Analysis and Evaluation of Service Oriented Architectures for Digital Libraries

Hussein Suleman

Department of Computer Science, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa
hussein@cs.uct.ac.za

Abstract. The Service Oriented Architecture (SOA) that underlies the Web Services paradigm of computing is widely regarded as the future of distributed computing. The applicability of such an architecture for digital library systems is still uncertain, as evidenced by the fact that virtually none of the large open source projects (e.g., Greenstone, EPrints, DSpace) have adopted it for internal component structuring. In contrast, the Open Archives Initiative (OAI) has received much support in the DL community for its Protocol for Metadata Harvesting, one that in principle falls within the scope of SOA. As a natural extension, the Open Digital Library project carried the principles of the OAI forward into a set of experimental derived and related protocols to create a testbed for component-based digital library experiments. This paper discusses a series of experiments with these components to confirm that SOA and a service-oriented component architecture is indeed applicable to building flexible, effective and efficient digital library systems, by evaluating issues of simplicity and understandability, reusability, extensibility and performance.

1 Introduction

Service-oriented computing is a relatively new paradigm of computing where tasks are subdivided and performed by independent and possibly remote components that interact using well-defined communications protocols [24]. In particular, the Service-Oriented Architecture (SOA) refers to a framework built around XML and XML messaging, with standards for how messages are encoded, how protocol syntax is specified and where instantiations of services are to be located. These are exemplified by the SOAP [7], WSDL [3] and UDDI [9] specifications respectively. It is often argued that SOA can be adopted by an organisation to increase reuse, modularity and extensibility of code, while promoting a greater level of interoperability among unrelated network entities.

From a somewhat different perspective, the Open Archives Initiative (OAI) attempted to address interoperability first and foremost, by designing a protocol for the efficient incremental transfer of metadata from one network entity to another. This Protocol for Metadata Harvesting (PMH) [11] has since been adopted by many large digital archives and has become the primary mechanism for digital library interoperability in 2004. The OAI-PMH is very closely related to SOA as it adopts a Web-distributed view of individual systems, where independent components - listed on the OAI website - interact through the medium of a well-specified request/response protocol and XML-encoded

messages. The OAI-PMH differs significantly from the SOA in that it is more concerned with a specific set of protocol and encoding semantics while SOA specifies only an underlying transport mechanism that could be applied to many different protocol suites. In this sense, a marriage of OAI-PMH and SOA is both possible and probable - initial experiments to verify the feasibility of this and expose possible areas of concern were carried out by Congia et al. [4].

In the interim, however, one of the reasons OAI-PMH has not as yet migrated to SOA is that the technology is not sufficiently well proven. In addition, OAI-PMH is aimed at interaction among entire systems, viewed as components of a super-system. SOA, however, is easily applied to a finer granularity, where reuse and modularity of components are crucial. To bridge this gap, and translate the core principles of OAI-PMH to fine-grained interaction among small components of a larger digital library, the Open Digital Library (ODL) project defined a suite of protocols based on the widely accepted principles of OAI-PMH, but aimed at the needs of digital library subsystem interaction [21][19]. There are some similar projects such as Dienst [10], which uses older technology, and OpenDLib [2], for which tools and reference implementations were not available for experimentation. These are discussed in detail in prior publications.

To maintain thematic consistency, the ODL protocols were designed in an object-oriented fashion, where each protocol built on a previous one, extending and overriding semantics as needed. A set of reference implementations of components were then created to provide the following services: **searching**; **browsing**; tracking of **new items**; **recommendation** by collaborative filtering; **annotation** of items as an independent service; numerical **ratings** for items in a collection; **merging** of sub-collections; and **peer review** workflow support.

A typical URL GET request to the search component, according to the ODL-Search protocol that is defined by ODL, is listed below:

```
http://www.someserver.org/cgi-bin/Search/instance1/search.pl?
  verb=ListRecords&metadataPrefix=oai_dc&
  set=odlsearch1/computer science/1/10
```

This request is for records 1-10 that match the query string “computer science”. The response is in a format very similar to that of the OAI-PMH, with records ranked according to probable relevance and one additional field to indicate the estimated total number of hits. This is the crux of the ODL-Search protocol, as implemented by the IRDB component. Other protocols and their associated component implementations use similar syntax and semantics.

The primary aim of the ODL project was to develop simple semantics for building experimental digital libraries - ODL was not meant for large scale production systems and there is no intention to standardise the protocols that were developed. Some users have noted that ODL protocols do not use SOAP/WSDL - this is largely because of their relationship to OAI-PMH and the fact that SOAP was not considered to be a W3C recommendation until mid-2003. However, the operation of ODL protocols is very much in keeping with the spirit of the SOA community and a move to SOAP/WSDL would involve only minor syntactic changes of negligible impact.

