the database. For existing WAIS data types, these two steps are easily performed by the two utilities. For new data types, the information provider is required to modify the parser to declare the form of the documents. The index utility need not be modified. The modification of the parser may or may not be trivial, depending on the form of the data and how easy it is for the provider to describe the raw data as a delimited list of documents, each of which is a collection of words that would be used for querying. Note that this scenario does not explicitly discuss the accessibility of this new data repository. In this case, to access the new repository, the client source description file would have to be modified to indicate the presence of the new source. WAIS scores a "+" for this scenario.

If we consider many instances of this scenario, each one requiring the introduction of a new type of data, we can see that WAIS is just as good as WWW. The number of data types supported by WAIS does not affect the task of introducing a new type, so WAIS scores a "+" for the augmented version of this scenario.

7.2.2 Add New information Access Engine

Referring to Table 1, we see that this indirect task affects the information-access (IA), user-interface (UI), and database-wrapper (DW) functions. WAIS supports three types of query mechanisms. One is a natural language interface that allows users to enter queries as arbitrary words that will be used to locate and rank documents in the database. Thesecond is the use of some boolean connectives to structure a query. The final query mechanism is called relevance feedback, in which the user indicates that the result from a previous query should be used in the next query to find documents most similar to the relevant document. Each of these query mechanisms are supported in the query panel and locate component. Introducing a new access engine would involve modifications to both of these components, assuming that the information in the index component (the inverted file list) is enough to perform the query. If that assumption is not valid for the new query mechanism, then a new index utility would have to be created. Given this relative inflexibility for defining new access engines, WAIS scores a "-" on this scenario. WAIS also scores a "+" on the augmented version of this scenario.

.

7.3 Prospero

Prospero [6] is a system that enables viewing files across the Internet according to a locally defined view. Its primary focus is on the *organization* of information rather than the searching or retrieving of information [7]. It is intended to be used in conjunction with other types of Internet systems such as Archie. In this context, its primary function is as a logical database component within a larger system. Our analysis, however, is of Prospero in isolation since it is presented in the literature as a stand-alone system. This discrepancy will reappear in our analysis.

The Prospero user establishes a local directory structure (organized in a fashion that is locally meaningful) that contains pointers to files spread out across the Internet. Associated with each local subdirectory is a filter that restricts and reorganizes the target of the associated link. The links and filters are executed at run-time so that the files that are accessed are current. It is possible for the target directory to itself have an associated filter so that the filters are composable. Prospero allows one user to organize a directory structure, for example, by papers then by topics then by language while another user organizes the same papers by language then by topic. Regardless of the organization, all of the papers are physically stored elsewhere with, potentially, a third organization.

Prospero is organized into clients and servers where the clients maintain the local views and the servers execute the links and the filters. The Prospero protocol contains items such as: create/delete link, list, and link forwarded. Prospero is designed so that existing applications (such as Unix commands) can run without modification and so that specialized Prospero utilities that are Prospero aware can also be written. Figure 4 shows a Prospero client and server.