Ultimately, the purpose of the ODL framework was to serve as a testbed for experimental work. Many, if not all, of the experiments that were conducted have tested

features of the SOA model applied to digital libraries. The components were analysed and evaluated to determine how applicable a fine-grained component model is to the construction of digital libraries, and possibly expose problems and shortcomings to be addressed in future research. While the results of these experiments are generalisable to SOA, they are also indicators for the success of componentised Web-based systems in general, thereby blurring the already fuzzy line between Web-based information systems and digital libraries.

2 Experiments: Simplicity and Understandability

The first set of experiments aimed to determine if components with Web-based interfaces can be composed into complete systems with relative ease by non-specialists. Three different user communities were introduced to the underlying principles of OAI and the component architecture devised and were then led through the procedure of building a simple digital library system using the components.

2.1 OSS4LIB

The first group, at an ALA OSS4LIB workshop, provided anecdotal evidence that the component-connection approach to building DLs was feasible. The approximately 12 participants were largely technical staff associated with libraries and therefore had minimal experience with installation of software applications. They were carefully led through the process of configuring and installing multiple components and were pleasantly surprised at the ease with which custom digital libraries can be created from components. This led to a second, more controlled, experiment as detailed below.

2.2 Installation Test

The second group comprised 56 students studying digital libraries. The aim of this study was to gauge their level of understanding of OAI and ODL components and their ability to complete a component composition exercise satisfactorily. The objective was to install the following components and link them together to form a simple digital library, as illustrated also in Fig 1:

- XMLFile: a simple file-based OAI archive
- Harvester: an OAI/ODL harvester
- IRDB: an ODL search engine
- IRDB-ui: a simple user interface for IRDB

Students were given an hour-long introduction to OAI and ODL and then given detailed instructions to perform the component composition exercise. Upon completion, they were asked to fill out a questionnaire to evaluate the experience of building this system from components. Table 1 displays a summary of the responses to the core questions asked of users.

In addition to these questions, typical demographic information was collected to ascertain skills levels and exposure to the technical elements of the experiment. The

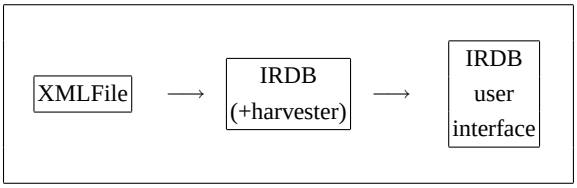

Fig. 1. Architecture of simple componentised digital library

Table 1. Summary of Responses to Component Composition Experiment

Question/Response	S. Agree	Agree	Neutral	Disagree	S. Disagree
Understand concepts of OAI-PMH.	9	38	9		
Understand concepts of ODL.	6	36	14		
Instructions were understandable.	35	20	1		
Installing components was simple.	36	18	2		
Configuration was simple.	33	21	2		
Connecting Harvester+XMLFile was simple.	28	25	3		
Connecting IRDB+XMLFile was simple.	26	25	5		
Understanding of OAI/ODL has improved.	13	25	12	6	
I will use ODL and OAI components.	12	33	10	1	

different backgrounds of participants evident from the responses to demographic questions makes it difficult to analyse these results without taking into account all of the interaction effects that result from past experience. Since OAI and ODL utilise various different Web technologies, it is non-trivial to enumerate all of the pre-requisites and determine their independent effects. It may be possible to construct an experimental model to minimise the interaction effects, but this will require finding unique participants, each with a very particular background and training. This may prove difficult because of the cutting-edge nature of Web-based technology. Taking these difficulties into account, any analysis of such an experiment cannot easily determine general trends.

Nine respondents who indicated that they did not know how Web Services worked answered affirmative when asked if they had done Web Services-related development. This may be because they interpreted the question as referring to Web-related services other than SOAP, WSDL, and UDDI, or because they had done development work without understanding the underlying standards and information model of Web Services. Either of these is consistent with the vague understanding many people have of Web Services.

Judging from the responses to the first two questions in Table 1, most participants appear to have grasped the basic concepts related to OAI and ODL. The fact that some participants were unsure indicates that an hour and 15 minutes may not be enough for a person building a digital library to learn enough about OAI and ODL. This raises the question of just how much training a person needs before being able to effectively use OAI and ODL technology (or any Web-service-related technology). Also, more of the participants were able to understand OAI rather than ODL; this is expected since ODL builds on OAI.

Most participants agreed (or strongly agreed) that the instructions were understandable. The instructions were very detailed so that even if participants did not understand one section of the exercise, they were still able to complete the rest of the steps.

Installation and configuration of individual components as well as interconnecting different components was deemed to be simple. As these are two basic concepts underlying ODL (that all services can be independent components and that systems are built by interconnecting service components), it supports the hypothesis of this experiment that ODL is simple to understand and adopt - which commutes to Web Services.

There was not much agreement about the ability of the exercise to improve the participants understanding of OAI and ODL. This can be attributed to the sheer volume of new concepts covered during the presentation and exercise. Given that approximately half of the participants had never created a CGI-based Web application before, the learning curve was quite steep. In practice, those who adopt OAI and ODL technology are usually digital library practitioners who already have experience with the construction of dynamic Web-based information systems.

In spite of all these factors, two-thirds of the participants indicated an interest in using similar components if they have a need for such services. Thus, even without a thorough understanding of the technology and other available options, the simple and reusable nature of the components seemed to appeal to participants.

Thirteen participants provided optional feedback in the survey, and these ranged from positive to somewhat skeptical. Eight of the reactions were positive, including the following comments:

- "I think the idea is very good and the approaches to build digital libraries is easy."
- "They provide a way to get up and running very quickly with a Web application."

Some participants were not sure about the workflow as indicated by the comment:

- "We have high level idea but detailed explanation will be great."

One comment in reference to the questions on simplicity of installation and configuration included:

- "I dont know; custom config might not be simple."

This summarises the notion that components should be simple enough to bootstrap a development process but still powerful enough to support a wide range of functionality. In particular, the above comment refers to the XMLFile component that is simple to install and use in its default configuration, but can be non-trivial to configure if the records are not already OAI-compatible. In such a situation, XSL transformations can be used to translate the records into acceptable formats. However, irrespective of the complexity of configuration for a particular instance, the OAI/ODL interface to such components always is the same.

Some questions raised during the lab sessions revealed very important issues that need to be addressed in future development of ODL or related Web-based standards:

- Confusion over baseURLs
  - Some participants were confused regarding which baseURL to use in which instance. Since all URLs were similar, it was not obvious – this ought not to happen in practice with any system built on OAI, ODL or Web Services technology.
  - Entering URLs by hand resulted in many typographical errors. Ideally, such links must be made using a high-level user interface that masks complex details like URLs from the developers.
  - The user interface was sometimes connected to the wrong component. While it is possible for a user interface to Identify the service component before using it, this will be inefficient. As an alternative, user interfaces can themselves be components, with associated sanity tests applied during configuration.
- Failures during harvesting
  - Harvesting will fail if the baseURL is incorrect, but there are no obvious graceful recovery techniques. The components used in the experiment assume a catastrophic error and stop harvesting from the questionable archive pending user intervention. Better algorithms can be devised to implement exponential back-off and/or to trigger notification of the appropriate systems administrator.

2.3 Comparison Test

Finally, a third experiment was conducted to contrast the component approach to system building with the traditional monolithic system approach. 28 students in digital libraries were asked to install a system similar to the one in the previous experiment as well as a version of the Greenstone [23] system, and compare and contrast them from the perspectives of ease of use and installation.

The responses highlighted both positive and negative aspects of both systems. The majority of respondents indicated that Greenstone was easier to install, being a single cohesive package. However, it was also agreed by almost all respondents that the component approach was more flexible and powerful, and therefore applicable to a larger set of problem domains than the monolithic equivalent. There was tension between the higher degree of architectural control possible with ODL and the increase in complexity it introduced for those not wanting such control. The service-oriented approach was also preferred for its scalability, genericity and support for standards, which was not as evident in the monolithic approach. A number of respondents were undecided as to an outright preference, given that each solution had its advantages and disadvantages - leading to the conclusion that an ideal solution would capitulate on the strengths of both approaches, somehow giving end users the advantages of component-based customisation and flexibility as well as the advantages of cohesion and simplicity inherent in the non-component approach.

3 Experiments: Reusability and Extensibility

To test for reusability and extensibility, the suite of components was made available to colleagues for integration into new and existing systems. A number of digital library systems have since made use of the components, either directly, or composed/aggregated into other components. The following are a discussion of how some projects have integrated service-oriented components and protocols into their architectures.

3.1 AmericanSouth.org

AmericanSouth.org [8] is a collaborative project led by Emory University to build a central portal for scholarly resources related to the history and culture of the American South. The project was initiated as a proof-of-concept test of the metadata harvesting methodology promoted by the OAI. Thus, in order to obtain data from remote data sources, the project relies mainly on the OAI-PMH.

The requirements for a central user portal include common services such as searching and browsing. AmericanSouth.org used ODL components to assist in building a prototype of such a system. The DBUnion, IRDB and DBBrowse components were used in addition to XMLFile and other custom-written OAI data provider interfaces. Many questions about protocol syntax and component logic were raised and answered during the prototyping phase, suggesting that more documentation is needed. Alternatively, pre-configured networks of components can be assembled to avoid configuration of individual components. Both of these approaches are being investigated in the DL-in-a-Box project [14].