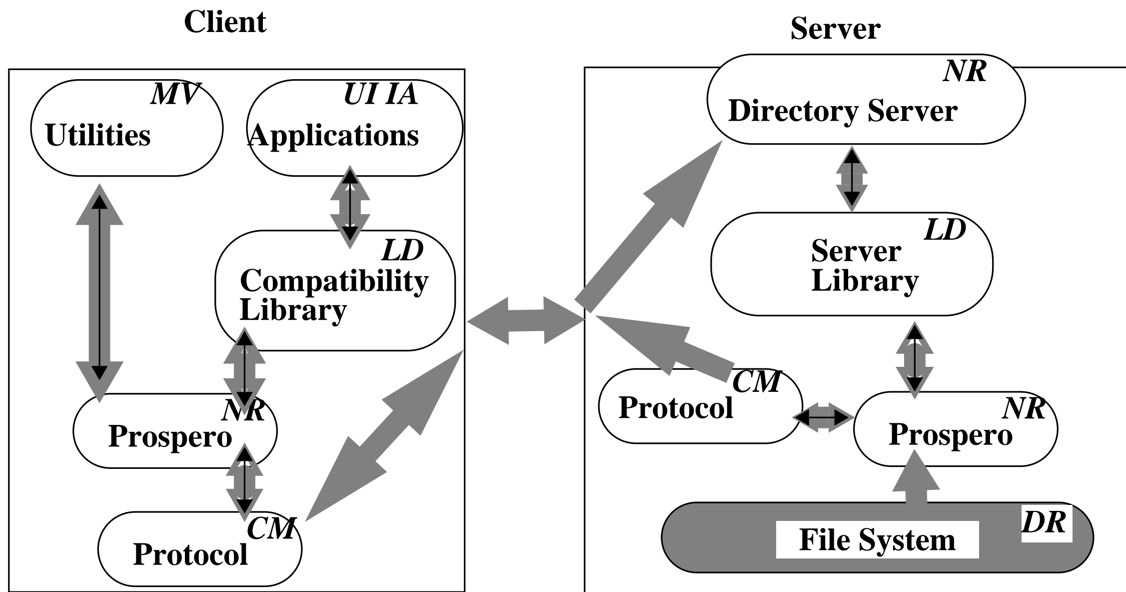

Figure 4: Prospero Architecture

7.3.1 Add New Type of Data Repository

Referring to Table 1, we can see that this indirect task affects the data repository (DR) and the data-wrapper (DW) functions. Examining Figure 4, we can see that the data repository function is mapped onto the file system, and the data wrapper does not exist. We need to consider how adding a new type of data repository affects those structures.

The new type of data repository would presumable exist within the native file system, so the DR is not affected by this change.

Because the data wrapper does not exist, one must be created to execute this scenario on Prospero. Presumably the data wrapper would convert the new type of data repository into the Prospero protocol (although this notion does not make a whole lot of sense). Since Prospero is intended to be used with other Internet systems, they would be the ones for which this evaluation is meaningful. We included Prospero to exemplify the types of situations where the system being evaluated does not support at all some of the task scenarios at all.

For this scenario Prospero gets a “-”.

7.3.2 Add New information Access Engine

Once again referring to Table 1, we see that this indirect task affects the information-access (IA), user-interface (UI), and database-wrapper (DW) functions. Again since Prospero does not have a database-wrapper function, it receives a - for this scenario.

8 Comparisons

The results of the analyses for the indirect scenarios are given in the following table:

Table 2: Comparison of Analyses

	adding new type of data repository	adding many new types of data repositories	adding new type of information access engine	adding many new types of information access engine
WWW	+	+	-	-
WAIS	+	+	-	-
Prospero	-	-	-	-

The interpretation of this table depends on the desires of the evaluator. Each scenario has to be weighted to reflect its importance in the particular evaluation being performed. In our case, since the indirect scenarios make no sense with respect to the user of Prospero, the result would be either that Prospero would be scored very low as a candidate or that these scenarios would have a very low weighting.

9 Conclusions and Further Work

This paper provides a method for evaluating a system at the architectural level for various non functional qualities. A requirement (in fact, a useful by-product) of the method is that the non functional qualities be manifested as task scenarios rather than in the abstract.

The method presented is a refinement of our previous analysis of user-interface construction systems [4] and is applied to a fundamentally different domain. As such, we feel that we have responded to the most fundamental criticism of our previous work—its domain specificity.

The method requires a great deal of work to set up the evaluation and very little to apply it. As such, it is suitable for inclusion in a handbook of standard architectures where the production of the handbook is a high-skill, labor-intensive exercise, but the application of the handbook is intended to be lower skill and not labor-intensive. This is true of most engineering handbooks.

There are still two criticisms that could be made of the current work:

1. The mapping of the functional partitioning onto candidate systems could be problematic. We consciously evaluated well-known IISs. We did this so that this work could be verified. These systems, however, did not all have well-defined, published architectures. In some cases, we generated the architectures presented from partial descriptions and from systems manuals. It is clearly possible that our lack of understanding of the systems being evaluated caused us to perform the mapping incorrectly. The scenario of usage that we envision for the method, however, would be to evaluate a collection of competing architectures for a particular problem where the evaluators would have far greater knowledge of the proposed architectures. In this case, the mapping from the functional partitioning to the candidate systems would not be problematic.
2. The functional partitioning is not repeatably generated from the particular scenarios chosen. Again, with the view of this method as being appropriate to a handbook developed by experts this criticism is not a concern. The functional partitioning would be developed by those people in the domain who had the most knowledge and were the most appropriate to perform this task. Thus, although the functional partitioning might not be repeatable, it would represent the best knowledge in the domain.

One of the shortcomings of the current work is that it focuses only on functional partitioning and the allocation of function to structure as an evaluation mechanism. There are clearly qualities and associated scenarios that depend heavily on the coordination model of the architecture, and these qualities cannot be analyzed by the current method. In addition, our notion of "connection" is still quite primitive, having only two types: data and control. A better understanding of our methodology, as it applies to coordination and connections, is a direction that we intend to pursue in the future.

10 References

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

[2] H. Frystyk, “WWW Library Internals—Overview”, URL=http://info.cern.ch/hypertext/WWW/Library/User/Guide/Overview.html.

[3] M. Gray, “Measuring the Size and Growth of the Web”, URL=http://web.mit.edu/afs/sipb/user/mkgray/ht/web_growth.html.

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

[5] R. McCool, “The Common Gateway Interface”, URL=http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

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

[7] B.C. Neuman, “Prospero: A Tool for Organizing Internet Resources”, Electronic Networking: Research, Applications and Policy, 2(1):30-37, Spring 1992.

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