The production system for AmericanSouth.org still uses multiple instantiations of XMLFile but the ODL components have been replaced with the ARC search engine [13] largely because of concerns over execution speed of the IRDB search engine component. This in itself indicates the ease with which service-oriented components can be replaced in a system whose requirements change over time.

3.2 CITIDEL

CITIDEL - the Computing and Information Technology Interactive Digital Education Library [6] - is the computing segment of NSF's NSDL - the National Science, Technology, Engineering and Mathematics Digital Library [12]. CITIDEL is building a user portal to provide access to computing-related resources garnered from various sources using metadata harvesting wherever possible. This user portal is intended to support typical resource discovery services, such as searching and category-based browsing, as well as tools specific to composing educational resources, such as lesson plan editors.

From the initial stages, CITIDEL was envisioned as a componentised system, with an architecture that evolves as the requirements are refined. The initial system was designed to include multiple sources of disparate metadata and multiple services that operate over this data, where each data source and service is independent.

CITIDEL uses components from various sources. In terms of ODL, this includes the IRDB and Thread components to implement simple searching and threaded annotations, respectively. The IRDB component was modified to make more efficient use of the underlying database, but the interface was unchanged.

3.3 BICTEL/e

The BICTEL/e project, led by the Universite Catholique de Louvain, is building a distributed digital library of dissertations and e-prints within the nine French-speaking universities in Belgium. The project adopted use of OAI and ODL components to support dissertations and e-prints collections at each university and at a central site, alongside some non-ODL components.

3.4 Sub-classing

Some component implementations were created by sub-classing existing components. All of the component modules were written in object-oriented Perl, which allows for single inheritance, so this was exploited when possible. Since the DBRate and DBReview components also store the original transaction records submitted to them, they were derived from the Box component. In each case, some of the methods were overridden to provide the necessary additional functionality.

3.5 Layering: VIDI

The VIDI project [22] developed a standard interface, as an extension of the OAI protocol, to connect visualisation systems to digital libraries. A prototype of the VIDI reference implementation links into the search engine of the ETD Union Catalog [20] to

obtain search results. The search engine used in the ETD Union Catalog understands the ODL-Search protocol. Thus, additional services are provided as a layer over an ODL component, without any reciprocal awareness necessary in the ODL system.

3.6 Layering: MAIDL

MAIDL, Mobile Agents In Digital Libraries [17], is a federated search system connecting together heterogeneous Web-accessible digital libraries. The project uses the "odlsearch1" syntax, as specified in the ODL-Search protocol, in order to submit queries to its search system. Further communication among the mobile agents and data providers transparently utilize the XOAI-PMH protocol [19].

4 Experiments: Performance

A number of performance tests were conducted to determine the effect of Web-based inter-component communication. The aim of these experiments was to demonstrate that the use of an SOA model would not have a significant adverse impact on systems in terms of performance. In addition, these experiments highlighted techniques that could be applied to ameliorate the effects that were noticed.

Measurements were taken for heavily loaded systems, systems that rely on multiple components to respond to requests (e.g., portals) as well as the contribution made to system latency by different layers in the architecture.

4.1 Application/Protocol Layering

The most critical of measurements looked at the effect of additional Web-application layering on the execution times of individual components of a larger system. The IRDB search engine component was used for this test because search operations take a non-trivial (and therefore measurable) amount of time and the pre-packaged component includes a direct interface to the search engine that allows bypassing of the ODL protocol layer.

For test data, a mirror of the ETD Union Archive was created and this then was harvested and indexed by an instance of the IRDB component. 7163 items were contained in this collection, each with metadata in the Dublin Core format.

The test was to execute a search for a given query. Three queries were used: "computer science testing", "machine learning", and "experiments". At most the first 1000 results were requested in each case. Each query was executed 100 times by a script to minimise the effect of the script on the overall performance. The first run of each experiment was discarded to minimise disk access penalties, and an average of the next 5 runs was taken in each case.

Six runs were made for each query:

1. Executing lynx to submit a ListIdentifiers query through the Web server interface.
2. Executing wget to submit a ListIdentifiers query through the Web server interface.
3. Using custom-written HTTP socket code to submit a ListIdentifiers query through the Web server interface.

4. Executing the search script directly from the command-line, thereby bypassing the Web server.
5. Executing testsearch.pl to bypass both the Web server and the ODL layer.
6. Using direct API calls to the IR engine, without spawning a copy of testsearch.pl in each iteration.

The time was measured as the "wall-clock time" reported by the bash utility program "time" from the time a run started to the time it ended. The script that ran the experiment controlled the number of iterations (100, in this case) and executed the appropriate code in each of the 6 cases above. In each case, the output was completely collected and then immediately discarded - thus, each iteration contributed the complete time between submitting a request and obtaining the last byte of the associated response, hereafter referred to as the execution time.

It was noticable from the measured times that execution time increases as more layers are introduced into the component. This increase is not always a large proportion of the total time, but the difference between Test-1 and Test-6 is significant. The time differences between pairs of consecutive tests is indicated in Table 2.

Table 2. Time differences between pairs of consecutive tests

Query	Test1-2	Test2-3	Test3-4	Test4-5	Test5-6
"computer science testing"	4.52	1.04	0.67	0.33	8.57
"machine learning"	3.72	0.63	0.58	0.22	10.62
"experiments"	4.26	0.94	0.35	0.66	8.53

Test-1, Test-2 and Test-3 illustrate the differences in times due to the use of different HTTP clients. In Test-1, the fully-featured text-mode Web browser lynx was used. In Test-2, wget was used instead, and the performance improved because wget is a smaller application that just downloads files. Test-3 avoided the overhead of spawning an external client application altogether by using custom-written network socket routines to connect to the server and retrieve responses to requests. The differences are only slight but there is a consistent decrease for all queries.

The difference between Test-3 and Test-4 is due to the effect of requests and responses passing through the HTTP client and the Web server. While no processes were spawned at the client side in Test-3, a process was still spawned by the Web server to handle each request at the back-end. This script was run directly in Test-4, so the difference in time is due solely to the request being routed through the Web server. This difference is small, so it suggests that the Web server does not itself contribute much to the total execution time.

The difference between Test-4 and Test-5 is due to the ODL-Search software layer that handles the marshalling and unmarshalling of CGI parameters and the generation of XML responses from the raw list of identifiers returned by the IR engine. This dif-

ference is also small, indicating that the additional work done by the ODL layer does not contribute much to the total time of execution.

The difference between Test-5 and Test-6 is due to the spawning of a new process each time the IRDB component is used. This difference is substantial and indicates that process startup is a major component of the total execution time.

In general, the execution times for the IRDB component (as representative of ODL components in general) were much higher than the execution times for direct API calls. However, this difference in execution time is due largely to the spawning of new processes for each request. The ODL layer and the Web server contribute only a small amount to the total increase in execution time.

4.2 Nested Requests and Persistence

In order to avoid duplication of metadata entries, some of the ODL components (such as IRDB) do not store redundant copies - instead, every time a record is needed, it is fetched from the source archive by means of further internally-generated requests to the Web-based ODL interface. This procedure is hereafter referred to as a nested request.

An initial experiment compared nested requests that invoked Web-based services 10 times for each query processed with requests that avoided the use of Web-based services in favour of internal APIs. This experiment did not make use of any performance optimisation technology. The results confirmed that the response time is roughly proportional to the number of nested requests and that process startup time is the most significant contributor to the delays.

To deal with the process startup time, the SpeedyCGI package [1] was installed and components were configured to use it instead of regular CGI. The effect of this change was then tested and compared against the case where no optimisations are used. SpeedyCGI is a tool that speeds up access to Perl scripts without modification of the script or the Web server. Instead of running Perl with each invokation of a script, the script is run by a relatively small SpeedCGI front-end program that connects to a memory-resident Perl back-end, creating the back-end process if necessary. Thus, the process startup time is determined by the execution speed of the front-end script rather than the Perl interpreter. The objectives of this study were to calculate the response times for single and nested requests, both with and without using SpeedyCGI, and to compare SpeedyCGI usage with the fastest approach thus far, that of directly utilising a programming API.

The IRDB search engine component was used for this test. The test was performed in the same experimental environment as for the Layering experiment. The test was to execute a search for a given query. Three queries were used: “computer science testing”, “machine learning” and “experiments”. The first run of each experiment was discarded and an average of the next 5 runs was taken in each case. For the first part of the experiment, all requests were submitted by executing wget, thus involving the Web server and the ODL interface in generation of the response. At most the first 10 results were requested in each case. Each query was executed 10 times by a script. Four runs were made for each query as follows:

1. By submitting ListIdentifiers (LI).
2. By submitting ListRecords (LR).

3. By submitting ListIdentifiers, where the components use SpeedyCGI (LIS).
4. By submitting ListRecords, where the components use SpeedyCGI (LRS).

Table 3 displays the comparisons from the first part of the experiment using SpeedyCGI and not, for both ListIdentifiers and ListRecords requests submitted to IRDB.

Table 3. Regular CGI vs. SpeedyCGI speed comparisons (seconds)

Query	LI	LR	LIS	LRS
“computer science testing”	1.67	16.50	0.44	2.22
“machine learning”	1.57	16.34	0.33	2.04
“experiments”	1.55	16.35	0.31	2.06

For the second part of the experiment, at most 1000 results were requested and each query was executed 100 times. The requests were submitted by executing wget, thus involving the Web server and the ODL interface in generation of the response. A single ListIdentifiers run was conducted for each query, and this was contrasted with the data obtained during the direct API measurements taken in the Layering experiment.

Table 4 displays the comparisons from the second part of the experiment using SpeedyCGI and comparing this to the previous measurements for the case with direct API use.

Table 4. SpeedyCGI vs. direct API speed comparisons

Query	SpeedyCGI	API
“computer science testing”	40.27	39.70
“machine learning”	16.61	15.81
“experiments”	32.39	32.78

Results from the first part of the experiment indicate that there is a significant improvement in execution speeds for both ListIdentifiers (single requests) and ListRecords (nested requests) when SpeedyCGI is used. This is largely due to the elimination of the need to spawn new processes to handle each request to the Web server. The second set of results indicate that there is very little difference in execution times between using direct API calls and using a fully layered IRDB component when SpeedyCGI is used.

This is a significant result for the applicability of SOA in situations where performance is an issue. Generalising, these experiments confirm that there is little cause for concern, performance-wise, if Web technology is chosen wisely – for example, using persistent Web applications such as servlets for Java applications or SpeedyCGI for Perl

applications. Other technologies exist for these and other languages and some of these have been evaluated in the ODL [18] and X-Switch [15] projects.

4.3 System Load

Under real-world conditions, response times can be drastically different as situations vary. The aim of this experiment was to assess the ability of a component to perform acceptably under high loads. To assess this, a server was artificially loaded and then the response times of typical requests were measured under different load conditions.

The Box component was used for this test because it has very little component logic and therefore provides lower bounds for execution speed that are indicative of the ODL component architecture and not the component logic. The component was installed in the same server environment used in the Layering experiment. A second identical machine was used to simulate client machines by running multiple processes, each of which submitted ListRecords requests to the server in a continuous loop. The server was primed with 100 dummy records for this purpose. The test was to submit GetRecord and PutRecord requests to the local server. The first run of each experiment was discarded and an average of the next 5 runs was taken in each case. The experiment was then repeated using SpeedyCGI for the Box component.

Table 5 lists the average execution times for GetRecord and PutRecord operations under load conditions generated by 5, 10 and 50 simultaneous processes.

Table 5. Average execution times under different load conditions

Operation	5 clients	10 clients	50 clients
PutRecord	1.04	2.39	9.31
GetRecord	1.39	2.06	11.27

Table 6 lists the average execution times for GetRecord and PutRecord operations under load conditions generated by 5, 10 and 50 simultaneous processes, when the Box component uses SpeedyCGI.

Table 6. Average execution times when using SpeedyCGI

Operation	5 clients	10 clients	50 clients
PutRecord	0.691	0.702	2.146
GetRecord	0.449	0.316	1.801

There is variability in execution time because of the non-deterministic nature of client-server synchronisation and process startup. It is apparent, however, that the time

taken to respond to a request increases as the load on the server increases. Using the persistent script mechanism of SpeedyCGI results in a reduction of the execution time as compared to the case without SpeedyCGI, but there is still an increase with increasing load, as expected.

In both cases, a high load on the server causes an increase in execution time for component interaction. The SpeedyCGI module, as representative of persistent script tools, helps to minimise this effect. Ultimately, however, ODL components are Web applications and the only way to get better performance for a heavily loaded Web server is to use more and/or faster servers. This suggests that a server farm or component farm based on cluster computing or grid technologies might be one possible solution for digital libraries that require a lot of processing power for computations in either the pre-processing (e.g., indexing), maintenance (e.g., reharvesting) or dissemination of data (e.g., online querying) phases.

4.4 User Interfaces

While inter-component communication speeds are important to system designers, it is the speed of the user interface that matters the most to users. To test this, requests were submitted to a mirror of CSTC (Computer Science Teaching Centre), based completely on ODL components, to determine its effectiveness. The objective was to submit requests to the CSTC interface, simulating typical user behavior, and then measure the response time.

The client machine was a 2Ghz Pentium 4 PC with 1GB of RAM running a pre-installed version of Red Hat Linux v7.3. The server used was a non-dedicated 600MHz Pentium 3 with 256MB RAM running Red Hat v6.2. The Web server was Apache v1.3.12 and data for all components was stored in a mySQL v3.23.39 database. All components used the SpeedyCGI tool to remain persistent in memory.

The first test involved simulating a browse operation. The second test involved simulating the viewing of metadata for a single resource. In both instances, the test was repeated 10 times per invocation of the test script. The test script was executed 5 times and the average of these was computed.

Table 7 displays the user interface execution times for the operations tested.

Table 7. Execution times for user interface actions

Action	t (Time taken for 10 requests)	t / 10
Browse first screen of items	9.28	0.93
Display metadata for first item	9.81	0.98

The browsing operation required 1 request that was submitted to the DBBrowse component, as well as 5 nested requests sent to the DBUnion component in order to fetch the metadata, resulting in a total of 6 requests. The display operation required

1 request for the metadata, 1 for the rating, 4 for the recommendation and 3 for the feedback mechanism - resulting in a total of 9 requests. The time taken is not simply proportional to these request counts because different components contribute varying amounts to the total execution time. However, as indicated in Table 7, the total response time in both instances is less than 1 second for data transfer.

In general, the time taken to generate user interface pages is reasonably small for the new CSTC system, running on a production server. In combination with a higher load (as tested in the previous experiment), it can be expected that the response time will increase further and this is additional motivation for a generic SOA-based component grid/cluster where services/components can be replicated as the needs of the system change over time.

5 Conclusions

The Service Oriented Architecture is still a fairly new concept in DL systems, with most systems supporting one or two external interfaces, for example OAI-PMH. This work has investigated the applicability of SOA as a fundamental architecture within the system, an analysis of which has demonstrated its feasibility according to multiple criteria, while exposing issues that need to be considered in future designs. In summary, this works provides evidence in support of the following assertions:

- From a programmer's perspective, SOA is simple to understand, adopt and use.
- Components in an SOA can easily be integrated into or built upon by external systems.
- Performance penalties from additional layering can be managed.
- Performance issues do not have to permeate the architectural model - these can be handled as external optimisations.
- There are no inherent architectural restrictions that prevent the modelling of specific digital library systems.

6 Future Work

The most important aspect highlighted by past experiments was the need for better and simpler management of components, so that the complexity of deconstructing a monolithic system into service-oriented components did not fundamentally increase the complexity of overall system management. To this end, the ongoing "Flexible Digital Libraries" project is investigating how external interfaces can be defined for remote management of components, thus enabling automatic aggregration and configuration of components by installation managers and real-time component management systems. The first of such systems to be built, BLOX [16], allows a user to build a system visually using instances of Web-accessible components residing on remote machines. Experiments conducted with BLOX by Eyambe [5] have demonstrated that users generally prefer high level programming of digital library systems by assembling building blocks visually as opposed to the traditional low-level programming API approach.

In addition, there is an ongoing project looking into how components designed with well-defined administrative interfaces can be packaged and redeployed, thus bridging the gap between components and monolithic systems from a system installation perspective. Ideally, a future digital library or online information system will be designed on a canvas and a package will be generated for distribution based on a concise specification of the system. Then distribution and installation will be as simple as possible for the ordinary user; but the administrator wishing to customise the system can still modify the specification of the system at a later date to upgrade or modify the components in use.

At the same time, some effort needs to go into how services are orchestrated and composed/aggregrated at a higher level to perform useful functions needed by users. The WS-Flow and WS-Choreography activities are useful starting points but more investigation is needed into their suitability as integration mechanisms between "front-end" and "back-end" systems.

User interface components are being developed in a related project, to address the common perception that information management components are usually only part of "back-end" systems. Initial experiments with the BLOX system have demonstrated that a user interface can be plugged into the system as easily as any other component.

Current directions for this research include extension of the core architecture to allow migration and replication of components to support "component farms" as a replacement for "server farms", adopting notions from the cluster and grid computing paradigms, where services are needs-based and location-independent. Thus, the component approach to building digital libraries will demonstrate scalability in addition to flexibility.

Eventually, it is hoped that SOA will form an integral part of an architecture for flexible online information management systems, with all the advantages of monolithic systems as well as component-based systems, and with the ability to meet the needs of both the resource-constrained trivially-small system and the scalability requirements of large-scale public knowledge bases - an architecture that can be easily and readily adopted by future generations of systems such as DSpace, EPrints and Greenstone.

References

1. SpeedyCGI, 2005. Website `http://daemoninc.com/speedycgi/`.
2. Donatella Castelli and Pasquale Pagano. OpenDLib: A Digital Library Service System. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 6th European Conference*, number 2458 in Lecture Notes in Computer Science, pages 292–308, Rome, Italy, September 2002.
3. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C, 2001. Available `http://www.w3.org/TR/wsdl`.
4. Sergio Congia, Michael Gaylord, Bhavik Merchant, and Hussein Suleman. Applying SOAP to OAI-PMH. In R. Heery and L. Lyon, editors, *Research and Advanced Technology for Digital Libraries, Proceedings of the 8th European Conference*, volume 3232 of *Lecture Notes in Computer Science*, pages 411–420, Bath, UK, September 2004.

5. Linda K. Eyambe and Hussein Suleman. A Digital Library Component Assembly Environment. In G. Marsden, P. Kotz, and A. Adesina-Ojo, editors, *Proceedings of SAICSIT 2004*, pages 15–22, Stellenbosch, South Africa, October 2004.
6. Edward A. Fox, Deborah Knox, Lillian Cassel, John A. N. Lee, Manuel Pérez-Qui nones, John Impagliazzo, and C. Lee Giles. CITIDEL: Computing and Information Technology Interactive Digital Educational Library, 2005. Website `http://www.citidel.org`.
7. M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. F. Nielson. SOAP Version 1.2 Part 1: Messaging Framework and Part 2: Adjuncts. Technical report, W3C, June 2003. Available `http://www.w3.org/TR/2003/REC-soap12-part1-2003-0624/` and `http://www.w3.org/TR/2003/REC-soap12-part2-2003-0624/`.
8. M. Halbert. AmericanSouth.org, 2005. Website `http://www.americansouth.org`.
9. Ariba Inc., IBM, and Microsoft. UDDI Technical White Paper. Technical report, September 2000. Available `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`.
10. C. Lagoze and J. R. Davis. Dienst - An Architecture for Distributed Document Libraries. *Communications of the ACM*, 38(4):47, 1995.
11. Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simeon Warner. The Open Archives Initiative Protocol for Metadata Harvesting Version 2.0. Technical report, June 2002. Available `http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm`.
12. Carl Lagoze, Walter Hoehn, David Millman, William Arms, Stoney Gan, Dianne Hillmann, Christopher Ingram, Dean Krafft, Richard Marisa, Jon Phipps, John Saylor, Carol Terrizzi, James Allan, Sergio Guzman-Lara, and Tom Kalt. Core Services in the Architecture of the National Science Digital Library (NSDL). In *Proceedings of Second ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 201–209, Portland, OR, USA, July 2002.
13. Xiaoming Liu, Kurt Maly, Mohammad Zubair, and Michael L. Nelson. Arc: an OAI service provider for cross-archive searching. In *Proceedings of First ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 65–66, Roanoke, VA, USA, June 2001.
14. Ming Luo. Digital Libraries in a Box, 2005. Website `http://dlbox.nudl.org`.
15. Andrew Maunder and Reinhardt van Rooyen. Universal Web Server: The X-Switch System. Technical Report CS04-20-00, Department of Computer Science, University of Cape Town, 2004. Available `http://pubs.cs.uct.ac.za/archive/00000157/`.
16. David Moore, Stephen Emslie, and Hussein Suleman. BLOX: Visual Digital Library Building. Technical Report CS03-20-00, Department of Computer Science, University of Cape Town, 2003. Available `http://pubs.cs.uct.ac.za/archive/00000075/`.
17. Nava Mu noz and Sandra Edith. Federación de Bibliotecas Digitales utilizando Agentes Móviles (Digital Libraries Federation using Mobile Agents). Master's thesis, Universidad de las Américas, Puebla, Mexico, 2002.
18. H. Suleman. *Open Digital Libraries*. PhD thesis, Virginia Tech, Blacksburg, VA, USA, December 2002. Available `http://scholar.lib.vt.edu/theses/available/etd-11222002-155624/`.
19. H. Suleman and E. A. Fox. Designing Protocols in Support of Digital Library Componentization. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 6th European Conference*, number 2458 in Lecture Notes in Computer Science, pages 568–582, Rome, Italy, September 2002.
20. H. Suleman and E. A. Fox. Towards Universal Accessibility of ETDs: Building the NDLTD Union Archive. In *Fifth International Symposium on Electronic Theses and Dissertations (ETD2002)*, Provo, Utah, USA, May 2002.

21. Hussein Suleman and Edward A. Fox. A Framework for Building Open Digital Libraries. *D-Lib Magazine*, 7(12), December 2001. Available `http://www.dlib.org/dlib/december01/suleman/12suleman.html`.
22. J. Wang. A Lightweight Protocol Between Visualization Tools and Digital Libraries. Master's thesis, Virginia Tech, Blacksburg, VA, USA, 2002.
23. I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge. Greenstone: A Comprehensive Open-Source Digital Library Software System. In *Proceedings of Fifth ACM Conference on Digital Libraries*, pages 113–121, San Antonio, Texas, USA, June 2000. ACM Press.
24. J. Yang. Web Service Componentization. *Communications of the ACM*, 46(10):35–40, October 2003.